THE UNIVERSITY OF THE SOUTH PACIFIC
LIBRARY

Author Statement of Accessibility- Part 2- Permission for Internet Access

Name of Candidate: Dinash Kumar
Degree: MSc in Computing Science
Department/School: School of Computing, Information and Mathematical Sciences
Institution/University: University of the South Pacific
Thesis Title: Animating Facial Expressions with Facial Action Coding System
Date of completion of requirements for award: 19 August 2010

1. I authorise the University to make this thesis available on the Internet for access by USP authorised users. [Yes/No]

2. I authorise the University to make this thesis available on the Internet under the International digital theses project. [Yes/No]

Signed: ____________________________
Date: 19/08/2010

Contact Address
C: /SSTEMS /FSTE,
USP Lautoka Bay Campus
Suva.
PH: (w) 8232623
email: kumar_dj@usp.ac.fj

Permanent Address
56 Nayava Road,
Ravua, Suva.
Animating Facial Expressions with Facial Action Coding System

by

Dinesh Kumar, PGDCS, BA

A thesis submitted for the partial fulfillment of the requirements for the degree of Master of Science in Computing Science.

University of the South Pacific

August, 2010

© Dinesh Kumar 2010
Contents

Abstract vii

Acknowledgments viii

Declaration ix

1. Introduction 1
   1.1 Motivation 1
   1.2 Our Research 2
   1.3 Outline 5

2. Facial Animation Techniques 7
   2.1 Introduction 7
   2.2 The Human Face Model 8
   2.3 Facial Modeling Techniques 9
      2.3.1 Traditional modeling 9
      2.3.2 Photograph and digitize 14
      2.3.3 Sculpt and digitize (with a 3D digitizer) 15
      2.3.4 Scanning (laser) 16
   2.4 Facial Animation Techniques 16
      2.4.1 Key-frame Interpolation 18
      2.4.2 Parameterized Systems 22
      2.4.3 Muscle Based Systems 23
      2.4.4 Motion Capture 24
   2.5 Summary 25

3. Facial Action Coding System 27
   3.1 Introduction 27
   3.2 The Anatomy of Human Face 28
   3.3 FACS Action Units (AUs) 30
      3.3.1 FACS Action Descriptors 33
   3.4 FACS-based Facial Animation 33
      3.4.1 Co-occurrence Rules (AU combination types) 36
3.5 Summary

4. The Reference Face Model 39
   4.1 Introduction 39
   4.2 Waters Muscle Model 40
      4.2.1 Linear Muscle Model 40
      4.2.2 Sphincter Muscle Model 43
      4.2.3 Sheet Muscle Model 48
      4.2.4 Jaw Rotation 51
   4.3 Suitability of Waters Muscle Model for Facial Animation 53
   4.4 Summary 54

5. Experiments & Results 55
   5.1 Introduction 55
   5.2 System Overview 55
   5.3 Development Information 56
      5.3.2 Resources used as background 59
   5.4 Facial Animation Player (FAP) 61
      5.4.1 Program Enhancements 63
      5.4.2 Animation of Waters Primitive Expressions 67
      5.4.3 Expression Editor 68
      5.4.4 The Remote Player (receiver side) 71
      5.4.5 The AU Transmitter (sender side) 84
      5.4.6 Test Results 86

6. Conclusion & Future Work 89

Appendix A 92
Appendix B 95
Appendix C 99
Appendix D 101
Bibliography 103
# List of Figures

<table>
<thead>
<tr>
<th>Number</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1</td>
<td>11</td>
<td>Polygonal Surfaces (Mesh). (Picture courtesy of Ladislav Kunc, Computer Graphics Group (CGG), Prague)</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>12</td>
<td>A human head constructed with NURBS curves. Source: <a href="http://www.3drender.com/nurbana/">www.3drender.com/nurbana/</a></td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>13</td>
<td>First three steps of Catmull–Clark subdivision of a cube with subdivision surface below. (<a href="http://commons.wikimedia.org/wiki">http://commons.wikimedia.org/wiki</a>)</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>14</td>
<td>Photograph and Digitize (Source: Lecture notes: CSE169: Computer Animation UCSD Winter 2005)</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>15</td>
<td>Sculpting and digitizing process</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>16</td>
<td>Microscribe 3D Laser Systems (<a href="http://microscribe.ghost3d.com/gt_microscribe.htm">http://microscribe.ghost3d.com/gt_microscribe.htm</a>)</td>
</tr>
<tr>
<td>Figure 2.7</td>
<td>17</td>
<td>Classification of Facial Modeling and Animation techniques. Source: Survey of Facial Modeling and Animations Techniques. [14]</td>
</tr>
<tr>
<td>Figure 2.8</td>
<td>18</td>
<td>3 key-frames for the falling ball animation sequence.</td>
</tr>
<tr>
<td>Figure 2.9</td>
<td>20</td>
<td>Linear Interpolation</td>
</tr>
<tr>
<td>Figure 2.10</td>
<td>21</td>
<td>Cosine Interpolation</td>
</tr>
<tr>
<td>Figure 2.11</td>
<td>24</td>
<td>Waters original muscle model</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>29</td>
<td>The anatomy of muscles on a human face [24].</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>30</td>
<td>Six primary expressions. Source: [26]</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>32</td>
<td>Emotional axes. Source [26]</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>37</td>
<td>Dependencies between Action Units [9].</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>41</td>
<td>Linear Muscle Model</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>42</td>
<td>Linear Muscle Contraction with $\Omega = 35.0$, $R_s = 7.0$, $R_y = 14.0$, $K = 0.75$, elasticity = 1.0. Source [2].</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>43</td>
<td>Sphincter Muscle Model</td>
</tr>
</tbody>
</table>
Figure 4.4: Sphincter Muscle Contraction. Source [28].

Figure 4.5: Application of the sphincter muscle on the area around the mouth.

Figure 4.6: Vertices converging to the semi-major axis

Figure 4.7: Sphincter muscle diagram for use around the eyes.

Figure 4.8: Using sphincter muscles around the eyes. $K = 0.2$ for left eye (to the reader) and $K = 0.4$ for the right eye. $Z = 30$.

Figure 4.9: Sheet Muscle Model

Figure 4.10: Relationship between $P$ and the sheet muscle for calculation of $q$.

Figure 4.11: Implementation of the sheet muscles. $K = 0.4$, $L - |A_l| = 2$

Figure 4.12: Discontinuity at the lip corners. [4]

Figure 4.13: Weights allocated to vertices forming the lower part of the jaw.

Figure 5.1: Architecture of our Facial Animation Player

Figure 5.2: Original Waters program interface

Figure 5.3: Muscles implemented in FAP (new muscles outlined in blue)

Figure 5.4: The upper layer UI for our FAP.

Figure 5.5: Rendering modes available in FAP

Figure 5.6: Rendering modes in FAP. (a) Polygon (b) Wireframe (c) Transparent (d) Muscles (e) Textured (f) Verts (g) Jaw tags (h) Eye tags

Figure 5.7: Transformation from neutral to happy in steps of 10 in-between keyframes with timer callback set at 5ms.

Figure 5.8: Animation Engine Algorithm

Figure 5.9: Expression editor UI

Figure 5.10: Remote Player UI

Figure 5.11: AU Reader Algorithm

Figure 5.12: Implementation of sub functions A and B

Figure 5.13: Implementation of sub functions C and D

Figure 5.14: FIFO queue implementation of Playout buffer
Figure 5.15: Payback controller algorithm  
Figure 5.16: Example format of file read by AU Transmitter 
Figure 5.17: Expression results of the 6 basic expressions by applying co-occurrence rules and maximum mixer. 
Figure 5.18: Results of some of the combinations of the 6 basic expressions.
List of Tables

<table>
<thead>
<tr>
<th>Number</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3.1: Sample single Action Units. <em>Source [14].</em></td>
<td>31</td>
</tr>
<tr>
<td>Table 3.2: Example set of AUs for basic expressions. <em>Source [14].</em></td>
<td>32</td>
</tr>
<tr>
<td>Table 5.1 Average mixer technique used in FAP</td>
<td>84</td>
</tr>
<tr>
<td>Table 5.2 Maximum mixer technique in FAP</td>
<td>84</td>
</tr>
<tr>
<td>Table 5.3: Some standard statistics of FAP rendering on 2 test platforms.</td>
<td>86</td>
</tr>
<tr>
<td>Table 5.4: Application of co-occurrence rules for the 6 basic expressions</td>
<td>86</td>
</tr>
<tr>
<td>Table 5.5: Bandwidth usage for single expressions.</td>
<td>88</td>
</tr>
</tbody>
</table>
Abstract

This thesis focuses on the development of an easy to use real-time 3D human facial expression and animation system driven entirely by Facial Action Coding System developed by psychologists Paul Ekman and Wallace Frieson in 1977 [1]. The Waters muscle based facial model [2] has been used and extended to allow the user to calibrate the facial deformations illustrated by examples images of each AU in [3]. This becomes the basis of the facial expression creation. Our facial model is then animated in a physics-based manner by use of muscle contractions that in turn cause deformations on the facial geometry; the simulation is efficient enough to be rendered real-time on current PC hardware.

Keywords: Facial animation, Facial Expression, Computer Graphics, Facial Action Coding System (FACS), Action Units (AUs), Muscle models, Low-Bandwidth teleconferencing.
Acknowledgments

Carrying through with this thesis would not have been possible without the help and patience received from many people. First and foremost, I owe a special debt to my thesis advisor, Associate Professor Jito Vanualailai, for his encouragement and support, and for making it possible for me to carry out my research at USP. I am particularly grateful to him for taking on the supervisory role after departure of my original supervisor for this thesis, for his enthusiasm and interest in the facial animation aspect of my work as well as for helpful comments and provision of materials.

I also thank all my colleagues at SCIMS USP who helped proof read this thesis and providing valuable advice on improving the presentation.

Thank you All!
Declaration

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of Master of Science in Computer Science is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my work.

Signed ....................................

Date 19th August 2010
Chapter 1

1. Introduction

1.1 Motivation

Face modeling and animation has long fascinated computer graphics researchers, not only for the complexity of the face geometry, but also for the inherent problems in creating facial deformations to create realistic expressions. Recent progress in facial modeling and animation now promises to provide useful and capable tools both for research and entertainment. There are some areas where a good model of a human face is essential for success. For example, a good facial model could be used in such applications as:

- character animation in entertainment industry and advertising,
- low-bandwidth teleconferencing,
- advanced human-computer interfaces,
- computer synthetic facial surgery,
- realistic human representation in the virtual reality space,
- visual speech synthesis for the deaf and hard of hearing,
- face and facial expression recognition and models for the analysis of facial expression by psychologists.
Despite its demand for use in various practical fields, research and applications; the creation and animation of computer generated human faces is still a challenging problem even to this day.

Facial Action Coding System (FACS) has been used as a guide in many facial expression systems. This guide contains classification of a series of Action Units (AUs). Each of these Actions Units describes possible actions that can be performed on a human face in terms of the smallest visible unit of muscular activity. However little has been done to animate faces based on actual AUs. We are hence inspired to investigate this combination (animation and FACS).

1.2 Our Research

In this thesis we study the Facial Action Coding System (FACS) and its application to facial expression animation. We also investigate the area of facial animation by examining some common facial modeling and animation techniques and implementing a Facial Animation Player (FAP). In FAP we demonstrate the application of the FACS standard on geometry of the face based on Waters muscle model (a popular technique in facial expression creation). The animation of the face therefore is controlled by incoming stream of FACS Action Unit (AU) parameters. This ties our system closely with the already available information about human facial behavior given in FACS.

To some extent the computational cost of facial animation has been solved with the advent of high processing power and advanced graphics rendering capabilities of modern computers. What is left therefore is the creation of a human face model and a novel way to animate the model from one state to another over time.
At this point we realize that the first step in the animation process is creation of a synthetic face model. There are many commercial 3D modeling software that allows development of 3D face models for animation. However this process is rather tedious and very time consuming.

Next step is the creation of individual facial expressions. Mathematically describing, facial animation is simply a set of facial expressions played over time. Each expression therefore describes a human emotion such as smile, anger and surprise etcetera. In many facial animation systems developed the user simply creates an expression by tweaking parameters that directly deforms parts of the synthetic face model. In other words basic human expressions are pre-programmed into the system. The resultant expression is normally validated through visual inspection. These expressions are then sent to animation engine. Therefore most facial animations systems present no systematic way for creation of these facial expressions.

This situation motivated my thesis: I propose a 3D expression and animation system that takes the creation of individual expressions to the atomic level. That is instead of generating and recording known expressions we propose a mechanism that will allow us to create and store each atomic facial distortion. We can then combine some of these singular distortions to create meaningful expressions. FACS AUs is one such technique that describes the simplest visible movement, which cannot be decomposed into more basic ones. Hence we use this as the basis for creating atomic facial distortions using visual aids given in [3].
In summary the key contributions made to the field in this thesis are:

- Enhancements to the Waters anatomy-based head model: We use the basic Waters muscle based facial model from [2] but provide enhancements to the basic model by improving its UI, adding sheet and sphincter muscles, provide an alternate implementation to the jaw rotation function, and presenting a new sphincter muscle equations that can be used around the eyes.

- Development of an expression editor that allows creation of atomic facial expressions based on information about human facial behavior given in FACS with ease.

- Generation of compound expressions by activating two or more pre-programmed Action Units (AUs) and viewing their result.

- A simple technique proposed to apply the degree of activation of AUs in expression creation.

- Development of a simple animation engine for simulation.

- Implementation of a low-bandwidth teleconferencing, a real time facial animation player driven only by FACS AUs transmitted as plain text over TCP sockets.

It is therefore envisaged that our system can prove useful to researchers studying the relationship between the generating of facial expression using FACS AUs as the basis.

It is not necessarily our goal to implement the animation of all possible facial expressions; rather we wish to demonstrate the feasibility of using the FACS paradigm for the animation of facial expressions. We have hence selected a subset of AUs and implemented their behavior.
Also this research does not attempt to produce highly realistic facial expressions, but rather to understand some of the principles that are used to model face expressions (FACS) and implement a basic animation system on these principles that would simulate such affects.

This thesis therefore deals with FACS and animation but does not cover FACS AU recognition.

### 1.3 Outline

This thesis can be loosely classified into two parts – the first part covering the theory and the second part dealing with implementation. Chapters 2 and 3 set the pace, covering the theoretical aspects while chapter 5 covers mainly technical implementation details. Chapter 4 is sprinkled with both theory and implementation and forms the basis for chapter 5.

Following the gentle introduction of chapter 1, in the second chapter (Chapter 2) we present a study of the facial modeling and animation techniques developed over the years in this field of research. Advantages and disadvantages of some of the techniques are also discussed.

Since FACS plays an important role in our thesis, a complete chapter is dedicated to this (Chapter 3). In here we explore what exactly FACS is and the idea of using FACS for facial expression generation.

In Chapter 4, we will introduce Waters muscle models [2] which form the basis of our facial animation model. The mathematical principles behind the muscle modeling are explained. This section also explains improvements in terms of more muscles added to the face geometry and changes to the jaw rotation function and their results.
In Chapter 5, the final implementation of the software for this thesis will be described. The animation results produced will also be evaluated.

At the end Chapter 6 we briefly reflect over the entire thesis, both theory and implementation.
Chapter 2

2. Facial Animation Techniques

2.1 Introduction

Facial expression has been the subject of scientific investigation for more than one hundred years. Due to its subject and output type, it is also related to many other scientific and artistic fields from psychology and behavioral biology to traditional animation.

Computer based facial expression modeling and animation is not a new research field [4]. Initial efforts in this area now go back almost 40 years. During this time a variety of algorithms and techniques were developed that made it possible to create highly realistic looking characters [2, 5, 6, 7, 8]. Surprisingly theories and techniques developed decades ago are still being used in more recent research as in [9, 10].

Paralleled with the development of personal computers capable of rendering high quality graphics, computer facial animation has produced more and more realistic facial expressions to enrich human-computer communication. However despite the technological advances, facial animation is humbled by some simple issues. As stated by Ekman and Frieson [1], humans are highly sensitive to visual messages communicated by the face. Consequently, facial animation requires specific algorithms able to render with a high degree of realism the natural characteristics of the motion.
Due to the demand for creation of synthetic faces to look more like a real person, we become less forgiving of imperfections in the modeling and animation. If it looks like a person we expect the facial movements to be similar to real facial movements. This is due to the fact we are extremely sensitive to reading small to even subtle facial characteristics. A slight change in facial distortion can convey a completely different message. For example a mother's frown (tightening of the *frontalis* muscles) causes a child to behave. Due to this complexity, an alternative is to create characters that have non-human characteristics, such as using animal faces like dogs and cats. In this case we are less sensitive to imperfections in the modeling and animation because we have no experience of talking animals.

In this chapter we examine some of the various modeling and animation methods developed over the years. Finally the reasons for choosing one of these techniques of facial modeling and animation are given.

### 2.2 The Human Face Model

The human face is an important and complex communication channel. At first sight, all faces look the same and have the same basic components: pair of eyes, a nose situated in the middle, a mouth in the lower part of the face, eyebrows, chin and cheeks etc. Yet it is the face that plays an important role in communication between people. It provides information about the identity of a person, sex and age. Furthermore it is the face that communicates the emotions through the varied forms of expressions, which of course is an integral part of our daily life.

The human face model consists of the following important visible components used in communication:

---

1 A detailed discussion of the facial muscles is made in Chapter 4
Each of these may require a different technical strategy for development. The following section discusses various techniques of developing such a model.

### 2.3 Facial Modeling Techniques

Preparing the facial geometry and all the necessary expressions can be a lot of work. There are several categories of facial modeling techniques namely:

- Traditional Modeling
- Photograph and digitize (in 2D with a mouse)
- Sculpt and digitize (with a 3D digitizer)
- Scanning (laser)

#### 2.3.1 Traditional modeling

Under traditional modeling, the geometry of the head is normally modeled in an interactive 3D modeler. There are several techniques in use:
2.3.1.1 Polygonal Surfaces

In this type of modeling the geometry of the head is constructed (usually in 3D space) using interconnected vertices. Connection is done either using three vertices (forming a mesh of triangles) or using four vertices (forming a mesh of quads). Polygon meshes (sometimes also called a wireframe) is by far the most commonly used way to implement surfaces in geometric model representations.

The Water's model [2] used in this research is a classic example where the head model is constructed using triangular meshes. The closer the vertices, the smoother the face topology is.

**Advantages:** The modeler has specific control and placement of the vertices. This is an exacting way to get topology (points) where you need it on a face and not where you do not. Also this is an easy format to render, since most graphic cards are optimized to handle triangles. In this research we have used the original mesh dataset from [4] which is further modified to include topology for the eyelids.
Disadvantages: Constructing the face geometry can be a painstaking task as user manually has to work with the creation and manipulation of polygonal data. However the existence of interactive 3D modeling software such as Maya \( http://usa.autodesk.com \), VRMesh \( http://www.vrmesh.com \), Blender \( http://www.blender.org \) which is open source has solved many of the problems associated with creation of both 2D and 3D facial models.

2.3.1.2 Parametric Surfaces (Patches)
In this type of modeling patches (or a set of splines) indirectly define a smooth curve from a set of control points. There are several types of parametric surfaces and they are built like patchwork quilts, constructed by parametric curves. Examples include B-splines, Beta-splines, Bezier patches, NURBS. NURBS which stands for non-uniform rational B-Splines, allows each control point to have its own weight that can affect the "pinch" of the curve at the control point. These curves later can be combined (sewn) to form surfaces.
Detailed information on how NURBS planes are generated can be found at [11].

![Figure 2.2: A human head constructed with NURBS curves.](source: www.3drender.com/nurbana/)

**Advantages:** NURBS describes curves more accurately than polygon. They work very well for smooth surfaces hence suited for facial modeling.

**Disadvantages:** To create a face from patches, one must use control points evenly across the face but the face tends to have areas of high detail (curvature) for example around the eyes, mouth, ears and areas of low detail for example cheeks and forehead. In such cases, one can use one large patch with high density everywhere or a low density patch for the entire face and high density patches specific to areas requiring higher details (eyes, ears etc) and then stitching them to the overall face patch.
2.3.1.3 Sub-Division Surfaces

Subdivision is a method where surfaces based on polygon meshes are divided into finer detail levels [12]. The smooth surface can be calculated from the coarse mesh as the limit of a recursive process of subdividing each polygonal face into smaller faces that better approximate the smooth surface.

![Figure 2.3: First three steps of Catmull–Clark subdivision of a cube with subdivision surface below.](http://commons.wikimedia.org/wiki)

**Advantages:** Subdivision surfaces are easy to implement and the continuity of the surface can be controlled locally.

**Disadvantages:** Although subdivision surfaces have been known for nearly fifteen years, their use has been hindered by the lack of a closed form, that is they are defined only as the limit of an infinite procedure.
2.3.2 Photograph and digitize

In this modeling technique, facial models are created by digitizing live subjects or physical models. 2D photographs of the subject are taken and then be reconstructed into 3D models by using a digitizer; tracking points or grids that are marked on the photograph.

Figure 2.4: Photograph and Digitize (Source: *Lecture notes: CSE169: Computer Animation UCSD Winter 2005*)

**Advantages:** Digitizing models are very much used for many applications such as ones that require customs facial models pertaining to a particular subject or person.

**Disadvantages:** The data generated by the digitizer are usually too high resolution and may not be suitable for animation.
2.3.3 Sculpt and digitize (with a 3D digitizer)

Uses the same technique as the above except the models are 3D prototypes (sculptures) created with traditional sculpting tools rather than 2D photographs. Once the sculpture is ready, a grid is painted on it's surface. This grid is important to create a 3D representation of the prototype. Translation from the physical model to the 3D model is done by a 3D digitizing device.

Figure 2.5: Sculpting and digitizing process

**Advantages:** This technique allows the modeler the freedom to create the look that they want.

**Disadvantages:** Digitizing an object is a time consuming process. It requires the creation of curves on the model that can be then digitized. Then after control points are identified on the curve, the digitization process can start.
2.3.4 Scanning (laser)

As mentioned above creating each curve manually takes a lot of time. This time can be saved drastically by using a 3D laser scanning devise such as the Microscribe 3D Laser Systems [13]. The downsides of these devises are that they are extremely expensive.

2.4 Facial Animation Techniques

As stated by Ekman [1], humans are highly sensitive to visual messages communicated by the face. Consequently, facial animation requires specific algorithms able to render with a high degree of realism the natural characteristics of the motion. Research on basic facial animation and modeling has been extensively studied (for the past 40 years) and several models have been proposed [14, 15, 16, 17].

There are two major approaches to facial modeling and animation, those based on geometric manipulations and those based on image manipulations.
Figure 2.7: Classification of Facial Modeling and Animation techniques. Source: Survey of Facial Modeling and Animations Techniques. [14]

Geometric animation is based on deformations on the model of the face. Examples include, key-framing and geometric interpolations, parameterizations, finite element methods, muscle based modeling, simulation using pseudo muscles, spline models and free-form deformations. Image manipulation on the other hand uses a collection of example images captured of a human subject to create the model. Examples include, image morphing between photographic images, texture manipulations, image blending and vascular expressions.

Figure 2.7 shows the various facial animation techniques under these categories. The following paragraphs discuss some of the most commonly used techniques for facial animation.
2.4.1 Key-frame Interpolation

Key-framing is the most widely used animation for creating expressions. It is one of the oldest yet still useful and popular methods. In this method the animator creates key poses such as the start and end poses of a model and the animation system interpolates, or "tweens" the "in between" frames of these key frame data. Further the smoothness of the animation can vary depending on the interpolating technique being employed (discussed later in the sub sections).

For example when animating a falling ball (Figure 2.8), one key frame might be of the ball in rest position, the next key frame may be the ball in mid-air, and the key frame after that would be the ball touching the ground (again). All of the in-between frames may then calculated by the animation software automatically, giving a smooth rise and fall of the ball, making actual process of animation a matter of creating a few key frames.

There are many categories of key-framing technology in use but the underlying concepts are the same.

- Morphing
- Morph Targets
- Multi-Target Blending
- Vertex Blending
- Geometry Interpolation.

The main attraction of key-framing is that it allows the user to have control of the animation through the provision of key-frames [18]. Therefore it is still not surprising that key-framing is still used in movie industry - the principle used to bring characters like Mickey Mouse and Bugs Bunny to life.

The drawbacks however are that it tends to require a lot of setup time since each key frame requires complete specification of the model geometry and not just the areas of the model that are expected to change. Sometimes the animation may look unrealistic due to the interpolating function producing sharp changes (rises/downs) in vertex position values. Due to this the viewer may experience some "jerkiness' in the animation sequence. The animator may remedy this by using different interpolation technique(s) or by simply increasing the number of "in-between" frames between key frames, however, different interpolation methods results in different properties in the motion of the animated character. Some interpolation methods cause acceleration in the motion, while linear interpolation results in a constant motion. Therefore, it is important to select a method that offers the best or rather optimized result. This thesis implements two of the most common interpolation methods - the linear interpolation and the quadratic interpolation, hence they are discussed next.

2.4.1.1 Linear Interpolation

Linear interpolation is the simplest method of getting values at positions in between the data points. The points are simply joined by straight line segments. Each segment is bounded by two data points and the points can be interpolated in-between.
In this thesis the linear interpolation formula used is given as follows:

Let \((x_1, y_1)\) and \((x_2, y_2)\) be two points bounding a straight line. Then \((x_{\text{new}}, y_{\text{new}})\) can be calculated as:

\[
x_{\text{new}} = x_1(1 - \mu) + x_2\mu
\]

(2.1)

The parameter \(\mu\) defines where to estimate the value on the interpolated line, it is 0 at the first point and 1 and the second point. For interpolated values between the two points \(\mu\) ranges between 0 and 1. Simply controlling the value of \(\mu\) allows the animator to vary the number of “in-between” points to generate. \(y_{\text{new}}\) can be calculated by simply inserting \(y_1\) and \(y_2\) values the formula. Figure 2.9 illustrates this formula.

This formula is then implemented as a function in C++:
double LinearInterpolate(double x1, double x2, double mu)
{
    return (x1 * (1 - mu) + x2 * mu);
}

2.4.1.2 Cosine Interpolation

Though linear Interpolation is very easy to implement it has a serious drawback; that is it creates discontinuities (non-differentiable) at key points (illustrated in Figure 2.9). A cosine interpolation produces a smoother transition between key points. Instead of using numeric constants for $\mu$ as in linear interpolation, in cosine interpolation $\mu$ needs to be adjusted as follows using a function of cos.

$$\mu_{new} = \frac{1 - \cos(\pi \mu)}{2}$$

(2.2)
The cosine interpolation formula can then be implemented as a function in C++ as:

```cpp
double CosineInterpolate(double x1, double x2, double mu)
{
    mu = (1 - cos (3.1415 * mu))/2.0;
    return (x1* (1 - mu)+ x2 * mu);
}
```

### 2.4.2 Parameterized Systems

The first parameterized 3D facial animation system was created by Frederic Parke in 1972 [4]. This allowed him to create facial expressions by changing a set of control parameters. Unlike interpolation techniques where the entire facial data set is interpolated from one key frame to another, parameterization allows control of specific parts of the model to generate expressions. Parameterization technique works by tying specific parts of the face geometry to control parameters. The animator can then create different facial expressions by adjusting these parameter values. This allowed creation of a large range of facial expressions, however there were some serious drawbacks as reported by Waters in [2] given as follows:

1. He pointed out that in cases where parameters were affecting the same vertices often led to noticeable motion boundaries.

2. The complete generic parameterization was not possible as the parameter values were tied to particular facial mesh topology. If the animator wished to change to a different facial model, the parameter set values would require adjustments to fit the new mesh topology.

3. Setting the parameter values and fine tuning them to produce realistic motions often proved tiring.
Nonetheless, parameter based systems proved useful in animations where control for specific areas of the facial topology was required.

The most recent technology that uses the parameterization like technique for facial animation is in the MPEG-4 standard. MPEG-4 specification specifies facial animation parameters, each corresponding to a particular facial action deforming a face. The MPEG technique requires settings of the feature points on the static 3D model which defines the regions of deformation on the face, and generation and interpretation of parameters that will modify those feature points in order to create the actual animation. A good explanation on MPEG-4 encoding scheme is given in [19].

### 2.4.3 Muscle Based Systems

Platt and Badler [8] in early 1980's developed the first muscle action based models. These models were the first to make use of the FACS$^2$ as the basis for facial expression control. The late 1980's saw the development of a series of physically based pseudo-muscle driven facial models, example [2, 20] and the development of an abstract muscle action model by Magnenat-Thalmann and colleagues [7].

Physically based models attempt to model the shape changes of the face by modeling the properties of facial tissue and muscle actions. These models often use subsets of the FACS system to specify the muscle actions. However no model to date includes the complete set of FACS AUs or graphically model specific AUs.

There have been several muscle-based systems developed over the years but the one that has been selected, examined and used in this thesis is the Waters muscle

---

$^2$ Facial Action Coding System (FACS) - discussed in detail in chapter 4.
based model [2]. The Waters model consists of a facial topology (a triangular mesh depicting skin) with underlying muscles. These muscles are of two types, linear muscles that pulls vertices under its region of influence towards the point of attachment on the skin; and a sphincter muscle around the mouth that pulls the vertices under it’s region of influence towards the center of an ellipse. Another type of muscle is proposed by Waters – the sheet muscle depicting the frontalis (forehead and cheek muscles) but lacks implementation in source code provided in [4]. However this thesis shows an implementation of the sheet muscles and well as modifications on the sphincter muscle formula. A comprehensive discussion on Waters model is given in Chapter 4.

2.4.4 Motion Capture

Motion Capture technique achieves its animation by recording a live performance, usually done by actors. The actors are placed in some special suit containing sensors that are susceptible to movements and thus record the movements of the limbs as they move. The data from the sensors are then applied to the model by using special 3D software to create animations. Another
technology used in motion capture is the use of special cameras that detect motions by tracking reflective like stickers that are placed on the actor’s body. Motion capture technology allows creation of animations that otherwise difficult to generate by conventional animation techniques discussed above. More so motion capture can also be applied to non-human character models.

There are however a couple of downsides to Motion capture technology – most notably its cost which leads to only the rich movie making studios to acquire them. Also motion capture data has to be fine tuned at times to make the animation look realistic.

There are several movies where motion capture technology has been used, most notably in the movies Beowulf (2007), The Polar Express (2004) and Monster House (2006) just to name a few.

### 2.5 Summary

This chapter presents a summary of the various modeling and animation methods developed over the years, summarizing the theoretical approaches, their strength, weaknesses and performance. Since this thesis belongs in the area of computer graphics and specifically 3D animation, the focus lies on geometry model representations.

Finally the Waters model was chosen for running the simulations of this research based on the following reasons:

1. It employs a geometry based model using polygonal surfaces to construct the face. Hence facial data construction data was already available. These facial data (available as 3D vertex coordinates) were easy to render on the
computer’s graphics engine. Further animation could be easily be
generated by simply interpolating these coordinates.

2. The model describes deformation of the face geometry due to facial
muscle action. This is exactly how in real facial expressions are generated.

3. The model already had descriptions of three types of muscles; linear,
sphincter and sheet muscles. In our research we simply had to add and
modify the implementation of some of them, while using the rest as they
were in the original implementation.

4. The usage of a muscle based system was crucial as the Facial Action
Coding System (FACS) used in this research defines facial expressions as
a function of muscle actions.

5. The availability of a simple software implementation of the Waters model
which we have enhanced to for our simulations.
3. Facial Action Coding System

3.1 Introduction

A widely accepted theoretical foundation for the control of facial animation is the Facial Action Coding System (FACS). FACS is based on an anatomical analysis of facial movements and has been developed by psychologists P. Ekman and W.E. Frieson [1]. In 2002, a new version of FACS was published, with large contributions by Joseph Hager [21].

It was designed primarily to measure facial movement relevant to emotion. The aim was to develop a comprehensive system which could distinguish all possible identifiable facial movements. Since every facial movement is the result of muscular actions on the face, a system could be obtained by observing how each muscle of the face acts to change appearance of the face, thus allowing a way to measure different facial expressions in terms of individual muscle actions. The heart of FACS is the classification of Action Units (AUs) which is mapped to a finite set of possible basic actions performable on a human face in terms of the smallest visible unit of muscular activity. Hence in order to understand FACS one must have some knowledge on the anatomy of the human face.

In this thesis, FACS is used to generate facial expressions based on the AUs on the 3D (modified) Waters face model of a human face. There is numerous research in which FACS has been used to measure or validate the facial
expressions created [9, 10, 22, 23]. However little has been done to actually
develop a system that will allow users the flexibility to adjust the facial model
based on individual FACS action unit muscle activations. This missing bit is taken
care of in our research which allows recordings of individual FACS Action Units
and then produces expressions based on the summation of one or more action
units.

In the following sections we focus on describing the human facial muscles. This
discussion is important for understanding FACS. Following this is the discussion
on FACS Action Units. The advantages and disadvantages of using FACS for
facial animation are also discussed as well as latest research in facial animation
using FACS.

3.2 The Anatomy of Human Face

The human head mainly consists of bone, of which the only movable part is the
jaw, skin and muscles. Any facial expression results from the movements by a
combination or interaction of any of the 268 muscles of the face. These muscles
on the face can be categorized into three types.

- Linear muscles
- Sheet muscles
- Sphincter muscles

Mixtures of these muscle types coordinate to produce each facial expression.

**Linear muscles**: Work in one direction. The contraction of the muscle pulls
the facial tissue towards the point of attachment which is usually the bone.
Examples include the corrugator, buccinator and labii muscles to name a few.
**Sheet muscles**: Behaves like a linear muscle but are more flatter and wider and their attachment area is broader. Example the frontalis muscle on the forehead.

**Sphincter muscles**: Usually surround body openings, for example the area around the mouth and eyes. They are ring or elliptic shaped and will enlarge or reduce the opening depending on whether the muscle is relaxed or contracted. Examples are the orbicularis oris and the orbicularis oculi.

![Figure 3.1: The anatomy of muscles on a human face [24].](image-url)
3.3 FACS Action Units (AUs)

Ekman and Frieson [1] discovered that there are only a finite set of the possible basic actions performable on a human face in terms of the smallest visible unit of muscular activity. These smallest visible units are called Action Units (AUs), and each Action Unit is referred to by a numerical code. There are 32 single AUs defined in FACS; meaning these Action Units are generated by an underlying muscle action. Figure 3.1 shows some of the muscles influenced by AUs.

These independent AUs interact in several different ways to create facial expressions. For example, combining AU1 (inner brow riser), AU6 (Cheek Raiser), AU12 (Lip Corner Puller), AU14 (Dimpler) creates a happy expression [1]. Table 3.1 lists the names, numbers and anatomical basis of some sample Action Units. It is these AUs that are implemented in this thesis. The map to which muscle the AU influences is also shown. A complete list of single AUs can be obtained from [25].

![Figure 3.2: Six primary expressions. Source: [26]](image-url)
Having looked at approximately 55000 photos of different facial expressions, Ekman and Frieson [1] decided upon 6 primary expressions. Table 3.2 lists these 6 primary expressions with the corresponding combination of AUs required to produce a given expression. The FACS name given in the tables are shorthand, not meant to describe the appearance changes but a convenient way to remember them. Figure 3.2 shows the states of these 6 primary expressions.

<table>
<thead>
<tr>
<th>Au</th>
<th>FACS Name</th>
<th>Muscle Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Inner Brow Raiser</td>
<td>Frontalis</td>
</tr>
<tr>
<td>2</td>
<td>Outer Brow Raiser</td>
<td>Frontalis</td>
</tr>
<tr>
<td>4</td>
<td>Brow Lowerer</td>
<td>Corrugator supercilii, Depressor supercilii</td>
</tr>
<tr>
<td>5</td>
<td>Upper Lid Raiser</td>
<td>Levator palpebrae superioris</td>
</tr>
<tr>
<td>6</td>
<td>Cheek Raiser</td>
<td>Orbicularis oculi</td>
</tr>
<tr>
<td>7</td>
<td>Lid Tightener</td>
<td>Orbicularis oculi</td>
</tr>
<tr>
<td>9</td>
<td>Nose Wrinkler</td>
<td>Levator labii superioris alaeque nasi</td>
</tr>
<tr>
<td>10</td>
<td>Upper Lip Raiser</td>
<td>Levator labii superioris</td>
</tr>
<tr>
<td>12</td>
<td>Lip Corner Puller</td>
<td>Zygomaticus major</td>
</tr>
<tr>
<td>13</td>
<td>Cheek Puffer</td>
<td>Levator anguli oris</td>
</tr>
<tr>
<td>14</td>
<td>Dimpler</td>
<td>Buccinator</td>
</tr>
<tr>
<td>15</td>
<td>Lip Corner Depressor</td>
<td>Depressor anguli oris</td>
</tr>
<tr>
<td>16</td>
<td>Lower Lip Depressor</td>
<td>Depressor labii inferioris</td>
</tr>
<tr>
<td>17</td>
<td>Chin Raiser</td>
<td>Mentalis</td>
</tr>
<tr>
<td>20</td>
<td>Lip stretcher</td>
<td>Risorius</td>
</tr>
<tr>
<td>23</td>
<td>Lip Tightener</td>
<td>Orbicularis oris</td>
</tr>
<tr>
<td>26</td>
<td>Jaw Drop</td>
<td>Masseter</td>
</tr>
</tbody>
</table>

Table 3.1: Sample single Action Units. Source [14].
<table>
<thead>
<tr>
<th>Basic Expression</th>
<th>Involved Action Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surprise</td>
<td>AU1, 2, 5, 15, 16, 20, 26</td>
</tr>
<tr>
<td>Fear</td>
<td>AU1, 2, 4, 5, 15, 20, 26</td>
</tr>
<tr>
<td>Disgust</td>
<td>AU2, 4, 9, 15, 17</td>
</tr>
<tr>
<td>Anger</td>
<td>AU2, 4, 7, 9, 10, 20, 26</td>
</tr>
<tr>
<td>Happiness</td>
<td>AU1, 6, 12, 14</td>
</tr>
<tr>
<td>Sadness</td>
<td>AU1, 4, 15, 23</td>
</tr>
</tbody>
</table>

Table 3.2: Example set of AUs for basic expressions. *Source [14].*

Rotenberg [26] in his lecture notes showed an interesting way to determine other expressions (apart from the 6 universal ones) by graphing the expressions on a 2D-axis system where:

- x-axis represents *sadness to happiness* state (-ve to +ve)
- y-axis represents *relaxed to excited* state (-ve to +ve)

This is illustrated in figure:

![Emotional axes](source)

*Figure 3.3: Emotional axes. Source [26]*
The FACS coding manual [21] gives a detailed description of all the appearance changes occurring with a given Action Unit. This description lists the parts of the face that have moved and the direction of their movements, the wrinkles that have appeared or have deepened, and the alterations in the shape of the facial parts.

### 3.3.1 FACS Action Descriptors

In addition to single AUs, FACS also defines a set of Action Descriptors that differ from single AUs in the sense that they do not have a specified muscular basis for the action they represent. Some of these action descriptors are listed in Table 3.3. For a complete list of Action Descriptors the reader is directed to [25].

<table>
<thead>
<tr>
<th>AU</th>
<th>FACS Description Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>Head turn left</td>
</tr>
<tr>
<td>52</td>
<td>Head turn right</td>
</tr>
<tr>
<td>53</td>
<td>Head up</td>
</tr>
<tr>
<td>54</td>
<td>Head down</td>
</tr>
<tr>
<td>55</td>
<td>Head tilt left</td>
</tr>
<tr>
<td>56</td>
<td>Head tilt right</td>
</tr>
</tbody>
</table>

Table 3.3: Example FACS action descriptors. Source: [http://en.wikipedia.org/wiki/Facial_Action_Coding_System](http://en.wikipedia.org/wiki/Facial_Action_Coding_System)

### 3.4 FACS-based Facial Animation

The study of FACS falls in the area of psychology, initially used by psychologists to identify and quantify different emotional states of the human face according to muscle actions. However in the past few decades it has proven useful to computer animators as well. There are already numerous facial models that use FACS as the basis for controlling the face deformation parameters [2, 8, 27]. In
all these systems, the AUs are either pre-programmed or automatically translated into the defined parameter sets controlling muscles or directly affecting the vertices to create facial animations. Hence for implementation on a computer, FACS is very suitable since each facial expression can be described as a list of the names of the AUs involved.

In this thesis, our face animator is based on Waters Muscle model [2, 4] which uses FACS to control the parameters defining the activations of the muscles on the face to generate expressions. The source code for Waters model implements only the 6 primary expressions as discussed above. We have extended the program to include all FACS AUs given in Table 3.1. *(a complete description of our software implementation is discussed in Chapter 5).*

There are however some notable drawbacks of FACS. Firstly, it is not suitable to describe changes to the face that do not involve movements, like blushing. Secondly excluded from FACS are: facial sweating, tears, rashes, pimples and permanent facial characteristics. And lastly FACS’s poor information on the degree of activations that must be applied to individual AUs to produce an expression. For example, to create the expression “Sadness”, the following combination of AUs is used – AU1, 4, 15, 23.

Following this procedure apparently gives rise to the following questions:

1. **How do we deal with the degree of activation for the various AUs to produce an expression?**

For instance to create the expression “Sadness”, is AU1 – is Inner Brow Raiser - activated to its maximum or partially? Keeping the same combination of AUs (1, 4, 15 and 23) to produce “Sadness”, varying the degree of activation of these AUs can produce expressions that can be crudely classified as “very sad”, “little sad”
or result in an expression that may no longer appear to be “Sad”. FACS, therefore has this limitation that it does not provide for an analysis on the degree of activation on AUs to produce facial expressions. What this simply meant that if the degree of activations need to taken into consideration, this would lead to a huge number of individual facial expression.

FACS solves this problem crudely by introducing three additional categories of AU intensity called *low*, *medium* and *high*. They are denoted by appending to AUs number one of the three letters *x*, *y* or *z* respectively. Example AU1x, AU1y, AU4z etc.

2. **How do we deal with cases when our face animator is presented with a combination of AUs that in theory does not compose to a valid expression?**

For example, given the following AU combination AU51, 52 [AU51 Head turn left and AU52 Head turn right] can not occur at the same time, hence does not constitute to a valid expression.

3. **How to deal with cases where one or more AUs influence the same area of the face geometry?**

That is changes which appear on the face while activating several AUs, can differ a lot from changes induced by AUs applied individually [9].

Ekman and Frieson [1] did realize the above two problems and devised 5 ways in which AUs can co-occur or influence each other.
3.4.1 Co-occurrence Rules (AU combination types)

**Rule 1: Additive**

Given a set of AUs, the changes applied to the face are just the sum of all changes caused by each AU scored separately. Additive combination is applied usually when AUs involved influence different areas of the face, example (AU1 and AU26). In this case the order the AUs are applied to generate animation does not matter.

**Rule 2: Dominance**

In this rule one AU can dominate the other. That is the dominant AU can cancel the effect of the submissive AU or can shadow the appearance changes of the submissive AU. In these kinds of scenario both AUs are scored and may be applied in animation, but the appearance changes caused by the submissive AU may be difficult to detect.

**Rule 3: Alternative**

The alternative rule states the non-possibility scoring of a combination of AUs at the same time. In such cases the AU that best fits the given scenarios is chosen.

**Rule 4: Substitution**

In this case, a combination of AUs can be replaced by another single AU if their appearance changes fit that of the single AU. For example the combination of AU13 + AU14 can be scored by AU12 alone.
Rule 5: Different

All combinations of AUs that do not belong to the above are classified as different combinations. Each AU that falls in this category inflicts distinctive appearance changes, changes that are not scored by other AUs.

Wojdel in [9] introduced two more co-occurrence rules.

Exclusion: Described to be similar to the dominance rule except that this rule prohibits the scoring of the submissive AU. For example AU18 dominates AU28. However in cases where the activation of AU18 is minute, only then AU28 can be scored or applied in animation.

Opposition: This rule deals with cases where the occurrence of two AUs simultaneously is practically not possible. For example the relationship between AU51 and AU52 (these AUs describe two opposite movements of the head).

In summary the above examples are further explained in the following figure.

Figure 3.4: Dependencies between Action Units [9].
3.5 Summary

This chapter presents a discussion on what Facial Action Coding System (FACS) is and its suitability and accuracy in creating facial expressions. Also discussed is the co-occurrence rules developed by Wojdel [9] that proves important in using in our facial animation system in order to avoid creation of unrealistic facial expressions. Using FACS therefore allows our muscle based animation system to generate expressions that are artificially posed by selecting the desired action units.
Chapter 4

4. The Reference Face Model

4.1 Introduction

Modeling and animating a human head is a non-trivial task. The human face is a complex architecture and varies in appearance from one individual to another. It has the capability of creating myriads of expressions (emotional states) on the fly. The ease at which a human face changes expression has always posed a challenge to researchers and animators of the day to mimic the same life like expressions into their computer generated models for animation.

Several techniques in facial modeling and animation have been developed over the years, some of which has already been discussed in Chapter 2. One such technique is using pseudo-muscles to deform facial geometry. The foundation of this thesis is to test the application of FACS to generate facial animations. Since FACS AUs describe distinct deformations on the face caused by muscle actions on the same area, a muscle based facial model would prove ideal to use. Hence in this thesis we use the original Waters Muscle Model, published in [2].

This chapter is therefore dedicated to describing Waters Muscle models, some of its drawbacks, enhancements done to overcome these drawbacks as well as introducing new features on Waters Face Model to enhance animation.
4.2 Waters Muscle Model

Waters model, first reported in 1987 [2] is one of the most popular models used in research and animation. It is based on facial anatomy which describes that facial expressions are created by relaxation and tightening of facial muscles. In [2] Waters modeled two types of muscles: vector muscles that forms majority of the facial muscles and sphincter muscles for the *Orbicularis Oris* - muscles around the mouth. The third muscle type called Sheet muscles is described in detail in [10]. The anatomical description of these categories of muscles has already been discussed in Chapter 3. Hence in the following sections we look at the mathematics behind these pseudo-muscles. The equations we use here are drawn from [22, 28] which is based on the original Waters model publication, however in a simplified form for understanding.

4.2.1 Linear Muscle Model

Waters models a linear muscle as a vector hence also called a vector muscle. Anatomically described, the linear muscle is made up of two points, one that is attached to the bone and the other to skin. When contracted the muscles pull towards the attachment to the bone causing skin deformation on the surface. The muscle is modeled as a vector from \( v_2 \) to \( v_1 \) where \( v_1 \) defines the point of attachment to the bone and \( v_2 \) the point of insertion giving the impression that the muscle is attached to the skin. When the muscle is contracted, its effect is to pull the surface from the area of the muscle insertion point to the muscle attachment point. Figure 4.1 illustrates the vector muscle with the following definitions.

\[ v_1 : \text{Attachment point of the vector } v_1v_2 \text{ to bone (tail).} \]

\[ v_2 : \text{Insertion point of vector muscle } v_1v_2 \text{ to skin (head).} \]
$P$: An arbitrary facial skin point.

$p'$: The new position of the facial skin point after displacement.

$R_s, R_f$: Fall-off radius start and finish respectively (indicates the radius of the segment – defining the area of influence)

$p_r, p_s$: Defines the boundary points for the vector muscle’s ($\nu_1 \nu_2$) area of influence.

$\Omega$: The maximum angle of influence of $\nu_1 \nu_2$.

$\mu$: Angle between the vector $\nu_1 \nu_2$ and $p'$.

$D$: Distance between the vector $\nu_1 \nu_2$ attachment point and the arbitrary skin point.

Figure 4.1: Linear Muscle Model
Given any point $P(x, y)$ located at a mesh node, within the segment $v_i p_r p_s$ is displaced towards $v_i$ along the vector $Pv_i$ resulting in a new $P'(x', y')$ hence:

$$p' = P + AKR \frac{Pv_i}{|Pv_i|}$$

(3.1)

where $K$ is the muscle spring constant and $A$ and $R$ are calculated as:

$$A = \cos(\mu)$$

(3.2)

$$D = |Pv_i|$$

(3.3)

$$R = \begin{cases} \cos \left(1 - \frac{D}{R_s}\right), & \text{for } P \text{ inside region } (v_i p_r p_s) \\ \cos \left(\frac{D - R_s}{R_f - R_s}\right), & \text{for } P \text{ inside region } (p_r p_s p_m) \end{cases}$$

(3.4)

Figure 4.2: Linear Muscle Contraction with $\Omega = 35.0$, $R_s = 7.0$, $R_f = 14.0$, $K = 0.75$, elasticity = 1.0. Source [2].
4.2.2 Sphincter Muscle Model

The sphincter muscle is modeled by Waters in an elliptical shape and can be simplified to a parametric ellipsoid. Vertices within the ellipsoid are drawn towards the center of the ellipsoid (like a drawstring bag) with maximum movement depended on how far the vertex position is from the center of the ellipsoid. Figure 4.3 illustrates the sphincter muscle in the $x,y$ plane with the following definitions.

$P$: An arbitrary facial skin point.

$c$: Epicenter of the sphincter muscle influence area.

$l_x$: The semi-major axis of the sphincter muscle influence area.

$l_y$: The semi-minor axis of the sphincter muscle influence area.

The following equation therefore can be used to compute the new position $P'$ of the vertices at an arbitrary point $P$ in the $x,y$ plan within the ellipsoid.

$$P' = P + KD \frac{P_c}{|P_c|}$$

(3.5)

where $K$ is the muscle spring constant and $D$ is calculated as:
\[ D = \begin{cases} \frac{f}{g} \text{ for } |P_c| > l_y \\ 0 \text{ for } |P_c| \leq l_y \end{cases} \]  

(3.6)

where \( f \) is the coefficient of the vertex displacement for the sphincter muscle in the \( x - y \) plan, and \( g \) is the proportional distance of the vertex position from the center of the ellipse, i.e.,

\[ f = 1 - \frac{\sqrt{l_y^2 P_x^2 + l_x^2 P_y^2}}{l_x l_y} \]  

(3.7)

\[ g = \frac{|P_c|}{l_x} \]  

(3.8)

The calculation of \( D \) above protects the central area of the ellipse (bounded by a circle with radius equal to \( l_y \)) from vertices from piling up on top of each other towards the center of the ellipse by muscle contractions.

The above equations simulate the working of the sphincter model in 2D. In real lips, the skin protrudes (bulges) towards the center of the lips, hence
displacement in the z direction is needed 3D. This can be achieved by calculating for \( p_z \) as:

\[
p'_z = p_z + \frac{K(1 - fg)}{Z}
\]

(3.9)

where \( Z \) is a constant greater than 0 used to control the peak of the protrusion. Implementation of the sphincter muscles around the mouth is shown in Figure 4.5. The example expression is created with \( K = 0.3 \) and \( Z = 10 \).

![Figure 4.5: Application of the sphincter muscle on the area around the mouth.](image)

In this research we modify the original Waters program to use three sphincter muscles; one each around the eyes and one covering the mouth opening area. The sphincter muscle used around the mouth area is derived from Waters original sphincter muscle equation where the vertices converge towards the center of the muscle. However the orbicularis oris muscle that surrounds the eye behaves a bit differently. It does not have the ability to pull the corners of the eyelids towards the center of the eye. Instead it is only able to pull the upper and lower eyelids towards each other. Hence to use the use the above sphincter equation for the
eyes we had to modify it so that the vertices converge towards the semi-major axis of the ellipse defining the influence area around the eyes. This gives the model the ability to “squint” eyes.

Figure 4.6: Vertices converging to the semi-major axis

Figure 4.6 illustrates our eye sphincter muscle in the $xy$ plan with the same definitions as for the normal sphincter muscle. We note here that the displacement is along the $y$-axis.

Figure 4.7: Sphincter muscle diagram for use around the eyes.

The new equation to compute the new position $p'$ of the vertices at an arbitrary point $P$ in the $x-y$ plan within the ellipsoid in figure 4.6 is.

$$ p' = P + KD \frac{Px}{|Px|} $$

(3.10)
where $K$ is the muscle spring constant and $D$ is calculated as:

$$D = fg$$  \hspace{1cm} (3.11)

Here, $f$ is the coefficient of the vertex displacement for the sphincter muscle along the $y-axis$ plan and $g$ is the proportional distance of the vertex position perpendicular to the $x-axis$, i.e.,

$$f = 1 - \frac{\sqrt{\|y^2 P^2_y\}}{\|y\}}$$  \hspace{1cm} (3.12)

$$g = \frac{|P_x|}{\|y\}}$$  \hspace{1cm} (3.13)

For displacement in the $x-axis$ direction, the same formula as equation (3.9) is re-used, where $Z$ is a constant greater than 0. A small value for $Z$ causes the eyelids to bulge out hence in our implementation we have set this constant to 30, which achieves good results.

![Image of a face with muscles around the eyes](image)

Figure 4.8: Using sphincter muscles around the eyes. $K = 0.2$ for left eye (to the reader) and $K = 0.4$ for the right eye. $Z = 30$.  

47
4.2.3 Sheet Muscle Model

Unlike the linear and sphincter muscle, sheet muscles are made up of a series of almost-parallel fibers spread over a rectangular area. The displacement of vertices therefore is towards one end of the rectangle with the other end of the rectangle attached to the bone with maximum displacement towards the attachment point of the sheet muscle to the bone. In anatomy, sheet muscles models muscles such as the frontalis which is located on the forehead on a human face. Waters in his paper [2] does not have a description of sheet muscles hence we use the description from [10] in this thesis. Waters in his face program used linear muscles for the forehead. However we will use Waters linear muscles combined with two sheet muscles (one on each side of the forehead) to achieve the desired effect in our effort to produce realism in the expressions created.

Figure 4.9 illustrates the sheet muscle with the following definitions:

\[ P \]: An arbitrary facial skin point.
\[ A_1, A_2 \]: Boundary points defining the attachment line of sheet muscle to bone.
\[ I_1, I_2 \]: Boundary points defining the insertion line of sheet muscle to skin.
\[ A_c \]: Middle point of sheet muscle attachment line.
\[ I_c \]: Middle point of sheet muscle insertion line.
\[ L, W \]: Length and width of the rectangular defining the zone of influence of the sheet muscle respectively.
\[ I_l \]: Distance between arbitrary skin point P and the attachment line.
Figure 4.9: Sheet Muscle Model

The new position $p'$ of the vertices at an arbitrary point $P$ in the $x, y, z$ plane within the rectangular zone of influence can be calculated as:

$$p' = KD \frac{A_i l_c}{A_i l_c}$$  \hspace{1cm} (3.14)

where $i$ indicates the values of the $x, y, z$ coordinates of the vertex. $P$ and $K$ is the muscle spring constant and $D$ is calculated as:

$$D = \begin{cases} 
    \cos((1 - \lambda^n) \frac{\pi}{2}) & \text{for } 0 \leq \lambda \leq 1 \\
    \cos((\frac{\lambda^n}{\delta^n} - 1) \frac{\pi}{2}) & \text{for } 1 < \lambda \leq \delta 
\end{cases}$$  \hspace{1cm} (3.15)

where $n$ is the sheet muscle strength factor and set at 1 in our implementation. $\lambda$, and $\delta$ are calculated as:
\[ \lambda_i = \frac{q}{A_l} \]  

(3.16)

\[ \delta = \frac{L}{A_l} \]  

(3.17)

The relationship between the rectangular sheet muscle and the arbitrary point \( P \) is shown in the following figure.

![Diagram](image)

Figure 4.10: Relationship between \( P \) and the sheet muscle for calculation of \( q \).

Therefore \( q \) can be calculated using the following:

\[ q = \frac{|AP| \cos \Omega}{180 / \pi} \]  

(3.18)

where \( \Omega = \arccos \left( \frac{AP \cdot AB}{|AP| |AB|} \right) \)  

(3.19)
4.2.4 Jaw Rotation

In order to create more convincing expressions it is important for the facial model to allow for jaw rotations. The human jaw is made up of two parts namely the upper and lower jaw. The upper jaw remains fixed while it is the lower jaw that moves. Waters models the moving lower jaw relative to the pivot point of the facial model mesh rather than employing any muscle technique. In his model the jaw contains vertices of the lower part of the face which is rotated along the $y-axis$ and $z-axis$. Hence this rotation can be explained using the following definitions and equation.

$P$: Vertex forming the lower jaw.

$K$: Angle of rotation of the jaw vertex relation to the root of the facial mesh and is a value between 0 and 4 with 4 indicating the maximum opening.

$$p_y = \cos\left(\frac{K\pi}{180}\right)P_y + \sin\left(\frac{K\pi}{180}\right)P_z \quad \text{for rotation along } y-axis \quad (3.20)$$
This formula is applied on all vertices that form the jaw. The animator therefore has to specify manually in advance which vertices would be part of the jaw of the face. Adjusting the value of $K$, simulates jaw opening and closing with maximum rotation produced at the center of the lower lip. This however creates discontinuity at the corners of the lips as reported by [10].

Figure 4.12: Discontinuity at the lip corners. [4]

To keep the cohesion at the corner of the lips we have given each vertex forming the moveable part of the lower jaw a weight value depending on its proximity to the center of the lips. Since the formula above created maximum rotation at the center of the lip, all vertices of the lower jaw around the center are allocated a higher weight value with decreasing weights towards the corner of the lips. Hence the corner points of the lips are allocated a weight of zero meaning no rotation.

Figure 4.13 illustrates the allocated of weights to vertices forming the lower jaw. After $p'$ has been determined from the equation (3.20) and (3.21), it is fed into
the following two equations to determine the final position of vertex \( P \). Here \( w \) is the weight at \( P \).

\[
p_y' = p_y' w + P_y (1-w) \quad \text{for rotation along } y \text{-axis} \tag{3.22}
\]

\[
p_z' = p_z' w + P_z (1-w) \quad \text{for rotation along } z \text{-axis} \tag{3.23}
\]

Figure 4.13: Weights allocated to vertices forming the lower part of the jaw.

4.3 Suitability of Waters Muscle Model for Facial Animation

Unlike techniques where producing animation requires direct manipulation of the facial mesh, Waters muscles are independent of the facial mesh. This means that the animator has the freedom to change the facial mesh yet still achieve the same
effect as in using the original mesh. This allows flexibility in using different character mesh topologies. It allows for compact representation of facial geometry where the only information needed are the muscle parameter values and the skin mesh data in order to produce expressions. The muscle functions work on triangulated vertices, which most graphics hardware are optimized to use, hence rendering is fast. Further expression parameters can control groups of muscles at once resulting in multiple muscle deformation applied to the facial mesh at once.

The problem areas include the difficulty of placing muscles on the geometry of the face which obviously has to be done manually. The muscle functions basically represent flat architectures hence they may not be attached to the skin (facial mesh) at all. Due to this, muscle curvature and volume is also not taken into account. At times during animation some mesh vertices can fall prey to influence from multiple muscle actions, though this problem can be solved by taking “average” influences of the muscle on the vertex.

4.4 Summary

Waters muscle models have been used and tested widely by researchers over the years [7, 10, 20, 22, 28]. Given the aim of this thesis to animate facial expressions governed by FACS, and that FACS basically describes expressions based on affected muscles on the face, Waters muscle model becomes the ideal choice for use.

The following chapter describes all the implemented work in this thesis project and their results.
Chapter 5

5. Experiments & Results

5.1 Introduction

This chapter describes all the implemented work in this project; the developments done and an analysis of their results. The results of extending the Waters muscle models particularly the sphincter muscle to use around the eyes; the application of sheet muscle to control the forehead vertices; and the trick in controlling the jaw rotation using weights so that discontinuities are eliminated from the lip corners is already discussed in chapter 4. These features all form part of the Facial Animation Player (FAP), which is the heart of our system. Our FAP is implemented in such a way that enables it to be used as a low-bandwidth teleconferencing player, a real time facial animation player driven only by FACS AUs transmitted as plain text over TCP sockets. In all these FACS will be used to validate the expressions generated.

5.2 System Overview

Our Facial Animation Player (FAP) was developed in three parts. The first part included work done on the facial model itself. Here, Waters initial program available free for download from [29], was used and enhanced by giving it an easy to use graphical user interface (GUI) to control the functionalities of the program. Also greater control over rendering and animation is provided. The
second part involved work done on the expression editor. Waters original linear muscle set used in the program is extended to include more muscles (in particular sphincter and sheet muscles) in order to create more interesting facial expression. The expression editor gives user the control over all muscles used on the face model including jaw rotation. Using this control the user can contract and relax muscles and can record expressions for each FACS AUs quite easily. The recordings for individual AUs are used by the Remote Player which is the third component developed. The remote player enables live video only streaming. It opens up a TCP socket and receives streaming data from a remote source. This streaming data is made up of series of AUs defining individual expressions. A small animation engine is also designed within this player. The information about expressions (that is AUs) is processed according to the co-occurrence rules (if chosen) stated in chapter 3 and then played over time creating animation on our 3D face model. Figure 5.1 illustrates the architecture of FAP.

5.3 Development Information

5.3.1.1 Platform

Our FAP is developed on a Windows XP platform using Visual Studio 6.0 as the development IDE. It runs on a DELL Optiplex 755 Workstation, 2194 MHZ with 1GB memory. The graphics display uses Intel Q35 Express Chipset hardware with max 384MB for graphics memory and DirectX\(^3\) version 9.0. Most

---

\(^3\) Collection of Windows APIs (Application Protocol Interface) for handling tasks related to game programming on the MS Windows OS. This set of development libraries for high performance games allows software developers "direct access" to the low-level functions of PC peripherals.
Figure 5.1: Architecture of our Facial Animation Player
recent windows PCs of the day are well resourced in terms of processor speed and memory. Hence the only other hardware system requirement would be the availability of a good graphics card capable of rendering 3D graphics.

5.3.1.2 Programming language and libraries
All programming in this project has been done using C++ and OpenGL API. OpenGL, since its introduction in 1992 has become industry’s most widely used and supported 2D and 3D graphics application programming interface (API). The beauty of OpenGL is its simplicity. It does not contain complex functions for creations of models. Instead all models in OpenGL have to be created using primitive shapes such as lines, triangles and quads. Animations itself is done by applying geometric transformations such as rotations and translations on the object. Since OpenGL architecture strives to be platform independent, no windowing functionality is provided. This problem is solved by using another OpenGL based API called GLUT. GLUT provides a programming interface with relatively simple methods for setting up window systems in OpenGL applications. To give a graphical user interface (GUI) to our program a third API is used called GLUI. GLUI is based on GLUT, offering common user controls. A comprehensive description of the OpenGL, GLUT and GLUI APIs are given in [30], [31] and [32] respectively. All these APIs are written in C++ hence easily integrated into our system.

Apart from the graphics libraries, our system is also POSIX thread libraries for the implementation of the Remote Player component of the system. We use threads in FAP to distribute the reading of the stream of AU numbers from the actual animation process. This allows FAP to do the animation while in the background a separate thread continues to read the AUs and feeds them to the animation engine. POSIX threads are discussed and explained on how they are used in the section on Remote Player Controls later on in this chapter.
5.3.2 Resources used as background

We have used Waters facial animation program, available free for download with source code from Waters Computer Facial Animation textbook website [29]. In particular we have adopted the “enhanced muscle-based” animation system from Appendix A additions from this website. This enhanced version follows from the original code that accompanied the first edition of Waters Computer Facial Animation text [27] which was derived from Waters paper on “A Muscle Model for Three-Dimensional Facial Animation” [2]. It is implemented in “C” using OpenGL and GLUT. However all new additions have been programmed in C++ instead. Both native C code and C++ was well supported by our Visual Studio IDE with no compilation issues.

The downloadable repository basically contains two categories of files, the data files (in plain text format) and source code files.

5.3.2.1 Data Files

The following basic input files accompany the original source code distribution. These files are essential in running of the program. Their descriptions are as follows:

- **faceline.dat** - face date file containing 3D coordinate values of indices
- **index.dat** - the polygon index file describing the pattern to join indices to form face wireframe.
- **muscle.dat** - the muscle data file containing description of all linear muscles used in the system.
- **expression-macros.dat** - six expression macro vectors

The program constructs the face model from the first two files – faceline.dat and index.dat. faceline.dat contains the 3D coordinate positions in of all indices (vertices)
that will form the facial geometry. Which vertices connect which is defined in the pre-determined index.dat file. The vertices are interconnected in 3-point style forming triangular polygons that is rendered and displayed. The indices that are used to create the jaw of the face are used in the program for jaw rotation. Though there is mention of eye indices defined to allow opening and closing of the eye, this is not included in the distribution of the source code. Here, we'll create our own eye-blink animation discussed in later sections.

18 linear muscles are used in this program to create expressions, 9 on each side of the face. The muscle data file muscle.dat contains information about these muscles. Each linear muscle is defined in terms of the following (ref. to Figure 4.1):

- 3D coordinates of its point of attachment (tail - \( v_1 \)) and insertion point in skin (head - \( v_2 \))
- the fall start and finish values (\( R_s \) and \( R_f \))
- the angular zone (\( \Omega \))
- the muscle bias factor which basically is controls the elasticity of the muscle and is set to a value between 0 and 4.

To create an expression such as a smiling face requires deformation of one or more muscles of the face. expression-macros.dat contains information (macros) on the degree of muscle activations required on the 18 linear muscles used to create the expression. Six macros are defined, one for each universal expressions (happy, sad, fear, angry, disgust and surprise).

5.3.2.2 The Source Code
The version 2 of Waters program comes with the following source code files.

- head.h – contains functions for creation of face data structures.
• memory.h – utility header file for dynamic memory allocation/deallocation.

• fileio.c - I/O utilities to read ASCII text files mentioned in the previous section.

• display.c – contains routines to display various components of the face such as muscles, wireframe, texture maps etc. (dependent on OpenGL)

• SimpleFace.c - contains main and initialization procedures (startup file). This file is renamed main.cpp in our implementation

• makeface.c – contains routines for creation of the face geometry from information provided in the ASCII files.

• muscle.c – contains manipulation algorithms for the linear muscle model.

Except for memory.h all other files are modified with inclusion of other .cpp and .h files to implement the functionality of our system.

5.3.2.3 Basic Controls
The user interacts with this program through simple keyboard commands. These keyboard bindings are established with the GLAux toolkit since OpenGL does not provide any keyboard bindings itself. The GLAux toolkit has been deprecated and we’ve used the GLUT keyboard bindings instead. However we still use the GLAux libraries for texture manipulation. Figure 5.2 shows the execution mode of original Waters program.

5.4 Facial Animation Player (FAP)
This section describes the work done on our Facial Animation Player (FAP) driven by FACS in three distinct categories:
1. Enhancements and improvements made to the original Waters program to provide a better user interface (UI) and greater control over rendering options and animation.

2. Work done on our expression editor that allows the user control over all the muscles on the face to create an expression. This includes inclusion of sphincter and sheet muscles to control the facial geometry for expression creation. This gives the user the flexibility to score expressions for individual FACS AUs and even record them. These expression recordings for individual AUs are used by the Remote Player.
3. The Remote Player itself. Once a TCP connection is established with the server transferring in sequence batches of AU numbers for expressions, the players runs these batches over time creating animation.

5.4.1 Program Enhancements

The original Waters face geometry is retained and used in this project. The information from ASCII data files *faceline.dat* and *index.dat* are used to build a triangular mesh (wireframe) programmatically. This creates an advantage since most graphics hardware are accelerated to render triangles. The geometry comprises of 236 interconnected vertices forming 876 triangular polygons.

5.4.1.1 Facial Muscles

In addition to the 18 existing linear muscles, our model introduces 2 sheet muscles and 3 sphincter muscles. The mathematical modeling and their mesh deforming capabilities have already been discussed in chapter 4.

![Figure 5.3: Muscles implemented in FAP (new muscles outlined in blue)](image)

Figure 5.3: Muscles implemented in FAP (new muscles outlined in blue)
5.4.1.2 Improved User Interface (UI)

Program usability is made easy with the provision of an event driven graphical user interface using GLUI alongside the rendering window. The GUI of our system contains commonly used windows components (such as buttons, radio buttons, checkboxes and textboxes) as in other programs. It consists of five upper layer control groups. Each group expands to reveal more options pertaining to the given action.

Figure 5.4: The upper layer UI for our FAP.

5.4.1.3 Advanced Rendering Options

The original Water program provides a limited but useful set of rendering options which the user can select by toggling from one rendering mode to another using specific keyboard key. The disadvantage of this was that two or more modes could not be rendered at the same time. This part of the UI has been created to give user this flexibility on choosing the one or more rendering modes simultaneously for our geometric model. Depending on which rendering mode that is activated, different geometric details are rendered. Here are the rendering modes in our system (including those from the original waters program):
• **Polygon mode** - This is the default mode in the application, displaying the facial geometry as smooth shaded polygons. The smooth lighting effects are obtained by providing a normal at each vertex in relational to the light source. Hence the colors of the polygons at each vertex are blended in giving a smooth shading effect.

• **Wireframe mode** - renders the wireframe representation of the surface.

• **Transparent** - here the outer facial mesh behaves like a transparent physical material and shows objects behind it as obscured, particularly the eyelids and ‘skin’ that’s in the shadow of other skin areas.

• **Texture mode** - in this mode a textured version of the model is rendered.

• **Muscles** – renders all the muscles implemented in our system. Figure 5.3 was created using this option. It is best viewed with all other rendering options turned off. However if one wishes to visualize the positions of the muscles in relation to the face geometry, then used in conjunction with the wireframe mode gives best results.
• **Verts** – renders all the 256 vertices the face model comprises of. Best viewed with all other rendering modes turned off.

• **Jaw tags** – renders only the vertices that form the lower jaw (vertices that are affected by the jaw rotation function).

• **Eye tags** – vertices that form the eyelids are rendered in this mode. These vertices are used in animating the blink of the eyes.
• **Blink** – in an effort to bring realism into the face model, this option turns on the blinking of both the eyes. First the vertices forming the eye lids were identified followed by OpenGL commands that render the boundary of the eyelids (Figure 5.6 (h)). The lower part of the eyelid remains fixed in movement. To achieve animation upper eyelid vertices are moved towards the lower eyelids. These vertices are either linearly (default) or cosine interpolated over time resulting in animation of blinking eyes. The timing between the opening and closing of the eyelids are set at 25 milliseconds.

5.4.2 Animation of Waters Primitive Expressions

Waters original program implements the six universal expression – happy, sad, fear, surprise, disgust and angry. However in his implementation the expressions are displayed only as a ‘switch’ from one state to another. For example from a neutral (rest) state to happiness. This produces sharp transitions in-between states that is visible to the eyes. In real animation, the transition from one state to another should be gradual. In our system this is achieved by denoting the two states as two keyframes. In our explanation we consider the animation of the facial model from the neutral state to one that appears happy. Here the neutral state is considered the start key frame. Vertex positions for the end frame are calculated by activating all muscles that are needed to produce expression happy. The activation values of the muscles concerned are pre-defined in the original Waters expression data file - *expression-macros.dat*. Once the end frame vertex positions are determined, the system simply creates and interpolates additional frames in-between the ‘start’ and ‘end’ frames. The calculation of the in-between frames is done by defining a `Timer` function. This function is called from with
the GLUT timer function `glutTimerFunc`\(^4\) at predefined intervals. Upon each call the Timer function calculates and advances the model one frame towards the end state. In total 50 in-between states are interpolated, with the `glutTimerFunc` firing off every 5 milliseconds.

![Figure 5.7: Transformation from neutral to happy in steps of 10 in-between keyframes with timer callback set at 5ms.](image)

The technique used forms the development for our animation engine. Figure 5.8 summarizes the algorithm used.

### 5.4.3 Expression Editor

The Expression editor forms an integral part of our FAP. It gives the user control over all the muscles (including the jaw) used in our face model hence allowing a wide range of expressions to be generated with ease. Since our aim is to animate facial expression based on FACS AUs, it gives the flexibility to generate expressions that can approximate the various AUs defined in FACS. Using the expression editor the user can manipulate one or more muscles to match a particular AU expression. During this activity the calibration is rendered in the

\(^4\) A GLUT specific function that registers a timer callback to be triggered in a specified number of milliseconds [38].
Figure 5.8 Animation Engine Algorithm
graphics window for the user to note the deformations on the face model. This allows the user to fine tune the expression. Once the desired expression is generated to a satisfactory level it can then be recorded with the AU number as an identifier. There is also the possibility to modify an existing AU expression if desired by the user. A total of 64 AU expressions can be defined.

The UI for our expression editor is fairly easy to use (Figure 5.9). Each rollout contains groups of muscles for easier identification. The GUI elements are simply binded to the activation parameters of the muscles. When adjusted the muscles contract or relax depending on whether the decrement (⇠) or increment (⇢) button is pressed, deforming the facial mesh as a result. A minimum value of 0.0 and maximum value of 2.0 can be chosen as activation values for each muscle. Beyond activation value of 2.0 gives unrealistic results as far as facial mesh deformation is concerned. However the user can use a value from 0.0 to 4.0 (inclusive) for the jaw control.

By the definition of FACS, complex facial expressions are generated by applying one or more individual AUs. When these AU expressions are programmed and stored in FAP, it gives us the possibility to generate many other compound expressions simply by using combinations of AUs.

![Figure 5.9: Expression editor UI](image)

By the definition of FACS, complex facial expressions are generated by applying one or more individual AUs. When these AU expressions are programmed and stored in FAP, it gives us the possibility to generate many other compound expressions simply by using combinations of AUs.
The *Save AU* command saves the muscle activation values for the chosen AU in a C++ specific random access file. The choice of choosing to save to and retrieve data to and from a random access file is motivated by the fact that we can access specific locations regarding AUs in the file and only change those AUs. Each record in the random file is constructed using the definition of the following C++ struct where the struct elements contain the AU number and the activation values of muscles. Since only a few muscles are needed to be activated for each AU, the rest of the element values are set to 0.

```c
typedef struct AU {
    int AU_num; // Action Unit number
    float m[18]; // 18 linear muscles
    float _jaw; // jaw value activation
    float _rfrontalis; // right sheet frontalis
    float _lfrontalis; // left sheet frontalis
    float _mouth_sphincter; // mouth sphincter
    float _eye_rsphincter; // right eye sphincter
    float _eye_lsphincter; // left eye sphincter
} AU;
```

A gallery of the implemented AUs is given in the appendix A.

### 5.4.4 The Remote Player (receiver side)

Improving the Waters facial model by including the sphincter and sheet muscles and fixing the jaw movement has been the first major development task. This allowed creating more interesting expressions. Further the expression editor designed above gives us a novel way to blend in FACS results with our modified facial model. This has been the second major task. However a system that only allows creation and recording of AU expressions will seem not of much use. It will be interesting to see hence how our system will behave if we present to it a series of random AUs to animate. Surely the end result is worth noting and whether it corresponds to a valid human expression.
This led to the implementation of a small low-bandwidth teleconferencing system, a real time facial animation player (video only) driven only by FACS AUs transmitted as plain text over TCP sockets. The reasons for developing this are in twofold:

1. It will allow the testing for facial expression creation and animation by streaming AU numbers as plaintext over the internet instead of high-end media files.

2. Given the geographic nature of USP campuses, such a system would prove useful for example in live streaming or streaming recorded lecture presentations from across campuses. Instead of clogging bandwidth with high resolution video data to be transmitted, if AU numbers representing facial deformation of the person presenting the lecture can be indentified and sent, this would significantly reduce the usage of the bandwidth.

In general, multimedia content has a large volume, so media storage and transmission costs are still significant; even in today’s computing world where bandwidths have increased significantly. To offset this somewhat, media are generally compressed for both storage and streaming. This however reduces the quality. Our system attempts to solve this problem by eliminating the need to send the actual video of the person delivering the lecture. Instead the user will be presented with a facial model on the client side which will deform according to AUs sent from the server side.

Developing media players capable of streaming live video over the internet are not trivial. There are three fundamental problem areas that have to be addressed.

- Bandwidth - If the sender transmits faster than the available bandwidth then congestion occurs, packets are lost, and there is a severe drop in video quality.
If the sender transmits slower than the available bandwidth then the receiver produces sub-optimal video quality.

- Delay jitter – This is a problem because the receiver must receive/decode/display frames at a constant rate, and any late frames resulting from the delay jitter can produce problems in the reconstructed video, e.g. jerks in the video.

- Loss rate - Losses such as lost packets, bit errors or burst errors can have a very destructive effect on the reconstruction of the video on the receiver side.

It is desirable that playback can begin while data is still being delivered. This requires the usage of an appropriate buffering algorithm and the use of an appropriate size buffer which is basically shared by two processes – one that stocks up the buffer with video data to play and the other that actually plays it.

However unlike conventional media players, our remote player is very much different in the sense that it is driven only by streaming AU numbers. The following paragraphs and sections describe these developments with custom algorithms designed for buffering and playback sequences.

The remote player system consists of an AU Reader, an AU Blender, a Mixer and a Playback Controller. All these components are part of FAP (the main program) except the AU Reader that runs as a separate thread from the program. The first step is to provide all the parameters necessary to run the remote player efficiently as illustrated in Figure 5.10.
5.4.4.1 The Remote Player UI components

**IP address and Port:** The connection is achieved via a TCP socket. The TCP socket is opened by providing the IP address of the server and the port number that is open on the server side to accept receiver such as our remote player connection requests. Once a connection is established the server can then start sending the AU # to the receiver.

**Buffer size and PlayerBufferSize:** These are values used to dimension the two buffers used by our player. Buffer size dimensions the Receiver buffer that stores the incoming AU sets transmitted by the server. PlayerBufferSize dimensions the Playout buffer that stores the AU sets the Playback controller needs to play. Both these buffers are designed and constructed as First-in First-out queues. The use of these two buffers is essential to eliminate the problem of delay jitters. By default it is expected that the size for the Playout buffer will be either less than or equal to Receiver buffer.

**PlayerSpeed (1-1000 (milliseconds)):** This parameter controls the speed at which the player releases the AU set to the animation engine for playback. Graphics rendering capabilities differ from one workstation to another mainly due to
differences in CPU speed and memory size. A workstation for example with 1 GHZ CPU/256 MB RAM would render less frames per second (FPS) than a workstation with double the CPU and RAM size. To compensate for these differences the user can adjust the player speed accordingly. (An appropriate value is 250ms since the animation engine interpolates 50 frames at 5 ms intervals (250 ms per key frame)).

*Server Buffer Threshold (1-100 (%))*: This value indicates the space in *Receiver* buffer that must be filled in before the received AU set can be transferred to the *Playout* buffer for playback. For example a value of 40 indicates that once 40% of *Receiver* buffer is full, the AU set can then be ported to the *Playout* buffer up to the size that the *Playout* buffer can accommodate (given by *PlayerBufferSize* input). This gives the Playback controller adequate AUs to process and produce jerk free animation. By the time the *Playout* buffer becomes empty, additional AUs (if any) is already received and stored in the *Receiver* buffer which are then simply transferred to the *Playout* buffer again for playback.

*Player Animation Parameters*: The options listed in this group govern the use of the AU Blender and Mixer modules of the system. These are explained later in this chapter.

5.4.4.2 **AU Reader**
The AU Reader is a key component in the remote player. It is responsible for receiving the transmitted data from the server. In fact the functionality of the AU Reader is handled by a *thread* that is created as soon as the user initiates a connection. The use of a thread is desirable because our OpenGL based FAP process is already running in an endless loop drawing frames every few milliseconds. What we needed is a background process that will simply work alongside our FAP, receiving and recording AU and feeding them to the Playback
controller. This leads to the obvious advantage that the parent FAP process is not overloaded to do the job of receiving AU from the server which if it did would obviously create disruptions in rendering.

Technically, a thread is defined as an independent stream of instructions that can be scheduled to run as such by the operating system. There are many different implementations of threads; however the one used in our system is known as pthreads or POSIX threads. Pthreads are defined as a set of C language programming types and procedure calls, implemented with a pthread.h header/include file and a thread library. There are many advantages of pthreads, but for the purpose of this thesis, its ease of use and the fact that it’s available for free were two most important reasons for its adoption. For a complete reading of POSIX threads programming the reader is directed to [33].

Once the AU Reader thread has been successfully created, it first tries to establish a connection with the AU transmitting server by using the winsock API for Win32 platform. For implementation of the winsock sockets the PracticalSocket library is utilized. This library available from [34] provides wrappers for basic socket functionality. For example to connect to the server PracticalSocket/TCPSocket implemented connect function is used simply as:

TCPsocket socki.connect(servIPAddress, ServPort);

Once the connection is successful, the player is ready to receive transmission. In order for the receiver and server to communicate with each other efficiently the following literal constants are used:

1 – successful receipt of packet. (Clients signals server that packet received in good state, no loss). A ‘packet’ in our case is simply a series of AU numbers (a set) that corresponds to an expression. For example, for the expression -
happiness, the ‘packet’ would simply contain the string (plaintext) \{1, 6, 12, 14\},
for anger \{2, 4, 7, 9, 10, 20, 26\} and \{-2\} for end of transmission etc.

-1 – client signals server to pause transmission. (The likely cause of this would be
the Receiver buffer has become full and the playback controller is slow in freeing
up Playout buffer)

-2 – server signals client for end of transmission.

-3 – client signals server to resume transmission. (Receiver buffer now has space
to accommodate more AUs)

-4 – client signals server to stop transmission. (This is when the user decides to
stop the streaming pre-maturely)

AU Reader also accesses and manipulates the two buffers – Receiver and Playout.
Since the Playout buffer is also accessed by the Playback controller (a shared
resource), to avoid any deadlock situation the following condition variable is
defined and used.

`player_buffer_state` – This is set to zero (0) when the Playout buffer is or
becomes empty and set to one (1) when the Playout buffer is not 100% empty. A
value of one also indicates to the AU Reader that the Playback controller is
processing the current set of AUs. The AU reader fills the Playout buffer when it
finds the state of this condition variable to zero. Figure 5.11 describes the
algorithm used in AU Reader. Complete algorithm is described in Figure 5.12 and
Figure 5.13.
Figure 5.12 Implementation of sub functions A and B
Figure 5.13 Implementation of sub functions C and D
5.4.4.3 The Playback Controller

The Playback controller works on the concept of Keyframes. That is two key frames \((\text{start/\text{end}})\) representing two different expressions are determined. The in-between frames are then interpolated resulting in animation. The \text{end} frame becomes the \text{start} frame in the next sequence and so on. A control variable \((\text{current\_frame\_state})\) is used to keep track whether interpolating between \text{start} and \text{end} frame has been completed. It is set to zero \((0)\) if the player has finished playing the \text{end} frame, zero \((1)\) otherwise. Each entry in the \text{Playout} buffer represents one key frame in the form of a set of AUs. Each set therefore describes one expression.

Once initiated by the AU Reader the Playback controller function executes at a constant rate controlled by the GLUT timer function. Figure 5.15 describes the playback controller algorithm.

![Figure 5.14: FIFO queue implementation of Playout buffer](image)

Figure 5.14: FIFO queue implementation of \text{Playout} buffer
Set last frame rendered as start frame

More frames in Playout buffer

De-queue next frame from Playout buffer

Process frame in AU Blender function

Process frame in Mixer function

Calculate end frame by activating all muscles for the new frame.

Set current_frame_state to 1

Send frame to animation engine for playback

Streaming has finished/stopped and Playout buffer is empty

Fill Playout buffer

Receiver buffer is not empty

Close connection

Delete Receiver and Playout buffers

END of Playback

Figure 5.15: Playback controller algorithm
5.4.4.4 AU Blender
The function of the blender is to receive a set of AUs and apply the co-occurrence rules discussed in Chapter 3 Figure 3.4 thus resulting in a refined set of AU which can then be animated. A total of 16 AU expressions are implemented in our system (Table 3.1). The reason for these choices of AUs is that they are used to produce the 6 basic expressions (happy, sad, anger, fear, surprise and disgust). These basic expressions are implemented in the original waters model. However they have been produced by manipulating the muscles of the face so that the desired expression can be achieved. Our system tries to arrive at the same expression but by applying individual expressions governed by recorded AUs in the program. The AU Blender therefore implements the same dependencies as illustrated by Figure 3.4, but only for the 16 AU given in Table 3.1.

5.4.4.5 Mixer
Facial expressions used in a real life rarely contain only one single AU activation [9]. This means meaningful facial expressions consists of activations of more than one AU. Each AU activation in our system is recorded as activation values for the 18 linear muscles, 3 sphincter, 2 sheet muscles and jaw rotation – a total of 24 values. Some of the values are none-zero if the corresponding muscle is activated and zero otherwise. In order to accumulate changes resulting from activation of multiple AUs to create an expression, we use the following mechanism.

Average – In this mixer the muscle activation values for each AU in the set of AUs to generate an expression is investigated in parallel. If the corresponding muscles are activated in the AUs, their values are added and averaged according to the number of AUs which have that muscle activated. The following example illustrates the average mixer functionality (only a subset of muscles is shown):
Muscle Activation

<table>
<thead>
<tr>
<th>Muscle</th>
<th>AU a</th>
<th>AU b</th>
<th>AU c</th>
<th>Average Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Zygomatic Major</td>
<td>1.2</td>
<td>0.0</td>
<td>0.6</td>
<td>0.9</td>
</tr>
<tr>
<td>Left Angular Depressor</td>
<td>0.8</td>
<td>0.2</td>
<td>1.4</td>
<td>0.8</td>
</tr>
<tr>
<td>Left Frontalis Inner</td>
<td>0.0</td>
<td>0.0</td>
<td>0.9</td>
<td>0.9</td>
</tr>
</tbody>
</table>

... ... ... ...

Table 5.1 Average mixer technique used in FAP

It shows from the above technique that for muscles that are commonly used in many AUs, their activation values are blended across the number of AUs.

**Maximum** – Here, activation values of common muscles across AUs are analyzed and the maximum activation value selected. The advantage of this mixer is that subtle changes would be overshadowed by higher muscle activation values giving maximum deformation. Table 5.2 shows the implementation of the maximum mixer.

<table>
<thead>
<tr>
<th>Muscle</th>
<th>AU a</th>
<th>AU b</th>
<th>AU c</th>
<th>Maximum Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Zygomatic Major</td>
<td>1.2</td>
<td>0.0</td>
<td>0.6</td>
<td>1.2</td>
</tr>
<tr>
<td>Left Angular Depressor</td>
<td>0.8</td>
<td>0.2</td>
<td>1.4</td>
<td>1.4</td>
</tr>
<tr>
<td>Left Frontalis Inner</td>
<td>0.0</td>
<td>0.0</td>
<td>0.9</td>
<td>0.9</td>
</tr>
</tbody>
</table>

... ... ... ...

Table 5.2 Maximum mixer technique in FAP

### 5.4.5 The AU Transmitter (sender side)

The remote player (receiver side) works on the assumption that there exists a system that is capable of recognizing AU from live video frames. These AU can then be sent to our remote player for animation. There has been some research
done in this area [23, 35, 36, 37], however due to the scope of this project we’ve implemented a system that simply transmits AU over the internet.

The AU Transmitter is developed as a separate program. It is implemented using the PracticalSocket and POSIX thread libraries. Its main function is to listen to client requests, that is requests from the remote player (receiver side), and then transmit to the client a list of AUs. Once a connect request is received from the receiver, a new thread is created to handle that client. This gives the AU Transmitter capability to handle multiple clients at once. The task of the thread is to simply communicate with the receiver (using the given protocols discussed earlier) and end once all AUs have been transmitted.

The list of AUs is read from a plaintext file which employs a special format. The tags **Beg_Stream** and **End_Stream** define the AU list. In between these tags are combinations of AUs that make up an expression. Each combination of AUs is treated as a packet which is sent to the receiver. As an example, an abstract of the file is given below.

![Figure 5.16 Example format of file read by AU Transmitter](image-url)

Figure 5.16 Example format of file read by AU Transmitter
5.4.6 Test Results

We test the Remote Player’s ability to animate expressions based on a given set of AUs. We use the AU set defined for the 6 basic expressions in [14]. We also test the system combining some of these basic expressions and evaluate their results.

<table>
<thead>
<tr>
<th>Configurations</th>
<th>System 1</th>
<th>System 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>INTEL Core 2 Duo 2194 Mhz</td>
<td>INTEL Centrino 1860 Mhz</td>
</tr>
<tr>
<td>RAM</td>
<td>1GB</td>
<td>1GB</td>
</tr>
<tr>
<td>GFX</td>
<td>INTEL Q35 Express 384MB</td>
<td>INTEL(R) 915GM/GMS Chipset</td>
</tr>
<tr>
<td>OS</td>
<td>Win XP</td>
<td>WinXP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Average FPS</th>
<th>System 1</th>
<th>System 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>800x600</td>
<td>463.6</td>
<td>359.0</td>
</tr>
<tr>
<td>1024x768</td>
<td>469.6</td>
<td>347.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model Statistics</th>
<th>Vertices</th>
<th>Polygons</th>
<th>Muscles: Linear</th>
<th>Muscles: Sphincter</th>
<th>Muscles: Sheet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>236</td>
<td>876</td>
<td>18</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.3: Some standard statistics of FAP rendering on 2 test platforms.

<table>
<thead>
<tr>
<th>Expression</th>
<th>AUs Transmitted</th>
<th>Resultant list of AUs after applying co-occurrence rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surprise</td>
<td>1, 2, 5, 15, 16, 20, 26</td>
<td>2, 5, 11, 15, 26</td>
</tr>
<tr>
<td>Fear</td>
<td>1, 2, 4, 5, 15, 20, 26</td>
<td>2, 4, 5, 11,15, 26</td>
</tr>
<tr>
<td>Disgust</td>
<td>2, 4, 9, 15, 17</td>
<td>2, 4, 9, 11, 15, 17</td>
</tr>
<tr>
<td>Anger</td>
<td>2, 4, 7, 9, 10, 20, 26</td>
<td>2, 4, 9, 11, 15, 26</td>
</tr>
<tr>
<td>Happiness</td>
<td>1, 6, 12, 14</td>
<td>6, 11, 14, 15</td>
</tr>
<tr>
<td>Sadness</td>
<td>1, 4, 15, 23</td>
<td>4, 11, 15, 23</td>
</tr>
</tbody>
</table>

Table 5.4: Application of co-occurrence rules for the 6 basic expressions
Figure 5.17: Expression results of the 6 basic expressions by applying co-occurrence rules and maximum mixer.

Figure 5.18: Results of some of the combinations of the 6 basic expressions.

The complete result of the expressions generated by applying different mixers and the co-occurrence rules is given in Appendix B.

The following table gives results of a simple test performed to measure the bandwidth usage of transferring single expressions, the duration of playback and the amount of time the remote player spent on receiving each expression. The data for this table was gathered by performing 5 executions cycles for each single expression.

Given the bytes transferred for each single AU, the approximate bandwidth usage is calculated as:
\[ \% \text{bandwidth usage} = \frac{\text{bytes transferred}}{\text{bandwidth size}} \times 100 \]

Using the following conversions:

100Mbps (Megabits per second) = 12.5 MBps (Megabytes per second)
1 Megabyte = 1024 Bytes

<table>
<thead>
<tr>
<th>Bandwidth size:</th>
<th>100Mbps LAN</th>
<th>Client Platform:</th>
</tr>
</thead>
<tbody>
<tr>
<td>System 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Playout Buffer Size:</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Receiver Buffer Size:</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Player Speed:</td>
<td>250ms</td>
<td></td>
</tr>
<tr>
<td>Threshold:</td>
<td>40%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Server Side</th>
<th>Receiver Side</th>
<th>bandwidth used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bytes Transferred</td>
<td>Transmission send Time (average-seconds)</td>
<td>Bytes Received</td>
</tr>
<tr>
<td>Surprise</td>
<td>19</td>
<td>0.0189</td>
<td>19</td>
</tr>
<tr>
<td>Fear</td>
<td>18</td>
<td>0.0228</td>
<td>18</td>
</tr>
<tr>
<td>Disgust</td>
<td>13</td>
<td>0.0155</td>
<td>13</td>
</tr>
<tr>
<td>Anger</td>
<td>18</td>
<td>0.0149</td>
<td>18</td>
</tr>
<tr>
<td>Happiness</td>
<td>11</td>
<td>0.0154</td>
<td>11</td>
</tr>
<tr>
<td>Sadness</td>
<td>11</td>
<td>0.0214</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 5.5: Bandwidth usage for single expressions.

As an example from the above analysis, it can be calculated that approximately 33 frames of “surprise” like expressions can be transmitted by using only 5% of the total bandwidth with the remaining 95% to be used for other purposes (including audio transfer). This shows video transferred only as a series of AU activations is clearly an effective way to minimize bandwidth usage and at the same time achieve animation of the desired expression.
Chapter 6

6. Conclusion & Future Work

Given the nature of the facial modeling and animation field with availability of a vast number of existing techniques and the ever increasing realm of 3D facial animation in the now digital world; proved difficult to make choices at the beginning on techniques to employ to arrive at our research object. This research also led to study of other areas of research such psychology (study of FACS) and anatomy (study of the human face muscles aspect – due to adoption of the Waters muscle model).

The applications of a good facial animation system are many. For example, a good facial model could be used in such application as character animation in entertainment industry and advertising, low-bandwidth teleconferencing, advanced human-computer interfaces and computer synthetic facial surgery to name a few.

Considering the geographic locations of the member countries of The University of the South Pacific (USP), the ever increasing need for greater communication, and the costs involved in maintaining these communications, a major advantage of this project is the possibility of setting up a low-bandwidth teleconferencing system. A facial expression recognition system capable of recognizing AUs from video footage of people’s faces could be used to transmit information of AUs detected in plain text form to our animation system in the format required. In this way our FACS based system could be used as a remote player.
Our main research objective was to show the feasibility of using the Facial Action Coding System (FACS) paradigm for the animation of facial expressions. The results obtained by our research are an affirmation that indeed FACS can be used as a basis for facial animation. Hence in trying to achieve our research objective, the following new contributions have been made to the field.

- Enhancements to the Waters anatomy-based head model: An implementation of this model obtained from [4] was enhanced by:
  - improving its UI,
  - adding additional sheet and sphincter muscles,
  - providing an alternate implementation to the jaw rotation function, and;
  - presenting a new sphincter muscle equations that can be used around the eyes.

- Implementation of an expression editor that allows with ease expressions to be artificially posed by setting the desired muscle parameters for a particular action unit.

- Using co-occurrence rules developed by Wojdel [9], in creating facial expressions by activating two or more pre-programmed Action Units (AUs) and viewing their result.

- Development of a simple animation engine for simulation; and

- Implementation of a low-bandwidth teleconferencing, a real time facial animation player driven only by FACS AUs transmitted as plain text over TCP sockets.

Our Facial Animation Player is quite basic and primarily an implementation of the concepts discussed in earlier chapters. There could have been more graphical effects added to it thus improving the visual quality.
The Waters muscle model opens a gateway to excellent research in facial animation controlled by muscles. However we do note the muscles (linear/sphincter/sheet) in no way resemble how real muscles look, their shape, size and volume due to their representation in 2D. In fact real muscles when contracted, contract with the skin rather than just displacing skin; have different and unequal volumes across the muscle shape that determines how much it can be contracted; and overlap each other where in many cases activation of one muscle triggers the activation of adjacent muscles. Hence future direction of this research could be development of a 3D muscle model. This would perhaps have more fantastic effects.
# Appendix A

## Gallery – Implemented AUs

The example images are sourced from

http://www.cs.cmu.edu/afs/cs/project/face/www/facs.htm

<table>
<thead>
<tr>
<th>AU</th>
<th>FACS Name</th>
<th>Example image</th>
<th>Implemented Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Inner Brow Raiser</td>
<td><img src="image1.png" alt="Example" /></td>
<td><img src="image1%E7%9A%84%E8%A1%A8%E6%83%85.png" alt="Implemented Expression" /></td>
</tr>
<tr>
<td>2</td>
<td>Outer Brow Raiser</td>
<td><img src="image2.png" alt="Example" /></td>
<td><img src="image2%E7%9A%84%E8%A1%A8%E6%83%85.png" alt="Implemented Expression" /></td>
</tr>
<tr>
<td>4</td>
<td>Brow Lowerer</td>
<td><img src="image4.png" alt="Example" /></td>
<td><img src="image4%E7%9A%84%E8%A1%A8%E6%83%85.png" alt="Implemented Expression" /></td>
</tr>
<tr>
<td>5</td>
<td>Upper Lid Raiser</td>
<td><img src="image5.png" alt="Example" /></td>
<td><img src="image5%E7%9A%84%E8%A1%A8%E6%83%85.png" alt="Implemented Expression" /></td>
</tr>
<tr>
<td>6</td>
<td>Cheek Raiser</td>
<td><img src="image6.png" alt="Example" /></td>
<td><img src="image6%E7%9A%84%E8%A1%A8%E6%83%85.png" alt="Implemented Expression" /></td>
</tr>
<tr>
<td>No.</td>
<td>Muscle</td>
<td>Image 1</td>
<td>Image 2</td>
</tr>
<tr>
<td>-----</td>
<td>-------------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>7</td>
<td>Lid Tightener</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
</tr>
<tr>
<td>9</td>
<td>Nose Wrinkler</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
</tr>
<tr>
<td>10</td>
<td>Upper Lip Raiser</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
</tr>
<tr>
<td>12</td>
<td>Lip Corner Puller</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
</tr>
<tr>
<td>14</td>
<td>Dimpler</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
</tr>
<tr>
<td>15</td>
<td>Lip Corner Depressor</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
</tr>
<tr>
<td>16</td>
<td>Lower Lip Depressor</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Chin Raiser</td>
<td><img src="image1.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Lip stretcher</td>
<td><img src="image2.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Lip Tightener</td>
<td><img src="image3.png" alt="Image" /></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Jaw Drop</td>
<td><img src="image4.png" alt="Image" /></td>
<td></td>
</tr>
</tbody>
</table>
Appendix B

Expression Results

The following tables including images illustrate the test results of transmitting the AUs of the six basic expressions and animating them on our Remote Player. A comparative analysis is made using Waters six primitive expressions as a benchmark. The AUs selected to create the expression animations are defined in Table 3.2 (Chapter 3).

Animation Results using the Average Mixer

<table>
<thead>
<tr>
<th>Expression</th>
<th>Using Average Mixer</th>
<th>Waters Expression (benchmark)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>With co-occurrence</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rules</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Without co-occurrence rules</td>
<td></td>
</tr>
<tr>
<td>Surprise</td>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
<tr>
<td>Fear</td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td>Emotion</td>
<td>Image 1</td>
<td>Image 2</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>Disgust</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
</tr>
<tr>
<td>Anger</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
</tr>
<tr>
<td>Happiness</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
</tr>
<tr>
<td>Sadness</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
</tr>
</tbody>
</table>
Animation Results using the Maximum Mixer

<table>
<thead>
<tr>
<th>Expression</th>
<th>Using Maximum Mixer</th>
<th>Waters Expression (benchmark)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>With co-occurrence rules</td>
<td>Without co-occurrence rules</td>
</tr>
<tr>
<td>Surprise</td>
<td><img src="image1" alt="Surprise with" /></td>
<td><img src="image2" alt="Surprise without" /></td>
</tr>
<tr>
<td>Fear</td>
<td><img src="image3" alt="Fear with" /></td>
<td><img src="image4" alt="Fear without" /></td>
</tr>
<tr>
<td>Disgust</td>
<td><img src="image5" alt="Disgust with" /></td>
<td><img src="image6" alt="Disgust without" /></td>
</tr>
<tr>
<td>Anger</td>
<td><img src="image7" alt="Anger with" /></td>
<td><img src="image8" alt="Anger without" /></td>
</tr>
<tr>
<td>Happiness</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sadness</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
</tbody>
</table>
### Appendix C

#### FACS AUs Tables

Table 1: Single action units

(Source: [http://psychology.wikia.com/wiki/Facial_Action_Coding_System](http://psychology.wikia.com/wiki/Facial_Action_Coding_System))

<table>
<thead>
<tr>
<th>AU No.</th>
<th>FACS Name</th>
<th>Muscular Basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Inner Brow Raiser</td>
<td>Frontalis Pars Medialis</td>
</tr>
<tr>
<td>2</td>
<td>Outer Brow Raiser</td>
<td>Frontalis Pars Lateralis</td>
</tr>
<tr>
<td>4</td>
<td>Brow Lowerer</td>
<td>Depressor Glabellae; Depressor Supercilli; Corrugator</td>
</tr>
<tr>
<td>5</td>
<td>Upper Lid Raiser</td>
<td>Levator Palpebrae Superioris</td>
</tr>
<tr>
<td>6</td>
<td>Cheek Raiser</td>
<td>Orbicularis Oculi; Pars Orbitalis</td>
</tr>
<tr>
<td>7</td>
<td>Lid Tightener</td>
<td>Orbicularis Oculi; Pars Palebralis</td>
</tr>
<tr>
<td>9</td>
<td>Nose Wrinkler</td>
<td>Levator Labii Superioris; Alaeque Nasi</td>
</tr>
<tr>
<td>10</td>
<td>Upper Lip Raiser</td>
<td>Levator Labii Superioris; Caput Infraorbitalis</td>
</tr>
<tr>
<td>11</td>
<td>Nasolabial Fold Deepener</td>
<td>Zygomatic Minor</td>
</tr>
<tr>
<td>12</td>
<td>Lip Corner Puller</td>
<td>Zygomatic Major</td>
</tr>
<tr>
<td>13</td>
<td>Cheek Puffer</td>
<td>Caninus</td>
</tr>
<tr>
<td>14</td>
<td>Dimpler</td>
<td>Buccinator</td>
</tr>
<tr>
<td>15</td>
<td>Lip Corner Depressor</td>
<td>Triangularis</td>
</tr>
<tr>
<td>16</td>
<td>Lower Lip Depressor</td>
<td>Depressor Labii</td>
</tr>
<tr>
<td>17</td>
<td>Chin Raiser</td>
<td>Mentalis</td>
</tr>
<tr>
<td>18</td>
<td>Lip Puckerer</td>
<td>Incisivii Labii Superioris; Incisivii Labii Inferioris</td>
</tr>
<tr>
<td>20</td>
<td>Lip Stretcher</td>
<td>Risorius</td>
</tr>
<tr>
<td>21</td>
<td>Neck Tightner</td>
<td>*non-facial muscle</td>
</tr>
<tr>
<td>22</td>
<td>Lip Funneler</td>
<td>Orbicularis Oris</td>
</tr>
<tr>
<td>23</td>
<td>Lip Tightener</td>
<td>Orbicularis Oris</td>
</tr>
<tr>
<td>24</td>
<td>Lip Pressor</td>
<td>Orbicularis Oris</td>
</tr>
<tr>
<td>25</td>
<td>Lips Part</td>
<td>Depressor Labii or Relaxation of Mentalis or Orbicularis Oris</td>
</tr>
</tbody>
</table>
26 Jaw Drop Masetter; Temporal and Internal Pterygoid Relaxed
27 Mouth Stretch Pterygoids; Digastric
28 Lip Suck Orbicularis Oris
31 Jaw Clencher *non-facial muscle
38 Nostril Dilator Nasalis, Pars Alaris
39 Nostril Compressor Nasalis, Pars Transversa and Depressor Septi Nasi
43 Eyes Closed Relaxation of Levator Palpebrae Superioris
45 Blink Relaxation of Levator Palpebrae and Contraction of Orbicularis oculi, Pars Palpebralis
46 Wink Orbicularis Oculi

Table 2: Action unit descriptors

(Source: http://psychology.wikia.com/wiki/Facial_Action_Coding_System)
Appendix D

Installation Instructions

FAP has been written in C/C++ language and was developed using MS Visual Studio 6.0 on a Windows XP platform. To execute the source code, the reader needs to download and install the following libraries.

OpenGL based graphics libraries

- GLUT (version 3.7.6) and GLUI (version 2.0)

Install the following files in the respective directories:

- glut32.dll    - %windows%\system32
- glut32.lib    - %...%\Microsoft Visual Studio\VC98\lib
- glut.h        - %...%\Microsoft Visual Studio\VC98\include\GL
- glui.h        - %...%\Microsoft Visual Studio\VC98\include\GL
- glos.h        - %...%\Microsoft Visual Studio\VC98\include\GL

POSIX (pthread) libraries

- pthread (version VC1)

Install the following files in the respective directories:

- pthreadVC1.dll - %windows%\system32
- pthreadVC1.lib - %...%\Microsoft Visual Studio\VC98\lib
- pthreadVC1.h    - %...%\Microsoft Visual Studio\VC98\include
- sched.h         - %...%\Microsoft Visual Studio\VC98\include
- semaphore.h     - %...%\Microsoft Visual Studio\VC98\include
Project Settings in MS Visual Studio 6.0

Add the following libraries (in-order separated by spaces) in the Project→Settings window by clicking on the link tab in the Object/library modules textbox. Care must be taken not to delete the existing library entries.

opengl32.lib glu32.lib glaux.lib wsock32.lib pthreadVC1.lib

And you’re good to go!
Bibliography


[3] “Facs - facial action coding system (example images),”


[12] B. Sharp, “Subdivision surface theory,” in


http://dipaola.org/stanford/facial/class8notes.html.

[16] F. Parke, D. Terzopoulos, T. Sejnowski, P. Stucki, L. Williams, D. Ballard,
L. Sadler, J. Hager, D. Psaltis, and J. Zhang, “NsF report - facial expression


University - Sweden, 2005.


investigator’s guide - html demostration version,” A Human Face, Salt Lake
City, UT, 2002.


expression recognition based on faces action units,” Proceedings. Third IEEE
http://terasonic.egloos.com/814963.


[29] K. Waters, “Computer facial animation website,” in


[34] “Practicalsocket.h file reference,”
http://fastlabinc.com/CSL/docxygen/_practical_socket_8h.html.


