Linköping University
Department of Electrical Engineering
Image Coding Group

Editing, Streaming and Playing of MPEG-4 Facial Animations

Master’s thesis by

Piotr Rudol, Mariusz Wzorek

LITH-ISY-EX-3352-2003

Examiner: Robert Forchheimer

Linköping, February 2003

© 2003 Piotr Rudol & Mariusz Wzorek
No part of this publication may be reproduced, stored in a retrieval system, or be transmitted, in any form or by any means, electronic, mechanic, photocopying, recording, or otherwise, without prior permission of the authors.
Computer animated faces have found their way into a wide variety of areas. Starting from entertainment like computer games, through television and films to user interfaces using “talking heads”. Animated faces are also becoming popular in web applications in form of human-like assistants or newsreaders.

This thesis presents a few aspects of dealing with human face animations, namely: editing, playing and transmitting such animations. It describes a standard for handling human face animations, the MPEG-4 Face Animation, and shows the process of designing, implementing and evaluating applications compliant to this standard.

First, it presents changes introduced to the existing components of the Visage toolkit package for dealing with facial animations, offered by the company Visage Technologies AB. It also presents the process of designing and implementing of an application for editing facial animations compliant to the MPEG-4 Face Animation standard. Finally, it discusses several approaches to the problem of streaming facial animations over the Internet or the Local Area Network (LAN).
Abstract

Computer animated faces have found their way into a wide variety of areas. Starting from entertainment like computer games, through television and films to user interfaces using “talking heads”. Animated faces are also becoming popular in web applications in form of human-like assistants or newsreaders.

This thesis presents a few aspects of dealing with human face animations, namely: editing, playing and transmitting such animations. It describes a standard for handling human face animations, the MPEG-4 Face Animation, and shows the process of designing, implementing and evaluating applications compliant to this standard.

First, it presents changes introduced to the existing components of the Visage|toolkit package for dealing with facial animations, offered by the company Visage Technologies AB. It also presents the process of designing and implementing of an application for editing facial animations compliant to the MPEG-4 Face Animation standard. Finally, it discusses several approaches to the problem of streaming facial animations over the Internet or the Local Area Network (LAN).
We would like to take this opportunity to thank a number of people for their support in this work. Their contribution, not only scientific, has been a great help for us when working on this thesis and during the long hours spent in the Fo-Lab 3C-501.

Jörgen Ahlberg, our supervisor, for the Friday meetings and numerous discussions about the thesis as well as many other topics.

Igor Pandic, our supervisor from Croatia, for a number of e-mail discussions and for sharing his ideas with us.

We would also like to thank our examiner Robert Forchheimer for guiding us through the process of preparing the thesis.
# Table of contents

1 Introduction  
   1.1 Overview and the report structure ................................................................. 1  
   1.2 Project provider .............................................................................................. 1  
   1.3 Introduction to the Visage|toolkit software package ....................................... 1  
   1.4 Task formulation ............................................................................................ 1  
   1.5 Commonly used abbreviations ...................................................................... 2  

2 Background  
   2.1 Model-based coding of video sequences .......................................................... 5  
   2.2 The MPEG-4 Facial Animation standard ......................................................... 5  
   2.3 Programming in the Windows environment ..................................................... 7  
      2.3.1 The Document / View architecture ......................................................... 8  

3 New Visage|toolkit  
   3.1 Task analysis ................................................................................................ 11  
   3.2 Functionality specification ............................................................................ 11  
   3.3 The Visage|toolkit application structure ....................................................... 12  
   3.4 Implementation description .......................................................................... 13  
   3.5 Implementation, evaluation and conclusion .................................................. 14  

4 Visage|edit  
   4.1 Task analysis ................................................................................................. 15  
   4.2 Prototyping ..................................................................................................... 16  
   4.3 Functionality specification .......................................................................... 19  
      4.3.1 Detailed functionality requirements for the tree view ......................... 19  
      4.3.2 Detailed functionality requirements for the high-level view ............... 19  
      4.3.3 Detailed functionality requirements for the low-level view ............... 20  
      4.3.4 The use of virtual track ....................................................................... 20  
   4.4 Description of the user interface of the Visage|edit application ...................... 21  
      4.4.1 The Tree View ....................................................................................... 22  
      4.4.2 The High-Level View .......................................................................... 22  
      4.4.3 The Low-Level View .......................................................................... 22  
      4.4.4 Preview View ....................................................................................... 22  
   4.5 Application object structure ....................................................................... 22  
      4.5.1 Overall object structure ....................................................................... 22  
      4.5.2 Document / view object structure ....................................................... 23  
   4.6 Implementation evaluation and conclusion .................................................. 24  
   4.7 Future work .................................................................................................. 24  

5 Streaming of facial animations  
   5.1 Introduction .................................................................................................. 25  
   5.2 Task Analysis ................................................................................................. 26  
      5.2.1 Overview ............................................................................................... 26  
      5.2.2 Real-time vs. pseudo-streaming mode .................................................. 28  
   5.3 Functionality specification .......................................................................... 30
5.4 Prototyping ........................................................................................................31
5.5 Simple MPEG-4 Protocol (SM4P) ....................................................................31
  5.5.1 Underlying protocols and the SM4P functionality ....................................31
  5.5.2 SM4P data packet format ........................................................................33
5.6 Application object structure ...........................................................................34
5.7 Implementation and evaluation .....................................................................39
5.8 Conclusion .......................................................................................................41
5.9 Future extensions ............................................................................................41

6 Summary and Conclusion 43

Appendix A: Simple MPEG-4 Protocol (SM4P) implementation 45
  A.1 Available queries to server and responses .................................................45
  A.2 Object structure .........................................................................................46

Appendix B: Facial Animation Parameters (FAPs) specification 47
  B.1 Facial Animation Parameters (FAPs) .........................................................47
1 Introduction

This chapter briefly presents the structure of the thesis and gives information about the project provider.

1.1 Overview and the report structure

The thesis deals with developing applications for model-based coding of human faces based on the MPEG-4 standard.

The thesis can roughly be divided into these main parts:

- Short introduction to MPEG-4 facial animation.
- Problem analysis. Description of existing applications and implemented functionality. Discussion on possible solutions.
- The new Visage|toolkit application.
- The Visage|edit application.
- Streaming of MPEG-4 facial animations.

1.2 Project provider

The thesis has been prepared for the company Visage Technologies AB [3] in cooperation with the Image Coding Group, Department of Electrical Engineering, Linköping University.

1.3 Introduction to the Visage|toolkit software package

The package includes visual speech synthesis, video tracking, face animation editing, plug-ins for major 3-D software packages, a real time animation module for live performances or for integration in other products, unique features such as automatic morph-target cloning and plugin-free web publishing, as well as customizable web applications such as talking email, virtual newscasters and intelligent virtual hosts. For more detailed and up-to-date information see Visage Technologies AB web page [3].

1.4 Task formulation

Although the Visage|toolkit* constitutes a full software solution for creating, editing, storing and transmitting animations using the MPEG-4 Face Animation standard, the need for more sophisticated editing possibilities arose along with the necessity to improve some aspects of the existing software.

* Visage|toolkit is the name of the package and of one of its components.
The main task of the thesis was to extend the functionality and improve usability offered by the existing software. In order to make the work as efficient as possible, identification of the shortcomings was necessary. The main concern about the applications was related to limited editing options. Editing operations that could be performed included:

- Composing new animations of several existing ones.
- Each animation could have a filter file associated with it.
- Operations introduced by filter files included: adding offset, changing speed, applying loop, modifying values of FAPs by setting multiplication factors.

A filter file had a form of a separate text file describing mentioned operations. No graphical manipulation was possible, which is desired nowadays with WYSIWYG (What You See Is What You Get) user interfaces.

Taking under consideration all limitations and expectations, an idea of two separate applications emerged. The first is the animation player and a binding element of the package. The second is an advanced animation editor with possibility to perform all operations in a graphical way.

Another shortcoming of the existing software components of Visage|toolkit was inefficient memory management. Some parts of the code were transformed from plain C to C++ programming language without taking care of appropriate memory allocation/deallocation procedures. The resulting memory leaks were not problematic at that stage, but made it difficult to extend existing functionality.

Additionally, several objects are shared across many elements of the Visage|toolkit software package and it was desired not to change those elements so that other parts of the software (which are not subject of this thesis) would not have to be changed.

Finally, apart from extending existing components, several experiments regarding animation transmission were needed to test the possibility to use the software components in new applications. The main goal was to investigate possibility to establish a real time and pseudo-streaming communication systems based on the MPEG-4 Facial Animation standard.

1.5 Commonly used abbreviations

- kbps – kilobits per second, \(2^{10}\) bits per second
- kB – kilobyte, \(2^{10}\) bytes
- FAP – Facial Animation Parameter
- FPS – Frames Per Second
- GUI – Graphical User Interface
- MB – megabyte, \(2^{20}\) bytes
- MDI – Multiple Document Interface
- MFC – Microsoft Foundation Classes
- RTP – Real-Time Protocol
SDI - Single Document Interface
SM4P – Simple MPEG-4 Protocol
TCP – Transmission Control Protocol
UDP – User Datagram Protocol
This chapter describes the concept of the model-based coding and gives an introduction to the programming environment used during the software implementation.

2 Background

2.1 Model-based coding of video sequences

In traditional video coding techniques, images are transmitted pixel-by-pixel, or by coefficients describing the waveform of the images constituting the animation frames. This approach gives very good image quality in many applications, but produces high bitrates of data to be transmitted.

In a model-based coding scheme, the image is handled as a 2-D projection of 3-D objects in a scene. To achieve this, parameters describing the object(s) are extracted, coded and transmitted (e.g. Facial Animation Parameters – FAPs). To achieve satisfactory level of fidelity to the original image a texture can be also transmitted. For facial animations using certain techniques, bitrates lower then 1kbit/s can be achieved [1][18].

A general scheme for a video system utilizing model-based coding can be described as follows: At the encoding side the image from the camera is analyzed, using computer vision techniques, and the relevant object(s), for example a human face, is identified. A general or specific model is then adapted to the object - usually the model is a wireframe describing the 3-D shape of the object. At the receiver side of the system, the parameters are decoded and the decoder’s model is modified accordingly. The model is synthesized as a visual object using computer graphics techniques [1].

To put it in simple words and give a down-to-earth example, instead of transmitting information about every pixel of an animation that presents a bouncing ball, using model-based coding it is enough to send information about changes of position of the ball in 3-D space (a model representing shape and texture can also be transmitted). The resulting visual effect is similar, or at least the decrease of volume of data to be transmitted makes it a very interesting technique.

2.2 The MPEG-4 Facial Animation standard

MPEG-4 is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group), the committee that also developed standards known as MPEG-1 and MPEG-2. From the very first version of the standard (MPEG-4 version 1) it includes
definitions of how to decode parameters for facial animation. MPEG-4 version 1 became an international standard in early 1999 [4].

The MPEG-4 Facial Animation standard specifies a face model in its neutral state (Figure 2-1). Additionally, a number of feature points (FPs) as a reference is defined. The standard also specifies facial animation parameters (FAPs), which move the FPs producing the animation. The face model (called the ‘facial animation object’) can be used to render an animated face. The main purpose of the FPs is to provide spatial reference to specific positions on a human face such as major muscles, bones etc. The MPEG-4 standard defines 84 FPs (Figure 2-2) and 68 FAPs.

* Feature points affected by FAPs
© Other feature points

Figure 2-1 A face model in its neutral state.

Figure 2-2 The facial feature points (FPs) as defined by the MPEG-4 standard.
These 84 points were selected to reflect facial anatomy and motion of a human face. Feature points are arranged in groups like cheeks, eyes, head etc. For details about grouping and more information about FAPs, see Appendix B.

The values of a certain parameter (e.g. FAP number 50 head_roll) changing over time, can be presented in form of a plot, as illustrated in Figure 2-3.

![Figure 2-3 Example of head_roll FAP value plot.](image)

To sum up, the MPEG-4 Face Animation defines a framework, which allows employing model-based coding of human faces in a systematic and standardized manner.

### 2.3 Programming in the Windows environment

The software, which have been produced for this thesis has been implemented in the Microsoft Visual C++ 6.0 IDE which enables programmers to take full advantage of the Microsoft Windows framework.

Programming in Windows requires a great deal of knowledge about the system’s structure. It is a very large and compound environment, which offers a lot of possibilities. In the beginning however, it requires a notable amount of time to get used to and to be able to fully utilize its potential. Such an approach is often very frustrating for beginner programmers since it is quite hard to write an application in Windows, without caring about how the system works. On the contrary, after
acquiring sufficient level of knowledge about the environment, it allows the programmer to develop advanced applications in a quick and efficient manner.

To help beginners but also to add systematic structure to the framework, Microsoft provides the Microsoft Foundation Classes (MFC). They provide an intermediate layer on top of the operating system to hide many aspects of programming in Windows. MFC include application templates ready to be used for specific kinds of applications. On the other hand, in applications not following the MFC model it is often very tiresome to fit them into a template. Such an approach gives an obvious trade-off between the speed and flexibility. In most cases however, a suitable compromise can be found, and the MFC support is in many aspects very useful.

2.3.1 The Document / View architecture

Applications created with MFC almost always use the document/view architecture, because it provides a very useful separation between data and its visual representation (it is also possible to implement applications based on dialog windows, but in practice it is used only for very simple programs). A document object is used to store and manage data, whereas a view constitutes a graphical representation of the data. This scheme is illustrated in Figure 2-4.

![Figure 2-4 Dependencies between main objects in SDI Windows applications.](image)

The picture schematically shows how an application in the Windows environment works. The Frame Window Object (also called main frame) is the main graphical element that “holds” view(s). The View Object is a graphical representation of the data stored in the Document Object and it is embedded into the Main Window Object (class CFrameWnd or derived). The main frame of the application takes care of interaction with the user via menus, toolbars, shortcut keys etc. Users interact with the document data through the View Object (class CView or derived). An example of this scheme is a text editor where the view object takes care of keystrokes and modifies the document data accordingly.

There are two types of document/view architectures in the MFC framework – Single Document Interface (SDI) and the Multiple Document Interface (MDI). The first operates on one document at a time (e.g. the Windows Notepad) and the latter uses the main frame window as a workspace in which the user can open zero or more
document frame windows, each of which displays a document [2] (e.g. the Microsoft Office 97 and earlier versions). SDI is mainly used nowadays even for applications working on multiple documents. In such situation, multiple instances of the SDI application are used.

Additionally, a document can have multiple views. This feature can be exploited, for example in a database application. One view presents the data in form of a table and the second in form of a graph. Functions for switching and managing views are supplied.

In conclusion, the MFC is a very flexible framework suitable for many applications. It gives enough skeleton code for the beginners and can be modified by more experienced programmers [14].
This chapter describes the design process and changes introduced to the Visage|toolkit application. It focuses on identification of main shortcomings and presents the chosen solution.

3.1 Task analysis

The Visage|toolkit application constitutes a central part of the software package. Although its task is very simple (i.e. playing animations) it forms a metaphor of a Wizard from which users can launch other applications of the toolkit.

3.2 Functionality specification

The main functionality requirement for the Visage|toolkit application is to be simple but powerful, both in terms of easy interaction with a user and robust program structure. The main requirements and needed changes can be summarized as follows:

- New simplified application structure taking advantage of the Single Document Interface (SDI).
- Improved graphical user interface (GUI).
- Drag-and-drop functionality, all supported file types should be possible to open this way.
- Ability to browse through the animation using a slider.
- Displaying information about loaded files.
- Quick launch functionality – combo boxes for choosing from available animation and face model files in the current directory.
- Different views – full screen mode and compact mode with basic controls.
- Popup menu for quick access to most of the functions.
- Configuration parameters stored in the Windows registry for easy sharing with other components of the software package.
3.3 The Visage\toolkit application structure

Although the original Visage\toolkit application (Figure 3-1) was based on the Single Document Interface (SDI) it did not use the structure properly. For example the Document Object, which plays a very important role in the SDI framework, was not used at all. Thus, the original application did not exploit almost any of the advantages of the interface functionality. Instead, the main application played the role of the document. This approach made the structure simpler but at the same time disorganized and harder to extend. As a result the Visage\toolkit application had to be restructured and rebuilt to fully exploit the SDI architecture.

![Diagram of original Visage\toolkit application structure.](image)

According to the SDI framework described in Chapter 2, the main workload should be handled by the document and the view objects. The main responsibility of the Application Object is to create, manage and destroy those objects. See also [7][17].

The Document Object manages data, for example loaded animation or a face model. The view objects display the results of the operations on the data handled by the Document Object. Thus, the new structure can be illustrated as in Figure 3-2.

The general object structure of the application is matched to the MFC framework’s document/view interface. The application takes advantage of all features offered by the framework. The document holds objects for handling application data (e.g. the FAPlayer object that manages face models and animation files). Example of the View Object is COpenGLWnd, which displays the 3-D model.
This approach ensures that extension of the application functionality can be made in an easy and organized way. The object structure allows proper method calling without unnecessary interaction with the main application object. The work is done between document, main frame and view(s) objects.

3.4 Implementation description

The main window of the application consists of two parts (Figure 3-3). The central view is the OpenGL rendering engine for displaying the 3-D model. The second holds controls for managing the playing functionality. The status bar shows the information about played track, such as: duration, frame rate, name etc. Combo boxes in the main toolbar constitute the quick launch functionality. Additionally, both animation and face model files can be loaded into the player using drag-and-drop technique.

The main menu can be used to launch other components of the Visage|toolkit software package. Right click on the OpenGL window brings up a pop-up menu which enables users to access most of the functions of the Visage|toolkit application.
3.5 Implementation, evaluation and conclusion

Almost all functionality requirements were fulfilled. The only aspect of the application that needs to be extended in the future is the method of playing animations. The current one uses a pseudo threading technique (additional thread is not used but system messages are handled while playing animation, thus the application does not freeze). Although doing its job very well, it is hard to extend into a more flexible solution. For this reason, it is not possible in the current implementation to browse through the animation. A possible solution for this problem is the implementation of the playing functionality using multithreading. It would demand separate threads for controlling the rendering process and for handling the sound. Threads would be synchronized to each other and would be independent from the main application. This solution introduces advanced timing issues of threading in the Windows environment. Standard implementation of the Sleep method does not have enough resolution for the thread synchronization. Executing the command Sleep(1) can put a thread to sleep for five to ten milliseconds (instead of 1 millisecond), depending on the operating system version. This problem however can be overcome with the help of articles [15][16].
4 Visage|edit

This chapter describes the design process, stages of implementation and the evaluation of the Visage|edit application – editor of face animations compliant to MPEG-4 Facial Animation standard.

4.1 Task analysis

During the design process of the Visage|edit application, the following aspects were considered:

- Identification of a user and his/her needs.
- Suitable data representation and manipulation techniques.
- Required/desired functionality specification.

It has been decided that a user is not required to be familiar with the MPEG-4 Facial Animation standard, although knowledge of computer (Windows) environment is required. The idea behind this assumption was that a person dealing with for example computer graphics should be able to use this software without additional learning. The reasoning about the rest of the aspects of the software design was conducted according to the problem/solution scheme. The main problems were identified and solutions proposed. The most significant are presented below with reasoning about each of them:

Problem 1: A frame in the MPEG-4 Facial Animation standard is described by 68 parameters values (FAPs). An animation consists of a number of such frames. From these facts it is obvious that manipulation of animations can be done in two dimensions. First of all, it is possible to change the duration of an animation – change number of frames. Additionally, values of FAPs can be modified.

Solution: A suitable representation of such data is a plot of values. However, displaying 68 plot curves on the same plot makes it impossible to interact with them in a clear way. Some mechanisms for selecting parameters to edit should be presented. The plot should be easily scrolled and zoomed/shrunk in case of large animations (with large numbers of frames).

Problem 2: Animations can be modified or created from scratch. It should be possible to fine tune existing animations (e.g. closing mouth) and create new tracks (e.g. animation with head movement).

Solution: Advanced methods of curve shaping should be introduced for easy manipulation in both scenarios.
Problem 3: From the user’s point of view, it is desirable to make the interaction with the editor easy for the beginners and powerful for the advanced users. Moreover, for so-called “expert users” additional enhancements are needed to make the work with the Visage|edit fast and efficient.

Solution: It is desirable to make it possible for a user to work on different levels of detail. Beginners should be able to work almost without former knowledge about the MPEG-4 Facial Animation standard. Advanced users, who are familiar with the standard, should be able to take advantage of it and to work on a higher level of detail – directly on FAP values. Expert users, who know both the standard and the applications, should be able to use key shortcuts and some degree of task automation.

Problem 4: Besides editing a single animation, it should be possible to edit a few animations in parallel and merge them. A typical scenario is adding predefined head movement (or eye blinks) to existing animation produced by the available Text-to-Speech function, which creates only animation of mouth area.

Solution: A notion of project or workspace should be introduced to enable users to edit multiple animations concurrently in order to produce a single animation, which would be the result of a merging.

Problem 5: It is desired to change (extend) the existing solution of manipulating of the tracks, which uses filter files to apply manipulations such as offset or multiplication factors. The idea of a separate filter file needs to be changed, or a user interface should be introduced to allow graphical manipulation of the values.

Solution: A new concept of multiplication factors should be introduced. The factors can be applied on separate FAPs, on groups of FAPs, and also on the track as a whole.

Problem 6: In a scenario when the user scales the track and then tries to edit FAP values, the scale needs to be applied (the number of frames should be changed) to the track before allowing editing. This also applies to other manipulations, e.g. offset, reverse, or loop.

Solution: An option to apply changes (separate ones or all at once) should be provided.

All problems and solutions described above (and many other) were taken under consideration when going into the next step of the design process.

4.2 Prototyping

During the early stages of the project, low fidelity prototypes were prepared and evaluated. Later on, many high fidelity prototypes were implemented and tested. Screenshots of several prototypes are presented in Figures 4-1 to 4-4.
Figure 4-1 Low fidelity prototype showing the main concept of the Visage|edit application.

Figure 4-2 High fidelity prototype for testing plotting area and range selection.
Figure 4-3 High fidelity prototype for testing track merging and applying filers.

Figure 4-4 High fidelity prototype for testing curve shaping using Bézier functions.
Some elements of the user interface were dropped after evaluation, for example a double slider for selecting a range of frames to be displayed, which turned out to be unnecessarily complicated. A simple solution with a single slider proved to be sufficient. For similar reason shaping of the curve using Bézier functions was rejected from the final product. The phase of prototyping, although time consuming, was necessary to fit the software specification to users’ expectations.

4.3 Functionality specification

After evaluating several approaches, a detailed functionality specification was decided. The basic idea was to divide the application window into four parts: tree view, high-level view, low-level view and a view showing a face model for previewing the changes.

In order to enable users to manage a project consisting of several animations, the idea of a tree view was introduced. It makes it possible to display information about several tracks on different levels of detail at the same time. Additionally, the tree is a common reference for both high- and low-level views – information shown in the two latter views corresponds to options chosen in the former.

It has been decided that users should be able to work in high- and low-level modes. The high-level mode allows editing tracks without going into details of FAP values. This mode can be used to manipulate already existing tracks. The obvious example of this mode is merging two animations in order to add eye blinks to a track obtained from the Visage|track application (which does not track the eyelids). Operations that can be performed in the high-level mode include adding offset, scaling, looping and reversing the track and additionally this mode allows applying multiplication factors on all FAPs, groups of FAPs and, on separate FAPs. The final value of the multiplication factor for a separate FAP is the product of the three multiplication factors and a corresponding factor from the “virtual track” (see Paragraph 4.3.4). The low-level mode allows direct manipulation of FAP values. This mode is useful for fine-tuning the parameters of an animation and when “producing” new animation tracks, for example creating an eye blink track from scratch. Such an eye-blink can be looped to produce the eyelids’ movement for the whole animation.

4.3.1 Detailed functionality requirements for the tree view

Detailed functionality requirements for the tree view include the following:

- The tree constitutes a reference for the high- and low-level views.
- For better visual effect, every second track and its parameters are highlighted with a colored background.
- Multiple FAPs can be selected at a time.
- If more than one FAP is selected, each of them is assigned a different color.
- FAP names and grouping is done according to the MPEG-4 specification (see Appendix B).

4.3.2 Detailed functionality requirements for the high-level view

Detailed functionality requirements of high-level view include the following:

- Information shown in this view corresponds to the options selected in the tree view.
• Graphical manipulation of such parameters as: offset and scale of a track, manipulations are possible both by sliding a bar representing the track and by typing values into corresponding edit boxes.
• Possibility to set loop and reverse modes using checkboxes.
• Setting loop and reverse modes results in changing colors for better visual effect.

4.3.3 Detailed functionality requirements for the low-level view

Detailed functionality requirements for the low-level view:

• Setting the range of frames to be shown on the plot.
• Setting the range of values of FAPs to be shown on the plot.
• Scrolling of the plot (also possible by dragging the plot area).
• Scaling the plot to fit all values in the visible area.
• Displaying/manipulating of multiple FAPs at a time.
• “Pulling neighbors” functionality – when a point representing a FAP value for a certain frame is moved with a mouse cursor, the neighboring points are also moved. The value of drag changes according to the distance from the selected point. This gives a feeling of pulling neighboring points. It is possible to set the “strength” of the pull – if set to “zero” – only one point is moved, when set to “one”, points representing all frames are moved. Values in between give the effect of pulling.
• “Pulling neighbors” in all displayed curves.

4.3.4 The use of virtual track

The reason for introducing the virtual track was twofold:

• It “holds” all the tracks together – setting the length and FPS (Frames per second) rate of the edited animation.
• Allows applying multiplication factors for all tracks in the project from one place.

First of all, there needs to be a way to set timeline of the project. It is necessary for example when a track is looped. In such a case, the length of the virtual track sets the boundaries for the loop operation. Additionally, the FPS parameter needs to be the same for all the tracks in the project. Moreover, the possibility to apply multiplication factors in the virtual track allows applying factors for all tracks in the project without changing them in all tracks separately. If a project consisted of ten tracks and a user wanted to “turn off” head movement in all tracks, it would be a time consuming operation.

Virtual track is the root of the tree and additional tracks of the project constitute leaf nodes. It has its representation in form of a bar, which is not editable – no sliding or resizing is possible. Additionally, editing of FAP values in low-level view is not allowed since the values are built up from values from all tracks in the project.

The use of virtual track in a project with two tracks is presented in Figure 4-5.

After specifying required functionality, the application has been implemented.
4.4 Description of the user interface of the Visage|edit application

The user interface consists of four views as shown in Figure 4-6.

![Figure 4-6 Main application window of the Visage|edit.](image-url)
Each view can be shown/hidden or resized for better screen area utilization. In the case of large monitors or multi-screen configuration, all views can be shown at the same time allowing efficient editing. Detailed description of each view is presented in sections 4.4.1 to 4.4.4.

4.4.1 The Tree View

The Tree View is used for hierarchical selection. The first node of the tree is a virtual track. This track constitutes a reference for the project being edited. Additional nodes represent tracks included in the project. The use of hierarchical tree enables user to select FAPs on the level of track, groups of FAPs and also separate FAPs. Additionally, it makes it possible to show information for different levels in different tracks on the screen. Items, which are displayed for informational purpose only, are “disabled” so that the modification is not possible (e.g. FPS rate).

4.4.2 The High-Level View

The High-Level View shows information in rows according to items visible in the Tree View. Each track in a project is represented by a bar, which can be moved (changes offset) and resized (changes scale in time) using mouse. To set or unset loop and reverse options a checkbox is displayed. To enter multiplication factors text fields are supplied.

4.4.3 The Low-Level View

This view consists of a plot area, which can be scrolled using scroll bars or by “dragging” the plot area with Shift key pressed. There are two directions of zooming. Vertical zoom allows setting the range of FAP values to be shown. Horizontal zoom allows setting the range of frames to be displayed. There is also a “rectangle zoom” option, which zooms the view to the selected area (using Ctrl key).

4.4.4 Preview View

Preview View displays a 3-D model. All changes made in the High- and Low-level views are instantaneously reflected in this view.

4.5 Application object structure

Analysis of the object structure of the Visage|edit application can be divided into two analyses. First of all, the general structure describes the overall organization of an object without going into details of specific items. Second of all, a more detailed look at the document/view object configuration is introduced.

4.5.1 Overall object structure

The general object structure of the application is matched to the MFC framework’s document/view interface. The application takes advantage of all features offered by the framework. The Visage|edit uses the Single Document Interface (SDI) and four views to allow an easy and efficient data manipulation according to the functionality specification. The general object structure is illustrated in Figure 4-7.
The main actors in the scene are: the Document Object for managing data, View Objects responsible for presenting and manipulating data, Main Frame Object for managing View Objects and the Application Object, which holds all the elements together. The main work is done between the document/view couple, whereas the Application Object is used mainly for setting up, creating and destroying those objects.

4.5.2 Document / view object structure

The Document Object includes the FBAEditor object, which consists of several FBATrack objects. FBATrack contains FAP values for every frame and methods for manipulating those values. The manipulations include looping, reversing, scaling, adding offset and applying multiplication factors. Several FBATracks are grouped together in the FBAEditor object, which is also responsible for playing the animations. The FBAEditor object implements methods for track organizations, such as adding, removing or merging. Manipulations are applied on every FBATrack separately. It is worth to notice that the FBATrack object wraps around an FBAAction object used originally by Visage|toolkit applications. Although such wrapping has implications in application performance, such approach does not require the original code to be rewritten.

View Objects are created once per application lifetime and even if the Document Object is exchanged, the Views are reused.
4.6 Implementation evaluation and conclusion

The required functionality was fully implemented. The user interface was designed to take full advantage of the MFC framework. This includes access to application functions by many ways, e.g. from menu, toolbar or keyboard shortcuts.

Methods used for manipulation of FAP values are sufficient for easy and efficient editing of animations compliant with the MPEG-4 Face Animation standard.

The application structure allows unproblematic extension of the functionality. Proposed and implemented object architecture proves to be robust.

4.7 Future work

Although the implemented application fulfills the requirement specification, additional features can be implemented to enhance the functionality of the software:

- sound manipulation,
- introducing flexible mechanism for additional manipulations,
- interpolation of values when scaling,
- copy/paste/cut clipboard operations,
- undo/redo functionality.
In this chapter, investigation of possible streaming systems is discussed. Looking from network point of view, feasible solutions are proposed and implemented. The general discussion is made on modes of streaming used in the experiment and their delay constraints.

5.1 Introduction

The concept of streaming media is less than a decade old and yet it has experienced impressive growth. Millions of people use the available streaming technology to watch live sport events or movies. The increase in available connection bandwidth makes the streaming more and more popular [8].

The MPEG-4 facial animation has many potential applications considering streaming media. To point out some of them:

- Intelligent human-like user interfaces.
- Talking, interactive virtual characters.
- Personal communications, messaging.
- Teleconferencing.
- Broadcasting (e.g. news).

The main goal of the “network” part of this thesis was to investigate potential ways to stream facial animations over the Internet or a Local Area Network (LAN). It includes a few experiments on different solutions based on existing libraries.

The purpose of the experiment was to implement two versions of applications operating in two modes of streaming described later in this chapter. One of the applications would use the existing Visage|track as a real-time source of facial animation data. The Facial Animation Parameters (FAPs) values could be extracted from a sequence of captured images or a real-time video sequence.

The second application should use existing facial animations stored in files and operate in pseudo-streaming mode.

Both applications should use the Simple MPEG-4 Protocol (SM4P), developed during the experiment (Paragraph 5.5).
5.2 Task Analysis

5.2.1 Overview

**Problem 1:** Choosing a protocol for streaming MPEG-4 facial animations. In order to send data across the network a protocol should be used. This includes basic communication schemes between server and clients, requests, replies and data packet format.

**Solution:** One possibility is to use one of the existing protocols such as the Real-Time Protocol (RTP) [11]. The RTP is well documented, has lots of features and is a well developed, verified solution. It can be used in many streaming media applications, which makes it very universal.

To fully examine and understand possibilities of streaming MPEG-4 facial animations the need for a more specific protocol arose. The new protocol would focus more on the streaming of MPEG-4 facial animations.

The Simple MPEG-4 Protocol (SM4P) was proposed and developed.

The SM4P protocol is an application protocol (Figure 5-2) that uses User Datagram Protocol (UDP). More details on the implementation are given in Paragraph 5.5.

![Figure 5-1 System architecture.](image-url)
**Problem 2:** The need for robust application architecture.

Time is the main constraint in streaming media applications. The extended discussion on delay is made in Paragraph 5.2.2.

**Solution:** To maintain good time utilization, multithreading can be used. [17]

A Possible scenario includes two threads – one which handles the connection with the peer and the second, which deals with rendering of received frames. The first one is called the Connection Handle Thread and the latter the Render Thread.

Threads access shared Receiving Buffer.

The Connection Handle Thread has the following tasks:

- Receive packets and write them into the Receiving Buffer.
- Sort received packets (possible to implement in pseudo-streaming mode, not considered to be included in the experiment).
- Control the Receiving Buffer to avoid overloading, by sending certain messages to its peer (server).

The discussed solution, based on two threads is sufficient and demands only simple thread synchronization. One could consider putting one more thread, which would deal with packet ordering in the Receiving Buffer – since the underlying protocol is User Datagram Protocol (UDP), which does not guarantee delivery in order [9][10].

The main drawback of such a solution is the more complicated thread synchronization. Details of the chosen solution are presented in Paragraph 5.6.

**Problem 3:** Shortcomings of the existing MPEG-4 codec.

The existing implementation of an MPEG-4 facial animation codec consists of two parts: encoder and decoder. Both of them have several problems:
1. Encoder:
   - Initialization can be made only 255 times – after that the application crashes.
   - Poor performance of coder using existing configuration files.
   - Tiresome to configure using encoder configuration file (.epf).

2. Decoder:
   - Memory allocation problems. For each run, the decoder allocates 3-4 MB of memory and does not release it. For example, when the coder decodes 50 packets, each containing 10 frames, the average memory allocation reaches 200 MB of memory.
   - Decoding time. Destroying the decoder object in some cases takes more than 120 milliseconds. This corresponds to the amount of time needed to display approximately 3 frames (at 25 frames per second).

**Solution:** The most important problems to be solved are memory allocation in the decoder and the time needed to destroy the decoder object. Those two are necessary to be solved in order to conduct the network experiments.

The encoder problems are not as critical, but should be solved in the future.

**Problem 4:** Too high bitrate in the real-time mode caused by flaws in the codec.

**Solution:** One possible solution that would not influence the animation quality (or would not be noticeable) is to send every second or third frame instead of the whole stream. Missing frames can be regenerated by interpolation from received frames.

Reconstruction of three (or more) missing frames can cause a loss of single fast eye blink. Thus, it is not reliable to use linear interpolation with more than three lost frames.

Presented problems and solutions (and few other not mentioned here) were necessary to investigate in order to conduct experiments.

5.2.2 Real-time vs. pseudo-streaming mode

In the following, the discussion on two modes of streaming is conducted.

The main difference between the two modes is the allowed amount of total delay. This is the crucial part considering streaming of media over the network, especially in real-time applications.

**Real-time mode**

The delay is the critical aspect in real-time applications. According to [8], we define real-time mode as a scenario, in which the user receives continuous stream – near instantaneously (with a “minimum” delay). The duration of the transmitted and received stream is exactly the same. In practice the delay should not exceed 200 milliseconds, including the time needed for:

- generating frames (possibly using Visage|track application),
- encoding frames,
- sending the encoded stream over the network,
• decoding frames,
• rendering 3-D model on the other side of the communication channel.

The total delay can be calculated as follows:

**Total delay = source delay + network delay + encoder delay + render delay**

*Source delay*

In the experiment, the Visage|track application will be used. The average time needed for tracking a frame is 30 milliseconds. For more detailed description see [1].

*Network delay*

This component can take half of available total delay time. Assuming round trip time (RTT) between the server and the client to be between 50 and 190 milliseconds (Table 5-1), packet needs at least 25 to 95 milliseconds to reach the peer.

<table>
<thead>
<tr>
<th>Location</th>
<th>RTT [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poland 1</td>
<td>77</td>
</tr>
<tr>
<td>Poland 2</td>
<td>59</td>
</tr>
<tr>
<td>Poland 3</td>
<td>100</td>
</tr>
<tr>
<td>Sweden 1</td>
<td>11</td>
</tr>
<tr>
<td>Sweden 2</td>
<td>23</td>
</tr>
<tr>
<td>USA 1</td>
<td>189</td>
</tr>
<tr>
<td>USA 2</td>
<td>151</td>
</tr>
</tbody>
</table>

*Encoder delay*

Encoder delay is a sum of encoding and decoding delays. With the currently available implementation it varies from 10 to 20 milliseconds (if the decoder object is not deleted. Due to a bug in the implementation destroying the decoder object takes in some cases more than 100 milliseconds).

Decoding delay is the time needed for:

• Cutting a packet into separate frames.
• Writing frames into a file.
• Creating the decoder object.
• Loading the written stream from a file.
• Decoding the read stream.

*Render delay*

The render delay depends on performance of the client machine. It is not in the scope of our investigation.
To sum up, in real-time mode it is almost impossible to buffer any frames on the client side. The average total delay time for a single frame can be estimated to 150 milliseconds. Assuming 200 milliseconds as an upper bound on the total delay time, it is impossible to buffer any data ahead. If it was possible to receive some packets ahead, error concealment could be implemented.

**Pseudo-streaming mode**

The pseudo-streaming mode does not have strict delay constraints. According to the definition [8], in this mode the media starts playing a few seconds after the download starts. If the Internet connection speed is faster than the media rate, the user can watch the video while it is downloading. For the user it looks like streaming but, in effect, it is downloading.

The client can buffer some amount of packets ahead before starting rendering. This gives the application extra time that can be utilized for example for error concealment – handling packets out of order, recreating lost packets, and stabilizing the connection.

The maximum data received ahead can be easily calculated:

Frames per packet: 5
Frame rate: 25 frames per second
Receiving buffer size: 10 packets

\[
\left( \frac{10 \text{ packets} \cdot 5 \text{ frames}}{\text{packet}} \right) \div \frac{25 \text{ frames}}{\text{second}} = 2 \text{ seconds}
\]

In the above scenario, maximum of two seconds of data can be received. This makes error concealment feasible.

### 5.3 Functionality specification

The main goal is to implement two applications, which will investigate the possibilities to stream data in two described modes. The first application should operate in pseudo-streaming mode and should include streaming of animations from stored files.

The client will be able to negotiate connection, which will include:

- getting list of available animations,
- setting animation to stream,
- getting number of frames per second (FPS),
- getting/setting number of frames per packet (FPP),
- getting/setting number of frame to start,
- getting/setting number of frame to end.

The second application should operate in real-time mode. In the first stage of the experiment the server and the client objects will be implemented in the same application in order to make debugging simpler. The application will use interpolation of frames in order to lower the bitrate.
In the final stage the server will be integrated into the Visage\textregistered track application developed by Jörgen Ahlberg [1]. Visage\textregistered track will serve as a source of a real-time MPEG-4 facial animation data.

5.4 Prototyping

During experiments several approaches were verified. Starting from simple streaming of non-encoded data over TCP connection and ending on the complete solution for pseudo-streaming mode using the SM4P protocol.

The available MPEG-4 Face Animation codec implementation turned out to be the main obstacle to conduct all of planned experiments. The application operating in the real-time mode using Visage\textregistered track did not fulfill delay constraints, thus was not successful.

5.5 Simple MPEG-4 Protocol (SM4P)

5.5.1 Underlying protocols and the SM4P functionality

SM4P stands for Simple MPEG-4 Protocol. It is built on top of the User Datagram Protocol (UDP). The reason behind choosing UDP was its simplicity and the small packet delay it introduces.

SM4P is to be used in the real-time or the pseudo-streaming applications to stream facial animations. In that kind of applications – streaming media – a protocol based on Transmission Control Protocol (TCP) would work perfectly. Application would get packets in order with a guarantee of delivery [11]. Unfortunately in contrast to UDP, TCP introduces large packet delays (it needs time for connection establishment, retransmissions etc.).

UDP instead is a simple protocol – stateless, without delivery guarantee, packets can arrive out of order. This is the price to pay for a small delay and simplicity. Many real-time application protocols are built on top of UDP. In case of network congestion packets still will be lost or delayed [10].

SM4P is simple request/reply protocol that enables exchange messages between the server and the client. Requests and replies are integers in form:

 REQUEST value

If the client sends a “SET_FPP 10” message, that corresponds to the integer value 44010.

The server replies with the same message (REQUEST value) if the value was agreed – if not, it sends a REQUEST back to the client. Additionally client sets a timeout if the answer from the server was lost. For more details on available messages in the SM4P protocol see Appendix A.
Figure 5-3 Time line of connection establishment, negotiation, data exchange and closing connection.
The solution is simple and enough for that kind of applications. Negotiation stage allows clients to establish common connection parameters, for example how many frames should be included in a packet. The client is able to:

- get list of available animations,
- set animation to stream,
- get list of available face models to download,
- get/set start frame from which the server will stream data,
- get/set end frame to which the server will stream data,
- get/set how many frames is to be sent per packet,
- get/set frames per second parameter of animation.

After negotiation stage, data is sent in format presented in Paragraph 5.5.2 (Figure 5-4, 5-5).

### 5.5.2 SM4P data packet format

A SM4P packet consists of a sequence number, which is used by the client to identify received packets, and data. Encoded frames and their sizes are put together in the data field as illustrated in Figure 5-4.

To avoid fragmentation during the transmission over the network, the data field has to have an upper bound on length. Assuming the Ethernet at the link layer, where the Maximum Transfer Unit (MTU) is 1500 bytes, maximum data field size can be calculated. The IP, UDP and SM4P headers need 20, 8 and 5 bytes respectively. Thus the maximum data field size is 1467 bytes [9].

![Figure 5-4 SM4P packet format.](image-url)
5.6 Application object structure

The application object structure described in this chapter is the common basis for all experiment applications. The difference between applications lies in the implementation of CServer, CClient and CMessage objects.

Application objects, as well as the CMainFrame Object (server application) that are present in the actual implementation are omitted in Figure 5-6, since they are standard MFC SDI framework objects and they remain unmodified.

The server application consists of three main objects. The CDialog Object handles messages from the CServer Object in order to support feedback from connection threads (display communicates).
The CServer implements two kinds of working threads. The first one is the Main Server Thread, which waits for new connections on a certain port.

The actual connection is handled by the Connection Handle Thread. That thread is started by the Main Server Thread after receiving a connection request from the client.

The Connection Handle Thread holds the CMessage Object, which implements methods for encoding frames and also takes active part in the communication with the client. In addition, it keeps information about available animations in pseudo-streaming mode.

The client application is built up with four objects. The CMainView Object plays similar role as the CDialog Object in the server application – handles messages from the CClient Object to support feedback from threads (display communicates).

The COpenGL Object displays a face model with given FAP values applied. It is used by the Render Thread, implemented in the CClient Object.

The main communication with the server is handled by the CClient Object. This object implements two working threads: the Connection Handle Thread and the Render Thread. In addition it includes several methods for managing connection with the server.

Figure 5-6 Basic application object structure.
In the pseudo-streaming mode the Connection Handle Thread is responsible for connection with the server – receiving packets and storing them in the Receiving Buffer. It also signals the Render Thread, when animation frame is available for rendering. A common situation is illustrated in Figure 5-7.

The synchronization scheme consisting of two threads (Connection Handle Thread and Render Thread) is quite simple. The Connection Handle Thread starts the connection and then waits for packets. When a packet arrives it locks access to the Receiving Buffer – in order to ensure atomicity, only one thread can read/write to the buffer at a time – and puts the data from received packet at the buffer’s tail. Before looking for the next packet to arrive, the Connection Handle Thread triggers an event to alert the Render Thread, since there is data to be displayed.

Signaled Render Thread locks the Receiving Buffer and reads the data to render – always from position number 0. Next, it shifts packets in the Receiving Buffer to the left, because packet from the position number zero has been read. Shifting is implemented using very fast memcpy() method. It takes a small amount of time, since it is based on operations at the low level (processor registers).

Obviously, if the number of frames per packet is more than 3 or 4, the Connection Handle Thread will put packets faster than the Render Thread is able to read. Considering animation with 25 frames per second, one frame lasts for 40 milliseconds. Packets consisting of 3 or 4 frames will be displayed in the time of 120 or 160 milliseconds. Thus, the Connection Handle Thread will supply frames faster than the Render Thread can consume (display).

The situation is different for the real-time mode where the source does not have frames ahead and they are generated as fast as they should be displayed. The delay is critical and the client does not have time for buffering frames. Thus, the Connection Handle Thread besides receiving packets cuts and decodes frames. Then it triggers the Render Thread, which displays decoded frames.

Additionally in the real-time mode, the frame interpolation has been implemented. It allows to lower bitrate to half of its normal value. Interpolation works on one frame at a time using simple linear interpolation. The server sends every second frame and the skipped frames are interpolated.

Figure 5-7 and 5-8 illustrate timelines of synchronization of two threads.
Figure 5-7 Time line of common connection. Thread synchronization.
Figure 5-8 Synchronization of the Connection Handle Thread and the Render Thread.

1. Received packet number 12
2. Lock access to the Receiving Buffer
3. Write received packet to the buffer
4. Trigger render thread

1. Lock access to the Receiving Buffer
2. Read data packet
3. Shift packets in the buffer
4. Cut and decode data packet
5. Display frames on the screen
5.7 Implementation and evaluation

Most of the specified tasks were implemented during experiments. Starting from the pseudo-streaming mode, the client and the server application were created and tested. The application object structure, presented in Paragraph 5.6.1, was used and evaluated giving very good results.

Applications working in the real-time mode could not be fully implemented due to shortcomings in the existing MPEG-4 codec implementation, which introduces large delays. Only the first stage in the real-time part was achieved. In the implementation, the client and the server objects are incorporated into one application for easy debugging and testing. The Server simulates a real-time source by producing animation frames with certain delays. In the client object, interpolation of frames was implemented resulting in doubling framerate with bitrate reduced by half.

The Simple MPEG-4 Protocol (SM4P) was developed, implemented and fully tested in created applications.

During the implementation, several problems concerning network issues were encountered. Starting from basic implementation of the server and the client communication scheme using datagram sockets [12], ending on the advanced socket techniques, for example select() method for setting timeouts [13].

Screenshots from implemented applications are presented in Figures 5-9 to 5-11.

Figure 5-9 Screenshot from the experiment client application operating in the pseudo-streaming mode.
Figure 5-10 Screenshot from the experiment server application operating in the pseudo-streaming mode.

Figure 5-11 Screenshot from the experiment application operating in the real-time mode.
5.8 Conclusion

The main goal of experiments was to look into feasible solutions for streaming MPEG-4 facial animations. Examination of possible application architectures was conducted and the most suitable model was chosen and implemented. One of the obstacles was shortcomings in the available implementation of the MPEG-4 facial animation codec. It demanded few fixes concerning mainly memory allocation and the decoding time. It was not possible to repair all noticed errors, thus the achieved results are not fully satisfactory. The delay introduced by the encoder did not allow building a true real-time video system (stage two of the real-time application). In spite of that, created applications have shown very promising results and with the new encoder library it will be possible to build such a system. The experiment confirms Jörgen Ahlberg’s discussion on a real-time system [1].

*Pseudo-streaming mode*

Regarding a potential application architecture that can be used in the pseudo-streaming mode, implemented proposal shows very good and promising results. Two threads synchronization works very smoothly resulting in good time utilization. The developed protocol allows the client to negotiate connection’s parameters, for example the animation’s file name or the number of frames per packet. Proposed scheme for the connection (Figures 5-3, 5-7) fulfills the expectations. The BACK_OFF message turned out to be an efficient way for stopping the server from overloading the client’s receiving buffer.

*Real-time mode*

The integration of the client and the server objects into one application during the first stage of the real-time experiment allowed to test and verify that it is possible to build a true real-time system based on the proposed approach.

The server object simulates a real-time video source by producing frames at a given rate. The client object uses interpolation of missing frames to lower bitrate to the half of its value. Although, the encoder implementation has some drawbacks, interpolation technique worked very good, enhancing the animation quality.

To sum up, the experiment results are more than satisfactory giving hope for a wide use in the existing computer network architectures. The most important conclusion is that the true real-time system built up with the tracking application (e.g. Visage|track) is feasible.

5.9 Future extensions

Future extensions in applications working in the pseudo-streaming mode include the investigation of possible error concealment schemes. The main question to answer is if recovering from packet loss and reordering of packets is feasible.

With the new encoder library the second stage of the experiment can be conducted. Integration of a server object into the Visage|track will introduce synchronization problems, which can be solved using multithreading techniques. With the current implementation of the real-time application the integration should be easy and quick. Additionally, the Simple MPEG-4 Protocol (SM4P) header should be extended to include timestamp information. The encoded stream, according to the
MPEG-4 Face Animation standard, includes the timestamp information, but if the stream consists of a single frame there is need to send that information separately.
6 Summary and Conclusion

In this thesis several applications in the area of facial animations were developed:

- the new Visage|toolkit application (Chapter 3),
- the Visage|edit application (Chapter 4),
- experimental applications for testing network related issues (Chapter 5).

Regarding the first two, functionality specifications were prepared, prototypes evaluated and the software was produced. During the development stage several approaches were proposed and verified. Examples are:

- double slider approach for choosing the range of frames (Figure 4-2) which turned out to be not efficient and the final version was exchanged with different solution (Figure 4-6)
- curve shaping using Bézier functions (Figure 4-4)
- manipulating already existing filter files (Figure 4-3)

All of the proposed solutions can be used if necessary in the future extensions of the Visage|toolkit and the Visage|edit applications. Additionally, possible future work was specified.

In the network part of the thesis feasible solutions for streaming MPEG-4 facial animations were discussed and verified (Chapter 5).

The first part deals with applications working in the pseudo-streaming mode. The application architecture was proposed and verified giving good results. The animation streaming scheme using two threads performs very well resulting in a clear and complete application architecture. Applications are able to exchange messages to negotiate connection parameters, for example animation name or number of frames per packet.

Although the second stage (real-time mode) of the network experiment could not be fully conducted because of the shortcomings in the MPEG-4 Face Animation codec, achieved results are satisfactory. The experiment application simulating the real-time mode showed promising results. The interpolation function works very well, but the result shows that interpolating more than 2 frames is not possible without lowering the animation quality. The new implementation of the codec will allow incorporating the final solution in the Visage|track application resulting in a true real-time system.

Both applications use the Simple MPEG-4 Protocol (SM4P) developed and extended during experiments. Although SM4P is a simple protocol, it turned out to be a very robust and efficient solution for streaming of the MPEG-4 facial animation data. A simple scheme to avoid buffer overflow (one single back off message with the duration specified) on the client side was tried and proved to be sufficient. Future extensions include expansion of the existing SM4P header to contain timestamp information in case of sending a single frame at a time.
## Appendix A: Simple MPEG-4 Protocol (SM4P) implementation

### A.1 Available queries to server and responses

<table>
<thead>
<tr>
<th>Group</th>
<th>Name</th>
<th>REQUEST</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting connection</td>
<td>GET_NEW_PORT</td>
<td>111</td>
<td>Client-&gt;Server starts a new connection</td>
</tr>
<tr>
<td></td>
<td>NEW_PORT</td>
<td>112</td>
<td>Server-&gt;Client connection started on port</td>
</tr>
<tr>
<td>Negotiation</td>
<td>SET_FPS</td>
<td>430</td>
<td>Client-&gt;Server sets no of frames per second</td>
</tr>
<tr>
<td></td>
<td>GET_FPS</td>
<td>435</td>
<td>Client-&gt;Server gets no of frames per second</td>
</tr>
<tr>
<td></td>
<td>SET_FPP</td>
<td>440</td>
<td>Client-&gt;Server sets no of frames per packet</td>
</tr>
<tr>
<td></td>
<td>GET_FPP</td>
<td>445</td>
<td>Client-&gt;Server gets no of frames per packet</td>
</tr>
<tr>
<td></td>
<td>SET_START_FRM</td>
<td>450</td>
<td>Client-&gt;Server sets no of start frame from which Server will start stream</td>
</tr>
<tr>
<td></td>
<td>GET_START_FRM</td>
<td>455</td>
<td>Client-&gt;Server gets no of start frame from which Server will start stream</td>
</tr>
<tr>
<td></td>
<td>SET_END_FRM</td>
<td>460</td>
<td>Client-&gt;Server sets no of end frame</td>
</tr>
<tr>
<td></td>
<td>GET_END_FRM</td>
<td>465</td>
<td>Client-&gt;Server gets no of end frame</td>
</tr>
<tr>
<td></td>
<td>SET_ANIM</td>
<td>477</td>
<td>Client-&gt;Server sets animation to stream</td>
</tr>
<tr>
<td></td>
<td>GET_ANIM_LIST</td>
<td>488</td>
<td>Client-&gt;Server gets list of available animation on the Server side</td>
</tr>
<tr>
<td></td>
<td>GET_MODEL_LIST</td>
<td>499</td>
<td>Client-&gt;Server gets list of available face models on the Server side</td>
</tr>
<tr>
<td>Streaming</td>
<td>START_STREAM</td>
<td>411</td>
<td>Client-&gt;Server triggers streaming</td>
</tr>
<tr>
<td></td>
<td>BACK_OFF</td>
<td>511</td>
<td>Client-&gt;Server no of milliseconds to sleep by the Server in order to not overload client</td>
</tr>
<tr>
<td>Closing connection</td>
<td>BYE</td>
<td>999</td>
<td>Client-&gt;Server closes connection</td>
</tr>
</tbody>
</table>
A.2 Object structure

Objects: Cserver, Cclient, Cmessage.

CMessage object holds messages and methods to create and handle them.

<table>
<thead>
<tr>
<th>CServer</th>
<th>CMessage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MainThread</strong></td>
<td><strong>Connection management:</strong></td>
</tr>
<tr>
<td>ConnectionHandleThread</td>
<td>• decodeAnimation</td>
</tr>
<tr>
<td>StartMainServer</td>
<td>• sendFbaStream</td>
</tr>
<tr>
<td>StopMainServer</td>
<td>• msgAddHdr</td>
</tr>
<tr>
<td></td>
<td>• msgCreate</td>
</tr>
<tr>
<td></td>
<td>• writeBits</td>
</tr>
</tbody>
</table>

**Data management:**

- SetAnimation
- AnimList
  - mAnimList[ANIM_COUNT]
- FaceList m_faceList[FACE_COUNT]
- long m_lTimeStamp[MAXNFrames];
- GetFPS / SetFPS / GetFPP /
  - SetFPP / GetStartFrm SetStartFrm /
  - GetEndFrm / SetEndFrm /
  - GetMaxFrm / GetAnimName /
  - GetAvailAnimCount / GetFaceName /
  - GetAvailFaceCount

<table>
<thead>
<tr>
<th>CClient</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sendSTART_STREAM</td>
<td><strong>Connection management:</strong></td>
</tr>
<tr>
<td>sendRequest (int iRequest, int iValue, int iTimeout)</td>
<td>decodeStream</td>
</tr>
<tr>
<td>sendBYE</td>
<td>msgStripHdr</td>
</tr>
<tr>
<td>SetServerAddress</td>
<td>msgCut</td>
</tr>
<tr>
<td>SetServerPortNo</td>
<td>writeBits</td>
</tr>
<tr>
<td>startConnection</td>
<td></td>
</tr>
</tbody>
</table>

Figure A-1 Objects and methods in Simple MPEG-4 Protocol (SM4P) implementation.
Appendix B: Facial Animation Parameters (FAPs) specification

B.1 Facial Animation Parameters (FAPs)

MPEG-4 defines 68 Facial Animation Parameters (FAP): 2 high-level FAPs, visemes and expressions, and 66 low-level FAPs, as shown in table below. Table also shows grouping of FAPs and their respective units [5].

Table B.1-1 Facial Animation Parameters.

<table>
<thead>
<tr>
<th>#</th>
<th>FAP name</th>
<th>FAP description</th>
<th>Units</th>
<th>Uni-or Bidir</th>
<th>Pos Motion</th>
<th>Grp</th>
<th>FDP Subgrp Num</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>viseme</td>
<td>Set of values determining the mixture of two visemes for this frame (e.g. pbm,</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>1</td>
<td>na</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fv, th)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>expression</td>
<td>A set of values determining the mixture of two facial expression</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>1</td>
<td>na</td>
</tr>
<tr>
<td>3</td>
<td>open_Jaw</td>
<td>Vertical jaw displacement (does not affect mouth opening)</td>
<td>MNS</td>
<td>U</td>
<td>down</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>lower_t_midlip</td>
<td>Vertical top middle inner lip displacement</td>
<td>MNS</td>
<td>B</td>
<td>down</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>raise_b_midlip</td>
<td>Vertical bottom middle inner lip displacement</td>
<td>MNS</td>
<td>B</td>
<td>up</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>stretch_L_cornerlip</td>
<td>Horizontal displacement of left inner lip corner</td>
<td>MW</td>
<td>B</td>
<td>left</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>stretch_R_cornerlip</td>
<td>Horizontal displacement of right inner lip corner</td>
<td>MW</td>
<td>B</td>
<td>right</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>lower_t_lip_lm</td>
<td>Vertical displacement of midpoint between left corner and middle of top inner lip</td>
<td>MNS</td>
<td>B</td>
<td>down</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>lower_t_lip_rm</td>
<td>Vertical displacement of midpoint between right corner and middle of top inner lip</td>
<td>MNS</td>
<td>B</td>
<td>down</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>raise_b_lip_lm</td>
<td>Vertical displacement of midpoint between left corner and middle of bottom inner lip</td>
<td>MNS</td>
<td>B</td>
<td>up</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>raise_b_lip_rm</td>
<td>Vertical displacement of midpoint between right corner and middle of bottom inner lip</td>
<td>MNS</td>
<td>B</td>
<td>up</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>12</td>
<td>raise_L_cornerlip</td>
<td>Vertical displacement of left inner lip corner</td>
<td>MNS</td>
<td>B</td>
<td>up</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>raise_R_cornerlip</td>
<td>Vertical displacement of right inner lip corner</td>
<td>MNS</td>
<td>B</td>
<td>up</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>thrust_jaw</td>
<td>Depth displacement of jaw</td>
<td>MNS</td>
<td>U</td>
<td>forward</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>shift_jaw</td>
<td>Side to side displacement of jaw</td>
<td>MW</td>
<td>B</td>
<td>right</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>push_b_lip</td>
<td>Depth displacement of bottom middle lip</td>
<td>MNS</td>
<td>B</td>
<td>forward</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>push_t_lip</td>
<td>Depth displacement of top middle lip</td>
<td>MNS</td>
<td>B</td>
<td>forward</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>#</td>
<td>FAP name</td>
<td>FAP description</td>
<td>Units</td>
<td>Uni- or Bidir</td>
<td>Pos Motion</td>
<td>Grp</td>
<td>FDP Subgrp Num</td>
</tr>
<tr>
<td>----</td>
<td>------------------</td>
<td>----------------------------------------------------------------</td>
<td>--------</td>
<td>---------------</td>
<td>------------</td>
<td>-----</td>
<td>----------------</td>
</tr>
<tr>
<td>18</td>
<td>depress_chin</td>
<td>Upward and compressing movement of the chin (like in sadness)</td>
<td>MNS</td>
<td>B</td>
<td>up</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>19</td>
<td>close_t_l_eyelid</td>
<td>Vertical displacement of top left eyelid</td>
<td>IRISD</td>
<td>B</td>
<td>down</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>close_t_r_eyelid</td>
<td>Vertical displacement of top right eyelid</td>
<td>IRISD</td>
<td>B</td>
<td>down</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>close_b_l_eyelid</td>
<td>Vertical displacement of bottom left eyelid</td>
<td>IRISD</td>
<td>B</td>
<td>up</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>close_b_r_eyelid</td>
<td>Vertical displacement of bottom right eyelid</td>
<td>IRISD</td>
<td>B</td>
<td>up</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>23</td>
<td>yaw_l_eyeball</td>
<td>Horizontal orientation of left eyeball</td>
<td>AU</td>
<td>B</td>
<td>left</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>24</td>
<td>yaw_r_eyeball</td>
<td>Horizontal orientation of right eyeball</td>
<td>AU</td>
<td>B</td>
<td>left</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>25</td>
<td>pitch_l_eyeball</td>
<td>Vertical orientation of left eyeball</td>
<td>AU</td>
<td>B</td>
<td>down</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>26</td>
<td>pitch_r_eyeball</td>
<td>Vertical orientation of right eyeball</td>
<td>AU</td>
<td>B</td>
<td>down</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>27</td>
<td>thrust_l_eyeball</td>
<td>Depth displacement of left eyeball</td>
<td>ES</td>
<td>B</td>
<td>forward</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>28</td>
<td>thrust_r_eyeball</td>
<td>Depth displacement of right eyeball</td>
<td>ES</td>
<td>B</td>
<td>forward</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>29</td>
<td>dilate_l_pupil</td>
<td>Dilation of left pupil</td>
<td>IRISD</td>
<td>B</td>
<td>growing</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>30</td>
<td>dilate_r_pupil</td>
<td>Dilation of right pupil</td>
<td>IRISD</td>
<td>B</td>
<td>growing</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>31</td>
<td>raise_l_i_eyebrow</td>
<td>Vertical displacement of left inner eyebrow</td>
<td>ENS</td>
<td>B</td>
<td>up</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>raise_r_i_eyebrow</td>
<td>Vertical displacement of right inner eyebrow</td>
<td>ENS</td>
<td>B</td>
<td>up</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>33</td>
<td>raise_l_m_eyebrow</td>
<td>Vertical displacement of left middle eyebrow</td>
<td>ENS</td>
<td>B</td>
<td>up</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>34</td>
<td>raise_r_m_eyebrow</td>
<td>Vertical displacement of right middle eyebrow</td>
<td>ENS</td>
<td>B</td>
<td>up</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>35</td>
<td>raise_l_o_eyebrow</td>
<td>Vertical displacement of left outer eyebrow</td>
<td>ENS</td>
<td>B</td>
<td>up</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>36</td>
<td>raise_r_o_eyebrow</td>
<td>Vertical displacement of right outer eyebrow</td>
<td>ENS</td>
<td>B</td>
<td>up</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>37</td>
<td>squeeze_l_eyebrow</td>
<td>Horizontal displacement of left eyebrow</td>
<td>ES</td>
<td>B</td>
<td>right</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>38</td>
<td>squeeze_r_eyebrow</td>
<td>Horizontal displacement of right eyebrow</td>
<td>ES</td>
<td>B</td>
<td>left</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>39</td>
<td>puff_l_cheek</td>
<td>Horizontal displacement of left cheek</td>
<td>ES</td>
<td>B</td>
<td>left</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>puff_r_cheek</td>
<td>Horizontal displacement of right cheek</td>
<td>ES</td>
<td>B</td>
<td>right</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>41</td>
<td>lift_l_cheek</td>
<td>Vertical displacement of left cheek</td>
<td>ENS</td>
<td>U</td>
<td>up</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>42</td>
<td>lift_r_cheek</td>
<td>Vertical displacement of right cheek</td>
<td>ENS</td>
<td>U</td>
<td>up</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>43</td>
<td>shift_tongue_tip</td>
<td>Horizontal displacement of tongue tip</td>
<td>MW</td>
<td>B</td>
<td>right</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>44</td>
<td>raise_tongue_tip</td>
<td>Vertical displacement of tongue tip</td>
<td>MNS</td>
<td>B</td>
<td>up</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>45</td>
<td>thrust_tongue_tip</td>
<td>Depth displacement of tongue tip</td>
<td>MW</td>
<td>B</td>
<td>forward</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>#</td>
<td>FAP name</td>
<td>FAP description</td>
<td>Units</td>
<td>Uni- or Bidir</td>
<td>Pos Motion</td>
<td>Grp</td>
<td>FDP Subgrp Num</td>
</tr>
<tr>
<td>----</td>
<td>-------------------</td>
<td>------------------------------------------------------</td>
<td>-------</td>
<td>--------------</td>
<td>------------</td>
<td>-----</td>
<td>----------------</td>
</tr>
<tr>
<td>46</td>
<td>raise_tongue</td>
<td>Vertical displacement of tongue</td>
<td>MNS</td>
<td>B</td>
<td>up</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>47</td>
<td>tongue_roll</td>
<td>Rolling of the tongue into U shape</td>
<td>AU</td>
<td>U</td>
<td>concave upward</td>
<td>6</td>
<td>3, 4</td>
</tr>
<tr>
<td>48</td>
<td>head_pitch</td>
<td>Head pitch angle from top of spine</td>
<td>AU</td>
<td>B</td>
<td>down</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>49</td>
<td>head_yaw</td>
<td>Head yaw angle from top of spine</td>
<td>AU</td>
<td>B</td>
<td>left</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>head_roll</td>
<td>Head roll angle from top of spine</td>
<td>AU</td>
<td>B</td>
<td>right</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>51</td>
<td>lower_t_midlip_o</td>
<td>Vertical top middle outer lip displacement</td>
<td>MNS</td>
<td>B</td>
<td>down</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>52</td>
<td>raise_b_midlip_o</td>
<td>Vertical bottom middle outer lip displacement</td>
<td>MNS</td>
<td>B</td>
<td>up</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>53</td>
<td>stretch_l_cornerlip_o</td>
<td>Horizontal displacement of left outer lip corner</td>
<td>MW</td>
<td>B</td>
<td>left</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>54</td>
<td>stretch_r_cornerlip_o</td>
<td>Horizontal displacement of right outer lip corner</td>
<td>MW</td>
<td>B</td>
<td>right</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>55</td>
<td>lower_t_lip_lm_o</td>
<td>Vertical displacement of midpoint between left corner and middle of top outer lip</td>
<td>MNS</td>
<td>B</td>
<td>down</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>56</td>
<td>lower_t_lip_rm_o</td>
<td>Vertical displacement of midpoint between right corner and middle of top outer lip</td>
<td>MNS</td>
<td>B</td>
<td>down</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>57</td>
<td>raise_b_lip_lm_o</td>
<td>Vertical displacement of midpoint between left corner and middle of bottom outer lip</td>
<td>MNS</td>
<td>B</td>
<td>up</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>58</td>
<td>raise_b_lip_rm_o</td>
<td>Vertical displacement of midpoint between right corner and middle of bottom outer lip</td>
<td>MNS</td>
<td>B</td>
<td>up</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>59</td>
<td>raise_l_cornerlip_o</td>
<td>Vertical displacement of left outer lip corner</td>
<td>MNS</td>
<td>B</td>
<td>up</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>60</td>
<td>raise_r_cornerlip_o</td>
<td>Vertical displacement of right outer lip corner</td>
<td>MNS</td>
<td>B</td>
<td>up</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>61</td>
<td>stretch_l_nose</td>
<td>Horizontal displacement of left side of nose</td>
<td>ENS</td>
<td>B</td>
<td>left</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>62</td>
<td>stretch_r_nose</td>
<td>Horizontal displacement of right side of nose</td>
<td>ENS</td>
<td>B</td>
<td>right</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>63</td>
<td>raise_nose</td>
<td>Vertical displacement of nose tip</td>
<td>ENS</td>
<td>B</td>
<td>up</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>64</td>
<td>bend_nose</td>
<td>Horizontal displacement of nose tip</td>
<td>ENS</td>
<td>B</td>
<td>right</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>65</td>
<td>raise_l_ear</td>
<td>Vertical displacement of left ear</td>
<td>ENS</td>
<td>B</td>
<td>up</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>66</td>
<td>raise_r_ear</td>
<td>Vertical displacement of right ear</td>
<td>ENS</td>
<td>B</td>
<td>up</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>67</td>
<td>pull_l_ear</td>
<td>Horizontal displacement of left ear</td>
<td>ENS</td>
<td>B</td>
<td>left</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>68</td>
<td>pull_r_ear</td>
<td>Horizontal displacement of right ear</td>
<td>ENS</td>
<td>B</td>
<td>right</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>
Since the FAPs are required to animate faces of different sizes and proportions, the FAP values are defined in face animation parameter units (FAPU). FAPU are defined as fractions of distances between key facial features. These features like eye separation, eye-nose separation, mouth-nose separation, and mouth width, are defined for the face in its neutral state. They allow interpretation of the FAPs on any facial model in a consistent way, producing reasonable results in terms of expression and speech pronunciation. The standard defines 6 movement measurement units. These units are always relative to a particular face model, and must be known for any MPEG-4 compliant face [5].

Table B.1-2 Facial Animation Parameters Units.

<table>
<thead>
<tr>
<th>Description</th>
<th>FAPU value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRIS Diameter (by definition it is equal to the distance between upper and lower eyelid) in neutral face</td>
<td>IRISD = IRISD0 / 1024</td>
</tr>
<tr>
<td>Eye Separation</td>
<td>ES = ES0 / 1024</td>
</tr>
<tr>
<td>Eye - Nose Separation</td>
<td>ENS = ENS0 / 1024</td>
</tr>
<tr>
<td>Mouth - Nose Separation</td>
<td>MNS = MNS0 / 1024</td>
</tr>
<tr>
<td>Mouth - Width Separation</td>
<td>MW = MW0 / 1024</td>
</tr>
<tr>
<td>Angular Unit</td>
<td>AU = 10^{-5} rad</td>
</tr>
</tbody>
</table>
List of figures

Figure 2-1 A face model in its neutral state. ................................................................. 6
Figure 2-2 The facial feature points (FPs) as defined by the MPEG-4 standard. .......... 6
Figure 2-3 Example of head_roll FAP value plot. ........................................................ 7
Figure 2-4 Dependencies between main objects in SDI Windows applications .......... 8
Figure 3-1 Original Visage|toolkit application structure ............................................ 12
Figure 3-2 New object structure ................................................................................ 13
Figure 3-3 Screenshot from new Visage|toolkit application ........................................ 14
Figure 4-1 Low fidelity prototype showing the main concept of the Visage|edit application. ............................................................................................................ 17
Figure 4-2 High fidelity prototype for testing plotting area and range selection .......... 17
Figure 4-3 High fidelity prototype for testing track merging and applying filers .......... 18
Figure 4-4 High fidelity prototype for testing curve shaping using Bézier functions ...... 18
Figure 4-5 Use of the virtual track .............................................................................. 21
Figure 4-6 Main application window of the Visage|edit .............................................. 21
Figure 4-7 General object structure ............................................................................ 23
Figure 4-8 Document/view object structure .............................................................. 24
Figure 5-1 System architecture .................................................................................. 26
Figure 5-2 Protocol stack ........................................................................................... 27
Figure 5-3 Time line of connection establishment, negotiation, data exchange and closing connection. ................................................................. 32
Figure 5-4 SM4P packet format ................................................................................ 33
Figure 5-5 Encapsulation of data as it goes down the protocol stack ....................... 34
Figure 5-6 Basic application object structure ............................................................ 35
Figure 5-7 Time line of common connection. Thread synchronization .................... 37
Figure 5-8 Synchronization of the Connection Handle Thread and the Render Thread .... 38
Figure 5-9 Screenshot from the experiment client application operating in the pseudo-streaming mode ................................................................. 39
Figure 5-10 Screenshot from the experiment server application operating in the pseudo-streaming mode ................................................................. 40
Figure 5-11 Screenshot from the experiment application operating in the real-time mode..... 40
Figure A-1 Objects and methods in Simple MPEG-4 Protocol (SM4P) implementation. ...... 46
List of tables

Table 5-1 Examples of RTT for several locations. ......................................................... 29
Table B.1-1 Facial Animation Parameters. ................................................................. 47
Table B.1-2 Facial Animation Parameters Units. ....................................................... 50
References


[18] Roberto Pockaj, Fabio Lavagetto, Maurizio Costa, Carlo Bonamico, Interpolating MPEG-4 FAPs For Very Low Bitrate Facial Animation, DIST – University of Genova, Italy