SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Date: January 16, 2015

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Alex DeBoni
Jesse Harder

ENTITLED

GPU-Accelerated Lip-Tracking Library

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Thesis Advisor

Thesis Reader

Department Chair

GPU-Accelerated Lip-Tracking Library

by

Alex DeBoni
Jesse Harder

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
January 16, 2015

GPU-Accelerated Lip-Tracking Library

Alex DeBoni
Jesse Harder

Department of Computer Engineering
Santa Clara University
January 16, 2015

ABSTRACT

A major part of having correct pronunciation when learning a new language is moving your lips in the correct way. One solution to this is software which will track a student's lip movements and provide feedback. This paper describes how a C++ library will be created to accurately track lips in provided images. This library will attempt to use a CUDA-enabled GPU and will fall back to a CPU implementation if such a GPU is not found. As a result, the lip tracking library will run on Windows, Linux, and OS X, as well as Android devices.

The lip tracking algorithm is a hybrid between several approaches. It will perform edge detection and color segmentation as a preprocessing step before using an Active Shape Model to find the final lip contour. This approach is highly efficient and parallelizable for a GPU.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Background

A major part of learning a new language is to be able to pronounce words correctly. To be able to pronounce words correctly, a student's mouth needs to make proper movements. This is difficult to teach in a classroom setting, especially when a student needs to repeat the same phrase many times. The student can become frustrated with the teacher and not want to continue, especially if he or she feels as though he or she is being viewed as incompetent by his or her peers. However, when using a computer, the student can feel more comfortable trying the same phrase over and over without embarrassment.

Currently, there are desktop and mobile applications to help with a student's pronunciation by listening to him or her speak and providing feedback. If the student does not make the correct mouth shape, however, he or she is far less likely to pronounce the word correctly. Applications that only analyze audio can, at best, only guess what the student might be doing incorrectly with his or her mouth so these applications may not provide the help the student really needs.

1.2 Solution

Our solution is to create a mobile application to help students pronounce words correctly by not only listening to them talk and analyzing the audio, but also by watching their mouths and providing feedback for both the audio and video information. We will create an algorithm to accurately track lip movement. A generic tracking algorithm will not work for lip motion tracking since mouths change shape too much when they move. For this reason the lip tracking algorithm that we intend to implement will be specialized to mouth contours. The data provided from this software will then be combined with audio input to analyze how the student may improve his or her pronunciation. For example, the application can tell the student if he or she held his or her mouth open too long

or didn't open his or her mouth enough for a particular word. In the end, this tool would make it easier and faster for students learning a new language to master correct pronunciation.

Chapter 2

Requirements

We have several requirements that need to be met in order to implement our desired system. These requirements can be divided into two categories. Functional requirements are those which our system must do. Generally, they can be evaluated only as either true or false, depending on whether or not the system actually does or does not do what it should. Nonfunctional requirements are those which describe the manner in which the functional requirements should be achieved. They are evaluated based on a degree of satisfaction. Additionally, our system also has a couple design constraints, which are limits on the design and implementation of the system. The following requirements are to be met for successful implementation of our lip-tracking library.

2.1 Functional

The system will:

- Receive camera images from the user.
- Return a point array of lip edges.

2.2 Nonfunctional

The system will be:

- Efficient - it will run quickly enough for use in a real time system.
- User Friendly - it will have an easy to use interface.
- Reliable/accurate - it will produce correct data.
- Robust - it won't crash. (i.e. It's crash frequency will be within acceptable norms.)
- Reusable - it will be easy to use in other systems.

2.3 Design Constraints

The system must:

- Be able to run on Android devices and desktop computers.
- Use CUDA.

Chapter 3

Conceptual Model

The interface to our lip-tracking library will consist of six functions. Three will be used to obtain a lip contour model as an array of points, which is passed into the function. They will also return a status code indicating success or failure. The array that is passed in must have already been allocated by the user, and the `getNumberOfContourPoints` function returns the size that the array should be. The other two functions will allow the user to get and set a threshold value used in the first three functions. This threshold value will determine the required confidence in the lip-tracking algorithm. The lower the threshold value, the less accurate the algorithm will be. If the value is too high, however, the algorithm may fail to obtain a lip contour at the desired confidence level, in which case the functions will return error codes.

3.1 API Functions

- `int getLipContourPGM(unsigned char image[], unsigned int height, unsigned int width, unsigned int contour[]);`
- `int getLipContourPPM(unsigned char image[], unsigned int height, unsigned int width, unsigned int contour[]);`
- `int getLipContourBMP(unsigned char image[], unsigned int height, unsigned int width, unsigned int contour[]);`
- `int getNumberOfContourPoints();`
- `int setAcceptanceThreshold(unsigned int threshold);`
- `unsigned int getAcceptanceThreshold();`

Chapter 4

Use Cases

Based on our conceptual model, our system has a number of actions that users might perform. Any action that a user might take can be considered a use case, which is defined as the steps needed to achieve a goal on the application. Presented below are the use cases for our library, which effectively correspond to the different functions in the library. Under each use case, the actor, goal, precondition, postcondition, sequence of steps, and exceptions are listed. All the use cases of our application are shown below in Figure 4.1. Following the diagram are detailed descriptions of each use case.

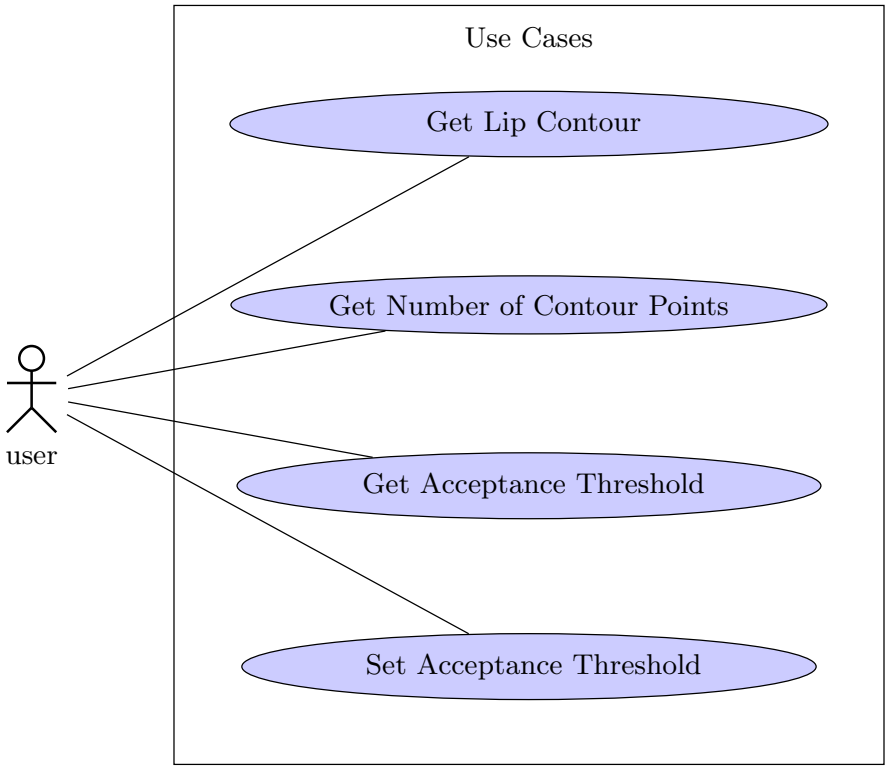

Figure 4.1: Use case diagram

4.1 Use Case 1: Get Lip Contour

Actor: User.

Goal: Obtain lip contour data from a face image.

Preconditions: User has imported our library, has an image to process, and has allocated an array of the proper size.

Postconditions: User has a lip contour for the provided image.

Sequence of Steps: User calls one of the `getLipContour` functions on the facial image.

Exceptions: The image provided is not of a face: the function will return an error code.

4.2 Use Case 2: Get Number of Contour Points

Actor: User.

Goal: Obtain the number of contour points provided by the lip-tracking process.

Preconditions: User has imported our library.

Postconditions: User has the number of contour points.

Sequence of Steps: User calls the `getNumberOfContourPoints` function.

Exceptions: None.

4.3 Use Case 3: Get Acceptance Threshold

Actor: User.

Goal: Obtain a value for the acceptance threshold for the lip tracking process.

Preconditions: User has imported our library.

Postconditions: User has acceptance threshold value.

Sequence of Steps: User calls the `getAcceptanceThreshold` function.

Exceptions: None.

4.4 Use Case 4: Set Acceptance Threshold

Actor: User.

Goal: Change the value for the acceptance threshold for the lip tracking process.

Preconditions: User has imported our library.

Postconditions: Acceptance threshold value is changed to provided value.

Sequence of Steps: User calls the `setAcceptanceThreshold` function.

Exceptions: User provides an invalid value to the function: acceptance threshold value will remain unchanged. Function will return 1 to indicate failure.

Chapter 5

Technologies Used

Below we list all of the technologies that we use in our system and the basic reasons why we chose to use each of them.

5.1 C++

We are using C++ because it is a relatively efficient programming language. Additionally, it allows us to create one library that can be cross-compiled and reused on any platform. Further, both of us have a significant amount of experience with the language.

5.2 CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by Nvidia for their GPUs. We are using CUDA because Android devices are now starting to get dedicated GPU chips that are optimized for CUDA.

5.3 Nvidia Tegra K1

We are using the Nvidia Tegra K1 GPU because it is the currently the most common mobile GPU being marketed, with phones like the Google Nexus 9 utilizing it.

5.4 OpenCV

We are using the OpenCV library to do image processing such as noise filtering and edge detection. OpenCV also detects if there is a CUDA-capable GPU on the system and will use it if it finds one. Additionally, it is compatible with Windows, Linux, OS X, Android, and iOS, so it will work for all of our implementations.

5.5 CMake

CMake is a cross-platform make utility which detects what compilers are installed on the system and selects an appropriate to use for the project. This will make it easy for us to test changes on different platforms by streamlining the compilation process.

Chapter 6

Design Rationale

In this chapter we list and explain the reasoning for the design choices we have made for the lip-tracking library.

6.1 Technologies

A description of the technologies we are using is found in Chapter 5. We chose these technologies because they are very common and well supported, in addition to being available on every platform we are working with (Windows, Linux, OS X, Android).

6.2 Application Programming Interface

We wanted to make our API as simple as possible for the user, while still being flexible. As a result, we allow images to inputted in three different formats: Portable Grey Map (PGM), Portable Pixel Map (PPM), and Bitmap (BMP). These formats allow for both color (BMP and PPM) and greyscale (PGM) images, in formats that are very common.

The user can also change the acceptance threshold for lip detection. If there are too many false positives, then the threshold can be lowered to compensate, and vice versa.

6.3 Algorithm

There are many different algorithms that have been used in the past for feature tracking. These can be divided into three categories, geometric-based approaches, appearance-based approaches, and model-based approaches. Ahmad Hassanat has provided a literature review comparing these approaches in his paper *Visual Speech Recognition* [1].

6.3.1 Geometric-Based Approaches

These approaches involve edge detection and then trying to match ratios of the edge's height, width, area and perimeter with some predefined values.

6.3.2 Appearance-Based Approaches

These approaches involve color segmentation of the image and then trying to match a segment with a predefined set of colors. This is the most common approach currently taken [3][5], but does not always have good results.

6.3.3 Model-Based Approaches

These approaches involve creating a statistical model of a feature shape by inputting a set of hand-landmarked images into a neural network. This model can then be used to very accurately find a similar shape in an image. There are three common model-based approaches, the Active Contour Model (ACM), Active Shape Model (ASM), and Active Appearance Model (AAM). These three models are closely related in that they all use the same fundamental algorithm. However, ASM is an improvement over ACM and AAM is an improvement over ASM. These improvements bring more accuracy to the algorithm, however, they also increase complexity and runtime.

Additionally, both ASM [2] and AAM [4] have been shown to be parallelizable with CUDA.

6.3.4 Our Approach

We have chosen to take a hybrid approach that combines all three previous approaches. The geometric-based and appearance-based approaches alone do not produce results that are accurate enough for us. They run very quickly, however, and can produce a better input image for a model-based approach. The model-based approach we are using is the Active Shape Model. The model is a good balance between the Active Contour Model, which is too simple and not accurate enough, and the Active Appearance Model, which is complicated and relatively slow.

Chapter 7

Architectual Diagram

We are using a data-flow architecture for our system, also known as a pipe-and-filter architecture. We have an array of pixel values for an image which gets modified by each filter it passes through. The entire data flow for our system can be seen in Figure 7.1.

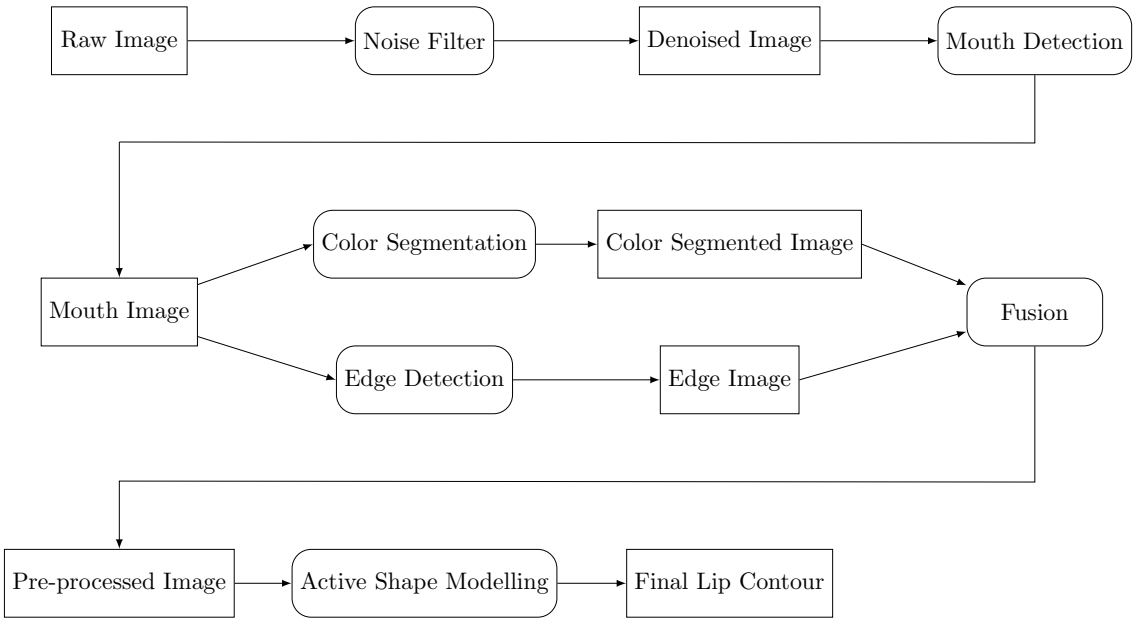

Figure 7.1: System architecture

Chapter 8

Test Plan

It is necessary for us to test our system in order to ensure that our lip-tracking library is fully functional and meets all of the requirements we have identified for it. There are a number of different ways that we plan to test our system, the main points of which are listed below.

8.1 Alpha Testing

We have a library of videos and images of people's lips. We will visually analyze the output of our program for these inputs to see how accurately it tracks the lips.

We will also perform automated unit testing, including the following sets of test.

- Checking a variety of images that either do or do not have lips in them to see whether the library properly identifies the lips or lack thereof in the image.
- Iterating through a range of acceptance threshold values on various images of lips to see when detection starts and stops.

8.2 Beta Testing

We will get a group of people with a diverse set of facial features to test our library with a webcam and visually analyze the results. The variables we will try to test include mustaches, beards, teeth presence, lip color, and skin color.

Chapter 9

Project Risks

Table 9.1 below shows the potential risks we have identified for our project. Listed with each risk are the consequences should the risk occur, the probability of the risk occurring, the severity of the risk, the impact of the risk (calculated by multiplying probability and severity), and the mitigation strategies we employed to avoid the risk from occurring.

Risk	**Consequences**	**Probability**	**Severity**	**Impact**	**Mitigation**
Running out of time	Not having a finished product.	0.4	7	2.8	Follow gantt chart schedule. Prioritize features of system.
GPU transition problems	Loss of time. Potentially only CPU implementation finished.	0.3	8	2.4	Consider GPU implementation while working on CPU version.
Team member becomes sick or otherwise disabled	Loss of time.	0.4	4	1.6	Stay informed on each other's progress.Soft deadlines.
Major issues with chosen technologies	Loss of time and progress.	0.2	7	1.4	Early testing of technologies. Consider alternative technologies in advance.
Failure to gather sufficient beta testers	Incomplete testing.	0.5	2	1.0	Scout beta testers in advance. Have thorough alpha testing plan.

Table 9.1: Risk analysis table

Chapter 10

Development Timeline

The figures below show a timeline of when we will be working on and finishing each major task of this project, including the writing of the library, the documentation, and preparation for presentations. By following this plan, we can keep ourselves on schedule for each deadline.

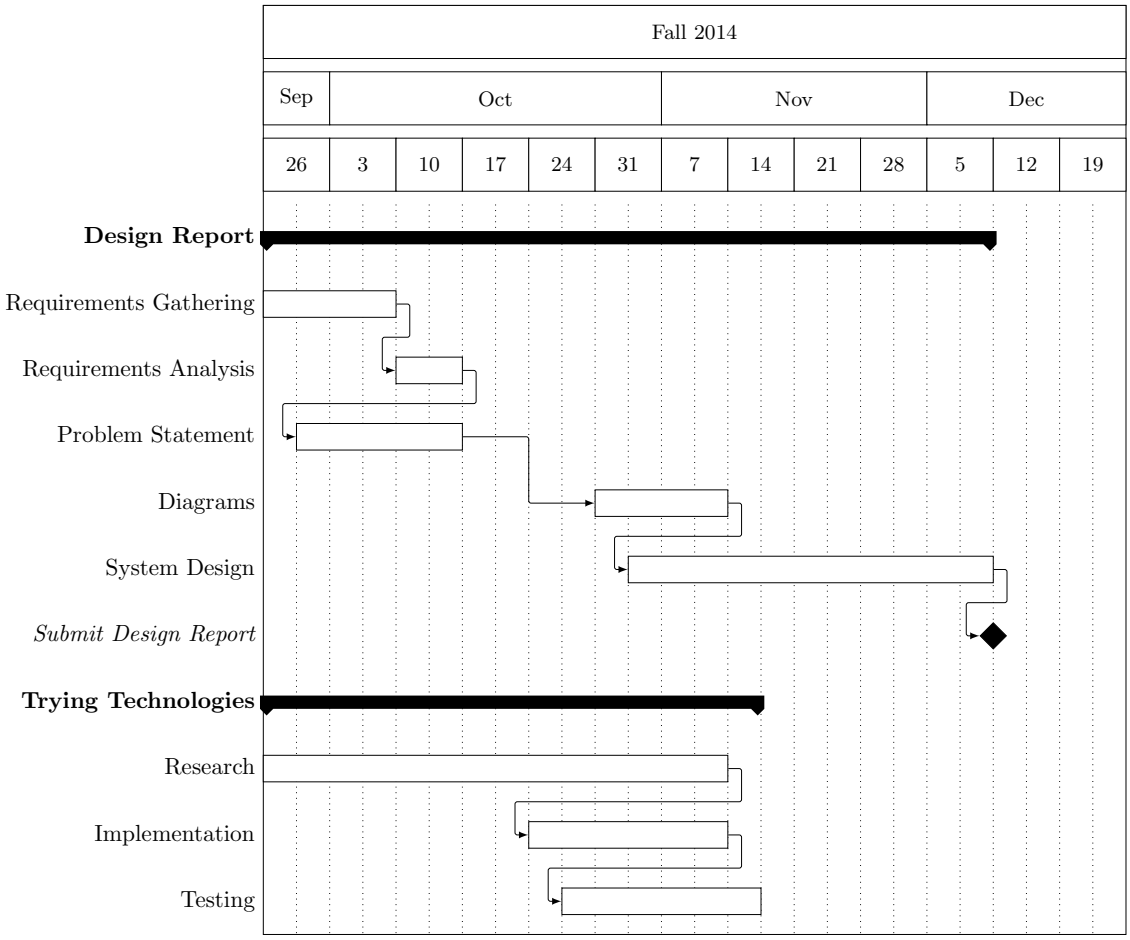

Figure 10.1: Fall quarter gantt chart

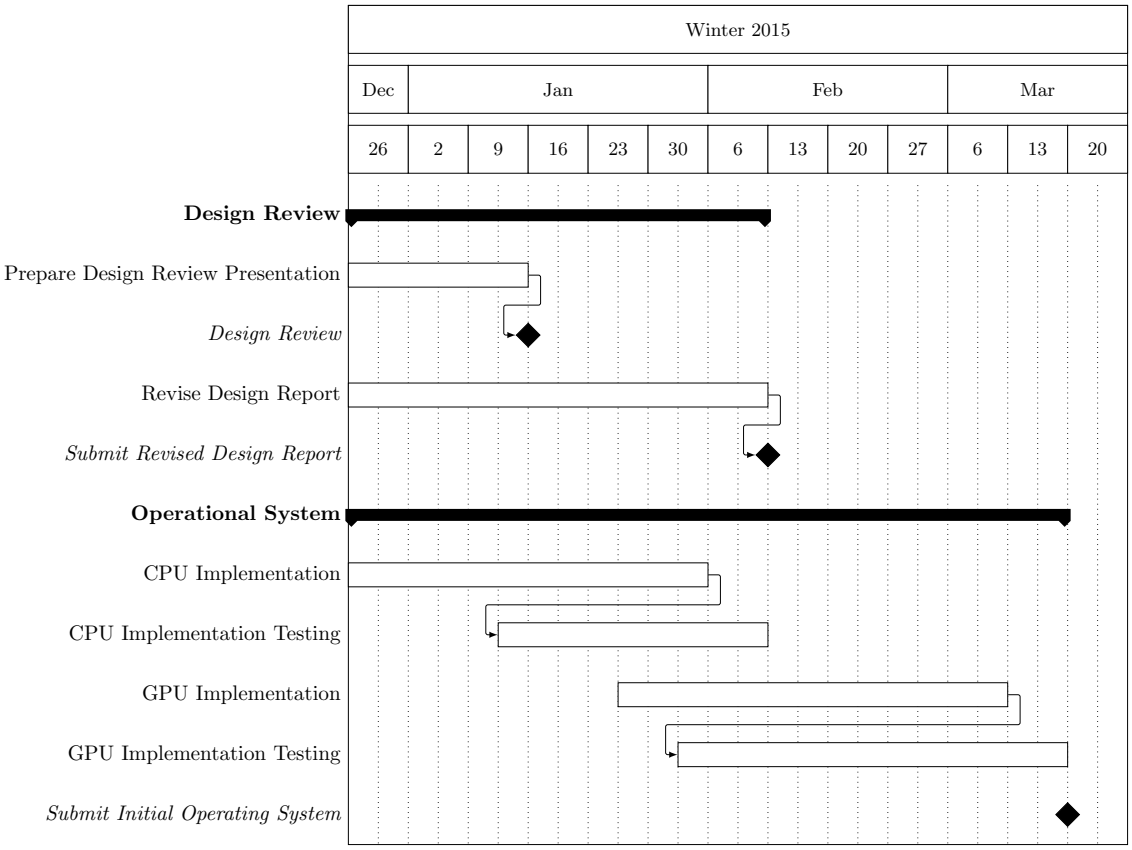

Figure 10.2: Winter quarter gantt chart

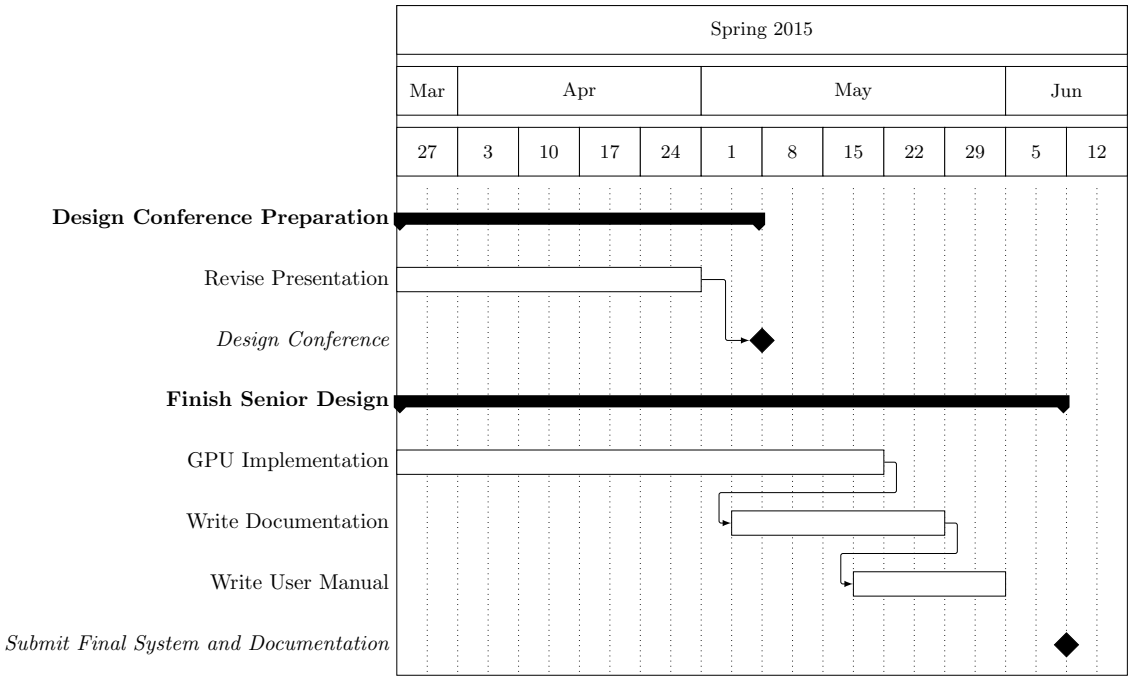

Figure 10.3: Spring quarter gantt chart

Bibliography

[1] Ahmad B. A. Hassanat. Visual speech recognition. *CoRR*, abs/1409.1411, 2014.

[2] Jian Li, Yuqiang Lu, Bo Pu, Yongming Xie, Jing Qin, Wai-Man Pang, and Pheng-Ann Heng. Accelerating active shape model using gpu for facial extraction in video. In *Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on*, volume 4, pages 522–526, Nov 2009.

[3] Sbastien Stillittano, Vincent Girondel, and Alice Caplier. Lip contour segmentation and tracking compliant with lip-reading application constraints. *Machine Vision and Applications*, 24(1):1–18, 2013.

[4] Jinwei Wang, Xirong Ma, Yuanping Zhu, and Jizhou Sun. Efficient parallel implementation of active appearance model fitting algorithm on gpu. *The Scientific World Journal*, page 13, 2014.

[5] Mau-Tsuen Yang, Zhen-Wei You, and Ya-Chun Shih. Lip contour extraction for language learning in vec3d. *Mach. Vision Appl.*, 21(1):33–41, October 2009.