Make it Smile :)
A Muscle-based Approach on Facial Animation

Emanuel DIMA[1], Corina DIMA[2], Dan CRISTEA[1,2]
[1] *"Al. I. Cuza" University of Iași, Faculty of Computer Science*[*]
[2] *Romanian Academy – the Iasi branch, Institute for Computer Science*
{dimae, gvrinceanu, dcristea}@info.uaic.ro

***Abstract*—Facial expressions are very important in human communications. Creating them programmatically, using a muscle-based system, can greatly reduce the amount of time needed to produce an animation. The main advantage of the muscle approach is that the only thing that has to be done before starting to animate a new character is to tailor the muscle system to fit the new facial features. We describe a mass-spring system used for the physical simulation of the structure and dynamics of the facial muscles and the skin.**

***Index Terms*—animation, biological system modeling, dynamics, muscles, simulation software**

I. INTRODUCTION

Creating realistic facial animations has always been a challenge in computer graphics. Traditionally, animating a human-like character is considered a difficult task, since it requires a lot of talent and modeling experience. We propose a muscle-based approach to the facial animation problem. This approach gives the animator the possibility to control the facial expressions of the character using the muscles defined in the system. The muscles are connected to each other, to the skin and to the skull. Their contraction induces the movement of the skin, thus creating facial expressions of the avatar.

The goal of the project is to create a three-dimensional model of a human face, able to convey emotions resembling in detail to human-like natural expressions to the users who interact with it. When created, an emotional language will permit the translation of the parameters of the system into generally known emotions.

The possible applications of such a system are countless: creating avatars for different interactive applications, that are able to read a text or carry a conversation; create 3D replicas of human beings and use them in teleconferencing (so only the audio signal and the captured parameters are sent through the network); create artificial tutors, that can teach lessons and show emotional feedback (nod their heads, gaze at the students, approve or disapprove their actions); medical applications on which to study the new look of expressions after facial or mandible surgeries and, of course, creating computer-generated characters that play a part in video games or movies.

This paper focuses on the mass-spring system that we used as a physical framework, and on the structure and dynamics of the muscles and the skin. It is organized as follows: the second section makes a tour among related work, section 3 describes the physical system underlying the muscles, sections 4 and 5 present the muscle and skin models, section 6 summarizes the main developments that we intend to add to the system in order to enhance its behavior, and the last section gives the conclusions.

II. RELATED WORK

There are different approaches to creating facial animations. They vary in the means they employ for obtaining the animation and in the final result. The explanation for such a wide variety of animation techniques is simple: creating state-of-the-art facial animation is difficult and expensive, both from an artistic and computational point of view. Very complex animations, which take into consideration accurate simulation of different anatomical parts of the human head usually involve huge computational load and could be cost prohibitive. Therefore, most systems are focused on a specific goal and they include into the model only the minimum resources that allow them to fulfill the goals, thus minimizing costs.

The most widely used method for creating facial animations is the morphing method. It implies having a set of facial expressions of the model and a set of feature points. The animation is obtained by interpolating the positions of these points between one facial expression and the other.

There is a standard for defining the facial expressions, called FACS (Facial Action Coding System). Paul Eckman developed it in 1978 [1]. FACS proposes a mapping between different movements and contractions of the facial muscles and the name of the resulting facial expression. The state of several muscles at a certain point in time is labeled as an *action unit* (AU). Each of these action units corresponds to a visible change in the facial expression of the model. The original system included 46 AU. An example is AU 26, *lid tightener*, created with the help of the *Orbicularis Oris* and *Pars Palebralis* muscles.

Nedel and Thalmann [2] proposed a simulation system where the forces inside a muscle are described using the concept of an action line. An action line connects the origin of the muscle to its insertion point. The action lines follow the external shape of the muscle. To add volume to the muscle, the action lines are subsequently connected horizontally, using equally spaced ellipses. The resulting lattice is introduced into a mass-spring system, and particles are considered at the intersections between the action lines and the horizontal ellipses. To preserve the volume of the muscle during the contraction some extra springs are added,

[*] The work on this paper was funded by the InterOb project, through contract 131-CEEX II03/02.10.2006

called angular springs. For each pair of original springs, x_0x_1 and x_1x_2, two extra springs are added: one that connects the ends of the original springs, x_0x_2, and one that connects the particle x_1 with the middle of the x_0x_2 spring. The interior of the muscle in not modeled.

Another muscular model was the one used by Kähler [3]. In his approach, a muscle is represented as a bundle of fibers, each with a piecewise linear polygon as a control structure. Geometric shapes are attached to the segments of this control polygon. The fibers can form sheet muscles by grouping, while the union of the geometric shapes attached to the segments represents the muscle surface. This model supports two types of contraction, a linear muscle contraction and a sphincter muscle contraction.

III. THE PHYSICAL SYSTEM

In order to simulate movement of the skin and the muscles in a physically correct manner, we need a system of objects that observe some clearly defined physical laws. Our choice was to implement a mass-spring system, that is, a system of mass **particles** connected by **springs** that observe Newton's laws of motion and Hooke's law of elasticity.

Each particle in the system has an identity, a position in a three-dimensional space (*x*, *y*, *z*), a mass (*m*) and a velocity (*v*). The fact that the particles have identity is not directly used, but it implies that two particles with the same position and mass are actually distinct. The mass of the particles in the system is a parameter; at the initialization of the system, these masses as well as their initial positions have to be specified. The particle masses are computed by dividing the volume of the muscle by the number of particles modeling it (this assumes that all muscles have an uniform density). The positions of the particles are modified by the system during the animation, according to the laws of motion that take into account the velocity.

The springs (considered ideal) are characterized by an elasticity constant (*k*), a default length (l_r) and an actual length (*l*). Each spring connects exactly two particles. The elasticity constant and the default length are parameters of the spring; the actual length is computed as the distance between the two particles. A spring acts upon its particles with an elastic force proportional to the elasticity constant and the difference between its default and its actual length.

Although the model allows for a particle to be free (not connected to any spring), in our model all particles are bound to one or more springs. Our model also allows two particles to be connected by more than one spring.

The mass-spring system has an initial default state, as defined by the current position of all the particles. The system advances from one state to another according to the mathematical expressions of the physical laws that it emulates. The new state is usually computed using numerical integration methods.

We decided to use the Verlet integrator [4], as it provides a good trade-off between speed and accuracy. It is a second order integrator, usually used in molecular dynamics simulations, faster than the Runge-Kutta method [5] and more accurate than the Euler integrator [5]. Moreover, it simplifies the process of introducing constraints in the movement of the particles.

According to the Verlet integrator, the mathematical formula for the approximation of motion is

$$\vec{x}(t+\Delta t) = 2\vec{x}(t) - \vec{x}(t-\Delta t) + \vec{a}(t)\Delta t^2 + O(\Delta t^4)$$

where *x* is the position of the particle, *t* is the current time, *Δt* the quanta of time used in the iteration, and *a* is the acceleration of the particle.

In our implementation, the advancement of the system from one state to another is called a *tick*. The actual *tick* is detailed in two steps: first, all the elastic forces, created by the springs, are computed. At the end of this step, for each particle, we know the net force of all the springs connected to that particle.

The second step consists of computing the actual movement of each particle, according to the previous equation. The acceleration is computed using Newton's second law, as the ratio between the net force that acts upon the particle and the mass of the particle.

The complexity of the algorithm we used is $O(n_s+n_p)$, where n_s is the number of springs and n_p is the number of particles, so it is very efficient. However, in practice we would like to have real time simulations for systems of tens of thousands of particles and tens of thousands of springs. In order to achieve real time performance we also implemented a multithreaded version of the simulation engine. Running the system on a machine with only two processing cores got inconclusive. We expect, however, that moving the model on a real parallel machine would bring a significant drop in computational time.

IV. MUSCLE STRUCTURE

We are keen to the idea that the realistic movement of a muscle cannot be achieved unless its inner structure is adequately simulated. We believe that a computer program will reveal the subtle shades of human expression only when the tissues of the human face will be modeled to the last cell. Till then there is still a very long way. Although our model hides the accurate details of organic tissue structure under the spring model, which has distilled only the movement aspects, in itself the model is able to accommodate such a level of detail that would be added at some time in the future.

Our muscle structure is based on two three-dimensional objects, created with a commercial modeling application[1]. The first one represents the surface of the muscle, in our case manually modeled based on anatomical references. There are models of human anatomy created automatically using CT scans[2], but these are not suitable for our engine due to differences in the way the anatomical parts are modeled. However, a future adaptation of one of these more exact representations is not excluded.

The second object is a three-dimensional poly-line that represents the contraction axis of the muscle, specified by a human modeler. This contraction axis is only an approximation, as we don't know precisely the actual orientation of the muscular fibers.

The muscle structure consists of a three-dimensional particle lattice (particles connected by springs) that fills the inside of the muscle, having a featured subset of springs that are oriented on the direction of the muscle contraction axis.

The generation process of this lattice starts from the poly-line. We consider several equally spaced reference points on it; the distance between these points is a user-defined parameter. Each such point is contained in a plane that is perpendicular to the contraction axis. This plane intersects the surface of the muscle, thus generating a planar polygon. The inside of this polygon represents a muscle cross section that is perpendicular to the contraction axis in the corresponding reference point.

We then consider the cross section with maximum area as the main cross section. This cross section is filled with a planar

[1] 3ds Max, http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=5659302

[2] Zygote project, http://www.3dscience.com/3D_Models/Human_Anatomy/index.php

lattice of mass particles and springs, actually a tessellation made of equilateral triangles with the area specified at the initialization. For each of the remaining cross sections, possibly having a different form than the main one, we build an analogous lattice through a bi-dimensional transformation, keeping constant the number of particles. Currently this transformation is based on radial similarity around the center of mass of the cross section: this means that the ratio between the distance from the center to a particle location and the distance from the center to the polygon outline is constant over all cross sections. Obviously, the lattices from the secondary cross sections are not regular anymore: the triangles become distorted. Yet, this approach has the advantage that the three-dimensional lattice is not twisted, but is only squeezed (Fig. 1).

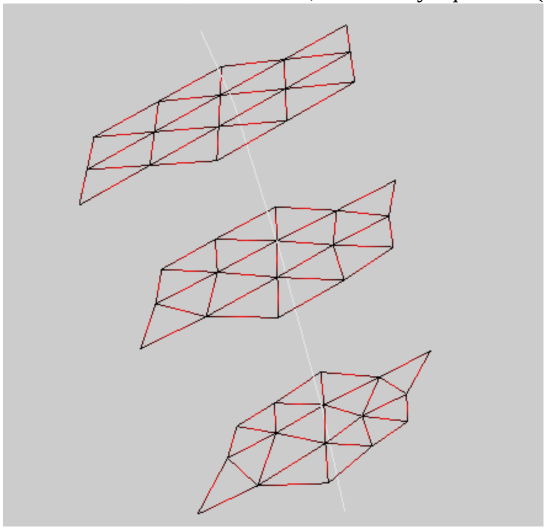

Figure 1. Detail of the structure of a muscle (three consecutive cross sections with the main one located on the top).

One can notice that the distance between the points on the contraction axis and the area of the equilaterals that tessellate the main section determines the density of the lattice. Although we tend to make these parameters ever smaller, in order to maximize the resemblance with a real muscle, the computational complexity stops us from modeling the muscle as a really dense grid.

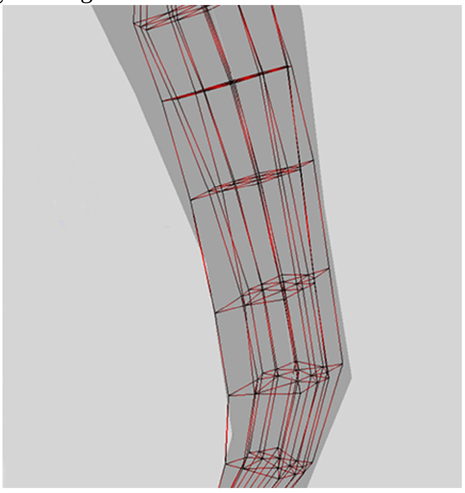

Figure 2. Detail of the structure of a muscle (cross section particles are connected by longitudinal springs).

The final step consists in connecting all the planar lattices through springs. As each cross section has the same number of particles, a longitudinal spring links two corresponding particles located on consecutive cross sections (Fig. 2). The contraction of a muscle is achieved by decreasing the default lengths of these longitudinal inter-section springs. These springs can be considered as the basic units of the muscle; they are the basic force generators.

Programmatically, the contraction of a muscle is specified by a value between 0 and 1, 0 being the state of complete relaxation (the initial state) and 1 the state of complete contraction, when the muscle tends to flatten completely (Fig. 3).

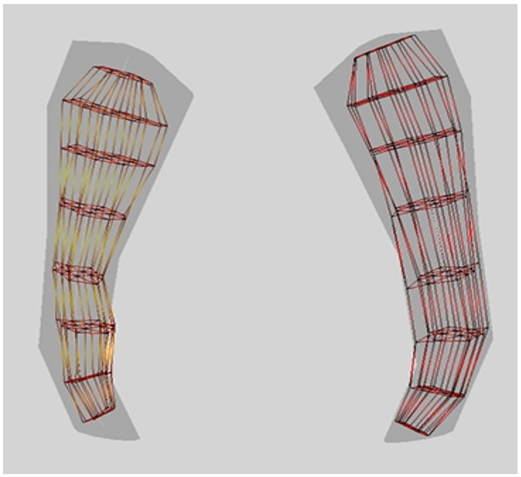

Figure 3. Muscle contraction (*Masseter Superficial*). The right muscle is completely relaxed; the left one is contracting. The muscle tension is reproduced by the lighter color

V. SKIN STRUCTURE

It is generally recognized that the skin is difficult to simulate and render realistically. Our implementation barely touches, for now, the challenges of skin simulation.

Our approach is to generate the skin automatically, based on the geometric information provided by the skull. The skin geometry is realized through a web of particles, in the same way as the muscles have been shaped. The particle system that simulates the physical movements of the skin is made of several layers, which simulate the three layers of the anatomical skin: *epidermis*, *dermis* and *hypodermis* (Fig. 4). Three skin particles are placed on a vector, normal at the skull, in a skull point. The outmost of them is characterized by an elevation, which gives the approximate thickness of the facial tissue (known as *tissue depth*). The values of the depth in different points have been computed by interpolating the values taken from a map of skin depths of landmark points on the skull [6].

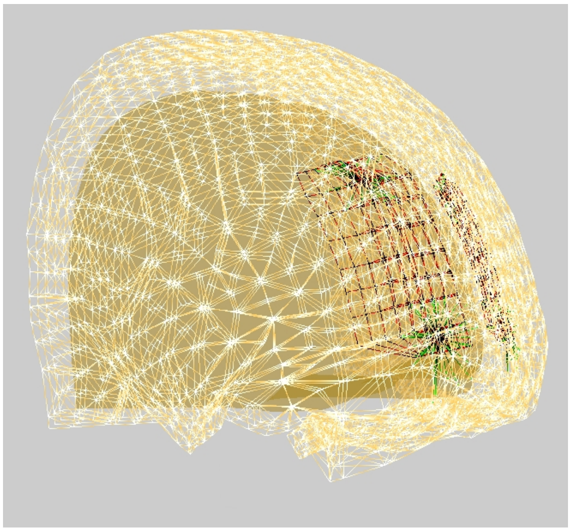

Figure 4. The three layers of the skin. Under the skin, in the right side, some muscles are visible.

Muscle insertions are only made in the innermost layer; a contraction of one muscle propagates to the surface through the intermediate layers, preventing unnatural deformations of a very small patch of skin where the muscle insertion is made.

VI. TOWARDS MORE REALISM

In this section we describe several improvements that can made to our system.

One way to improve the model is adding more (anatomically correct) details to it. The facial muscles have been made by hand using a modeling software package. In the present implementation, out of the over 50 muscles of a human head, only 15 are modeled. It is obvious that the less muscles are included in the simulation, the less accurate the expressivity of the avatar will be. The first improvement will be to fill in the model with the missing muscles.

To further improve the accuracy of the model, the muscles could be created from CT scans. This would ensure their anatomical correctness and the completeness of the muscular system. A different approach, similar to the ones employed in 3D facial reconstruction would be creating the muscles based on an existing skull and a set of anthropomorphic measurements.

Still, we should be prepared to recognize that adding so much extra detail would increase the computational needs of the system to the extend that the simulation will no more be supported by a consumer level computer, as it is now.

It is straightforward to customize the 3D model to fit a certain human face, once a skull and a muscular system are in place. This means that the hardest step towards creating and animating the avatar of a real person would be importing the specific facial features of that person into the model. This can be achieved either by 3D scans (easy to accomplish but expensive), or by using only photos (difficult but cheap).

In the present state the model lacks the ability to correctly simulate the volumetric deformation of the muscle. When we contract the muscle lengthwise, thus decreasing its length, the volume of the muscle actually shrinks, which is not realistic. In reality, the muscle’s width should increase to compensate (to a certain degree) its longitudinal deformation. We are currently working to include in the model this aspect of the muscle dynamics.

VII. CONCLUSION

We have presented a physical system used to create real time facial animations. The muscles and the skin are connected, so the user must only specify different contraction values for the muscles in order to create different facial expressions (Fig. 5)

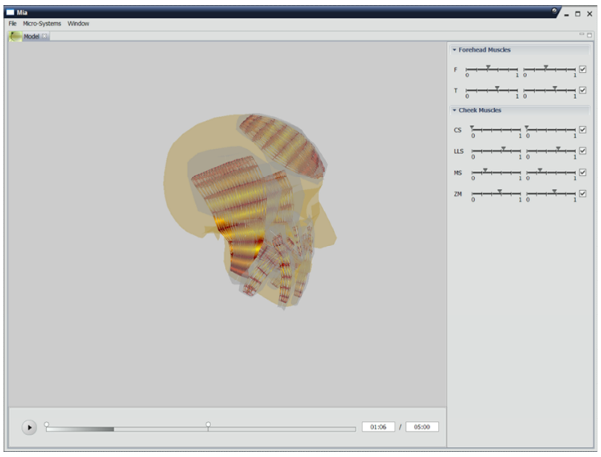

Figure 5. Main view of the system.

The internal structure of the skin and the muscles is modeled. This way, contracting a muscle that is connected to another one induces deformations in the second muscle, although its parameters were not explicitly altered. Moving from one animation frame to the other means computing all the interactions between the different components of the system. The animation is done in real time.

As a next phase on our research we intend to animate the model through commands expressed in an emotional language (similar to the mapping between facial expression and muscle contractions in the FACS system [1]). This means that muscles elongations should be put in correspondence with types of facial expressions. This would take the interaction with the avatar to a higher (more abstract) level.

ACKNOWLEDGMENTS

We would like to thank to our colleague Laurentiu Militeanu for providing the geometry for some of the muscles and the skull.

REFERENCES

[1] Ekman Paul, Friesen V.Wallace, Hager C. Joseph, Facial Action Coding System Investigator's Guide. A Human Face, 2002.

[2] Nedel Porcher Luciana, Thalmann Daniel, Real Time Muscle Deformations Using Mass-Spring Systems. Swiss Federal Institute of Technology. Proceedings of the Computer Graphics International, 1998.

[3] Kähler, K., "A Head Model with Anatomical Structure for Facial Modeling and Animation", Master's Thesis, Available: http://www.gamasutra.com/education/theses/20050526/Dissertation_Kaehler.pdf

[4] Ercolessi, Furio, “The Verlet Algorithm”, A molecular dynamics primer, Available: http://www.fisica.uniud.it/~ercolessi/md/md/node21.html

[5] Press, William H., Teukolsky, Saul A., Vetterling, William T., Flannery, Brian P., chap. Integration of Ordinary Differential Equations in Numerical Recipes in C. The Art of Scientific Computing, second edition, Cambridge University Press, 1992, pp. 710.