Real-time Speech Animation System

by

Jieyun Fu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2011

ARCHIVES
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUN 2 1 2011
LIBRARIES

© Massachusetts Institute of Technology 2011. All rights reserved.

Author .... [illegible] ....
Department of Electrical Engineering and Computer Science
May 20, 2011

Certified by .... [illegible] ....
James Glass
Principle Research Scientist
Thesis Supervisor

Certified by .... [illegible] ....
D. Scott Cyphers
Research Scientist
Thesis Supervisor

Accepted by .... [illegible] ....
Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

Real-time Speech Animation System

by

Jieyun Fu

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2011, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

We optimize the synthesis procedure of a videorealistic speech animation system [7] to achieve real-time speech animation synthesis. A synthesis rate must be high enough for real-time video streaming for speech animation systems to be viable in industry and deployed as applications for user machine interactions and real-time dialogue systems. In this thesis we apply various approaches to develop a parallel system that is capable of synthesizing real-time videos and is adaptable to various distributed computing architectures. After optimizing the synthesis algorithm, we integrate a videorealistic speech animation system, called Mary101, with a speech synthesizer, a speech recognizer, and a RTMP server to implement a web-based text to speech animation system.

Thesis Supervisor: James Glass
Title: Principle Research Scientist

Thesis Supervisor: D. Scott Cyphers
Title: Research Scientist

Acknowledgments

I would like to first thank my research advisors, Dr. Jim Glass and Scott Cyphers, for their guidance and support. This thesis would not have been possible without their encouragement and invaluable suggestions. In particular, I thank them for their patience for helping me to understand the problem and to formulate my thesis clearly and succinctly. I would also like to thank Dr. Lee Hetherington for setting up the speech recognizer which is essential to this thesis.

I wish to thank my fellow graduate student Ian McGraw for his insightful discussions on the integration with other speech applications and enjoyable companionship.

Lastly, I thank my family. I am grateful to my parents, Jun Fu and Bibo Long. I thank them from the bottom of my heart for their unconditional love, guidance, and sacrifices so that I can pursue my dreams. To my love, Katherine Lin, I am blessed to have met you. I am grateful for your friendship over the past few years and look forward to many more wonderful years ahead. To my family, I dedicate this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This research continues and expands upon the work started in the Trainable Videorealistic Speech Animation System (also referred to as Mary101) [7]. Mary101 is capable of generating a speech animation of a human based on the training and analysis of previous video recordings. Specifically, the human is first asked to read a particular paragraph. After training, it is possible to synthesize an animated video of any sentence, as if it had been spoken by the same human. Previous user experience surveys have concluded that this system has great potential for improving the quality of user machine interactions with the videorealistic human serving as an "agent."

In this thesis, we focus on optimizing the video synthesis process to enable the system to generate new video in real time. In other words, the synthesis rate must be faster than the video playing frame rate. After optimizing the synthesis process, we integrate the speech animation system with a speech synthesizer so that we can generate an audio stream from text, and merge the audio with the appropriate facial animation. Finally, we build a simple web interface so that the users can generate and view speech animation from any input text.

1.1 Motivation

Major applications of speech animation systems include user-interfaces for desktop computers, televisions, cellphones, etc. Traditionally, dialogue user interfaces can only use text or images to interact with users. Although more and more devices now have the capability to allow users to interact via speech or a touchpad, there is still very little sensory stimulation during the interaction with the machine interface, aside from the user reading text prompts or looking at static images.

A speech animation system is a valuable addition to the traditional interface. Because a videorealistic human is displayed on the screen instead of audio, text, or static images, users feel like they are speaking to a live agent, rather than interacting with a machine.

However, since the synthesis rate of the original Mary101 system is only about 6 frames per second and the desired video playback rate is about 30 frames per second, we must wait until the synthesis is completed before playing back any hypothetical speech animation clips. This is a major limitation to future applications of the system and to improving the user-machine experience because, in order to make users really feel comfortable speaking with a machine, the system needs to be able to generate responses quickly based on the context of the conversations. Therefore, the optimization of the synthesis algorithm is critical to realizing real-time interaction.

1.2 Thesis Organization

The remainder of the thesis is organized in the following manner.

In chapter 2, we present related work completed in the speech animation area. We compare early development of videorealistic speech animation technology, Video Rewrite and other 3-D Cartoon speech animation systems with Mary101. We also discuss the results of a user experience survey that compares the improved usability of a user interface with the videorealistic speech animation to other simple user interfaces.

In chapter 3, we explain the work flow of Mary101 in detail. First, we discuss how Mary101 analyzes a pre-recorded video to infer motion patterns using a Multidimensional Morphable Model (MMM). We then discuss in detail how MMM is applied to create novel video.

In chapter 4, we present performance optimization techniques. We first present profiling results from the synthesis algorithm of Mary101. We obtain and compare the runtime of each component in the program. Through profiling, we identify the bottlenecks of the original synthesis procedure. We discuss the details of the optimization approaches we take to improve the synthesis rate, including precomputation and parallelization. We list implementation details and the effects of each suggested approach.

In chapter 5, we present how we integrate Mary101 with a speech generator. We also build a web interface for the integrated system. Human users or any third party software can generate speech animation based on any text they provide. This chapter also includes technical details on the usage of FFmpeg libraries, the underlying open source libraries used to process videos.

Finally, in chapter 6, we suggest areas of potential future work and summarize the contributions we have made in this thesis.

Chapter 2

Related Work

In this chapter, we provide an overview of related work completed in the field of speech animation. In the first section, we introduce an early videorealistic speech animation venture, Video Rewrite. In the second section, we discuss and compare a few 3-D cartoon animation technologies. In the third section we briefly introduce the Mary101 system. Finally we present a user experiences survey that compares the improved usability of a user interface equipped with videorealistic speech animation with other "naked" user interfaces.

2.1 Video Rewrite

Bregler, Covell and Slaney [6] describe an image-based facial animation system, the Video Rewrite, which is one of the earliest ventures in speech animation. As the name suggests, Video Rewrite does not generate previously non-existing, novel video by analyzing real video recordings. Instead, it creates new utterances from applying a "cut-and-paste" approach to the pre-generated corpus.

Video Rewrite uses a Hidden Markov Model to label previously recorded video with segments of triphones. A new audiovisual sentence is constructed by concatenating the appropriate triphones sequences from the pre-recorded corpus. The transition among triphone sequences is achieved by using the middle phoneme as a "pivot" phoneme and allowing the outer phonemes to overlap and cross-fade each other.

When new video is created using Video Rewrite, the triphones in the video must be properly aligned to the speech transcription. In this last step, Video Rewrite uses a mask to determine the transformation of mouth movement in different sequences and correct the different head positions of the background face.

Video Rewrite generates relatively good quality videos and the techniques it uses for correcting head positions are helpful for later research. However, since Video Rewrite does not generate new video from analyzing the underlying patterns, to be feasible for deployment, it requires recording vast amounts of real video. This imposes a huge burden on the human subject being recorded as well as on the infrastructure used to store the videos. Furthermore, since the system relies on the triphones before or after the current triphone to implement transitions adapting the system to any real-time streaming applications is difficult.

2.2 3-D Cartoon Animation

There have been several attempts to create a 3-D cartoon speech animation system. These systems are quite different from Video Rewrite and Mary101 since cartoon animation is not videorealistic. However, the idea of combining various facial motions for different speech utterances is transferable. This section discusses three different speech animation technologies: Baldi (a 3-D cartoon animation system that does not involve training of previously recorded video, but instead maps an image of a person onto a pre-defined computer graphical model), Hofer's research (which predicts lip motions by tracking the four points around the mouth) and Wang's research (which creates a 3-D cartoon avatar by training a Hidden Markov Model).

Baldi has been successfully deployed in classrooms for profoundly deaf children and has been very influential in the speech animation field. Primarily developed by Massaro, Baldi is an avatar system that can engage users in simple structured dialogue. In the Baldi system, any image from a person can be mapped onto a 3-D polygon surface to make Baldi into a familiar face with animation controlled by a wireframe model. The system can change parameters to move the vertices on the face by geometric operations. Simple emotions can also be displayed in Baldi (e.g. sad, angry, surprised, etc.). Baldi can either create speech or use pre-recorded speech from a real human. The Baldi speech animation system does not have a "training" process that infers the individual's mouth/head motion pattern nor does it try to achieve photorealistic video. However, the successful application of a speech animation system in a classroom setting for deaf children is encouraging.

Hofer, Yamagishi, and Shimodaira describe a speech driven 3-D cartoon animation system in which lip motions on a human avatar are created from a speech signal [19]. Hofer's system tracks four points around the lips and predicts lip motions from a speech signal by generating animation trajectories using a Trajectory Hidden Markov Model [13]. Compared with a regular Hidden Markov Model, which provides very disjoint actions, the Trajectory Hidden Markov Model provides much more visually satisfying results. The major limitation in Hofer's research is that in the current implementation, only four points around the lips were tracked, leading to impoverished models. However, Hofer points out that that more than four points can be tracked easily, which can result in more types of lip motion and overall better video quality.

Research conducted at Microsoft Research Asia by Wang has some similarities to the research presented in this thesis. Wang presents a real-time text to audio-visual speech synthesis system [12]. Wang uses an HMM approach to create a speech synthesizer [17]. The HMM models for the synthesizer are pre-trained for spectrum, pitch and duration. To give the cartoon avatar natural head movements while speaking, an HMM trained from a female announcer is used to generate head movements of the cartoon avatar, correlated with the speech prosody features. There are two main differences between Wang's work and our work: first, Wang focuses on a 3-D cartoon avatar while we create a videorealistic avatar. Second, Wang concentrates on head movement while we focus on lip motion.

2.3 Trainable Videorealistic Animation (Mary101)

Mary101, developed by Ezzat [7], is the main system we expand and improve. It uses Multidimensional Morphable Models (MMM) [5, 10, 11] to model facial expression. Mary101 first analyzes real recordings to obtain a set of prototype images, which MMM can use to synthesize new video frame by frame. We give detailed technical background and implementation in Chapter 3.

There are a few differences that distinguish Mary101 from Video Rewrite and other 3-D animation systems. Comparing to Video Rewrite, Mary101 synthesizes output video from the pattern, recognition and analysis instead of cutting and pasting video segments from stored recordings. Mary101 compared with 3-D cartoon animation techniques, outputs realistic video rather than cartoons.

2.4 User Interface Survey

Pueblo [14] conducted thorough research on the applications of speech animation systems on user interfaces. Two systems that adapt Mary101 are used in Pueblo's research, Mary101.NET and CGI.Mary. Mary101.NET plays on the website using pre-generated video while CGI.Mary only plays locally using its own player. In Mary101.NET, we have to pre-record all the videos that will be played to the users. This is not very expandable to a more comprehensive system in which more interactions between users and machines are needed. CGI.Mary achieves semi-realtime playing by using its own player, which inserts dummy head movements when video synthesis lags behind playback rates. This approach results in a decrease in the videorealistic and smooth quality desired in a speech animation system. Furthermore, reliance on its own player prevents us from embedding the CGI.Mary with web-based applications.

Pueblo's study is conducted with a flight reservation system. Users test and rate three scenarios. The baseline scenario displays text and plays the corresponding audio. The cartoon scenario presents the same setup as the baseline scenario, but adds a static cartoon image of a human as the text is read. The video scenario augments the baseline by adding a speaking avatar generated by Mary101.

In the study, users were asked to judge whether an interface is natural. Based on the collected feedback, Pueblo found that users experience higher comprehension in the video scenario than in the cartoon scenario, despite the fact that the text being communicated is exactly the same. Furthermore, the video scenario also received significantly higher ratings than the cartoon scenario – 14 out of 40 responses described the video scenario as natural, while only 1 out of 40 found the cartoon scenario natural and 8 out of 40 responses think that the baseline scenario natural. The positive ratings for video speech animation from this survey are very encouraging for the further improvement of the speech animation system.

Chapter 3

Workflow of Mary101

In this chapter we discuss the details of the workflow of Mary 101, on which the backbone of this thesis relies. In the first section, we discuss the training algorithm. In the second section, we give an overview of the video synthesis algorithm. Finally in the third section, we discuss in detail how we apply the Multidimensional Morphable Model (MMM) to video synthesis. An overview of the workflow is shown in Figure 3-1.

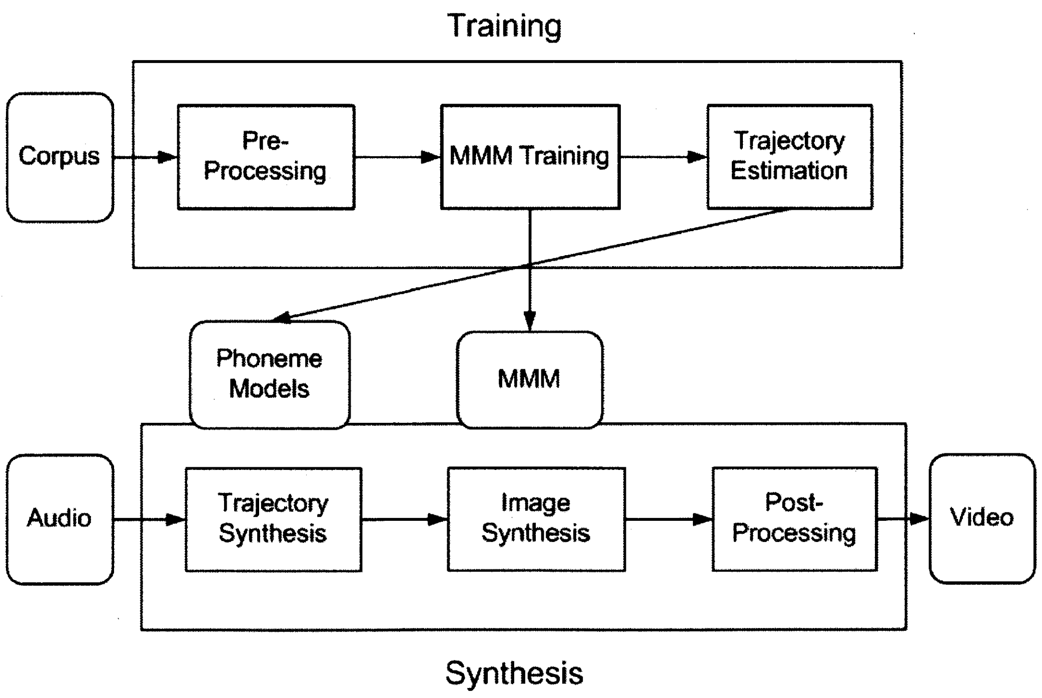

Figure 3-1: An overview of the workflow of Mary101.

3.1 Training

The training stage of Mary101 begins with the pre-processing of the recorded video of pre-designated speech which is converted to a sequence of video frame images. These images are normalized with a mask to remove head movements. Finally, we train a Multidimensional Morphable Model (MMM) space from these image frames.

MMM is a facial modeling technique [5, 10, 11]. The MMM models mouth movements in a multi-dimensional space in which the axes are the mouth shape and mouth appearance. Mouth appearance is represented in MMM by a set of prototype images (in this case 46 prototype images) extracted from the recorded corpus. Mouth shape is represented in the MMM as a set of optical flows [2, 3, 9]. The MMM space is parameterized by shape parameters, α, and appearance parameters, β. The MMM can be viewed as a black box capable of performing two tasks. First, given a set of input parameters (α, β), the MMM is capable of synthesizing an image of the subject's facial expression. Secondly, from a set of prototype images, the MMM can find the corresponding (α, β) parameters of another image.

To construct the MMM, we first find the set of prototype images. For the purposes of efficient processing, principle component analysis (PCA) is performed on all the images of the recorded video corpus. After this analysis, each image in the video corpus can be represented using a set of multi-dimensional parameters. We use an online PCA algorithm, termed EM-PCA, which performs PCA on the images in the corpus without loading them all into memory [15, 16]. After the PCA analysis, we run a K-means clustering algorithm to divide all the PCA parameters into $N = 46$ clusters. The image in the dataset that is closest to the center of each cluster is chosen to be the final image prototype of the cluster. The 46 prototype images we used are shown in Figure 3-2.

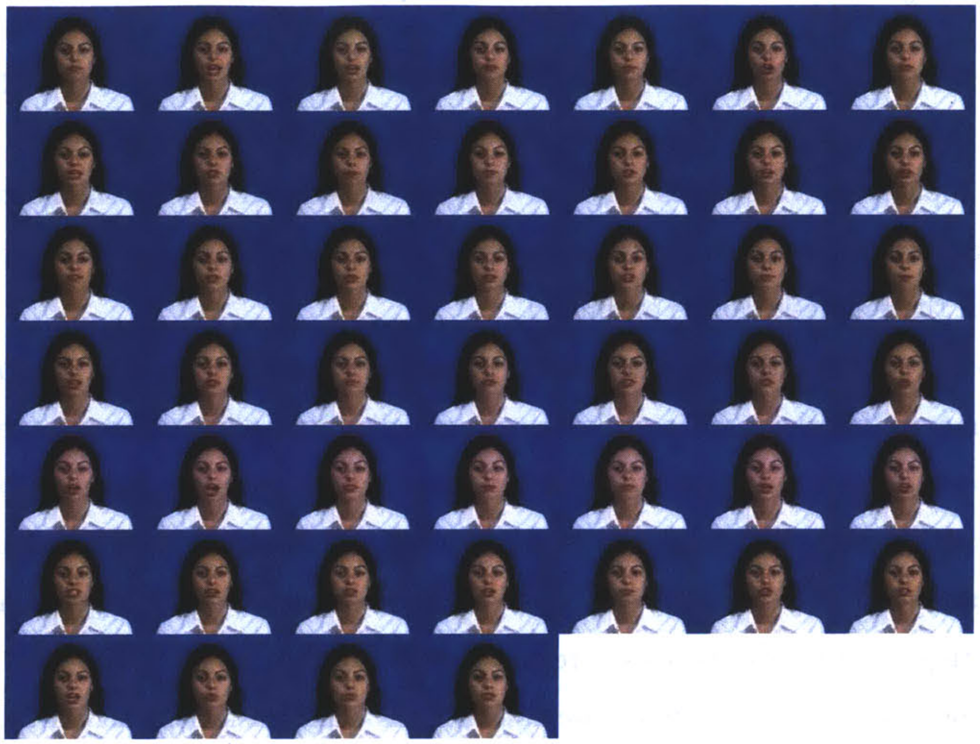

Figure 3-2: 46 prototype images obtained from the training

After obtaining the prototype images, optical flow vectors are computed. First, one of the prototype images is arbitrarily designated as the reference image. In Mary101 this reference image is one in which the person's mouth is slightly open and eyes are open. The optical flow is computed with respect to the reference image and the data is stored into two displacement vectors, one horizontal and one vertical. The optical flow of the reference image is zero because it has no displacement from itself.

After the prototype images and the optical flows are obtained, the MMM is able to estimate (α, β) trajectories for the video stream. We represent each phoneme p as a multidimensional Gaussian with mean μ_p and covariance Σ_p. Each entry in μ_p corresponds to the mean of a coefficient from (α, β) and each entry in Σ_p corresponds to the covariance between the coefficients in (α, β). We analyze and obtain the (α, β) parameters for each frame. Since each frame is associated with a phoneme p, we can estimate a distribution of (μ_p, Σ_p) from all the (α, β), used in the synthesis process.

3.2 Synthesis

The synthesis of new videos can be divided into two steps. First we analyze the audio (which can be from an audio synthesizer or from a human recorded audio) and compute the trajectories (α, β) of the phoneme sequence. We then synthesize the video frames based on the trajectories (α, β) computed from the first step. Both steps of the synthesis can be optimized.

To analyze the audio and compute the phoneme trajectories, we align the stream of phonemes so that the audio input has a corresponding time-aligned phoneme sequence $\{P_t\} = (/a/, /a/, /uh/, /uh/...)$. Each element in the phoneme stream represents one image frame. As mentioned in section 3.1, we represent each phoneme p as a multidimensional Gaussian with mean μ_p and covariance Σ_p. Given μ_p and Σ_p, generating trajectories becomes a regularization problem [8, 18]. The idea is that for a coefficient α_1, for example, we consider the same coefficient across the entire video stream $\alpha_1^{i=1..S}$ and set up a linear equation to measure the smoothness of this coefficient. We find solutions to $\alpha_1^{i=1..S}$ such that the overall smoothness is maximized. The same procedure is applied independently to all the other coefficients.

To generate a video based on the trajectories (α, β), we first synthesize each frame of the video from the estimated (α, β) corresponding to each aligned phoneme. Given a set of (α, β), synthesis is performed by morphing the prototype images to corresponding shape-appearance configurations. Note that the MMM is only used for mouth motion and we still need to integrate the background (including blinking eyes, slight head motion, etc) with the mouth motions to ensure the output video is realistic. In the original Mary101, all these video frames are saved to a file, and an external FFmpeg program is used to glue the video frames together to generate a video file.

Each frame of the video is "relatively independent." Although they are correlated (since when we are finding the coefficients of (α, β) we need to consider all the frames to make the video smooth), the frames are relatively independent because once we know the (α, β) of a frame, the synthesis of this frame is entirely independent of the

(α, β) of the other frames. This relative independence property is critical for the parallel algorithm to be introduced in section 4.2.

3.3 Video Synthesis Using the Multidimensional Morphable Model

In this section, we discuss in details how the MMM is applied to generate novel videos. Video synthesis consists of two steps: trajectories (i.e. α, β coefficients) computation and frame image synthesis. Details of MMM training are not presented since it is not the focus of this thesis.

3.3.1 Computing trajectories $(\alpha,\ \beta)$

We compute each coefficient of the trajectory independently. For each coefficient, (e.g. α_0) Construct a vector y:

$$y = \begin{bmatrix} \alpha_0^{t=0} \\ \alpha_0^{t=1} \\ \dots \\ \alpha_0^{t=T} \end{bmatrix}$$

.

Then synthesizing vector y can be expressed as an optimization problem [7]: finding y to minimize

$$E = (y - \mu)^T D^T \Sigma^{-1} D(y - u) + \lambda y^T (W^T W)^z y \tag{3.1}$$

where

$$\mu = \begin{bmatrix} \mu_{P_1} \\ \mu_{P_2} \\ \cdots \\ \mu_{P_T} \end{bmatrix}$$

,

$$\Sigma = \begin{bmatrix} \Sigma_{P_1} & & \\ & \cdots & \\ & & \Sigma_{P_t} \end{bmatrix}$$

and P_i is the phoneme in the i-th frame. μ_{P_t} and Σ_{P_t} are defined in section 3.2.

The term $(y-\mu)^T D^T \Sigma^{-1} D(y-u)$ is the *target term* used to find the probability that maximizes the joint Gaussian mixture of the coefficients. Matrix D is an $O(T)$ by $O(T)$ diagonal duration weighted matrix which emphasizes the shorter phonemes and de-emphasizes the longer ones. The intuition behind this expression is from the density of joint Gaussion distribution:

$$f(y) = (2\pi)^{-\frac{k}{2}} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(y-\mu)^T \Sigma^{-1}(y-\mu)}$$

We want to find a set of y that maximizes the likelihood. Therefore we want to maximize $-\frac{1}{2}(y-\mu)^T\Sigma^{-1}(y-\mu)$, or equivalently, minimize $\frac{1}{2}(y-\mu)^T\Sigma^{-1}(y-\mu)$. In this application we add a weighting function that emphasizes the shorter phonemes, so we minimize $(y-\mu)^T D^T \Sigma^{-1} D(y-u)$.

$\lambda y^T (W^T W)^z y$ is a *smoothing term*, which aims to minimize the jump between the coefficients. λ is the trade-off between the target term and the smooth term. The constant z is a positive integral order. One has to choose W so that the smoothing term can measure how smooth the trajectories are. In Mary101 we choose W as

$$W = \begin{bmatrix} -I & I & & & \\ \dots & -I & I & \dots & \\ \dots & & \dots & & \\ \dots & & -I & I & \end{bmatrix}$$

By choosing this W, we essentially subtract each entry in y from its neighbor, and add the squares of these differences. If the trajectories are not smooth, the sum of the squares of the differences will be large. To achieve smooth trajectories, we need to minimize these differences.

By taking the derivative of (3.1) we find that the minimal value is obtained by solving

$$(D^T\Sigma^{-1}D + \lambda(W^TW)^z)y = D^T\Sigma^{-1}D\mu \tag{3.2}$$

Experiment shows that $\lambda = 1000$ and $t = 4$ give the best results for the trajectory computation.

3.3.2 From Trajectories (α, β) to Images

After we obtain trajectories (α, β), we can synthesize new frame images from the trajectories. We first proceed by synthesizing a new optical vector C^{synth} by combining the prototype flow

$$C_1^{synth} = \sum_{i=1}^{N} \alpha_i C_i$$

Forward warping (see Appendix A.1) may be understood as pushing the pixels of the reference image along the optical flow vector C_1^{synth} to form a new image. We denote this operation by $W(I, C)$ that operates on an image I with correspondence map C. Generally, there is a hole filling step after warping, since optical flows may not cover the whole set of pixels in the new image (See Appendix A.2).

However, since we only have the optical flow from the reference prototype image to the rest of the images, a single forward warp will not take into account all the image texture from all the prototype images (see [7]). In order to utilize all the prototype images, we reorient as follows (see [4]) to obtain the other correspondence vectors

$$C_i^{synth} = W(C_1^{synth} - C_i, C_i) \tag{3.3}$$

The third step is to warp the prototype images I_i along the re-oriented flow C_i^{synth} to generate a set of N warped image textures I_i^{warped}

$$I_i^{warped} = W(I_i, C_i^{synth}) \tag{3.4}$$

Finally, we blend all the warped images with the β coefficients:

$$I^{morph} = \sum_{i=1}^{N} \beta_i I_i^{warp} \tag{3.5}$$

Combining all the above equations, we obtain

$$I^{morph} = \sum_{i=1}^{N} \beta_i W(I_i, W(\sum_{j=1}^{N} \alpha_j C_j - C_i, C_i))$$

In practice, we also need a mask to fix the position of I^{morph} so that head motion is normalized. In the training stage, we estimated (α, β) from a recorded image, using the inverse of the above steps.

Chapter 4

Optimization

In section 4.1, we present the runtime profiling results to identify bottlenecks in the original implementation of Mary101. In section 4.2, we present a parallelization algorithm used to improve the synthesis procedure. In section 4.3, we introduce another optimization technique, precomputation. Finally, in section 4.4, we discuss how to optimize the trajectory computation procedure.

4.1 Runtime Profiling

The synthesis of new video can be roughly divided into three modules: Frame Image Synthesis, Background Integration and Color Space Optimization. This is a rough division and the modules are not executed in sequence. In order to quantify how computational resources are used in Mary101, we use our system to synthesize a video with 615 frames and obtain the running time of each module. The time consumed by each module of the system is presented in Table 4.1.

Table 4.1: Major Time Consumption (in seconds)

Modules	Operations	Time	Total time
Frame Image Synthesis	Image Warping	7.28	51.8
	Hole Filling	16.99	
	Flow Subtraction	1.88	
	Flow Reorientation	5.96	
	Flow Hole Filling	7.04	
	Morphing Images	12.66	
Background Integration	Fetching Frames	23.2	27.8
	Combining Textures	4.6	
Miscellaneous	RGB to YUV	1.52	10.9
	YUV to RGB	1.47	
	Writing Buffered Files to Disk	7.90	

The frame image synthesis module includes many computer graphics techniques. Image Warping and Hole Filling corresponds to the operations in Eq. (3.4). Flow Subtraction, Flow Reorientation and Flow Hole Filling are from Eq. (3.3). Morphing images are from Eq. (3.5). The details of each operation are in the appendix. We can see that the frame image synthesis part of the program consumes the bulk of processing.

We also spend much time fetching the background images because background images are stored in the harddrive and frequent reading from harddrive is expensive. Finally, we also consume a significant amount of time converting between the YUV color space and the RGB color space and writing buffered files for color conversion purposes.

In the following sections we outline two optimization approaches: parallelization and precomputation.

4.2 Parallelization

4.2.1 A Naive Parallelization Attempt

There are two ways to parallelize computations. The first and most straightforward approach is to parallelize the synthesis of each frame. Operations such as warping, hole filling, etc are highly parallelizable because the operations on each pixel are independent (see Appendix A.1, A.2). For example, one can divide an image or a optical flow vector into four parts and use each of the four threads concurrently to work on each part of the matrix, as illustrated in Figure 4-1

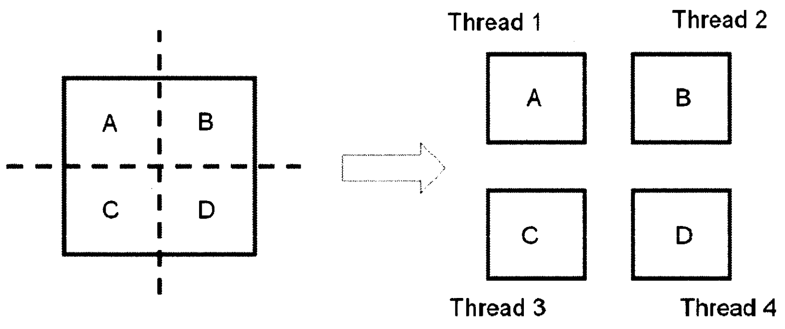

Figure 4-1: Dividing an operation on an image into four sub-images

We implement this approach with four threads on a quad-core machine. We profile the runtime of the program and Table 4.2 shows that this approach is not successful.

Table 4.2: Time Consumption with Naive Parallelization

Module	Operations	Before	After	Total Time Saved
Frame Images Synthesis	Images Warping	7.28	5.01	5.27 (10.17%)
	Hole Filling	16.99	N/A	
	Flow Subtraction	1.88	N/A	
	Flow Reorientation	5.96	2.96	
	Flow Hole filling	7.04	N/A	
	Morphing Images	12.66	6.33	

Since this naive parallelization only applies to synthesizing the frame images, we analyze the time savings only from this module. In Table 4.2, N/A means this parallelization approach did not speed up that operation and we would be better off continuing to use the serialized version. As we can see, only 10.17% of the time was saved in the module, despite running the parallelized program with four times the processors compared to the serial version.

The naive parallelization does not improve the synthesis rate by much for two reasons. Parallelizing each frame synthesis operation results in expensive thread synchronization overhead, especially as many short-lived threads are created. For example, if we are warping on an image using four threads, all threads have to wait idly until the longest thread finishes, as shown in Figure 4-2. This idle waiting is costly, because there are many thread synchronization requirements in this naive parallelization approach, and we cannot predict the work load of each thread to guarantee that the threads will start and stop at the same time.

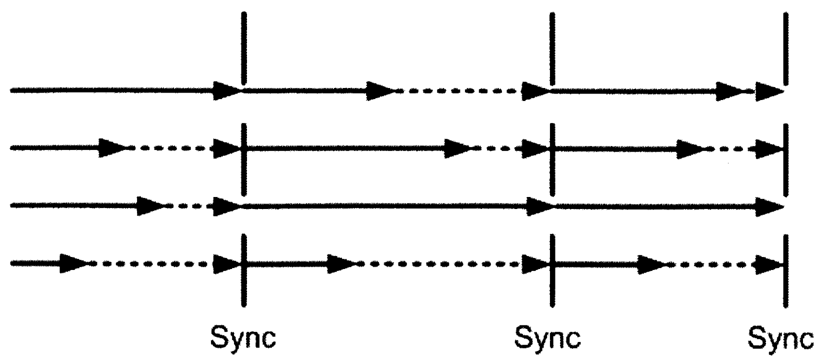

Figure 4-2: Thread synchronization results in idling. Dotted arrows represent wasted resources.

4.2.2 Another Parallelization Attempt

In this thesis, we adopt a more complex but efficient parallelization approach. We define three states for each frame of the novel video: *unsynthesized*, *synthesized* and *encoded*. Unsynthesized frames are the frames that have the trajectories (α, β) calculated but have not yet been synthesized. All frames are initially unsynthesized. Synthesized frames are the frames whose images have been produced but have not yet been added to the movie. Encoded frames are the frames that have been added into the novel movie. Frames do not have to be synthesized in order because of the relative independency mentioned in section 3.2, but they have to be encoded in order because of the video format requirement.

Each worker also has two states: *synthesizing* and *encoding*. The worker is in synthesizing state when it is synthesizing a new frame, and the worker is in encoding state when it is adding synthesized frames to the movie. At anytime, at most most one worker can be in the encoding state.

We maintain two arrays and a global pointer. Array A is initialized to contain the set of all *unsynthesized* frames in order. Array B, initially empty, acts as a buffer in which frames are stored after being synthesized and removed from Array A. Frames are removed from Array B, in order, as they are encoded into the video. Pointer P keeps track of the frame to be encoded from Array B. These data structures allow workers to efficiently parallelize the synthesis and encoding process, where a worker can be a different thread on the same machine or a process on a different machine.

At the beginning of the program execution, all workers are in the *synthesizing* state. Each worker fetches an *unsynthesized* frame in A and synthesize it. One worker cannot fetch the same frame that another worker is already synthesizing. After the synthesis of a frame is finished, the worker places the frame into array B. This frame's state is flagged as *synthesized.* In practice, the process of fetching a frame to be synthesized is implemented by a mutex on Array A (See Figure 4-4).

After a worker places a frame into array B, it checks if any other worker is in the *encoding* state. If so, this worker will remain in the synthesizing state and fetch another unsynthesized frame. Otherwise, this worker checks if pointer P points to the frame it just synthesized. If so, the worker enters the encoding state. When the worker is in the encoding state, it adds the frame pointed to by pointer P into the movie, changes the state of the frame to *encoded*, and move P forwards, until P points to a frame that has not been synthesized. Then the worker will change to the synthesizing state. In the actual implementation, we protect pointer P with a mutex, which prevents more that one threads from accessing P. When a worker is in the *encoding* state, it locks the mutex. When the worker finishes synthesizing a frame, it checks if the mutex is locked. If not, it continues to synthesize other frames. If yes, it checks if the pointer P points to the next frame to be encoded (See Figure 4-4).

We use one short example to demonstrate this parallel algorithm. Assume there are two workers in the system.

1. Initially, worker 1 is synthesizing frame 1 and worker 2 is synthesizing frame 2. Both workers are in the *synthesizing* state. Array B is empty and P points to frame 1.

2. Suppose worker 1 finishes earlier. Since P points to frame 1, worker 1 enters the *encoding* states and starts to encode frame 1 to the video, while frame 2 is being synthesized. Pointer P is locked.

3. Worker 2 finishes its frame while worker 1 is encoding frame 1 to the video. Since worker 1 is in the *encoding* state, worker 2 moves to synthesize frame 3.

4. Since now frame 2 is *synthesized*, after encoding frame 1, worker 1 continues to encode frame 2. Frames will be marked as *encoded* after they are added to the movie.

5. Assuming worker 2 is still synthesizing frame 3 after frames 1 and 2 are encoded, since worker 2 "claimed" frame 3, worker 1 will change to *synthesizing* state and synthesize frame 4.

6. Continue the above procedure until all the frames are *encoded*.

In Figure 4-3, we show a global view of the parallelization structure.

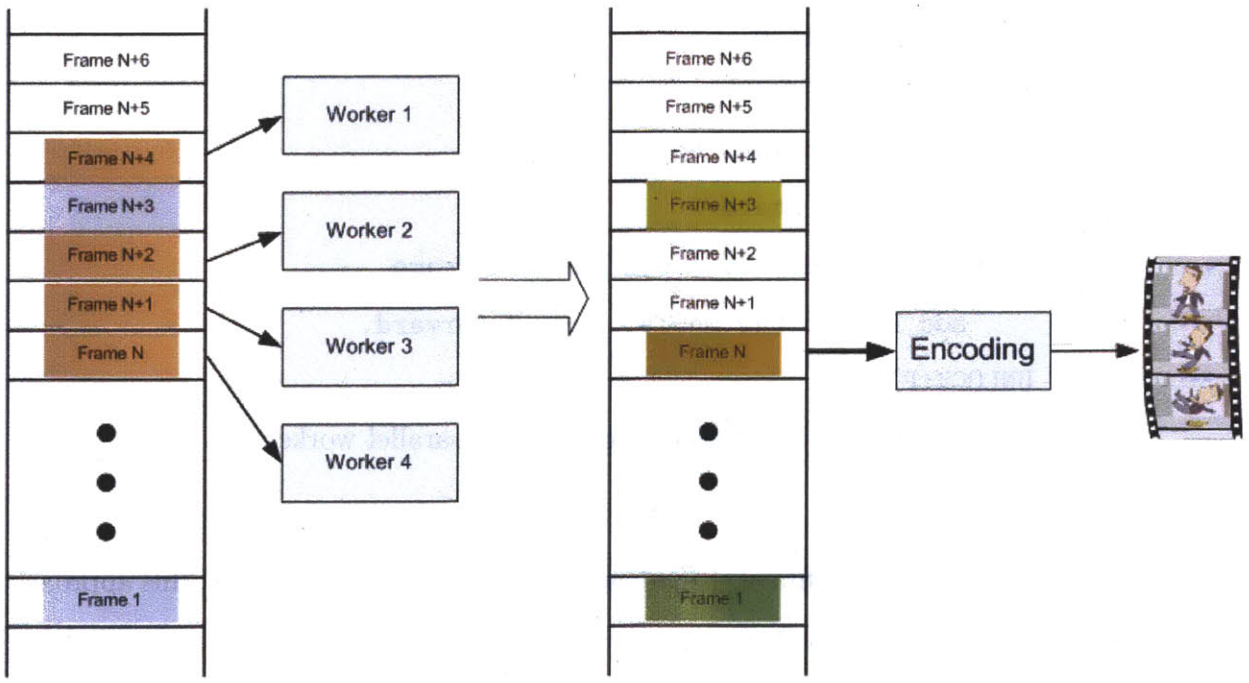

Figure 4-3: A global view of the parallelization structure.

On the left in array A, blue frames are synthesized, orange frames are being synthesized by the workers and white frames are unsynthesized. On the right in array B, the green frames has been encoded, orange frame is the next frame to be encoded and the yellow frame is synthesized but not yet encoded.

In Figure 4-4, we show the implementation of the worker.

```
Construct a list A of ``unsynthesized frames'', initialized with all the frames
Construct a list B of ``synthesized frames'', initially empty
Keep a pointer P that points to the next frame to be added to the movie

FUNCTION worker:
  while not all frames are encoded:
      // synthesizing state
      LOCK(A)
        grab the earliest unsynthesized frame F from A and synthesize F
      UNLOCK(A)
      put F into list B
      TRYLOCK(P)
      If pointer P points to F
        // encoding state
        while P points to a synthesized frame
          add F into the movie, move P forward.
      UNLOCK(P)
```

Figure 4-4: Pseudocode of each parallel worker

There are several benefits of this advanced parallel algorithm. First, this approach is very scalable; we can easily expand to accommodate different architecture – either using more threads, or running on different computers. Second, this algorithm is very effective. We avoid the synchronization overhead of frequently starting new threads. Instead, once a worker begins, it will be kept busy until the entire video is synthesized and encoded. Finally, this algorithm minimizes the probability of lock starvation. The only time a worker would wait for a lock release was when it fetches the next unsynthesized frame. We implement this algorithm on Dell R415 Rack Server with AMD 4184 dual 6-core processors (12 cores in total). The synthesis rate obtained is presented in Figure 4-5.

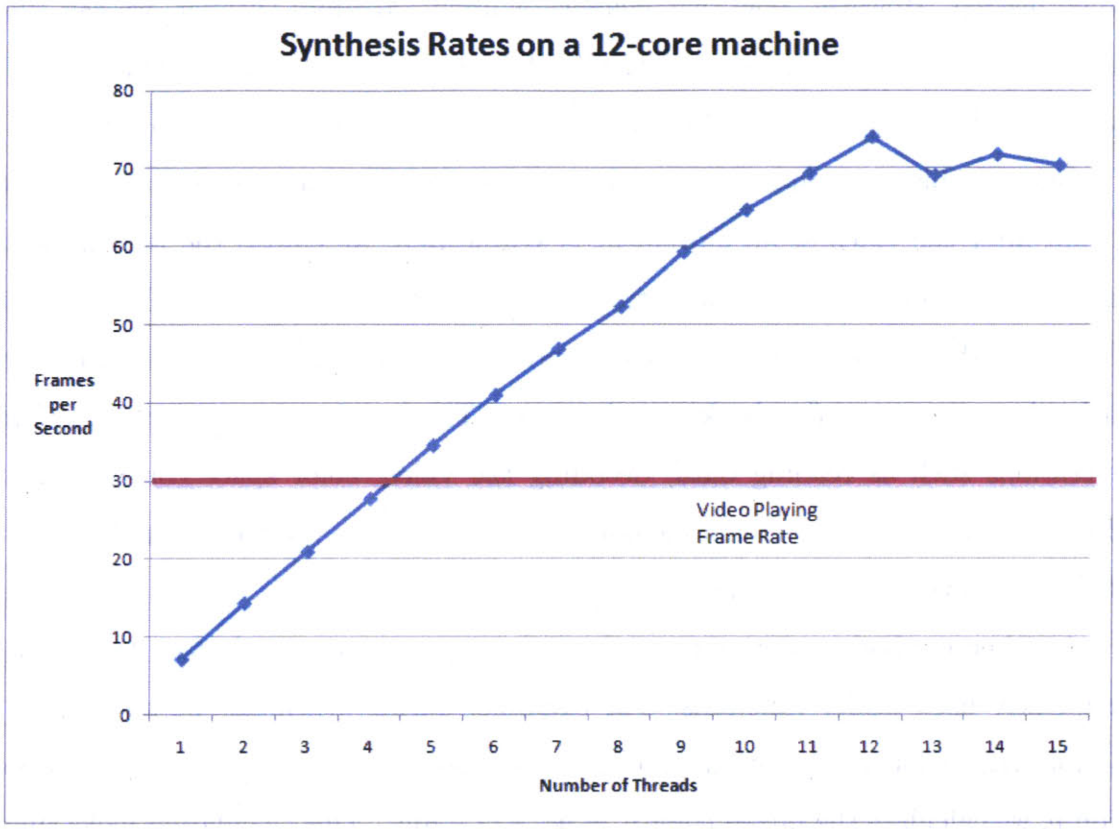

Figure 4-5: The synthesis rates with respect to number of threads, implemented on a 12-core machine

From Figure 4-5, we see that the synthesis rate grows proportionally with the number of threads used until the number of threads exceeds the number of cores. We can see that the optimal number of threads is equal to the number of cores we have; using twelve threads can reach a synthesis rate of 74 frames per second. The threshold required is five threads, at which the synthesis rate exceeds video playback frame rate, allowing us to stream the novel video in real-time.

4.3 Precomputation

The idea behind precomputation is to trade space for speed. When Mary101 was first developed in 2002, memory was much more expensive so we could not store much information in the memory. We had to either store the data on disk or recalculate when needed. However, now that memory is about 20 times cheaper, we can speed up the synthesis program by storing all relevant information in memory instead of writing and reading from the disk or repeatedly computing the same information.

One possible precomputation involves the background mixture. Recall in chapter 3 that after the video of lip movement is synthesized, we embed the mouth motion into a background video and add natural noise to make the output video look realistic. Currently, the background images are stored in a file and we have to read frames from disk whenever we need a background image. This involves many time consuming operations such as reading data, decoding the video, color space conversion, etc. However, since we have enough memory, we can just store the background images in RAM and save the access time.

Another method to optimize synthesis is to precompute and store the value of frequently used utility functions. For example, during synthesis, we often have to perform the color interpolation operation: for colors c_1, c_2 and an interpolation position p, we calculate the mixed color $c' = pc_1 + (1-p)c_2$. This calculation looks simple enough, however it consumes much time on the floating point operation. Precomputation can optimize it. Because the RGB color components range from 0 to 255, we can store a three dimensional table of size 256 by 256 by 100 (the interpolation position can be rounded to two decimal digits without noticeably affecting the image quality, i.e. position takes values of 0.01, 0.02, 0.03 etc). We can save much computing effort by populating the interpolation table with precomputed function values and looking up values from the tables as needed.

Precomputation speeds up the synthesis runtime significantly. Without parallelization, using the above precomputation techniques alone can speed up the synthesis rate by about 33%.

4.4 Optimization of the Trajectory Computation

With the approaches described in the previous two sections, we are already capable of synthesizing the video at the rate faster than the playing rate. However, as mentioned in section 3.3.1, computing the trajectories (α, β) is a prerequisite to synthesizing the video. Therefore, we also need to optimize the trajectory computation.

To find the best trajectory is equivalent to solving the equation (see 3.3.1)

$$(D^T\Sigma^{-1}D + \lambda(W^TW)^z)y = D^T\Sigma^{-1}D\mu \tag{4.1}$$

We note that D, Σ and W are all $O(T)$ by $O(T)$ matrices, where T is the number of frames. Since multiplying or raising a matrix to a constant power has a complexity of $O(d^3)$, where d is the dimension of the matrix, synthesizing the trajectories grows in the order of $O(T^3)$. This complexity becomes intolerable for longer videos, for example, a popular video format has a frame rate of 29.97 frames/second. A 43-second such video has about 1300 frames and takes about 30 seconds to compute (see Table 4.3). Since trajectory computation must complete fully before video frames synthesis can begin, users must wait at least 30 seconds before starting to watch a 43-second video.

One way to prevent this excessive wait for trajectory computation is to localize it. If we divide a sequence of T frames into M trunks, each with T/M frames, then the complexity would be $O(M(T/M)^3) = O(T^3/M^2)$. With this approach, we have to make sure the conjunctions between trunks are smooth. This approach does not intrinsically improve the theoretical complexity, but it does reduce the time usage in practice.

Another approach takes advantage of the special form of the D and W matrices to bypass formal matrix calculations and speed up the computation of (α, β). First, computing $D^T\Sigma^{-1}D$ can be easily avoided, since D and Σ are both diagonal matrices.

In section 3.3.1 we defined the matrix to calculate the smooth-ness as

$$W = \begin{bmatrix} -I & I & & \\ \dots & -I & I & \dots \\ \dots & & \dots & \\ \dots & & -I & I \end{bmatrix}$$

Since all entries of W are either on or next to the diagonal, entries of $(W^TW)^z$ will also locate around the diagonal. We can simply compute the product manually and hard-code the matrix into the program. After this optimization, the complexity is reduced to $O(T^2)$, which is the complexity of solving linear equations.

The runtime of trajectory computation is listed in Table 4.3:

Table 4.3: MMM Trajectory Computation Time Usage

Frames	Video Length	Old Trajectory Computation	New Trajectory Computation	Δ
262	8.75 sec	0.564 sec	0.114 sec	0.450 sec
431	14.38 sec	1.273 sec	0.322 sec	0.951 sec
856	28.56 sec	8.614 sec	1.245 sec	7.370 sec
1302	43.44	29.633 sec	2.860 sec	26.773 sec

From the above table, we can see that a significant amount of time was saved after adopting the improved trajectory computation algorithm. For a half-minute video, we just need slightly over 1 second to synthesize all the trajectories. The last column shows how much time is saved from the optimized trajectory computation algorithm.

Chapter 5

Integration with the Speech Generator and Web Services

In this chapter, we explain in detail how we expand Mary101 to a speech animation system that can generate real-time streaming from the provided text. In section 5.1 we explain how we integrate the speech generator and recognizer with Mary101 to process the user input text. In section 5.2 we discuss how Mary101 was connected to a web service and an RTMP server to stream the video in real-time. Finally in section 5.3 we analyze the latency of the expanded system.

5.1 Integration with the Speech Generator and the Speech Recognizer

Mary101 can generate video based on audio, whether the audio is recorded from a real human subject or from a speech synthesizer. The only requirement is that the audio must be aligned with a phoneme sequence. In order to achieve a text-to-audiovisual speech animator, we need to first generate the audio and the corresponding phoneme sequence.

We use Nuance Vocalizer, a piece of commercial text-to-speech (TTS) software developed by Nuance Communications to synthesize the novel speech, and the TTS engine runs on a server that provides HTTP APIs so that any program can submit a text string and receive a synthesized waveform. This 8KHZ waveform is upsampled to 22.05KHZ fit the FLV output format.

Ideally, after obtaining the waveform, TTS should also be able to return a transcript of the phonemes. However, due to the constraints of current API, we obtain the phonemes from a speech recognizer. We use an XML RPC protocol to an automatic speech recognizer (ASR) perform a forced alignment and obtain a phoneme sequence. The speech recognizer might return an error if a word is not in the dictionary. Error handling is introduced in the next section.

Interaction with the TTS and ASR is implemented in Python, embedded in a C++ program, as Python provides easy interfaces to handle HTTP and XML RPC requests. Table 5.1 shows the time usage of several synthesizing examples. The total time required to obtain the waveform and phonemes is relatively short:

Table 5.1: Time Usage to Obtain Phonemes and Waveforms

Frames	Video Length	Get Waveform and Phonemes
262	8.75 sec	0.23 sec
431	14.38 sec	0.25 sec
856	28.56 sec	0.43 sec
1302	43.44	0.58 sec

We can see from the table that the speech synthesizer and recognizer have very efficient runtime performance. For a half-minute video, less than 0.5 second is taken.

5.2 Web Interface

The facial animation module is connected to a web interface that allows users to stream the video online or download the video. The communication between the web front-end and the facial animation module is implemented by sockets, since this allows for the separation of web server and synthesizing server. The front end of the web service is a CGI script that accepts the string to be synthesized, which is then forwarded via Mary101 to TTS engine and the speech recognizer. Depending on whether the speech recognizer successfully returns a sequence of phonemes, the facial animation module returns a success or failure message.

Should the audio synthesis and recognition succeed, the facial animation module starts to synthesize the video. When synthesizing, it calls an FFMpeg program externally to stream the video to the RTMP server (popular choices are Red 5 or Wowza). An RTMP server is able to forward the video streamed from the facial animation module to the client side player, bit by bit, without modifying or storing the video. Therefore, the client is able to view the video on a real-time basis.

We choose to use JW Player [1] as our client side player. JW Player is a flash-based player that supports JavaScript control. This allows us to customize the look of the player. It also allows the application be embedded in other applications.

When the webpage is loaded, the player plays a background video in which the human subject is silent with some slight head movements. This background video loops infinitely. When streaming the new video, JW Player stops the background video and loads the new video. Since there is a latency of buffering the video, a static preview image will be loaded to make the transitioning between the background video and the novel video smoother. After the novel video is finished, JW Player loads the background video again. In this way, the user always feels that there is a human subject behind the screen.

A screenshot of such an client interface is shown in Figure 5-1 and Figure 5-2:

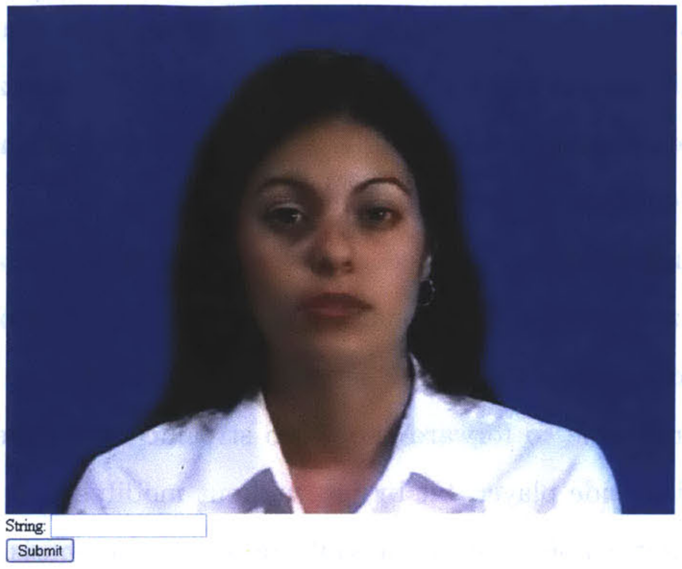

Figure 5-1: Interface snapshot when the client is idle

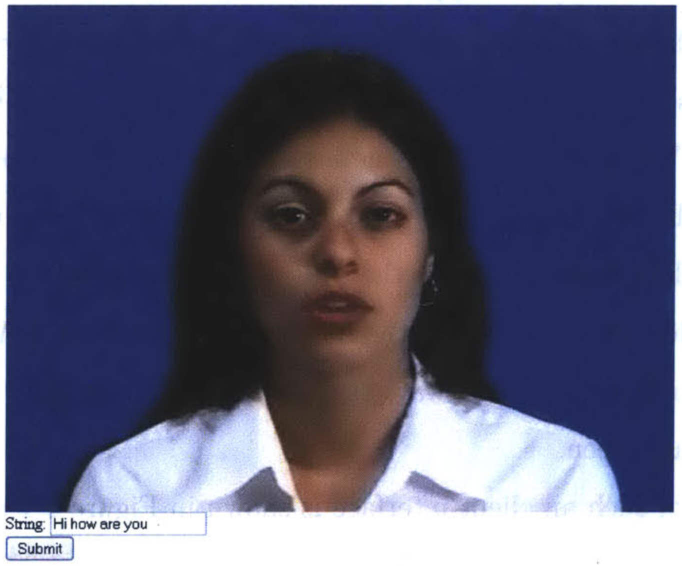

Figure 5-2: Interface snapshot when the client is streaming

Figure 5-3 illustrates the relationship between different components of the facial animation module, after it is integrated with the RTMP server, speech synthesizer and recognizer. The direction of arrows indicates which component initiated the connections. All these components can be arbitrarily combined to run on different machines. This provides flexibility for the future applications of the system, and also allows us to use the computational power of the best multi-core machine exclusively for the novel video generation, which is more comptutationally expensive.

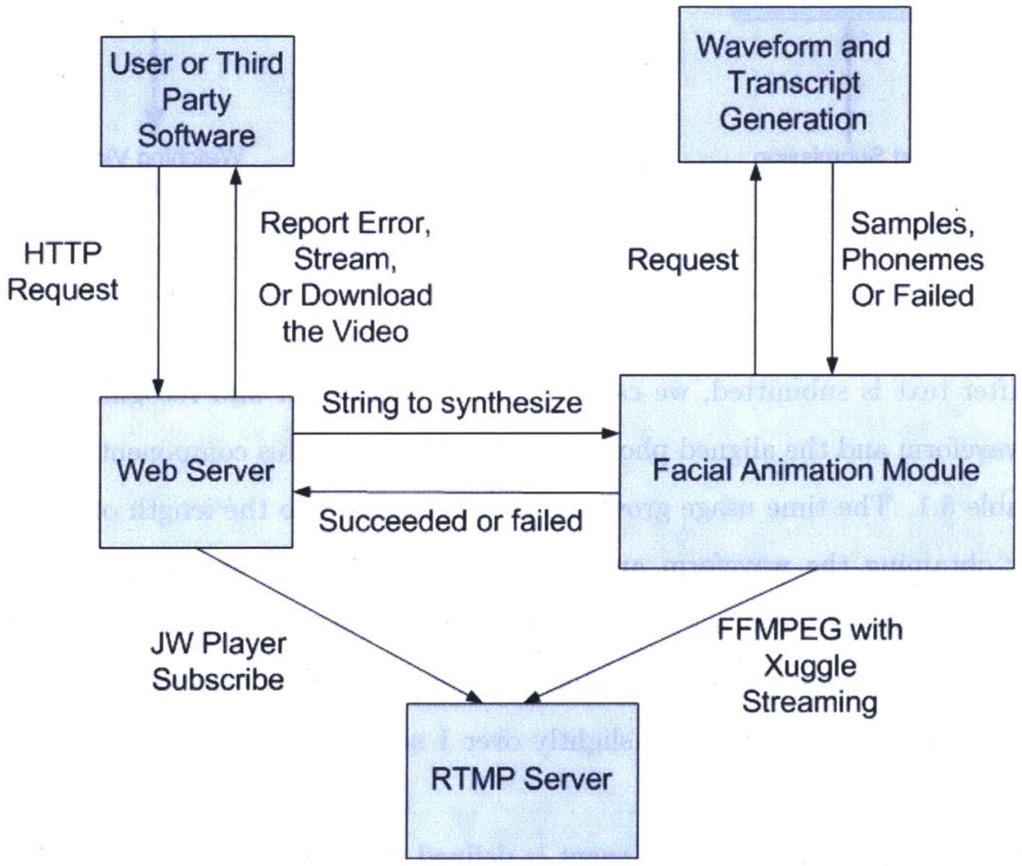
User or Third Party Software
Waveform and Transcript Generation
HTTP Request
Report Error, Stream, Or Download the Video
Request
Samples, Phonemes Or Failed
String to synthesize
Web Server
Facial Animation Module
Succeeded or failed
JW Player Subscribe
FFMPEG with Xuggle Streaming
RTMP Server

Figure 5-3: The interactions between components.

5.3 Latency

We have already achieved real-time speech animation synthesis, elaborated in Chapter 4. In this section we will discuss the latency from submitting a text to streaming the video. There are a few components that make up this latency, shown in Figure 5-4;

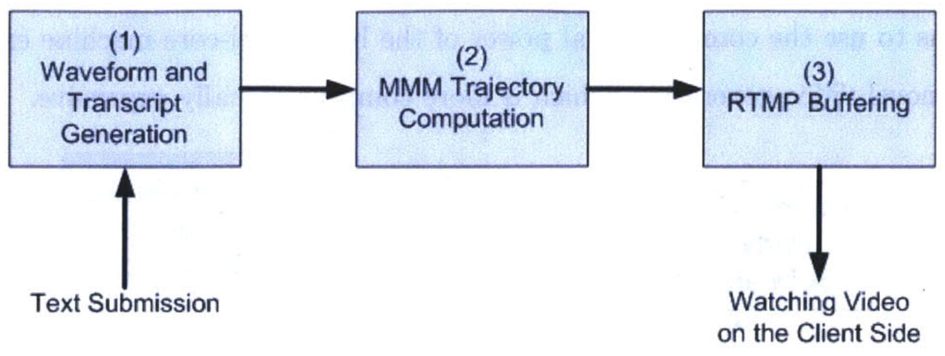
(1) Waveform and Transcript Generation
(2) MMM Trajectory Computation
(3) RTMP Buffering
Text Submission
Watching Video on the Client Side

Figure 5-4: The composition of latency

After text is submitted, we call the speech synthesizer and recognizer to obtain the waveform and the aligned phonemes. The latency of this component is quantified in Table 5.1. The time usage grows slowly with respect to the length of the latency. After obtaining the waveform and the aligned phonemes, we compute the MMM trajectories of the video, as shown in Table 4.3. This part's complexity is $O(T^2)$, where T is the number of frames. However this complexity is acceptable, since for a half-minute video, we just need slightly over 1 second to synthesize the trajectories (see Table 5.2).

The latency in the last component is defined as the time elapsed from the epoch when the facial animation streams the video to the server, to the epoch when the client is able to view the video. Through experimentation, this latency is independent of the length of the video, but only related to the configuration of the RTMP server and client side player. Using JW Player and Red 5 server, the lowest latency we achieve without sacrificing the audio/video quality is about 1.1 seconds.

We quantified the end-to-end latency in Table 5.2:

Table 5.2: End to End Latency

Frames	Length	Labeled Waveform	MMM	RTMP Buffering	Total
262	8.75 sec	0.236 sec	0.114 sec	1.112 sec	1.462 sec
431	14.38 sec	0.251 sec	0.322 sec	1.232 sec	1.804 sec
856	28.56 sec	0.436 sec	1.245 sec	1.154 sec	2.835 sec
1302	43.44	0.581 sec	2.860 sec	1.002 sec	4.443 sec

One can see that when the video is short, the latency is dominated by the RTMP buffering time. When video is long, MMM trajectories computation will be the most dominate component. However, for typical sentences, the latency is under 2 seconds and can make most of the users comfortable.

Both Red5 and Wowza can provide the RTMP streaming capability. Red5 has lower buffering latency, though when streaming a video longer than 15 seconds, the corresponding audio might have some shrill high pitches (this is a confirmed bug by Red5 development community). In Wowza, the latency is usually about 0.5 second longer (i.e., the buffering latency would be about 1.6 seconds rather than 1.1 seconds). However, the quality of the audio streamed through Wowza is much more satisfactory. We implemented two sets of APIs so that future users can switch between Wowza and Red5 conveniently.

Chapter 6

Future Work and Conclusion

6.1 Future Work

6.1.1 User Interface Study

With a synthesis rate that allows real-time streaming, more advanced user interfaces may be developed. In section 2.4 we discussed a user interface study to compare the UI with a speech animation, with audio and with text. However, the speech animation is achieved by first generating a video and then playing back. This limits the possible responses a speech animation system can give, since all the responses a UI can give have to be pre-recorded. UI designers have to anticipate all the possible scenarios and store all the speech animation beforehand. The ability to generate more intelligent responses is very limited.

Now that Mary101 can synthesize video in real-time, machines can develop responses to the user instructions in real-time and this would provide a much more realistic environment. We imagine a live agent can be simulated in such a system, for example, a kiosk or an online banking system.

6.1.2 Including Facial Expression

Another possible improvement is to incorporate more facial expressions. Currently we require that human subjects being recorded have no facial expression (happy, sad, etc.), and we only focus on synthesizing the lips movement. We can consider allowing more facial expressions. There are two challenges with this issue. First, we need to expand our MMM space. Second, we need to implement the dynamic transition between emotional states, i.e, how does one transition from a happy MMM to a sad MMM.

6.1.3 3-D Modeling

One possible future development of the system is 3-D modeling. Currently, we require the human subject to face the camera. Therefore, the synthesized video has only one point of view, which is facing straight towards the camera. If we have a more efficient algorithm to train the dataset and synthesize the video, we can consider incorporating large changes in head pose, changes in light conditions, or changes in viewpoints. It is possible to envision a 3-D scanner that is capable of recording and synthesize a 3D video corpus of speech. We might need to overhaul our trajectory computation algorithm to enable 3-D synthesis.

6.2 Conclusion

This thesis optimizes a speech animation system, Mary101, to achieve real-time streaming. The main approaches we applied are a parallel algorithm and precomputation techniques. We also optimized the MMM trajectory computation procedure so that the waiting period between the submission of the text and the starting of synthesis is reduced. Finally, we integrated Mary101 with a speech recognizer, a speech synthesizer, a RTMP server and a web server to implement a complete system that provides text to real-time speech animation solution. From the results of the past user surveys, we foresee various potential applications of the system.

Appendix A

Graphical Operations

This appendix includes the most frequently used flow operations that are used to synthesize the image frames.

A.1 Forward Warping

Forward warping is very similar to "moving" the pixels of an image along the flow vectors C. In this thesis warping operations were denoated as an operator $W(I, C)$ and produces a warped image I^{warped} as the output. A procedual version of our forward warp is shown below:

```
FUNCTION W(I', I)
 for j= 0 ... height
    for i = 0 ... width
         x = ROUND( i + dx(i,j) )
         y = ROUND( j + dy(i,j) )
         if (x,y) is inside the image
                I' (x,y) = I(x,y)
     return I'(x,y)
```

We also use $W(C_1, C_2)$ to warp around another flow vectors. In this case, x and y components of $C_1 = (d_x(C_1), d_y(C_1))$ are treated as different images, and warped individually. The newly produced warped flow vector would be $(W(dx, C_2), W(dy, C_2))$

The forward warping algorithm can be parallelized. Noticing that we are iterating all the pixels, and for each pixel (i, j) we "push" it to position $(i + dx(i, j), j + dy(i, j))$ of the new image. Since all these pixels are considered as independent in this algorithm, we can use different threads to "push" the pixels in different parts of the image.

A.2 Hole Filling

Warping will inevitably produce a lot of black holes. This is because not every single pixel (x, y) has a (i, j) such that `x = ROUND( i + dx(i,j)` and `y = ROUND( j + dy(i,j) )`. The hole filling algorithm fills in the holes of these holes by finding the nearest non-black pixels and average them.

```
Hole-Filling($I'$, $I$)
 for j= 0 ... height
   for i = 0 ... width
      if (x,y) is black
         find closest non-black pixels in four directions
$(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$
         $I'(x, y) =$ average of  $I(x_1, y_1), I(x_2, y_2), I(x_3, y_3), I(x_4, y_4)$
  return  $I'$
```

There are many ways to average the non-black pixels. In this problem we use the simple average. Similarly, hole filling algorithm can be parallelized, since for each hole, we looking for the nearest non-black pixels and this procedure can be done independently, we can divide the image into several blocks and each thread takes care of one block.

Bibliography

[1] JW player. `http://www.longtailvideo.com/players/jw-flv-player/`.

[2] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, 1994.

[3] J. Bergen, P. Anandan, K. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *Proceedings of the European Conference on Computer Vision*, pages 237–252, Santa Margherita Ligure, Italy, 1992.

[4] D. Beymer, A. Shashua, and T. Poggio. Example based image analysis and synthesis. Technical report, MIT AI Lab, 1993.

[5] D. Beymer and T.Poggio. Image representations for visual learning. *Science*, 272:1905–1909, 1996.

[6] M. Covell, C. Bregler, and M. Slaney. Video rewrite: Driving visual speech with audio. In *Proc. Interspeech*, Brisbane, Australia, June 1997.

[7] T. Ezzat. *Trainable videorealistic speech animation.* PhD thesis, Massachusetts Institute of Technology, 2002.

[8] F. Girosi, M. Jones, and T. Poggio. Priors, stabilizers and basis functions: From regularization to radial, tensor and additive splines. Technical report, MIT AI Lab, June 1997.

[9] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[10] M. Jones and T. Poggio. Multidimensional morphable models: a framework for representing and matching object classes. In *Proceedings of the International Conference on Computer Vision*, Bombay, India, 1998.

[11] A. Lanitis, C.J. Taylor, and T.F. Cootes. A unified approach to coding and interpreting face images. In *Proceedings of the International Conference on Computer Vision*, Cambridge, MA, 1995.

[12] L. Ma, Y. Qian, Y. Chen, L. Wang, X. Qian, and F. K. Soong. A real-time text to audio-visual speech synthesis system. In *Proc. Interspeech*, Brisbane, Australia, September 2008.

[13] T. Masuko, T.Kobayashi, K. Tokuda, T. Yoshimura, and T. Kitamura. Speech parameter generation algorithms for hmm-based speech synthesis. In *Proc. ICASSP*, page 1315, June 2000.

[14] S. Pueblo. Videorealistic facial animation for speech-based interfaces. Master's thesis, Massachusetts Institute of Technology, 2009.

[15] S. Roweis. *EM algorithms for PCA and SPCA*, volume 10. MIT Press, 1998.

[16] M. E. Tipping and C. M. Bishop. Mixtures of probabilistic principle component analyzers. *Neural Computation*, 11(2):443–482, 1999.

[17] K. Tokuda, H. Zen, J. Yamagishi, T. Masuko, S. Sako, A. W. Black, and T. Nose. The HMM-based speech synthesis system (hts). `http://hts.ics.nitech.com`.

[18] G. Wahba. Splines models for observational data. *Series in Applied Mathematics*, 59, 1990.

[19] J. Yamagishi, G. Hofer, and H. Shimodaira. Speech-driven lip motion generation with a trajectory hmm. In *Proc. Interspeech*, Australia, September 2008.