MPEG-4 Facial Feature Point Editor

Jonas Lundberg

LITH-ISY-EX-3269-2002

*Linköping, September 2002*
MPEG-4 Facial Feature Point Editor

Master thesis at the Image Coding Group at Linköping University

By

Jonas Lundberg

LITH-ISY-EX-3269-2002

Examiner: Robert Forchheimer

Linköping, September 2002
The use of computer animated interactive faces in film, TV, games is ever growing, with new application areas emerging also on the Internet and mobile environments. Morph targets are one of the most popular methods to animate the face. Up until now 3D artists had to design each morph target defined by the MPEG-4 standard by hand. This is a very monotonous and tedious task. With the newly developed method of Facial Motion Cloning [11] the heavy work is relieved from the artists. From an already animated face model the morph targets can now be copied onto a new static face model.

For the Facial Motion Cloning process there must be a subset of the feature points specified by the MPEG-4 standard defined. The purpose of this is to correlate the facial features of the two faces. The goal of this project is to develop a graphical editor in which the artists can define the feature points for a face model. The feature points will be saved in a file format that can be used in a Facial Motion Cloning software.
Abstract
The use of computer animated interactive faces in film, TV, games is ever growing, with new application areas emerging also on the Internet and mobile environments. Morph targets are one of the most popular methods to animate the face. Up until now 3D artists had to design each morph target defined by the MPEG-4 standard by hand. This is a very monotonous and tedious task. With the newly developed method of Facial Motion Cloning [11] the heavy work is relieved from the artists. From an already animated face model the morph targets can now be copied onto a new static face model.

For the Facial Motion Cloning process there must be a subset of the feature points specified by the MPEG-4 standard defined. The purpose of this is to correlate the facial features of the two faces. The goal of this project is to develop a graphical editor in which the artists can define the feature points for a face model. The feature points will be saved in a file format that can be used in a Facial Motion Cloning software.
Acknowledgements

First of all I would like to thank Sandra for putting up with me during all this. For the time spent listening to me rant about it, proofreading and helping me with the presentations.

Next up I would like to thank my tutor Igor. We made it through some good times and bad times and it all worked out in the end. Also I would like to thank my examiner Robert and project involved person Jörgen at ISY.

All the people in the thesis worker room for keeping my caffeine level high, and last but not least all friends who stopped by from time to time and made sure I did not work too hard.
1 Introduction............................................................................................................. 1
  1.1 Background ....................................................................................................... 1
  1.2 Project provider ............................................................................................... 1
  1.3 Structure of the report ..................................................................................... 1
  1.4 Commonly used abbreviations ....................................................................... 2

2 Background – Facial animation and cloning ..................................................... 3
  2.1 Brief history of facial animation ................................................................. 3
  2.2 Facial parameterisation ............................................................................... 3
  2.3 Use of facial animation today ...................................................................... 4
  2.4 The facial animation scene and its actors ............................................... 4
  2.5 Model-based coding ..................................................................................... 5
  2.6 Morph targets ............................................................................................... 5
  2.7 Facial animation with the MPEG-4 standard ............................................. 6
    2.7.1 Feature points ....................................................................................... 6
    2.7.2 Facial Animation Parameters ............................................................. 7
  2.8 Facial motion cloning ................................................................................... 8

3 Problem analysis ............................................................................................... 10
  3.1 Problem definition ....................................................................................... 10
  3.2 Why is it needed? ......................................................................................... 10
  3.3 Who will use it? ........................................................................................... 11
  3.4 How will it be used? .................................................................................... 12
  3.5 Analysis of the requirements ...................................................................... 12
    3.5.1 Graphical representation of the face model ........................................ 12
    3.5.2 Two functions – one program ............................................................. 12
  3.6 Programming in a Windows environment .................................................. 13
    3.6.1 MFC .................................................................................................. 13
    3.6.2 Document / view architecture ............................................................ 13

4 Components and techniques used ................................................................... 15
  4.1 3D graphics .................................................................................................. 15
    4.1.1 How 3D graphics works ..................................................................... 15
    4.1.2 3D models .......................................................................................... 15
    4.1.3 Different display modes ...................................................................... 15
    4.1.4 Hierarchical 3D scenes ..................................................................... 16
  4.2 Implementing an OpenGL renderer .............................................................. 17
    4.2.1 Design idea of the view ...................................................................... 17
    4.2.2 How a VRML scene is rendered to the screen .................................... 17
    4.2.3 Selecting vertices .............................................................................. 18
    4.2.4 Displaying selected vertices ............................................................. 18
    4.2.5 Using the renderer in lip editing mode .............................................. 19
    4.2.6 Problems with different model sizes ............................................... 19
  4.3 Implementing the document ......................................................................... 20
    4.3.1 Role of the document ........................................................................ 20
    4.3.2 Data contained in the document ...................................................... 20
1 Introduction

Facial animation has been used since long time. The first effort to animate a face using a computer dates more than 25 years back [14] and as a result of today’s search for new ways to use computers to convey information to humans this field has greatly expanded during the last years. The human face is very complex and thus hard to model. A subtle change in for example the eyebrows can change a kind looking expression to a vicious image.

It is today that the research is paying off and the animated faces actually begin to resemble human ones. With the constant increase in computing power and more advanced hardware emerging today realistic results of facial animation can now be seen in movies such as Final Fantasy, Star Wars, Antz etc.

1.1 Background

The facial animation community has recently adopted the first standard for facial animation – the MPEG-4 standard. This is a great leap forward in the field of facial animation unifying facial artists, facial animation researchers and image coding researchers. The MPEG-4 defines 66 low level facial animation parameters (FAP), and two high level that can be used for expressions and visemes. All of these are used to control the feature points on a face. A common way to define the face for all FAP is to use morph-targets. The morph-targets depict the face in positions defined by the FAP. These morph targets are then morphed in an animation decoder to produce the actual facial animation. This is a tedious and monotonous process for the artists to model all these facial expressions. Therefore a method of cloning the expressions from an already animated face to a static face has been developed - the facial motion cloning process (FMC) has been proposed. For this process to function both the animated and the static face must have certain feature points defined, points that allow the FMC to correlate the faces to each other and then copy the expressions. In defining these feature points there is need for an editor where the artist can simply load the face, click on the vertices for the feature points and save them to a file for use in a FMC software. It is the purpose of this project to implement such an editor.

1.2 Project provider

This project has been a collaboration between the Department of Electrical Engineering (ISY) at Linköping University and the Swedish public network television (SVT). The product was intended to be used in a TV-show named Tredje Makten (The Third Power). The TV-show would contain a 2-minute computer animation of a celebrity in Sweden discussing a current matter.

1.3 Structure of the report

Chapter 2 provides the background needed for this report. The reader is assumed to have this knowledge when reading the problem definition.

In chapter 3 the problem is analysed and a solution outline is provided.

Chapter 4 expands the analysis describing more in depth the chosen components and how they fit into the solution.

In chapter 5 the actual result is described, exposing snapshots of the program.

Chapter 6 describes the .fdp file format.

In chapter 7 a conclusion of the project is made.

Finally chapter 8 provides some examples of future improvements that could be done.
1.4 Commonly used abbreviations

These abbreviations are provided as a quick reference.

- FMC – Facial Motion Cloning
- FAP – Facial Animation Parameters
- FP – Feature Point
- API – Application Program Interface. A set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks. A programmer puts the blocks together
- VRML – Virtual Reality Modelling Language
- MFC – Microsoft Foundation Classes. A programming framework provided to simplify programming Windows applications
- IDE – Integrated Development Environment. An editor program provided with a compiler that helps edit and create files for the compiler
2 Background – Facial animation and cloning

This chapter provides background to the facial animation field and explains the facial motion cloning technique.

2.1 Brief history of facial animation

The first ever computer animated facial images were created by Frederic I Parke as part of a graphics course at the University of Utah in the early seventies. In the beginning the face was animated very crudely with polygonal representation of the head. A few years later he managed with the new polygon shading techniques just emerging to create a somewhat realistic animation. He did this by collecting data from photos, transferring their features to his polygon model and then interpolating between these different expressions [2].

In 1980 Platt presented a physically based muscle-controlled facial model. The model had its design founded in the human facial muscles. There was also concurrent research in the field of 2D facial animation. In 1985 LaChapelle and Bergeron made a new landmark in facial animation when they created the short film *Tony de Peltrie* using 3D facial animation with synchronized speech. In 1987 Waters provided a new muscle-based model that allowed a wide range of expressions to be created by controlling the underlying muscles. 1988 Pixar won an Academy Award for their short film *Tin Toy*, a story containing a computer animated baby. The baby’s face was depicted using facial animation [2].

The emerging of optical range scanners in the early 90ies provided a much easier way to gather data for the facial animation. Up till then the most common way of gathering facial data was simply using photos. With the more and more enhanced field of computer image processing, more powerful computers and graphics cards facial animation is today closing in on realistic results. Also there has been great interest from the computer image processing research in tracking facial features. That is, through image processing extract the facial expressions and position of a human head. This data can then be used to control a facial model [2].

2.2 Facial parameterisation

Directly linked with the actual animation is the parameterisation of a face. That is a way of describing a face using parameters to control it so that different systems can exchange information on the animated faces. The first person to actually try describing human faces was Charles Darwin in 1872 with the book *Expressions of the Emotions in Man and Animals* where he tried to categorise human expressions [2].

There has also been extensive research in this area dating back to the 70ies when the FACS system was developed. FACS is an abbreviation for Facial Action Coding System. This was intended to be a way of scoring human expressions, but with the systematic approach this also appealed to the facial animation researchers. The FACS method defines 46 action units, or basic facial movements, on a face [4].

Up until today there has been presented several models using direct parameterisation, pseudo-muscle based, muscle-based or interpolation. The ideal parameterisation should of course be able to express every possible expression on a human face with the model. This is not achievable as the amount of parameters would be tremendous.
2.3 Use of facial animation today

Today, facial animation is widely used in films, computer games and medical equipment. The latest addition to the facial animation family is the use of interactive figures on the web [9]. The adding of an animated face can make the user experience more natural than using a keyboard or a mouse. Also there is a vast field for the physically disabled persons who are unable to use today’s means of interaction with a computer [10].

The aim for facial animation today is of course real-time rendering. In web applications the interactive face must immediately give some feedback to the user. A face turning into an hourglass while generating an answer would not be a nice solution.

2.4 The facial animation scene and its actors

For many years now the facial animation field has not been uniform. There have been several camps, none eager to communicate with the other. Since this chapter deals with facial animation it is natural to discuss the researchers of the field. These persons are interested in the actual modelling of a face; how it is best parameterised, coded and re-constructed. Achieving realistic results is of course the main goal. But the collaboration with the end users - the facial artists producing animations and modelling faces has not been so good. The task of actually constructing an animatable face was often not considered, often leaving the facial artists with tremendous tasks of designing the faces.

The facial artists on the other hand were only interested in modelling the faces and animating them. There was often a new solution every time a face would be animated and of course a lot of time was spent on re-inventing the wheel at every occasion. They did not care much about how the face was parameterised or coded, simply that the appearance of the face was realistic.

The third camp is the image coding researchers. Model-based image coding research focuses amongst other things on how to recognize objects from still pictures. For example to supervise traffic moving cars must be recognized from images. This is applicable when dealing with facial animation as well, namely how the parameters could be extracted from moving images and then used to make a facial animation. This way an actor could be recorded, and his features be captured from video and transferred to an animated face. This gives higher realism since the animated face would behave exactly as a human. There has been some collaboration between the image coding camp and the facial animation camp.
2.5 Model-based coding

When using model-based coding, a face is constructed according to a specific model. The feature of a face is converted to a set of parameters. The parameters characterizing a face are (hopefully) chosen so that the feature of the face can be recreated using these parameters. The obvious gain using model-based coding compared to sending pre-rendered images is when communicating in low bit-rate networks. This stream of parameters can also be compressed to save even more bandwidth. They are then sent over the network to the receiving software that re-constructs the face using the same coding scheme [4]. Another advantage is that the receiver is given the liberty to choose whichever representation that is wanted. If the representation presented is not satisfactory, just change it, as long as the desired face complies with the coding model. Of course both coder and decoder of the parameters must use the same scheme. The only industrial standard coding scheme today is the MPEG-4 standard [6]. Figure 2.1 shows an example of a model coded face - Candide. The model is controlled by global and local action units. The global ones correspond to rotations around three axes. The local action units control the mimics of the face so that different expressions can be obtained [15].

![Figure 2.1 Candide – one of the first coded models of a face.](image)

2.6 Morph targets

Morph targets are a very common way of describing the different expressions of a parameterised face. Just describing the parameters on a face is not enough; it has to be known how they move on a face. A morph target describes how a face looks in a specific position. If a parameter is the corner of the lip it has to be known how the face looks like when this parameter is at its minimum and maximum position. From this new faces can be produced using these two morph targets and then morphing them together, creating an animation of the face moving from one expression to another. There are two major ways of morphing: weighted morphing and segmented morphing.
Weighted morphing concerns when a base face is morphed with two or more targets, and the result then is combined [3]. This way for example a face pronouncing a vowel and a face frowning can be morphed into one, creating a frowning face pronouncing a vowel. The morph targets can be assigned weights of how great percentage they should affect the outcome.

Segmented morphing is when separate areas of the face are morphed individually [3]. The power of this technique is total control of the separate parts individually. Motion in one group does not affect the others. First of all, separate areas of the face have to be defined. Then different morph targets affecting that area have to be modelled. This is often a very tedious task for the artists. There is really no limit on how many morph targets that can be created, only the artists level of perfection sets the boundaries.

Both techniques can be used in facial animation, though the latter one is more widely used as it coincides more with the method of a parameterised face.

![Example of three morph targets](image)

### 2.7 Facial animation with the MPEG-4 standard

As previously mentioned the facial animation community was out of focus. The different camps were if not ignorant towards each other least not cooperating very well. There was clearly a need for a standard that could unify the researchers, image coders and facial animation artists. A group under the flag of the Motion Picture Expert Group (MPEG) standard set out to form a standard for facial animation.

In 1999 the MPEG-4 standard was adopted - the standard is aimed for multimedia for the fixed and mobile web. In this standard, a way of sending parameters to an animatable face is defined. There is also a parameterisation of the face developed, as well as how these parameters move and relate to each other.

#### 2.7.1 Feature points

The MPEG-4 uses the notion of a parameterised facial model, as discussed in chapter 2.2. MPEG-4 labels these parameters feature points. These feature points have been chosen as the best way of representing a face and were chosen so that it would be general for any face and not force the artist to design only one type of faces. The feature points were chose so that they could be used not only for human faces but also with cartoon-like characters that have, to say the least, exaggerated human features and expressions [4]. The standard defines 84 feature points on a neutral face as shown below in Figure 2.3.
The format of a feature point is `<group>.<index>` where the group is a facial feature such as eyes, mouth, ears. The index is just a way to label the feature points contained in this group. The location of these feature points has to be known for a MPEG-4 compliant face model [4].

The feature points are used to control the segmented morphing (facial animation).

### 2.7.2 Facial Animation Parameters

The feature points themselves only define the vertices that must be known on a face. They cannot themselves produce animatable results. The feature points have to be controlled somehow so that at a certain point in time, the feature points are positioned to form the facial expression desired by the artist. Thus arises the need for defining morph targets for the face. These morph targets define how the face is deformed when a feature point is moving.
The MPEG-4 standard defines a way to control these 84 feature points using the notion of Facial Animation Parameters (FAP). The MPEG-4 standard defines 66 low level FAP and 2 high level FAP [6]. The low level FAP are defined as the position of the 84 feature points, as shown in Figure 2.3, and is closely related to the movement of facial muscles. Each of the 66 low level FAP define a movement for a specified number of feature points, for example FAP number 46 denotes raise_tounge. When FAP 46 is issued the feature points associated with this FAP is morphed in some manner (often simply interpolated) from the current look of the face to the new look. This way an animation of the face is achieved.

In the MPEG-4 the FAP are stored in a compressed stream and then fed to the facial animation system, which decodes them and uses the values of the FAP to position the feature points of a face. With this stream a flowing animation of several sequences can be achieved as the decoder animates between the key frames of the face.

There are also 2 high level FAPs – one for expressions and one for visemes. Expressions can at most be two out of a list of 6 pre-defined modes such as anger, fear, surprise, neutral. The viseme parameter can contain at most two out of a list of 14 pre-defined visemes. These expressions and visemes are also fed to the animation system, which adds their influence to the feature points. This way the same talk sequence can be made to look angry, surprised, terrified etc.

The actual animation is produced by an animating system decoding the MPEG-4 stream. It receives frames consisting of different FAP and morphs the face in-between frames to animate it.

2.8 Facial motion cloning

If the artists conform to use a MPEG-4 stream enabled facial animation, it is still necessary to animate every single FAP that MPEG-4 defines. This work is tedious and often fairly the same from face to face. Would it not be nice if the artist could just copy these expressions from an already defined face onto a new, just having to fine-tune the new model? A method to create all of these morph-targets called facial motion cloning has been proposed.

Although not stated in the MPEG-4 standard, the FAP could be called morph targets since they define the face in a certain key position, a position in which the artist has to model the face. Facial motion cloning (FMC) copies the low and high-level FAP from one model to another. Defining a subset of MPEG-4 feature points, the motion of one defined face is copied to a new static face. The role of the feature points is to define the correspondence between the two faces. With the feature points defined, the FMC software knows for example where the nose is on both faces. This is then used to copy the facial movements [11].

![Figure 2.4 Simplified description of FMC](image-url)
Lips are critical regions that have to be specially treated. Lips are very close to each other geometrically, but their motions are completely opposite to one another. The result of applying the wrong motion to a vertex would be disastrous. To avoid this, the face is classified in three regions, the upper lip region, the lower lip region, and no lip region. This is done by identifying all the vertices in the triangle between the nose feature point and the inner lip contour (both upper and lower) as shown in Figure 2.5. All vertices that are on the upper side of the base of the triangle (within the triangle) are classified as upper lip vertices. All that are below the base of the triangle are classified as lower lip vertices.

![Figure 2.5 Classifying upper and lower lip vertices](image)

But what if a vertex in the upper lip contour is situated below the base of the triangle? It will be classified as a lower lip vertex, and thus have a downward motion, whilst in reality belonging to the upper lip region.

![Figure 2.6 Erroneous classification of upper lip vertices](image)

Figure 2.6 shows an upper lip assignment that will produce erroneous results. The upper lip vertices between the two feature points are below the base of the triangle, and will thus be classified as lower lip vertices whilst in reality belonging to the upper lip. The cloning process must contain a feature to check the assignment of upper and lower lip vertices.
3 Problem analysis

This chapter first defines the problems and requirements of the project. Then the problem is analysed and broken down into smaller parts.

3.1 Problem definition

As described in chapter 2.8, facial motion cloning is a technique to animate a static face model, creating all the necessary morph targets. This method copies the morph targets from an already animated face model to the new static face. This lifts a great burden of the 3D artists back, not having to animate all morph targets for a face, but now simply being able to copy them.

For the facial motion cloning method to work, the static and the animated face must have certain feature points defined (a subset of the MPEG-4 standard feature point set). Thus arises the need for an editor in which the artists simply can click on the vertex for each feature point, save these vertices to a file and use it in a FMC software.

The FMC method uses VRML to describe the face models. Since the editor will be rendering models that will later be used in a FMC software the format for the face models in the editor will also be VRML. See appendix A.2 for a description of VRML.

The purpose of this thesis project is to design and implement such an editor. The requirements were found to be:

- Windows application
- Use graphical representation of the face model allowing the artist to select vertices as feature points and saving the selected feature points into a .fdp (feature point definition) file that can be used in an FMC software. Use OpenGL for rendering of the face model
- The face model renderer should also be used in other programs, and so the rendering component must be portable
- Since FMC needs correct assignment of upper and lower lip vertices the editor should include a lip region selecting mode
- Implement in VisualC++

OpenGL is explained in Appendix A.1.

3.2 Why is it needed?

The proposed method of facial motion cloning shortens the 3D artists work a great deal. The FMC software produces the necessary morph targets to use the MPEG-4 standard to animate the face model. For the FMC software to work both faces – the animated and the static one – need to have a subset of the MPEG-4 feature points defined. The feature points are stored in a file format called .fdp (feature point definition). These two .fdp files define the correspondence between the two faces needed by the FMC software. In order to produce an .fdp file a graphical tool is needed where the artist can define these feature points and save them to a file.
Without a graphical editor it will be very tedious and complicated work to cut and paste the coordinates from an editor to a file. If then the structure of the face were to change just a little bit the whole procedure would have to be repeated. The editor could have been made as a plug-in to a 3D editor but this has a drawback, that the artist is restricted to one program. By using VRML and a standalone editor, there is no need to know which editor the artist is using.

3.3 Who will use it?
The most common user of this tool will be the 3D artist who designed the face model. This artist is used to highly graphical tools, such as 3D editors and image processing programs. To make the artist feel at ease using the program it must resemble the tools they are accustomed to. The face model must be graphically presented in the same quality as the 3D editors they modelled the face in. As shown in Figure 3.1, a quick glance at a modern 3D editor shows that the same model is often presented in several views, and all views can be rotated, translated and zoomed.

**Figure 3.1 Snapshot of a modern 3D editor**

There is typically also some toolbars that control the objects displayed. The objects can be displayed in several rendering modes and a quick preview is included.
3.4 How will it be used?

When producing 3D animations on a computer it is, as with most other things, never perfect the first time. Since realism is of such importance in facial animation the artists are prone to make at least a couple of test runs before producing the final animation and even then there might be more changes afterwards. Simply put, the tool will be used several times with the same face in search of the perfect animation. So the .fdp editor must quickly reflect the recent changes made by the artist. This motivates the need for realism in graphical representation of the face in the .fdp editor. A small change in a 3D editor must be equally visible in the .fdp editor.

3.5 Analysis of the requirements

When starting to break down the task into a design of a program, some factors must be realized of what the program should achieve. A good start is to look at what has been done before.

3.5.1 Graphical representation of the face model

The first programs to be inspired by are of course the 3D editors that the artists are used to. When modelling an object it is crucial that the artist can rotate, translate and zoom the object. This is mainly to be able to see obscured vertices and small details. When selecting vertices as feature points the same criteria apply as well. It cannot be assumed that all vertices are visible. They might be obscured by some other part of the face. Thus the rotation and translation tools must be included in the editor. Also when picking vertices, there can be a high resolution of the face (many vertices) and the artist can have difficulties distinguishing the right vertex so zooming is also crucial.

In a 3D editor the artist can select the display mode of the face, as for example wireframe, solid or textured. Selecting the display mode would also greatly help in the feature point editor. Selecting vertices in solid or textured mode will be very hard. The artist would be greatly helped by viewing the model in wireframe.

The view has to be interactive, not just passively displaying the face model. When the user clicks in the face model window, this click must be intercepted and examined to see if the user clicked a vertex. If a vertex was clicked, this must be internally stored and the result presented to the user by marking this vertex as selected. The user should be able to, at all times, save the selected vertices.

3.5.2 Two functions – one program

As stated in the requirements the program should have two modes. One part where the user can define feature points for the face model. But, as described in chapter 2.8 there could arise problems when using FMC. Certain vertices could not be classified as upper or lower lip or classified in the region and others could be assigned wrong – and as the correctness of the lip vertices is crucial to the cloning the assignment has to be verified by the user.

Thus arises the need for the second functionality – the upper / lower lip editing part. This can only be done when all feature points have been defined. The user can only be admitted into the lip editor when all other feature points have been defined. Once the user has defined all these there should be an option whether the user would like to check the lip assignment.
It is quickly realized that lip editing is not that different from defining feature points. A face model still has to be displayed, only this time several vertices should be displayed in different colors (depending on whether they are classified as upper / lower or no lip region). The user should only be able to select these colored vertices, and assigning them to the proper region. When this is done the user should be able to save the assignments to the .fdp file.

The underlying structure of the program is almost the same for the two modes. In feature point definition mode, the program has to keep track of the current feature point, and keep an internal list of the already defined feature points. In lip editing mode, the program needs to keep track of which vertices that are classified as upper, lower and no lip region. The user must be able to save and read feature points / lip vertices at all times.

3.6 Programming in a Windows environment

Since VisualC++ and the Windows environment was to be used it is a good idea to look at existing programs and also the VisualC++ IDE (integrated development environment) to get some ideas of how a program is developed.

3.6.1 MFC

As the Windows operating system has grown over the years it has become a rather complex programming environment. To write an application requires a great knowledge in how Windows works. This is not helpful if a programmer needs to quickly develop an application in Windows, not really caring about the operating system functions of the program but rather be interested in the functionality of the program. Some years ago Microsoft launched MFC (Microsoft foundation classes).

When programming in VisualC++ it is hard to avoid using MFC. This is an application framework provided by Microsoft to help the developers program in Windows [12]. Most of the details in a Windows program are taken care of in MFC, letting the programmer concentrate on the functionality of the program. It is often still possible to tamper with every window aspect, but as this is seldom necessary the MFC hides the details. This is useful for programmers who are not experts in Windows and are satisfied with pre-created templates for a program. On the other hand it certainly obstructs alternative designs of programs not complying with the program model used by MFC. It gives development speed in trade-off for flexibility. MFC is supported in VisualC++.

When an empty MFC project is created in VisualC++ template application, menus, toolbars and a document / view is created.

3.6.2 Document / view architecture

When working with MFC it supports (and almost forces) the programmer to use the document / view structure [12]. The document is the non-visible part of the program containing the data the user is editing. For example in a word processor application the document is the object containing the actual text, and methods to affect this text in some way. The view is what the user actually sees on screen. It is a presentation of the data contained in the document. A document can have several views, either multiple views at a time or switching between views when the user wants to see the data in another way.
The main reason to use document / view is the simplicity of having data and the presentation of the data separated. If a good API (application program interface) towards the document is created, the contents of the document can easily be changed without having to re-implement the view. Another advantage is that there can be several views connected to one document. There already exists MFC provided methods to obtain views from a document and vice versa. Thus is communication between documents and views well supported. If a change is made in one view that affects other views, the document will simply obtain these other views and make the change.
4 Components and techniques used

In this chapter the components and techniques used to implement the editor are explained more thoroughly.

4.1 3D graphics

A large part of this project was to render 3D graphics. This chapter describes the fundamentals of 3D graphics.

4.1.1 How 3D graphics works

Displaying graphical images is much like painting a picture. For centuries the artists have known how to make an image appear to have depth even though the canvas has just two dimensions. 3D graphics on computers deal with the same issue, of how to make a 3 dimensional image look realistic on a 2 dimensional surface – the computer screen.

It all comes down to methods fooling the brain that the screen actually appears to have depth, to make objects that are further away from the viewer appear smaller whilst the same sized object closer to the viewer appears larger.

4.1.2 3D models

3D graphics is all about displaying three-dimensional objects on a two-dimensional screen. The 3D objects have to be described somehow. The most common way of describing a model is to use a 3D coordinate system. The model consists of several vertices. A vertex is a point defined in 3D space. These vertices are then linked together to form polygons. A typical 3D object consists of a number of vertices and then a list that defines the polygons.

The numbers of vertices that a polygon consists of vary from program to program. There is an increasing use of triangles, since these have appealing properties when rendering. Most modern graphics cards use an internal format of triangles and if a quadrangle is fed to the card it is internally triangulated.

In the end, all graphics programs comes down to feeding the card with vertices forming polygons which are projected and then showed to the user on the screen.

4.1.3 Different display modes

When displaying a 3D object on a 2D screen, there can be more than one way of depicting the object. When the object is animated all features are of course enabled such as textures, materials and lighting. But when modelling the object this might obstruct the modelling and it also decreases the frame rate. Artists modelling objects often display the object in a mode called wireframe. This is the polygons of the model outlined with only lines.
4.1.4 Hierarchical 3D scenes

Describing a 3D object as a simple list of vertices is acceptable, if it contains up to a few hundred vertices. When modelling, the scene rarely contains one object but rather several smaller objects that compose a scene. Even a face is often modelled as several smaller objects, for example the eyeballs are modelled as separate objects and then placed in the face. All objects could be stored as just a list of vertices, but it is more desirable to have a hierarchical structure of the objects so that they have a logical connection with each other. It is also easier for the artist to model the parts separately and then assemble them. This makes a tree / hierarchical structure of the scene appealing. Objects are situated in local coordinates, where the object is situated around origin. When the object is inserted into the graph scene, the object might be translated, rotated and scaled to fit into the scene. The concept of modelling the object around a local origin and then place it in the real world suits the hierarchical structure. The scene starts at the top or bottom of the hierarchy, then the objects are placed in the real world at the nodes and the actual leafs contain the objects in local coordinate system.

A requirement of the FMC method is that the moveable parts of the face such as tongue, eyeballs etc. are modelled as local objects in the world model [11]. Figure 4.2 shows an example of how a face is described in VRML. The parts of the face are often referred to as <surface>.<vertex> where surface is the current sub-part of the face, for example the tongue can be surface five. Vertex four on the tongue is then referred to as <5>.<4>.
4.2 Implementing an OpenGL renderer

According to the requirements the program should be able to render the face models graphically and allow the artist to select vertices saving them into an .fdp file. Given that the program should be a Windows application written in VisualC++ there was little doubt of making the view of the application into an OpenGL renderer using the document / view structure as described in chapter 3.6.2. This chapter describes how the view was designed and implemented.

4.2.1 Design idea of the view

When developing this program the idea of separating the data and the presentation of the data is very appealing. Placing the .fdp and face model structure in the document whilst letting the view present the face model and manipulate the .fdp through the document ensures good portability for the view, as the requirements stated the rendering module should be easily portable to other programs. As long as the program which the module is to be ported to uses this structure, the view is simply imported to display whatever VRML model the document of that application might contain. Where it was possible the OpenGL calls were made in a separate part of the view so that the actual rendering could be detached from the view.

The view was made very general and not specialized only to the document used in this project. The view can actually be seen as a somewhat specialised VRML renderer.

4.2.2 How a VRML scene is rendered to the screen

The purpose of the OpenGL enabled view is to present graphically the face models to the user. The face models are, as stated in the requirements, defined in VRML. As described in appendix A.2 the VRML is a hierarchical structure also as mentioned in appendix A.1 OpenGL is a state machine. A state machine remembers its last state until a new state is entered. What this means when actually rendering a scene is that when a transform for an object in a scene is entered into OpenGL, this transform is kept until a new transform is issued. This conforms to the fact that VRML nodes inherit its parents transforms.

Before any inputting of vertices is done, the whole scene must be rotated, zoomed and translated. The angle of rotation, units of translation and zooming is what the user inputs with the mouse. This is where the interactivity of the rendering is performed. This transform will affect the rest of the rendering.

In Figure 4.2 the scene starts with a root-node, the body / bust shape. This defines the transform for all its children. The shape is inputted into OpenGL. Next in the tree is the face, which has a transform attached to it. The transform followed by the face is inputted into OpenGL. After the face comes the upper teeth shape. It has also a transform attached to it. So the transform is inputted, but this transform only concerns the upper teeth. When the upper teeth are done, the previous transform has to be undone. The state machine is rolled back one step and the next object is then inputted. This continues in the same fashion until the whole scene is inputted into OpenGL. When then the whole scene have been fed into OpenGL the rendered image can then be shown on screen.
When rendering the face it is sometimes desirable to render just a part of the face. This will for example simplify picking of vertices that are obscured by other objects (for example the tongue which is often obscured by the lips). As discussed in chapter 4.1.4 the VRML file consists of several objects (in VRML denoted as shapes) – each shape describing a separate part of the face. It is now very simple to control the rendering so that only one shape is rendered to the screen. In the example described above we only add a condition that if the part we want to render coincides with the number of the shape, we feed it to OpenGL, otherwise we just skip the shape until the right one is traversed. If the lips obscure the tongue, we simply switch mode and display only the tongue.

### 4.2.3 Selecting vertices

An integral part of the renderer was to intercept clicks of the user selecting vertices. OpenGL does not provide such a mechanism to intercept clicks on a single vertex. In OpenGL there exists something called selection mechanism. This mechanism enables the programmer to label objects and when the user then clicks on an object, the label of the object will be returned. However this mechanism has limitations as it is provided for clicking large objects rather than single vertices. The number of labels is typically only 64 and the face models used by artists will at least contain hundreds of polygons. The selection mechanism was not a good solution. Thus vertex selection had to be done in some other way.

OpenGL contains a function for projecting a 3D point in space to its 2D coordinates onscreen. When loading a VRML model all the coordinates of the model are transformed to world coordinates, see chapter A.2.2, and then stored in a list. When the user presses the mouse button onscreen, the screen position of the mouse cursor is captured. All coordinates in the model are then projected to screen-coordinates using the OpenGL function and compared to the mouse cursor coordinates (values that are off screen are of course first discarded). If a coordinate is close enough to the mouse coordinate it is stored in a hit list. If two or more coordinates are close enough their depth-values in the model are compared and the vertex closest in depth to the viewer is selected. This way the mouse click can be compared to the vertices in the VRML model. We now know the 3D coordinate of the selected vertex and can by this information obtain which surface and vertex in the model this coordinate corresponds to and store them.

As discussed in the previous section we can control which part that is rendered. This can also be applied to the selection of vertices, by adding a condition that the selected vertex must be a part of the rendered shape.

### 4.2.4 Displaying selected vertices

When the user has selected a vertex, this vertex must somehow be highlighted so that the user receives visual feedback on which vertex that was selected. The first approach was to use a 3D object as a marker for the selected vertex with the 3D marker centred on the position of the selected vertex. Since face models can vary in size this 3D marker had to be scaled depending on the size of the face model. The problem with this approach was that when the user then zooms in on the selected vertex the 3D marker is zoomed as well. When close enough the 3D marker would obscure the selected vertex making it difficult to see. So a solution to the problem could be to use a 3D object, but at some point in space close enough to the user start scaling the object so that when zoomed in after this point the object would retain constant size. But this approach is hard and computationally costly and not a good solution.
The solution was to use an OpenGL mechanism of drawing points in 3D space. When drawing a point a constant pixel size can be set. Thus the selected vertex is drawn as a point with constant pixel size no matter how far or close the point is to the viewer. This approach too has problems if the solution of the screen is set very high (for example 1600x1200 or above) the point will appear too small so the pixel size has to be based on the current resolution. Letting the user set the desired resolution of the marker can solve this.

4.2.5 Using the renderer in lip editing mode
As stated in the requirements there was a need for a lip editing mode as well. The only thing that differs the two modes is the selection and displaying of vertices. In feature point editing mode the user can select any vertex in the model and only one vertex can be selected at a time. In lip editing mode the ambiguous vertices are displayed and the user can only select between these.

The view can clearly be re-used in lip-editing mode. The only real difference is that in feature point editing mode a list of all coordinates in the model is kept. When the user selects a vertex this is compared to this list. In lip-editing mode the list consists of upper, lower and ambiguous lip vertices. When the user clicks a vertex it is first checked that this vertex is a member of the lip list. Also all of the vertices in the lip list has to be displayed onscreen allowing the user to select only these vertices. The rest could be re-used.

4.2.6 Problems with different model sizes
When rendering an object, the object must be translated to a sufficient depth so that the whole object is visible. When modelling objects they are often situated around the origin. Positioning the camera at origin will have the user in the middle of the face (inside it), which is not desirable. Thus the model has to be translated somehow to fit into the viewing field when loaded into the editor. Using a constant distance to translate the model is not a good idea since models can greatly vary in size. The solution is to traverse the model storing maximum and minimum coordinates in all three axes. Then the greatest distance between the two extreme points on each axis is calculated. The viewer is then placed a constant times this distance away from the model. Empirical tests showed that a factor of ten produced a good result.

The difference in size also causes problems when translating and zooming the object. Translating 5 units on an axis might be sufficient for one face model whilst another will not appear to move until it is translated 100 units. Thus an acceleration constant depending on the size of the model had to be calculated. The movement of the mouse pointer is multiplied by this constant to translate and zoom the object.
4.3 Implementing the document
This chapter describes the structure of the document used in the project.

4.3.1 Role of the document
Since the application uses MFC and thus adopted the document/view structure it was pretty obvious that the role of the document would be to contain all data and act as a centre for all the views acting upon the data. If a view makes changes in the document, the document makes the necessary updates in other affected views as well.

The document also had to handle all disk activity, reading of face models and .fdp files and storing on disk of the .fdp structure.

4.3.2 Data contained in the document
The primary content of the document is of course the face models for which feature points will be defined. The views obtain the face model from the document and render it to the screen. Also the .fdp structure has to be stored in the document. If a .fdp file already exists for the document, this should be read from disk into the structure. When editing feature points the document also keeps track of which is the current feature point. The document has an internal list of which feature points that has been edited since the last save.

When the user selects a vertex this is transferred from the intercepting view to the document where it is stored internally. If the user accepts this vertex as a feature point this is transferred to the .fdp structure, but if the user fails to select the correct vertex this must be thrown away and a new vertex is selected.

When in lip editing mode the document contains the list of upper, lower and ambiguous vertices. The view obtains these from the document and displays them so that the user can decide weather they are correct or needs re-assigning. If the user selects a lip vertex and assigns it another value, this is transferred to the document that stores the new value.
5 The complete system

In this chapter the user interface of the program is discussed and the views are displayed in action.

5.1 UI Design

Since the task performed by the artist in the editor is quite monotonous the artists will soon get used to the interface and the need for short-cuts to speed up the work is also good. Thus the interface must be designed so that the artists can do the work the fastest possible.

The first part of the editor has the purpose of defining feature points. These feature points are specified according to the MPEG-4 standard and will always be situated on the same region of a face. To simplify for the user where these feature points are situated on a face a guide was included. This guide will give the user a quick hint of where the feature point is situated on a known face. If for example feature point <6>,<1> is edited the user has to remember the position of this feature point by heart. A quick glance at the guide will tell the user that it is the feature point situated on the tip of the tongue. The guide is situated in the top right corner in Figure 5.1.

This guide view shows how easily the OpenGL view can be re-used. The view is based on the OpenGL view rendering a VRML model that acts as a guide. Only the selection mechanism has been disabled in this guide. The user can still zoom, rotate and translate the guide face. The model is stored in the document and can easily be changed to whichever face the user desires as a guide. The only requirement is that the guide face has a valid .fdp file defined. This file is read into the document.

![Figure 5.1 Snapshot of the editor interface](image)

Underneath the guide face are the controls, letting the user step through the feature points and also telling the user which points that has been edited or not.
5.2 The views in action

To simplify the selecting of vertices two rendering modes are included - the image as it is with textures and materials and a wireframe mode.

![Screenshot of the editor displaying faces in wireframe](image)

**Figure 5.2** Screenshot of the editor displaying faces in wireframe

The vertices now appear much more visible to the user. Figure 5.3 shows the editor with just one shape rendered as discussed in chapter 4.2.2.

![Snapshot of the editor with just one shape rendered](image)

**Figure 5.3** Snapshot of the editor with just one shape rendered

This mode simplifies the picking of vertices that are obscured by other parts of the face.
5.3 Lip editing mode
As discussed in chapter 2.8 there is also a need for an upper / lower lip editor where the user can assign ambiguous vertices to upper or lower lip region. In this mode there is no use for a guide face so this view is closed down. Here the versatility of the OpenGL view comes into work using the same view to display ambiguous lip vertices.

Figure 5.4 Snapshot of the lip editing mode

The different colors of the vertices indicate whether they are upper or lower lip vertices. The user can now select only these vertices and assign them to either upper or lower lip region.
6 FDP file structure

This chapter describes the format of the .fdp file structure.

The .fdp structure is based on the MPEG-4 feature point interchange format [5]. In the facial motion cloning process it is required that a subset of the MPEG-4 feature points are known.

6.1 Feature point structure

One of the requirements for the editor was to save the defined feature points into a .fdp file format. This file would then be used in a FMC software. See Appendix B for an example of a file.

In the .fdp file format both the 3D coordinate, and the <surface>.<index> are stored. The main goal with this strategy is to be able to restore a corrupt .fdp file. It would be naïve to assume that a face model is correct and looks perfect the first time. There is bound to be some re-modelling during the course of a facial animation process. But when the face model is altered, what happens with the already created .fdp file? There are three cases when re-modelling the face:

1. The artist adds a number of vertices to the model but does not alter the position of already existing feature points. In this case, the 3D coordinate of the feature point is still intact, but the vertex index is corrupted. Thus the model is re-read by the parsing program and the corresponding vertex to the correct 3D coordinate is stored in the file.

2. The artist re-positions feature points, but does not alter the number of vertices in the model. In this case, the parsing program can re-read the model and store the correct 3D point for each corresponding vertex.

3. The artist alters the defined feature points and the number of vertices in the model. In this case there is bound to be trouble but the whole file might not need to be re-done. The parsing program will go through each feature point and check the model for 3D points and vertex indices. Some indices (up to a certain point, depending on how the 3D editor adds new vertices) can be correct. When an erroneous feature point is found the program will simply ask the user to re-position it.

6.2 Upper / lower lip structure

As discussed in chapter 2.8 there can arise problems with ambiguous vertices when cloning a face. When the face model is tested for ambiguous lip vertices the user can assign them to upper or lower lip region. This must also be stored in the .fdp file for usage in the FMC software.

The first item defined in the file after the feature points is the number of upper lip points. Then follows the number of upper lip vertices and the actual vertices. Using the same analogy as when re-constructing a corrupt file as above, in chapter 6.1, both the 3D coordinate and the <surface>.<index> are stored for the upper / lower lip vertices.
7 Conclusion

The facial animation field including computer animated characters is a quickly expanding field. It is only now that hardware exists that makes it worthwhile to use facial animation. The first standard was adopted just a couple of years ago. Facial motion cloning is an almost unique process. There exists one or two similar methods, but they differ to some extent to the facial motion cloning. Since the method was so new, changes were added during the project that affected the outcome of this project. This chapter will sum up the project, what was good, what could have been done better.

7.1 How the editor conforms to the requirements

The project was a collaboration between Linköping University and the Swedish television (SVT). The aim of the project was to develop an editor to be used in an animation in a television program. This however fell through during the period of the project.

In preparation of producing the animations the editor was tested by a 3D artist at the Swedish television and after solving some minor problems there were successful results produced. The editor was used with a FMC software and a cloning was produced, which implies that both the requirement of graphical representation and saving of the feature points was successful. Also the lip editing part were used and proven successful in the tests.

The OpenGL view was successfully ported to another program with good results. The view was flexible and only minor adjustments had to be done to the new application. Thus the requirement of a portable view was also fulfilled.

7.2 The editor in comparison with other programs

There does not exist any programs to compare this with. The FMC method is as mentioned very new and there has not been any need for an editor of this kind before. If the project provider can settle for one 3D editor, a good solution would be to integrate the feature point editor into a 3D software. This way the artist can both model and define feature points at the same time. Now the artist needs to switch between two programs in order to change the appearance of the face. This will compromise the benefit of having an independent format, such as the VRML and thus settling for just one editor.

7.3 What could have been done better?

One thing that was not completely satisfactory was the vertex selection in solid mode. In solid mode, if vertices not visible can be selected (i.e. behind model) the user has no idea that the vertex is selected. Thus involuntary changes can be made. This can be solved using a polygon hit algorithm. A straight line is projected from the viewer into the scene. If the line hits a polygon and that hit-point has greater (nearer) z-value than the selected vertex, nothing is selected.

Also the vertex selection tests all vertices. If the designer knows that models will be very complex a bounding box test can be made to only test the vertices visible to the user. Alternatively a bounding box can be made to represent the mouse volume, a very thin box (deeper than the model). All vertices within this box are hit-points. The vertex with the nearest z-value is selected.
Also there should have been some testing of the user interface (UI) with lo-fi prototypes (the interface simply drawn on paper). Since the editor is such a highly graphical tool, it is crucial that the user understands the functions. A lot of misunderstandings could have been avoided if a prototype had been used to design the UI.

One drawback with using VRML and OpenGL is that in VRML the user can specify transparency on materials. This does not exist (yet) in OpenGL and is a known problem amongst OpenGL programmers. There are ways to implement transparency, but these are very specific and not so good solutions.
8 Future improvements

The program is by no means perfect. In this chapter some future improvements and extensions are suggested.

8.1 Vertex selection

Since vertex selection is done by comparing the mouse-click to all coordinates projected onscreen there can be a case where the user wants the vertex closest to the camera, but in the model there is a vertex further away that is projected closer to the mouse pointer. In this case it can be difficult for the user to see which vertex that is selected since the depth of the selection is not indicated. This can be solved by for example by highlighting the vertices closest to the selected one while the rest are toned down. Another solution would be that the model could rotate so that the selected vertex comes closest to the user.

Another feature to simplify the selecting of vertices would be to scale the marker of the selected vertex. This should ideally be done such that if the selected vertex is far away it is small, and is zoomed up to a certain point, but then keeps the same size (to avoid obscuring the scene). This is not a simple problem. A first approach would be to use a 3D object that is scaled down after a certain point on the z-axis. But models vary greatly in size and it is hard to set a parameter when to start scaling the object. Scaling it uniformly also involves some heavy math to calculate the projection – to make the scaling factor keep the object in the same size. Through this the depth of the selected vertex would more visible.

8.2 Face model format

There is a need to switch from VRML to a more versatile format. VRML is becoming outdated and will probably in a quite near future be surpassed by other formats. The advantage VRML has today is that it is nearly the only standard that most 3D modelling software supports. One great defect with VRML is that it does not support multiple textures. Today most 3D editors support texture on any material channel – the alpha channel, intensity, ambient etc. But VRML only allows texture on the color channel [1], making it almost obsolete today.

8.3 More options in the program

Little focus was paid to the adaptability of the program. Options in the program would greatly increase the usability. As it is now, the user cannot choose guide model. The user should be able to define a personal order of feature points since it is not necessarily optimal for an artist to just go through the feature points in a straight order. The color of selected vertices cannot be selected. If the face model that the artist happens to model has the same color as the marker of selected vertices it will be very hard to select feature points. More rendering modes where the user could select to turn on and off textures, lighting, materials etc would increase the adaptability. The rate at which the flashing marker is flashing should be an option.
Appendix A  Required components

In this chapter the required components of the project are explained. These components were a choice by the project provider and not considered a part of the solution to the problem. They are presented here to the readers not familiar with the formats.

A.1 OpenGL

3D graphics is today extremely popular. Most of today’s computer games rely on 3D graphics where the user moves in three dimensional worlds. Other areas where 3D graphics is used is in virtual reality, medical software, flight simulators, spreadsheets, CAD, city planning etc. There have even been experiments with 3D window managers. OpenGL (Open Graphics Library) is one of the strongest API standards in 3D graphics.

A.1.1 What is OpenGL?

OpenGL is strictly defined as “a software interface to graphics hardware” [13]. Today’s vast array of graphics cards, all with their own drivers and internal configurations, makes it impossible for a programmer to write a program that conforms to every existing card. Neither can the interface Windows provides harness the power of the graphics cards today. Therefore almost every manufacturer of graphics cards supports the OpenGL API. This API exists for most programming languages.

The API solves the problem with different drivers and protocols for each graphics card. OpenGL provides all the commands and graphics routines commonly used in 3D graphics. The setting of resolution, color format and other nuts and bolts is now made through an API call, leaving the details to the drivers of the card.

The OpenGL is designed to be hardware and operating system independent. This is to simplify porting of programs. This makes it very flexible; both in not having to bind programs to certain hardware, but also not to bind them to a specific operating system or window-manager. The operating system calls are done on top of the OpenGL code, so that these “gates” to the OpenGL can easily be exchanged [13].

Another advantage is that the online OpenGL community is very active. There exists a multitude of examples and free sources on the Internet. Searchable discussion groups exists covering the most of questions – there is always help at hand.

A.1.2 How does OpenGL work?

The most common way of handling a call to OpenGL is that the command is re-made as a series of driver calls to the graphics card. This way the maximum speed is obtained by not using operating system specific calls [13].
For maximum speed, OpenGL is designed as a state machine. A state machine remains in the same state until a command that affects the state (so that it moves to a new state) is issued. Rendering states are essentially flags or integers with some value. The API can be inquired for the specific value of a flag. Commands given affect the forthcoming commands immediately, until a new command changing the state is issued. So if a command enabling texture mapping is issued, texture mapping will remain on until a new command disabling it is given. There are no delays in issuing commands to the OpenGL. Once a command has been issued, it will immediately affect every forthcoming command. There can be several instances of OpenGL running on a single machine. Each instance of the OpenGL has its own state machine and they do not affect each other.

There also exits commands that do not affect the state of OpenGL. These are primarily for input of the 3D object to render. When rendering an object it all comes down to inputting vertices, normals, textures etc. OpenGL then processes all these internally, performing 2D projection, texture mapping, lighting etc. All vertices that are fed to OpenGL are drawn either directly to the screen or into a doublebuffer. When the scene is completely rendered the doublebuffer is transferred to the screen. The usage of a doublebuffer is to avoid flickering while drawing the image.

A.1.3 Other alternatives to OpenGL

There are of course a number of alternatives to OpenGL. One of the strongest contenders on the Windows platform is DirectX. DirectX has most of the features that OpenGL provides. The advantage DirectX provides is when programming for the Windows environment, since DirectX is provided by Microsoft this is more tightly coupled to Windows applications. OpenGL requires more setup to work in a Windows application. The obvious strength that OpenGL has is that it exists for most programming languages and operating systems.

There are more alternatives but these are the two main contenders on the Windows platform.
A.2 VRML
This chapter describes the VRML (Virtual Reality Markup Language) standard and structure.

A.2.1 What is VRML?
VRML has several layers of usage. At its bare core it is a 3D graphics interchange language. In it is defined the most common graphics building blocks. It defines polygons, simple objects such as spheres, boxes etc. Objects are stored in a hierarchical manner, so that a scene can contain several sub-objects that in turn are separate scenes [7]. It has capabilities of defining material, textures, light sources—all of the commonly used tools used in 3D modelling software to build 3D worlds. It is just as HTML, not a compiled language but rather an interpreted language of a VRML viewer—used as a standardized way of describing graphical objects for any given application.

On top of this functionality, VRML has capabilities of integrating sound and text with graphics. The aim of the standard was to produce the graphical counterpart of HTML giving the designer the power to create 3D homepages and more importantly being able to link this world with other virtual words on the Internet. The user can with a VRML viewer move around, interact and explore the objects confined in the 3D world. Just as HTML the user can either look at the source or use a browser [8].

A.2.2 VRML structure
The content of a VRML file is called a (virtual) world or scene. The user can move around in this world exploring it. A world can link to other worlds via the use of URLs. The VRML is a hierarchical structure defining nodes to describe the shape and property of an object in the world. These nodes are the building blocks of a VRML world. There exist a number of different nodes, all from shapes to material of the shape and position of light source and camera. There are no limits as to how many nodes a world can consist of [8]. Everything usually starts with a root node, a VRML file can contain zero or more root nodes [7].

The order of the nodes matters in VRML. The first node found in a file is carried out, and then the next node is read and executed. Since the VRML is a hierarchical structure, a node can have children belonging to it as well. All changes made in the parents node applies to its children, for example if a rotation node has a cylinder and a cube as children nodes—both shapes are rotated according to the parent before drawn. This is referred to as local coordinate system. The changes in the coordinate system from the parents apply to the children. The coordinate system in which the root node resides is called world coordinates [7].

A.2.3 Important nodes
The VRML standard contains many nodes. For this project the two most important ones are the transform node and the shape node.

The transform node is a grouping node that defines a coordinate system for its children that is relative to the coordinate systems of its ancestors. This puts the object with its local coordinate system into the world coordinate system. It contains translation, rotation and scaling around the three axes [7].
The shape node defines only two children, the appearance node and the geometry node. The appearance node contains information about the material and texture of the object. The geometry node contains a description of the shape of the modelled object. In this project an indexed face set node describes the object. The indexed face set describes how a polygonal structure of an object is defined in VRML. Since all faces are modelled in 3D editors using polygons to construct the face, this is the only way a face will be described. VRML also contains many geometry nodes describing basic graphical building blocks such as boxes, lines, spheres etc. but these are not relevant in this project since it would be unrealistic to model a face with these primitives instead of using polygons.

A.2.4 Why use VRML

In this project VRML is used to describe the facial models. This is because VRML is the only real standard way of describing 3D objects. Of course there are a multitude of 3D editor formats but this would limit the editor to a specified format. There exists VRML exporter plugins for most 3D editors enabling the use of one format.

Also the MPEG-4 standard specifies its face models in a standard based on VRML [4].
Appendix B  FDP file format

Outline of the .fdp file format, see chapter 6 for reference.

The syntax of the .fdp file is the following:

```
<float version number> <string file name without extension> <string 3D mesh file name, 0 means unused> <string face texture file name, SGI RGB format, 0 means unused>  
# Possible comments
<int feature point group>.<int feature point> <float x> <float y> <float z> <int surface index>.<int vertex index>  
#the above line is repeated for each defined feature point
```

The lines below are optional, and only there if the file has been tested for upper / lower lip cloning errors, see chapter 2.8 for reference.

```
NUL <int number of upper lip vertices>  
UL <float x> <float y> <float z> <int surface index>.<int vertex index>  
# the above line is repeated for each error-prone (found in testing) vertex  
NLL <int number of lower lip vertices>  
LL <float x> <float y> <float z> <int surface index>.<int vertex index>  
# the above line is repeated for each error-prone (found in testing) vertex
```

Schematic figure of the file format:

```
File header

<feature point> <x coord> <y coord> <z coord> <surface index> <vertex index>
Repeat for each feature point required by FMC

NUL <number of upper lip vertices>

UL <x coordinate> <y coordinate> <z coordinate> <surface index> <vertex index>
Repeat for each upper lip vertex classified by FMC

NLL <number of lower lip vertices>

LL <x coordinate> <y coordinate> <z coordinate> <surface index> <vertex index>
Repeat for each lower lip vertex classified by FMC
```
Example of a short .fdp file (not complete):

# FDP file, created by visage|life software.
1.0 C:\My Documents\test face\Kungen 0 0 0 0 0 0
2.1 -0.465706 1.11888 0.766281 0.21
2.9 -0.0769459 0.9872 1.11523 0.35
3.1 0.349297 1.9445 0.975959 5.41
4.2 -0.0953243 1.9651 1.11924 3.77
6.1 -0.0151413 0.976439 0.959201 7.1
8.2 6.13694=005 0.8769 1.1281 2.356
9.10 0.0428235 1.1128 1.21202 2.365
9.13 0.127852 1.5859 1.11105 4.2
11.5 -0.0004856522 3.48 -0.216998 2.56

NUL 3
UL -0.3525 1.22543 0.005673 0.5
UL 2.97853 1.357345 4.237454 0.10
UL 0.320513 0.652401 1.027583 0.54
NLL 4
LL 0.127852 0.363243 1.002432 0.45
LL 1.201263 1.353637 0.997612 0.46
LL 0.367441 0.534275 0.059986 0.50
LL 2.134992 1.998476 2.001024 0.57
References


På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Jonas Lundberg