Universidade Técnica de Lisboa
Instituto Superior Técnico

Real-Time Facial Expression and Animation

Guilherme Coelho Barreira Raimundo
Licenciado em Engenharia Informática e de Computadores

Dissertação para a obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientadora: Doutora Ana Maria Severino Almeida e Paiva

Júri

Presidente: Doutora Ana Maria Severino Almeida e Paiva
Vogais: Doutora Catherine Pelachaud
Doutor João António Madeiras Pereira

Maio 2007

Resumo

Parke e Waters afirmam [22] que "O desenvolvimento de parameterizações de controlo e o desenvolvimento de sistemas de animação devem ser *dissociados*.". Esta abordagem faz sentido uma vez que cada um destes processos tem requisitos diferentes, razão pela qual devem ser tratados separadamente. No entanto, os sistemas parameterizados de animação facial tendem a ter estes processos interligados até um certo grau.

Esta tese propõe uma solução de forma a que esta separação seja explícita. Para tal, o sistema de animação facial desenvolvido abstrai a noção de parâmetros como valores unidimensionais com semântica associada. Estes podem tomar forma concreta através de instanciações de um conjunto de técnicas de deformação. Este conjunto é criado de forma a garantir um extenso número de representações, não comprometendo propriedades como o desempenho em tempo-real.

Relativamente ao movimento duas propostas são apresentadas: animação por *key-frames* e uma abordagem baseada em fases. Estas usam interpolação através de uma spline cardinal para calcular *frames* intermédias.

É também considerado comportamento facial de mais alto nível. Para criar movimentos *idle* utilizou-se Perlin Noise. Um modelo simples de coarticulação também foi implementado de forma a permitir animação sincronizada com a fala.

Palavras-Chave: Expressão Facial, Parameterização, Animação, Tempo-Real, Deformação, Personagens Sintéticos

Abstract

Parke and Waters state that [22], "The development of control parameterizations and the development of animation systems implementations should be *decoupled*.". This approach is sound since each of these processes has different requirements and therefore should be dealt separately. Nonetheless, parameterized facial animations systems still tend to have this two processes intertwined to some degree.

This thesis proposes a solution to make this separation explicit. To achieve this, the designed facial animation system abstracts the notion of parameters as simple unidimensional values that have associated semantic. These can in turn take concrete form through instantiations from a set of deformation techniques. This set is created in a manner that guarantees an extensive number of possible representations, while not compromising other properties like real-time performance.

Regarding motion, two proposals are provided: the classical key-frame animation and a phase based approach. These make use of a tailored cardinal spline interpolation method for the determination of inbetween frames.

High-level behavior in facial animation is also considered. Perlin noise is contemplated for the creation of idle motions and a simple coarticulation model is used for speech synchronized animation.

Keywords: Facial Expression, Parameterization, Animation, Real-Time, Deformation, Synthetic Characters

Acknowledgements

I would like to thank my parents for all their support throughout my still evolving academic path. I am ever grateful for their counseling that has kept me in the right direction even when things seemed bleak.

I would like to thank all members of GAIPS. I must say that being part of such a friendly, cohesive and willing to help team makes me very happy. Without forgetting anyone i would like to particularly thank:

Carlos Martinho, for all the great mind challenging discussions we have and for his to the point analysis of problems that often makes me see things from a different perspective.

João Dias and Rui Figueiredo, my "office mates", for all the incentive and companionship they have given me during this work.

Marco Vala, for his neverending patience in listening to my sometimes far fetched ideas and for his guidance when the path to take was uncertain.

Rui Prada, whose advices proved useful throughout the writing of this dissertation.

To all the people that helped me revise this dissertation, thank you.

Finally, I would like to thank my supervisor Ana Paiva for the challenge given to me with this project and for all the guidelines that kept me on track.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"The face is the mirror of the mind, and eyes without speaking confess the secrets of the heart."

Saint Jerome, Letter
(374 AD - 419 AD)

1.1 Motivation

The face plays a central role in our social lives by providing an efficient, and frequently honest, way of communication [12]. Among other things, it serves as a conversational moderator. Our facial expressions give out cues that let the other interlocutor know that we are waiting for a reply or that we did not understand what he said. With our face we show to the world what we are feeling and it often betrays us when we try to lie. From a facial expression we can infer motivations and what action someone is about to carry out.

The range of information we convey with our face is so vast that the brain developed dedicated mechanisms for its recognition during mankind's evolution. Since an early age one can recognize familiar faces and during our life time we tend to see faces everywhere. The human brain is so well trained in the task of facial expression recognition that we are able to tell even the subtlest of changes in a face.

Given the importance of facial expression, it is no surprise that animators devote so much attention to it when creating synthetic characters. However, due to its complexity and the brain's natural ability to detect false expressions, creating convincing artificial expressions is not an easy task. Nonetheless, over the years many specialized techniques and systems have been created to not only enhance the final result but also facilitate the process of achieving it.

An excellent example of how far facial animation techniques have come is the "monologue" of Gollum/Smeagol in the movie "The Lord Of The Rings: The Two Towers". In this conversation, facial expression plays a fundamental role in portraying the character(s). Although there is only one face, one can tell that Smeagol and his alter-ego Gollum are clearly different personae. Through their facial expression one can see that their motivations, intentions and emotions are very far apart.

Given that in movies the end result is not subject to real-time constraints, 3D animators can make use of complex graphical techniques that yield a high-quality outcome. In fact, animators will make use of all available instruments to achieve the intended effect. However, this focus on the outcome usually leads to an ad hoc solution for the particular facial animation of a given character. This one time solution means that none or little work from that developed solution can be reused in the future.

Pre-rendered facial animation systems are not alone in their evolution. It is now possible to consider the use of the necessary nuances of facial expression in complex real-time virtual environments populated by many synthetic characters. This is accomplishable due to the powerful and increasingly cheapper computational power available today. The gaming industry is perhaps where this can be more easily asserted. While not many years ago, game character's faces were nothing more than a lump with a texture, today they are rich in detail.

Valve's Half Life 2 game [33] provides an illustration of this evolution. In this sequel of the original Half life, facial expression plays a key role in the creation of empathy between the user and game characters. Figure 1.1 shows the evolution in facial expression from the original Half Life game to its sequel.

Regarding the facial animation system itself, the gaming industry tends to follow a similar approach to the movie industry except that it is bound to real-time restrictions. Each character usually has a library of fixed animations which can be played efficiently within the game. Therefore, character-user interactions, through animation, tend to be limited to a combination of previously determined behaviors.

As we depart from scripted environments, into more emergent scenarios where embodied intelligent agents take a preponderant role, higher demands are set upon facial animation systems. The increased autonomy in agents is no longer satisfied by predetermined and fixed animations. It yearns for a deeper and finer control of the face in order to adequately express the inner complexities of the agent. Here, the agent becomes the puppeteer controlling each

Figure 1.1: Evolution of the G-man character from Half life 1 released in 1998 (on the left) to Half Life 2 released in 2004.

string to achieve the final performance which is facial expression.

This latter example is apart from the others in the sense that the structure of the animation controls is as important as, or in some cases more important than, the final graphic result. Embodied Conversational Agents (ECAs) provide a good case of such non-scripted interactions. ECAs aim to mimic all expression channels, being one facial expression, that are used by humans when engaged in a conversation. Due to the rich and complex nature of such task it is required that the ECA possess a structured control over its face. This must not only provide precise and accurate control of the face but also yield a meaningful interface for the processes that manipulate it.

Even applications where structured manipulation is not mandatory can gain from using such an organized approach. For example, if all synthetic characters in a movie use the same set of controls and if all animations are defined through how these vary over time, then animations may be interchanged freely between characters.

1.2 Objectives

In the previous section, a few examples of applications of facial animation systems were depicted. It was shown that some of these facial manipulations were carefully structured while others were not. Nonetheless, it can be argued that even the latter have an implicit structure. This structure is constituted by the set of applied techniques that the animator used to achieve the animations. From this reasoning we can say that ultimately, all facial animation can be

seen as the manipulation of individual control parameters over time. Therefore, creating a facial animation system for a given purpose consists roughly of 1) deciding which parameters to use and 2) creating a way for these to be employed on a synthetic character's face.

In spite of the fact that in many developed solutions these two processes are intertwined, Parke and Waters [22] argue that they should be treated separately: "The development of control parameterizations and the development of animation systems implementations should be *decoupled*.".

This clear separation of the two processes is legitimate since each process has its own independent requirements to fulfill. The parameter set is important for the entity that manipulates it. This entity is interested in knowing what manipulations can be done and not how these are done. So, creating control parameterizations needs to take in account requisites such as easiness of use, range of possible expressions and predictability. On the other hand, the animation system implementation needs to supply methods that allow for the execution of the parameters while complying with visual quality and render efficiency factors. While essentially separate, these two processes interact in the sense that the animation implementation is a mapping of parameters.

Given the above, the challenge that this thesis proposes to solve is ***how to create a facial animation system where the parameterization/implementation detachment is adopted***.

1.3 Outline

This document is divided into six different chapters and one appendix.

Chapter 2 (Related Work) presents a taxonomy for facial animation systems. Here, several deformation techniques and parameterizations that were developed over the years for facial expression synthesis are described. Finally, a comparative analysis of the exhibited systems is brought forward.

In **Chapter 3** (Conceptual Model) the conceptual model for the proposed system is discussed. Parameterization, Parameter and Parameter instantiations concepts are introduced and their relations explained. In addition, the available high level control mechanisms are described.

Chapter 4 (Implementation) shows how the system implementation is structured. It is in

this chapter that the algorithmic details of the system are displayed.

Chapter 5 (Application) offers a more technologically detailed description of the system describing the developed application.

Chapter 6 (Case Study) provides an illustrative case study of how a parameterization can be used in the system. Here, concrete examples of system addressed problems are looked at.

Chapter 7 (Conclusion) offers a roundup of the developed work. Here, the encountered problems are reiterated and the proposed solutions summarized. In the end, a look into the possible extensions of the system is given.

Appendix A (Random Generated Facial Expressions) displays a series of images that were obtained by using the developed facial animation system.

Chapter 2

Related Work

In this Chapter we portray the main approaches to facial animation in computer graphics. As mentioned, we aim at creating a facial animation system where the *decoupling* of parameterization and implementation is followed. For this reason, we divide our analysis of facial animation systems in these two dimensions. First (Section 2.1), we look at what computer graphics techniques are used to create the actual facial deformations. A taxonomy for these techniques is presented and for each category some examples are given. Secondly (Section 2.2), we present the diverse parameterizations used by facial animation systems over the years. Finally, a comparative analysis of the several animation techniques and parameterizations is shown.

2.1 Deformation

As Figure 2.1 depicts, computer generated facial expressions can be divided into two major approaches: Geometry and Image manipulation. This division essentially separates vector graphics operations, that are usually made in three dimensions, from raster graphics ones.

Geometric manipulations can be further divided into Physics based models and Pure geometric algorithms. Both these categories eventually sum up to a set of geometric equations. However, there is something that differences them apart. Physics based approaches try to simulate the physical properties of the face such as muscle positions, skin elasticity and muscle contraction force propagation. On the other hand, pure geometric techniques are only concerned in obtaining the intended facial deformation in a visually pleasing manner.

Probably the most known and vastly used technique in computer animation is key frame

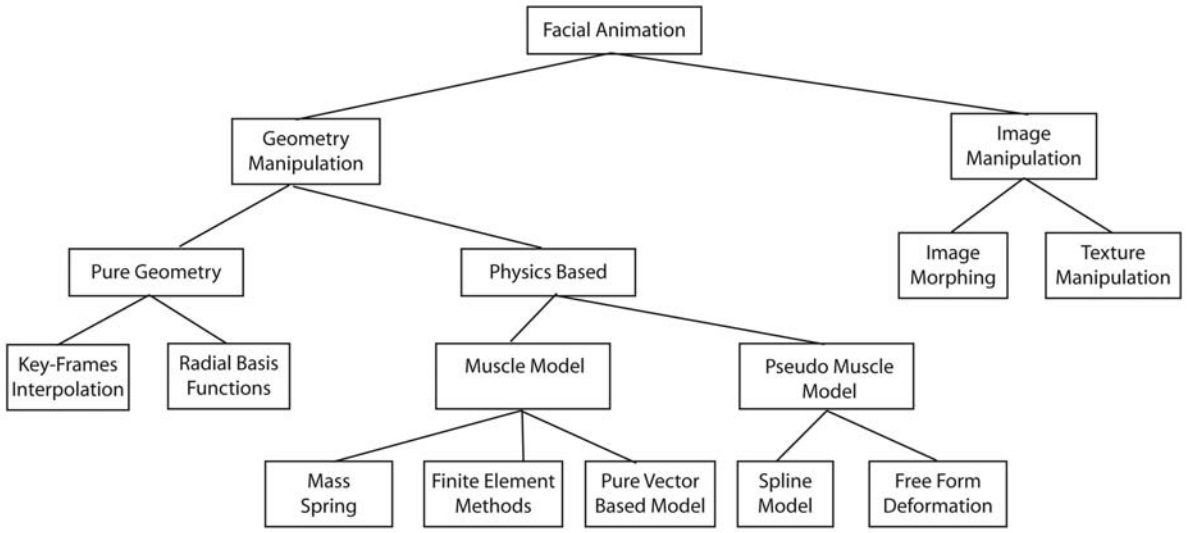

Figure 2.1: Facial Animation Taxonomy

interpolation. This, along with Radial Basis Functions, are examples of pure geometry techniques used in the creation of facial animation.

One can model the physical properties of the face in several degrees of detail. Techniques that try to more closely simulate reality, such as Mass-Spring systems, Finite Element methods and vector models are classified as muscle based models. Some techniques however, although modeling physical properties are not as deeply concerned with physical accuracy and can be said to engage an approach between a muscle and a geometry based model. Free From Deformations and Spline models are cases of this class that is denominated by Pseudo Muscle Models.

Some systems can not be fitted into a particular class depicted in Figure 2.1. Systems such as [27] , and the one presented in this thesis, combine different techniques to enhance the quality of the generated expressions.

2.1.1 Image Manipulation

Image Morphing

Image morphing is the process of changing from one image to another through a smooth seamless transition. To achieve such transition, the images must be warped while a cross-fade is performed between them. The warp function is used to make the mapping between pixels in the two images. This will assure that key features from one image will be warped to the

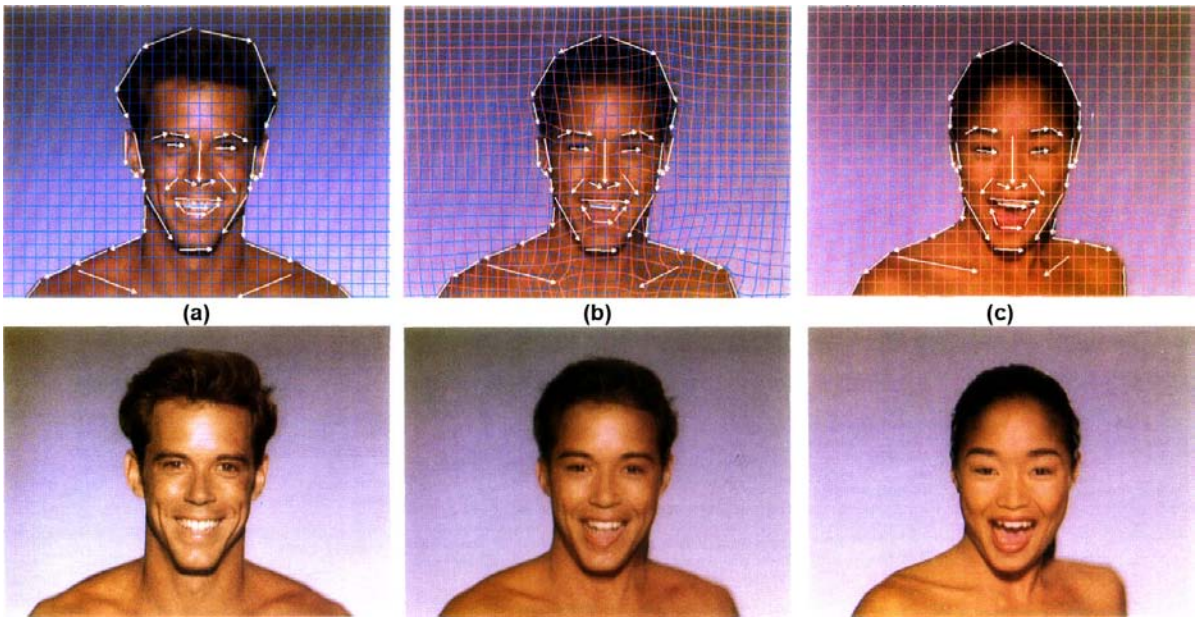

Figure 2.2: Top images with feature line segments: (a) Source image (b) Intermediate Morph (c) Target image

corresponding location of the target image. For example, the left eyebrow of the source image is moved to the location of left eyebrow of the target image. The cross-fade is basically a transition of the pixels RGB values from the source to the target image.

Beier and Neely [7] suggest an image morphing method that makes use of line segments as key features. The warp function then uses influence fields around the line segments to create the morph between images.

This approach allows for realistic animation but requires extensive manual interaction for feature correspondence, color balancing and tuning of warp parameters. Since the method is based only on 2D information, head rotations are difficult to represent due to the occlusion of features. Figure 2.2 depicts images from Michael Jackson video clip "Black or White" where the technique was applied.

Careful creation of key feature points allied with good warp functions may create quality facial animations. However, this process is intensive, dependent of viewpoint and is not generalizable to different faces.

Texture Manipulation

The human skin is rich in detail being composed of numerous elements. Ridges, furrows,

hair follicle openings, sweat gland openings and blood flow in blood vessels determine how light is deflected from skin surface. Although theoretically it is conceivable that the face geometry could model all these properties if it was detailed enough, pragmatically this is not feasible. To enable the representation of the subtle variations of skin at per pixel level, textures are typically used.

Pighin et al. [28] developed a photo realistic 3D model that uses blending of textures. A generic face mesh is fitted to a particular face using an interpolation technique on collected scattered data. Geometry fitting is performed and textures maps are extracted for each of the desired facial expressions. To create facial animations, 3D shape morphing between captured expressions is used at the same time as texture blending. View-dependent and view-independent texture maps are used along with weight-maps to blend between the multiple textures. The weight maps are created to consider self-occlusion, smoothness, positional certainty and view similarity. The use of view-dependent texture blending allows for the adjustment of the blending weights for the current view. This adjustment minimizes the blurring that occurs by the use of view-independent texture blending. However, the use of view-dependent textures is more memory demanding and more sensitive to light variations. Figure 2.3 shows an example of this work.

Figure 2.3: A global blend between (a) “surprised” and (b) “sad” produces image (c)

Besides face deformation, skin color changes are also part of facial expression. In [14] Kalra proposes a facial model that contemplates vascular expressions. In this work, skin color changes are related to an emotional state. Here emotion is defined as a function of two parameters. One is associated with the geometrical deformation while the other handles color variations due to vascular activity. Texture mapping and pixel manipulation is then used to approximate vascular effects for given face regions.

2.1.2 Geometry Manipulation

2.1.2.1 Physics Based Model

Vector Muscles

Vector muscles were introduced by Waters [35, 23]. This method attempts to to mimic the primary biomechanical characteristics of facial tissue displacement using a geometric distortion function. By focusing on the most pertinent visual effects of muscle contraction, this approach is able to use a simple model that still shows quality results.

A real muscle is composed of numerous fibers that run along the muscle area. In vector muscles it is assumed that all these fibers run in a given direction toward a given frontier that represents the muscle attachment in the bone. Waters modeled three types of muscles: linear, sheet and sphincter. All of these are defined by a vector whose ends define the muscle skin and bone insertion points. When the muscle is contracted by a given force the skin insertion point is the one that suffers the most displacement. In the remaining influence region of the muscle this force is dissipated according to geometrical factors. It is in the shape of the influence region and force dissipation factors that the several types of muscles differ amongst themselves.

The influence region of linear muscles is defined by a spherical cone in the three dimensional space (Figure 2.4(a)). When the muscle contracts all points in this region are displaced toward the cone vertex that is also the bone insertion point of the muscle. The displacement is higher for the closest points of the skin insertion point. The force, and hence the displacement, is then dissipated with the distance to the bone insertion point and angle formed with the muscle vector. Figure 2.4(b) depicts the contraction effect of a linear muscle in a plane mesh.

Sheet muscles have a circular cylinder influence region (Figure 2.4(c)). The bone insertion point is placed in the center of one of the tops. When muscle contraction occurs, the points inside the influence region move toward the cylinder top that contains the bone insertion point. Contraction force is dissipated with the distance to this cylinder top and to the cylinder axis. Figure 2.4(d) shows an example of a contracted sheet muscle in a plane mesh.

Sphincter muscles influence region has an ellipsoid shape (Figure 2.4(e)). When a sphincter muscle contracts the points inside its influence region move toward the center of the ellipsoid. The contraction force is dissipated with the distance to the center of the region. Figure 2.4(f) portrays a sphincter muscle action in a plane mesh.

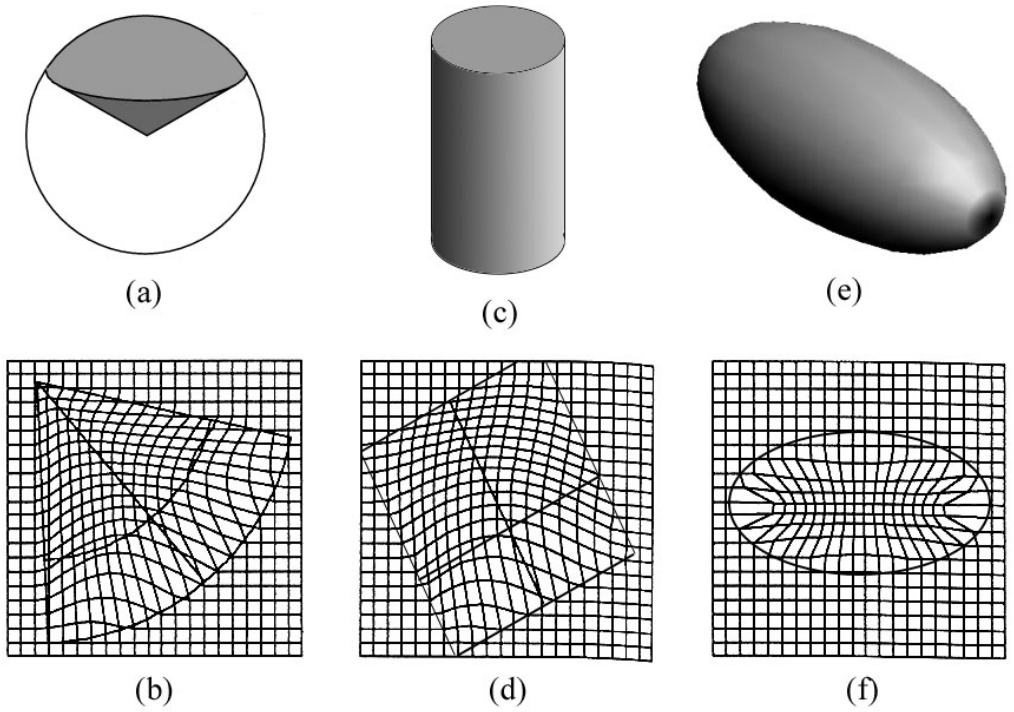

Figure 2.4: Water's muscle model influence regions and contraction examples. (a) Linear muscle influence region (b) Linear Muscle example (c) Sheet muscle influence region (d) Sheet muscle sample (e) Sphyncter muscle influence region (f) Sphyncter muscle example

The muscle vectors contraction behavior can be modeled by the following generic function:

$$P' = P + f \times K \times d \times \vec{v}$$

where P' is the new position for a given point in the muscle influence region; P is the point location when no force is applied in the muscle; f is the value of the force; K is a fixed constant that represents the elasticity of the skin; d is a function that determines the force dissipation factor; and $\vec{v}$ is the normalized vector that represents the contraction direction for the point.

This method simulates facial tissue primary dynamics using a computationally inexpensive method that yields good results. Nonetheless, creating each muscle and fine tuning its characteristics to render realistic results can be time consuming.

Mass-Spring

In mass-spring systems deformable objects are represented by a lattice of point masses that are connected by springs. Figure 2.5 shows an example of such a structure. When an external

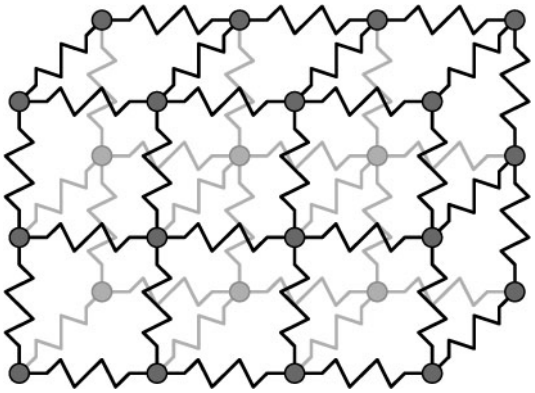

Figure 2.5: Mass-spring lattice: circles represent point masses and lines between them correspond to springs.

force is applied to a given point it causes a displacement in its position. This displacement will create a force in the connected strings that will then be propagated to neighboring points.

In this type of system the movement of a given mass point in the lattice is given by Newton's Second Law :

$$m_i \frac{d^2 x_i}{(dt)^2} = -\gamma_i \frac{dx_i}{dt} + \sum_j g_{ij} + f_i$$

For a given point i, m_i is its mass and $x_i \in \Re^3$ is its position. In the right-hand of the formula we have γi that is a velocity dependent damping force coefficient, g_{ij} is the force exerted on i by its connecting springs to points j. Finally, f_i is the resulting force of all other exterior forces applied to mass point i. If we concatenate the position vectors of a system with N points into a single 3N-dimentional vector one could describe the motion for the total system through the following equation:

$$M \frac{d^2 x}{(dt)^2} + C \frac{dx}{dt} + Kx = f$$

In this equation, M, C and K are the $3N \times 3N$ mass, damping and stiffness matrices respectively. Being a second order differential equation, fast numerical integration methods, such as the explicit Euler method, are usually used to resolve it.

Platt and Badler [29] were the first to apply this type of model to facial animation. Their system consisted of a two-dimensional surface warped around an ovoid surface that simulated the skin trough the use of linear springs. Muscle contraction is comprised of applying a force to a given region of mass points. The movement of a mass point is then propagated outward to adjacent mass points. Figure 2.6 shows some results of Platt and Badler's work.

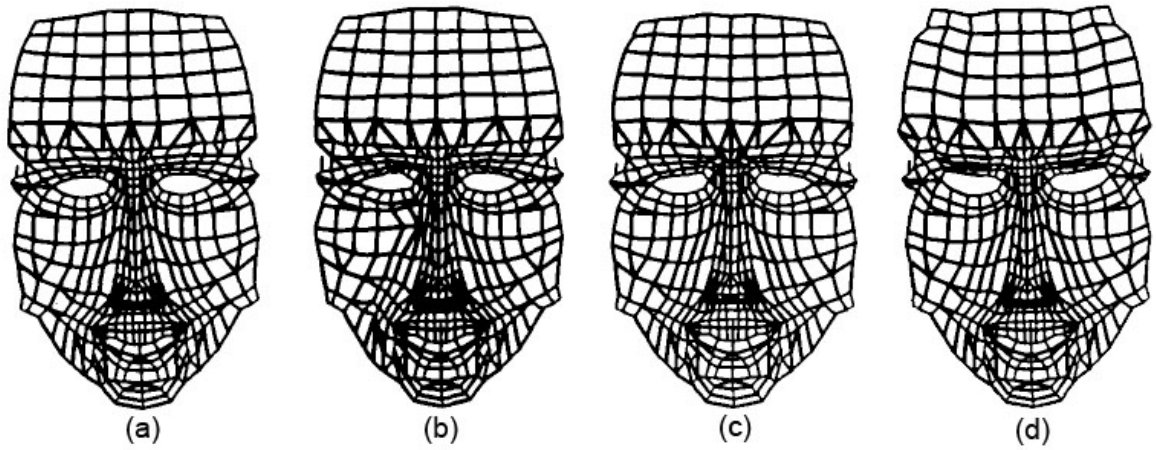

Figure 2.6: (a) Face with no expression (b) AU 6 - Cheek Raiser (c) AU 4 - Brow Lowerer (d) AU 2 - Outer Brow Raiser

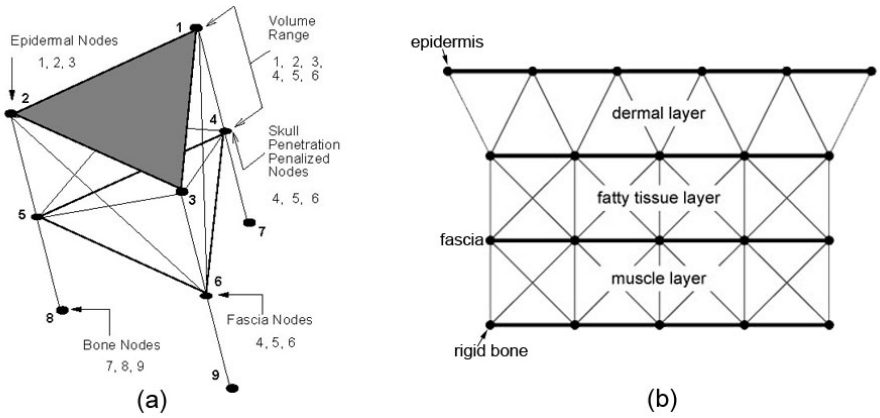

Figure 2.7: Three-Layer skin mesh sample. (a) 3D single triangle view (b) 2D cut of the three layers

Terzopoulos and Waters [32] introduced dynamic mass-spring systems to facial modeling and animation. The face is modeled by a three-layer mesh (Figure 2.7) of mass points inspired by the composition of facial tissue: the dermis, subcutaneous fat tissue and muscle. To model the different properties of the several layers, different spring constants were used.

To simulate the action of the muscles, Waters vector model [35, 23] is used in the lower muscle layer. Vector muscles are attached with one end connected to bone nodes and the other to fascia nodes. After the contraction of a muscle the mass-spring system becomes unstable and from its stabilization results the desired facial deformation. To overcome the limitation that mass-spring models do not hold constant volume, extra forces are applied. This volume

conservation allows for the automatic emergence of wrinkles, bulging and nasolabial furrows.

This work was later extended [18] to contemplate automatic fitting of muscles to meshes obtained through 3D scanners. The system not only conforms the positions and scales of facial muscles but also estimates the skull structure over which the new synthetic facial tissues model can slide.

Käler et al. [16] also proposes a mass-spring muscle model. The system has as its input an arbitrary triangle mesh that represents the skin topology. Based on this mesh, skull and jaw geometries are inferred and through the use of an editor muscles can then be created (Figure 2.8(c)).

The muscle model is based on piecewise linear representation where contraction is expressed by shortening of linear segments (Figure 2.8(a)). Along these segments ellipsoids are created to model muscle geometry (Figure 2.8(b)).

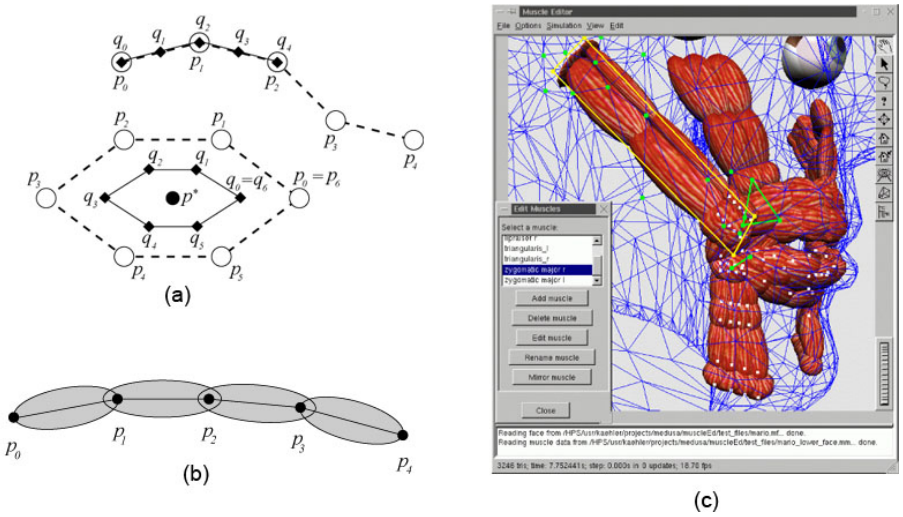

Figure 2.8: (a) Linear and Circular muscle contraction (b) Linear muscle with ellipsoid representation (c) Muscle editor

Muscles can be of two types depending on how they contract. If they contract toward an end point on the skull they are linear muscles. If they converge to a given point they are circular muscles. Muscle fibers can be combined together to form sheet muscles. When muscles contract, their bulging is simulated by scaling the height of the elipsoids.

Mass-spring based models simulate the physical properties and the dynamic changes of the face. For this reason, they are able to give very realistic results. However, they are computationally expensive, require a relatively complex geometrical model based in several

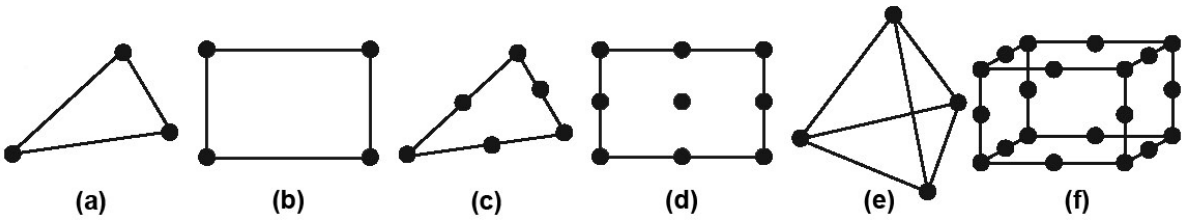

Figure 2.9: Typical 2D and 3D elements used in FEM: (a) linear triangular element with 3 nodes (b) linear rectangular element with 4 nodes (c) quadratic triangular element with 6 nodes (d) lagrangian element with 9 nodes (e) tetrahedral element with 4 nodes (f) 20-node brick element

layers and are difficult to control due to force-based functions.

Finite Element Methods

Finite Element Method (FEM) is used to find an approximation for a continuous function that satisfies some equilibrium expression such as a deformation equilibrium. In this method, an object is divided into elements that are inter-connected at discrete node points. Figure 2.9 shows some of the typical elements used in FEM.

For each of these elements a function that solves the equilibrium is found taking in account element boundaries so that continuity is preserved. As we have stated before, in mass-spring systems the equilibrium equation is discretized and solved at finite mass points. In FEM systems the solution is found by representing the desired function within each element as a finite sum of element-specific interpolation functions. For example, consider $\Phi(x, y, z)$ a scalar function that we wish to represent. Then its value at point (x, y, z) can be approximated by:

$$\Phi(x, y, z) \approx \sum_i h_i(x, y, z)\Phi_i$$

where h_i are the interpolation functions for the element containing (x, y, z) and Φ_i are the values of $\Phi(x, y, z)$ at the element's node points. Solving the equilibrium equation is then a matter of determining the finite set of node values Φ_i that minimizes the total potential energy in the body.

In [6], Basu et al. built a FEM 3D model of the lips. The parameters of the model are determined from a training set of measured lip motions in order to minimize the strain felt through the linear elastic FEM structure. With this approach Basu et al. are able to minimize the dificult control problem present in muscle based systems.

FEM provides a more physically realistic simulation than mass-spring models with less node points. However, it is based on linear elastic theory that assumes small object deformations. In materials such as the human skin where deformation can be high this assumption no longer holds. To surpass this limitation recalculations can be performed. Nonetheless, this leads to a heavy increase of the amount of computation needed at each time step.

Spline

A spline is defined by a set of control points and a set of basis functions that determine the shape of the surface curve it represents. This is a broad definition and a large variety of splines exist such as, among others, B-Splines, Catmull-Rom Splines, β-Splines and hierarchical B-Splines.

Polygon based models are vastly used to represent the face in computer graphics. This representation is for most cases sufficient to approximate face topography. However, to take in account face smoothness and curves, large amounts of data is necessary when using this kind of representation. Splines supply a way to represent continuous surfaces efficiently. Moreover, through the use of their control points it is possible to adapt the splines to simulate muscle deformation.

In the film "Tin Toy", Pixar used bicubic Catmull-Rom spline patches to model the baby's face. More recently, in the movie "Geri's Game" Pixar uses a variant of Catmull-Clark sudivision surfaces.

Wang et al. [34] propose a system that integrates hierarchical spline models with simulated muscles based on local surface deformations. The system uses hierarchical bicubic B-Splines because they offer the required flexibility and smoothness. In typical B-Spline models when a finer patch resolution is required it is necessary to subdivide an entire row or column of the surface. Moreover, additional unnecessary control points are added. In Wang et al. system, the hierarchical splines model permits for local refinements of B-Spline surfaces with new patches only being added within the specified region.

Free Form Deformation

Free Form Deformation (FFD) [30] is a technique that allows deformation of volumetric objects through the handling of a 3D cubic lattice. The object that will be manipulated can be conceived as being contained within a flexible control box with movable control points (Fig-

ure 2.10). As these controls are manipulated to deform the box, so is the contained object. To map this deformation from one three dimensional space to another, a trivariate tensor product Bernstein polynomial is used.

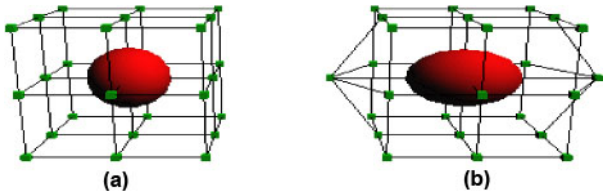

Figure 2.10: Red ball with cubic control lattice. (a) Undeformed (b) After control point movement

Extended Free Form Deformation (EFFD) allows the extension of the control lattice to a cylindrical structure that provides more flexibility to shape deformation. Rational Free Form Deformation (RFFD) is another extension to FFD that assigns weights to control points through the use of rational functions. Having the ability to weight the influence of each control point provides an additional degree of freedom.

Kalra et al. [15] developed an interactive technique to simulate facial muscle actions using RFFD. To simulate muscle actions on the skin surface, first a control parallelepiped is associated with the region that corresponds to the anatomically desired region. After this region is defined one can emulate muscle behavior by moving the control points of this lattice and by changing their weights.

Free Form Deformation has several advantages such as being able to be used locally or globally. Moreover, they can be applied to several surfaces like planes, quadrics, parametric surface patches and implicitly defined surfaces. Nevertheless, this technique does not account for volumetric changes nor for skin behavior such as wrinkling.

2.1.2.2 Pure Geometry Model

Key-frame Geometrical Interpolation

This kind of technique is perhaps the oldest and the most widely used in the field. It consists in the automatic creation of inbetween frames from a range of pre-made mesh postures (key-frames). This method assumes that the several key-frames have meshes with the same topology. This means that all key-frames are composed by meshes having the same number of faces with each face being composed of the same vertexes. Figure 2.11 shows an example of a linear

interpolation between two key-frames.

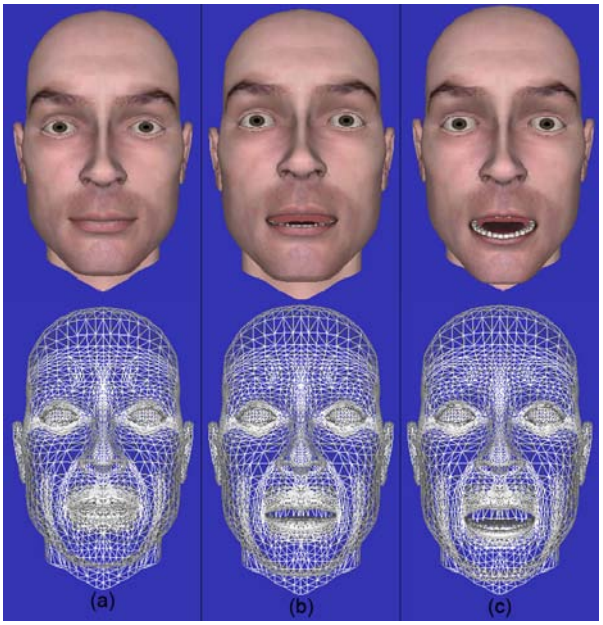

Figure 2.11: Interpolation example. (a) Origin key-frame (b) Interpolated frame (c) Destination key-frame

A point in a mesh going from position A, in the origin key-frame, to position B, in the destination key-frame, can take an infinite number of routes. To determine this path, several methods can be used depending on the desired effect. The most common interpolation functions used in geometric interpolation for animations are Spherical Linear Interpolation (SLERP), Spline Interpolation and Linear Interpolation.

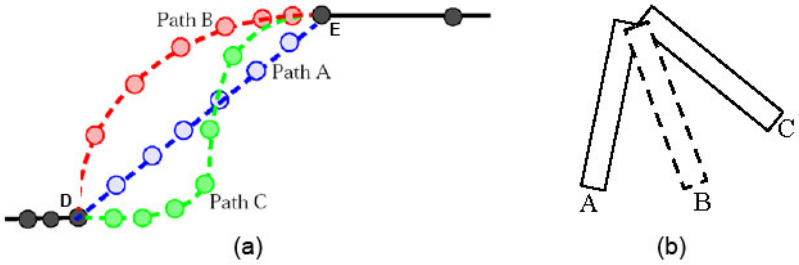

Figure 2.12: (a) Spline Interpolation (b) Spherical Linear Interpolation

In SLERPs a spherical line with a given radius is followed. The SLERP can be obtained through the following equation where θ is the angle that defines the followed arc and $0 \leq t \leq 1$:

$$SLERP(t) = \frac{\sin((1-t)\theta)}{\sin(\theta)} p_0 + \frac{\sin(t\theta)}{\sin(\theta)} p_1$$

An application for this type of interpolation is the rotation of an arm of an animated character. An example is depicted in 2.12(b) where the rectangle is rotated from position A to C having position B calculated through spherical linear interpolation.

Spline Interpolation is mainly used in trajectory calculus and in surface creation. Spline functions of several degrees can be used. In particular, cubic spline interpolations are used to maintain properties such as first and second order continuity. In 2.12(a) three possible spline interpolations are shown for a point that moves from position D to position E.

Linear interpolation is the simplest way to interpolate between two points and thus the fastest one to compute. Like the name indicates inbetween positions will follow a straight line between the start and end points. Linear interpolation can be obtained by applying the formula below:

$$LinearInterpolation(t) = (1-t) \times p_0 + t \times p_1 \ , \ t \in [0,1]$$

In facial animation geometrical linear interpolation is commonly used due to its simplicity [28]. However, as we will show in 4.2.1, to have a more realistic motion, cosine or other non-linear interpolation functions may be applied to the temporal parameter in order to achieve acceleration and deceleration effects. Besides being applied to the topology of a face, interpolation may also be used in parameters. [31] shows an example of this kind of application using a linear interpolation of the spring muscle force parameters.

Geometric linear interpolation is a good choice in facial animation for three main reasons: the length of the path taken by a point on the face is small and can be approximated to a line with virtually no error; facial expressions create flexible deformations where the relation between points is not maintained and its efficient to compute. However, the freedom of being able to create a myriad of deformations comes at a price. The designer must specify for each point what its position will be when the deformation is done. This process usually takes much time and resources.

Radial Basis Functions

The idea behind this variety of deformation is to have a small number of control points scattered over the face. Each of these control points will have an influence area in which they will affect the movement of neighboring mesh points. This influence is experienced through a normalized decay function that depends on the distance to the control point and the displacement suffered by the associated control point. This displacement can be expressed through

Figure 2.13: Noh RBF based deformation examples

the following: Let $\phi(r) : [0, rMax] \rightarrow [0, 1]$ be the radial function of influence decay and r be the distance between the neighbor point and the feature point considered where:

$$Displacement(neighbor) = f(\phi(r), Displacement(ControlPoint))$$

In [37], Noh et al. present a solution for facial animation based in RBF methods that are usually used in mesh creation from scattered data. In this work the influence region for a given control point is called a Geometry Deformation Element (GDE) and its static border points are called anchor points. To determine which vertexes are affected by a control point, two distinct distance methods are used. Euclidean distance is used when the mesh region is irregular and has no discontinuities. Otherwise, a breadth first search with origin on the control point is used. To limit the search, a maximum number of edges to be visited is set. This particular way of calculating distance is useful when holes in the mesh, such as the eye sockets, appear.

To determine vertex displacement in the region of a control point Noh uses RBFs in the sense of classical approximation theory. In this approach a multivariate function $f(\vec{x})$ is approximated by a function $F(\vec{x}, \vec{c})$, where $\vec{x}$ and $\vec{c}$ are real vectors. Here, the approximation function is given by the following equation:

$$F(\vec{x}) = \sum_{i=1}^{N} c_i \sqrt{r_i^2 + s_i^2}$$

where N is the number of anchor points + 1 (control point), r_i is the euclidean distance of $\vec{x}$ to point i and s_i is a stiffness constant that is determined to create softer deformations when control points are more scattered. The coefficients c_i are determined once per frame using the control point and anchor points positions. The displacement of the neighbor points is then obtained applying the approximation function equation. Figure 2.13 shows some deformations performed with this system.

Pasquariello and Pelachaud present in [24] a 3D facial model compliant with Mpeg-4 specifications that uses RBFs to perform facial deformation. Mpeg-4 is referred in more detail in

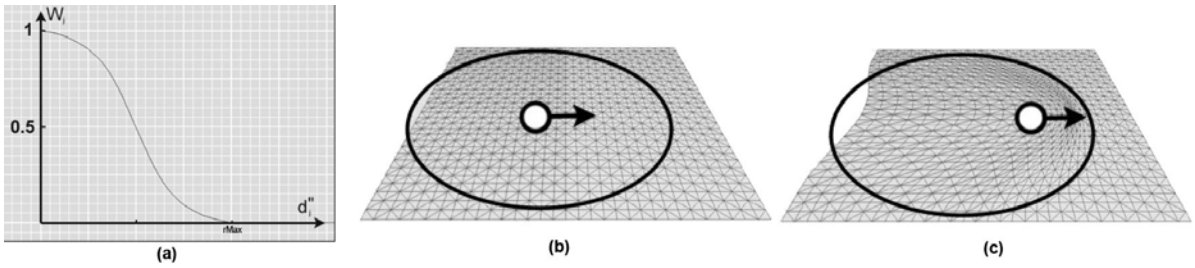

Figure 2.14: (a) GRETA RBF weight function (b) Deformation with null FAP intensity (c) Deformation with maximum FAP intensity

section 2.2.5 but there are two concepts that are important to explain here. In Mpeg-4 deformation of the skin is attained through Facial Animation Parameters (FAPs). These consist essentially in the movement of specific points of the face called feature points. In light of RBF terminology used so far, feature points are control points.

To create the actual deformation the radial basis function of decay depicted in Figure 2.14(a) is used. For example, given a vertex $\vec{v}_j = (x, y, z)$, that is influenced by a FAP_i, in the xx axis direction, its displacement is obtained through the following formula:

$$\Delta\vec{v_{i_x}} = W_j \times FAP_i$$

The system also performs simulation of furrows and bulges. This is achieved by having the model divided in different regions. When a given FAP influences two distinct regions a change of the surface normals is done along the regions border.

Kshirsagar et al. suggest in [17] a deformation method based in feature points. Although they do not use RBFs in the exact sense described above, the ideas behind their method are closely related. Like in RBFs, deformation in this approach is driven by the displacement of certain feature points (or control points as previously named). Here feature points also have an associated region of influence. However, unlike typical RBFs systems this region and its associated vertex weights are determined not only by the distance to the feature point but also by factors such as mesh topology and feature point distribution. This is done by creating all the regions at the same time in a iterative way. For each feature point its region grows one step at a time until it reaches another feature point region. This process leads to a division of the mesh in a Voronoi diagram where feature points are the generation points. This process is depicted in Figure 2.15

After this process, vertex weights are computed. The displacement of a given region vertex

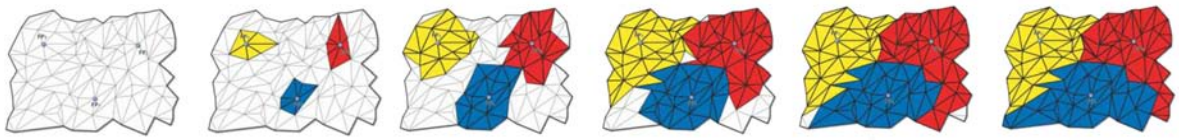

Figure 2.15: Construction process of feature point regions. Evolution from left to right.

is not only influenced by the feature point of that region but also by feature points that have a border with that region. These feature points are also known as neighbor points. In order for a neighbor point to influence a vertex that does not belong to its region it must meet with yet another condition. Consider a vertex V that is in the influence region of feature point P_1 which has a neighbor feature point P_2. Then, V is influenced by P_2 if $\angle P_2P_1V < \frac{\pi}{2}$. This process is computationally heavy but it is performed at a preprocessing phase. When all weights are calculated deforming the mesh is a function of the feature points displacement, vertex weights and distance of the vertexes to the feature points.

A great advantage of this method is that creating and exchanging animation parameters between models can be done with very little effort. While key-frame animation only makes sense for a given specific model, being strongly linked with it, the radial function deformation is at a somewhat higher level of abstraction. For each model what needs to be defined is which actual point in the model corresponds to a given control point and what distance will be considered for a point to be its neighbor. Defining a set of control points and corresponding distances is much less time and resource consuming than designing complete key-frames giving the displacements for each and every point that is intended to move.

2.2 Parameterization

Broadly speaking, parameterization is the process of deciding the set of essential differences between objects of a given class. An individual object can then be identified through the particular values of these differences. These differences are the parameters. A simple example is the class of Sphere objects. Spheres may differ between them in diameter, color, material, etc. If we vary these parameters we can create various unique spheres. In the context of facial animation systems the class in question is the class of facial expressions.

Given this, one is faced with the question “What parametrization should be used in facial animation systems?”. Over the years several solutions were suggested for this issue. In this

section we will present some of the most relevant.

2.2.1 Parke

Parametrized face models were introduced by Parke [21] in a attempt to solve the problems that arise from classic key frame interpolation approaches. His idea was to create a model that was able to represent a wide range of faces and facial expressions with few control parameters. From this model one could then produce a myriad of animations with little effort.

Parke considers two types of parameters, conformation and expression parameters. The first are related to the way the face looks like. It describes properties such as jaw width, forehead shape, nose length and width, cheek shape, chin shape, neck shape, eye size and separation and face proportions. Expression parameters pertain to the performing actions of the face like eyelid opening, eyebrow arch, eyebrow separation, jaw rotation, mouth width, mouth expression, upper lip position, mouth corner position and eye gaze.

2.2.2 Facial Action Coding System

Facial Action Coding System (FACS) is a system created by Paul Ekman and Wallace V. Friesen [13] that set a milestone in facial behavior categorization. Its intent is to supply a way to describe comprehensively all facial expressions that are possible to observe in the human face. For that, it focuses in the visible changes that occur in the face during a facial expression and in their relation with the underlying muscles. It is important to stress that only visible changes are considered in FACS. Changes that are too subtle to be observed are not taken into account. In order to describe what the face is able to perform, FACS uses *action units* (AUs) that correspond to the contraction of one or more muscles.

So, annotating a given facial expression in FACS consists of pointing out which AUs are active and to what degree. The level of intensity of an AU varies between *A* and *E*. Level A refers to a trace of the action; B, slight evidence; C, marked or pronounced; D, severe or extreme; and E, maximum evidence.

While FACS was not created with facial animation systems in mind, many such systems adopted it as a solution to control facial movement through the manipulation of muscle actions. This approach provided a "ready to use" parameter set whose substance already had been proven valid.

Although FACS offers a good set of expression parameters, that allows for the creation of

a vast number of facial expressions, it has also some drawbacks when used in an animation system. FACS does not specify information such as the timing of muscular contraction which is essential to the dynamics of animation. Also, the AUs contraction degree is specified in a qualitative manner. This fact makes it hard to determine exactly what geometric transformation should occur when interpreting a given value. Moreover, FACS detail on the mouth region is poor when considering the needs for speech synchronization.

2.2.3 Abstract Muscle Actions

Magnenat-Thalmann et al. [19] presented a system that uses a parametrization inspired by FACS. The system uses Abstract Muscle Actions (AMA) procedures that work on specific regions of the face that are defined when it is constructed. Similarly to FACS AUs, AMAs simulate the action of one or more muscles. However, unlike FACS AUs, each AMA is dependent of the others and the order by which they are executed results in different expressions. Also, AMAs intensities are quantified and normalized between -1 and 1 or 0 and 1 instead of having a qualitative value like FACS AUs.

AMA procedures take into special attention lip movement. In order to handle lip motion complexity, this approach decomposes it into smaller simpler motions that correspond to single AMA procedures. According to Magnenat-Thalmann et al. [19], having a limited, yet expressive, set of dedicated AMA procedures allows for the illusion of genuine lip movement without its complexity.

In this work, it is also proposed a higher level of control parameters which is named *Expression Level*. This level is built upon the lower level of AMA procedures and supplies two types of expression parameters: phonemes and emotions. Here, a phoneme is a facial expression which only creates motion in the mouth area being associated with speech. Emotion parameters act on any part of the face and cause motions related to "emotion" such as a smile or a kiss.

2.2.4 Minimal Perception Actions

Kalra et al. [15] created a parameterization where Minimal Perception Actions (MPAs) take the main role. MPAs define which visible effects are caused by the action of one or more muscles in a given region of the face. There are also some MPAs that correspond to non-facial muscle action such as turns of the head and eye movement. Similarly to what happens in AMAs, MPAs can be grouped together to create higher level parameters such as the facial

display of an emotion. Again, like AMAs, the intensity of each MPA is normalized between 0 and 1 for unidirectional MPAs, and between -1 and 1 for bidirectional MPAs.

In this work, the actual deformation caused by a given value of an MPA is model dependent. This means that MPAs can lead to very different animations between two different models even if using the same animation parameters values.

2.2.5 Mpeg-4 Facial Animation Parameters

Mpeg-4 [10] is an object-based multimedia compression standard that allows the encoding of different audiovisual objects in a scene independently. Unlike previous versions of Mpeg where only audio and video data signals were considered, version four takes in account some synthetic objects. The face and the body are two of the contemplated synthetic objects.

Although the standard is vast, defining many features such as encoding, VRML support, object-oriented composite files, support for externally-specified Digital Rights Management and various types of interactivity, here we will only focus on the specification of the face parameter set.

Mpeg-4 face parametrization [8] is composed of several key concepts: Neutral Face, Feature Points (FPs), Facial Animation Parameters (FAPs) and Face Animation Parameter Units (FAPUs).

In Mpeg-4 a facial animation results from the deformation of a neutral face. This neutral face can be seen as a face that has no contracted muscles and is defined by the following properties:

- All face muscles are relaxed
- The gaze is in the direction of the z-axis
- The eyelids are tangent to the iris
- The pupil is one third of the diameter of the iris
- The upper and lower lip are in contact
- The line of the lips is horizontal and at the same height of the lip corners
- The mouth is closed with the lower and upper teeth touching each other
- The tongue is flat, horizontal with the tip of the tongue touching the boundary between the upper and lower teeth

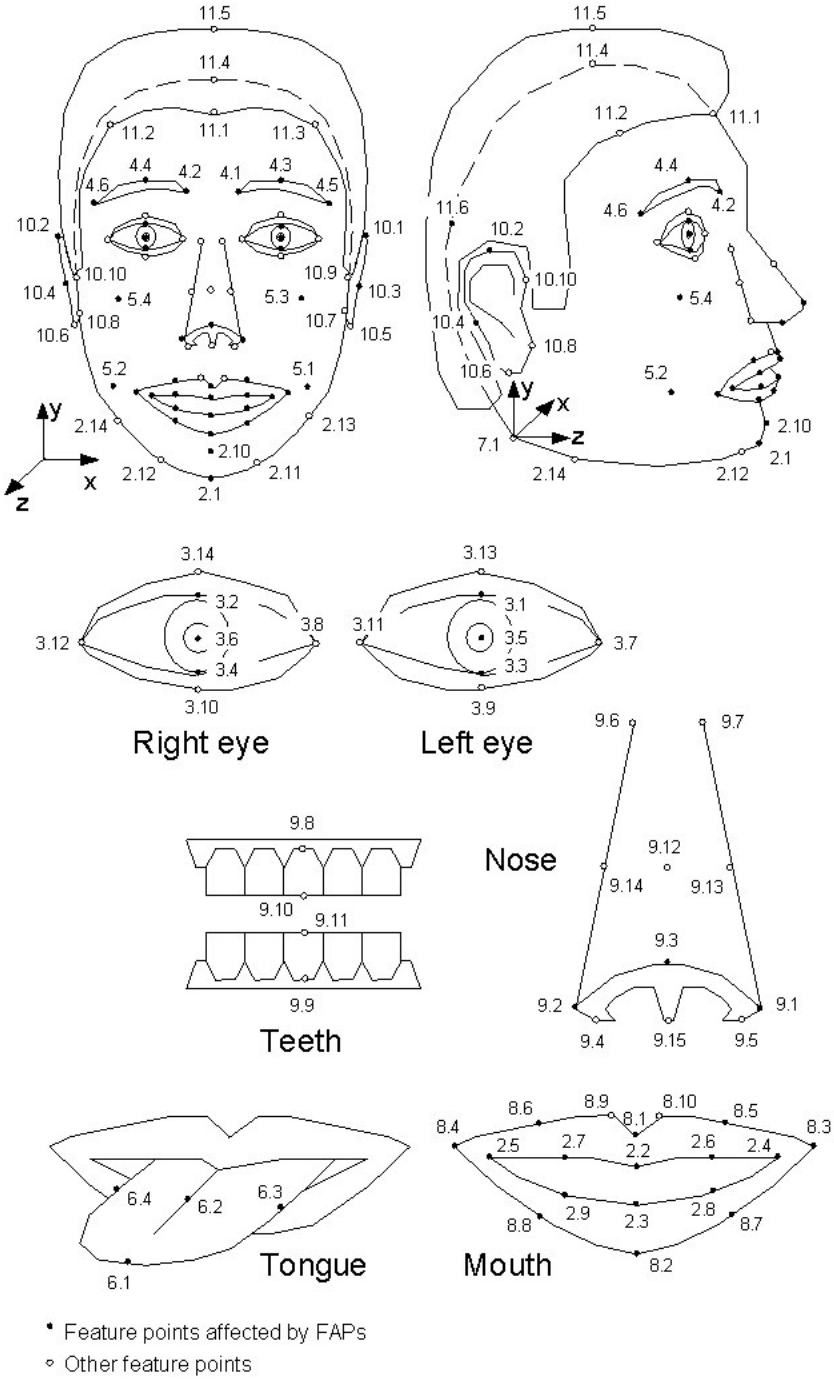

Figure 2.16: Mpeg-4 Feature Points

The morphology of the neutral shape can be acquired by a Mpeg-4 facial animation engine through the use of FPs. Knowing the position of these key points, the engine can adapt a generic face model to fit the desired shape. Alternatively, a compliant Mpeg-4 facial animation engine must be able to receive an indexed face set that defines the shape of the face. This latter option is aimed at being used when the fitting of the generic model is not precise enough. When providing the full face shape one must still indicate where the FPs are located. Figure 2.16 depicts Mpeg-4 Feature Points.

Besides of specifying shape, FPs also play a role in facial animation. In fact, most deformations in Mpeg-4 are created through the movement of FPs. The parameters associated with these movements are low-level FAPs. A low-level FAP defines a direction in which its related FP moves when the FAP value changes. Depending on the low-level FAP the movement can be uni or bi-directional. How do neighboring points of the FPs move due to FPs displacement is not specified in the standard, being left up to the developer implementation.

The displacement of a FP depends on the value of its associated FAP. This poses a question of how much should a FP move due to a given FAP value. This question is specially pertinent if we consider that the parametrization should be robust enough to support different face shapes. Thus, the same FAP value should create a similar visual effect on different faces. Mpeg-4 solves this issue by using measures that are related to key facial features of the particular face used. These measures are the FAPUs and they are depicted in Figure 2.17.

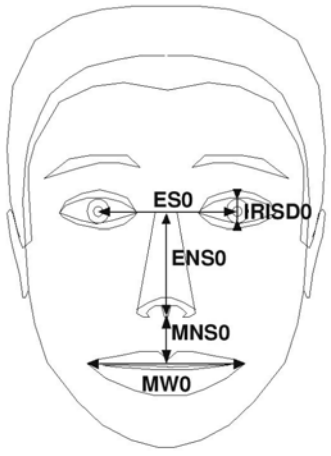

Figure 2.17: Mpeg-4 Face Animation Parameter Units defined on a neutral face

In addition to low-level parameters, Mpeg-4 defines two high-level parameters, visemes and expressions. The first are related with speech being a viseme the visual equivalent to

a phoneme. The set includes 14 static visemes that are considered to be the most clearly distinguished ones. Speech coarticulation is achieved through the blending of the current and next viseme using a weighting factor.

The latter define the six universal facial expressions identified by Ekman [11]. Each one of these correspond to an emotion of anger, joy, disgust, sadness, fear or surprise. A small description of how the expressions should look like is given by the standard to guide the designer that will model them. During animation, two facial expressions can be active simultaneously.

Regarding the notion of FAPUs, it does not apply to high-level FAPs. Instead, high-level parameter values vary in the interval $[0, 63]$ that is considered to have enough steps for smooth transitions.

Mpeg-4 does not specify how to handle using both high-level and low-level FAPs at the same time. Doing this, might lead to unexpected visual representations of the face when both level FAPs influence the same region.

2.3 Analysis

2.3.1 Deformation

All shown approaches have their advantages and disadvantages. Image manipulation permits the representation of skin vascular effects and good 2D animation. This approach can also be used in conjunction with geometric manipulation techniques to increase the quality of the animation. However, this method requires large amounts of data and is not easily generalizable for different models.

Physics based paradigms tend to be computationally heavier than pure geometric approaches. Generally speaking, the closer a method is to simulate real life tissue dynamics the less efficient it is. Both Mass-Spring and FEM models account for properties of the skin such as multiple layers that react differently to muscle contraction. Nevertheless this commitment to reality simulation comes at a cost. Besides being computationally heavier, these approaches require more detailed and complex models that are harder to create. Vector muscles present an interesting solution. Instead of simulating the complex process of muscular contraction, force propagation and consequent skin deformation, they use a relatively simple geometric function to model the main effects that occur on the skin due to the muscular contraction. This leads to a more efficient approach and a less complex face model that still yields good visual results.

Unfortunately vector muscles are also not without drawbacks. Each muscle represented this way still requires fine tuning of its characteristics and the achieved quality is not as good as Mass-Spring or FEM models.

Free Form Deformation is easily applied on many types of surfaces and can be used locally or globally. This approach supplies a comprehensible way to control face deformation due to its geometric nature. Splines supply a way to represent continuous surfaces efficiently. Although these methods supply a way to represent and manipulate the face efficiently the visual results achieved by them are not as realistic as physic based ones.

Key-framing method allows for the animation of any modeled face posture with high quality results and low computing power requirements. Unfortunately, it demands that every desired facial expression be painstakingly hand made. This can be eased to some extent by using blending of postures. RBF approach, unlike typical key-framing, focuses on local regions of the face. By using a generic function that depends on the distance of a point to its control point and on the control point displacement, RBF allows an easy exchange of animation parameters between face models.

2.3.2 Parametrization

Parke and Waters [22] presented several properties that an ideal parameterization should have. These were later reiterated and extended by Pandzik and Forcheimer [9] when they argue if Mpeg-4 facial parametrization is up to an ideal one. These metrics take in account aspects such as the ability for a parameterization to represent any face and expression, how easy a parameterization is manipulated and if it provides predictable results. Next we describe these metrics in detail and evaluate the shown parameterizations with them.

Range of possible faces

This property is related to conformation parameters. Ideally, these should allow for the representation of any given face. Attaining this goal strictly by the use of conformation parameters is extremely difficult even when one just considers human faces. When considering a broader range of faces such as human, cartoon and animal, the difficulty steepens even further.

Since FACS was not initially thought to be used in facial animation systems, this metric does not apply to it. FACS only contemplates a way to describe facial motion and not how to describe face physiognomy. Parke's parameterization supplies conformation parameters that

allow for some flexibility in the representation of possible faces. However, this flexibility is limited and considers only human faces. An advantage of this method is that expression parameters are already associated with the face model and ready to use. Both AMA and MPA parameter sets do not consider conformation parameters. Instead, parameters are associated with designer modeled face meshes. Thus, setting up a face in these cases consists of creating a face mesh, setting up the regions that expression parameters influence and configuring the deformation techniques associated with each expression parameter. Although this approach requires more setup work than the use of conformation parameters, it permits for the representation of virtually any desired face. Mpeg-4 facial parameter set uses a hybrid approach. It defines the use of a generic face model that can be shaped through the use of FPs as conformation parameters but also permits the use of any mesh. By doing so, one can use the generic model for some cases and supply a user defined face mesh when the generic model fitting is not sufficient. Thus, Mpeg-4 has the same advantages and disadvantages of the above mentioned parameterizations depending on the choice of the user.

Range of possible expressions

The number of expressions a human is able to perform is vast. Furthermore, similarly to the previous point, non human faces and exaggerated facial motion should also be taken into account.

The very basis of FACS is the categorization of all the visible motions in the human face. Nevertheless, it uses qualitative measures to describe these motions. This is a limitation to the number of expressions one can make through a facial animation system. Parke's parameterization was created with the goal of being expressive with few control parameters. Unfortunately, the number of expression parameters is rather restrictive leading to a relatively small number of possible expressions. AMA procedures and MPA have roughly the same scope of possible expressions that is somewhat in between of Parke's parameterization and Mpeg-4 FAP set. Mpeg4 provides a good range of possible expressions by covering the main facial articulation in detail, including exaggerated motions for cartoon animation. None of the presented parameterizations takes into account explicitly other expression factors such as wrinkling, furrows and vascular expressions.

Ease of use

The number of parameters, their complexity and "intuitiveness" defines the ease of use of a parameterization.

Parke's parametrization has a relatively small number of simple parameters being easy to use. FACS, AMA and MPA have a larger range of parameters than Parke's parameterization. The nature of the parameters is similar in complexity between them and is easily understood.

Mpeg4 FAP set has a large number of parameters which makes it harder to use. However, low-level parameters are easily understood because they are based on the movement of face points and high-level parameters, such as visemes and basic expressions, provide an abstraction for common needed functionality.

Subtlety

Even the slightest change in the face might have meaning. Our brains are highly skilled at finding and interpreting these differences and we use them in our daily conversations. Thus, a parameterization should be able to represent with precision fine movements.

Natively none of the parameterizations contemplates all of the subtle changes that can happen in a face. AMA gives special attention to the lip region allowing for detailed movement in that area. Minimal Perceptible Actions are defined to, as the name indicates, describe barely noticed motions of the face. Mpeg-4 follows MPA idea and uses parameters that use sufficiently small units to allow the representation of barely detectable movements. However, as previously said, none uses an explicit representation for subtle expression such as furrows, wrinkles and vascular expressions.

Orthogonality

In order to adjust parameters independently, a parameter should not influence another. That is, when one changes the value of a given parameter, adjustment of another one should not be required.

Having a completely orthogonal parameterization is virtually impossible. There will always exist regions of the face that are influenced by more than a parameter. However, some degree of orthogonality can be achieved. AMA procedures are explicitly non-orthogonal since the order by which they are activated is pertinent to the end result. FACS has different parameters, like AU2 outer brow raiser and AU4 brow lowerer, that are applied in the same region but have

opposed motions. This same behavior exists in MPA. High-level parameters typically affect the whole face, or a large region of it, and therefore enter in conflict with low-level parameters. Nevertheless, Mpeg-4 low-level parameters were designed to be orthogonal amongst them to a good degree.

Basis for a higher-level abstraction

High-level abstractions of low-level parameters patterns are useful to reduce complexity. Therefore, parameterizations should supply aggregation of low-level parameters into higher level controls such as visemes.

AMA, MPA and Mpeg-4 supply high-level parameters for the abstraction of visemes and some emotional related expressions. Parke's parametrization and FACS only contemplate low-level parameters. It can be argued however that higher level parameters can be built on top of a given parameterization. For this to be feasible, the low-level parameters must allow for a vast range of possible expressions.

Predictability

The outcome of all parameter combinations should be predictable. This means that the behavior of every point in a face must be defined in the parameterization. Doing so is extremely difficult, if not impossible, especially if we consider that a parameterization is to be applied at all types of faces.

Furthermore, considering the disassociation of the parameter semantics and actual implementation, means that the exact end result may be different for the same parameterization in two different implementations. At first, this may strike us as a problem but it is not necessarily one. The idea is that a parameterization should capture the essence of each parameter. It is simply not feasible to describe completely to every detail the dynamic changes of a complex system like the face. By capturing the essence of each parameter one can have sufficiently predictable results between implementations.

From this, we conclude that the more coupled a parameterization is to its implementation, the more predictable it is. Parke's, AMA's and MPA's parameterizations were thought of with a given implementation in mind. Therefore, they are relatively predictable. Mpeg-4 on the other hand, leaves much for its implementation to decide and opts for a semi-predictable parameterization. FPs movements are completely predictable while higher level parameters

exact behavior and FPs neighbor movements are left for the implementation. Since FACS was created to be a categorization system and not a parameterization for animation system, it is completely decoupled from a given implementation. Hence, FACS is the least predictable of the discussed parameterizations.

Portability

The parameterization should be easily applied to different face models giving the same result for the same parameters values.

As previously said, Mpeg-4 FAPs use FAPUs in order to be normalized with respect to the face they are applied to. Thus, the same parameters values in different faces give similar results. However, the other observed parameterizations are normalized within an absolute interval and therefore not easily portable.

Measurable parameters

Parameters values should be expressed in physically defined units. This allows for them to be measured visually on a face. This property is particularly useful for applications that use facial features tracking from video or markers.

Mpeg-4 is the only analyzed parameterization that takes into account this property. FACS parameters have qualitative values while the remaining have a normalized quantitative value that does not relate to any exact visual feature.

2.3.3 Summary

In this chapter we presented several facial animation systems which were analyzed in light of two aspects: deformation technique and parameterization. From this study we concluded that all deformation techniques have advantages and disadvantages. Image manipulation techniques provide an efficient way to represent per pixel changes but are not feasible for all deformation representation purposes. Geometric manipulation approaches provide a broad range of solutions which vary in degree of quality, computational requirements and setup complexity. Something similar is verified regarding parameterizations. None is able to completely satisfy all of the presented requisites with each being better suited for some aspects than others.

Chapter 3

Conceptual Model

This chapter describes the overall idea of the developed facial animation system. We start by explaining how parameter decoupling is achieved. To do so, what is understood to be a parameterization, parameter and parameter instantiation is presented. Next, we give an overview of higher level controls of parameters such as animations and animated synchronized speech.

3.1 Parameter Decoupling

In the previous chapter several parameterizations were presented and analyzed. From these, Mpeg-4 supplies without a doubt the best generic solution. This is probably so, because this parameterization was built upon the knowledge of its predecessors. It was created with the mentioned metrics in mind and had the contribution of a broad number of the community members. For these reasons it is the first facial animation parameterization standard. Nonetheless, no parameterization exists that is ideal. By improving a given desired property one is bound to make another worse. For example, by increasing the range of possible expressions one needs to increase the number of parameters making the parameterization harder to handle. Given this, one can argue that the best overall parameterization might not be the best for a specific task. The mentioned metrics should be balanced to fit the specific requirements of where the parameterization is to be used.

For these reasons, we chose not to adopt any particular parameterization. Instead, the developed system allows for the definition and use of user specified parameterizations. This choice follows the idea of parameterization *decoupling* from implementation.

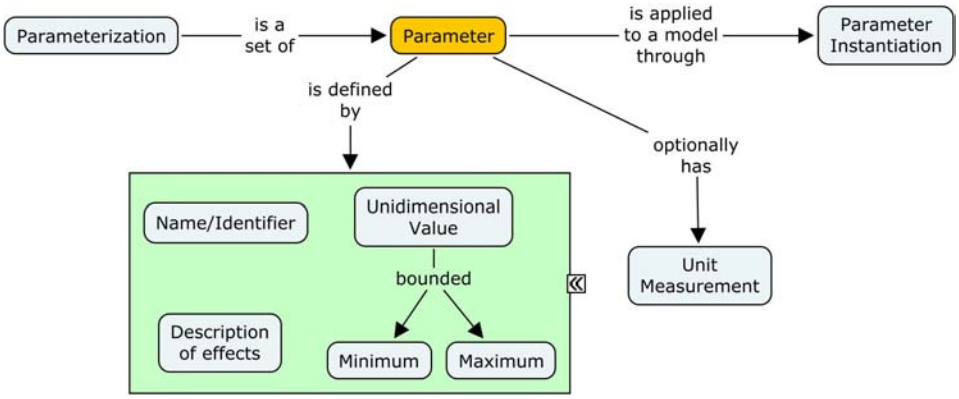

Figure 3.1: Parameterization decoupling from instantiation

Figure 3.1 displays how the concepts of *parameterization*, *parameter* and *parameter instantiation* are related in this work. In the developed system, a parameter represents solely the conceptual effects we wish to describe and not how exactly they are achieved. We took into account the previously analyzed parameterizations and modeled their properties into our parameter notion. The result is a parameter composed of a *name*, a *description*, a bounded unidimensional *value* and an optional *unit measurement*.

A parameter's name has the exclusive purpose of identifying it within a given parameterization. It must therefore be unique in a parameterization.

The parameter's description is the core of a parameter. It delineates what should occur when the parameter value varies. This description can be as simple or as complex as one desires it to be. For example, one might merely say "The central region of the left eyebrow is raised.". Contrasting, the same idea could be conveyed by stating "The parameter causes an upper motion of the left eyebrow central region. This movement is caused by the contraction of the frontalis major muscle. Wrinkles on the forehead region above the left eyebrow should be increasingly visible as the parameter value increases.".

The parameter value indicates how active the parameter is. It is important to notice that this value is unidimensional. Although it is possible to conceptualize a multidimensional parameter, such as the position of a given point in a face, we opted for the use of unidimensional parameter values. This choice still allows modeling multidimensional parameters through the use of several unidimensional parameters. Moreover, we believe it simplifies parameter control. The parameter value is bounded between a minimum and maximum value. When a parameter value is zero it means that no change is performed by that parameter. As expected, the

parameter effects reach its maximum when the parameter value has maximum value. At first, one might think that the minimum value should be necessarily zero. However, in certain cases it is useful to think of a less than zero parameter value. For instance, in parameters that have a directional meaning such as *RaiseLeftEyebrow*. Instead of creating another parameter for the lowering of the eyebrow one might use the same parameter but with a negative value. By doing so, one can reduce the number of parameters and thus enhance the parameterization ease of use.

The unit measurement is analogous to Mpeg-4 facial animation parameter units. This measurement does not specify an actual value. Instead, it specifies the unit of a parameter value as a model related distance, such as eye separation, or as a known notion such as degrees. Using this property is optional but its use allows for greater portability of parameters between models. This optional nature is due to this property not being applicable to all parameters. An example of such are the high-level parameters that correspond to a complete facial display such as the display of Joy.

While parameters specify ***what*** changes in a face, parameter instantiations describe ***how*** these changes take shape. With this clear separation one can focus on the problems of each process independently. When choosing what parameters to use one considers the already mentioned parameterization metrics without concerns for application details. Furthermore, parameter instantiations can be freely interchanged without affecting the processes that control the parameters. New and better instantiations can be created and used seamlessly.

However, when faced with such a generic definition of parameter, it is not a trivial task to identify and implement the necessary myriad of techniques to represent all possible effects that may occur in a face and thus support any conceivable parameter. Each type of parameter instantiation will then make use of a given method or technique to achieve a desired effect.

Since it is not feasible to supply a way to model all possible parameters in the scope of this work, we are forced to select methods that are as representative as possible. Furthermore, besides taking into account what a given method can achieve, we must also consider practical requirements such as real-time performance, easiness of modeling and visual quality.

Our first self imposed restriction is to not *explicitly* support conformation parameters. Although some of the chosen techniques can be used to model conformation parameters their application was not developed with this in mind. Therefore, the use of the supplied parameter instantiations for this effect might yield unexpected results. The choice of not supporting

conformation parameters is backed up by the usual limitation these evidence. Some structural changes, such as modifying face width or nose length, can be performed with conformation parameters but deeper shape alterations are not feasible through their sole use. We take into account a large range of faces from human to cartoonish animals which may vary greatly not only in their shape but also in their geometrical complexity. This precise modeling can only be performed by specialized software such as 3D Studio Max [1] and Maya [3].

Hence, we have focused our efforts in providing parameter instantiations that use techniques that allow the modeling of expression parameters. Figure 3.2 displays the chosen techniques for this purpose. Elements in green represent techniques that led to more than one parameter instantiation.

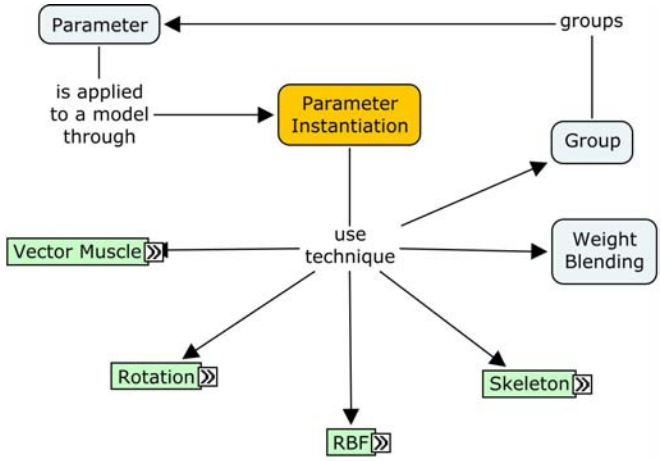

Figure 3.2: Parameter Instantiations

Some of the used techniques come from, or are based in the ones that were presented in the previous chapter. Vector muscles provide a way to mimic skin tissue behavior without demanding too much computing power. Weighted blending is based in key-framing that is a commonly used technique by 3D animators. It supplies a manner by which high quality expressions can be achieved through a powerful, efficient and familiar technique. Regarding RBF we opted for Kshirsagar et al. [17] method. This solution offers a reasonable lightweight process that is easily setup for different face models. All of these techniques were modified to some extent to either improve them or to better allow integration with the parameterization specification.

Furthermore, rotations and skeleton skin meshing techniques are used to control eye, head and jaw movement. Finally, group parameter instantiations are intended to group several

parameters into a single one.

The ensemble of these methods makes the system ability to represent expression parameters, flexible and powerful. All techniques and their relation with parameter instantiations are explained in detail in Section 4.1.

3.2 High Level Control

Precise manipulation of the face can be achieved by direct control of parameter values. However, there are a multitude of situations where such a fine control is not required or where a higher level of abstraction is desired. Mechanisms that, facilitate the variation of one or more parameter values over time, provide a way to play predetermined sequences of these variations and others that supply yet higher level control such as speech synchronized facial animation are useful in enhancing system expressiveness.

We consider two base animation functionalities to be important. The ability to have an animation library where previously created animations can be called upon and the capability to create on-demand animations for simple facial motion behavior. Both predetermined and on-demand animations are key-frame based animations (Figure 3.3).

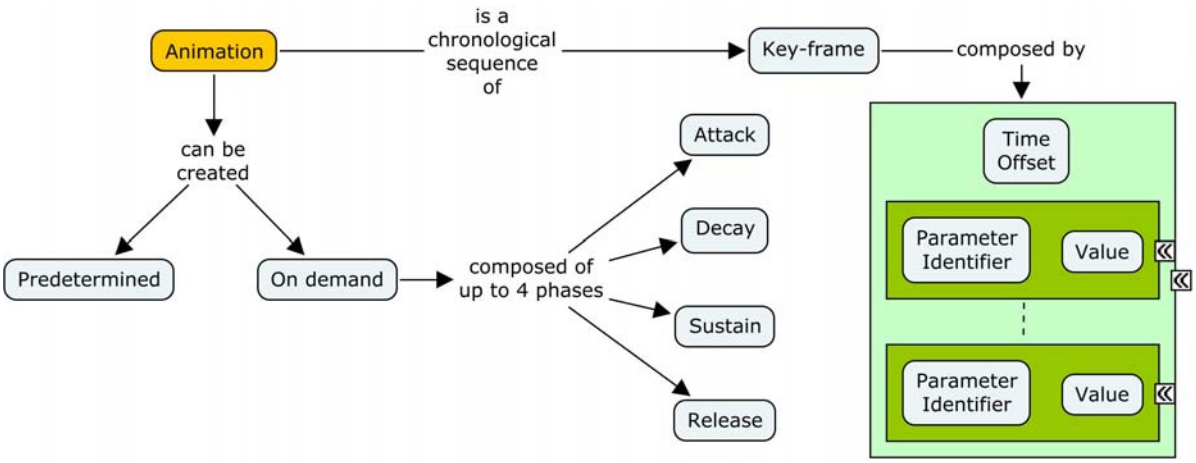

Figure 3.3: Animation

The concept we use in key-frame animation is similar to the technique of key-frame interpolation discussed in 2.1.2.2. Here, like in the previously described technique, an animation is defined by a set of key-frames that have a chronological order. Each of these key-frames defines how the face should look like at a given time offset after the animation has started.

The difference is that instead of defining, and interpolating, the facial display through vertexes positions we use parameter values. So, a key-frame defines what parameters are used and what are their values for the specified time instant. This approach is less labor intensive than full geometry interpolation and permits an animation to be directly reused in different face models that have the same parameterization.

The typical functionalities of a key-frame based animation player are supported by the system. It allows for animations to be played, paused, resumed and stopped. Furthermore, animations can be played with following styles:

- Forward: The animation is played going through key-frames as their chronological order was specified.
- Backward: The animation is played in the reverse direction of the forward play method.
- Ping Pong: The animation is played forward and then backward.
- Reverse Ping Pong: The animation is played backward and then forward.

Animations can also be played in loop where the animation cycle is repeated forever until it is stoped.

Unlike typical pre-determined animations the first key-frame of an animation is not required to have a time offset equal to zero. When an animation is played a snapshot of the current parameter values is taken and the zero time offset key frame created. Parameter values are then interpolated to the target key-frame specified values. This allows for a continuous flow of the facial motion without the need of animation state machines. These are commonly used for specifying the transitions between animations in order to guarantee that the geometry state is the same between animations. This approach is feasible due to the unidimensional parametric nature of our system.

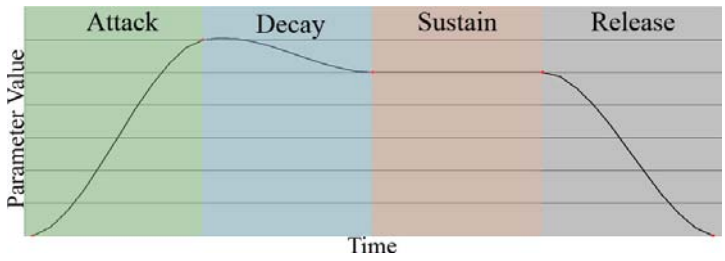

Figure 3.4: Attack, Decay, Sustain and Release animation phases

Attack Decay Sustain Release (ADSR) animation provides the on-demand method and intends to easily represent a frequently used motion behavior. Like its name implies, it is composed of four phases where the value of a given parameter raises (Attack), suffers a small loss (Decay), maintains value (Sustain) and finally tends to a neutral value (Release). In Figure 3.4 an example of the envelope is shown.

When a ADSR animation is played for a given parameter the corresponding key frames are created at that moment. This animation is then played following the interpolation process explained in section 4.2.1.

Alternately, one can choose to suppress one or more of the phases. For example, by not using the Decay phase we use another commonly used motion behavior model of Onset (Attack), Apex (Sustain) and Offset (Release).

Synchronized animated speech is perhaps the mostly used functionality in a facial animation system. In our system this is achieved through the use of a mapping between phonemes and parameters that represent visemes. The key-frame timings that construct the speech animation can be supplied directly by the user or by a text to speech engine. These and the speech wave are played together to accomplish the final result (Figure 3.5).

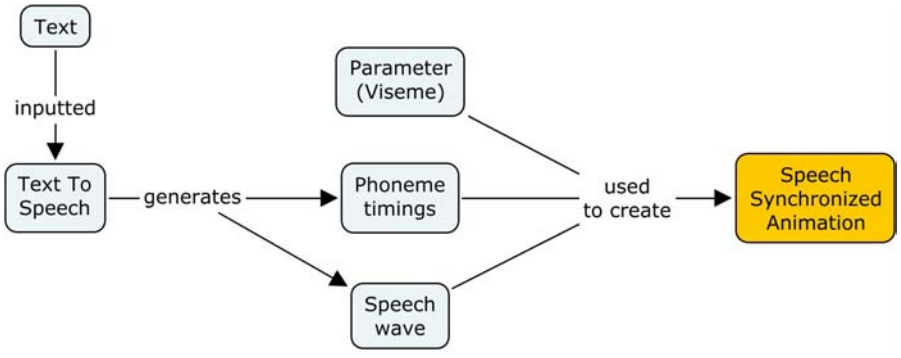

Figure 3.5: Synchronized Animated Speech

We found it to be interesting to provide a mechanism that allows the simulation of some sort of unpredictability. To achieve this goal we use Perlin Noise [25] to create idle facial motions and to influence animations (Figure 3.6).

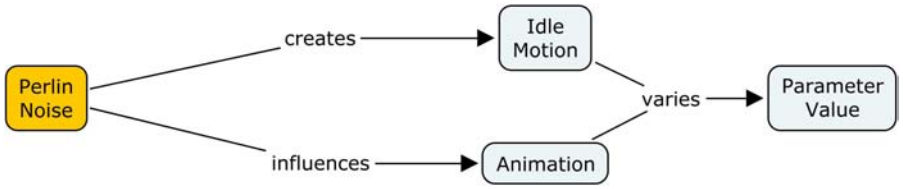

Figure 3.6: Perlin Noise

3.3 Summary

Parameter *decoupling* plays a central role in the developed system. In order to attain this, we have created an explicit differentiation between *what* a parameter represents and *how* concretely it is mirrored in a face model. A parameter is characterized by an unidimensional value whose variation has an intended effect. This effect can be modeled through a series of parameter instantiations which in turn make use of different techniques. This detachment permits not only the definition of user parameterizations but also the choice of the best techniques for a desired effect.

Direct control over parameter values grants accurate handling of the facial display. Nonetheless, higher level controls are also desired. The use of key-frame and phase based animations create an abstraction of parameter values variation over time. Synchronized animated speech can be created through the use of a sequence of phoneme timings, speech wave and parameters that represent visemes. Finally, Perlin Noise can be used to induce some "natural" motion qualities to the face.

Chapter 4

Implementation

In the previous chapter the difference between a parameter and parameter instantiation was presented. In addition, the notion of high level controls such as animations and synchronized speech was introduced. These concepts are organized in a three-layer architecture (Figure 4.1) resembling the one proposed by Perlin and Goldberg in [26].

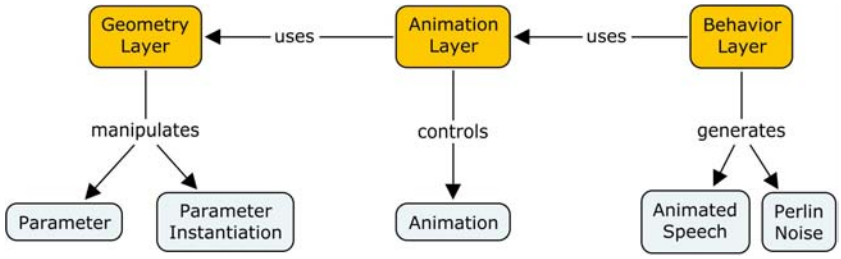

Figure 4.1: Three-Layer Architecture

This structure organizes information in a hierarchical manner regarding its level of abstraction. At the lowest level is the geometry layer that is responsible for the direct control of parameters. Data such as the synthetic character model, parameter instantiations and geometry manipulation is handled by this layer. Above it sits the animation layer. Through this layer functionalities to play key-frame animations are supplied. These make use of the direct control primitives provided by the geometry layer. The highest level of abstraction is in the behavior level. This layer can be used to create animated synchronized speech and idle motions.

This chapter describes all layers in ascending abstraction. In the geometry layer we explain in detail the chosen techniques and the parameter instantiations used for each of these.

The animation layer section describes how parameter values are interpolated in key-frame animations. In addition, the method chosen to do multiple animation playback is depicted and justified. Finally, the coarticulation model for animated synchronized speech and the use of perlin noise is shown in the behavior layer section.

4.1 Geometry Layer

This section explains the available techniques to achieve a given effect described in a parameter. For each of these we depict what parameter instantiations are created.

4.1.1 Model-specific transformations

In Chapter 2 several non-rigid deformation techniques were analyzed. These aimed at reproducing the visual effects that the contraction of facial muscles performs on facial tissue. However, these techniques did not take into account motions such as eye movement, head tilting and jaw opening. In the developed system these motions are tackled through the use of two techniques: skeleton manipulation and rotations. These are closely related to the way that a synthetic character is represented.

Broadly speaking, a character model in the developed system is composed of a skeleton and several meshes. The skeleton has 54 bones (Figure 4.2) that are connected to various mesh elements. As seen in Figure 4.3, eyes, higher and lower teeth, tongue and body skin are represented as different objects in order to facilitate the attribution of the aimed transformations.

The skeleton manipulation method gives rise to the control for head yaw, pitch and roll and jaw yaw and pitch. Skeleton control is part of the synthetic character body gestures handling which details can be found in [20]. Eye and tongue movement is achieved through a much simpler transformation. A mere rotation applied to these mesh objects suffices to accomplish the desired effect.

Parameter Instantiation

The mentioned skeleton manipulations and rotation transformations are closely related to how a synthetic character is modeled in our system. Given this, parameter instantiations are available for each of the referred transformations of yaw and pitch for the head, jaw, eye and tongue and roll for the head.

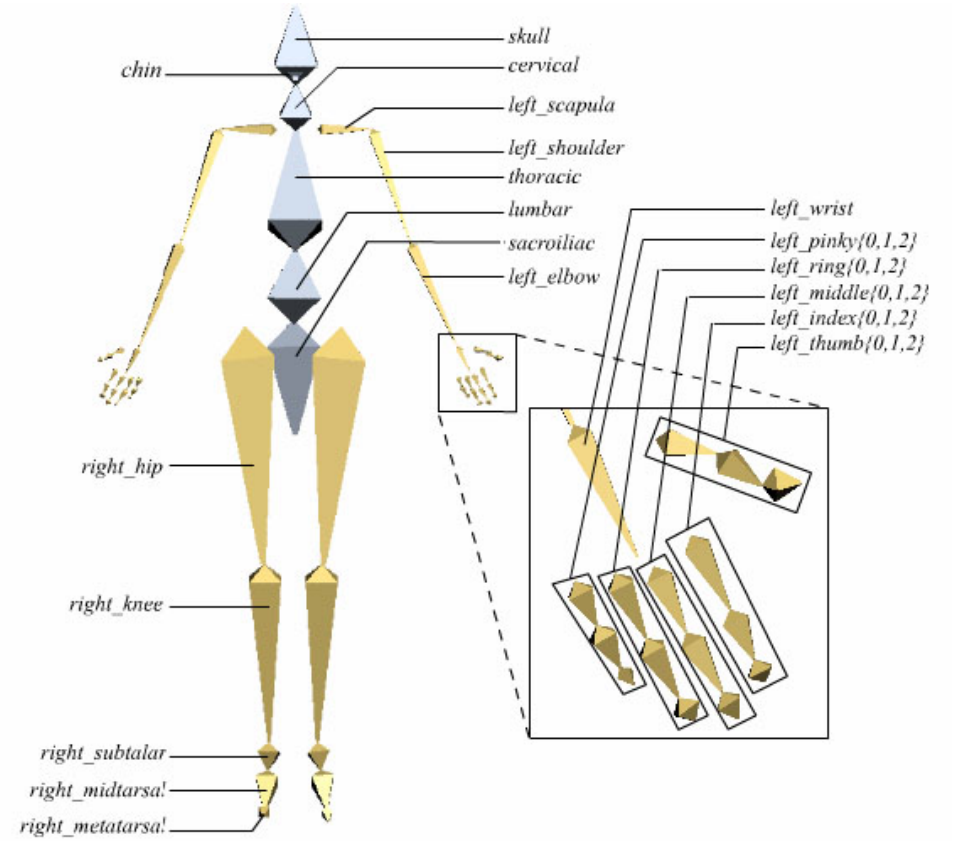

Figure 4.2: Character Skeleton

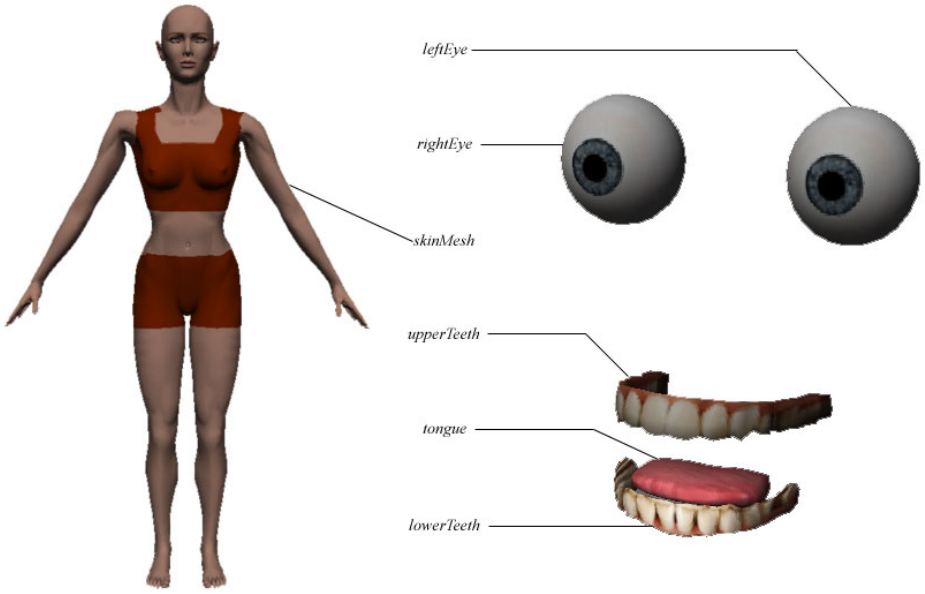

Figure 4.3: Character Mesh Objects

4.1.2 Vector Muscles

In the Related Work section a brief description of Waters' Vector Muscle model was presented. Here we will delineate Waters' method details, the modifications that were applied to it and its integration with the system parameterization logic.

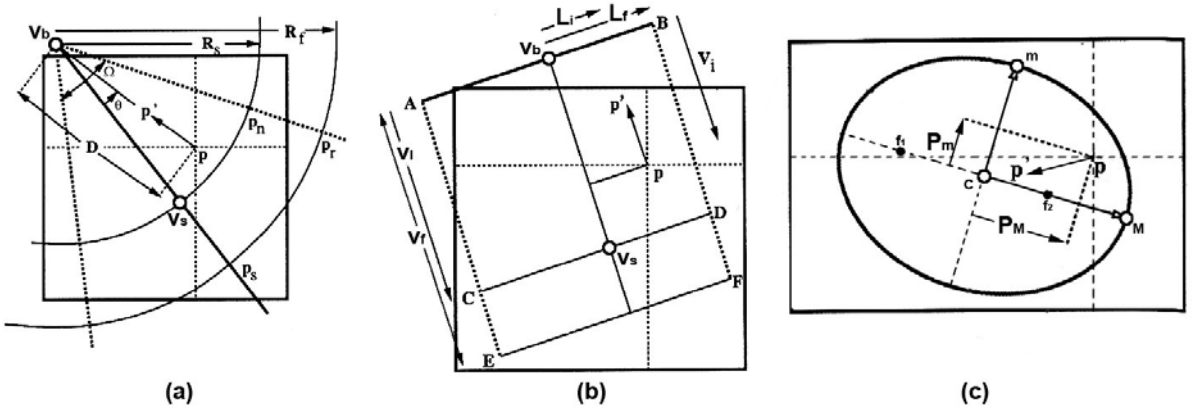

Figure 4.4: 2D Representation of Waters Muscle Models: (a) Linear Muscle (b) Sheet Muscle (c) Sphincter Muscle

As mentioned before, Waters defines three types of muscles: linear, sheet and sphincter. Linear muscles are defined by a bone insertion point, a skin insertion point, an angle of influence and a fall off distance. The bone insertion point is a static point that represents the connection end of the muscle to the bone. The skin insertion point is where maximum displacement occurs due to muscle contraction and it represents the link of the muscle with the skin. The angle of influence and fall off distance determine the region of skin that is influenced by muscle contraction. Figure 4.4(a) depicts a linear muscle that is defined by bone insertion point V_b, skin insertion point V_s, angle of influence Ω and fall off distance R_f.

The question that arises is: "How is the position of a given vertex P, within the influence region, affected by the muscle contraction force f?". Waters uses the following equation to determine the answer:

$$P' = P + f \times K \times a(\theta) \times r(D) \times \frac{\overrightarrow{PV_b}}{\left\|\overrightarrow{PV_b}\right\|}$$

where θ is $\angle V_s V_b P$; D is $\left\|\overrightarrow{PV_b}\right\|$; K is a fixed constant that represents the elasticity of the skin; $a(\theta)$ is the force dissipation angular factor; and $r(D)$ is the dissipation radial factor. The

angular dissipation factor is computed through:

$$a(\theta) = \cos\left(\frac{\theta}{\frac{\Omega}{2}} \times \frac{\pi}{2}\right)$$

and the radial displacement factor through:

$$r(D) = \begin{cases} \cos\left((1 - \frac{D}{R_s}) \times \frac{\pi}{2}\right) & , D < R_s \\ \cos\left(\frac{D - R_s}{R_f - R_s} \times \frac{\pi}{2}\right) & , R_s \leq D < R_f \end{cases}$$

Sheet muscles are similar to linear muscles. Here, skin and bone insertion points also exist, but the simulated muscle fibers contract parallel to each other toward a plane instead of a point. The region influenced by a sheet muscle can be determined by its insertion points and by two distances of influence. This gives it a cylindrical shape in three dimensions and a rectangular one in two dimensions (Figure 4.4(b)). In order to determine the displacement of a given vertex P within a region of influence defined by bone insertion point V_b, skin insertion point V_s, influence distance L_f, fall off distance V_f, due to a muscle contraction force f one uses the equation:

$$P' = P + f \times K \times d(V_i) \times r(L_i) \times \frac{\overrightarrow{V_s V_b}}{\left\|\overrightarrow{V_s V_b}\right\|}$$

where K is a fixed constant that represents the elasticity of the skin; L_i is the distance of vertex P to the center axis of the cylinder region of the muscle; V_i is the distance of vertex P to the perpendicular plane to $\overrightarrow{V_s V_b}$ that passes in V_b; and $d(V_i)$ and $r(L_i)$ are force dissipation factors that can be computed with:

$$d(V_i) = \begin{cases} \cos\left((1 - \frac{V_i}{V_l}) \times \frac{\pi}{2}\right) & , V_i < V_l \\ \cos\left(\frac{V_i - V_l}{V_f - V_l} \times \frac{\pi}{2}\right) & , V_l \leq V_i < V_f \end{cases}$$

$$r(L_i) = \cos\left(\frac{L_i}{L_f} \times \frac{\pi}{2}\right)$$

Sphincter muscles act like the tightening of a string bag. The skin is squeezed toward a point in the center of the contraction. The sphincter muscle model has an ellipsoid appearance (Figure 4.4(c)) and can be defined by the center point C, the end point of the semimajor axis M and the end point of the semiminor axis m. To compute the displacement a point P suffers due to muscle contraction force f the following equation is used:

$$P' = P + f \times K \times \left(1 - \frac{\sqrt{l_M^2 \times p_m^2 + l_m^2 \times p_M^2}}{l_M \times l_m}\right)$$

where K is a fixed constant that represents the elasticity of the skin; l_M is the length of the semimajor axis, $l_M = \left\|\overrightarrow{CM}\right\|$; l_m is the length of the semiminor axis, $l_m = \left\|\overrightarrow{Cm}\right\|$; p_M is the distance of point p to the minor axis, $p_M = \left\|Proj_{\overrightarrow{CM}}\overrightarrow{CP}\right\|$; and p_m is the distance of point p to the major axis, $p_m = \left\|Proj_{\overrightarrow{Cm}}\overrightarrow{CP}\right\|$.

Parameter Instantiation

The integration of Waters' technique with parameter instantiation is straightforward. For each muscle type there is a parameter instantiation that can be used. If these parameter instantiations represented exactly Water's model then the associated parameter value would correspond to the force of muscle contraction. However, a small change has been performed regarding the use of a force-based function.

As we have stated before, force-based functions are not intuitive in the sense that the user does not see a direct relation between the force value and the deformation. It is hard to know how much deformation will a given value of the force cause. In linear and sheet muscles this can be overcome by a simple change to the base formulas of deformation. It is known that the point that suffers the most deformation is the skin insertion point. In fact, in this point the dissipation factors are equal to one and thence the force is at its maximum value. We can then imagine a deformation method based on the movement of this point. By using a geometrical reference of how much the skin insertion point moves we improve the understanding of how the contraction behaves for a given value. In order to achieve this, all that is necessary is to substitute the terms of f and K by a given value of displacement v. Furthermore, we can make use of the notion of Unit measurement values to make the displacement portable between face models. To do this, all that is required is to use an extra parameter U in the equations. The generic deformation for the linear and sheet muscle would then be described by the following equation:

$$P' = P + v \times U \times d \times \vec{c}$$

where v is the value for the parameter associated with the vector muscle; U is the Unit measurement value for the considered model; d is the function that determines the dissipation factors mentioned above; and $\vec{c}$ is the normalized directional vector in which the contraction occurs for the given point.

It may seem as if all that we are doing is changing the name of two variables, namely f to v and K to U. Although mathematically this is all that it adds up to be, semantically the

implications are, as stated above, far deeper.

Unfortunately in sphincter muscles the same reasoning does not apply. Therefore, in this type of muscle parameter instantiation, the parameter value is perceived as a force.

A second modification related to the influence region shape was made. A face is a complex structure with its own shape quirks. Thus, using rigid shapes such as spherical cones, circular cylinder and ellipsoids as influence regions might not always fit the requirements. For this reason we allow users to remove mesh vertexes from the influence region of the muscle action. In spite of being a very simple modification this ability is of great value to the increase of the technique flexibility.

4.1.3 Radial Basis Function

As mentioned before, Kshirsagar et al. method [17] is based on the movement of control points that are set on the mesh surface. The fact that the algorithm only requires a mesh and several control points as its input is part of its advantages. However, in order to achieve the desired outcome it is necessary for the control points to be placed in reasonable locations. Control points should not only mark locations that are going to drive facial motion but also key positions that should stand still. This aspect in conjunction with a good mesh topology is essential for this method to work as expected.

Nonetheless, a face is composed of more than just control points. All the other vertexes positions must be influenced by the movement of these control points. To determine which and how nearby vertexes are affected the algorithm takes into account several properties such as, control point spatial distribution, mesh density and vertexes distance to control points. By using this information, influence regions are created with vertexes being assigned, with an associated weight, to control points. This process is the first step of the algorithm and is called *Initialization Step*. It performs heavy computations but is executed only once at a preprocessing stage.

After the influence regions and vertexes weights are computed, one has all the information required to apply the deformation of the model due to control points translation. This calculation constitutes the second step of the algorithm and is known as the *Deformation Step*. This second step requires much lesser computational power being sufficiently efficient to be executed once per rendered frame.

It is worthy of notice that, although we opted for not supporting conformation parame-

ters, this deformation technique could be used for that effect. In that case, the conformation parameters would be defined by key points on the face that could be moved to deform its neutral shape. However, as previously mentioned, the freedom of such a method is limited in the variety of faces it can represent.

4.1.3.1 Initialization Step

The first phase of this step consists in creating an approximation of a Voronoi diagram [36] where control points are the generation points. In a Voronoi diagram all points inside a cell are closer to its generation point than any other. Thus, different measurements of distance will yield different diagram topologies.

If one used the Euclidean distance the mesh shape would be vastly ignored and the space partition would consist simply on the partition of the three dimensional space. A more interesting approach is considering the partition space as the space defined by the mesh. Thence, distance is defined by the length of the smallest path that can be found between two points when crossing the mesh surface. By doing so, one takes into account desired mesh properties such as discontinuities caused by the eye and mouth holes. Kshirsagar et al. use an approximation to this surface distance. Instead of the smallest path that can be found on the surface they use the smallest path that can be found when traversing the mesh through its polygons edges. Figure 4.5 illustrates a simple two triangle mesh with the above mentioned distance measurements. One triangle is formed by the connecting the vertexes V_1, V_2 and V_3 and the other by connecting V_2, V_3 and V_4.

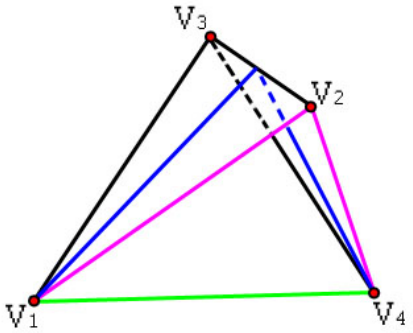

Figure 4.5: Distance measurements examples: (green) Euclidean distance (blue) Surface distance (magenta) Traversal distance

The process of creating the Voronoi diagram is an iterative one. The mesh is traversed starting from each control point, advancing to the vertex with the smallest distance to the considered control point, one vertex at a time (Figure 2.15). Hence, each control point cell grows until it reaches another control point cell. Control points that have cells with a common boundary are called neighbor control points.

After the first phase we have the mesh divided into Voronoi cells. Each cell contains the vertexes that are affected by the movement of the associated control point. However, at this moment each vertex is only influenced by a unique control point. If we were to animate the mesh with this configuration, hard edges would be noticeable between cells due to the exclusive influence. In order to achieve smoother and more natural deformation a vertex should not only be influenced by its cell control point by also by the neighbor control points. The influence of control points is computed in the second phase of the *Initialization Step*.

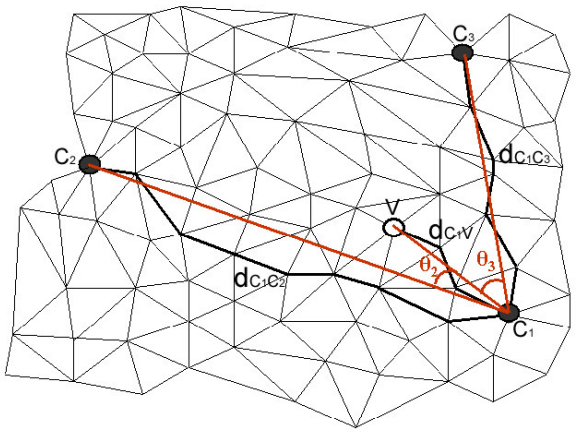

Figure 4.6: Generic surface mesh with 3 control points.

The second phase is better explained through a simple example. Consider Figure 4.6 depicting a generic mesh surface with three neighboring control points C_1, C_2 and C_3. To compute the weight associated with vertex V, that belongs to the Voronoi cell of C_1, one must consider the distances between control points, d_{C_1,C_2} and d_{C_1,C_3}, and the angles θ_2 ($\angle C_2C_1V$) and θ_3 ($\angle C_3C_1V$).

However, not all neighbor control points of C_1 are considered to influence point V. In order to influence the weight of V, neighbor control points must verify the following restriction: $\angle C_iC_1V < \frac{\pi}{2}$. In other words, the space is divided in two by a plane defined by $\overrightarrow{C_1V}.\overrightarrow{C_1P} = 0$ where P is a generic point in space. For a neighbor control point C_i of C_1 to be considered it

must verify $\overrightarrow{C_1V}.\overrightarrow{C_1C_i} > 0$.

The influence of neighbor control points is then expressed in the following weighted sum:

$$d = \frac{\sum_{i=1}^{N} d_{C_1,C_i} \times \cos\theta_i}{\sum_{i=1}^{N} \cos\theta_i}$$

where C_i are the neighbor control points of C_1 that satisfy the above mentioned restriction.

Finally, the weight associated with vertex V for deformation due to the displacement of a given control point C_j is calculated as:

$$W_{C_j,V} = \sin\left(\frac{\pi}{2}\left(1 - \frac{d_{C_j,V}}{d}\right)\right)$$

Notice that this weight is calculated not solely for the closest control point (C_1) of vertex V. The displacement of neighboring control points of C_1 will also make V to move (i.e. be assigned a weight in vertex V) if they satisfy $d_{C_i,V} < d$. It is this assigning to multiple control points in the vicinity that creates smooth transitions between control point influence regions. Thus, C_j constitutes the set of control points composed by the nearest control point to V and all its neighbor control points that verify the mentioned restrictions.

4.1.3.2 Deformation Step

At this point, for each vertex of the face mesh, we have a list of control points that influence its movement and an associated weight. Given the displacement D_j of these control points one must compute the caused displacement D_V on vertex V. This is achieved through the following equation:

$$D_V = \frac{\sum_{j=1}^{N} \frac{W_{C_j,V} \times D_j}{d^2_{C_j,V}}}{\sum_{j=1}^{N} \frac{W_{C_j,V}}{d^2_{C_j,V}}}$$

Parameter Instantiation

The parameter coupling with the RBF algorithm is performed by two parameter instantiations named *control point path* and *control point unidirectional*.

The *control point path* parameter instantiation is defined by a control point and a piecewise linear path that the associated control point should follow when the parameter value varies. The user specifies the locations that the control point should be at when a specified value of the parameter is reached. An example can be seen in Figure 4.7. The use of Unit measurements is not suitable for this parameter instantiation. This is due to the fact that the instantiation

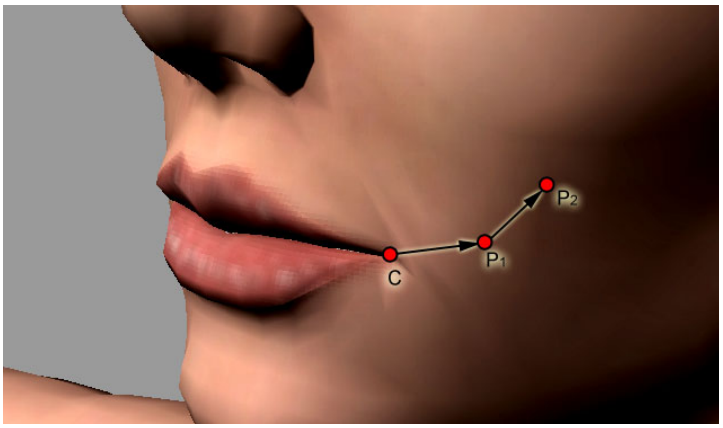

Figure 4.7: Control point path example. C represents the control point in a neutral position. P_1 and P_2 define where the control point C should pass by when a given value of the parameter is reached.

itself determines the relation between a parameter value and control point location. The position p' of a control point associated with a *control point path parameter* instantiation is then determined by the following function:

$$p' = StartPos(v,p) + \frac{v - StartVal(v,p)}{EndVal(v,p) - StartVal(v,p)} \times (EndPos(v,p) - StartPos(v,p))$$

where v is the parameter value; p is the parameter path; $StartPos(v,p)$ and $EndPos(v,p)$ are functions that return the start and end position respectively, for the associated segment of the path p that contains the value v; $StartValue(v,p)$ and $EndValue(v,p)$ are functions that return the starting and ending value respectively, for the associated segment of the path p that contains the value v.

The *control point unidirectional* parameter instantiation could almost be seen as a particular case of the above parameter instantiation where the path is defined only by two points. However, while the path instantiation defines exact locations for given parameter values, the unidirectional instantiation only specifies a given direction. This difference allows for the use of a parameter Unit measurement bringing with it its portability advantages. The position p' of a control point associated with a *control point unidirectional* parameter instantiation is determined by the following function of the parameter value v, normalized direction vector $\vec{d}$ and Unit U:

$$p' = NeutralPos + v \times U \times \vec{d}$$

where $NeutralPos$ is the position of the control point when the value of the associated param-

eter is zero.

4.1.4 Weighted Blending

This approach is based on the key-framing technique described in Section 2.1.2.2. It aims at taking advantage of the high quality expressions that can be hand crafted by animators.

In the key-frame interpolation method described in the Related Work section a key-frame is composed of a set of absolute positions for every point of the model meshes. So, animating the model between two key-frames consists in linearly interpolating these absolute positions. No matter what the initial positions of the starting frame are, the face will always deform into the posture defined by the ending key-frame. We look at this morphing process focusing in the dynamic change instead of the absolute positions.

Consider the notion of a neutral face mesh such as the one defined in Mpeg-4. All other face postures can be seen as a deformation of this neutral posture. We believe that it is possible to capture the essence of the deformed posture through the following simple subtraction:

$$\left[\vec{d_1}\ \vec{d_2}\ \cdots\ \vec{d_N}\right] - [\vec{n_1}\ \vec{n_2}\ \cdots\ \vec{n_N}] = [\vec{o_1}\ \vec{o_2}\ \cdots \vec{o_N}]$$

where N is the number of vertexes of the face mesh; $\vec{d_i}$ are the vertexes positions in the deformed posture; $\vec{n_i}$ are the vertexes positions in the neutral posture; and $\vec{o_i}$ are the resulting offset vectors of the subtraction.

Thus, the transformation of the neutral face into the intented posture is stored in an array of vertex offsets. Having a representation of the dynamic changes facilitates applying the desired deformation without losing other deformations present in the face. A simple example would be considering a face that displays surprise due to the action of several parameters. We could then combine the display of fear by adding the array of offsets that correspond to that display. However, the ability to mix several displays through this method is not without its limitations. Blending displays that contain severe deformations, or blending a large amount of displays, may lead to undesired effects due to accumulation of local deformations without restrictions.

Parameter Instantiation

A *Weighted Blending* parameter instantiation is essentially composed of an array of vertexes offsets . When the value of a parameter that uses this type of instantiation varies the induced

deformation can be computed through the following linear interpolation equation:

$$[\vec{p_1}\ \vec{p_2}\ \cdots\ \vec{p_N}] = [\vec{c_1}\ \vec{c_2}\ \cdots\ \vec{c_N}] + \frac{v - MinVal}{MaxVal - MinVal} \times [\vec{o_1}\ \vec{o_2}\ \cdots\ \vec{o_N}]$$

where N is the number of vertexes of the face mesh; $\vec{p_i}$ are the vertexes positions after the desired deformation; $\vec{c_i}$ are the current vertexes positions; v, $MinVal$ and $MaxVal$ are, respectively, the current, minimum and maximum value of the parameter; and $\vec{o_i}$ are the offset vectors associated with the instantiation.

4.1.5 Group Parameter Instantiation

The above mentioned parameter instantiations are all atomic in the sense that they can not be decomposed. Model-specific deformations, Vector Muscles and RBF are used as low-level parameter instantiations. Each of these affect only part of the face causing a local deformation. Weighted Blending is primarily intended to be used as a high-level parameter instantiation where the entire face is deformed into an expression. Nevertheless, it is conceivable that one could model local deformations with this technique. Besides Weighted Blending, the developed system provides another method to instantiate high-level parameters through the use of group instantiations.

A group instantiation is, as the name suggests, a parameter instantiation that influences a group of other parameters. An example of a parameter that could be instantiated with a group instantiation would be a Joy expression. This instantiation would manipulate low-level parameters such as eye squinting, zygomatic muscle contraction and lip spread.

Due to the multiple links between parameters and instantiations we shall, in favor of a better understanding, refer to a parameter associated with the group instantiation as *group parameter* and *element parameters* to the parameters it influences. When the value of a group parameter changes so do all values of its element parameters. It is important to stress that element parameter values ***vary***, in opposed to ***are set*** to an absolute value, with the increase or decrease of the value of the group parameter. This can be seen as a similar approach to the blended weights technique where positions suffer a displacement instead of being moved to a specific location when parameter values changes.

The value variation each element parameter suffers can be directly or inversely proportional to the parameter group value variation. Furthermore, the amount of influence is restrained to minimum and maximum value. These limits are specified per element parameter and are

defined as a percentage of the minimum and maximum value the element parameter can be independently set to.

Since a group instantiation only identifies the parameters it influences through their names and the limits of variation are set through percentages it supplies great flexibility. Group instantiations are not only independent of their element parameters instantiations but also of the concrete values of these. This property permits easy, and in some cases direct, reuse of group instantiations across different face models.

Similarly to what happens in weighted blending instantiation, in group instantiations the use of a model bond Unit does not apply.

A given parameter can be influenced by several group parameters. This can be seen as sharing resources and some implications arise from it. This issue also appears in the support for multiple animations. In section 4.2.2 we describe this point in detail.

To compute the new value V'_E of an element parameter P_E due to the value variation from V_{G_i} to V_{G_f} of the related group parameter P_G the following equation is used:

$$V'_E = \begin{cases} V_E + \frac{V_{G_f} - V_{G_i}}{V_{G_{Max}}} \times V_{E_{MaxVar}} & , V_{G_f} \wedge V_{G_i} \in [0, V_{G_{Max}}] \\ V_E - \frac{V_{G_f} - V_{G_i}}{V_{G_{Min}}} \times V_{E_{MinVar}} & , V_{G_f} \wedge V_{G_i} \in [V_{G_{Min}}, 0[\end{cases}$$

where $V_{E_{MinVar}}$, $V_{E_{MaxVar}}$ are respectively the minimum and maximum variation that the value of P_E can sustain through the action of P_G. If P_E value varies inversely then $V_{E_{MinVar}}$ should swapped by $V_{E_{MaxVar}}$ and vice versa.

4.2 Animation Layer

As previously mentioned animation is supported by this layer through the use of key-frame based animations. Each key-frame specifies the parameter values for a given instant in time. In this section we start by explaining how these values are interpolated between key-frames and follow with how multiple animation support is provided by the system.

4.2.1 Parameter Value Interpolation

As stated before there is a vast number of means to interpolate values. Figure 4.8 depicts 3 ways of interpolating between the parameter values of an animation with 5 key-frames. If we used linear interpolation we would discard any acceleration phenomenon. The facial

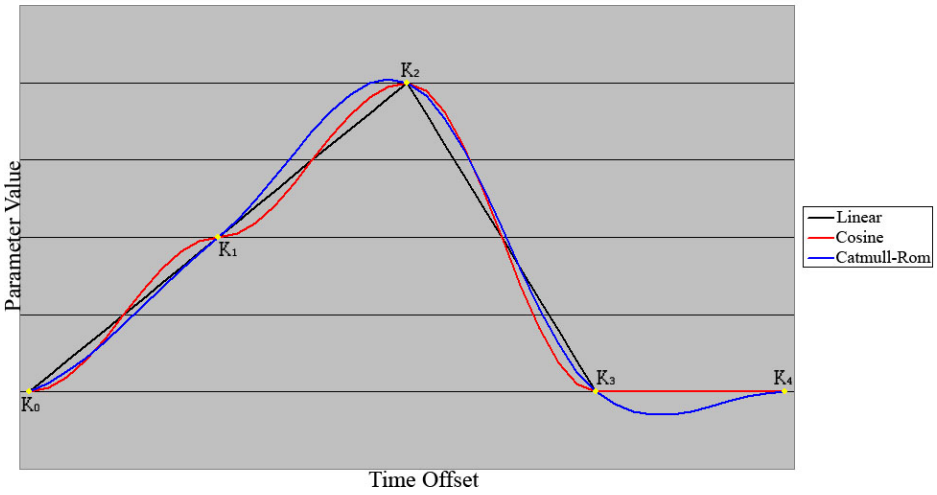

Figure 4.8: Linear, Cosine and Catmull-Rom interpolation of the parameter value on a 5 key-frame animation.

movements would have constant speed and change in direction would be instantaneous as can be seen in key-frame K_2. This would result in a very unnatural movement of the face. Given this, we should consider non-linear interpolators. Cosine and Sine functions are commonly used non-linear interpolators due to their simplicity and computational efficiency. In fact, we have used both of them when calculating displacement decay in Vector Muscles and RBF deformation techniques. Nonetheless, using these for parameter value interpolation yields some unwanted results that are not present when they are used in the deformation process. With cosine interpolators the velocity (parameter value variation over time) at key-frame points is null. While this is a desired behavior when the parameter value function changes its growth direction (increase to decrease or vice versa, e.g. K_2) it is not supposed to happen when the growth direction is maintained (e.g. K_1). A Catmull-Rom Cubic Spline can be used to surpass this problem. Unfortunately, this method also has its drawbacks. As can be seen in K_2, and more distinctly in K_3, there is an overshoot of the interpolated parameter value. This overshoot does not happen with the cosine interpolation method. What we would like to have is a function that behaves similarly to a cosine interpolator for some cases and like a Catmull-Rom Spline in others. To better explain our approach we will take a deeper look at the definition of splines, particularly cardinal splines.

A Spline is a piecewise polynomial function. This is a very generic definition and the term Spline in computer graphics is commonly used regarding to a specific sub-set of splines

that are composed of second or third degree polynomials that describe a parametric curve. We are interested in cubic splines meaning that the polynomials that compose them are of the third degree. This will achieve a smooth motion along the function due to its continuity. Hermite Splines are a family of cubic splines where the polynomials are in Hermite form. Hermite Splines are defined by control points (or knots) where the curve should pass and by the tangent values of the function in those points. Figure 4.9 depicts a Hermite parametric curve.

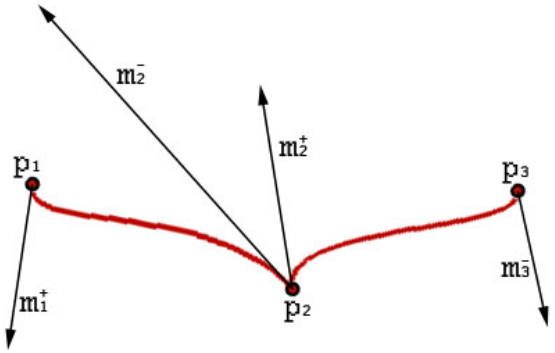

Figure 4.9: Hermite Spline Curve

For any given segment of the curve, between two control points p_{i-1} and p_i, the curve is defined by:

$$f(u) = (2u^3 - 3u^2 + 1)p_{i-1} + (u^3 - 2u^2 + u)m_{i-1}^{+} + (-2u^3 + 3u^2)p_i + (u^3 - u^2)m_i^{-} \quad , u \in [0, 1]$$

where m_{i-1}^{+} and m_i^{-} are the tangents at the right of p_{i-1} and at the left of p_i respectively; $f(0) = p_{i-1}$; and $f(1) = p_i$.

These tangents can be part of the user given parameters, like the control points, or be calculated through the use of other parameters. This later method gives way to the definition of several families of Hermite spline functions of which Cardinal Splines are part.

Cardinal Splines are Hermite Splines which tangents are defined in the following manner:

$$m_i = \tau(p_{i+1} - p_{i-1})$$

where τ is a *Tension* factor that affects how sharply the parametric curve bends at the control points. The effect of the tension factor can be seen in Figure 4.10. Catmull-Rom Splines are Cardinal Splines with a tension factor equal to 0.5. Unlike more generic Hermite Splines,

Cardinal Splines have necessarily equal tangents on the left and right side of control points giving them C^1 continuity.

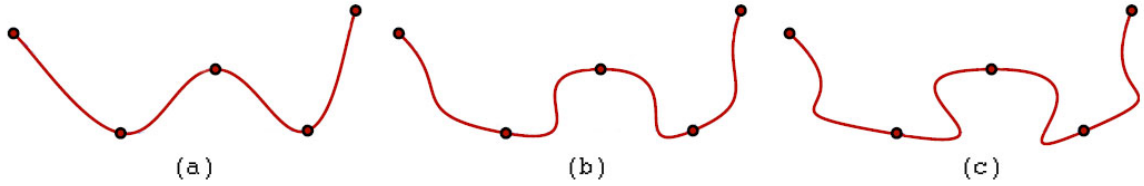

Figure 4.10: Effect of varying τ in a cardinal spline. (a) $\tau = 0.2$ (b) $\tau = 0.5$ (c) $\tau = 0.75$

It is important to note that there is a significant difference between the curves shown in Figure 4.8 and Figures 4.9 and 4.10. While in the latter two figures only the codomain of the curve function is presented, in Figure 4.8 both the domain and codomain are shown. In other words, we want to interpolate unidimensional values and not two-dimensional ones. Given this, the question that arises is "How does the interpolation function appearance changes with τ?". Figure 4.11 depicts this influence for values of τ varying between 0 and 1. From this figure one

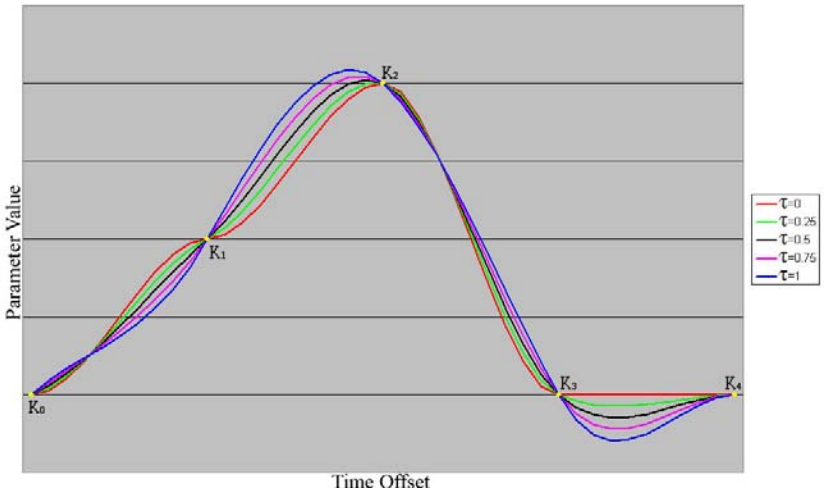

Figure 4.11: Cardinal spline interpolations with different tension factors.

can verify that τ scales $\frac{dv}{dt}$ at control points. With $\tau = 0$ the interpolation function behaves similarly to a cosine interpolator, with $\tau = 0.5$ like a Catmull-Rom Spline interpolator and with $\tau = 1$ like a simple Cubic interpolator.

Taking a step back we remind that our goal was to find an interpolation function that varied its behavior between a Cosine and a Catmull-Rom interpolator. As we have seen this is possible by manipulating the tension factor of Cardinal Splines. By observing Figure 4.8

we can deduct that the tension at point K_i can be calculated through the relation between the slopes formed between K_{i-1}, K_i and K_i, K_{i+1}. The idea is that when one of the slopes tends to 0 so should the derivative at K_i. This reduces the overshoot of the aimed value at the keyframe. When the slopes are equal the interpolant should have tension 0.5. This prevents the unwished phenomena that was obtained through cosine interpolation while attaining a smooth movement. Starting and ending points need special treatment that we will discuss later on. Given this, the interpolation function that is used in the system is described in matrix form by the following Cardinal Spline formula:

$$f(u) = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\tau_{i-1} & 0 & \tau_{i-1} & 0 \\ 2\tau_{i-1} & \tau_i - 3 & 3 - 2\tau_{i-1} & -\tau_i \\ -\tau_{i-1} & 2 - \tau_i & \tau_{i-1} - 2 & \tau_i \end{bmatrix} \begin{bmatrix} v_{i-2} \\ v_{i-1} \\ v_i \\ v_{i+1} \end{bmatrix}$$

where τ_{i-1} and τ_i are the tension factors at points K_{i-1} and K_i respectively; v_{i-2}, v_{i-1}, v_i and v_{i+1} are the parameter values of points K_{i-2}, K_{i-1}, K_i and K_{i+1}, respectively; and u is normalized parameter when the function varies between K_{i-1} and K_i and is given by:

$$u = \frac{t - t_{i-1}}{t_i - t_{i-1}} \quad , t_{i-1} \leq t \leq t_i$$

where t is the instant in time for which we are calculating the interpolated value; and t_{i-1} and t_i are the time instants of key frame points K_{i-1} and K_i, respectively.

To calculate the tension factor at a given control point K_i the following equation is used:

$$\tau_i = \begin{cases} 0.5 \left(\frac{min(S_{i^-}, S_{i^+})}{max(S_{i^-}, S_{i^+})} \right)^2 & , max(S_{i^-}, S_{i^+}) \neq 0 \\ 0 & , max(S_{i^-}, S_{i^+}) = 0 \end{cases}$$

where S_{i^-} is the absolute value of the slope left of point K_i and S_{i^+} is the absolute value of the slope right of point K_i. These are defined by:

$$S_{i^-} = \left\| \frac{v_i - v_{i-1}}{t_i - t_{i-1}} \right\| \quad S_{i^+} = \left\| \frac{v_{i+1} - v_i}{t_{i+1} - t_i} \right\|$$

In edge key points we use 0.5 as the missing slope value. This value was determined empirically by trial and error and was the value that displayed best overall results. In order for the function to interpolate pass by the edge points we must also supply two extra points. One at the start of the function and another at the end. We duplicate the start and end values to create this points. Again, this choice is an empirical one that showed to provide good

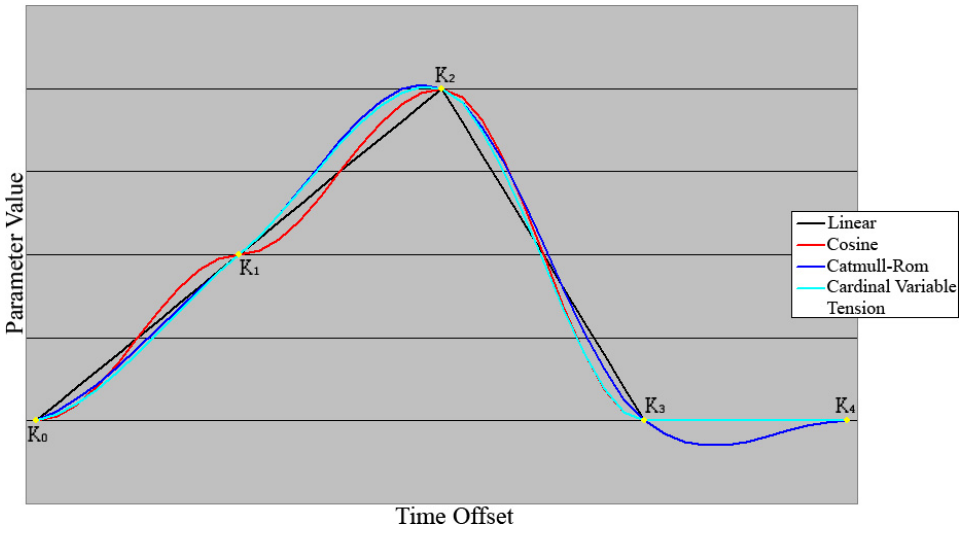

Figure 4.12: Used interpolation function with variable tension compared with linear, cosine and Catmull-Rom interpolation methods.

results. Figure 4.12 shows the final result of the cardinal interpolation function with variable tension while comparing it to the other described methods.

Summarizing, to interpolate key-frame parameter values we use a Cardinal Spline with variable tensions at different control points. Tensions are automatically inferred by the relation of the the slopes on the left and right side of the considered control point. By doing so we have an interpolation function that adapts the tension at control points in order to have a smooth "natural" accelerated motion with very little overshoot.

4.2.2 Multiple Animation Support

Playing multiple animations at the same time presents no problem if these do not have overlapping parameters. An example of this is a speech animation that is played along with a head nodding animation. However, if the same parameter is used concurrently in different animations a conflict arises. There is no ideal solution for this issue, especially without taking into account the specific meaning of each parameter. Since the developed system aspires to be as generic as possible, following the decoupling idea from Parke and Waters, and aiming at being capable of a multitude of parameterizations, it is not possible to take advantage of much *a priori* knowledge on the nature of the parameters. Nonetheless, the little information that is available on the meaning of the possible deformation techniques may be used to select a suitable approach.

Regarding deformation techniques that are aimed at modeling low-level parameters, namely Vector Muscles and RBF, we may apply two base interpretations:

- **Static**: A parameter value sets the parameter control point position. Each key frame in an animation sets exactly the way the face should look at that instant.
- **Dynamic**: A parameter value sets the displacement of the control point position. Each key frame in an animation describes the movement to achieve the intended expression.

This is a subtle difference that has no effect if we only have one animation influencing the considered parameter. On the opposite case however, it leads to different ways to merge the several parameter values. In the first case, we believe the most meaningful approach is finding the average point among all the specified ones. This is due to the absolute nature of the interpretation. On the second case, a sum of the set displacements makes more sense.

Although model specific parameters do not define point positions or displacements they define angular values. Therefore the same two interpretations can be applied to an absolute or relative angular value.

As we have mentioned, high-level parameters, such as group parameters and parameters that use weight blending deformation, are already by themselves based in the dynamic approach. In weight blending the influenced vertexes are displaced according to the defined parameter offsets that describe the quality of motion of the parameter. In addition, when a group parameter value varies, it creates increments or decrements in the several parameter elements that belong to it.

In face of these two interpretations we have opted to follow the dynamic approach. We believe that this choice is in accordance with the notion of neutral face defined in Mpeg-4. In this standard parameters are defined as changes to the neutral state of the face and thus follow the dynamic notion. Therefore when multiple animations are played simultaneously we perform a truncated sum of the concurrent parameter values set along the animations. The sum is truncated in order to assert that the parameter value bounds are maintained.

In particular cases better results could be achieved with more complex methods that are less semantic free. For example, a resource based method could be used to deactivate parameters on an animation giving its exclusive use to a higher priority animation. Synergies between parameters could also be taken into account using a weighted sum to compute the final parameter value.

4.3 Behavior Layer

This layer is the highest of the three in its level of abstraction. Its goal is to supply high-level mechanisms built upon the lower levels in order to facilitate the use of commonly used behaviors. Although the main focus of this thesis is on the Geometry and Animation layer, two simple functionalities were developed in the Behavior layer: Speech synchronization and Perlin noise.

4.3.1 Speech Synchronization

In this work, speech synchronized facial animation is achieved through a simple coarticulation model. It is not objective of this model to achieve truly realistic lip-sync but otherwise cartoon-ish animation. We believe that even a simple approach such as this one suffices a considerable number of applications where believability is more relevant than realism.

The coarticulation model takes into account two consecutive phonemes. Each existing phoneme has an associated parameter that represents the corresponding viseme. When consecutive phonemes are "spoken" the face is interpolated so that the corresponding visemes are shown. This process is depicted in Figure 4.13.

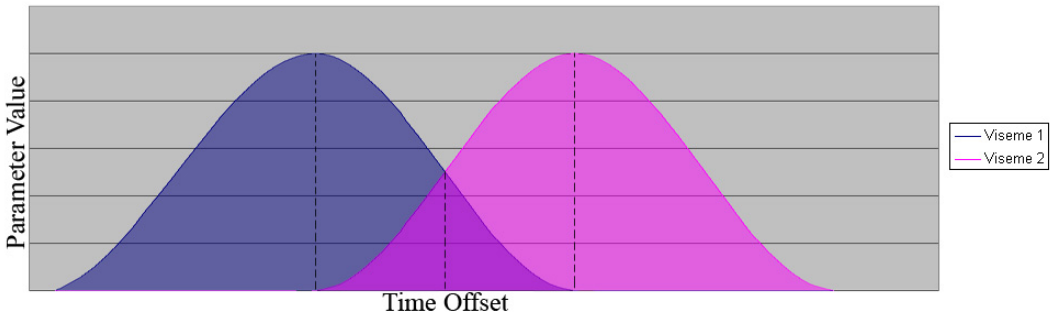

Figure 4.13: Coarticulation Model

There are two ways by which speech animation may be produced. The first is with Festival [2] text to speech engine. The user supplies the text he wishes the character to say and Festival produces synthesized wave speech and all the timing information relative to the phonemes articulation. Alternatively the wave and timing information may be supplied directly by the user. The above described coarticulation method is then applied to create and play the corresponding animations.

4.3.2 Perlin Noise

Perlin Noise [25] is a method that uses pseudo-randomness to create unpredictability in several types of applications such as animation and textures. In this work Perlin Noise is used to generate idle motions, like an eye gaze, or to modify the motion in animations in order to make them richer and more "natural".

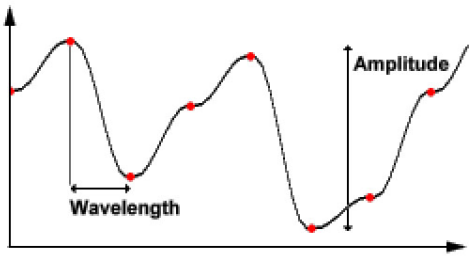

Figure 4.14: Wavelength and Amplitude of an interpolated noise function. The red dots represent the random values of the function.

A Perlin Noise function is the result of adding several interpolated noise functions. Each one of these will define the characteristics of the final result by using different values of *frequency* and *amplitude*. Amplitude is the difference between the maximum and minimum value that the function takes, frequency is $\frac{1}{wavelength}$ and wavelength is the distance between random points. These concepts are shown in Figure 4.14.

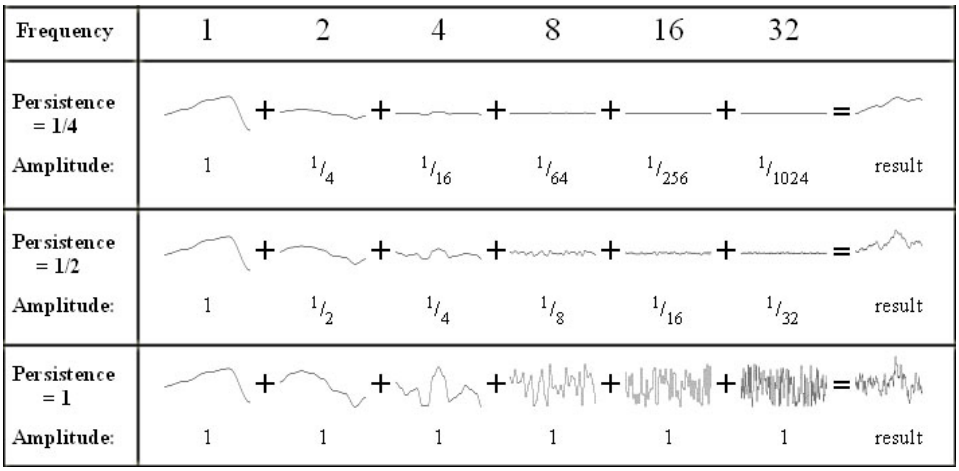

Figure 4.15: Resulting Perlin noise function using several persistence values.

These adding functions are called *Octaves* because their frequency doubles from one to another like in music octaves. The amplitude relation is set through a user specified factor

named *persistence*. This, along with the number of octaves, gives the user the ability to create a controlled pseudo-random function that models the intended behavior. Figure 4.15 shows the additive process and final result for different persistence values on a noise function with six octaves.

4.4 Summary

The system uses a three-layer architecture. The Geometry layer is responsible for maintaining all information regarding rendering, character model and facial parameterization. A character model is composed of a skeleton and several meshes. A facial parameterization is a set of "abstract" unidimensional parameters in the sense that they do not define a specific way to achieve the effect they describe. This is attained through parameter instantiations that use several techniques. The available parameter instantiations sorted by deformation technique are:

- **Skeleton Based:** Head Yaw, Head Pitch, Head Roll, Jaw Yaw and Jaw Pitch
- **Rotation:** Eye Yaw, Eye Pitch, Tongue Yaw and Tongue Pitch
- **Vector Muscle:** Linear Muscle, Sheet Muscle and Sphincter Muscle
- **RBF:** Control Point Path and Control Point Unidirectional
- **Weighted Blending:** Weighted Blending

Another parameter instantiation exists that is not associated with a particular deformation technique. The group parameter instantiation bundles parameters together under a single control parameter.

The animation layer supplies means to animate the face through key-frames. These specify, for a given instant, which parameters are active and to what extent. Given this, animating the face consists in interpolating parameter values between key-frames. An interpolation method was developed to attain smooth accelerated movement with a controlled overshoot.

The behavior layer provides speech synchronized facial animation and pseudo-random motion behaviors. Speech synchronization can be used with Festival text to speech engine or through user given wave files with phoneme timings. Perlin noise functions allow the user to

easily create “natural” idle movements and to modify the motion qualities of key-frame based animation.

Chapter 5

Application

This Chapter starts by debating the technological choices that were made to create the developed system. Next, we give a brief overview of how the programmable graphical pipeline might become a good option to create facial animation systems in the future. Finally, we depict the application that was developed to create and edit all of the notions described in the conceptual model. Furthermore, the relation of this application with the deformation and animation system is also explained.

5.1 Technology

Several aspects were taken into account when choosing the technology on which to implement the developed facial animation system.

Regarding the development language, C# was the chosen option. Managed languages such as Java and C# have long gathered support in a vast number of applications due to their ability to reduce development time. Intensive graphical applications are however still mainly produced in C++ for performance reasons. Microsoft believes this is about to change. Not long ago it has developed a managed version of its graphical API DirectX which now has evolved to a more broad goal to be a cross-platform game developing framework called XNA. The efficiency of the managed DirectX is for most cases almost identical to the unmanaged version. For these reasons and due to the synergy between C# and managed DirectX we have opted to develop the system upon managed DirectX. As should be expected there are always two sides to one coin. Although C# is a cross-platform language, its implementation on other operating systems, such as Mono [4] in Linux, is still not fully supported. DirectX support

on other systems besides Windows is also lacking. Nevertheless, ultimately the interaction between the facial animation system and the graphics API consists of two methods to *get* and *set* the mesh vertexes positions. So, if required, changing the API from DirectX to another one such as OpenGL should prove feasible.

Although C# and DirectX were chosen it is worth to mention that a GPU based approach was studied during the system development. In the following section we portray the advantages that such solution would bring and why it was not chosen.

5.2 Using the programmable pipeline

Electronic game entertainment has suffered a large growth in the past years. With it, large amounts of money were invested in the development of faster and better graphical cards. This investment provided the world of home desktop computers with extremely powerful dedicated graphical processing units (GPU). In fact, a top of the line central processing unit (CPU) such as Intel's latest quad-core Core 2 Extreme QX6700, consists of 582 million transistors while nVidia'a latest video card, the 8800 GTX, has 680 million transistors. Furthermore, during this growth, the graphical processing pipeline concept suffered a radical evolution. While initially the pipeline functionality was fixed it has been changing into an evermore flexible programmable pipeline. Nowadays the pipeline can be "customized" through the use of sets of instructions named shaders. These allow the change of graphical properties such as position, normals and color in the pipeline processing resorting to highly dedicated and efficient functions. There are three types of shaders: Vertex shaders, Pixel shaders, and Geometry shaders. As their name implies vertex shaders are applied per vertex while pixel shaders are applied per pixel. Geometry shaders do per primitive operations on vertexes grouped into primitives such as triangles and lines that are the output of vertex shaders. Unlike Vertex shaders, geometry shaders can make copies of the inputted primitives and thus create new vertexes if necessary. This last shader type is new and was not available during the development of this thesis.

Summarizing, the programmable pipeline allied with the current powerful GPUs supplies the developer with a programming platform that is focused and tailored to his or her needs. This approach was taken into consideration and Waters' vector muscles were implemented using it to test its feasibility. We found that the deformation algorithm itself could be fitted

into this paradigm easily. However, soon we realized that the pipeline was not as flexible as we hoped for. The main problem was the need for a generic system where each vertex requires having arbitrary information depending on the type of parameter instantiations that affects its displacement. Therefore, using a Vertex shader approach would limit the number of parameter instantiations per vertex. Also, due to a restriction in the number of possible instructions per pass it would be necessary more than one iteration to contemplate all deformation algorithms. Given this, and after testing its performance feasibility, we opted for the more generic and flexible CPU based approach.

Even so, we believe without a doubt that a GPU approach implementation is the logical solution to use in near future facial animation systems. The above mentioned limitations are being reduced or eliminated with each new version of the Shader model.

5.3 Face Editor

An editing application, the Face Editor, was created in order to facilitate the production and testing of all of the functionalities shown so far. Figure 5.1 depicts the flow of information when creating a model that is to be used by the developed system. The role of the face editor in this process can also be observed in this figure. An illustration of this application is shown in Figure 5.2.

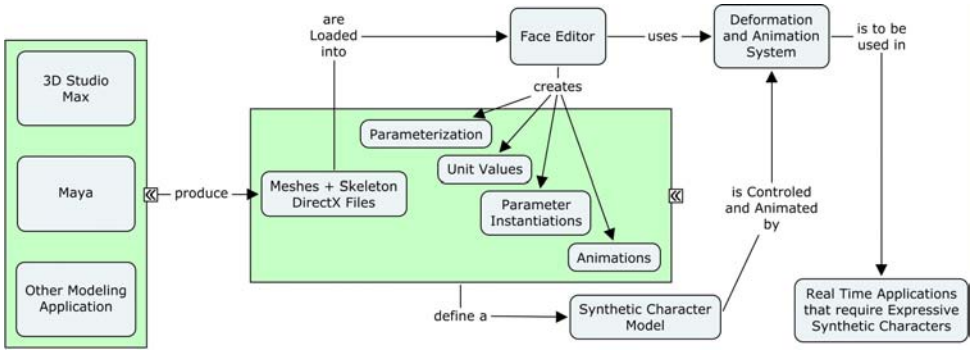

Figure 5.1: Information Flow

Two stages can be considered in order to animate a synthetic character in the developed system. The first stage is the *Setup* stage where the character Meshes, Skeleton, Parameterization, Parameter Instantiations and Units are created. The second stage is the *Deformation and Animation* stage where the information produced in the previous stage is used to animate

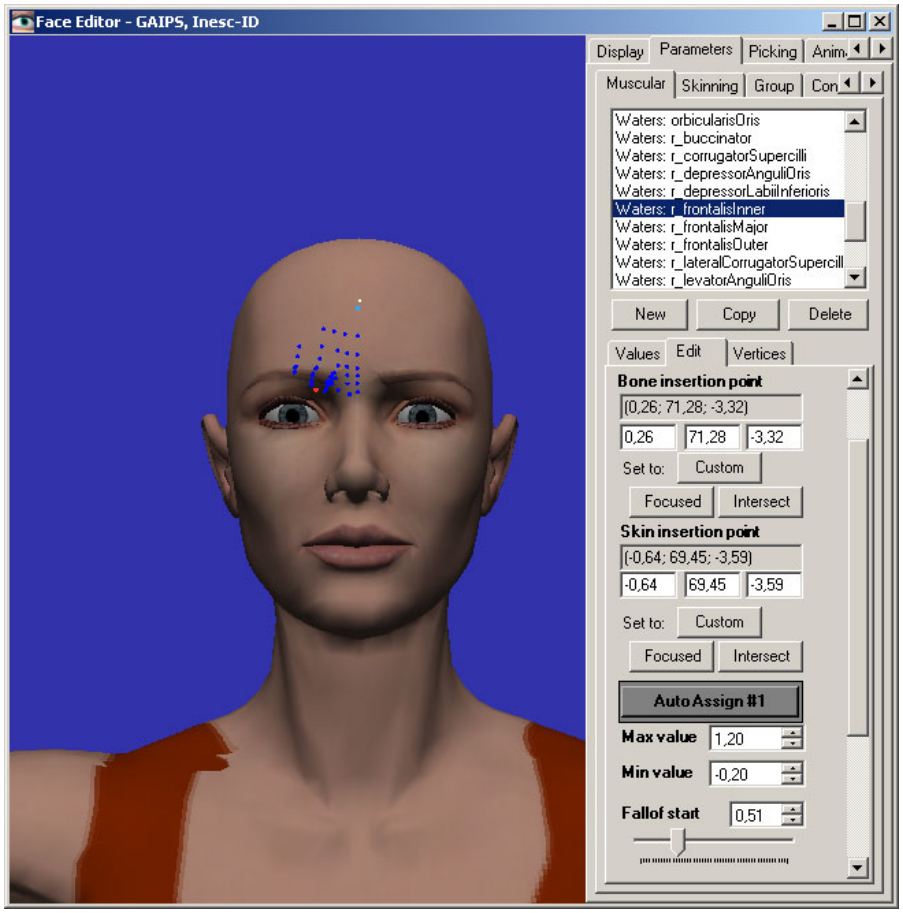

Figure 5.2: Face Editor application

the character. These stages will be explained in detail over the next two sections.

Even before completely defining all structures, it is useful to apply the available information to obtain feedback on how deformation and animation will look like. Therefore, the editor supplies an iterative paradigm of modeling allowing the user to receive an immediate feedback from the changes he produces. For example, changing a parameter value will show what deformation it causes through the associated parameter instantiation.

5.3.1 Setup

Defining a new model should follow a specific order of actions. First we specify what we want to be able to express through the facial expressions of the synthetic character. This step consists on the creation of the Parameterization. A Parameterization by itself means little for the animation system. It is composed of several parameters that are nothing more than variable unidimensional values with an associated identifier, an optional unit value used for scaling and

a textual description. This abstraction is more important for the person that will later use the parameters.

Next the model(s) meshes and skeleton should be created in a 3D modeling program. In order to use parameter instantiations associated with skeleton manipulation it is required that the skeleton follows the description given in [20].

The following step consists in defining the required unit values used in the parameterization. This is usually done by selecting two key points of the face which distance is then computed. Alternatively the user may specify a concrete value. Any point selection of the face mesh is done through point and click.

Finally the parameter instantiations are defined. Given the particularities of the model mesh and the requirements defined in each parameter specification, the user must choose which parameter instantiation models the parameter best. Depending on the type of instantiation, different information is stored. However, more than the strictly required information for animation is stored. In fact, we may say that the same information is stored twice with two different goals. One is aimed for future editing. It contains every detail that defines that specific parameter instantiation. The other is used for the actual animation. This latter form is optimized to reduce the necessary computations by pre-calculating everything that can be computed before the animation is performed. For example, let us consider a linear vector muscle instantiation. To animate this kind of parameter instantiation we are only required to know the displacement vector for each influenced vertex. Nevertheless, we also store the skin and skeleton insertion points, angle of influence, etcetera.

Although animations are not dependent on the parameters, they are usually created after having defined the above attributes. This is because it is much easier to create the animations having a visual feedback of the deformations caused by the variation of the parameter values.

All of the supra defined information is stored persistently in XML format.

5.3.2 Deformation and Animation

Parameter values change over time due to direct manipulation, animation playback or high-level behaviors. With these variations the face deforms and facial expressions arise. The process of modifying the face is achieved through an Update/Render cycle that is performed once for each rendered frame.

As the name indicates, the first action to take place is the update of the inner structures

according to the current parameter values. This update is executed in an hierarchical manner with the highest level processes taking precedence. First, the behavior layer is updated from which new animations may be created. Next, the animation layer uses the current elapsed time to update all active animations. Two particular cases require special attention when updating the animation layer.

The first is when an animation has become active in the current update and the initial key-frame is not defined for a time-offset equal to zero. This issue is solved by creating a snapshot of the current parameter values.

The second case is when the rendering thread is put on hold for some time. This can happen for example if the application window looses focus. This means that, when the update cycle is finally executed, an active animation can have several unprocessed key-frames which time-offsets are older than the current update time. Due to the incremental approach used, we are required to simulate the cumulative effect from all previous key-frames in order to achieve the intended facial display for the current update time. This problem is accentuated when several animations are active.

Once all parameter values are settled the geometry layer is updated. It is here that the actual deformation of the model mesh takes place. In order to enhance performance all deformations are made in a system memory copy of the model mesh. Only after all deformations take place, and only if changes actually were performed, is this copy written into the graphical card's memory. This is essential since reading and writing through the graphical bus is an expensive operation.

At last the model can be rendered and another pass in the cycle begins.

5.4 Summary

We used C# and DirectX as the base technologies to implement the developed system. The system interactivity with the graphical API is well delimited allowing for its easy replacement if necessary.

A GPU approach was studied for feasibility by implementing the Waters' muscle model in this paradigm. Such approach still imposes difficulties for systems that aim at being generic. At the moment structures in this type of approach still need to be extremely defined and static. However, this is a fast evolving field which we are sure to prove very good for such applications

in the near future.

An overview of the flow of information was given. In this, the Face Editor application plays a central role in aiding the creation of parameterized synthetic characters. The two stages of *setup* and *Deformation and Animation* were detailed and the main implementation problems shown.

Chapter 6

Case Study

In this chapter we illustrate a sample parameterization and how it can be used in the developed system. This parameterization contains high and low-level parameters similar to the ones found in FACS and Mpeg4 FAPs. We show how the available parameter instantiations are best fitted to each parameter depending on an array of properties. Furthermore, this process will be demonstrated for two different character models with different attributes such as high or low polygon count.

6.1 Character Models

Two models will be presented in this chapter. One of these is a cartoon-like character that represents a virtual storyteller and is depicted in Figure 6.1. This model has a low polygon count. Its head and neck are composed of 561 vertexes and it has a total of 1381 vertexes. Its mesh was created in 3D Studio Max from smaller elements, such as eyebrows and sideburns, that were aggregated together. As will be demonstrated, this design will impact the choice of what instantiation to use for certain parameters.

The other model was retrieved from Curious Lab Poser 5 program [5] and represents a "realistic" woman. Here we understand "realistic" as non-cartoonish or with typical human anthropometry. The model has a medium/high number of polygons with its head being composed of 2887 vertexes and the full body of 13966 vertexes. This model is shown in Figure 6.2.

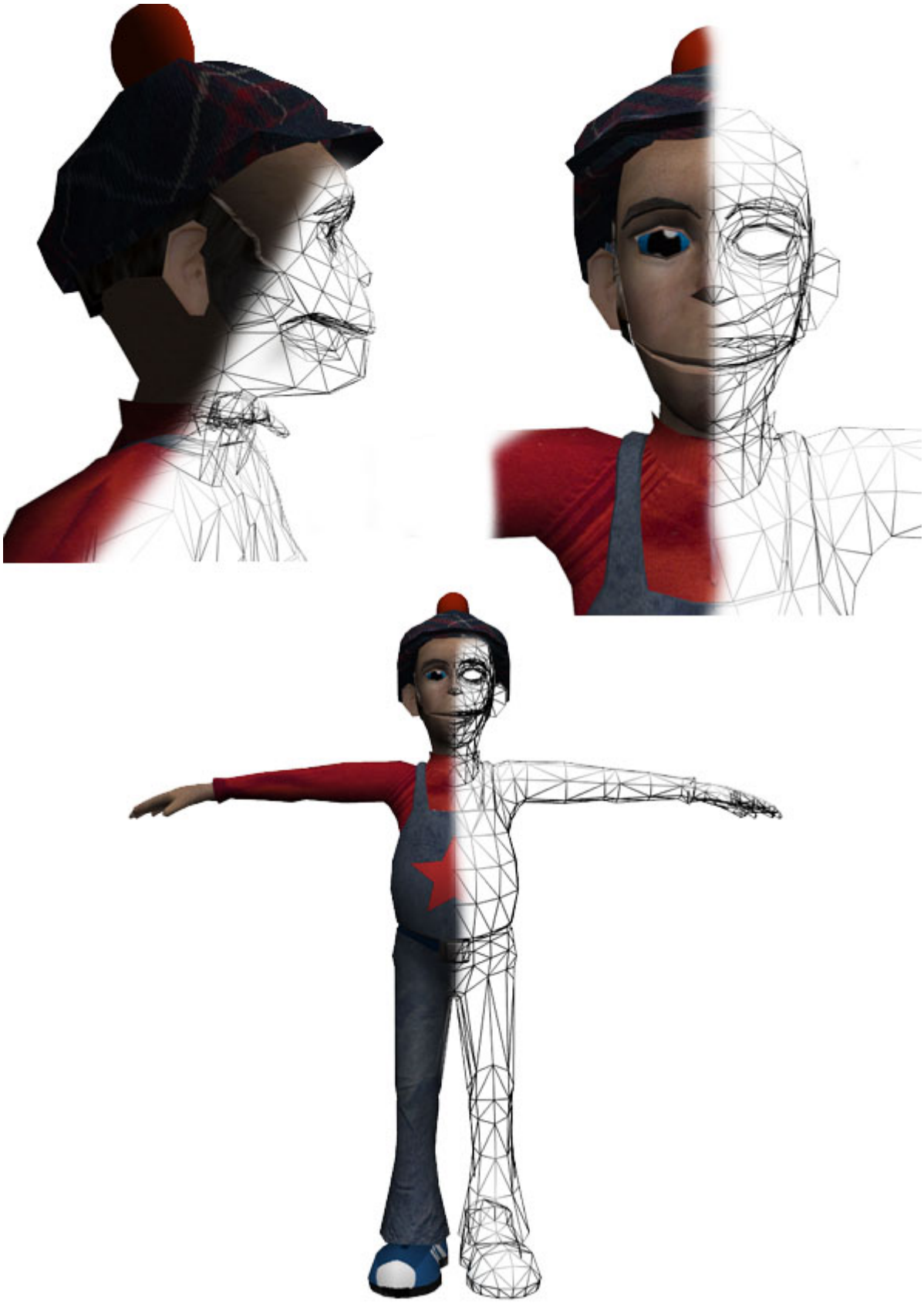

Figure 6.1: Storyteller character model

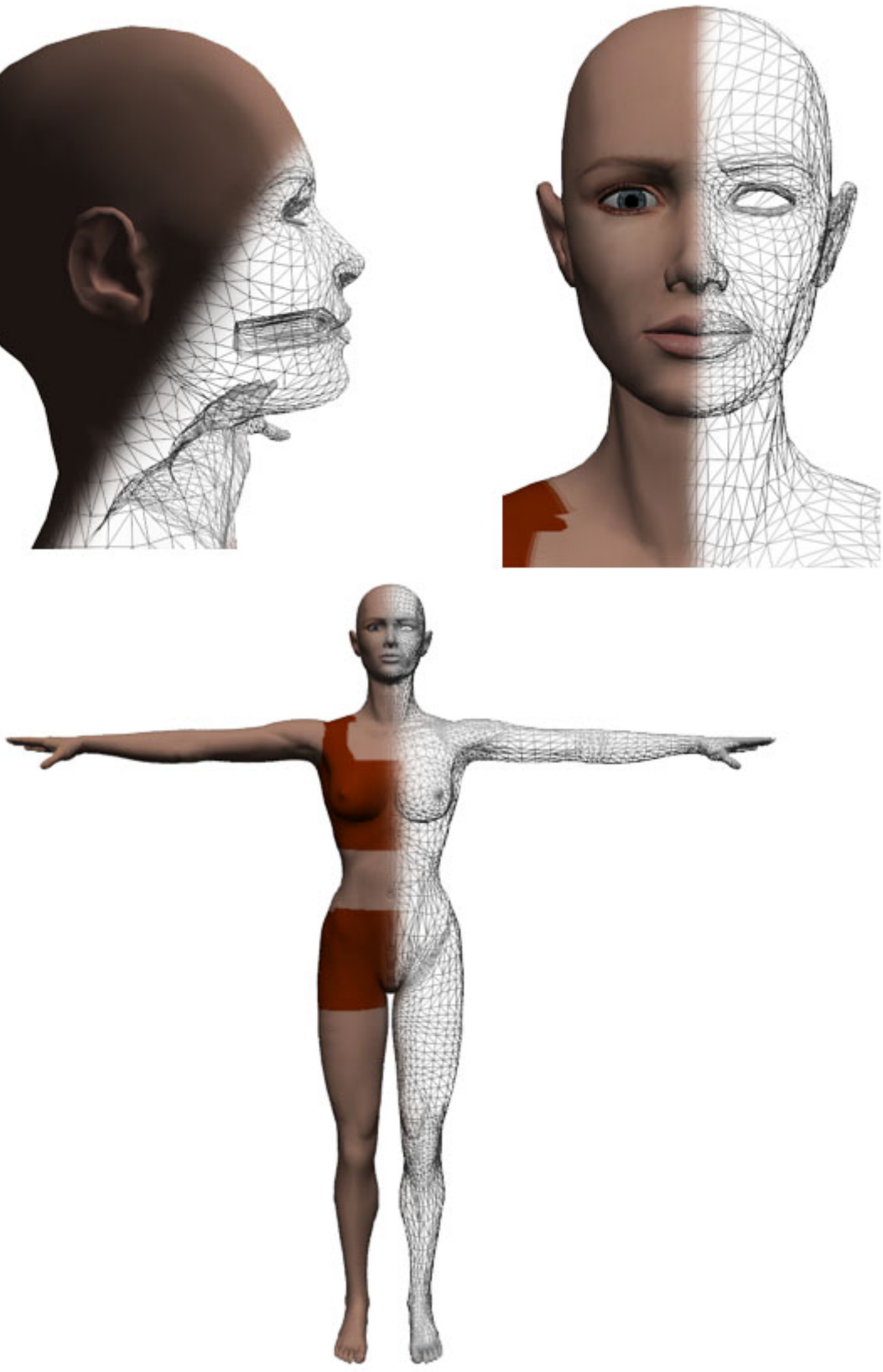

Figure 6.2: Woman character model

6.2 Parameterization and Instantiation

Table 6.1 displays the parameterization used for the illustrative example.

Name	Description	Min	Max	Unit
Open Jaw	Jaw rotates opening the mouth.	0	15	Degrees
Raise left middle eyebrow	Raises the left middle eyebrow region	-900	900	Mpeg4 ENS FAPU $\frac{EyeNoseSeparation}{1024}$
Raise right middle eyebrow	Raises the right middle eyebrow region	-900	900	Mpeg4 ENS FAPU $\frac{EyeNoseSeparation}{1024}$
Raise top lip	Raises the upper lip moving the middle lip region toward the nose tip.	-1000	1000	Mpeg4 MNS FAPU $\frac{MouthNoseSeparation}{1024}$
Lower bottom lip	Lowers the bottom lip moving the middle lip region toward the chin.	-1000	1000	Mpeg4 MNS FAPU $\frac{MouthNoseSeparation}{1024}$
Pull left mouth corner	Pulls backward the mouth left corner.	-600	600	Mpeg4 MW FAPU $\frac{MouthWidth}{1024}$
Pull right mouth corner	Pulls backward the mouth right corner.	-600	600	Mpeg4 MW FAPU $\frac{MouthWidth}{1024}$
Lip Puckerer	The lips are drawn together and slightly forward.	0	1	Not used
Surprise	Expresses surprise. The eyebrows are raised. The jaw and mouth are opened.	0	1	Not used

Table 6.1: Example parameterization

For the remaining of this section we shall demonstrate how each of the above defined parameters may be instanciated. Different options will be debated and their visual results for different parameter values are portrayed.

Open Jaw

Parameters such as this one, or others that imply the movement of a bone, are best represented by an instantiation based on the skeleton model. The Jaw Pitch parameter instantiation offers a direct representation of this concept. Figure 6.3 illustrates the visual results of this instantiation when its parameter value varies from neutral to maximum intensity.

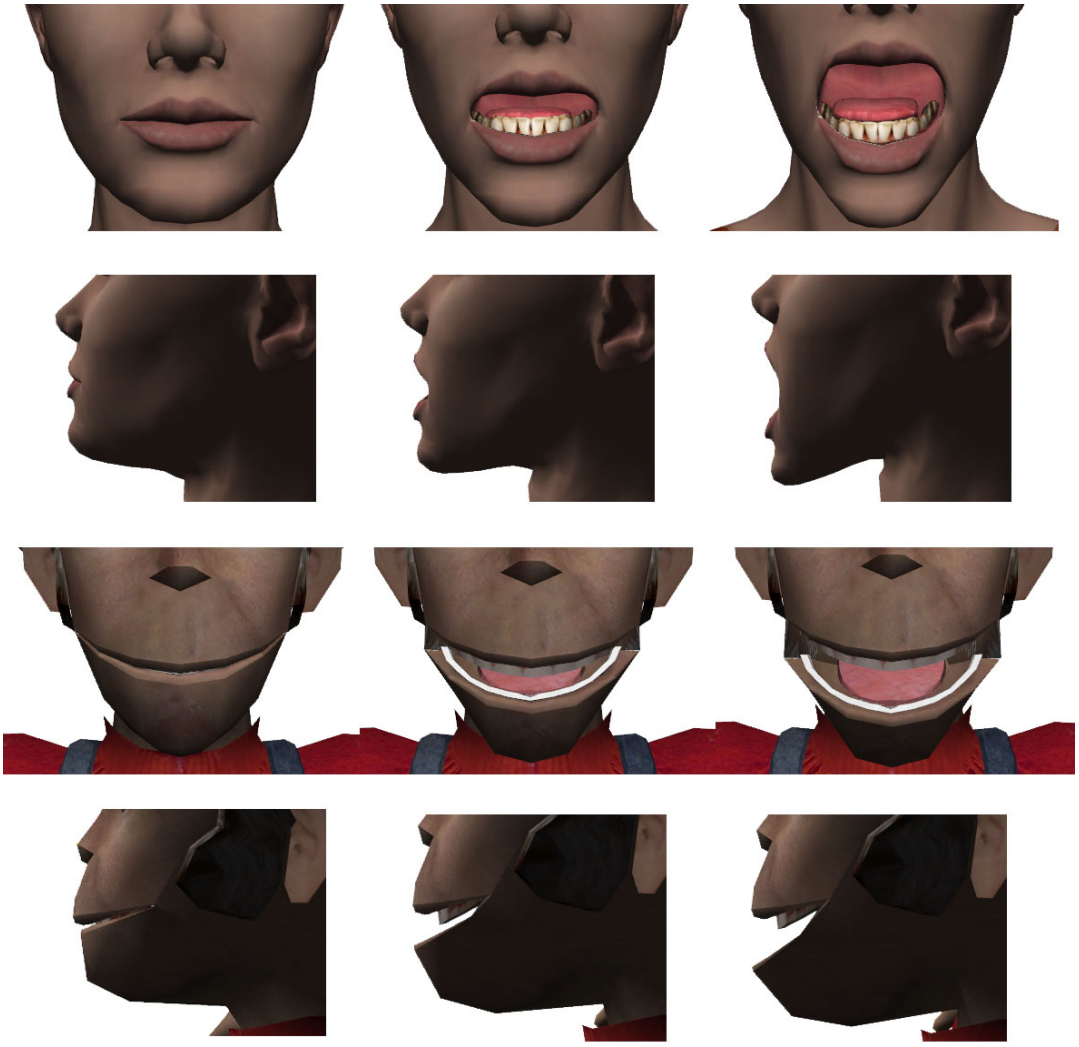

Figure 6.3: Open Jaw deformation results

Raise left/right middle eyebrow

To model these parameters, vector muscle and RBF parameter instantiations are suitable. RBF based parameter instantiations require the definition of not only the control points to move as well as neighboring points to delimit the deformation. In fact, when a parameterization

has few low-level parameters that represent the movement of face points it is common to have more delimiting points than parameter controlled ones. Figure 6.4 shows an example of the region delimiting points that would be required for the parameters discussed here.

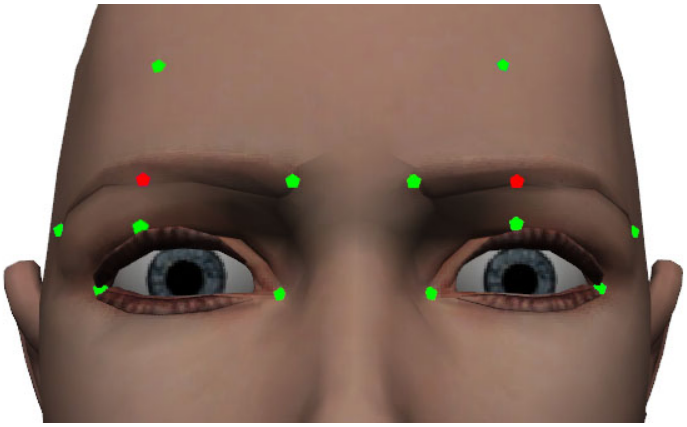

Figure 6.4: Control points for RBF eyebrow parameter instanciation. Points in red are movable. Points in green serve as region delimiters.

Furthermore due to the way that RBF instantiations compute their influence region some problems arise when using such technique on the storyteller model. When the parameter causes the associated control point to move, the eyebrow remains still. In Figure 6.5 this can be noted in the sequence to the left. On the right the gap that is left due to the eyebrow not being moved is evidenced.

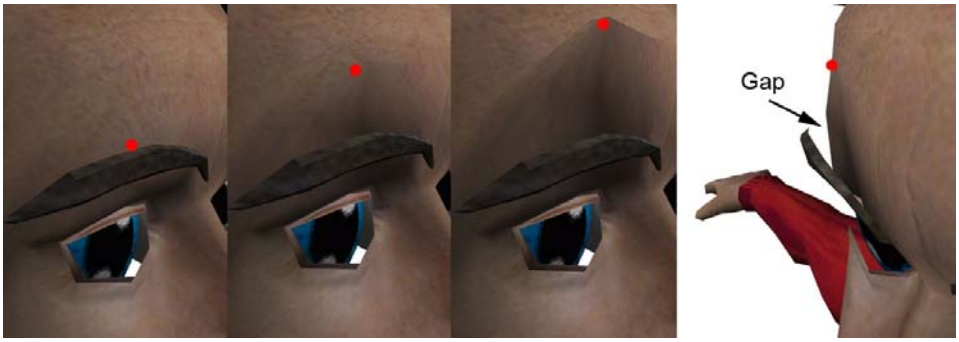

Figure 6.5: Example of issues that arise from the model construction.

As mentioned, this model was created through the aggregation of smaller elements. However, no actual merging was performed. This means that although the mesh structure of the model is only one, there are several meshes in it. In other words, there is no way to continuously traverse all mesh points from any given point. Since vector muscle instantiations use a

volumetric (as opposed to surface) method to determine the influence region, this problem is not evidenced when using them. Thus, in this case using vector muscle instantiations is best fitted. Figure 6.6 portrays the result of applying linear vector muscles for these parameters. The first two rows show the parameters value going from zero to its minimum value while the bottom two rows depict a variation to the maximum value.

Figure 6.6: Eyebrow parameters instantiated with linear vector muscles.

Raise top/bottom lip - Pull left/right mouth corner

These parameters pose similar conditions to the eyebrow parameters. However, there are two key differences. The first is that the top and bottom lip are close together and we wish to manipulate them separately. This issue makes it harder to define the influence regions for vector muscles. Special care must be taken and the influence region must be customized by removing unwanted affected vertexes. On the other hand a RBF parameter instantiation uses a surface propagation method to determine influence regions. This method deals perfectly with the problem of the lips being near each other. The second key difference is the quality of the visual deformation. While linear muscles are fitted to model the pull mouth corners parameters no single available muscle deformation provides the desired visual effect for the top and bottom lip vertical control. Therefore, we used control point unidirectional parameter

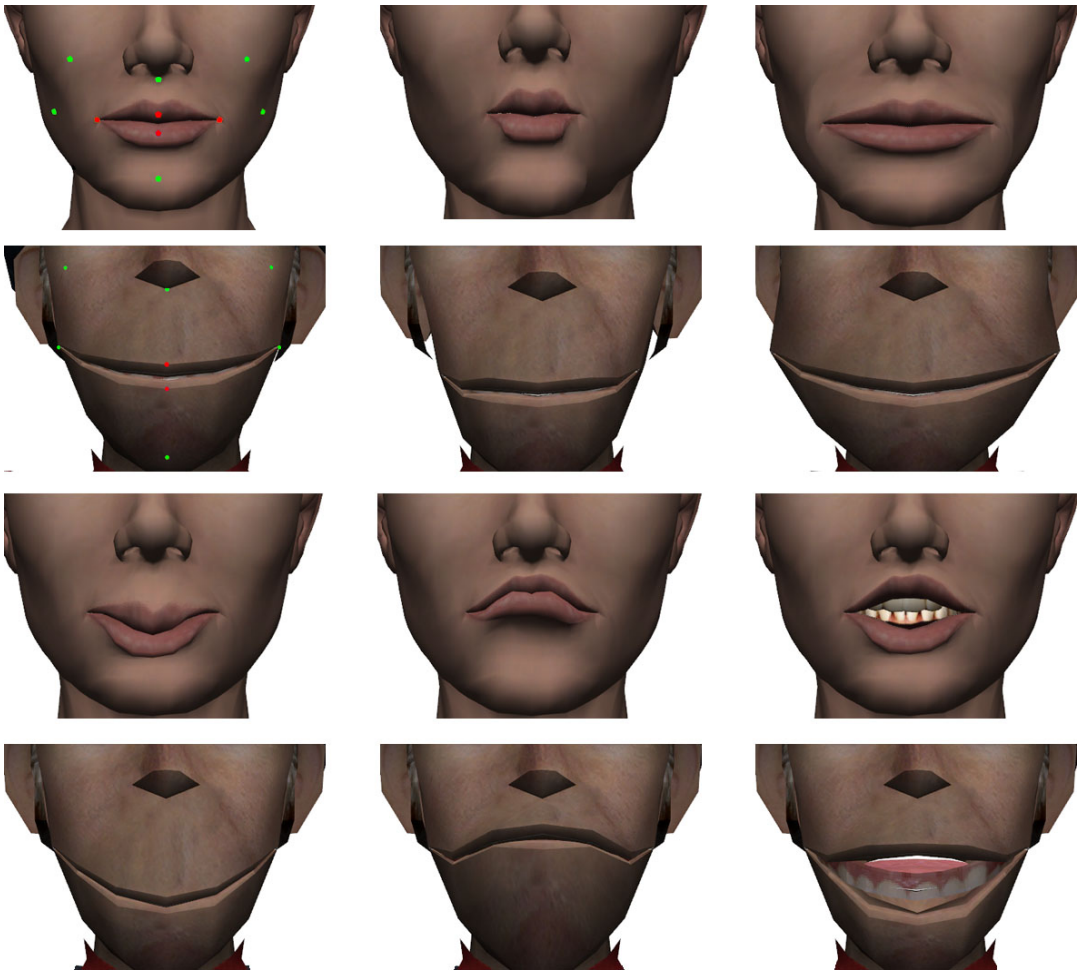

Figure 6.7: RBF instantiations applied to the mouth.

instantiations for these parameters. Figure 6.7 depicts the chosen control points and some visual results obtained by varying the associated parameter values.

Lip Puckerer

This parameter does not make use of unit values for inter-model portability. Thus, special care must be taken if we wish to obtain similar results between models. To achieve this goal it is necessary not only to choose a suitable parameter representation but also to empirically set the unit value for the instantiation. This latter definition is to be done in a manner that, when the maximum and minimum values of the parameter are set, the visual results in several models are alike. This evidently has some limitations. Doing such comparison is only possible if we have a reference model and is limited to personal judgment. Thus if no such model is available the same parameter might have significant different results between models.

From the description of this parameter the first parameter instantiation that occurs to mind is the sphincter muscle. Figure 6.8 shows the effects when using this type of deformation.

Figure 6.8: Applied sphincter muscle deformation example.

For the synthetic woman model another alternative is to use a group parameter instantiation. Unfortunately the extremely exaggerated mouth of the storyteller model prevents us from choosing this alternative. The parameterization defines four parameters that control the mouth. We can use them together to create the desired deformation described by this parameter. Figure 6.9 illustrates the deformation caused when using a group parameter that controls the four parameters defined for the mouth.

Figure 6.9: Group parameter representing lip protrusion.

Surprise

This is a high-level parameter. As such, it involves movement in several regions of the face. High-level parameters are typically modeled by either group or weight blending parameter instantiations. The latter is the more powerful in what regards quality of expression. However,

since it requires a hand crafted expression by an expert character designer, it is only used when such high quality is a necessity. The group parameter instantiation is commonly a better choice even if does not yield the same quality as a weighted blending one. This instantiation assures that the parameter elements that constitute it do not exceed their minimum and maximum values. Therefore, if any given face region is controlled by one, and only one, low-level parameter, and all multiple region influencing parameters, such as the one defined here, are group parameters, it is guaranteed that all the combinations of parameter values show a facial deformation within the expected limits.

Similarly to the Lip Puckerer parameter, the Surprise parameter does not define a Unit. Notwithstanding, it is interesting to notice that this fact does not have the same impact in this parameter. High-level parameters are often much more subjective than low-level ones. While low-level parameters represent precise notions, high-level ones correspond to more general descriptions. Thus, inter-model portability is not seriously hindered by the absence of a model relative measurement to scale the parameter value.

Figure 6.10 displays the result of using a group parameter instantiation with the above mentioned eyebrow and jaw parameters as its elements.

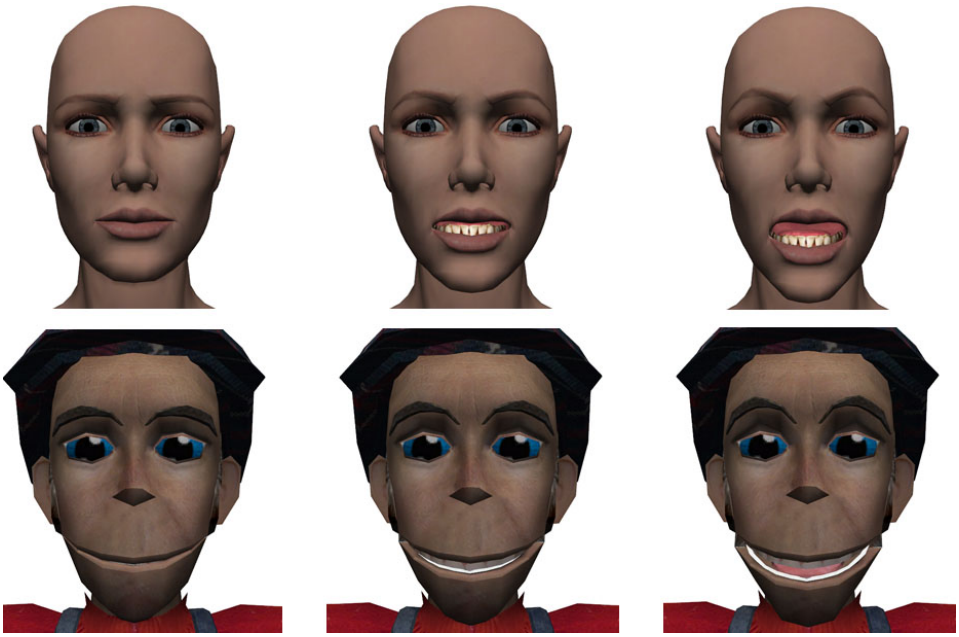

Figure 6.10: Surprise expression.

6.3 Summary

In this chapter we defined a small example parameterization. With it we depicted how the different parameter instantiations are suitable to be applied depending on different conditions. The deformation results for these instantiations are shown through the use of several figures.

Chapter 7

Conclusion

One might say that all research questions in this dissertation spawn from the previously quoted statement of Parke and Waters: "The development of control parameterizations and the development of animation systems implementations should be *decoupled*." However, this approach lead to the research goal that we tried to show in this thesis. *Is it possible to create a facial animation system that supports any given control parameterization?*

We argued that no parameterization is ideal. Its suitable level of abstraction, number of parameters and other properties depend on what context it will be used. Since one can specify a parameterization to be as detailed as one can imagine, having a system that is able to model efficiently *any* such parameterization is almost an utopy. Nevertheless, the developed system attempts to be as generic as possible in order to encompass as much parameter concepts as possible. To help achieve this goal, it draws on ideas from past developed parameterizations such as the use of model relative measurements.

In the end, the concept of a parameter is nothing more than a unidimensional value that has some semantic meaning associated with it. This meaning has to be modeled into the geometric model somehow. This brings us to our next question: *What techniques should be supplied to maximize the number of supported parameters?*

A thorough analysis of the existing facial animation systems was performed. These were classified according to their nature. Their advantages/disadvantages were evidenced and, based on these, three deformation techniques were chosen. The key-framing technique gave way to the Weight Blending parameter instantiation. With it, it is possible to model high quality high-level parameters. Vector Muscles model low-level parameters that correspond to the

contraction of facial muscles. This technique was modified in order to enhance its portability between models. Also, a radial based function approach was used. This latter technique allows for an easier setup of low-level parameters that consist on the movement of points of the face. Head and jaw movement can be achieved through skeleton based parameter instantiations. The group parameter instantiation is used to aggregate other parameters and can be applied to model parameters of variable level of abstraction.

Each of these parameter instantiations has its own properties that makes it suitable to be used under different requirements. Together they grant the system with the ability to model a myriad of parameters in the easiest possible way.

Still, parameters *per se* do not guarantee an expressive face. The quality of motion in the face is as important as the quality of the deformations. This leads us to yet another question: *In an animation how should parameter values vary with time?*

Animators are faced with this question for every animation they create. In order to come to a conclusion they take into consideration many aspects such as character's personality and conveyed emotions. They then interpret these aspects into system animation variables such as speed and acceleration. In this thesis we use a cardinal spline interpolator to model the quality of motion in an animation. We have created a method to automatically determine the tension at each key-frame so that motion is, in our understanding, smooth and "natural". Besides the typical play, pause, resume and stop animation functionalities the developed system supports multiple concurrent animation playback.

Two higher level concepts are supported by the system. Perlin noise allows the use of controlled pseudo-random motions. This affords a powerful and yet simple method to increase the believability of a given synthetic character. It is also possible to use speech synchronized facial animation through the use of a text to speech engine and a simple coarticulation model.

An editor application was created in order to support all of the above mentioned functionalities. This editor gives to the user the ability to create and interactively test parameterizations, parameters instantiations, animations, pseudo-random movements and speech.

Through the development of this work I believe I have gained considerable knowledge of facial animation systems and also to have fulfilled the goals that have been set upon me in the beginning of this rewarding endeavor.

7.1 Future Work

Throughout this dissertation a real-time parameterized facial animation system was presented. This system uses several deformation techniques to allow the expression of a broad range of facial behavior. However, not all possible deformations are contemplated explicitly. Such an example is the creation of wrinkles and furrows. Although ultimately these can be modeled by the use of the weighted blending parameter instantiation, this solution is not ideal. Other parameter instantiations should be created in order to model this, and possibly other, kinds of skin behaviors. The system was built in a way to facilitate the addition of more parameter instantiations. Thus, extending it with such abilities should be straightforward.

RBF parameter instantiations create deformations taking into account the complete set of defined control points. How much a given control point influences a vertex is determinated not only by itself but also by other measurements relating other control points in the vicinity. This is not true for vector muscle parameter instantiations. Here the influence of each muscle over a given vertex is determined independently. The end result consists of the simple sum of the displacement of each influencing muscle. The same is true for the overall usage of different deformation techniques. Each parameter instantiation of different deformation techniques is not "aware" that other parameter instantiations exist and may be active. From our experience this rather simplistic approach proves to be effective for most case scenarios. Nevertheless, when large deformations occur in the same region due to the influence of several parameters unwanted effects may appear. It would be interesting to extend the RBF deformation logic to all of the deformation techniques. How a given parameter instantiation influences a point in the face should take into account other parameters that influence the same point.

At the moment the support for multiple animations consists on the addition of the values of concurrent parameters. In some situations it would be useful to be able to deactivate some of the parameters in an animation. This could be achieved through explicitly deactivating parameters or with a mechanism of priorities. This would permit a larger reusability of existing animations.

This thesis focused on the kinetic aspects of facial animation. Another facet that may prove of interest is the facial skin itself. With the programmable graphical pipeline it should be possible to create a shader to emulate the skin behavior to light. The simulation of hair may also be a fruitful enhancement to the system.

Bibliography

[1] 3d studio max. Internet: http://www.autodesk.com/3dsmax, last seen in 13/09/2006.

[2] Festival, the festival speech synthesis systems.
Internet: http://www.cstr.ed.ac.uk/projects/festival/, last seen in 13/09/2006.

[3] Maya. Internet: http://www.autodesk.com/maya, last seen in 13/09/2006.

[4] Mono. Internet: http://www.mono-project.com/, last seen in 13/10/2006.

[5] Poser 5. Internet: http://www.e-frontier.com/go/poser5/, last seen in 13/09/2006.

[6] Sumit Basu, Nuria Oliver, and Alex Pentland. 3d modeling and tracking of human lip motions. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*, page 337, Washington, DC, USA, 1998. IEEE Computer Society.

[7] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 35–42, New York, NY, USA, 1992. ACM Press.

[8] Igor S. Pandzic (Editor) and Robert Forchheimer (Editor). *Mpeg-4 Facial Animation. The Standard, Implementation and Applications*, pages 17–40. John Wiley & Sons, Ltd, 2002.

[9] Igor S. Pandzic (Editor) and Robert Forchheimer (Editor). *Mpeg-4 Facial Animation. The Standard, Implementation and Applications*, pages 3–13. John Wiley & Sons, Ltd, 2002.

[10] Rob Koenen (Editor). Overview of the mpeg-4 standard.
Internet: http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.html, last seen in 14/09/2006.

[11] Paul Ekman. *The face of man: expressions of universal emotions in a New Guinea village*. Garland STPM Press, 1980.

[12] Paul Ekman. Should we call it expression or communication? *Innovations in Social Science Research*, 10(4):333–344, 1997.

[13] Paul Ekman and Wallace V. Friesen. *Manual for the Facial Action Coding System.* Consulting Psychologists Press, Inc., 1978.

[14] Prem Kalra and Nadia Magnenat-Thalmann. Modeling of vascular expressions in facial animation. In *Computer Animation*, pages 50–58, 1994.

[15] Prem Kalra, Angelo Mangili, Nadia Magnenat Thalmann, and Daniel Thalmann. Simulation of facial muscle actions based on rational free form deformations. In *1992*, pages 59–69, Cambridge, Eurographics '92.

[16] Kolja Kähler, Jörg Haber, and Hans-Peter Seidel. Geometry-based muscle modeling for facial animation. In *GRIN'01: No description on Graphics interface 2001*, pages 37–46, Toronto, Ont., Canada, Canada, 2001. Canadian Information Processing Society.

[17] Sumedha Kshirsagar, Stephane Garchery, and Nadia Magnenat-Thalmann. Feature point based mesh deformation applied to mpeg-4 facial animation. In *DEFORM '00/AVATARS '00: Proceedings of the IFIP TC5/WG5.10 DEFORM'2000 Workshop and AVATARS'2000 Workshop on Deformable Avatars*, pages 24–34, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V.

[18] Yuencheng Lee, Demetri Terzopoulos, and Keith Waters. Realistic modeling for facial animation. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 55–62, New York, NY, USA, 1995. ACM Press.

[19] Nadia Magnenat-Thalmann, E. Primeau, and Daniel Thalmann. Abstract muscle actions procedures for human face animation. *Visual Computer*, 3(5):290–297, 1988.

[20] Celso Melo. Gesticulation expression in virtual humans. Master's thesis, Universidade Técnica de Lisboa, Instituto Superior Técnico, 2006.

[21] Frederic I. Parke. *A parametric model for human faces.* PhD thesis, University of Utah, Salt Lake City, 1974.

[22] Frederic I. Parke and Keith Waters. *Computer Facial Animation*, pages 145–147. A K Peters, Ltd, 1996.

[23] Frederic I. Parke and Keith Waters. *Computer Facial Animation*, pages 229–234. A K Peters, Ltd, 1996.

[24] Stefano Pasquariello and Catherine Pelachaud. Greta: A simple facial animation engine. In *6th Online World Conference on Soft Computing in Industrial Appications, Session on Soft Computing for Intelligent 3D Agents*, 2001.

[25] Ken Perlin. Course in "advanced image synthesis"., 1984.

[26] Ken Perlin and Athomas Goldberg. Improv: a system for scripting interactive actors in virtual worlds. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 205–216, New York, NY, USA, 1996. ACM Press.

[27] F. Pighin, J. Aushlander, and D. Lischinski. Realistic facial animation using image-based 3d-morphing, technical report uw-cse-97-01-03. Technical report, 1997.

[28] Frédéric Pighin, Jamie Hecker, Dani Lischinski, Richard Szeliski, and David H. Salesin. Synthesizing realistic facial expressions from photographs. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 75–84, New York, NY, USA, 1998. ACM Press.

[29] Stephen M. Platt and Norman I. Badler. Animating facial expressions. In *SIGGRAPH '81: Proceedings of the 8th annual conference on Computer graphics and interactive techniques*, pages 245–252, New York, NY, USA, 1981. ACM Press.

[30] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 151–160, New York, NY, USA, 1986. ACM Press.

[31] H. Sera, S. Morishima, and D. Terzopoulos. Physics-based muscle model for mouth shape control. In *5th IEEE International Workshop on Robot and Human Communication*, pages 207–212, 1996.

[32] Demetri Terzopoulos and Keith Waters. Physically-based facial modeling, analysis and animation. *Journal of Visualization and Computer Animation*, 1(2):73–80, 1990.

[33] Valve. Half life 2. Internet: http://half-life2.com/, last seen in 12/09/2006.

[34] Carol L. Y. Wang and David R. Forsey. A new facial animation system. In *Computer Animation*, pages 59–68, 1994.

[35] Keith Waters. A muscle model for animation three-dimensional facial expression. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 17–24, New York, NY, USA, 1987. ACM Press.

[36] Eric W. Weisstein. Voronoi diagram. Internet: From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/VoronoiDiagram.html, last seen in 10/09/2006.

[37] Jun yong Noh, Douglas Fidaleo, and Ulrich Neumann. Animated deformations with radial basis functions. In *VRST '00: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 166–174, New York, NY, USA, 2000. ACM Press.

Appendix A

Random Generated Facial Expressions

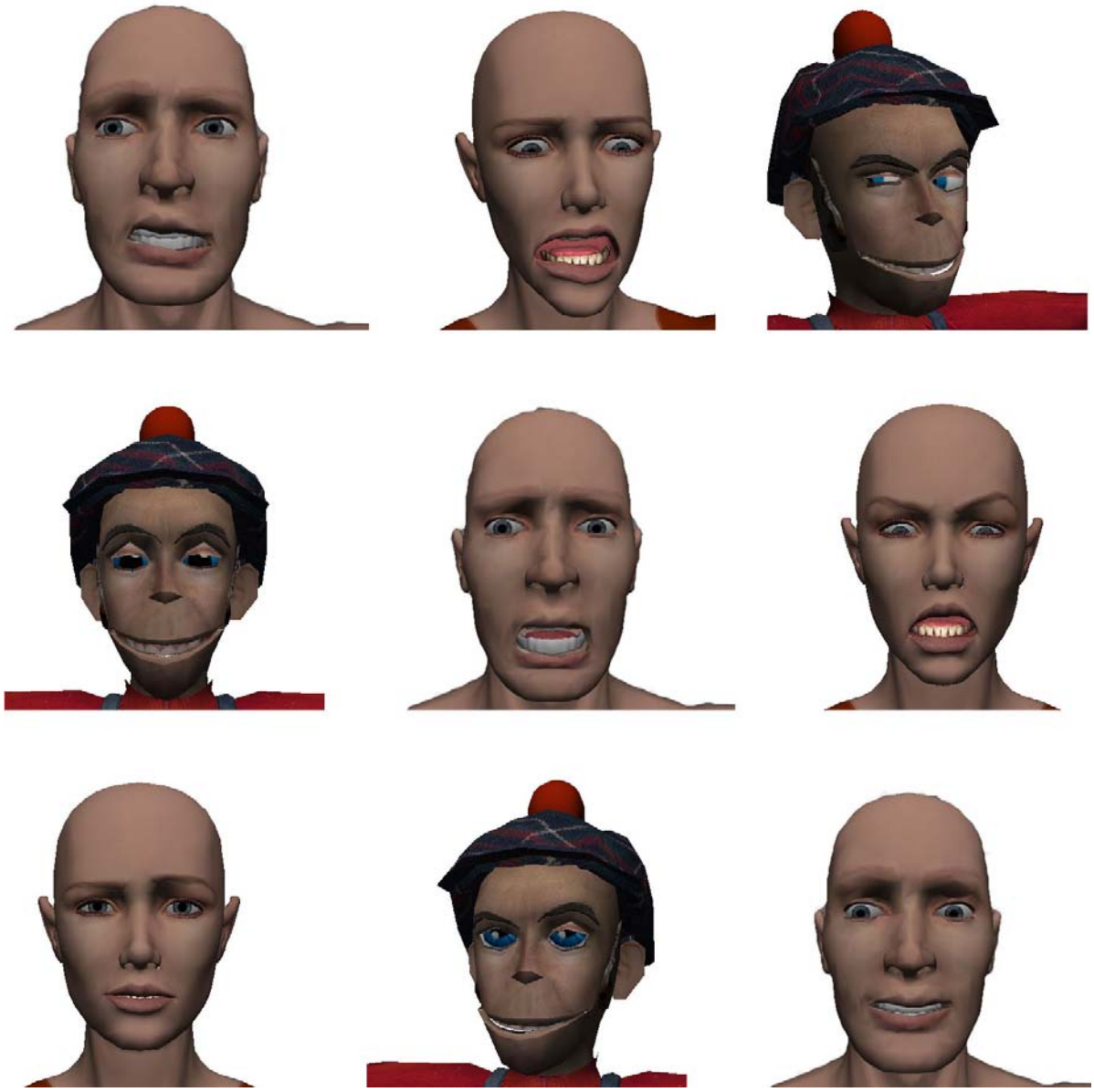

Figure A.1: Random generated facial expression set I

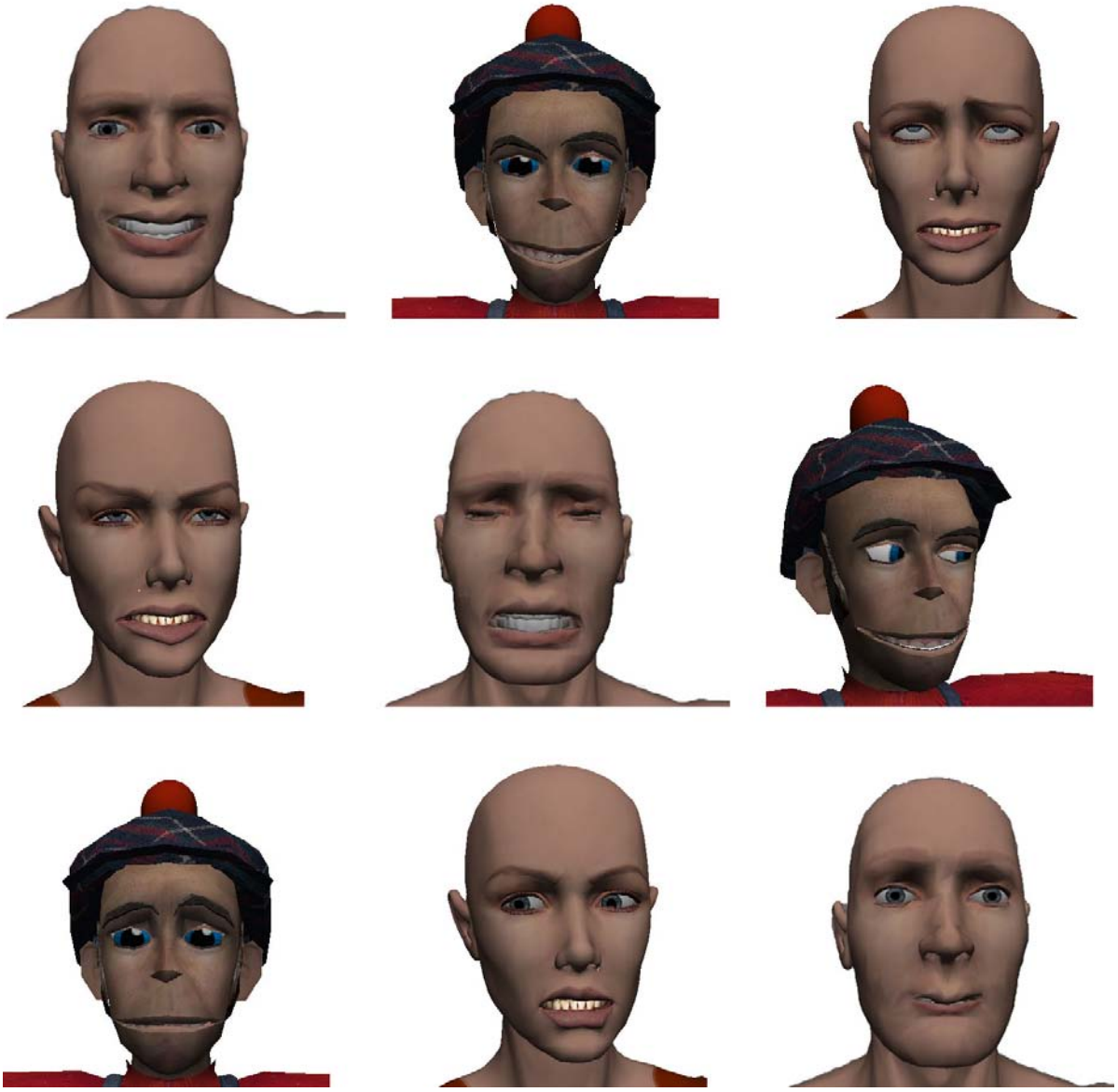

Figure A.2: Random generated facial expression set II

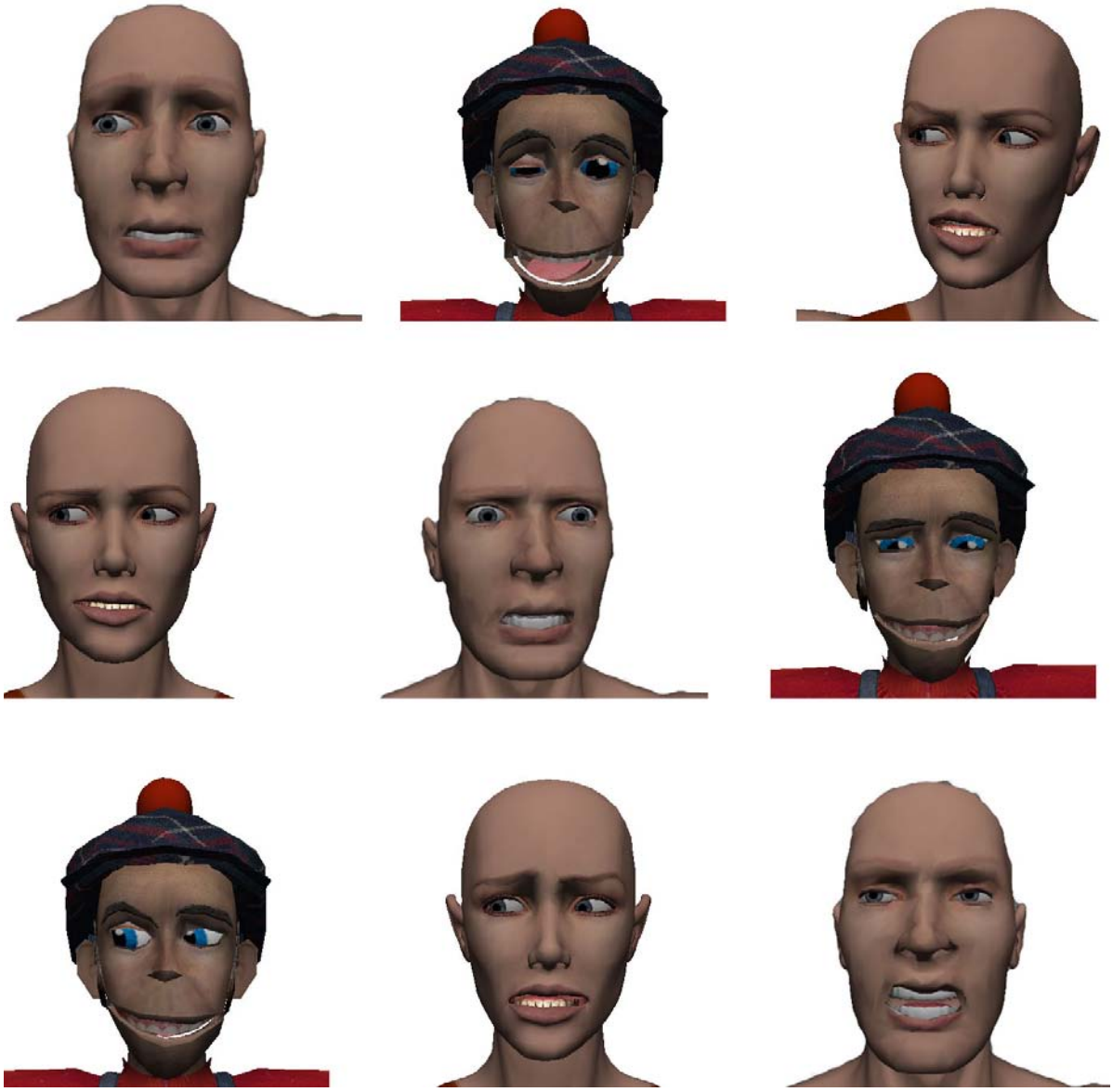

Figure A.3: Random generated facial expression set III