

# ANIMATED TELECONFERENCING: VIDEO DRIVEN FACIAL ANIMATION

Ian Andrew Buck

June 1999

This thesis represents my own work in accordance with University regulations

# Abstract

This work presents a method of using hand-drawn art for an animated teleconferencing system. This low bandwidth solution to teleconferencing uses a computer generated animated character to represent the expression of a human user as seen by a desktop camera. Using hand-drawn art provides many advantages over traditional teleconferencing, including privacy, bandwidth, and entertainment value. The framework of the system is divided into three components: the video tracker which is able to locate, in real-time, features on a human face from live video; an expression mapper which is able to intelligently select which artworks should be used to represent the current expression; and a rendering engine which is able to morph in real-time the artworks to produce a final image.

# Contents

<b>Abstract</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Communication on the Consumer PC . . . . .	1
1.2 Teleconferencing . . . . .	2
1.3 Animation . . . . .	3
<b>2 Approach</b>	<b>5</b>
2.1 Constraints . . . . .	5
2.2 Goals . . . . .	5
2.3 System Overview . . . . .	6
<b>3 Related Work</b>	<b>8</b>
3.1 Videophone . . . . .	8
3.2 Non-Photorealistic Performance-Driven Facial Animation . . . . .	9
3.3 Face Driven Animation . . . . .	10
<b>4 System Components</b>	<b>13</b>
4.1 Overview . . . . .	13
4.2 Feature Tracking . . . . .	13
4.2.1 Introduction and Goals . . . . .	13
4.2.2 Assumptions . . . . .	14
4.2.3 Design . . . . .	15
4.2.4 Mouth Tracking . . . . .	16

4.2.5	Eye Tracking . . . . .	17
4.3	Expression Mapping . . . . .	20
4.3.1	Introduction and Goals . . . . .	20
4.3.2	Approach . . . . .	21
4.4	Rendering Engine . . . . .	25
4.4.1	Introduction and Goals . . . . .	25
4.4.2	Composition . . . . .	25
4.4.3	Morphing the Images . . . . .	25
4.4.4	Morphing Algorithm . . . . .	26
4.4.5	Optimized Algorithm . . . . .	29
4.4.6	Blending . . . . .	30
4.4.7	Artwork Normalization . . . . .	32
<b>5</b>	<b>Implementation</b>	<b>34</b>
5.1	Tracker Implementation . . . . .	34
5.1.1	Video Source . . . . .	35
5.2	Communication . . . . .	36
5.3	Expression Mapper Implementation . . . . .	36
5.3.1	Precomputation . . . . .	37
5.3.2	Run-time System . . . . .	37
5.4	Rendering Engine Implementation . . . . .	38
5.4.1	Morph Files . . . . .	38
5.5	Using the System . . . . .	39
5.5.1	Drawing Artwork . . . . .	39
5.5.2	Drawing Feature Vectors . . . . .	40
5.5.3	Training . . . . .	40
<b>6</b>	<b>Results</b>	<b>42</b>
6.1	Tracker Analysis . . . . .	42
6.2	Expression Mapping . . . . .	42
6.3	Artwork . . . . .	43
6.4	Rendering . . . . .	44

6.5	Performance . . . . .	44
<b>7</b>	<b>Conclusion</b>	<b>49</b>
7.1	Future Work . . . . .	49
7.1.1	Head Rotation . . . . .	49
7.1.2	Gaze . . . . .	50
7.1.3	Generating Artwork . . . . .	50
7.2	Conclusion . . . . .	52
<b>A</b>	<b>SIGGRAPH Submission</b>	<b>54</b>
<b>B</b>	<b>Non-Photorealistic Performance-driven Facial Animation</b>	<b>55</b>
<b>C</b>	<b>Face Driven Animation</b>	<b>56</b>
<b>D</b>	<b>Data Files</b>	<b>57</b>

# Chapter 1

## Introduction

### 1.1 Communication on the Consumer PC

With processor speeds increasing, desktop peripherals becoming more common, and the internet providing a new medium for communication possibilities, technologies that would have been unthinkable a few years ago are quickly becoming commonplace. As technology advances and becomes accepted by consumers, it is often interesting to explore new and interesting applications of technology, as well as to improve and expand existing applications.

Not only has computer technology blossomed in recent years, the internet has also expanded the capabilities of the computer as a tool for communication. With 20 million people on-line, the internet offers a channel of communication that is unprecedented. Yet communication technologies are still restricted by bandwidth limitations. Passive and low bandwidth communications like email and web browsing are quite effective, however high bandwidth communication like video streaming and internet phones continue to suffer from the limited amount of data that can be transferred between server and client. As we develop new ideas for allowing the 20 million on the internet to communicate together, we must be aware that high bandwidth applications will continue to be susceptible to network constraints.

On par with the growth of the internet, computers too are developing into powerful interactive machines. As CPU speeds get faster, memory busses get wider,

and multimedia hardware becomes more commonplace, the computer is expanding its perception of the world around it. Desktop cameras, though previously rare are becoming a standard accessory on consumer PCs. Once only available on high-end research based machines, graphics cards with on-board geometry and rasterization hardware are now so inexpensive and commonplace that a new computer would not be sold without one.

The larger trend emerging is that the common desktop PC is becoming a device that can interact with the world around it. With the advent of the internet, the PC has a huge potential to become a powerful communication tool. By giving eyes to the common computer, a processor capable of handling live video, a network that has over 20 million subscribers, and an output device powerful enough to mimic reality, the computer can open new doors, facilitating global communication.

This work explores the possibility of using common PC capabilities to develop a new communication technology that not only takes advantage of the desktop PC, but also works within their current limitations.

## 1.2 Teleconferencing

Teleconferencing is a technology that has existed for many years as a viable form of face-to-face communication. The basic concept is quite simple. Typically it deals with the transmission of visual and auditory information to permit long-distance, face-to-face communication between the two participants. Users not only understand the speaker's words through hearing, but the facial expressions provide pivotal visual cues to accompany the dialog. According to one source, 45% of face-to-face communication comes from the spoken words, while 40% is conveyed by facial expressions, with the last 15% coming from remaining body language [1]. The importance of facial expressions makes transmitting the video along with the audio an important component of communication.

The bandwidth costs of transmitting video however, are quite expensive. Most video compression algorithms can lower the size of each video frame update. However, even some of the most popular compression schemes reduce the necessary bandwidth



Figure 1: An example of internet video compression. This example is a 24Kbit/second RealVideo obtained from the CNN web sight. Note the prominent compression artifacts which are considered acceptable for viewing.

only so far without introducing noticeable artifacts. In fact, most video streams which run over the internet today consider significant compression artifacts acceptable as shown in Figure 1.

### 1.3 Animation

This work explores the possibility of preserving the facial expressions in teleconferencing which are an important part in face-to-face communication while circumventing the bandwidth restrictions of internet. Facial expression does not need to be photo-realistic to convey the proper expression. For example, an animated character can expresses similar information as a human being. Animated characters can often be more expressive than that of a human face, since their expressive ability is only constrained by the artist's creativity. In the animation industry, facial expression is a



critical aspect of an animated character. Often animated characters can be more lively and engaging than a human actor.

This project explores the possibility of using a non-photorealistic representation of teleconferencing subjects to replace live video with a computer generated animated character. By utilizing modern desktop PC computer graphics capabilities, the system would only need to transmit the information necessary to produce the proper expression at the receiver's computer. By reducing the communication from high bandwidth video data down to low bandwidth expression data, the hopes are that a complete system would have plenty of internet bandwidth still available for audio information.

Furthermore, replacing actual video data with generated non-photorealistic data has other benefits besides improving bandwidth performance and expressiveness. Animated characters can also offer an anonymous presence for a user. Since their actual visual information isn't transmitted, a user could select an animated presence that suited them, protecting or altering their identity.<sup>1</sup>

---

<sup>1</sup>This is not to say that the human face is not interesting. However, an animated character is typically not subject to unfortunate facial *faux pas* as a human face, such as a "bad hair day".

# Chapter 2

## Approach

In order to obtain a working teleconferencing application that can replace live video, the project must achieve fairly aggressive goals which make this alternative viable. Furthermore, the system must work under the constraints of the desktop PC and still be able to perform all the necessary goals of each of the system's components.

### 2.1 Constraints

First we require that the system work with basic desktop PC capabilities, or at least technology that is emerging on the consumer market. With any new communication technology, it is critical that it be available to a wide base of users, otherwise its unlikely to be accepted. Furthermore, the system must operate with a network bandwidth that supports the internet.

### 2.2 Goals

The project must provide a low-bandwidth solution to streaming video communication.

1. Expression preservation is crucial. Our animated presence should be able to perform just the same as (or better than) the user, in order to preserve the expression information.

2. Real-time performance is required. As with any communication system, latency is cannot be tolerated. The system must perform at frame rate that is acceptable for both users. For most video applications, 30 frames per second (fps) offer a sufficient refresh rate such that the human eye cannot notice the individual frames in live video. However, most of today's animators only produce 10 fps animations since 30 fps results in jittery and anxious appearing characters. Therefore, we would like to operate at least as fast as today's animated features, 10fps or better.
3. Ease of use. Like any communication method, the user should not be forced to adapt to work with our system. This means that the feature information must be obtained without inconveniencing the user. No additions or visual aids should be required by the user which would deter them from using the system.
4. Everyone should be able to use the system. The parameters of the feature tracking should not be too restrictive for the people who will use the program. Ideally, this software should be available to all who want to use it.

Using these goals and operating under the given constraints, we can derive a system which provides for a viable implementation of animated teleconferencing.

## 2.3 System Overview

In order to create a working animated teleconferencing solution, the system must perform two main tasks. First, on the transmitting side, the system must be able to extract expression information from live video. Is the person smiling, frowning, surprised, or confused? On the receiving side, we require the ability to render an animated character with the proper expression received from the transmitter.

To accomplish these tasks, the system is broken up into three primary components. The first component is a computer based video tracking system or **Facial Tracker** to track key features of the sender's face. These data values can be transmitted across the internet to be processed by the receiver to generate the animated character.

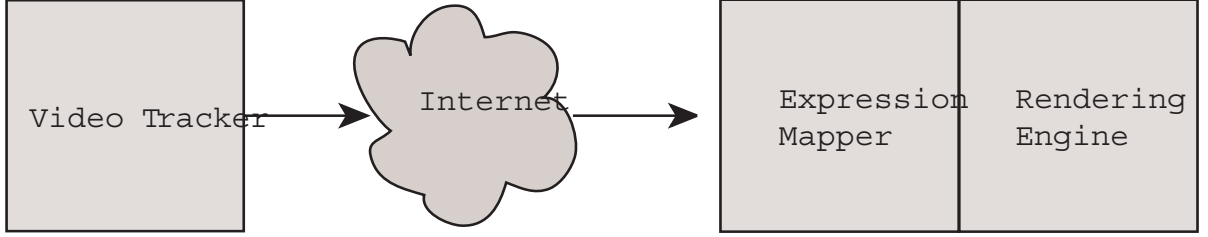


Figure 2: System Component Overview

On the receiver end, we have a set of drawings which consist of a base set of expressions that the animated character can perform. These expressions are manifested on the user's screen by a rendering system which consists of two main parts: (1) an **Expression Mapper** which is capable of translating the video tracking data into expression information that specifies which animated drawings best approximate the expression seen in the camera and (2) a **Rendering Engine** which computes the composition of the animated drawings selected to create the final output. These three components provide a framework on which to develop our real-time animated teleconferencing system, as shown in Figure 2. The approach each of the components is described further in Section 4: System Components.

# Chapter 3

## Related Work

### 3.1 Videophone

Many science fiction writers affiliate videophones as part of the invention and an inevitable progression of communication technology. However, the challenge of creating a practical form of transmitting facial images along with audio data has eluded the mainstream market for quite some time. Primarily, this is due to the high bandwidth necessary for smooth live video. While research has been done which explores low bandwidth and compression schemes, the results have yet to provide a viable solution [2, 3].

The research community has been flooded over the last few years with facial animation discoveries which focus on generating accurate images of human faces. Overall, works range from video compression techniques, to 3-D driven models, to morphing and blending applications. Many of these systems are real-time, however they are geared more towards production style environments and not common users. For example, many require the use of facial markers and controlled environments in order to be effective. Our system differs in its objectives from previous research since its goal is non-photorealistic renderings. For a complete analysis of related research, the reader is referred to Appendix A.

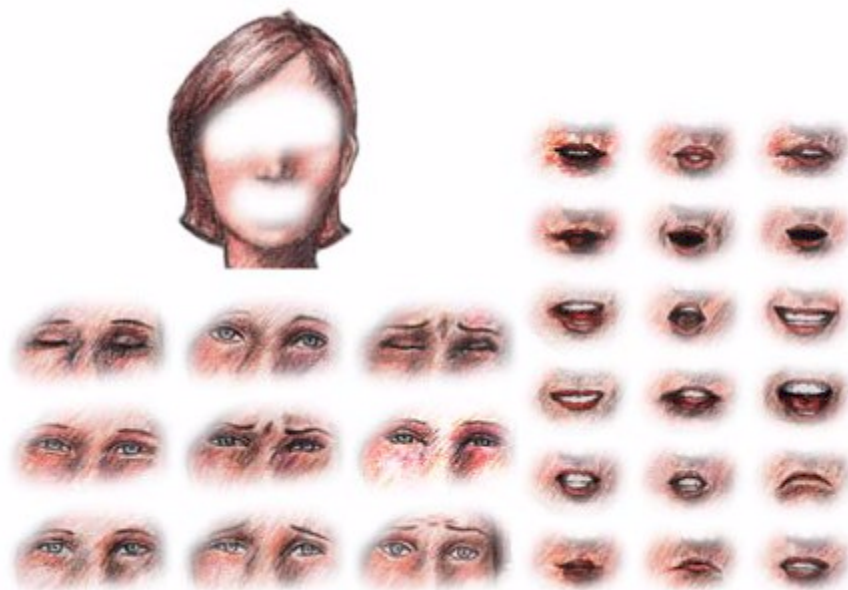


Figure 3: The artwork used for Seims *et al.* Notice the wide set of expressions for eyes and mouths which are independent of each other. Compositing the output animated face offers a wide range combinations for expression.

## 3.2 Non-Photorealistic Performance-Driven Facial Animation

The basis of this work was derived from two primary sources. The first work was done by Seims *et al.* on *Non-Photorealistic Performance-driven Facial Animation* (See Appendix B for a complete copy of their paper). Their research explored related objectives with a similar framework of facial tracker, expression mapper, and rendering engine. However, each of these components were implemented with a different approach which prohibits it from achieving the goals of our work. They begin with a set of hand-drawn art images which span the expressions for the animated character. Focusing on mouth and eye expressions, the artist draws separate eye and mouth images (see Figure 3).

Using a feature based video tracker, the Seims *et al.* approach associates each

artwork with specific video tracking data for that expression which is collected in a training phase. The resulting associations allow incoming live tracking data to be compared with each artwork for computing which artworks to use for output. The output is generated through a polymorph as described in Lee *et al.* [4].

The results of Seims *et al.* provide quite impressive animations, yet they still suffer from performance problems which prevent this from working in real-time. Their feature tracking takes 0.5 secs per frame on a 266MHz Pentium and an additional 15 seconds to render each frame, clearly well above target 10fps of our system. While the general approach of their work shows much promise, the algorithms are too computationally intensive for teleconferencing.

However, many aspects of their work provided a starting point for this project. First, the association of hand-drawn art with tracking data gathered in a training process provides a useful way to classify the art. The training data provides a way of comparing the live tracking data with the existing artworks independent of the content or the structure of the artwork. This allows the animated character's features to be independent from the subject's appearance. For example, the animated character's mouth may be of a different shape or size than the users. However, since the expression mapping only considers the training values which are directly taken from the video data, the algorithm will still select the correct artworks to be rendered.<sup>1</sup> This work is largely an extension of that paper and it utilizes many of the same techniques (eye and mouth independence, morphing the output animations, etc). Because producing real-time system implementation is critical for teleconferencing, many of our approaches have to be redesigned and implemented.

### 3.3 Face Driven Animation

The other main source for this project was work done by the author in the area of Face Driven Animation (See Appendix C for the complete reference). The primary

---

<sup>1</sup>Unlike the expression mapper, the rendering engine is highly dependent on the shape of artworks facial features. However, since the only input into the rendering engine is which faces to use and the corresponding weight, both which are generated by the expression mapper, the rendering engine is unaffected by these differences.



Figure 4: The results of Seims *et al.* produce a smooth animation which is quite impressive. This figure demonstrates the range of expressions the animated character can perform.



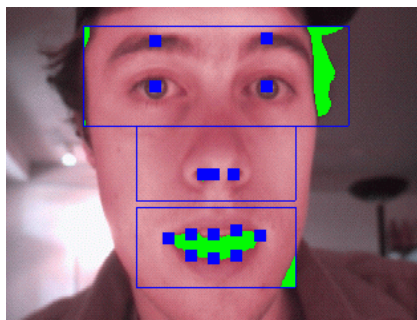


Figure 5: Face Driven Animation. The tracking program demonstrated on the left is capable of tracking 8 points around the mouth, both nostrils, pupil, and eyebrows. The tracking information was applied to a simple warp of a static image to simulate mouth movement. While the rendering results are simplistic, the tracker technology provides a faster alternative than Seims *et al.*

focus of that work was to develop a real-time video tracker which could record the positions of important facial features such as eye position, mouth shape, and eyebrow height, and apply them to a simple polygon model. The emphasis was to show that real-time video tracking can be performed without requiring the user to wear tracking aids. While the animated output of the tracker was quite simplistic, the work demonstrated that video tracking was definitely possible at 30fps and that it required only a fraction of the processor time, as shown in Figure 5.

# Chapter 4

## System Components

### 4.1 Overview

The system is broken up into three separate components: the **Feature Tracker**, the **Expression Mapper**, and the **Rendering Engine**. Each of these parts is placed in series and is delegated different tasks for converting camera frames into the animated output. A basic protocol exists between each of the three stages which allows each component to act totally independently of each other. The protocol could easily be implemented as a socket layer allowing for a fully networked implementation. In this chapter, we explore the approach taken in each component and the resulting algorithm.

### 4.2 Feature Tracking

#### 4.2.1 Introduction and Goals

Dealing with the video input stream is the first action the program needs to perform in order to feed the rest of the system. In defining the goals for the tracker, we looked at traditional teleconferencing applications, and used their system requirements as our reference point for our system.

The primary goal for the tracker is to detect enough information from the face in

order represent the different expressions, such as surprise, happiness, frowning, and anger, as well as other gestures performed in regular dialog.

For a practical teleconferencing program, the tracker must not require the use of artificial markers or sensors placed on the user's face. It must be an unobtrusive system that does not require additional burdens on the user beyond what is necessary for traditional video teleconferencing. Furthermore, the tracker must work on a variety of faces and be able to accommodate different facial structures. If custom code was required for each user on our system, the inflexibility of our technology would make it impractical for use. Finally, the tracker must in work under real-time conditions. There should be enough CPU cycles available after the feature tracking in order to perform the rest of the system tasks at the desired frame rate.

### 4.2.2 Assumptions

Since the overall goal of this project is not the development of a completely versatile real-time tracker, we made basic assumptions to simplify the design.

1. The user's head is in clear view of the camera and is generally looking into the camera. Some head turn is allowed, however both eyes should be clearly seen by the camera. This aids in the searching for facial features since they are assumed to always be present in the image.
2. Lighting conditions are assumed to be illuminating the face completely. This assumption allows the tracker to make full use of the color range of the camera and resulting color information in the image. Furthermore, it can be assumed that there are not large shadows appearing across the face. While minor shadowing is permissible and unavoidable, features like the eyes are not masked inside of shadows casts from lighting that may be directly overhead.
3. In order to make searching for the mouth simpler, the user is assumed to have a clean shaven face and a relatively mild complexion. Preliminary work showed that finding the mouth in a clean complexion was much easier than inside of

a bearded or whiskered face. This is probably the most restrictive assumption and would probably need to be changed for a real system.

By asserting these assumptions, the development of the tracker was made somewhat simpler. While a perfect solution would not impose such restraints, these allowed the direct use of the work explored in *Face Driven Animation*, Appendix C. Furthermore, plenty of related work in facial feature tracking lifts many of the assumptions while still performing in real-time.<sup>1</sup>

### 4.2.3 Design

The basis for the tracker relies on (a) assumptions about the structure of the human face, (b) color based searching techniques, and (c) the temporal coherence of streaming video data. In order to make the tracking fast, we made basic assumptions about the human face, i.e. the eyebrows are above the eyes, the mouth is below the eyes, etc. This permits the feature information that is discovered to be used to find other features. Secondly, the most obvious and basic data that can be extracted from the video frame are the color values. Desktop cameras often provide video streams which are uncompressed RGB data (although sometimes BGR, or YUV)<sup>2</sup> which is readily available to the program. This data is used to look for the red color of the lips or the darkness of the pupils. Also, the design takes advantage of the temporal nature to the video data to minimize the amount of searching that needs to be done on a frame by frame basis. Since the mouth position is unlikely to change dramatically between frames, we can limit the domain of the mouth search to only areas which are close to the previous frames results.

The algorithm tracks the following features:

- The x and y values each pupil (4 values).
- The distance between the upper and lower eyelids of each eye (2 values).

---

<sup>1</sup>See SIGGraph Submission, Appendix A for a further discussion of related facial tracking work.

<sup>2</sup>The endian of the machine usually dictates the byte order of the color values. On a x86 processor, most cameras output BGR (or ABGR) while MIPS based machines such as SGI workstations output RGB ordered data.

- The height of each eyebrow relative to the pupil (2 values).
- The distance between the left and right corners of the mouth (1 value).
- The height of the upper and lower lips, relative to the mouth center (2 values).

These were chosen to be both easily trackable and to represent a significant amount of expression in the subject's face. For example, the tilt of the head can be calculated from the angle of the line segment connecting the two pupils. Also, the points around the mouth offer enough information to infer the different mouth positions such as smiling, or frowning, or mouth openness.

#### 4.2.4 Mouth Tracking

The algorithm begins by searching for the mouth inside of a box which is determined from the search results of the previous frame. (In the case of the first frame, this box is defined to be encompassing a significantly larger area.) Before any searching is done, a simple image filter is applied to each pixel inside of the mouth box to locate areas which might be part of the mouth. This filter is designed to test the threshold of the normalized red component compared to the blue and green components of each pixel. If the comparison yields that the pixel is significantly red, the filter returns a *hit*, otherwise it is a *miss*. Also if the pixel's luminance is below a threshold, the filter will also return *hit*.

This filter has the effect indicating which pixels belong to the red lips and dark region inside of the mouth. The color normalization is done in order to make the filter independent of the luminance of the pixel. Under different lighting conditions or user complexions, the brightness of the skin and lips may vary significantly, but by dividing by the brightness of a pixel this is variation factored out.

After all of the pixels are tested, searching is performed on the result data from the filter rather than the image itself. It begins with finding a top and bottom of the mouth using the average of the x and y values of the filter hits. We seek from the bottom of the box upward until a significant clustering of the hits is found, indicating the bottom lip. The same is repeated for the top of the mouth. Once the top and

```

register int itot = PIXEL_LUMINANCE(r,g,b);
if (itot) {
    r = (cthreshold * (r)) / itot;
    g = (cthreshold * (g)) / itot;
    b = (cthreshold * (b)) / itot;
}
return (r < g || itot < threshold);

```

Figure 6: The image filter for locating mouth pixels. The function returns true if the pixel is either significantly red (implying a red lip) or its intensity is below a threshold (implying a dark area in the mouth). The constants **cthreshold** and **threshold** are adjustable through keyboard input to tweak the accuracy of the tracker.

bottom are found, the algorithm then computes the center of the mouth by averaging these points.

Next, the left and right points of the mouth are found. This begins by performing the same linear search from the edge of the box and walking toward the center until a clustering of hits is found. The algorithm then walks along the edge of the lip away from the center to find the corners of the mouth. At this point the top and bottom are recomputed as before but using instead the average x value of the corners. This allows for a better calculation of the top and bottom since almost all mouth expressions are symmetric. Finally as a last step, the box containing the mouth is recalculated for use in the next frame. The dimensions and position are derived from the box defined tracked points made 20% larger and saved for the next frame.

### 4.2.5 Eye Tracking

Next, the algorithm determines the eye, eyelid, and eyebrow features. Initial testing of tracking methods showed that in almost all cases, the pupils were the darkest points in the entire image. Due to the almost complete light absorption of the pupils, the darkest pixels around the area of the eye can be assumed to be the pupils. To calculate the darkest pixels, the eye tracker uses a filter to hit on dark pixels. See Figure 8.

Note that this filter does not compute the pixel luminance but rather simply sums

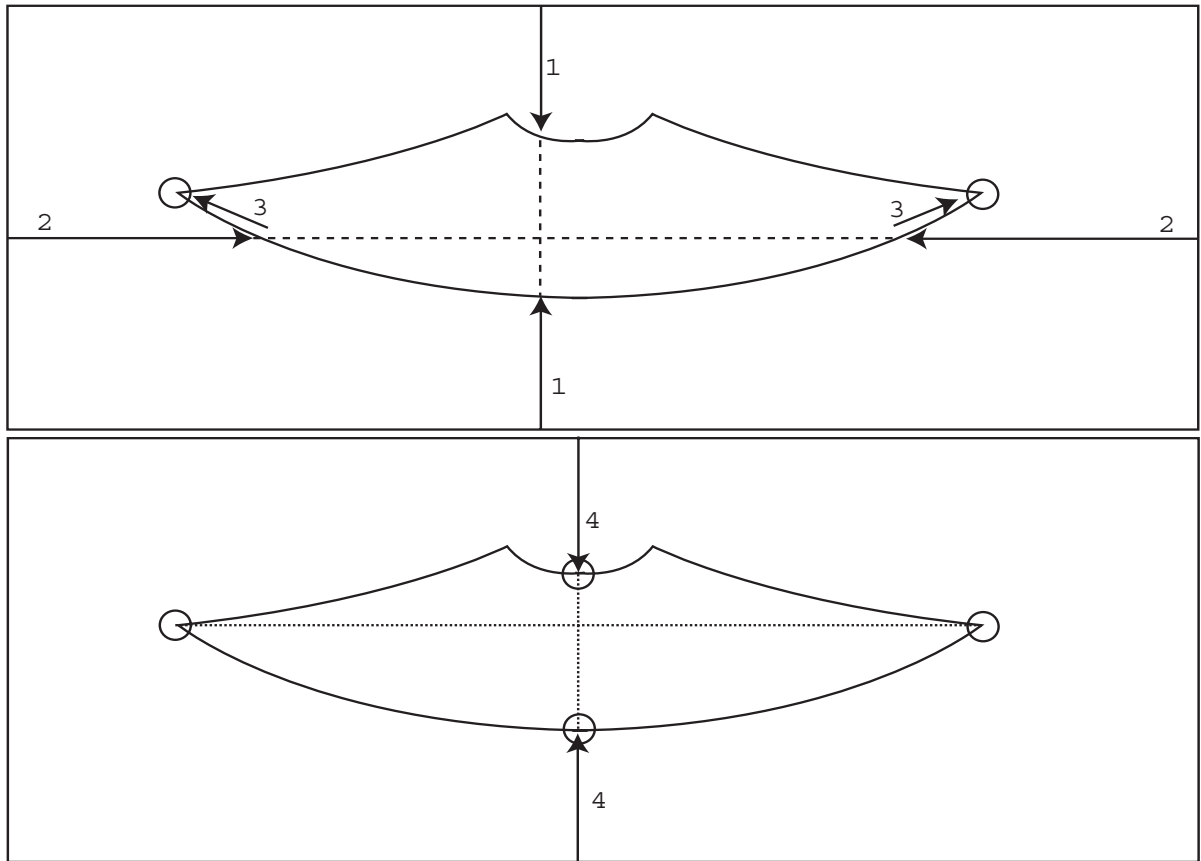


Figure 7: The mouth searching algorithm steps. In the upper figure: (1) Search from the top and bottom of the mouth box to find the upper and lower lip. (2) Using the midpoint of the upper and lower lip locations, as shown by the dotted line, search in from the right and left to find the approximate endpoints of the mouth. (3) Walk along the edge of the mouth to find the actual corners. In the bottom figure: (4) Using the midpoint of the corners, repeat the top/bottom search to find the actual upper and lower lip.

```
register int itot = r + g + b;
return (itot < threshold);
```

Figure 8: The eye tracking filter. A much simpler version of the mouth filter, this code simply checks to see if the sum of the color vector is below a certain threshold.



Figure 9: This image was generated by taking the sum of the color vector at each pixel and assigning the value to the luminance of the pixel. Notice the high contrast of the image and the dark pupils.

the different color components. The resulting effect is a bias toward the blue and red components over the green. This yields a higher contrast image and brings out the black of the eyes.

The algorithm begins with an eye box which defines a searching domain that is assumed to contain the two pupils. To find the pupils, the above filter is applied to each point in the eyebox and the hits are recorded. Next, the algorithm searches the hits starting from the center of the eyebox outward until a clustering of hits is found which is assumed to be the pupil.

Once the pupils are found, the algorithm seeks vertically to first find where the eyelid meets the eye and eventually the eyebrow. This is done with a similar intensity based search with a less sensitive threshold in order to find the dark points. Finally, the eye box is recomputed using the newly tracked pupil locations as shown in Figure 10.

### **Blinking**

If the search fails to find the pupils, the algorithm assumes that the user is blinking. The values for the pupil position are held constant from the previous frame and



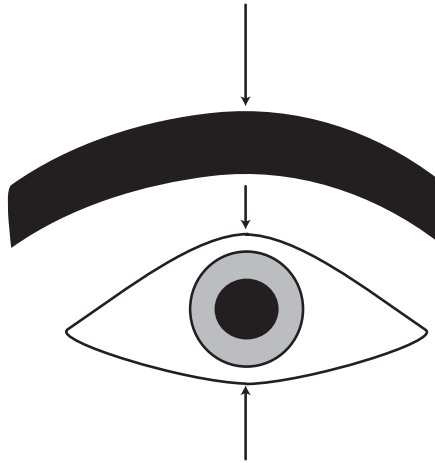


Figure 10: The eye searching algorithm uses the detected pupil location to locate the edges of the eye lid and the eyebrow height. Each search is performed as a vertical walk toward the pupil.

only the eyebrow position is recalculated. While this provides a suitable form of blink detection, the loss of pupil information can prevent related calculations from updating. For example, since the head position is calculated from the pupil location, when the user blinks their head will be fixed for the duration of the blink. In most cases the effects from this are hardly noticeable since blinking occurs in under a 1/10th of a second. However, if the eyes are closed for an extended amount of time, the tracker will output information that is significantly different from what the camera sees.

## 4.3 Expression Mapping

### 4.3.1 Introduction and Goals

The next phase of the system is the expression mapping. The purpose of this component is to determine from the tracking data which pieces of artwork should be combined to generate the output animated frame. Like all the different components of the system, it must perform quickly, leaving plenty of CPU time available for the

other parts of the system. The expression mapper's output should only specify a small number of art images to blend otherwise the rendering component may not be able to morph them all in real-time. Finally, the expression mapper's output should be smooth; if the user slowly opens their mouth from a closed position, the final output should yield results void of sudden changes in rendered image.

### 4.3.2 Approach

To design this component, we can analyze its function as a solution for scattered data interpolation. In this case, we have a set of artworks that represent the set of different mouths. We also have a stream of mouth tracking data and need to map a small subset of artworks to fit the data for a given frame. However, to perform the association, we need to provide the artwork with a quantitative value that can be compared with the tracked data. Therefore, we ask the user to perform each of the expressions appearing in the hand-drawn art and save the tracking values for each.<sup>3</sup> Now we have the tracking values of each of the artworks which can be compared input tracked stream. As we explain below, however, more calculations need to be performed on this data before we can begin to use it in the expression mapper.

The simplest form of expression mapping is to treat the tracking data as a  $m$  dimensional vector and to compute the distance from the live frame to all the artworks. Then weights could be applied to each depending on how close they were to the frame data. The problem is that the rendering component would have to morph all the artworks every frame, an expensive process.

The next step might be to use just the closest three of the artworks and just ask the rendering process to blend those. However, this could lead to excessive jumpiness if the live data was near a set of three or more artworks that had distances all nearly equal to the live data. For example, if four points were all near the live data, a minor change in the live data could cause the resulting output animation to change dramatically as shown in Figure 11. Furthermore, using only the nearest neighbors may not properly combine to yield the of the target face. This is shown in Figure 12

---

<sup>3</sup>This training processes is similar to Seimes *et al.*

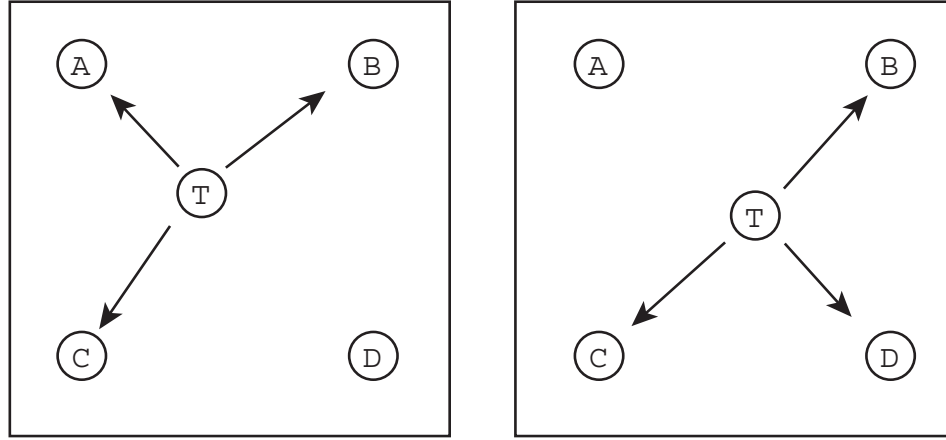


Figure 11: Selecting the closest three artworks for determining which faces to create the output face is subject to jittering problems as shown in this figure. Consider the four faces A, B, C, D arranged equidistant from each other. T is the target face the expression mapper is trying to represent. If T is placed equidistant to all four except slightly closer to face A, as shown on the left, the resulting output would be to combine faces A, B, and C with close to the same weights. But if T were to shifted slightly away from A to toward D, as shown on the right, the resulting output would be to combine B, C, and D equally. This is a dramatic change in the output image was the result of a very small change in the expression. This could produce unwanted changes in the output images.

where the closest three faces are all on one side of the target face.

Ideally we would like the expression mapper to have a smooth continuous transformation as a subject changes expression and a weighted combination which is close to the position of T. To solve this problem we can reduce the expression data into two dimensions by projecting the artwork tracking values onto the plane. This is done through a technique called Primary Component Analysis (PCA), by calculating the two largest eighenvectors of the artwork tracking data. We then calculate the components of tracking data in the direction of these two vectors for our two dimensional projection. Using the eighenvectors will cause the tracking data to be distinguished in the direction that is most differentiating.

Next, since we want to use only three pieces of artwork for creating the output frames, we triangulate the placements of each of the artworks in the projected plane

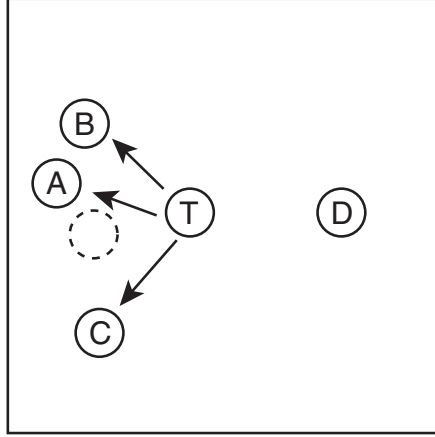


Figure 12: Selecting just the closest three may cause differences in the target image and the output image. In this example, faces A, B, and C are all closer to the target face than face D. However the weighted combination of the three produces a face, shown as the dotted circle, which does not have the same location at T. Ideally we would like to incorporate face D in the in the output such that the weighted combination would be close to T.

using a Delauney triangulation. For incoming live tracking data, the PCA projection is applied with the same eigenvectors to bring the values into the 2D plane of the artwork data. Whichever triangle the projected tracked live data falls into, the artwork at the vertices of the selected triangle are passed onto the rendering engine.

For selecting the weights of the different artworks, we use barycentric coordinates within the triangle as referenced in Polymorph [4]. Using this basis, the expression mapper does not suffer from the problem of the nearest artwork algorithm since moving across an edge of the triangle will yield a smooth transition and the correct weights. See Figure 13.

The algorithm uses separate triangulations for the eye and mouth artworks. Since in general the eye and mouth behave independently of each other, the design separates the two and computes separate outputs. All the video data that is not used for expression mapping, (i.e. the pupil locations) are passed straight through to the rendering component.

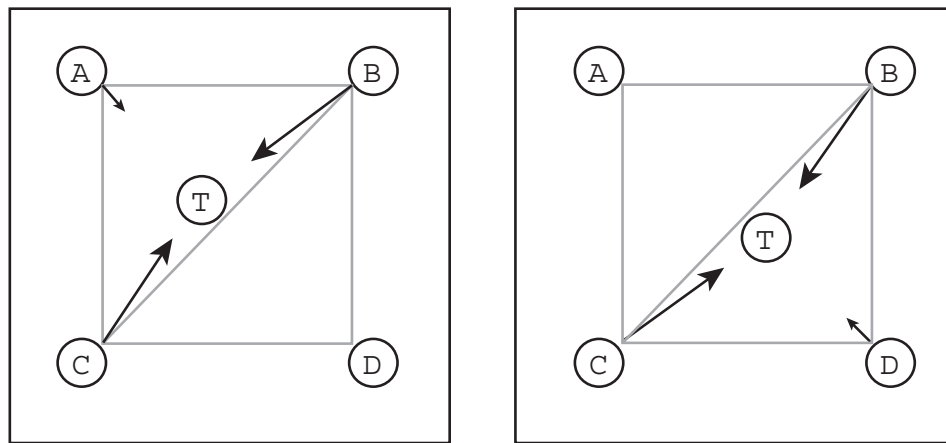


Figure 13: By using barycentric coordinates the problem of jitter and inaccurate weights is eliminated. On the left, the target face,  $T$ , falls inside the triangle  $ABC$ . Since it is close to the segment  $BC$ , the target image is assigned the barycentric coordinates with larger weights for  $B$  and  $C$  and a smaller weight for  $A$ . On the right,  $T$  has moved slightly into the triangle  $BCD$ . Yet since  $T$  is still close to  $BC$ , the resulting weights are predominantly  $B$  and  $C$  and very little of  $D$ . This smooth transition across two triangles elevates the problem of jitter present in the nearest artwork algorithm. Furthermore, the barycentric coordinates yield the correct weighted average which results in a face at the location  $T$ .

## 4.4 Rendering Engine

### 4.4.1 Introduction and Goals

The final component of the system is the rendering engine. The purpose of this component is to take the information from the expression mapper and render the actual output animation. The rendering output must be performed in real-time and void of visual artifacts. Of course, the rendered artwork should appear as if the artist themselves drew it. The basis for the rendering algorithm that is derived below has been used in other systems, such as the *Polymorph* paper by Lee *et al.* [4].

### 4.4.2 Composition

For each animated frame, we need to update different parts of the face separately since they are allowed to change independently of each other. The expression mapper specifies three artworks and weights for both the mouth and the eyes to be combined to produce a believable result. Furthermore, the rendering engine needs to incorporate a background head to be composited together with the eyes and mouths to create a complete head. (Since the same algorithm is applied to creating the final mouth images as the eye output, the following sections will discuss the general algorithm for one with the implication that it is applied to both features).

### 4.4.3 Morphing the Images

The rendering engine must take the three hand-drawn images and weights provided by the expression mapper and create an output that appears to be proper combination of expressions. The simplest implementation would be to blend the three images on top of each other with the weights proportioned to the alpha values. However this would lead to definite ghosting since the input artwork features would not line up.

The proper solution would be to use image morphing techniques to feature align the three artworks so that they may be properly blended. In order to properly morph the images, we needed to devise an algorithm that would be able to morph three images together with the proper weights, and to do so in real-time.

Traditional feature based morphing algorithms could produce the type of rendering output that is desired for this system. Beier-Neely morphing consists of a set of feature lines used to identify different features in a pair of images[6]. Morphing is done by interpolating the feature lines and resampling the image relative to the feature lines. Other works have extended this algorithm to allow multiple inputs by simply interpolating the different feature lines across three input images rather than just two [4].

The problem with most traditional morphing algorithms is that they are not designed to run in real-time. For example, the common Beier-Neely morph must perform a slew of calculations per pixel to create the output image which is much too computationally intensive for a real-time system. For our system to work in real-time, we need to make further enhancements to the Beier-Neely algorithm. The solution was to precompute most of the morphing calculations prior to actually rendering as described below.

#### 4.4.4 Morphing Algorithm

Instead of computing a complete morph on each frame of the animation, the algorithm computes the morph as part of a preprocessing phase and uses linear interpolation of the morphed data to create the output frame. To understand the process, consider a traditional two image morph, where we would like to create an image,  $A'$ , which is half way to between  $A$  and  $B$ . First we indicate feature lines indicate important features in the two images. We next interpolate the feature lines and perform a traditional reverse mapping morph, evaluating each pixel in the output image  $A'$  by sampling  $A$  and  $B$  relative to the feature vectors, as shown in Figure 14.

While the results of a reverse mapping morph provide a seamless transition between two target images, the cost of computing the morphed images is quite expensive. Every pixel of the target image must be processed with a series of vector math operations. For image resolutions that are at all practical, Beier-Neely morphing can quickly result in a sizeable amount of computing time for each frame of the morph. Without any changes to the algorithm, it is impractical for a real-time system.

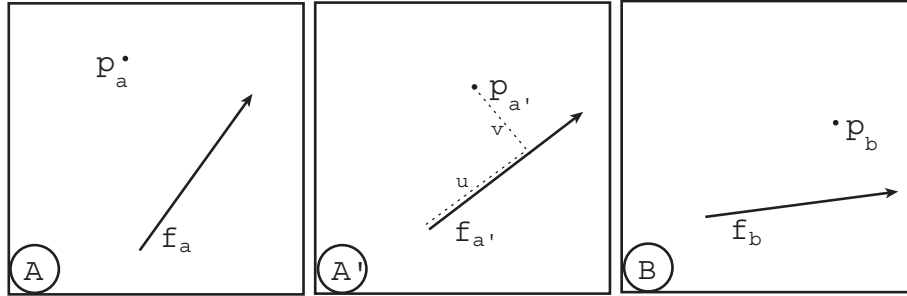


Figure 14: Traditional Beier-Neely morphing using reverse mapping. To compute an image which is a weighted combination of A and B, we define vectors across each image which signify important features to preserve (vector  $f_a$  and  $f_b$ ). Next we compute the weighted combination of the feature vectors to produce  $f_{a'}$ . To compute  $A'$ , we calculate each pixel of  $A'$  relative position to  $f_{a'}$ , as shown by  $u$  and  $v$ . We then sample image A and B at the position which is relative to feature lines  $f_a$  and  $f_b$ , using  $u$  and  $v$  to find  $p_a$  and  $p_b$ . The two color values are then blended to form  $p_{a'}$ . Also further weighting can be applied such that only pixels which are near the feature vectors are affected.

### Grid Sampling

One method for reducing the computations is to apply Beier-Neely morphing to a subset of the image pixels and interpolating the color values of the remaining pixels. This optimization can take advantage of traditional texture mapping hardware available on most modern graphics cards. The input image is loaded as a texture for a grid of points with a smaller resolution than the input image. The Beier-Neely morph is performed on the texture coordinates of each of the grid points, as shown in Figure 15. By applying the input image as a texture with morphed texture coordinates, the graphics hardware fills in the remaining pixels with interpolated values from the morphed points. As a result, the morphed image is generated with many fewer calculations and appears almost indistinguishable from the pixel level Beier-Neely morph.

While sample grid calculations may reduce the number of calculations necessary for each morphed frame, it is not enough of a simplification to reduce the computation time to use in real-time. Furthermore, if the sample grid is too coarse, the morphed image deviates significantly from a full pixel-level Beier-Neely morph. This is due



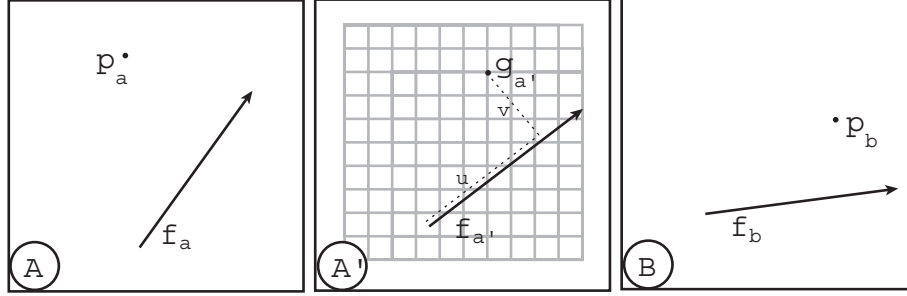


Figure 15: Grid sampling using Beier-Neely morphing reduces the amount of calculations required by sampling a grid of points that has a lower resolution than the actual image. To create  $A'$ , we perform a morph on the texture coordinates for each of the input images  $A$  and  $B$ . The two are blended together by rendering two morphed meshes on top of each other.

to the majority of pixels that are interpolated by texture mapping hardware which causes the feature lines to no longer have the precise control over the features in the image. To improve the morphing to real-time levels, the algorithm must be modified further to reduce the computation time.

### Interpolated Beier-Neely

To eliminate the computation expensive vector math, we apply the same subsampling optimization done with the grid technique to calculate the intermediate morphs between image  $A$  and  $B$ . This can be done by precomputing a Beier-Neely morph of image  $A$  using the feature vectors image  $B$ . The morphed texture coordinates can be saved to later create an intermediate morph frame from  $A$  to  $B$ . For example, to generate a frame that is half way between  $A$  and  $B$ , we can interpolate the values of the texture coordinates between the morphed  $A$  and the unmorphed  $A$ , as shown in Figure 16.<sup>4</sup> This technique was first published by Lee *et al.*[4].

Using this method, we only need to calculate a scaled vector per sample point, thus dramatically reducing the per-pixel math required. Furthermore, in the precompute

<sup>4</sup>Actually the factor should be one minus the contribution since for a morph which is mostly  $B$ ,  $A$ 's contribution is small. Yet  $A$  should be morphed inversely proportionally toward  $B$ , therefore  $w$  should be large.

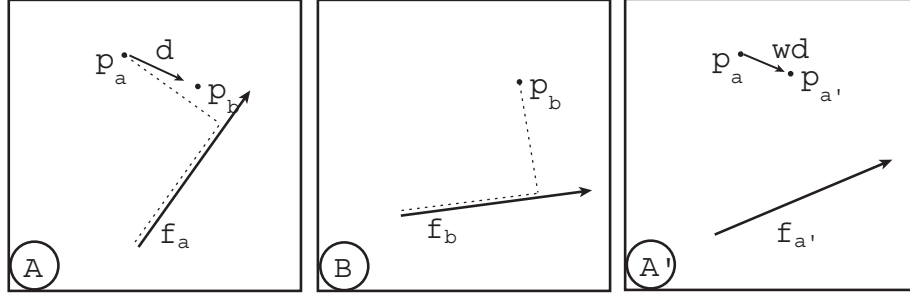


Figure 16: Precomputing the morph. In a precompute phase, we calculate the morphed texture coordinates for image  $A$  morphed to the feature lines of  $B$ . From these values, we compute the vector  $d$  which represents the change in value of the texture coordinate for that morph. To generate a morph image  $A'$  which is part way between  $A$  and  $B$ , we can compute the morphed texture coordinates for  $A'$ 's contribution by simply interpolating the vector  $d$  by the contribution  $w$ .

phase, only a single Beier-Neely morph needs to be calculated to generate the rest of the morphed frames.

It is important to note that this method does not generate the same morphed frames as a complete Beier-Neely calculation. The interpolated vector forces features in the image to move linearly in the morph. However, with the full Beier-Neely morph, features can progress in morphed frames with paths that are not necessarily linear due to feature vector rotation and interaction with other vectors.

#### 4.4.5 Optimized Algorithm

The rendering engine takes advantage of both grid sampling and the interpolated frames to generate the morphed frames in real-time. First, the algorithm preprocesses the input images to generate the morphed texture values. The full Beier-Neely morph is performed on each of the images that are connected in the triangulation generated by the expression mapper. Also, morph values must be calculated in both directions ( $A$  morphed to  $B$  and  $B$  morphed to  $A$ ) since the results from a Beier-Neely morph are asymmetric.

Once the morphing values are precomputed, the system can output interpolated

morph frames. The expression mapper dictates to the rendering engine which artworks should be morphed together with their associated weights. To perform a three-way morph, the algorithm morphs each of the three images individually. The weighted combination of the morphed texture values produces the target morph to which the artwork can be rendered.

$$\begin{aligned}
 (s, v)_{frame} = & a_{weight} * (s, v)_{Unmorphed} + \\
 & b_{weight} * (s, v)_{A \text{ morphed to } B} + \\
 & c_{weight} * (s, v)_{A \text{ morphed to } C}
 \end{aligned} \tag{1}$$

$(s, v)$  represents the texture coordinates at each grid point. This calculation is repeated for each grid point of the three input images to compute texture coordinates. The resulting meshes are feature aligned morphs which are ready to be composited.

#### 4.4.6 Blending

The morphing algorithm is applied both to the eye images and to the mouth images independently. The 2 images must then be blended together with a background image. First, the eyes and mouths are blended together using hardware alpha blending. Next, the rendering engine uses a preloaded background face with an alpha mask such that the eyes and mouth areas are transparent. The background is placed on top of the eyes and mouth to complete the rendered face, as shown in Figure 17.

To improve the temporal characteristics of the output animation, a different background head is used on odd and even frames. The two backgrounds have identical shape but were drawn by the artist separately so that the strokes would be different. This prevents the background frame from remaining static while the eyes and mouth update. As a result, it appears as if each frame was hand drawn.<sup>5</sup>

---

<sup>5</sup>There are some artistic styles where the background does not change every frame. The existing system allows for static background heads but care should be taken as to not reveal the mask and produce undesirable effects

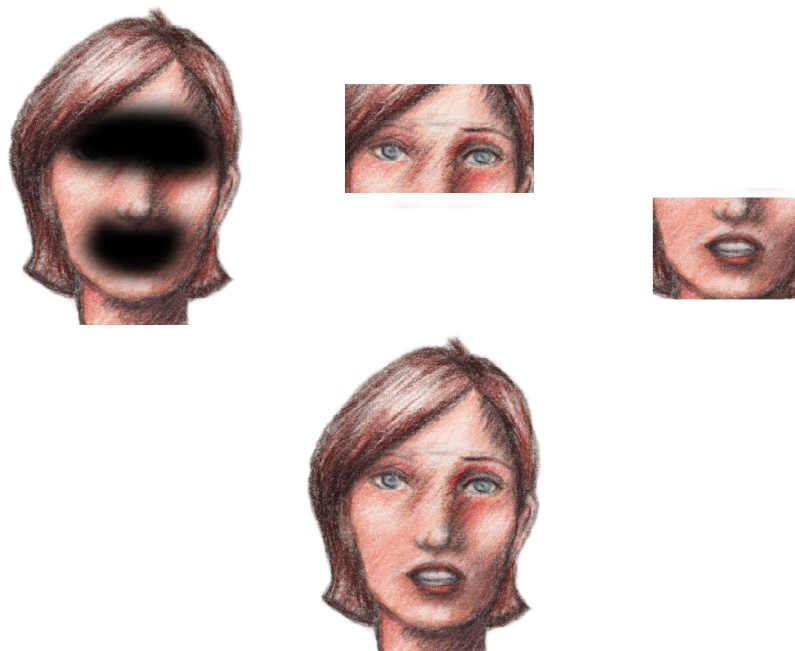


Figure 17: The blending of the rendering engine combines the three components of the animated face. After the eyes and the mouth are rendered, a background face is applied with the proper mask shown on the left to create the output face below.

### 4.4.7 Artwork Normalization

One problem that was immediately noticed in preliminary testing of the system was that, despite careful artist drawings, variations in the different drawings became quite apparent when rendered. For example, the location of the eyes in the images all tended to vary within each of the different artworks. As a result, when the rendering engine was morphing between different gestures, the eyes seemed to shift in the head. A similar effect was witnessed in the mouth.

To correct this problem, we normalized the artworks. For the eyes, we defined a local coordinate system that was based on the position of the eyes in that particular artwork (see Figure 18). Next we transformed all of the feature vectors into the normalized coordinate system based on the location of the two pupils. This aligned all the feature vectors such that any morphing would not result in unwanted translations of the eyes. Since the texture mapping grid was also defined in this coordinate system, an additional matrix multiply was required to return the image back into the screen space after the warping calculation was performed. A similar process was performed for the mouth, however, rotation was not preserved.

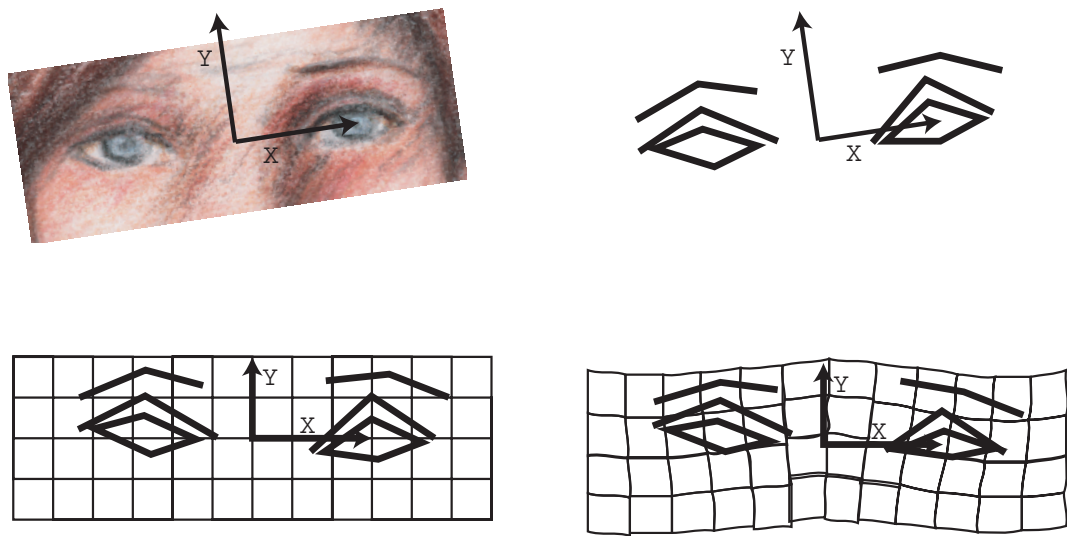


Figure 18: To normalize each eye, the X axis is defined to be from the center of the two pupils to the position of the right pupil. The Y direction is of unit length and is perpendicular to the X axis, as shown on top. The resulting transformation allows all of the eye artwork feature coordinates to be adjusted so that all the pupils are aligned. The rendering mesh, shown below, all have the same initial unwarped values which allows the artwork to be warped with other eyes that may be distorted differently. After computing the warp, the texture values are re-transformed back to the original space of the artwork for proper sampling.

# Chapter 5

## Implementation

With all three components connected together, the complete animation system can output a real-time video driven animation. However, while the general algorithm suffices to perform all the necessary tasks, this complex system requires an efficient implementation to output effective results. The following sections explain how the different components were implemented to create a effective working system. Although this is not a fully featured consumer application, it is important to understand how each of the three components work together so that future analysis will be able to leverage this design.

### 5.1 Tracker Implementation

The primary goal of the tracker implementation was to perform a fast and efficient search of the video obtained from the camera. The camera input is received as a pointer to a series of 24-bit RGB values per pixel. The implementation first processes the mouth area, followed by the eyes.

Although in a fully functioning facial animation system the video tracker would be a fully integrated component of the entire system, the existing implementation operates as a separate application. Video tracking is complicated and sensitive to things which a simple program cannot directly control, such as lighting or excessive motions by the subject. Since the code is designed to operate quickly and efficiently,

assumptions such as that the viewer is always in front of the camera help minimize time spent with error handling. Therefore, the implementation currently isolates the video tracking component separately from the rest of the system. This allows fine tuning of the tracking system on different video taped performances of users without hampering the rest of the system.

The video tracker has five constants that it uses to track the different components of the face: lip color, mouth intensity, pupil intensity, brow intensity, and eyelid intensity. Each of these affect the image filter that is used to detect trackable pixels. The values of each of these can be tweaked through the keyboard for a given video input sequence to maximize the tracker capabilities.

The tracker outputs two data files, `eyetrack.data` and `mouthtrack.data`. Each file contains the tracking data for that part of the tracker's results. `Eyetrack.data` contains the following information per frame:

- Left Brow Height
- Right Brow Height
- Average Eye Openness
- $(x, y)$  Left Pupil
- $(x, y)$  Right Pupil

`Mouthtrack.data` contains the following information per frame:

- Mouth Width
- Height of the Upper Lip
- Height of the Lower Lip

### 5.1.1 Video Source

To further aid in tracker development, the video input is loaded from disk instead of from a live or taped video feed. Loading the video frames from PPM image files makes reading in the video simpler for the tracker program and also makes the input video available to the rest of the components for display and comparison.



It is understood that a real system would have the tracker integrated with the other components in a seamless implementation. The rest of the system operates as a complete system with the video class being a simple stub class that loads the tracking data from the two files and passes them on to the rest of the system. The data is passed to the expression mapper through a video data structure below.

```
class VideoData {
public:
double eyePt[20];    // the tracking values for the eye
int eyeNum;          // the number of eye tracking values
double mouthPt[20]; // the tracking values for the mouth
int mouthNum;        // the number of mouth tracking values
double lefteye[2];   // the x,y coords for each eye.
double righteye[2];
};
```

## 5.2 Communication

The communication protocol currently is performed by the file system through the saving and loading of the two tracking files, `mouthtrack.data` and `eyetrack.data`. While a commercial application would incorporate an actual internet connection between sender and receiver, storing the tracking information in a file has distinct development advantages. The primary advantage is that saving the information allows it to be replayed exactly when the expression mapping and rendering components are run. This simplifies testing and development of both components.

## 5.3 Expression Mapper Implementation

The majority of the expression mapping implementation was written by Alison Klein ([awklein@cs.princeton.edu](mailto:awklein@cs.princeton.edu)) in both Matlab and C. The expression mapper has a preprocessing component as well as a real-time process. The preprocessor needs to compute the Delauney triangulation and projection matrix to be used by the real-time system. The real-time system takes the projection matrix and applies it to the video tracking data and computes which triangle it falls into to compute the weights.

### 5.3.1 Precomputation

The precomputing process is preformed entirely in MatLab. The tracking data of each of the artworks is inputted into MatLab through a data file that contains the tracking data and artwork ID numbers. The tracking data are the result of running the tracker on each of the expressions that the user was asked to perform in the training process. These tracking numbers are associated with artwork IDs. The IDs are identifiers that the rendering engine will use to distinguish one artwork from another.<sup>1</sup> (The origin of the artwork ID numbers will be explained later in the rendering implementation section.) Like most of the components, there are two separate files for the eye and mouth expression data.

The MatLab program performs the Primary Component Analysis and generates the x, y location of each artwork, identified by artwork ID, in the expression space. Also outputted is the triangulation information and the projection matrix to be used for transforming the tracking data down into the expression space. See Appendix D for examples of the input and output files of the preprocessor.

### 5.3.2 Run-time System

The run-time system loads the precomputed data file to initialize the expression mapper system. Implemented in C, this builds the Delauney triangulation. The expression mapper component actually contains within it two different instances of the math engines for computing the projection and triangle intersection; one is for the eye calculations, the other for the mouth calculations. Once it has received the tracking data from the facial tracker, the expression mapper passes the eye and mouth data to the different computations.

The output of the expression mapper run-time system contains all the information the rendering engine needs to output the next frame. Both the eyes and mouths have three artwork IDs and weights generated from the expression calculations. Weights range from 0.0 to 1.0 and the IDs correspond to the same identifiers used in the

---

<sup>1</sup>These identifiers could easily have been filenames instead of numbers. This would have resulted in a slightly more complex implementation but would have been easier to use.

precomputed data files. The expression data is then passed to the rendering engine through a basic data structure shown below

```
class ExprData {
public:
int numMouths; // the number of mouths to render
int numEyes;
int mouth[EXPRDATA_MAXDATA]; // the artwork IDs
int eye[EXPRDATA_MAXDATA];
float mouth_vals[EXPRDATA_MAXDATA]; // wieghts
float eye_vals[EXPRDATA_MAXDATA];
vec2 pos; // head position
float rotateang; // head rotation
vec2 featurecoords;
};
```

## 5.4 Rendering Engine Implementation

The rendering engine requires quite a bit of data to be loaded into the system before it can render the first frame. Not only do all the input artwork images need to be loaded into texture memory, but the feature lines must be loaded to compute the Beier-Neely morphs. Furthermore, we need to precompute all of our warps for use during real-time.

### 5.4.1 Morph Files

The input to the rendering engine is done though a configuration file which contains filenames of all the artworks used in the system with their corresponding feature vectors. The rendering configuration file contains the following information in ASCII format:

- Number of background images to cycle through
- Background images
- $X$  and  $Y$  rendering scale
- Mouth alpha mask image

- Eye alpha mask image
- number of input images
- Artwork filename
- Feature coordinates (not used)<sup>2</sup>
- Number of feature coordinates
- Number of vertices, Feature Specifier
- $(x, y)$  vertex

The configuration file contains all that is necessary for the rendering system to generate output frames. It is important to note that input artwork must be a factor of  $2^k$  to comply with OpenGL texture memory requirements. To make this issue less of a constraint, a separate x/y scale can be specified in the configuration file to correct for artwork stretching.

Each set of feature coordinates are specified by a series of x,y coordinates. These coordinates are offset from the origin in the center of the image width assumed dimensions of 496.0 by 560.0.<sup>3</sup> Also, feature vectors are specified as a set of linked segments since most of the features in the artwork are non-linear.

## 5.5 Using the System

### 5.5.1 Drawing Artwork

The first task in using the animated teleconferencing system is to draw artwork. Choosing a style of hand-drawn artwork should be done with an understanding of the constraints of the system. How well will the artwork style mask the blending and morphing artifacts? How should the background head update separately from the mouth and eyes? These are important questions when considering drawing styles. The original artwork chosen for this work was in the style of Bill Plympton, an

---

<sup>2</sup>The feature coordinates are generated by the expression mapper preprocessor. This entry remains for legacy reasons.

<sup>3</sup>This too was done for legacy reasons. The existing configuration file was based off of the previous data files written originally by Seimes, *et al.* See Appendix B

animator who specializes in color pencil art with a dynamic jitter that makes it quite appealing and works well within our system. The number of different expressions drawn can vary, however, the expressions covered in Figure 3 provided a good set for generating smooth animations.

### 5.5.2 Drawing Feature Vectors

In order for the rendering component to morph each of the artworks, feature vectors need to be drawn on each of the input artworks. These feature vectors must then be specified in the morph files. The choice of where to place the vectors is the user's. For most of the artworks used in this work, we placed the following feature vectors:

- 6 vectors for the outside of the mouth
- 6 vectors for the inside of the mouth
- 1 vector for the front teeth position
- 1 vector for the nose position
- 2 vectors for the mouth creases
- 2 vectors for each eyebrow
- 2 vectors for the upper eyelid
- 2 vectors for the lower eyelid
- 2 vectors for the upper eye socket.

The choice of feature lines is not restricted by the algorithm so the artist can use as many as they see fit as shown in Figure 19. All of the feature vectors should be stored in the `.morph` files.

### 5.5.3 Training

Next the system must be trained for a user's face. This involves asking the user to express each of the faces in the artworks, either individually or by performing a sequence of expressions in the form of a contrived sentence. The user then matches frames of the video with each of the artworks. The tracking data for the selected frames is fed through the MatLab program to produce the Delauney triangulation

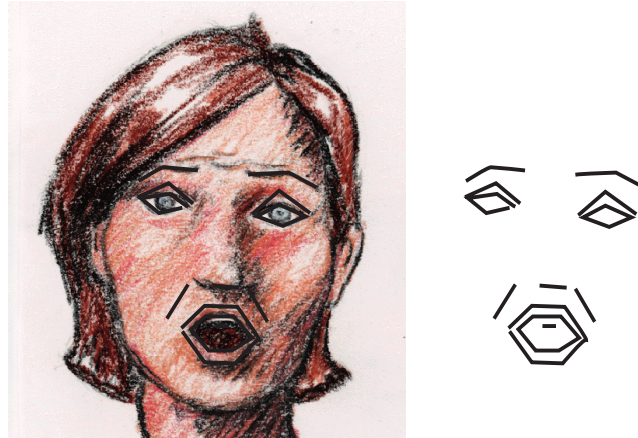


Figure 19: Specifying the feature vectors. A custom program was developed to aide in creating the morph files which contained the feature coordinates. On the left is the artwork combined with feature vectors. On the right, just the feature vectors are shown.

and PCA projection matrix. This information is placed in input file of the expression mapper to be loaded at run-time.

# Chapter 6

## Results

### 6.1 Tracker Analysis

The efficiency of the tracker allowed it perform quite well without using excessive CPU cycles. Since the tracker localizes the problem into different mouth and eye areas, most pixels are not processed by the tracker and the ones that are mostly only evaluated once. Figure 20 shows the tracker in action.

The performance of the tracker is quite impressive. The video images were captured using a standard Hi-8 camera and digitized by an Avid workstation, which outputted 30 frames for every second at 320x240 resolution. Leaving out the I/O of loading each frame from disk to memory, the tracker performs its operation in approximately 1ms on a 400MHz Pentium II.<sup>1</sup> Since it takes about 3ms to copy an entire video frame, which consists of 320x240 pixels at 24-bits per pixel (230Kb total), we can assume that the transfer from the camera into memory would take about the same. Therefore the whole tracking stage takes approximately 4ms per frame.

### 6.2 Expression Mapping

The expression mapper implementation allowed for a quick calculation of the morphing values. For most of the videos, only a small subset of artworks were used to create each

---

<sup>1</sup>All timing values will be given for a 400MHz Pentium II system.

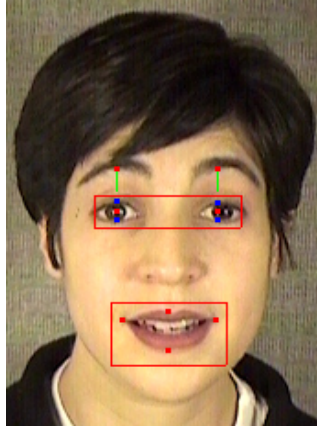


Figure 20: The feature tracker. Boxes define the search area of the mouth and eyes. The red and blue dots show the points which are tracked.

animation. This was to prevent excessive jumpiness in the output animation. Figure 21 illustrates one of the triangulations used for the Pam art.

### 6.3 Artwork

The artwork used for the results of this work were gathered from a variety of sources. There were 4 primary animated characters done for this work: **Meryl** a pencil line-art female character (Figure 22). **Pam** the colored pencil female character done in the style of Bill Plympton (Figure 3). **Blue Guy** is a pastel colored saddened male character (Figure 24). **Rob** is a three eyed monster done in charcoal (Figure 23). The different styles all worked quite well within the framework demonstrating that this application is applicable to many styles of artwork. The Rob art also demonstrated that the features of the animated art can differ significantly from a human face, as shown by Rob's three eyes, all of which were interactive.



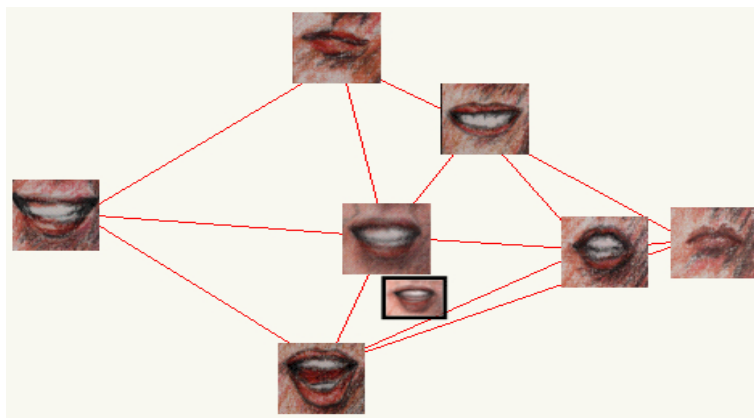


Figure 21: The expression mapper face triangulation. The horizontal and vertical axis represent the direction of most change as discovered by the Primary Component Analysis. A target mouth shown in the bottom right is created by merging the three mouths of the triangle which it falls into.

## 6.4 Rendering

Modifications were performed for different actors and artwork to improve the output. For example, the frame rate was increased for actors who talked fast, since at the default 10 fps, it can be difficult to capture all of the mouth movements. Updating the background art also was done differently for some of the art. In Pam, Meryl, and Blue Guy, the backgrounds were changed every frame. However, with Rob's green background, it was more effective not to update the background but to keep it static.

## 6.5 Performance

Table 1 shows the resulting times for each component of the system for a 400MHz Pentium II with a high-end PC graphics card. With the slowest component being the rasterization and blending, this will only get faster as graphics cards get better.

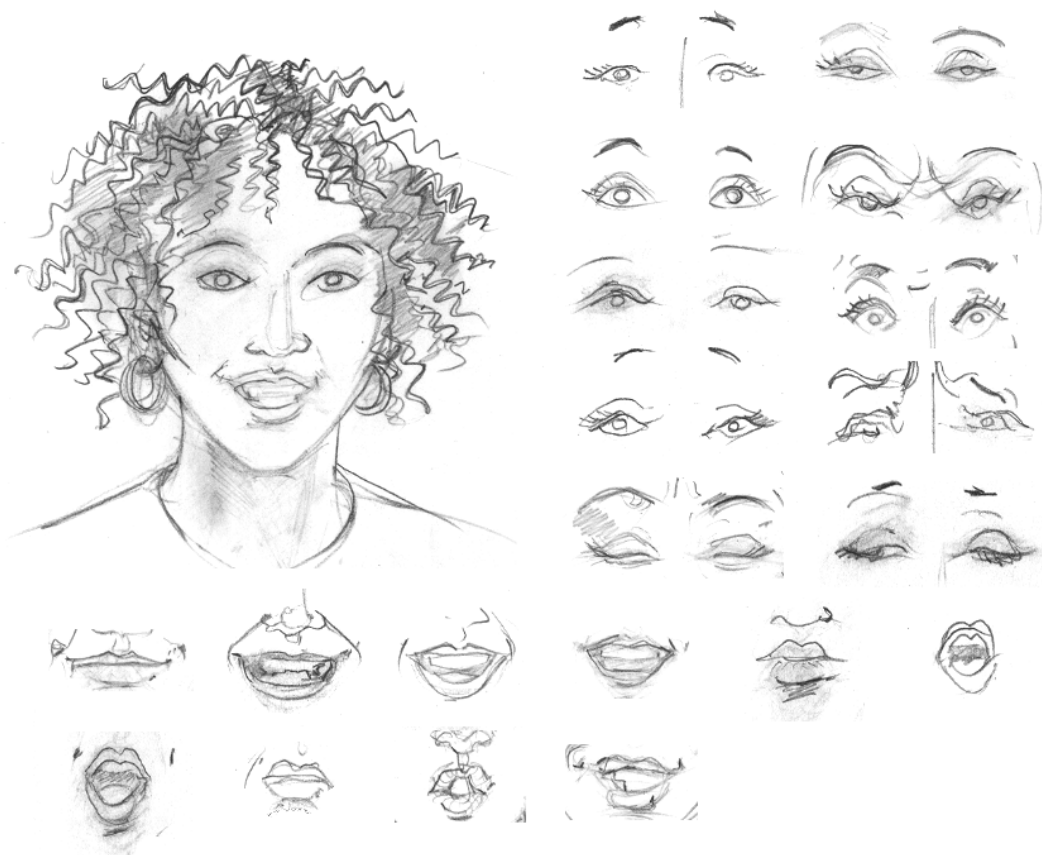


Figure 22: The Meryl Artwork.

Stage	Time (ms)
Copy 320x240 video frame to memory from camera	3
Track facial features in the frame	1
Project into feature space and find blending weights	0
Compute warps for 3 mouths and 3 eyes	3
Render 3 mouths, 3 eyes, and head at 640x480	26
Total	33

Table 1: Running times (400MHz Pentium II) for various stages of the pipeline.

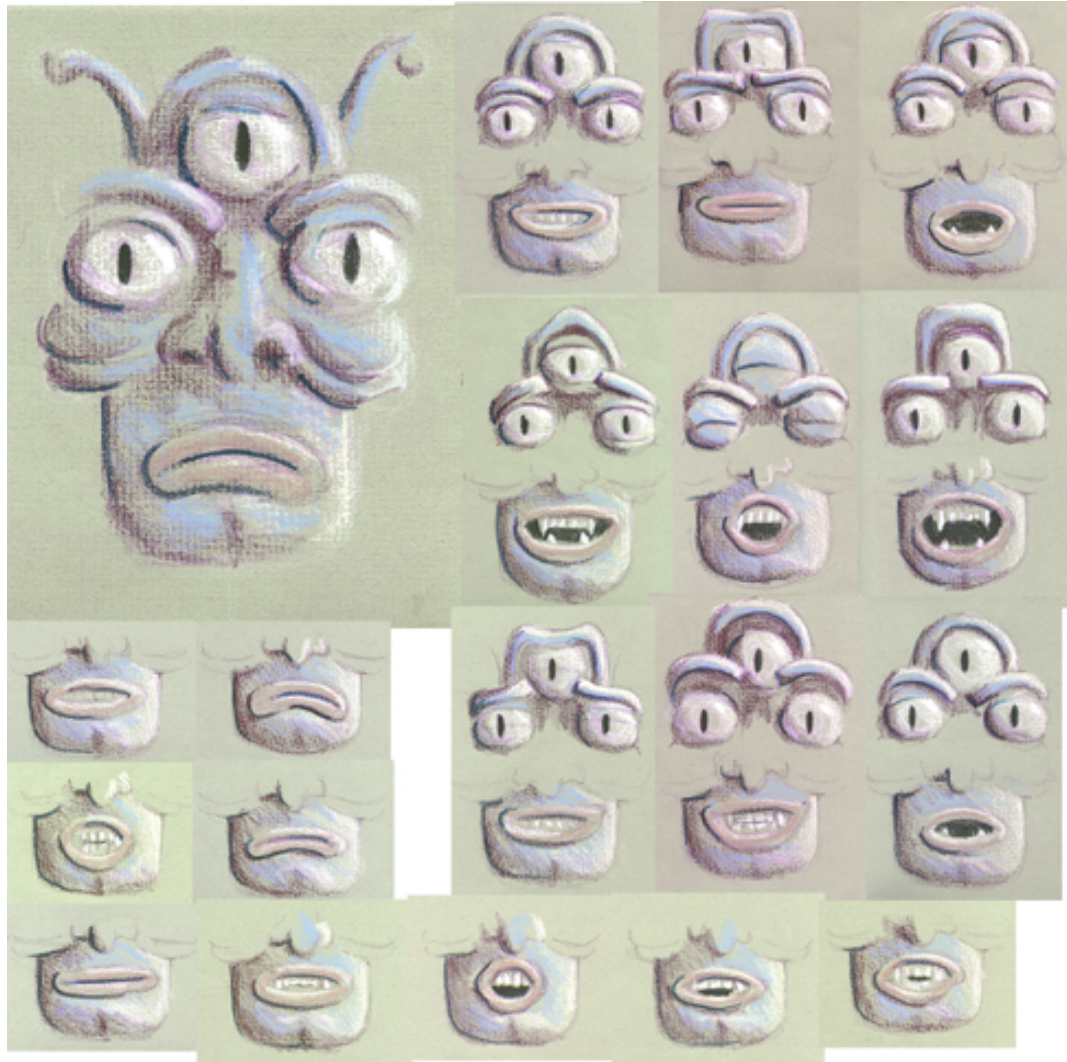


Figure 23: The Rob Artwork.

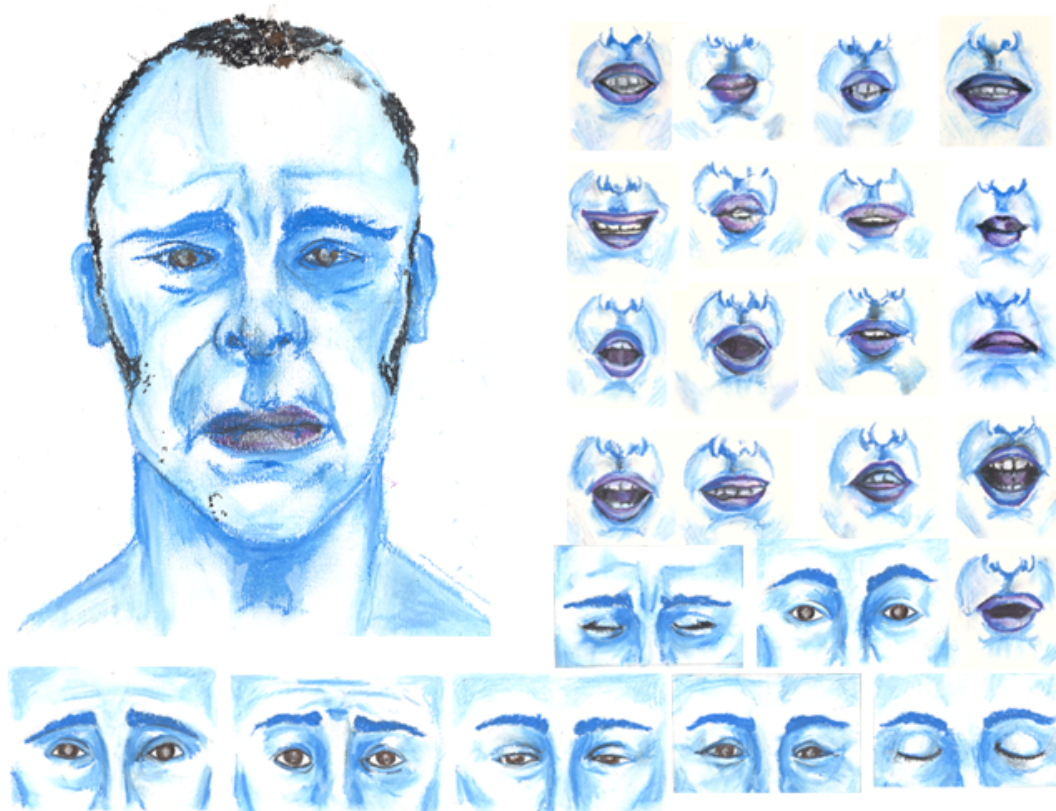


Figure 24: The Blue Guy Artwork.



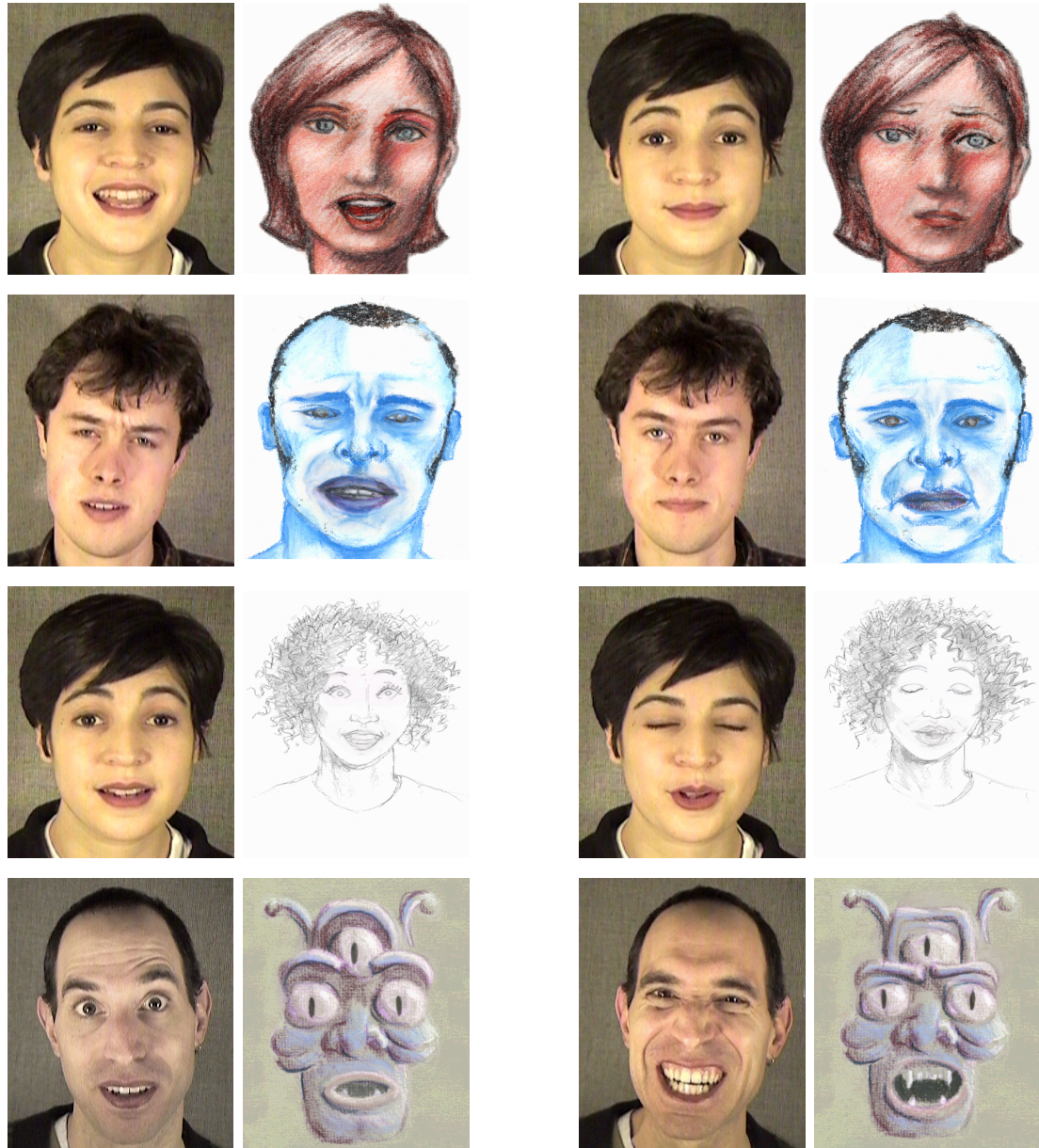


Figure 25: Snapshots of our system with the different artworks. The animated characters were able to output a variety of expressions ranging from happy, sad, pained, neutral, surprised, and aggressive.

# Chapter 7

## Conclusion

### 7.1 Future Work

While this work showed that an animated character can replace an image of a human face for teleconferencing, there is plenty more work that can be done to improve the system. Currently, the system can perform eye and mouth expressions and in-plane rotations of head. While the outputted animation is quite expressive, it still cannot perform all the gestures of a regular human user. Future work includes expanding the expression capabilities of the animated character to include head rotation, and eye gaze. Also work can be done to generate artwork instead of an artist spending time drawing fresh art.

#### 7.1.1 Head Rotation

One of the main limitations of the system is that head shakes and nods cannot be performed by the background head. Emotion and facial gestures are often demonstrated in head motion, such as nodding to show agreement and disagreement. The current system makes the that the user is always looking at the camera to simplify the tracker and the number of artworks required for rendering.

In allowing for head rotation, we need to add functionality to the tracker and the

rendering engine. Work done by Kentro Toyama at Microsoft Research has demonstrated that real-time tracking systems can accurately track all six degrees of freedom of the head. This is done by evaluating the eye position in the head.

The rendering engine needs to be expanded significantly to allow for head rotation. The background face can no longer be just a set of static images but rather it needs to change dynamically with the angle of the head. As with the eyes and mouth, we can use a similar technique of image based rendering to generate the appropriate background. By having a set of background artworks which represent each of the different angles of the head, we can use our same morphing engine to perform morph to generate a background which will output the proper art to convey head orientation. It is unclear what should happen with the eyes and mouth. If the head rotation is restricted, one may be able to use existing eye and mouth renderings. Figure 26 is a preliminary test of a head shake generated with our system to demonstrate that it is possible.

### 7.1.2 Gaze

An additional restraint on the animated expression is that the eye gaze always is directly at the camera. Again, many emotions such as wonder and bushfulness, are expressed through eye gaze. Incorporating independent eye gaze into the system could be done multiple ways. First we could incorporate new art which includes gaze and incorporate that art into expression mapping. The problem is that it may require an excessive amount of art and it is unclear whether morphs on the pupil would be convincing. The other solution would be to add another layer to the face to allow the eyes to roam in the eye socket. This requires no new artwork but may decrease the rendering performance of the system.

### 7.1.3 Generating Artwork

One of the most difficult and time consuming parts of this system is obtaining the artwork to drive the animation. Not only does it require a talented artist but time to import the artworks into the computer. One possibility for improving this could



Figure 26: Head shaking with morphed backgrounds. Using the same morphing engine, the background is morphed to correspond with the angle of head.



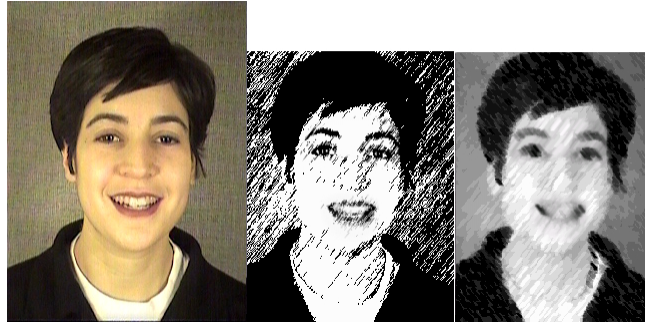


Figure 27: Artwork for the system can come alternative inputs from the input video. The two faces on the right were generated by the rendering system. The input art was created in Adobe PhotoShop by using non-photorealistic image filters on expression frames of the input video. The two images were rendered using input expression art created with a pen-ink and chalk-charcoal image filters.

be to use the existing video stream as a source for the artwork. Instead of having an artist draw a new character from scratch, images of the user with each of the different expressions could be used by the rendering engine as input artworks. Furthermore, standard non-photorealistic image filters could generate images that would appear to have been drawn by an artist, as shown in figure 27. This alleviates the work required to make the hand-drawn art and requires not coding changes to the existing system.

## 7.2 Conclusion

This work clearly demonstrated that a real-time animated teleconferencing solution is viable with today's consumer PCs and graphics hardware. The results showed a form of communication that is engaging, interactive, and most of all, fun. With Moore's law continuing to drive CPU speeds higher and graphics cards increasing in rendering throughput, animated teleconferencing is a technology that is definitely our near future.

As the power of internet communication becomes more available around the globe, communication technology is destined to become the next technological revolution.

Not only is it important that a communication technology be able to transmit expression performed by the user, but it also must be an engaging and enjoyable. Our system offers both to the users. The animated character can add expressive characteristics which expand the human range of expression. But what makes this work a sure success is that it allows people to have fun while communicating with each other.

# Appendix A

## SIGGRAPH Submission

## **Appendix B**

# **Non-Photorealistic Performance-driven Facial Animation**

## Appendix C

### Face Driven Animation

# Appendix D

## Data Files

# Bibliography

- [1] Gary Faign. Facial animation workshop at Industrial Light and Magic, San Rafael, CA, December 1998.
- [2] H. Li, P. Roivainen, and R. Forchheimer. 3-D motion estimation in model-based facial image coding. In *IEEE Computer Graphics and Applications*, pages 58–71, January 1998.
- [3] Don Pearson and John Robinson. Visual communication at very low data rates. *Proceedings of the IEEE*, 73(4):795–811, 1985.
- [4] Seung-Yong Lee, George Wolberg, and Sung Yong Shin. Plymorph: Morphing among multiple images. In *IEEE Computer Graphics and Applications*, pages 58–71, January 1998.
- [5] Verge, M.P., Causse, R., Hirschberg, A., (1995) A Physical Model of Recorder-Like Instruments *ICMC Proceedings, 1995*
- [6] Thaddeus Beier and Shawn Neely. Feature based image metamorphosis. In *SIG-GRAPH 92 Conference Proceedings*, pages 337-343, 1998

# Acknowledgements

Special thanks goes out to my advisor **Adam Finkelstein** and my partner in crime, **Allison Klein**. Also this project would not have been possible had it not been for the talented artist who created such interesting characters: **Rob Jensen '98** and **Robyn Williams '99** both gave up their time to make my thesis really cool. Thanks guys.