Upgrading and Automating the Virginia Tech Single-Plate Interferometer

by

Henry C. Grabowski III

submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science

in

Aerospace Engineering

APPROVED:

Dr. Joseph A. Schetz

Dr. Wing F. Ng

Dr. Charlie L. Yates

September 1999

Blacksburg, Virginia

Upgrading and Automating the Virginia Tech Single-Plate Interferometer

Henry C. Grabowski III

ABSTRACT

The single-plate interferometer is a powerful flow visualization and aerodynamic measurement tool. It can provide full-field data for the density distribution in a non-intrusive manner, and it can be used for highly unsteady flows. While the device itself represents a large decrease in complexity over other forms of interferometry, the data reduction procedure has traditionally been laborious and difficult. To remove these difficulties and to improve the accuracy of the Virginia Tech interferometer setup, the software has been revamped into a black box design removing the need to handle the code directly. Furthermore, the software has been made to be platform independent by implementing the algorithms using the *Java* programming language. New hardware has also been added which further simplifies the setup procedure.

The improved setup and the new software is used to study the flow around a film cooled turbine blade in the Virginia Tech cascade wind tunnel. The study of this flowfield is used as a validation for the new algorithms and to illustrate the ease of use of the system. Through this analysis, the density distribution for the entire flowfield is acquired. Furthermore the use of Plexiglas as window material was tried. This proved to work, however the manufacturing processing of these windows proved relatively difficult. Studying the film layer close to the surface proved difficult because of inherent limitations with the single-plate interferometer.

Acknowledgements

I would like to personally thank everyone who assisted in making this research and my thesis come to fruition. First and foremost I would like to thank God for giving me the abilities I have and gracing me with some good luck when I needed it. I would also like to thank my parents and family for raising me with care and love. They've provided a firm foundation on which to build on.

I would like to thank my committee chairman, Dr. Schetz for providing me with invaluable guidance in regards to research and life in general. I would also like to thank my other two committee members, Dr. Ng and Dr. Yates, for being flexible with their schedules to help me expedite the completion of this research.

I would also like to thank the fellow students who were on the research team at the same time as I. I thank Oliver Popp for his engineering insight and for designing an excellent film cooling experiment. His well thought out design made performing all of our experiments as easy as possible. I also would like to thank Jim Bubb and Dwight Smith for their efforts in building and tweaking the experimental apparatus and the miscellaneous software for the film cooling experiments. I would like to thank Angela Wesner for leaving behind a well established and easy to use interferometer system.

I would also like to thank all my friends for sticking by me when I was stressed out and grumpy from working too long. They definitely helped me get through the hard parts. In particular I would like to thank: Thomas, Matt, Hurst, Jennifer, Alex and Rob.

Thanks all.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Like many other testing methods used in aerospace and mechanical engineering, interferometry provides an effective way of studying relevant flowfields. The advantage of interferometry and other optical methods is their ability to study a flowfield without disturbing it. While these methods introduce their own problems and complexities, optical methods are frequently used to complement other forms of measurements in these fields.

Interferometry is especially effective in the study of high-speed flowfields because it offers a quantitative method to determine density fields. While all optical methods work on the principle of density changes, interferometry is the only one which allows for direct, quantitative measurement of the density itself. Some work has been done to correlate quantitative density data by use of schlieren and shadowgraph systems, however they have never been as effective as interferometry.

Until this body of work on the single-plate interferometer was completed at Virginia Tech, it was generally difficult to do interferometry experiments. This was because the only type of interferometer which provided quantitative data was the Mach-Zehnder interferometer [4]. While this device produced the required data, it was laborious and difficult to set up. The single-plate interferometer, on the other hand, is easy to setup and still provides the quantitative density information[9]. Unfortunately, no procedure for extracting this data in a routine manner had been developed. Over the last decade, a reduction procedure has been created and refined at Virginia Tech,

to create a viable means of reducing interferograms into actual density information using the single-plate interferometer.

The development of the single-plate interferometer began at Virginia Tech with the work of Tibor Kiss in the early 1990's[4]. Through Kiss' work, a basic data reduction procedure was produced. Previous to this work, the single-plate interferometer could only be used to do a qualitative study of a flowfield. The ability to use the single-plate interferometer for flowfield measurements is very important. Along with the initial theory, a preliminary reduction algorithm was also introduced.

The data reduction procedure at this point was very crude, because of problems with available software. Proprietary hardware and software based on Silicon Graphics equipment was necessary. Furthermore, the interferograms had to be manually digitized on a graphics tablet. The hardware for the apparatus was also lacking. Existing optical components were used to minimize cost and prove the concept. These provided a relatively cumbersome arrangement and one which was without great accuracy in regards to physical setup. The original imaging system was a traditional high-speed film camera. This provided acceptable pictures for the relatively steady wake field, but it is unacceptable for unsteady work involving faster events, such as passing shockwaves. The imaging system was furthermore expensive and difficult to use.

In 1996-1997 the interferometer system was greatly enhanced by Wesner and co-workers[10]. Although the optical hardware was basically the same, the addition a high-speed digital camera removed a large portion of the complexity of the original system. The system at this point became more turnkey. Unfortunately, the rest of the optics were still relatively crude, because they still consisted of existing spare parts. Furthermore, although the camera created acceptable photographs, it was neither owned by the research group nor was it optimal for our general optical setup.

The data reduction procedure was also greatly improved. A semi-autonomous image processing program was developed to help with interferogram data reduction. Unfortunately, the final product of these algorithms required manual correction, however modern off-the-shelf image editing software was able to be used for this process.

The actual data reduction program was also made to allow for stepping in all cardinal directions, not just one. This represented the first truly viable system available for single-plate interferogram reduction.

In the current work, a series of developments have made the single-plate interferometer a valid system for normal data collection systems in applications where interferometry is advantageous. The on-site hardware was updated to use precise positioning systems. This allows for angular and linear placement of the optics to a high degree of accuracy. The optical setup has been further augmented with a high-power laser and a high-speed digital camera specially suited for this application. The overall preparation of the interferograms during data collection has further been refined to avoid pitfalls which became apparent during previous experiments.

In addition to the incremental improvements in the optical system that have been realized, great strides have been made in the area of software development. The original fully functional software was based on C code which was made as nearly ANSI compatible as possible. This was done to reduce cross-platform compilation problems. By making software in this manner, however, the software was very limited and crude. No graphics or GUI commands can be used in ANSI compatible software, because no two platforms handle these commands in the same way. Furthermore, the initial program proved to be resource intensive in terms of memory, disk space and overall number of calculations. Certain conventions used in the original configuration were incorrect for general interferograms, and they proved to require significant manual correction for accurate data reduction. To make the program cross platform capable, but still give it graphics and GUI abilities, the code was rewritten in *Java* . This was part of an overall rewrite to make the code more efficient. While most people consider *Java* a poor choice for scientific computing, it is actually rather capable. Memory management and object oriented programming (OOP) are more streamlined in *Java* . Furthermore, the program can be run on any platform with a *Java* runtime environment. This represents the vast majority of operating systems, whether past, present or future. The performance penalty in terms of overall speed is on the order of only 30%-50%.

Several algorithms used in the program were streamlined to minimize memory and processor requirements. In order to make the program applicable to general interferograms, a new line-following algorithm was also developed. To allow for quick and easy tweaking of program parameters, a GUI and input file system was created.

The overall result of these efforts is a system which is easier and more accurate to set up. The new software is now capable of running on all major platforms and provides an intuitive and capable interface for the user. The software further requires less resources, less processing time and allows for easy reduction of interferograms to density field data.

These new procedures and software were tested by performing an experiment in conjunction with an existing project. During the development of the improvements, research on turbine blade film cooling was underway. This type of research is very applicable to interferometry. By performing the experiment, it was possible to find shortcomings in the system and to fix them. It was also possible to prove the ease of use of the new software and hardware.

This thesis begins with a comparison of various types of optical flow visualization procedures. The interferometer is then discussed in a theoretical way. Practical considerations in regards to hardware selection is then presented. The software development is then covered in detail to familiarize the reader with the overall design and improvements to the data reduction software. Finally, the application of this software to a representative test case is illustrated.

Chapter 2

Interferometry

Several optical methods are frequently used in high-speed aerodynamics[5]. All of these methods work on the fact that physical properties of light are not constant through areas of varying composition and density. In particular, the index of refraction and speed of light through a certain gas change with density. The change in these two physical properties form the basis of optical measurements in high-speed aerodynamics.

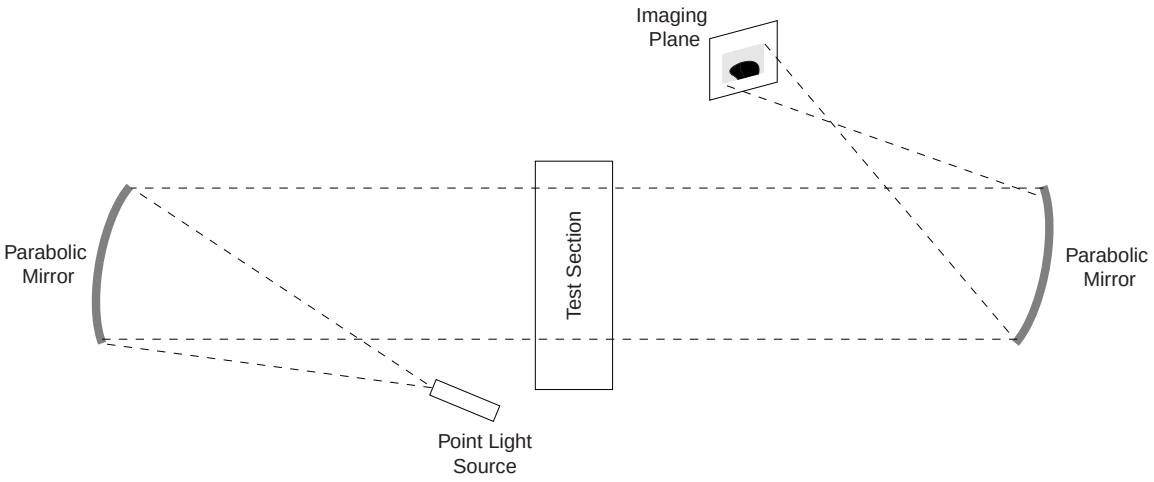

Figure 2.1: Illustration of a shadowgraph setup

The easiest of these systems is the shadowgraph system. This system is sensitive to the second derivative of the density of a flowfield. This provides an effective way of studying shocks and other abrupt and strong changes in density. The shad-

owgraph is also the easiest of the optical systems to set up. The lack of sensitivity, unfortunately, can pose a problem when studying gradual changes in density. A representative illustration of a shadowgraph system can be seen in Figure 2.1.

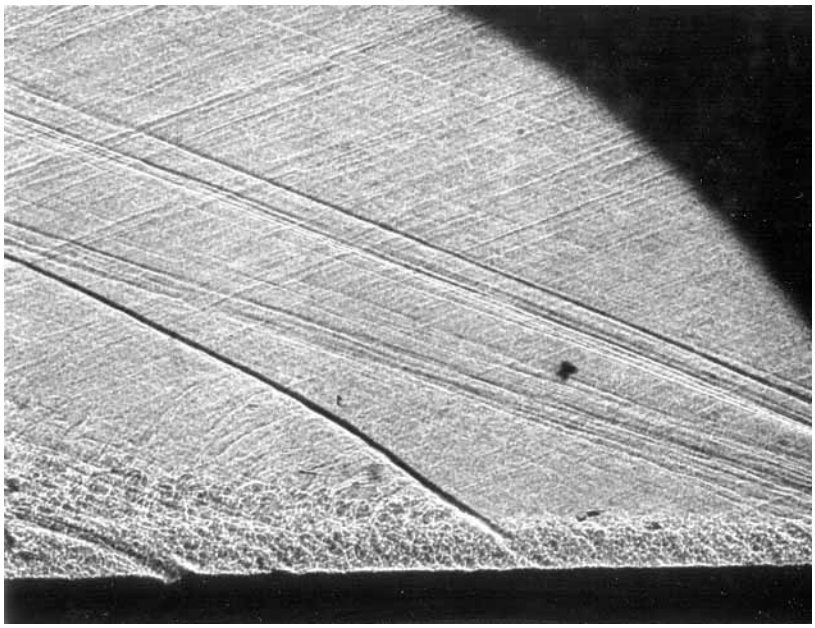

Figure 2.2: Example shadowgraph of a supersonic boundary layer (flow from right to left)

The light source for this type of system is a point source which is placed at the focal point of a parabolic or spherical mirror. This creates a collimated beam of light which is passed parallel to the test section surfaces. Changes in the density bend the light, as the index of refraction changes. This will create light and dark areas. The optics following the test section are used to manipulate the scale of the image which is focused on the image plane of the given imaging system.

A shadowgraph can be seen in Figure 2.2. This is a shadowgraph of supersonic flow over a flat plate in the Virginia Tech supersonic tunnel . The shockwaves are clear in the image. These are all caused by slight changes in the flow direction due to a flow disturbance or a defect in the surface. The strong shock is caused by a helium jet which is being injected into the flow from the bottom surface. The shockwave slightly downstream comes from the joining of two plates on the wall. It is clear that the sharp changes in the density which occur across a shockwave are clearly

represented in a shadowgraph. Also easily seen is the boundary layer, which is the turbulent flow near the surface.

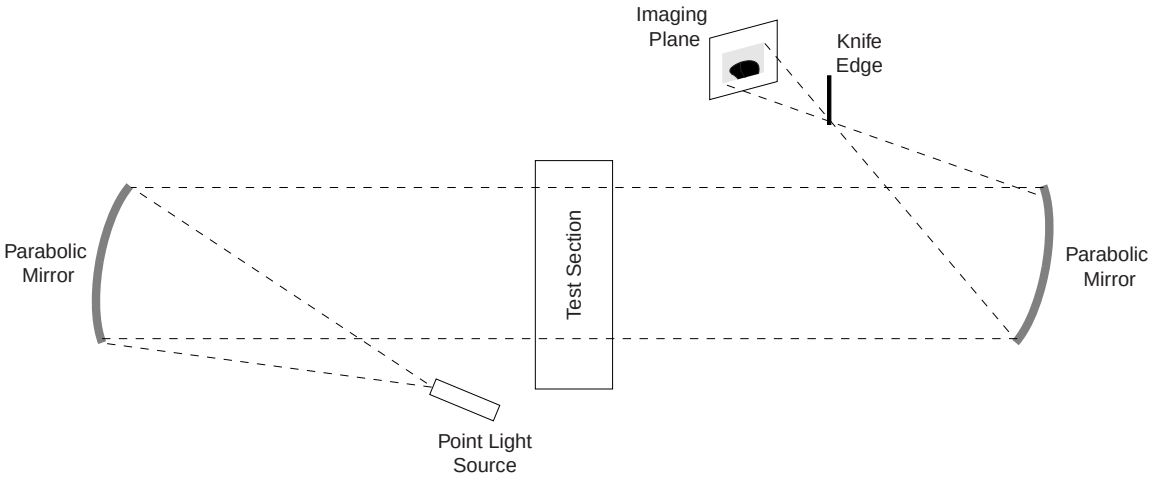

Figure 2.3: Schematic of a schlieren setup

To create a similar picture, but one which is more sensitive to density changes, a small modification to the above setup is made. By placing some sort of gradient selector, such as a knife edge, at the focal point before the imaging plane, it is possible to make the system sensitive directly to the density gradient. This is called a schlieren system, which in German means "defect." A diagram of this system is shown in Figure 2.3.

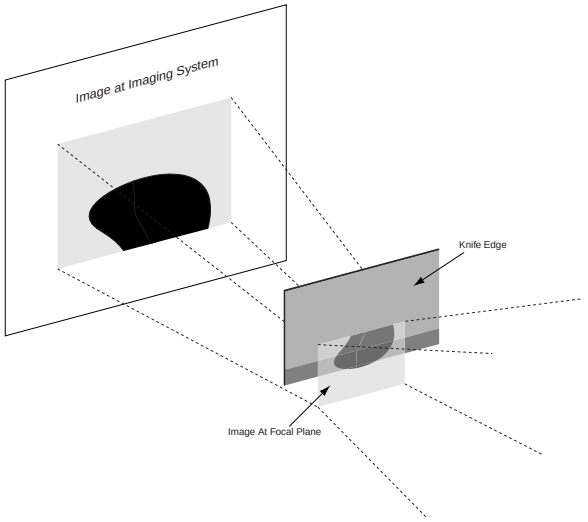

Figure 2.4: Illustration of "cutting" the light in a schlieren system

The knife edge at the focal point directly preceding the imaging plane is used to cut approximately half of the light out. A diagram of this can be seen in Figure 2.4. By cutting half of the focal point out, small changes in density will cause some light beams to be blocked by the knife edge. By analysis it is possible to show that this system is sensitive directly to the density gradient [3].

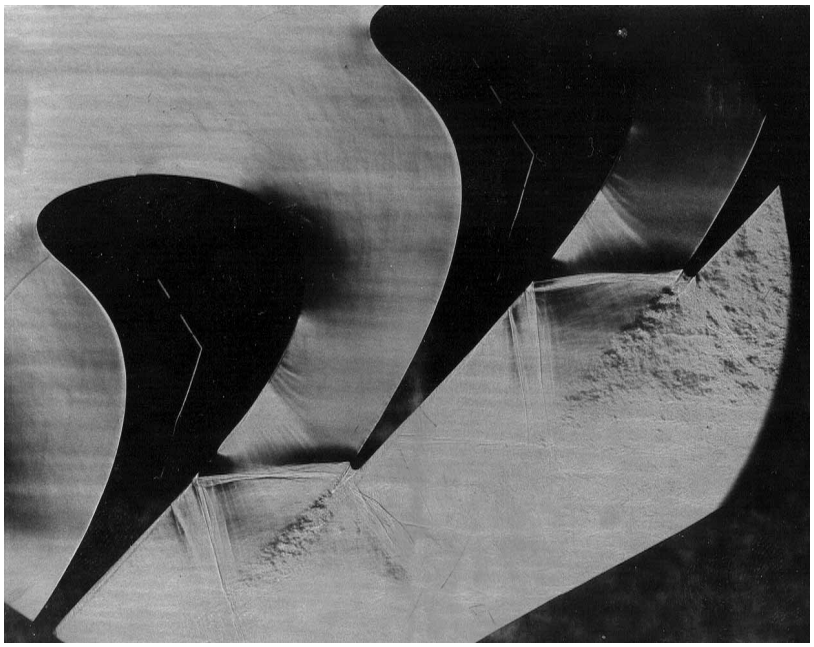

Figure 2.5: Schlieren photograph of transonic cascade flow

Figure 2.5 shows a schlieren of the flow in the Virginia Tech Cascade wind tunnel. The flow in this picture comes in from the left, and then exits through the blades parallel to the trailing edge surfaces. As the flow goes through the blade passages it is accelerated from a Mach number of 0.36 to Mach 1.2. The schlieren clearly shows Mach waves and shock waves being shed from the blade surface and trailing edge. Also apparent in the far passage, which is partially cut off, is a turbulence field being generated by the tunnel endwall. Comparing this image to the above shadowgraph, it is clear that the schlieren is significantly more sensitive to density changes than is the shadowgraph. A corresponding shadowgraph of the same flowfield would clearly show the shocks but would not show the detail in the wake and turbulence region in as much detail.

These two systems both use the changing index of refraction to create im-

ages. While these provide great qualitative representations of the flow, they cannot generally be used to extract actual density information. Part of the problem with both of these methods is that they do not record density directly, rather they are recording density changes. Numerical schlieren systems have been partially effective, by correlating a given brightness with a particular change in density[5]. However, even this system has it's shortcomings. For quantitative density measurements, an interferometer is used.

2.1 General Theory of Interferometry

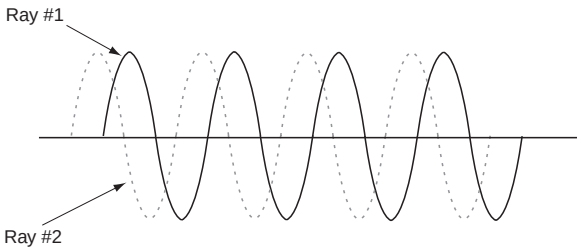

Figure 2.6: Two superimposed light waves

The interference of light results from the wave nature of light. If one were to imagine two light waves of the same frequency and magnitude superimposed on one another, as in Figure 2.6, it is clear that the phase of these waves affects the level of cancellation or amplification.

In all interferometry, if two previously parallel beams are split and then recombined, it is possible to create interference patterns. These interference patterns will result from the fact that the beams traversed paths of varying path lengths, or areas with varying index of refraction. Using a laser it is possible to emit waves which have high coherence; i.e., waves that will remain in phase over large distances or over a long time. This is a qualitative description of an important term called the **coherence length**. For single-plate interferometry, a high coherence length is important.

In flow visualization, interferometry uses the fact that the speed of light through a medium is directly related to the index of refraction by:

$$c = \frac{c_0}{n} \tag{2.1}$$

Therefore, if two beams travel over areas of different densities, the effective optical path length will be different for the two beams. If the coherence of the beam is high enough, this difference will create an interference pattern.

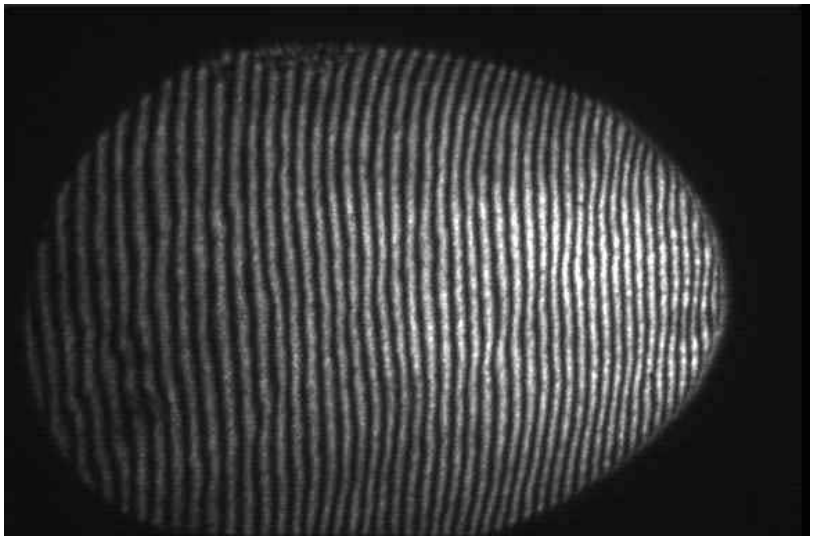

Figure 2.7: Interferogram with no field distortions

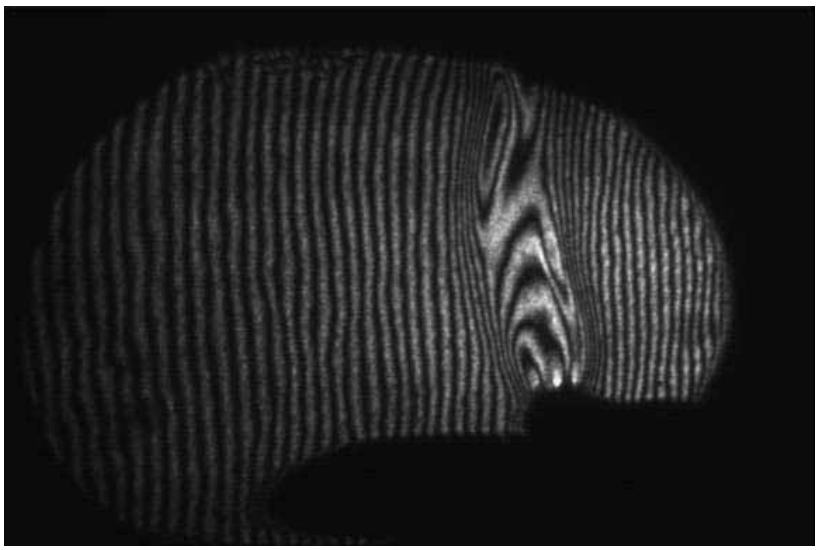

Figure 2.8: Interferogram of field with density distortions

Examples of this interference principle can be seen in Figures 2.7 and 2.8. In the first figure, we see the fringe formation that occurs because of the inherent optical

path difference built into the optical setup. All the fringes are uniform and parallel because of the way this system is arranged. It is also possible to have radial fringes instead of linear fringes. When a flame is introduced into the light path, the density and thus the effective optical path changes. This creates the fringe distortion seen in the second image. By measuring the fringe shifts it is possible to determine the density everywhere in the photograph.

2.2 Mach-Zehnder Interferometer

The Mach-Zehnder interferometer was designed in the early 1890's independently by Mach and Zehnder. Their design was actually an improvement made on an interferometer designed by Jamin in 1856 [5]. Because these original designs predate the invention of the laser by almost a century, these systems were made using a standard thermal light source, such as a mercury arc lamp, that was filtered and polarized.

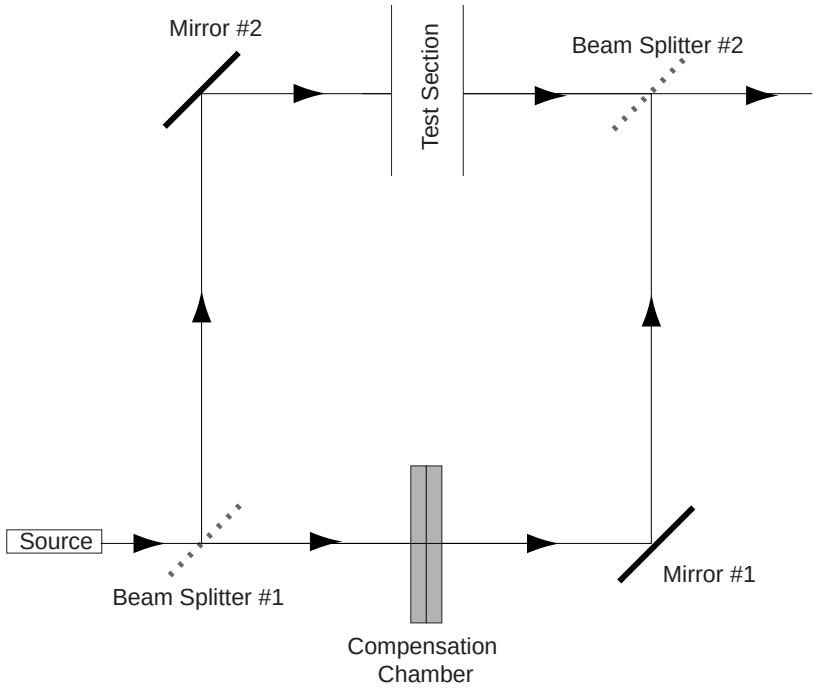

Figure 2.9: Schematic of a Mach-Zehnder Interferometer

The basic Mach-Zehnder arrangement can be seen in Figure 2.9. The light source for the system is a collimated beam of light, just as in the other optical setups,

but here it must be monochromatic. Upon reaching the first beam splitter the light is split into two distinct beams. The first beam continues traveling through the beam splitter towards the first mirror. The second beam reflects off the beam splitter surface and heads towards the second mirror. Both beams are then reflected off mirrors and travel towards the second beam splitter. The beam which reflects off the second mirror passes through the test section before reaching the second beam splitter. The other beam goes through a compensation chamber. This device is used to counter the effects of passing through the test section windows. Upon reaching the second beam splitter, the two beams are recombined to form an interference pattern.

One way of setting up this interferometer is to have all the mirrors tilted at 45° . In this configuration, in the absence of density changes through either of the beam paths, the wavefronts will remain perfectly aligned. Once density changes occur, fringe lines will begin to form. This is called **infinite fringe width** because the width of any two fringes in the no-flow case is infinite. If one of the mirrors is given an offset angle, then fringe formation will occur in the no-flow case.

Reduction of Mach-Zehnder interferograms is extremely simple. Given a fringe field generated by a Mach-Zehnder interferometer, knowing the density at any single point will allow for the calculation of the density at all points. The formula relating the density to the fringe formation is:

$$\rho_2 - \rho_1 = \frac{\lambda}{LK_{G-D}} \left(\frac{l}{d} \right) \tag{2.2}$$

where λ is the wavelength of the light, L is the test section width, K_{G-D} is the Gladstone-Dale constant for the fluid, d is the nominal fringe spacing, and l is the fringe count. This very simple analysis can even be performed manually in a trivial manner.

While the Mach-Zehnder interferometer has the advantage that the photograph taken has quantitative density information in it inherently, there are several draw backs to this type of interferometer. First of all, the physical arrangement of the optics requires a very high degree of precision. The optical components also need to be

mounted in a mechanism that limits vibration. Both of these concerns make the setup of this type of interferometer tedious and difficult. A second problem with this type of interferometer is its high susceptibility to density variations in the environment in which it is located. To avoid this problem, a very precisely controlled thermal control system needs to be employed in order to maintain a relatively constant and homogeneous temperature. Both of these above problems infer one last disadvantage of the Mach-Zehnder interferometer:cost. Because of the special mounting equipment, environmental controls and the extra precision with which the optical pieces must be manufactured, the Mach-Zehnder interferometer costs significantly more to implement than the single-plate interferometer.

2.3 Single Plate Interferometer

The single-plate interferometer is made possible by the invention of the laser. Until the laser, it was not possible to have two independent light beams of sufficient coherence to allow for an interference pattern to develop[9]. Unlike the Mach-Zehnder interferometer, the inherent path difference in the single-plate interferometer is created within a single optical component, the wedge plate. A diagram of the single-plate interferometer setup can be seen in Figure 2.10.

The lighting system for the single-plate interferometer is identical to the lighting systems used in the shadowgraph and schlieren systems. Unlike those systems, however the light source for the single-plate interferometer must be a laser with a sufficient coherence length. Following the test section is the wedge plate. The wedge plate is a standard optical quality piece of glass. This has a surface ground to $\frac{\lambda}{4}$ flatness. An exaggerated diagram of the wedge plate is show in Figure 2.11. Both the front and the back surfaces have an angle between them, β. In the case of the wedge plate this angle is slightly off 90° . The difference between this angle and a right angle is on the order of a few arcseconds. The equations to calculate the image separation distance and fringe spacing are developed in Kiss [4], but are presented in their final form below.

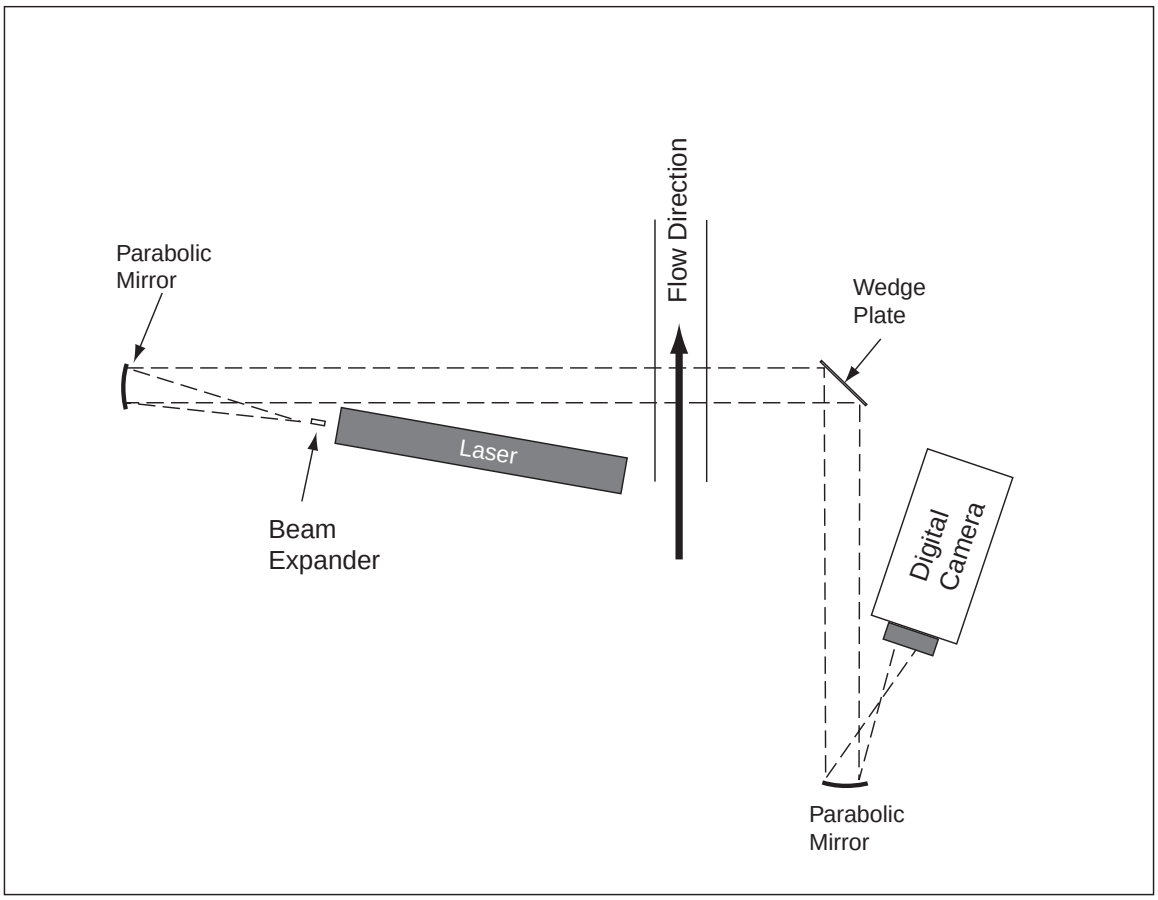

Figure 2.10: Diagram of the single-plate interferometer

The fringe spacing, a is given by:

$$\sin i_3 = n \sin\left[\arcsin\left(\frac{\sin i_1}{n}\right) + 2w\right] \tag{2.3}$$

$$\Delta i = i_3 - i_1 \tag{2.4}$$

$$a = \frac{\lambda}{\Delta i} \tag{2.5}$$

where n is the index of refraction, w is the wedge angle, and i_n is the incidence angle off a given face. Here, i_1 is the angle of the plate to the beam, which is generally 45°

.

From Figure 2.11 it is clear to see that a given beam will have two reflections, one off each surface. This fact creates an imaging effect known as the **double image**, which is unique to single-plate interferometry. If one has a solid object in the flow, such as a ball, then two images of the object will appear in the picture. The separation between two identical points in the image will be the image separation distance. To

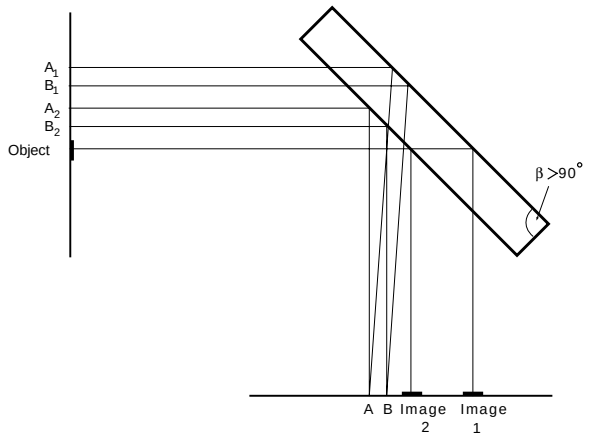

Figure 2.11: Diagram of the wedge plate

calculate the image separation distance, the equation below is used:

$$d = 2t\sqrt{\frac{\sin^2 i_1 - \sin^4 i_1}{n^2 - \sin^2 i_1}} \tag{2.6}$$

where d is the image separation distance and t is the thickness of the wedge plate. Complete details on the single-plate interferogram reduction process can be found in Chapter 5.

2.3.1 Practical Considerations

When developing a single-plate interferometer system, several important parameters need to be considered. The setup of the experiment is crucial for accurate and easy reduction of the interferograms to density data. In general, if the interferometer is being used solely for qualitative studies, then these are not going to be important.

One important parameter is the scale of the images. To reduce inaccuracies in the reduction process, one needs to know the exact scale of the interferogram. The best way of accomplishing this is to have a shape in the interferogram with a known size. By knowing the actual size and then measuring the number of pixels that the image covers, it is possible to determine the scale of the image in pixels per inch (ppi). With this information, all physical parameters can be accurately scaled into pixel coordinates. If there is an inherent, easily measurable object in the interferogram, such as a window border, then this could be used. Generally,

however, this is not the case. For these cases, a template could be used. Because it is also important to be able to consistently and accurately locate physical places in the image, the template should provide a fixed coordinate system. The template used in the present setup can be seen in Figure 2.12. This template was printed out on a clear mylar sheet and then mounted to the wind tunnel wall. Two dowel holes are provided on the template. These correspond to two dowel holes in the endwall. By mounting the template through these holes, it is possible to very consistently position the registration holes.

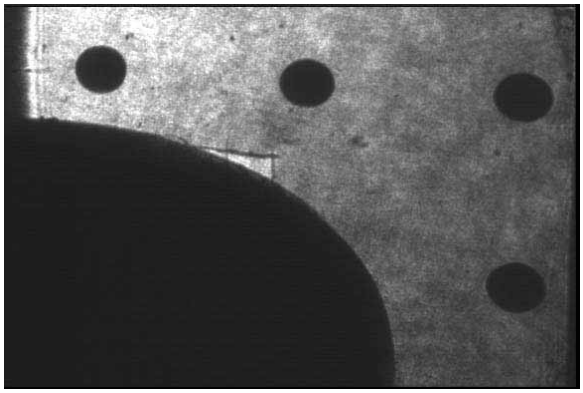

Figure 2.12: Registration Template

With the scale of the image available, it is now possible to calculate the exact location of the known density field. The creation of the known density field however, must also be done with care. Poor choices in field placement and sampling can lead to poor results.

For single-plate interferogram data reduction, one needs to known the density over a reference field (see Chapter 5 for details). The first important thing about the reference density field is that it should be in a relatively uniform region of the flow. If there are large fluctuations in the flow area around the known field, then errors in scale and placement are more likely to be apparent in future steps. This is because the sampled field will inaccurately place density changes which don't correspond to actual flowfield features, and this will taint calculations.

When sampling the density field it is also imperative that the sampled points be sufficiently spaced to provide a good resolution of flow features. If the sampled

pressures are too far apart, then flow details which need to be in the known field will be absent. This will taint the density calculations throughout the flowfield. The resolution of the samples will depend on how uniform the flow is in the area that is being sampled. As the flow variability increases, so too should the resolution.

With the sampled densities in hand, it will be important to interpolate the reference field to fill in all pixels between the sampled points. The choice of interpolation algorithms is critical in creating an accurate representation of the actual reference region. The algorithm used in our analysis is the Krige interpolation which is available in *Tecplot*. This algorithm provides smooth and realistic density distributions in areas of variable density. In general, the interpolation is done sampling all measured points, and then applying a decay constant to distant points. The details of this algorithm can be found in *Tecplot* manuals.

In general, there is no way to directly measure the density. Measurements of temperature and pressure can be used to calculate density via the Ideal Gas Law. Because of variability of these parameters between runs, for the current tests it was better to measure static pressure only. From the static pressure, it is possible to calculate the isentropic Mach number. By having the isentropic Mach number and the upstream stagnation temperature and density, it is possible to calculate the local static density. These are all based on the standard isentropic relations and the Ideal Gas Law.

2.3.2 Overview of the Data Reduction Procedure

The overall data reduction procedure can be easily put into a simple, universal form for qualitative understanding. With an outline of the data reduction procedure in hand it is easier to follow the details behind the software that is used in the processing. Given below is an outline of the basic steps for using the interferometer to calculate a density field.

- Sample a reference density field, preferably in a uniform region of flow. The field needs to be at least as large as the image separation distance in both directions

- Based on the sampled points in the density field, create a reference field via interpolation
- Take a picture of an interferogram with no density variations anywhere in the field (no-flow interferogram)
- Take a picture of an interferogram with the desired flowfield present (flow interferogram)
- Image process the interferograms so that all fringes are resolved into single pixel wide lines.
- Find the scale of the interferograms and determine the location of the reference field inside of the images
- Based on the local fringe count number in both pictures and the reference field, calculate points outside the reference field
- Repeat the last step until the entire flowfield has been computed

From this procedure it is clear that the reduction procedure is considerably more involved than in the Mach-Zehnder interferometer. In the past, this has made the Mach-Zehnder interferometer the choice device for performing quantitative density analysis. By encapsulating the single-plate interferogram reduction procedure into a software black-box, this is no longer an issue. However, discussion of the procedure is still important.

With this crude list of procedures, the basic concept behind the interferogram processing program can be understood. The last steps in the process, such as creating a fringe field and calculating unknown points, are all handled by the code given here. The experimenter now only has to be concerned with the experimental measurements and image processing.

2.3.3 General Program Structure

An overview of the object structure of the program used in this system is presented here to aid in the visualization of how the various pieces fit together. There are really two distinct areas of processing: image processing and density processing. The two main programs for these operations are *javaprocess* and *jbips* (*Java* Based Interferogram Processing System). These two units are discussed in detail below.

The convention used in the object charts is to show the association of each class definition with each other. The only classes shown in the object charts are non-standard classes. For example, the *grayImage* class uses certain *Java* media API's. These are not shown. However, the *grayImage* class uses a custom class called *imageGetter* and this is shown since it is proprietary.

The shading and line style convention separates the various types of classes. Classes which have a white background require that an object of that type be created before the methods are accessible. Classes with a gray background have been designed to be used by calling the routines statically. The class *iniFiles* was not developed by the author. This is a public domain class definition found on the internet written by Aresch Yavari [11].

Figure 2.13 shows the object chart for the the *Java* program *javaprocess*. The main object is the *javaprocess* object. This is the code which is run directly from the command line. Three other classes are directly used by *javaprocess*: *grayImage*, *binarize*, *hilditch*. Of the three, *grayImage* is the only one which is used to create an object. A grayscale image is loaded into a *grayImage* object. This is the central object which undergoes the various processing steps. The *binarize* class encapsulates all of the methods required to perform an adaptive binarization on the image. The *hilditch* class encapsulates all of the methods required to turn the binarized interferogram into a single point wide line.

The *grayImage* class itself uses the class *imageGetter* to import graphics. The *imageGetter* class encapsulates the required methods to load *GIF* and *JPEG* images into the *grayImage* object. In some cases, it may be advantageous to do some manual

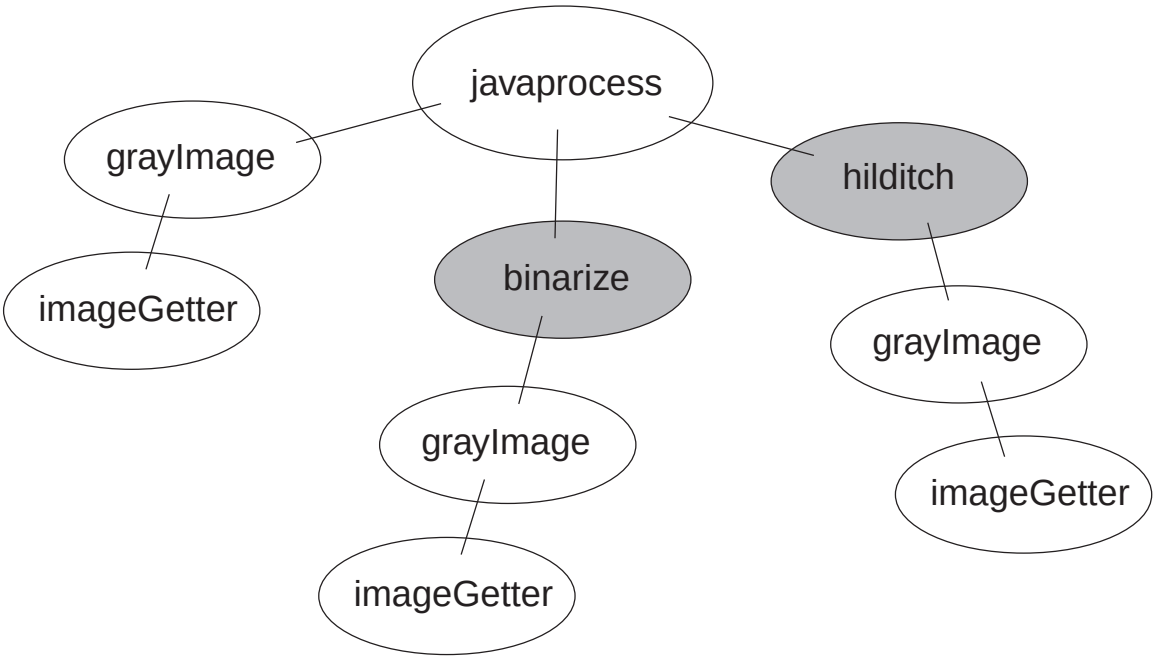

Figure 2.13: Object chart for program *javaprocess*

image correction between binarization and line thinning. To aid in this, both *binarize* and *hilditch* can be run as *Java* programs. In the case of all three, the required input is either as a *GIF* or a *JPEG* image. It is recommended that a *JPEG* be used for the initial picture format, and that a *GIF* format be used for the binarized and thinned images. This is because the *GIF* is a lossless compression. In some instances, the *JPEG* does not maintain a truly two bit image during compression. This can cause problems in other processing steps.

Figure 2.14 shows the object chart for the main interferogram processing program *jbips*. *iniFiles* objects are created to allow for the input of the initialization files. This routine removes the need to continually recompile the program. It further makes it so that no working knowledge of the *Java* programming language is necessary. The main storage object is the *interferogram* object. These objects store the basic fringe information and orientation of a given interferogram that has been loaded into that object. These objects are initialized by doing some image manipulations on a *grayImage* object. The static class *linepick* is used to label and then number the fringes of the given interferogram. The class definition *MathRead* is used to read in floating

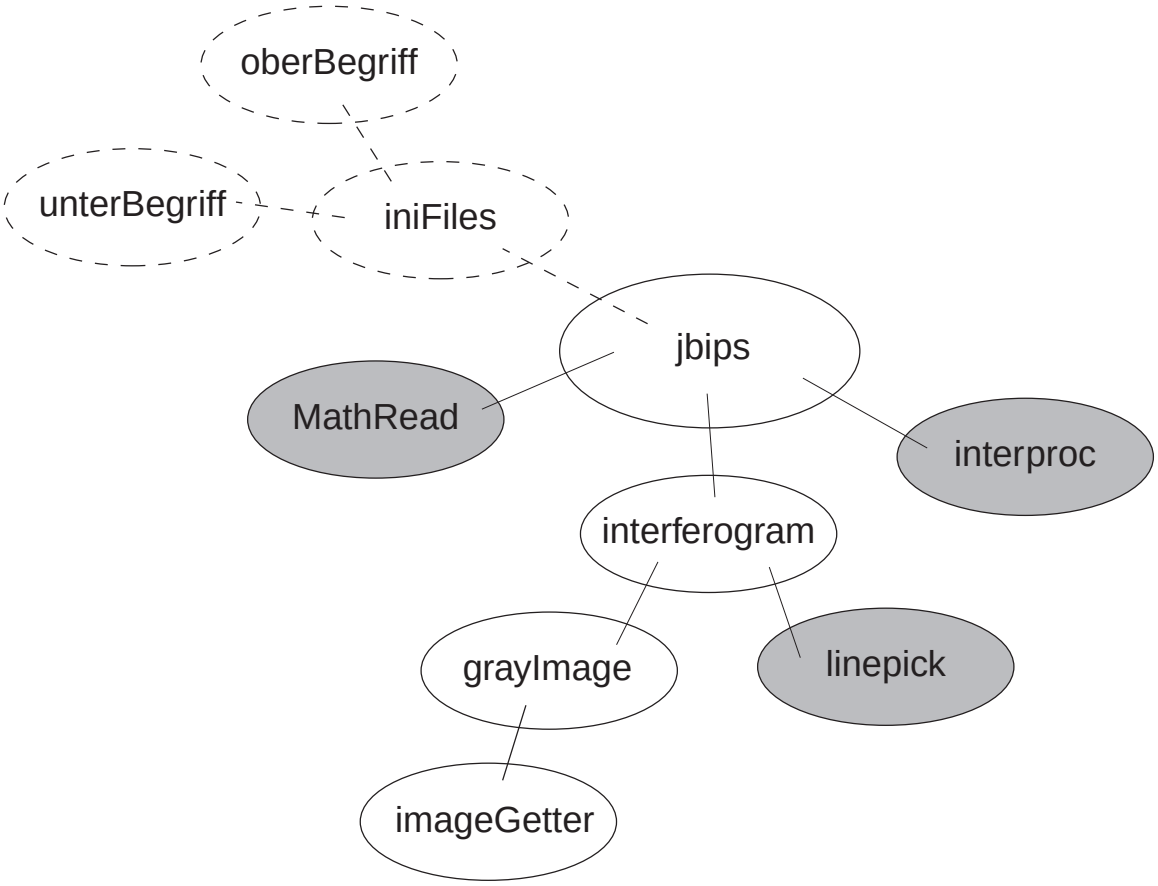

Figure 2.14: Object chart for *jbips*

point numbers from an *ASCII* file. The numbers must not be in scientific notation. The main stepping routines used in the density calculations are all encapsulated in the *interproc* object.

In the simplest form, the user should not have to interact directly with the *Java* code. The *.ini files have been designed to create a generic processing method, so that any interferogram processing problem can be approached with the compiled code.

Chapter 3

Optical System Design

For the upgrading of the hardware components used for flow visualization in the Virginia Tech supersonic wind tunnel lab, several optics pieces were procured. These pieces included optical components used solely for the interferometer and also pieces which can be used for general flow visualization. The requirements for general flow visualization and interferometry vary slightly. The discussion of procurement needs to begin with a general discussion of various optical hardware pieces. After the general discussion, the hardware chosen specifically for the interferometer is presented.

3.1 Optics

Because of the expense of optical components for flow visualization, it is important to look at what the requirements are for each of the optical components in the system. Although these devices are inherently expensive, one has to be careful not to set system design parameters too stringently. If system parameters are chosen carelessly, the cost of the system can easily increase by an order of magnitude. Further,it will take longer to build due to the use of non-standard parts.

3.1.1 Mirrors

Mirrors are required for every optical flow visualization experiment which is performed in the tunnel. Even the direct shadowgraph method, the simplest of all mentioned methods, requires at least one mirror. All mirrors in use need to be **first surface mirrors**. By this, we mean that the coating of the mirror needs to be on the surface facing the outside air. Mirror selection is based on two factors:

- How the mirror is going to be used in the optical system
- The type of coating needed on the mirror

3.1.1.1 Flat Mirrors

Obviously, the usage of a mirror is an important factor in the selection of the type of mirror. There are two purposes for a mirror: to collect or distribute light and to deflect light. A flat mirror is used to deflect light without changing the magnification factor of the optical system. It is said that a flat mirror "folds" the light path. A flat mirror is frequently used when the optical path needs to be turned or when the optical path needs to be lengthened.

Flat mirror geometry is strictly a function of the size of the flowfield that one wants to see. For example, if the flowfield in question is approximately four inches across, then a flat mirror larger than four inches is required. Care must be taken when selecting a flat mirror for an experiment, since flat mirrors are usually at a high angle relative to the oncoming light beam. For example, if one wants to turn the flow 45° , then the mirror will have a profile relative to the test section of an ellipse with a major axis equal to the diameter, and a semi-major axis equal to the diameter divided by $\sqrt{2}$. The situation gets worse as the angle increases. For this reason, one needs to know the incidence angle of the light beam relative to the flat mirror in use.

Furthermore, the surface flatness of the mirror is important. Flat mirrors in an ideal sense do not change the amplification factor of the flow. However, commercial mirrors with no flatness specifications will have mild surface curvature. This will cause uneven changes in the amplification factor in the optical system. For this reason the

surface flatness needs to be specified for these optical systems. Glass surface flatness is routinely measured relative to λ, where λ is the nominal wavelength of the light reflecting off the surface. For example,a surface flatness specification of "1-λ" is called **one wave flatness**. A surface flatness of "$\frac{\lambda}{4}$" is called **quarter wave flatness**. Selecting an adequate, though not overly stringent, flatness is key to purchasing a flat mirror.

One problem which occurs in specifying mirror flatness is cost. As the flatness of a mirror becomes more and more stringent, the cost rises considerably. Table 3.1 shows a list of flat mirrors of the same diameter as the flatness increases. The flattest mirror costs over six times more than the poorest mirror. Part of this is due to a change in material which is required in order to grind the mirror to such a stringent specification. This has been minimized by selecting the cheapest material for each of the selections. In general, mirrors for flow visualization systems should have at least quarter-wave flatness.

Surface Flatness	Price($)
$3\lambda/25mm$	86.00
$\lambda/4$	209.00
$\lambda/10$	252.00
$\lambda/20$	525.00

Table 3.1: Cost of a 10cm diameter flat mirror from Melles-Griot in 1999

One further problem can occur if the flatness is chosen unwisely. As the flatness requirement is made more stringent, the maximum size of the mirror becomes limited. In general for the flatness specified here, standard mirrors go up to 10-15cm in diameter. Larger mirrors, for example a 30cm diameter mirror, become harder to find as a stock item. Furthermore, mirrors of excessive size imply prohibitive costs.

3.1.1.2 Amplifying Mirrors

The other type of mirror that is used is a mirror which does affect the magnification factor of the system. This can either be a concave or convex mirror, however in general the mirrors used here are all concave. Selection of the geometry of these mirrors is

more difficult than in the case of the flat mirror. These mirrors all have different focal lengths and magnification factors. These factors heavily drive the selection process in this case.

In the case of a single mirror system, for example in the schlieren system, the mirror focal length requirement is governed by the physical dimensions of the laboratory and the required magnification size. If we define the **magnification factor** as:

$$N = \frac{\text{Object Size}}{\text{Image Size}} \tag{3.1}$$

then we know the physical dimensions of the rest of the system:

$$O = \text{Object Distance} = f(N+1) \tag{3.2}$$

$$I = \text{Image Distance} = f\left(\frac{N+1}{N}\right) \tag{3.3}$$

where f is the **focal length of the mirror**

The **object distance** is the total distance from the object which needs to be in focus and the center of the mirror. For example, if there is one foot between a flat mirror and the test section followed by ten feet between the flat mirror and the parabolic mirror, then the total length will be eleven feet. The **image distance** is the total distance between the mirror and the imaging system image plane. This simple formulation works for a system containing a concave mirror. The formulae for dealing with compound optical systems, containing two or more components, is beyond the scope of this thesis.

Obviously, for a single mirror system the required magnification is going to drive the focal length so that the system fits within a reasonable space. Unfortunately, there is a limited amount of freedom in selecting mirrors based on their focal length. In general, as a mirror's focal length decreases, the maximum diameter available for stock mirrors decreases as well. A common measure of a mirror's focal length is called its **f-number**. An f-number is defined as the ratio of the focal length to

the diameter. As the f-number decreases, the manufacturing process becomes more difficult. Therefore, an f-number of no less than f4 is recommended.

As the f-number decreases, the type of mirror surface shape must also be taken into consideration. There are two types of mirrors, parabolic and spherical. Spherical mirrors are much easier to manufacture and thus cost less. Unfortunately, there are some problems with spherical mirrors. As the edge of the mirror is approached, the effects of astigmatism increase. Therefore, if spherical mirrors are used, the light should not reflect off the outer surface of the mirror. Spherical mirrors should also be avoided in low f-number situations because this astigmatism effect is exaggerated. In general if the required f-number is smaller than f5 or f6, then a parabolic mirror should be used [3].

Parabolic mirrors do not suffer from astigmatism near their edges, because the surface is ground in the proper geometry for each point on its surface to have the same focal point. Near the center, both the spherical and parabolic mirrors have similar geometries. Near the edges however, rather than maintaining a constant curvature about the focal point, the edges follow a parabolic line. This reduces astigmatism.

One final variation on these types of mirror is the off-axis parabolic mirror. This can be used to reduce the effects of coma in the optical setup. In the case of the off-axis parabolic mirror, the light source can be perfectly aligned with the primary axis of the entire optical system. Although this is a superior setup for the lighting system, it is rarely needed.

3.1.1.3 Mirror Coatings

Besides the mirror geometry one has to take into consideration the coating used on the surface. A mirror coating is the reflective material applied to the surface. All mirrors for the currently used optical methods must be first-surface mirrors. This requirement creates a problem, since the coating is directly exposed to the elements.

Although it is possible to have a coating applied without any protection, it is unwise and rarely done. Without the coating, not only will the coating be prone to coming off when touched, but the metal used will begin to oxidize. Various coatings

are available under vendor specific names. In general, to minimize the deterioration due to periodic cleaning, the toughest available coating should be used. Regardless of the coating however, the surface should **never** be touched. Furthermore, cleaning should be done with a non-abrasive cloth and mild cleaning solution. Cleaning should be done as rarely as possible. To clean dust off the mirrors,clean compressed air should be used.

The wavelength of the light being reflected is very important when selecting both the metallic and protective coatings. Silver and aluminum have a good reflectivity over a very broad range of wavelengths. Unprotected silver mirrors have very high reflectivity from wavelengths of 250nm to 20μm. These correspond to the entire visible spectrum, the ultraviolet spectrum and the near infrared spectrum. Different protective coatings can limit the band over which the mirror has high reflectivity. In general, if the light source for the system is in the visible spectrum, then standard coatings will not be a factor. If on the other hand, one is working outside or near the edge of the visible spectrum, it is important to check the manufacturer's specifications on reflectivity. Furthermore, because silver is very susceptible to oxidation, aluminum is the coating of choice, despite the slightly lower reflectivity.

In the case of infrared imaging, silver and aluminum do not have a high reflectivity. In this case it is advantageous to use a mirror with a gold coating. Gold coatings are very efficient at reflecting in the near-infrared and infrared range. Unfortunately, they are very poor at reflecting at the lower end of the visible spectrum.

3.1.2 Windows

Window design is a critical part of the optical system design for supersonic wind tunnels . The window must provide enough structural integrity to withstand the pressure field generated by the tunnel, and it must provide clear optical access for the system being used. Several factors must be considered when designing the tunnel windows.

The first consideration is, of course, safety. The windows selected must withstand large pressure differences. A rudimentary calculation must be performed to

determine whether the tunnel windows will have excessive deflection. Besides failure, it must be ensured that the flowfield is not adversely effected by distortions in the window. Excessive stresses in the window will most likely create distortions in any of the flow visualization systems. These distortions will hamper the use of these images for study.

A second consideration that must be taken into account is manufacturing stresses. In the manufacturing process, it is possible to introduce surface distortions and residual stresses in the windows. In most cases, these will not be visible in a shadowgraph system, but will be readily apparent in a schlieren or interferometer system. These stresses can be minimized by carefully manufacturing the windows. For example, when drilling holes in plexiglas windows, the operator should continually cool the drill and drill the hole in steps, rather than in one pass. "Pumping" the drill will also reduce the distortions by allowing the surface to cool.

Material selection for windows is governed by both the test section design and the optical system requirements. If the windows need to carry any load parallel to the surface, then Plexiglas needs to be used. Because glass is a ceramic, it performs poorly under tension. Therefore, it is very difficult to successfully mount a load bearing device directly in a glass window.

Unfortunately, there are some cases where Plexiglas is inadequate for a good flow visualization system. If the window will be exposed to extreme pressure then Plexiglas will have larger material strains during a run than glass would. This is because the modulus of elasticity of plexiglas is lower than glass. These strains will become visible in both a schlieren system and an interferometer. For certain kinds of interferometry, the surface flatness is also critical. The required surface flatness for these optical systems is not possible with Plexiglas. In these instances, glass windows must be used.

3.2 Lasers

Lasers are useful when a system requires a bright, monochromatic continuous light source that is highly coherent. This is the case for high speed digital photography. Furthermore, lasers are used frequently, and they are required for some forms of interferometry. The term laser is actually used to describe a very broad range of devices. They all operate on the same basic principle, the description of which is beyond the scope of this thesis, but care must be taken when selecting a laser. The different lasers all have their advantages and disadvantages. For a more in depth description of lasers see Siegman[8].

3.2.1 Terminology and Basics

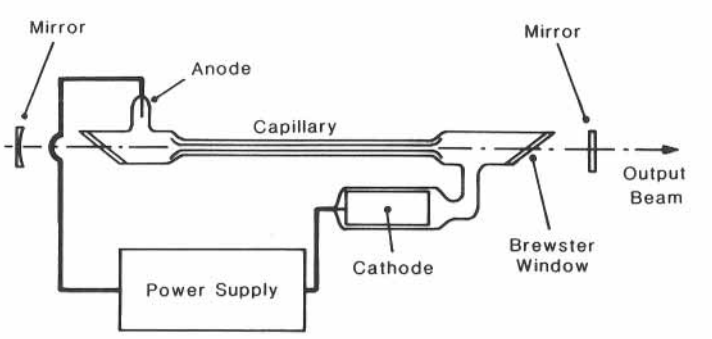

Figure 3.1: Basic schematic of a CW laser tube [2]

There are generally two types of lasers. The first type is a Continuous Wave (CW) laser. These lasers operate for long periods of time and provide a continual, non-varying light source. The second type of laser is the pulsed laser. These lasers are generally capable of much higher output powers, however they can only emit energy in short bursts. The type of laser required for a given setup would be driven both by the required light level and the type of imaging equipment used.

A basic continuous wave gas laser has a construction similar to what is shown in Figure 3.1. There is basically a tube of gas between two mirrors. One mirror is fully reflective, the other is partially reflective. Out of the latter mirror comes the

laser beam. While this is a very simplified version of the actual laser architecture, it is sufficient for the purpose of having a rudimentary understanding of lasers.

In general, multiple frequencies are emitted simultaneously. Through various efforts it is possible to have the laser emit a single frequency of light. This can be accomplished by limiting the tube length [2] or by adding a prism to the assembly which will only allow one wavelength to reflect efficiently. It is usually advantageous to have the laser emitting at one frequency, even though the overall power of the laser diminishes. In general, there are many frequencies (lines) that the laser will operate at, however usually only a couple are dominant. These are called the laser's **primary lines**. When operating the laser in single-line mode, it is best to use one of the primary lines.

While in an ideal case the laser would emit a perfect stream of radiation, this is obviously not the case in reality. There are certain beam parameters which are sensitive to certain experiments. One of these parameters is the **coherence length**. In a qualitative sense, the coherence length describes the overall correlation of wavefronts at a given point over a period of time[2]. The longer the coherence length, the higher the correlation between two points on a wavefront. In certain types of interferometry,this factor is important.

One problem with using a laser as a light source for flow visualization is that it has a relatively high coherence. Because of this, any dust particles or scratches on the objective will appear as Moire patterns.

3.2.2 The Laser as a Point Source

A laser beam has a very small divergence angle, and the beam diameter is generally very small to begin with (on the order of millimeters). For studying large flow fields, this small beam must be changed into a large diameter beam of light. This can be accomplished by putting a microscope objective in front of the laser. This will rapidly expand the beam. When sizing the objective it is important to remember that lasers beams have a Gaussian intensity distribution. For this reason, it is important to expand the beam in such a way that the flow field being study is covered exclusively

by the region of the expanded beam in the middle of the distribution.

3.2.3 Laser Architectures

By far the most popular gas laser is the Helium-Neon (HeNe) laser. This is a CW laser which has its primary line at 633nm. The HeNe laser's popularity is based on its ease of use. Unlike other gas lasers, a HeNe laser requires no tuning, unless the system is violently jarred. The HeNe laser is, therefore, a relatively "plug-n-play" type of system. The HeNe laser also has a very high coherence length, on the order of 100m. Maintenance for the HeNe laser is generally much easier than for other types of gas lasers as well. These lasers are not required to be run regularly, nor do they generally need any elaborate cooling systems. With newer dye based HeNe lasers, it is also possible to get beams of varying colors. The major problem with the HeNe laser is it's maximum output. In general HeNe lasers cannot be found to have output powers of more than approximately 100mW, and they usually will only have power between 0.5mW and 25mW . This proves relatively restrictive.

Another popular laser used in flow visualizations is the Argon-Ion (Ar^+) laser. This laser is also a CW gas based laser, however it is much more complicated than the HeNe laser system. The Ar^+ laser has its primary lines at 488nm and 514nm. There are other lines in the UV spectrum, however those will not be discussed here. In a standard mode, all lines are output, and selection can only be done with the use of a prism. The prism will only allow one line to be transmitted. If the prism is misaligned, then no energy will be transmitted through the beam. Because of the additional size and complexity the Ar^+ laser, the system must undergo continual tuning throughout its operation.

Unlike the high coherence length of light produced from the HeNe laser, the Ar^+ laser can have a very unstable beam. This can result in problems with single plate interferometry. To augment the beam stability, a device known as an **etalon** may be installed in the laser. This device can increase the coherence length considerably. The drawback to using an etalon is a beam power reduction of between 20%-50%.

The extra complexity of the Ar^+ system over the HeNe system is offset by the

gain in output power. Unlike the HeNe laser, the Ar^+ laser is cable of producing beam power on the order of several watts. The power output of the laser is generally adjusted by changing the input current to the tube. Power can further be increased by applying a strong magnetic field to the tube.

Ar^+ lasers that operate at levels below 100mW can be air cooled. If an Ar^+ laser is to be operated at high output powers, the laser will require a water based cooling system. The cooling system is usually a built-in component of the laser system. When operating a laser with this type of system, it is imperative to monitor the cooling of the laser at all times. Overheating the laser could result in a non-functioning laser and will most likely reduce the tube life.

There are several other types of lasers that are available for flow visualization. Some of them, like the Nd-YAG and ruby lasers are pulsed lasers. These lasers can have much higher power output than the Ar^+ laser. Copper-vapor lasers are also available for flow visualization. This type of laser is also a CW laser. Although the output power of this laser is much higher than the Ar^+ laser, it can not be used in interferometry because of its very poor coherence length. A description of the laser used in the current optical system is found in Section 3.4.

3.3 Imaging Systems

With all optical setups, there needs to be a device used to capture the image of the flow. This device is usually a camera of some sort. It is also possible to place a piece of smoked glass in the image plane and then take a photograph of that. This is useful in some applications.

There are two major subdivisions of imaging systems available, analog and digital. In the Virginia Tech facility the analog imaging system is a classic Polaroid camera. The digital camera in use most frequently in our tunnel is a high speed, multichannel digital camera. The details of each of these systems will be explained below.

3.3.1 Polaroid Photography

For taking individual, very high resolution photographs, Polaroid film is generally unsurpassable as an imaging system. In our shadowgraph and schlieren experiments, Polaroid 57 film is used. This is a very fast acting film, which has no grain. The lack of grain allows for very fine details to be resolved while still maintaining a relatively large view of the flowfield. The standard film area has a size 4x5 inches. This provides for a relatively large photographic area. Because of this, it is ideal for doing direct shadowgraphs.

Although it is possible to use this film with a continuous light source, by shuttering the camera, this is not done in our optical setup. In most flowfields under study, there are many important flow phenomena which occur at a very high frequency. For this reason a pulsed light source is used, since mechanical shutters can't operate at these frequencies. In general, the light source used for shadowgraphs is the Nanopulser. The Nanopulser is made by Xenon Corporation, and consists of a large capacitor with a small arc gap between the anode and cathode. This gap is jumped when the voltage is applied. This creates an intense spark with a duration between 10 and 20 nanoseconds. The exact position of the spark varies slightly for each firing. This makes the Nanopulser less than ideal for schlierens. For a schlieren system a Strobotac flash light source is used. This is a standard flash bulb attached to a Stobotac power supply. This system generates a flash in the same position consistently, however the flash is less bright and can only pulse for durations on the order of microseconds.

3.3.2 High Speed Digital Photography

For interferometry purposes, a CW laser is used. This requires a fast shuttering mechanism, since mechanical shutters can not operate fast enough to capture details such as passing shock waves. With the advent of digital photography, it is now possible to digitally shutter cameras very quickly. These cameras also have mechanical shutters, but the high shutter speeds come from digitally shuttering the imaging plate.

This will be discussed below. With digital cameras it is possible, even desirable, to use continuous light sources and then shutter the camera. Current technology allows for digital shutters to operate on the order of femtoseconds (10^{-15} seconds).

Unlike traditional photography, it is important to carefully scrutinize the digital camera that is specified for use. Usually there is a trade-off between the various camera specifications. Again, it is very easy to over-specify the camera requirements and therefore drastically increase the cost of the camera system.

Traditional commercial cameras are not acceptable for use in a flow visualization apparatus. The exception to this statement would be if the camera is used to photograph an image of a steady flowfield being projected on a screen.

One such parameter which needs to be considered is the camera resolution. All digital cameras use a Charge-Coupled-Device (CCD) to record the image. This CCD has a limited number of pixels in both the x and y direction. Furthermore, the pixel density is also a variable which needs to be considered. Standard CCD sizes are continually changing. CCD sizes vary widely, from as low as 400x320 to as high as 2048x2048. The cost of a given optical system obviously is a large function of the CCD size.

The importance of the CCD size becomes apparent when certain details need to be resolved in an image. In order to see a flow feature, it must occupy an area consisting of several pixels. In the case of a shock for example, it can be one or two pixels wide but must be many pixels long. The higher the number of pixels, the larger the physical area of the image may be. This is because there is a larger array for the image to fall on. The largest available CCD's are generally not required, and they cost too much money to be considered for most experiments.

Another factor which must be considered when sizing the CCD is the shutter speed of the CCD. The shutter speed required to accurately capture an image is governed by the unsteady flow phenomena which is being recorded. If the shutter speed is too long, then the image will be smeared. The required maximum exposure time must be determined ahead of time. For the case of a steady flowfield this is not a concern. As the required shutter time decreases, so too does the possible size of the

CCD.

The shutter in a high speed digital camera is generally a two-stage device. The first shutter is a mechanical shutter which operates on a relatively large time scale. This shutter is used to block light from the inside of the camera during long periods of time. Several milliseconds before the actual picture is taken, this mechanical shutter opens and exposes the CCD. At this point the second shutter is digitally activated. Although the CCD is continually exposed to the light, it will only integrate the light received over a period of time determined by its controller hardware. It is, therefore, possible to shutter the CCD very quickly.

One of the factors in minimum shutter times is image off loading. Once the image is captured by the CCD, the chip must then move the image into a secondary storage area. This may be another processor, a communications bus, etc. This operation becomes very prohibitive when the array size becomes excessively large. This is one limiting factor in CCD sizes for very high speed digital photography. The minimum speed continues to drop as the size of the arrays increases. For example, over a recent period of two years, the size and speed of the same priced camera from a given manufacturer has gone from 576x384 pixels at 10ns to 1280x1024 pixels at 5ns.

Minimum shuttering time is not just a function of the CCD itself, but also of the minimum light levels required. The amount of light available over very short shutter times may not be adequate to create viable pictures. One solution is to increase the light level. Generally however, the required increase is not possible. The alternative is to augment the imaging system with an **intensifier**. An intensifier is a device which sits on top of the CCD that acts like a light amplifier. Conceptually, a ray of light hits the intensifier, and the intensifier then emits several rays of light to the CCD. By doing so, the apparent light level to the CCD can be increased by several orders of magnitude.

Intensified cameras are also available in high-speed versions, but they generally cost more. It is further possible to pass the light through two intensifier plates. This can further increase the light level exposed on the CCD. Dual intensified systems,

however, do pose a problem. Unlike single-intensified systems, which are only damaged by exposure to direct sunlight, it is possible to cause irreversible damage to the CCD by exposing it to ambient room light. All CCD cameras can be irreversibly damaged by direct contact with a laser beam.

With digital cameras it is also possible to take multiple pictures. One way of accomplishing this is to expose a given CCD multiple times. This is sometimes preferred, for instance when studying shock propagation. In general however, this is not acceptable. A second alternative is to have multiple CCD's installed in a given camera. These more costly cameras allow for unsteady events to be captured in much fewer tunnel runs. Multi-channel cameras are available in stock configurations with between two and sixteen channels.

3.4 Current Interferometer Hardware

To improve the interferometer setup, several key pieces of the apparatus were replaced with new components. One problem with the old setup was the lighting system. The old laser was a 15mW HeNe laser. Because of the laser's age its actual output was probably well below 15mW. This low power level caused problems with high speed triggering, because all available cameras, with the exception of dual-intensified cameras, required more light than the laser could generate. To be able to see passing shockwaves in our current setup, a shutter rate of 300ns is required.

When selecting a new laser, besides increasing the power, the coherence length was an important factor. Several powerful CW lasers, such as the Copper-Vapor laser, do not have an adequate coherence length. Besides this fact, it was decided that a power level on the order of one watt would be acceptable for our shuttering purposes. This power can be accomplished with an Ar^+ laser. To reduce the acquisition cost of the laser, a used laser was purchased. The laser chosen was a Spectra-Physics Model 168 Ar^+ laser. This is a very popular, albeit old tube design.

The Model 168 Ar^+ laser has a maximum output of 3W; however, when in single line mode the maximum power is approximately 1.5W for the 514nm line and

1W for the 488nm line. By adjusting the current going into the tube, is possible to run the laser at output powers as low as 30mW. When running the laser at high power levels there can be coherence length problems. To mitigate this problem, an optional etalon was purchased. The etalon is used to stabilize the light beam and increase the coherence length. The etalon in the 168 system reduces power by about 50%.

Figure 3.2: Photograph of Imacon 468 Camera System

Along with a new laser, a new digital camera was purchased. It was decided that for efficiency reasons a multi-channel camera was required. After researching several designs, an Imacon 468 camera was finally purchased. The Imacon 468 is an intensified high speed digital camera(see Figure 3.2). The model is available in 4, 6 and 8 channel versions–all upgradable to 8 channels. The CCD's are arranged around a central prism, as shown in Figure 3.3. Each CCD is an 8-bit 576x385 array. The intensifier has a variable gain from 0 to 7000 times. The resolution of the intensifier is

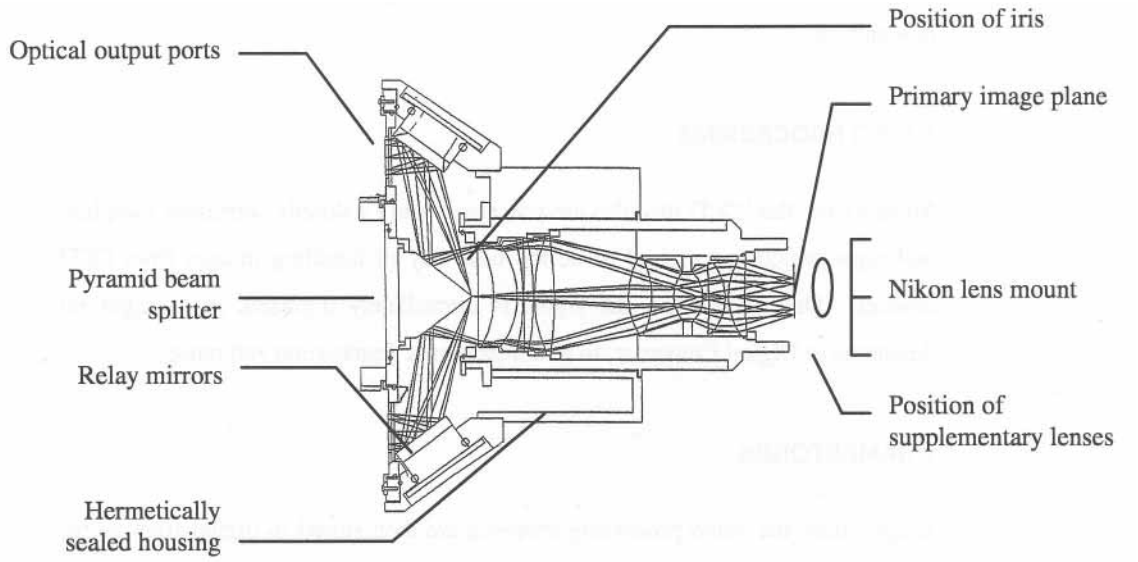

Figure 3.3: Illustration of the Imacon 468 CCD Camera prism assembly

22 line pairs per millimeter. This degrades when shuttering above one million frames per second and when there is a high gain. When operating at these conditions, the resolution drops to 15 line pairs per millimeter.

The control of the camera is done through a software package which runs under *Windows*. This software controls both the firing sequence for the camera as well as image loading and storing. The software allows for manual triggering and activation by a trigger signal which is received by the camera. It is possible to control the precise time from the trigger that the photograph is taken. Each CCD can have different shutter speeds and different numbers of exposures during a sequence.

3.4.1 Optical Components

Besides these main components several new pieces of optics hardware were installed to allow for a more accurate and efficient interferometer setup. In the new system, all optical components are mounted on a 3ft x 3ft optical table purchased from Melles-Griot. This table is used to provide for accurate placement and angular movements relative to the test section. The ideal interferometer system has the wedge plate at a 45° angle with the light beam. Because the light is perpendicular to the test section it is possible by aligning the optical table such that it is parallel with the test section,

to easily create a 45° angle with the wedge plate. The table is mounted on a custom optical stand. The height of the stand can be adjusted by a set of bolts which allow for the raising and lowering of the stand.

The wedge plate used for this experiment was the original wedge plate from previous years. This wedge plate has a six inch diameter and a 0.23 inch thickness. The wedge angle has been calculated to be 15 arcseconds. This gives an image separation distance of 0.21 inches and a fringe spacing of 0.09in.

To mount the wedge plate to the table, a custom mount made of aluminum was manufactured. The hole spacing on the table is one inch, therefore the length of the piece and the hole spacing was made such that the wedge plate can only be mounted at a 45° angle on the table. The original wedge plate mount was modified so that it could be joined to the new piece.

According to the equations, the wedge angle vector needs to be parallel to the 45° turning angle. To determine when this is the case, the wedge plate was loosely mounted in the optical system. A plate was placed directly following the wedge plate. As the plate rotates, the fringe lines will change their orientation. Once the fringe lines are vertical, the wedge angle vector will be in the same direction as the turning angle. In this setup, gradients in the horizontal direction can be measured. An illustration of the system can be seen in Figure 3.4.

To measure gradients in the vertical direction, the wedge angle and turning angle need to be reoriented by 90° . A second mounting piece was designed which allows for the wedge plate to be held at an exact 45° angle relative to the vertical. To get the wedge angle reoriented, the system wedge plate mount is loosened and the plate is turned until it is properly oriented.

After the wedge plate, the beam has a direction pointed up instead of horizontal. A flat mirror is therefore mounted on a rail system at a 45° angle relative to the vertical. This allows the beam to be redirected back towards the horizontal. An illustration of the vertical fringe assembly can be seen in Figure 3.5.

Along with better accuracy, the above system is much easier to use than previous systems. By having all the optics components placed in a 3ftx3ft grid, it is

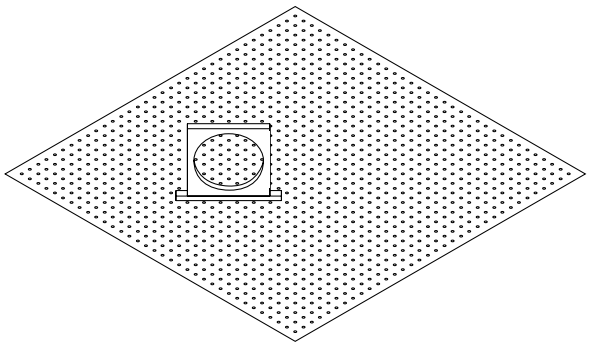

Figure 3.4: Illustration of optic table in position for a horizontal interferogram

possible to know exactly how much something is moved during a tiling exercise for studying a flowfield. This allows for an exact movement of the camera to adjust the magnificaton factor and focus when taking photographs of multiple areas of the flow.

Furthermore, because the wedge plate acts like an optical fold in the system, it is possible to replace it with a first surface flat mirror, without changing the optical system. Using the first surface mirror instead of the wedge plate turns this system into a shadowgraph/schlieren system. Using this arrangement, the flow visualization experiment can be changed between a shadowgraph/schlieren system and an interferometer in less than five minutes.

Changing from the horizontal to the vertical oriented interferometer is also relatively painless, because the exact coordinates of the original system are preserved. Usually changing to vertical fringes will require a slight adjustment of the optics following the wedge plate, however these are vernier changes.

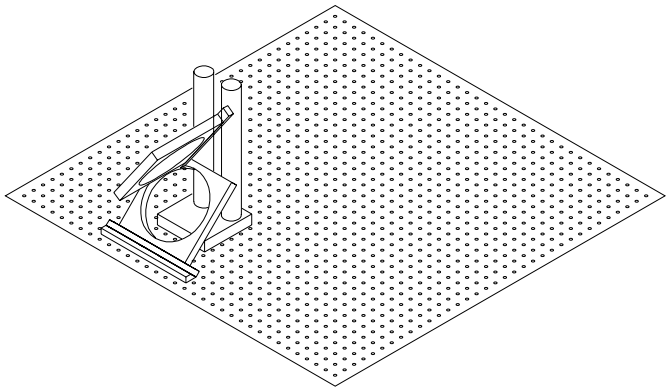

Figure 3.5: Illustration of optic table in position for a vertical interferogram

Optical Components		
Description	Manufacturer	Retail Price($)
Performance™ Optical Breadboard (3'x3')	Melles-Griot	1995
Performance™ Mirror Mount (6")	Melles-Griot	1000
6" Diameter Flat Mirror	Melles-Griot	1250
StableRod™ Mounting System	Melles-Griot	2000
6" f4 Parabolic Mirror	Edmund-Scientific	250

Table 3.2: Optical Prices

Chapter 4

Image Processing

The image processing procedure for single-plate interferograms used at Virginia Tech has remained unchanged since 1996. While the overall methodology has not changed, there have been some significant advances in the techniques and algorithms. These improvements have been implemented in the current image processing method.

All of the code has been translated into *Java* code to allow for cross platform abilities. Furthermore, *Java* allows for the direct manipulation of *JPEG* and *GIF* files, instead of dealing with only RAW files. RAW files are simply a stream of pixel information. In the case of grayscale images, this consists of a string of 8-bit values. There is no header to describe the image size or any other information. The size of these images is proportional to the area. The size and the lack of inherent image information is unacceptable.

With *GIF* and *JPEG* files, like all standard image formats, the image size and any other required processing information is included in the header files. Furthermore, both of these formats provide image compression. The compression ratio for these routines can be anywhere between 2:1 and 50:1, depending on the complexity of the image and the routine used. The standard *Java* API provides classes which allow for the manipulation of these files. This removes the need to write any code to process this information.

The sections below describe the algorithms or procedures that will be used to perform all the image processing to take an interferogram and turn it into a single

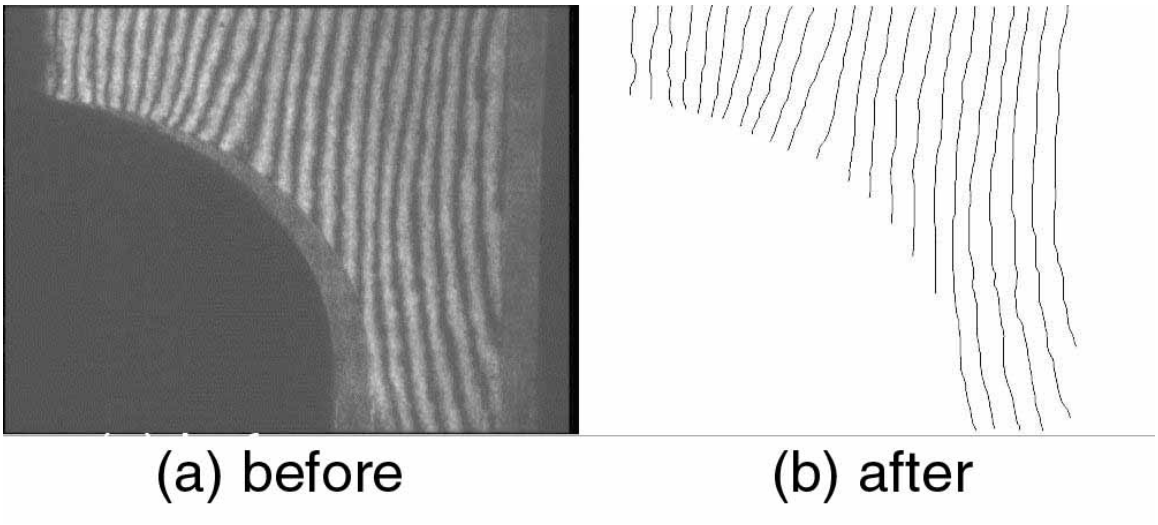

Figure 4.1: A before and after processing sequence

pixel representation. A sample before and after picture of these can be seen in Figure 4.1

4.1 Binarization

First, the image needs to be at a 1:1 ratio. This means that the physical scale of the image must be the same in both the x and y direction. This is sometimes not the case due to off-axis placement of optics in a given system. A simple image processing program can be used to stretch the image until it is at the proper aspect ratio.

Once the image is at the correct size, it must now be binarized. Binarizing is the process of converting the gray scale image into a black and white image. This does not use a dithering process. Instead it uses a variable threshold algorithm. Thresholding consists of studying a pixel's value, and then determining if the post-processed pixel should be black or white based on that value. In general, if the pixel value is higher than a given threshold, then it is white, otherwise it is black.

In adaptive binarization, the threshold value is calculated based on a local average of the surrounding pixels. An $N \times N$ area surrounding a given pixel is sampled. The threshold value will, therefore, be given by:

$$\text{Threshold Value} = \frac{\sum_i \sum_j f_{i,j}}{N^2} \tag{4.1}$$

where $f_{i,j}$ is the value of the pixel at coordinate (i,j).

In the original algorithm, at each pixel the sum was taken over the $N \times N$ surface. This means that each pixel is sampled N^2 times. Over an entire image, this proves to be cost prohibitive, as well as unnecessary. An improved version of this algorithm was designed.

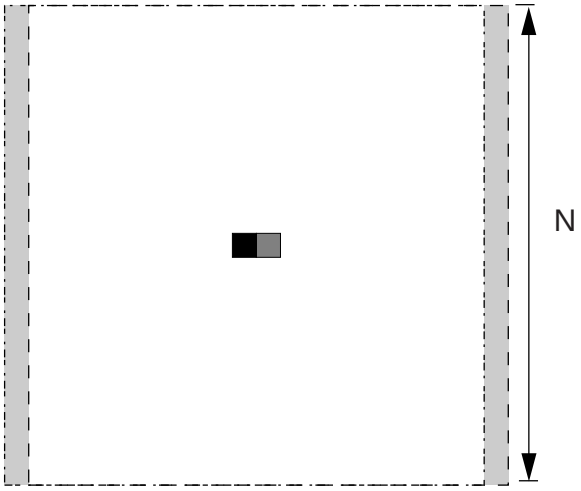

Figure 4.2: The graphical arrangement of two pixels being analyzed

From a graphical standpoint, consider binarizing two different pixels side-by-side as in Figure 4.2. The large squares surrounding each pixel represent the $N \times N$ sampled regions. The region which exists in both rectangles is white, while the regions that are not are shaded in. When studying the n^{th} pixel, the sum relative to the (n-1) pixel will simply have the values in the left hand strip subtracted and the pixels in the right hand strip added. Thus for the n^{th} step, the equation for the threshold sum is given by:

$$S_{i,j} = S_{i,j-1} - \sum_{i'=i-\frac{N}{2}}^{i+\frac{N}{2}} f_{i',(j-\frac{N}{2}-1)} + \sum_{i'=i-\frac{N}{2}}^{i+\frac{N}{2}} f_{i',(j+\frac{N}{2}+1)} \tag{4.2}$$

where $S_{i,j}$ is the sum at (i,j). S is stored from the previous step and then used in the calculation. Once S is calculated it can be divided by N^2. This value is then the threshold at the next step. This is the procedure for stepping in the j-direction, however, a similar process is used for calculating the sum used at the beginning of the next row. This procedure greatly decreases the number of operations that are performed in this part of the analysis.

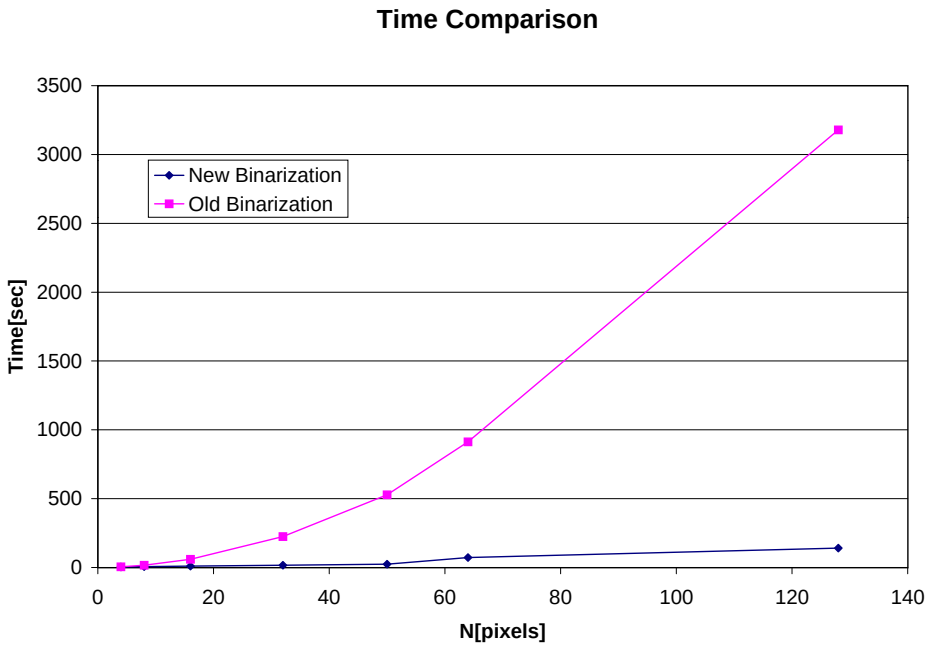

Figure 4.3: Comparison of old and new binarization algorithms

The increase in efficiency in this new method is readily apparent. In the old algorithm, the number of operations would go up exponentially as the size of N increased, while the new one only goes up geometrically. When working with large images and large sampling squares, this results in a large time savings. Figure 4.3 shows a series of timed binarization runs based on both the old and new algorithms. The image used was the standard horizontal interferogram, with a size of 1339x1000

pixels. The results show a significant increase in processing efficiency. For reference, the value of N that is optimum for this image is N=50.

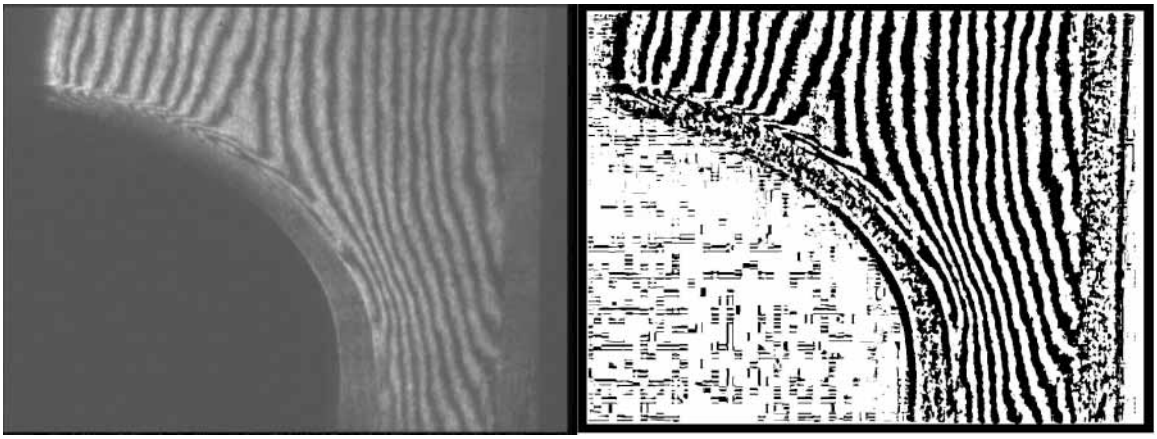

Figure 4.4: Comparison of an Interferogram before and after Binarization

A comparison of the image before and after binarization can be seen in Figure 4.4. In this image, it is clear that we have gone from a gray scale to a black and white image. This process is not perfect, and determining the correct sampling size can be a problem. If the size is either too large or too small, details can be washed out. This is apparent is some regions of this binarized image. These errors are corrected manually before processing at the next step.

4.2 Line Thinning

The final algorithm requires a one point wide line. It is clear from the above image, that the final results of our binarization is a line which is several pixels wide. This is unacceptable. Several algorithms exist for converting multi-point thick lines into single point lines.

To perform line thinning the popular Hilditch algorithm was used. The Hilditch algorithm starts with a black and white image of thick lines and then proceeds to analyze each black pixel by looking at its neighboring pixels. This can be visualized as a 3x3 grid of pixels. Based on some rules of thumb for patterns of this array, the pixel will either be deleted (turned white) or kept black. The details of this algorithm

and other line thinning methods can be found in Bush [1]

4.3 Manual Correction

The line thinning algorithms used do not produce perfect fringes. Often times there is a lot of manual correction to be done. In some cases, it is actually faster to create the final interferograms by hand. When the interferometer system was first designed this was a very laborious task which required special hardware. With modern image editing programs this is no longer an issue. Presented below is the method used for refining or creating the final interferograms using a Windows program called *Paint Shop Pro 4*. The same procedure can be done in many other editing programs.

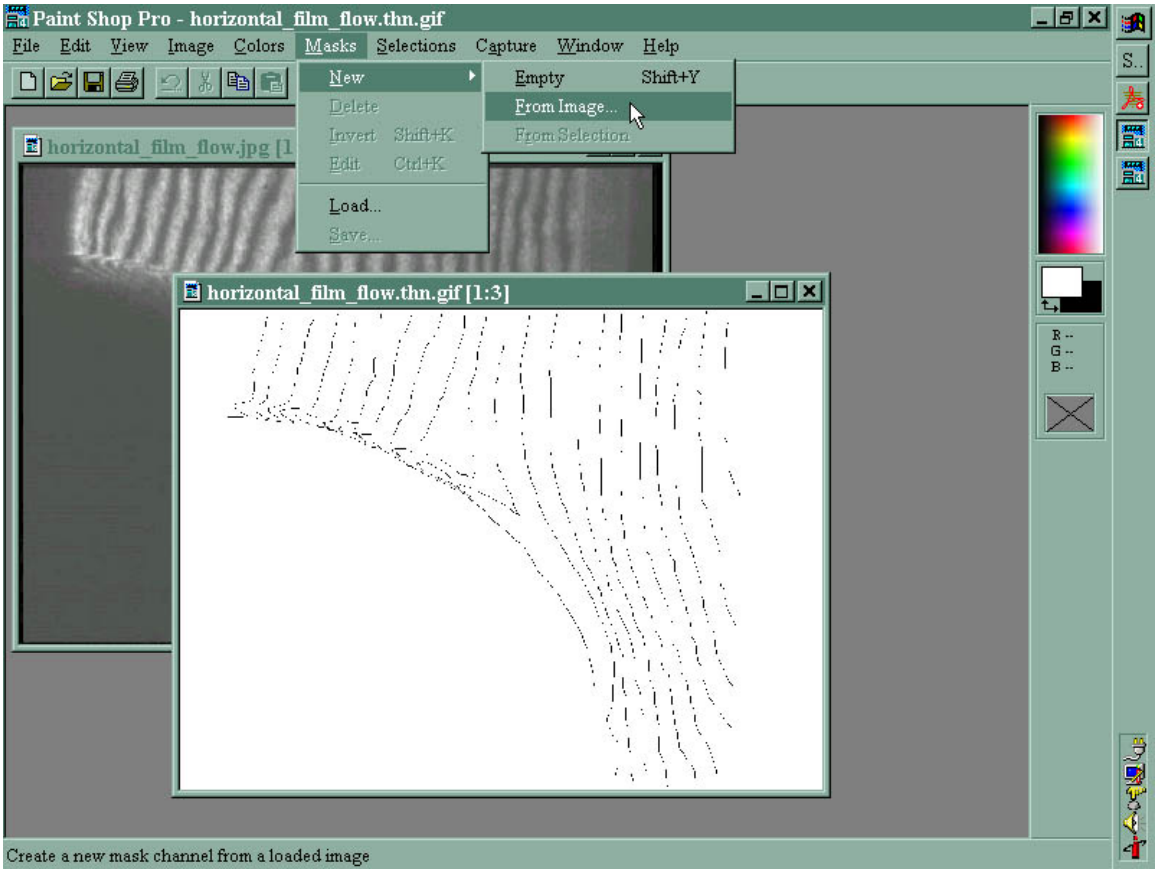

Figure 4.5: Screenshot showing how to select the mask creation options

The first step is to load the original interferogram and the thinned or new image. If a new image is being created, it should be an 8-bit grayscale image of the

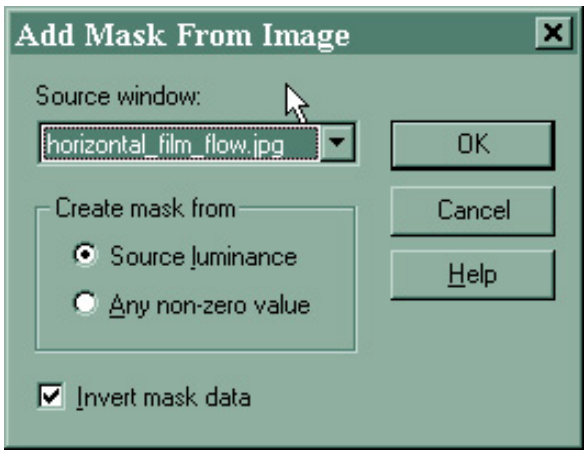

Figure 4.6: Window to select masks

exact same dimensions as the original interferogram. A mask of the original picture needs to made in the image that is being corrected. This mask will be used to trace out the fringe lines . A picture of the screen shot of the program can be seen in Figure 4.5. In the mask menu, select the option to create the mask from the original image, see Figure 4.6. If you want to trace out the dark fringes, it is better to invert the image, so that the dark images appear bright in the mask. To view through the mask, simply enter $< ctrl - alt - v >$. A final screenshot of the two images can be seen in Figure 4.7.

Tracing the lines out can be done with the paint tool, with the color set to total black (255). Once the editing is done, the mask should be deleted. Following this, the image should be reduced to a 2-bit image and then reset to an 8-bit image. It can now be saved as a *GIF* image. As stated previously, it is better to store the final image in a *GIF* image, because it uses a lossless compression. Slight errors can be introduced by saving it in a *JPEG* image, which is a lossy compression.

4.4 Alternative Methods

The image processing system still leaves much to be desired. While the automatic programs streamline the process, it would be ideal for the system to be completely autonomous. Before this can be done however, a new reduction process will have to

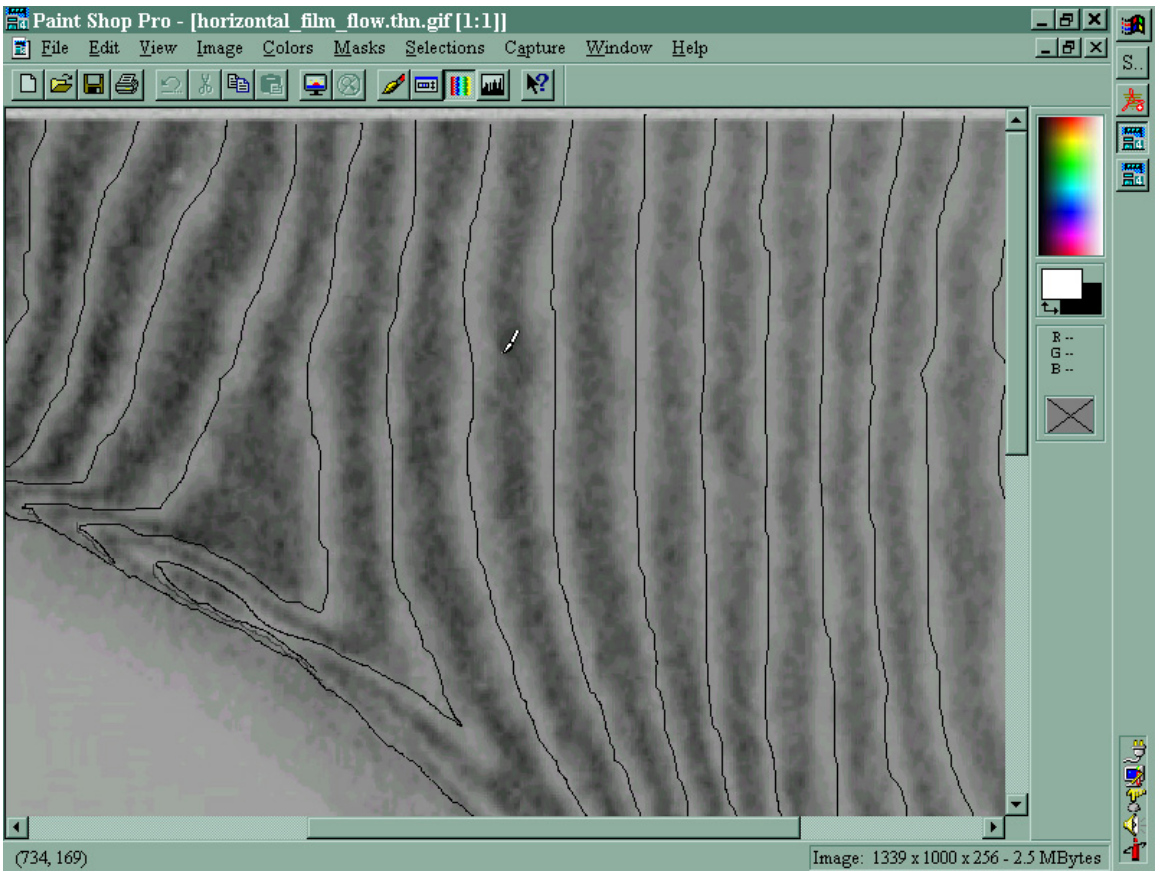

Figure 4.7: Viewing the final image through the inverted mask

be developed.

One alternative to the binarization-thinning method used here is to use a more advanced line detection pattern recognition algorithm. Most of these methods will work directly with a grayscale image. Some of them are based on a topographic analysis of the color values [1]. The bottom line is that much more advanced image processing methods will be required. This will only become significant if large numbers of interferograms is required to be analyzed.

Chapter 5

Interferogram Processing

Once all the image files are prepared, and the known density field has been sampled and interpolated, it is possible to calculate the density for the entire flowfield. A discussion of the hardware of this system is provided in Section 2.3. Here is presented solely the data reduction procedure. The basic premise behind the data reduction is that if you know three points in the field, it is possible to calculate the fourth point using the equation [4]:

$$(\rho_{A_1} - \rho_{B_1}) - (\rho_{A_2} - \rho_{B_2}) = \pm\frac{\lambda_0}{K}\left(f_{AB_f} - f_{AB_n}\right) \tag{5.1}$$

where λ_0 is the wavelength of the laser light, f_{AB_f} and f_{AB_n} are the number of fringes between points A and B in the flow and no-flow interferograms, and K is:

$$K = \frac{Lk}{n_0\rho_{ref}} \tag{5.2}$$

where L is the test section span in meters, n_0 is the index of refraction for air at STP, 1.0002506, ρ_{ref} is density at STP, $1.225\frac{kg}{m^3}$, and k is an optical proportionality constant which is a function of wavelength. The value of k varies between 0.000290 and 0.000298 for visible light. The location of the four points (A_1,A_2,B_1,B_2) changes depending on the direction which is being traversed. They are illustrated in detail later.

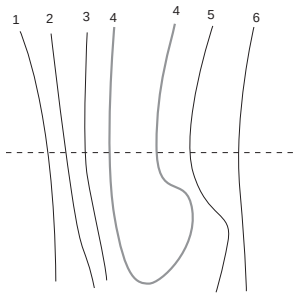

Figure 5.1: Schematic of a problematic fringe formation

5.1 Line Following

In the original program, the fringe function was calculated by simply counting across a horizontal or vertical line, depending on the fringe orientation and assigning each new black pixel a new integer value. The pixels between two fringe lines was then linearly interpolated. Unfortunately, this is not correct in general. If fringes are formed in a manner such that a given fringe line crosses the counting path multiple times, it is necessary for the program to know that.

An example of this problem can be seen in Figure 5.1.The numbers at the top correspond to the correct fringe count for this set of hypothetical fringes. In the original program, the algorithm would have thought that the second passing of line four was a new fringe line. The resulting analysis would, therefore, be wrong. To counter this in the original program, the fringe count was manually corrected for each interferogram. This represented the bulk of the time for processing the interferograms. A new line counting algorithm was, therefore, developed.

The basic idea behind the new algorithm is the assumption that the image is a 1-bit image, that the lines that are to be followed are black and a single point wide. Furthermore, lines cannot cross paths or intersect. In a given step, a 3x3 grid centered on a black pixel is analyzed, as shown in Figure 5.2. If more than three pixels in a particular corner are black, then this is not part of a line but a region, therefore it is ignored. If no pixels are black, then this is a stray point, and it is also ignored. If on the other hand, there are other black pixels, it is part of a line. Because we are dealing with single point lines and only adjacent pixels, it is not possible for more

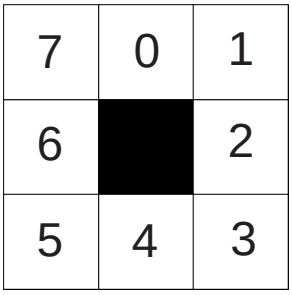

Figure 5.2: Surrounding grid of adjacent pixels

than one line to exist in this grid. Therefore, if any of the points are already labeled, then all the points in the grid will also be labeled, including the centered pixel. If none of the points are labeled, then a new label is created for the center pixel, and all pixels in the grid are given the same label.

For the actual algorithm, to insure that all points in a line are given the same label, points in a grid are followed. So for example, if pixel 2 in Figure 5.2 were black, then it would be given the same label as the center pixel. The algorithm would then apply the analysis to pixel 2 with a 3x3 grid centered around this pixel. In this way, lines are traced out throughout the picture.

The actual processing for this is encapsulated in its own static class, *linepick*. Before processing, the image is converted into a 2D array of 16-bit characters. The character index numbers go from -32767 to 32767, with a given index corresponding to a unicode character. The internal convention is presented in Table 5.1. The resulting array is then processed further by an interferogram object, as explained in the next section.

Index	Description)
0	Default Value
1	Black
2	White
>2	Flags

Table 5.1: Index convention used in class *linepick*

5.2 Interferogram Object Structure

Although the line following is taken care of by a separate class, by far the most important class is *interferogram*. This class encapsulates all the necessary routines to load, initialize and process an interferogram within the program. For the programmer, the creation of an *interferogram* object is as simple as loading an image. As in good OOP, all details are handled behind the scenes. The development of this object is out of logic and necessity. An *interferogram* has a physical counterpart in the real interferogram. Furthermore, after carefully looking at what needs to be processed and stored, it became apparent that there was great room for improvement over the original way of storing interferograms (original refers to the program generated *circa* 1996/1997).

In the original program, a raw image was read from a file. This image was then sent through a series of processing steps to generate a field of fringe counts. Once counting of the individual fringes was done, the entire flowfield was interpolated. This information was then stored in a 2D array of double precision numbers for the duration of the program's execution. It is clear that as image size increases, the necessary memory will grow too quickly. For example, in the 1996-1997 tests, the image sizes were 512x480. Assuming one wanted to sample only one set of pictures, the total memory requirement, including an equally sized double-precision array to store the density computations, would require 5.6MB. In the current analysis, the image sizes are now 1339x1000. If one again were performing only the most basic analysis, the memory requirement would be 30.65MB. Because one would want to step both up as well as across the flowfield,the minimum memory requirement for this program is therefore 51.1MB–just to store the interferograms. This quickly becomes unacceptable, even for today's average PC.

A quick look at what is actually being stored shows a place for a large improvement in efficiency. In fact, a given reduced interferogram is mostly white space. Only a small fraction of the total number of pixels will actually have an integer value. Most of the field is really interpolated values. It is, therefore, pointless to store anything

but the actual fringes and their corresponding values.

Inside the interferogram object are two 2D arrays: *fringes* and *coords*. For each line that is counted across or down, depending on orientation, the given array will store the fringe number and the coordinate along that line where that fringe is found. In this way, the memory requirements are minimized. The *interferogram* object keeps track of whether it is in a horizontal or vertical orientation, and it counts along the appropriate direction. Both of these arrays store the count as a 16-bit integer. This gives a maximum of 32767 fringes and by extension a maximum image size in the normal direction of at least 65536 pixels. This is more than adequate for any projected image size.

The memory savings by storing the fringe information in this way is enormous. For example, with our graphic files the standard image is approximately 1339x1000 pixels. To step left and right only would require at least two images. With our images, the original data structure size would have been 20.43MB. With the new data structure the size is 0.095MB. This is a 216 times saving in storage space. To study the entire area in our test case would have require four sets of interferograms, with each set comprising four interferograms. The memory footprint for these data structures would have been 163MB. With the new data structure the memory footprint would be approximately 0.76MB. The implications for speed and resource requirements is apparent.

The interferogram object loads a grayscale image in either *GIF* or *JPEG* format into this object in a multi-step method. These call various internal methods and external methods, and a brief overview is given here. Figure 5.3 shows an illustration of this process. First the interferogram object constructor reads in the appropriate *GIF* or *JPEG* image, which was specified in it's argument list as a *String*. Once loaded, the image is then converted to a character array using *linepick*'s *convertToCharMap* method. This information is then checked and sent to the *analyze* method in *linepick*. Inside of this method, each fringe line is assigned a label.

The resultant character array from *analyze* is then processed into an actual fringe count. Counting along a given line, the program assigns each new fringe pixel

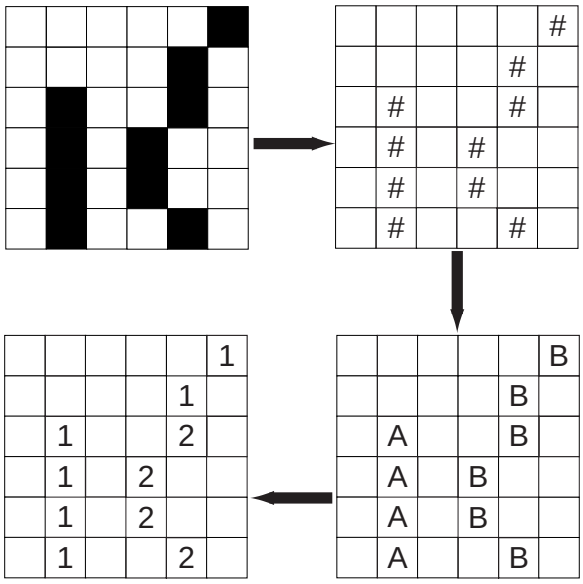

Figure 5.3: Illustration of the various stepping procedures

it crosses a new number. If a fringe is traversed more than once, it is assigned the previously assigned value. Once the entire line is scanned, the given coordinate and pixel count are stored in the corresponding row of the storage matrix.

To get the fringe count for a given pixel in the interferogram, the programmer calls the method *getCount*. This routine finds which two fringes the requested pixel is between and then does an on-the-fly linear interpolation between these two points. This value is then passed back through the function header as a double-precision floating point number.

5.3 Stepping Routines

The stepping routines all lie in the class *interproc*. This class also has all other necessary processing functions. All functions are described in the API reference in the Appendix. The material below just describes the coordinate orientations of the various stepping methods.

All of the methods use Equation 5.1 to calculate the unknown point. The relevant problem is how all four points are located relative to one another and the unknown field. This is represented graphically below in Figures 5.4 through 5.7.

In each picture, the four points which correspond to the points in the formula are explicitly labeled. The gray region represents the area for which densities are already known. This area is the known density field for a given step. The point labeled *origin* represents the starting corner used in sampling the density field. This is the point which is specified in the initialization files for a given step. The initialization file structure is discussed in detail below.

As of this writing, it is only possible to step in cardinal directions. Originally, it was hoped that a general stepping algorithm could be generated, however such an algorithm added unnecessary interpolations for our purposes. It could be developed in the future. All four routines take the same arguments and have meaningful names.

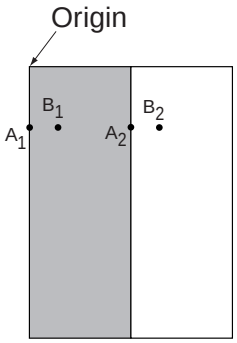

Figure 5.4: Coordinate convention used for stepping right, *HStudyStepRight*

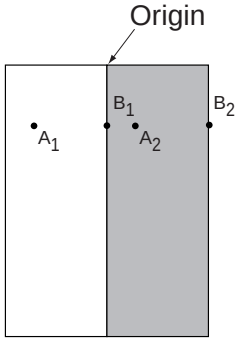

Figure 5.5: Coordinate convention used for stepping left, *HStudyStepLeft*

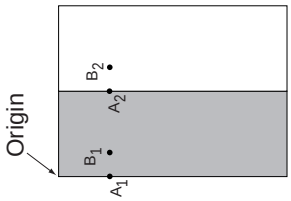

Figure 5.6: Coordinate convention used for stepping up, *VStudyStepUp*

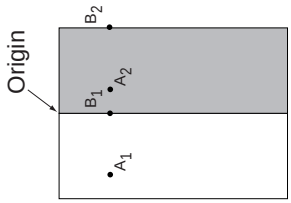

Figure 5.7: Coordinate convention used for stepping down, *VStudyStepDown*

5.4 Initialization File Structure

The program is modified for individual jobs via two different $*.ini$ files. These files can be given any name that the operator desires, but here they will be referred to as *loader.ini* and *runner.ini*. The first file is a command line option. This file is required to begin the program, and it gives *jbips* the required information to build the data structures. *runner.ini* is used to tell the program how to step through the field and how large of a domain to compute.

The files themselves are standard *ASCII* files which correspond to the same format as a standard Windows $*.ini$ file. An example snippet from a generic file:

```
...
[HEADER]
variable=value
....
```

The header is a title which divides major sections of the initialization file. The variable and value lines are self explanatory. The overall structure of both of the $*.ini$

files used in this program can be found in the Appendix. It should be remembered that these files are all case sensitive.

5.5 Program Operation

Once the initialization files are created, it is possible to run the final program. Here is a very elementary means of starting the code. A more in depth discussion of starting *Java* programs is provided in the Appendix. To start the program, go to the directory with the class files and type:

```
java jbips loader.ini
```

with the name *loader.ini* replaced by whatever the actual file is named. This will launch the *Java* program and begin initialization based on the information in *loader.ini*. As the data is loading, a control window will popup. A picture of this window can be seen in Figure 5.8. The text line at the top is used to enter the filename which corresponds to *runner.ini* for the actual configuration. It is further possible to enter a default Mach number here. The *Run* button is used to start the analysis. The various checkboxes are used to control the output from the program.

If *View Contour Plots* is checked, then the program will display the contour plots on screen. There will be one contour plot for the density field and one for the Mach number field. The next option can be used to output these pictures to a file. To access the actual density and Mach number values from within other programs, it is possible to output each of these files. These files are *ASCII* files conforming to the *Tecplot* file structure.

The final option can be used if the known field is in a uniform region. This will substitute any known field stored on disk with an average based on either the average entered in this window or the average entered in the *runner.ini* file.

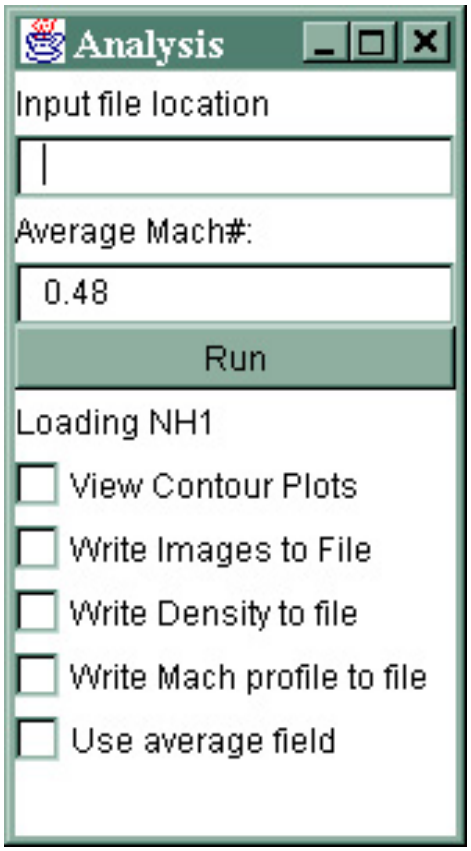

Figure 5.8: Screenshot of the *jbips* control panel

Chapter 6

Test Case

The world of turbomachinery is constantly attempting to push the thermal limits of the engine materials. By increasing the maximum temperature of the engine, designers are increasing the efficiency of the thermodynamic cycle. Without large advances in material science, the material limits have been held somewhat constant over the last several years. One method of increasing the maximum temperature of the engines is to inject a thin layer of cool air over the blade surface. Because of the complexity of the flowfield, it has been difficult for designers to accurately describe the flow. This makes computer modeling with modern computer resources problematic. Empirical studies of the flowfield are, therefore, often conducted in cascade wind tunnels. These studies attempt to determine the effects of shocks, wakes and freestream turbulence on the effectiveness of the film layer. From this information, designers have a base point from which to improve designs.

Using the single-plate interferometer, it is possible to study the density distribution over the entire flowfield, including through the film layer. In this manner, it is possible to correlate flow features with data obtained by surface measurements, and it is also possible to use this system as a means of validating CFD codes. This method also permits whole field measurements in highly unsteady flows.

Although such studies are possible, near the surface fringe information is difficult to interpret. One problem is the double image. The surface being studied can be partially obscured by the double image. Furthermore, in areas of overlap, the

actual surface location is not crisp. This is an inherent deficiency in the single-plate interferometer. Fringe counting near the surface can also be a problem. Some surface information is still available however.

Use of a Mach-Zehnder interferometer would actually be more advantageous for studying flow right at the surface. The Mach-Zehnder interferometer would certainly remove the above problems which become critical near the surface. The overall added complexity of the Mach-Zehnder system still makes it unrealistic for tests in our tunnel environment.

Here is presented the experiment apparatus and background information on the facility. Along with this information, the background program initialization and results are presented.

6.1 Testing Apparatus

6.1.1 Tunnel Design

The facility used for this experiment is the Virginia Tech Transonic Cascade Wind Tunnel (see Figure 6.1). This tunnel is a blow-down type facility run via charged reservoir tanks and an atmosphere exhaust. The tanks are charged by a four stage, Ingersoll-Rand reciprocating compressor. Air from the compressor is passed through a series of driers to reduce the humidity of the stored air. With the current system, the tanks can be charged up to 600psia. For the current run conditions, the ideal tank pressure is 180psia. This provides testing times of up to 20 seconds.

The air from the tank passes through two butterfly valves before entering the wind tunnel. Both of these valves are pnuematically actuated. The first valve is used as an on-off valve. This valve is controlled by a hardwired circuit. This circuit has several safety switches as well as a main, on-off switch, which prevents the tunnel from operating at a dangerous state. The power-off position of this valve is the closed position, so that in the event of a power failure, the tunnel will shut down.

The second valve is identical to the first valve. This valve is used to control

the tunnel stagnation pressure. This is accomplished by varying the opening of the valve to create an adequate pressure drop to achieve the desired stagnation pressure. For these experiments, a stagnation pressure of 23psia is ideal.

The correct opening angle for the flow control valve varies through the run as the tank pressure falls. To generate a step function pressure profile, it would be ideal to have a feedback control system. Unfortunately, the actuators have response times of approximately 1Hz. This makes the design of an adequate feedback system impossible. The tunnel control is, therefore, done through a feed-forward system. This is especially important during the tunnel starting procedure, when the pressure quickly rises from atmospheric pressure to the testing pressures. Once a level pressure has been reached, however, a minor feedback variable allows for small corrections to the valve opening. This provides an adequate pressure profile.

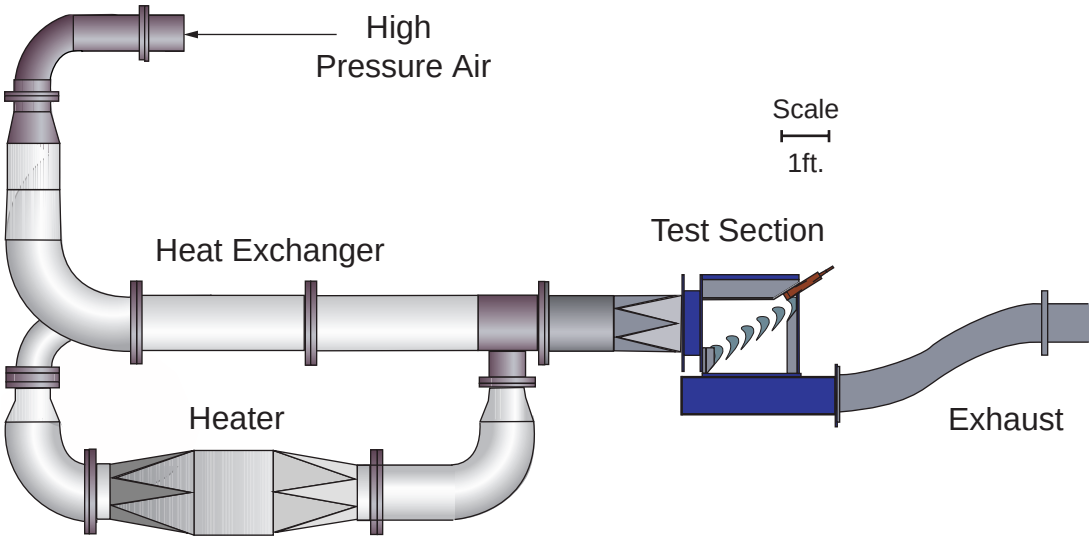

Figure 6.1: Schematic of the Virginia Tech Cascade Wind Tunnel

Heating of the freestream flow is accomplished through a heat exchange loop ahead of the test section. This can be seen in Figure 6.1. The heat exchanger consists of two bundles of copper tubes. Each bundle has 350 copper tubes with a 1.59cm diameter. These tubes are heated with a 36kW electric heater located beneath the heat exchanger. During heating operations, two flapper valves, one at each junction to the tunnel, are closed. This provides a closed loop through which the trapped

air is slowly circulated through the system. To begin a tunnel run, the two valves are again opened, and the flow has a direct line through to the test section. This is illustrated in Figure 6.2. Because activation of the tunnel with the valves closed would cause significant damage, the tunnel control circuit will not activate the on-off valve in this orientation. This system provides enough heating to allow for a peak stagnation temperature of 100°C .

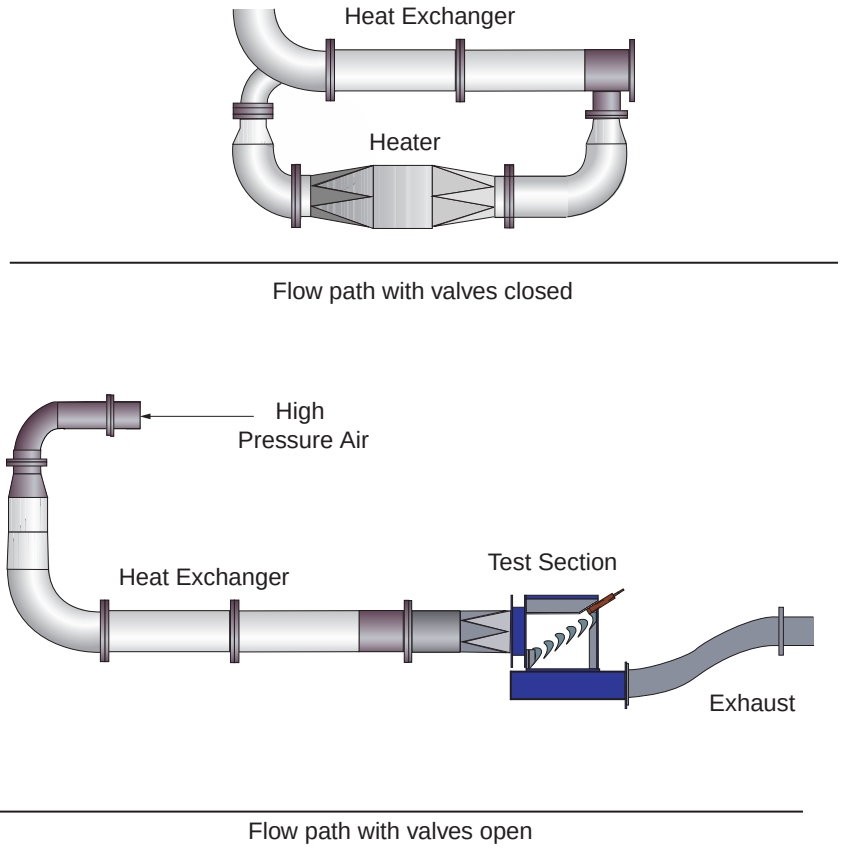

Figure 6.2: Schematic of the two flow conditions provided by the flapper valves

6.1.2 Cascade Test Section Design

The cascade test section used in this experiment consists of four full blades and two half blades. This provides for five passages. At test conditions, the upstream Mach number is approximately 0.36, and the exit Mach number is approximately 1.2.

The original endwalls were one inch thick optical grade Plexiglas. While this

provides adequate structural support and a clear viewing surface for shadowgraphs, they are unacceptable for interferometry. This will be explained in detail in later chapters. For the interferometer experiments, additional end walls made of aluminum were constructed. The aluminum endwalls have the same thickness as the Plexiglas endwalls, but have a window cut-out for flow visualization. The endwalls can be seen in Figures 6.3 and 6.4.

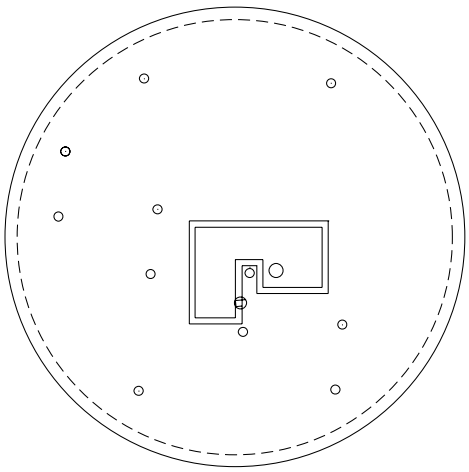

Figure 6.3: Schematic of aluminum endwall. Of note is the window placement

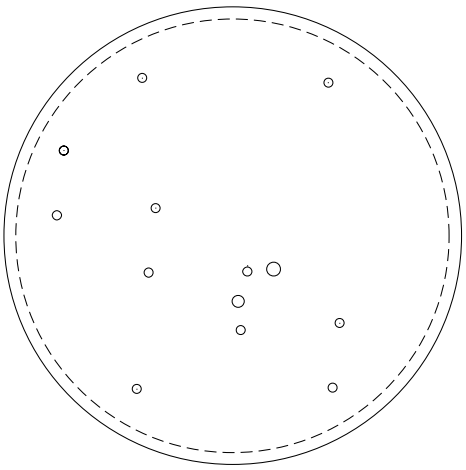

Figure 6.4: Plexiglas endwall

Making the windows out of Plexiglas proved to be very controversial. Based on the standard rule of thumb, crown glass with a $\frac{\lambda}{4}$ flatness should be used for

interferometry. During some initial bench testing, it was shown that reasonable fringe formations could be attained with optical grade Plexiglas. In order to more easily view the flowfield, this was chosen. By using Plexiglas, the problems inherent with machining glass were removed. One drawback to this decision has been that there is no way to tell the exact error introduced by the windows. Since the two images are subtracted, it was assumed that the resulting error would cancel and that as long as fringe distortion due to the Plexiglas is not severe enough to dominate the image these windows are adequate.

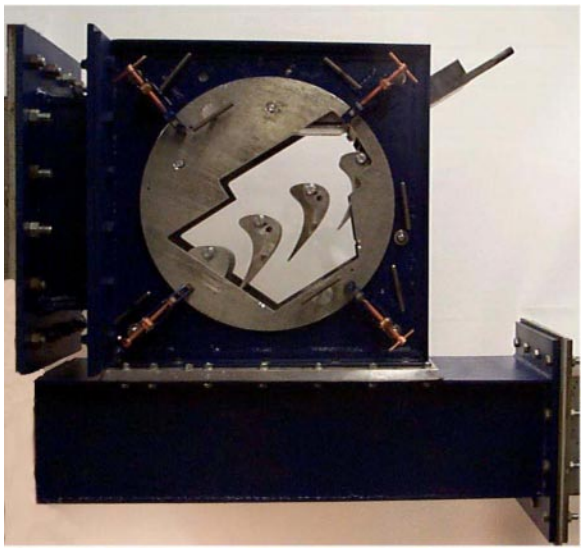

Figure 6.5: Photograph of test section

The blades are mounted in the endwall with a dowel pin and a shoulder bolt. These have been designed to carry the load generated by the lifting force on the blade during the run. The shoulder bolts also act to hold the endwalls together, during the run. The primary means of holding the endwalls together however, are a series of clamps lining the perimeter of the endwalls. These are welded to the outer wall of the wind tunnel. They can be seen in Figure 6.5.

6.1.3 Blade Design

The blade design used in this experiment represents a next generation first stage turbine blade. The geometry is provided by General Electric. A side view of the blade can be seen in Figure 6.6. The blade has a 4.5 in axial chord, a 5.31 in aerodynamic chord. The blade has a span of 6 inches.Under chocked flow conditions, the design upstream Mach number for this blade is 0.36. The flow is turned 127° and exits at Mach 1.2.

Because of the size of the blade, instrumentation requirements demand that the blade be partitioned. This allows for gauges to be installed without damaging the surfaces exposed to flow. The two pieces can be seen in Figure 6.6. The pieces are held together by a bolt. The alignment of the pieces is accomplished with a dowel pin which holds the upper piece at the correct orientation relative to the lower piece. The gap near the surface is sealed with silicon glue. For a more detailed description of the blade design see Popp[6].

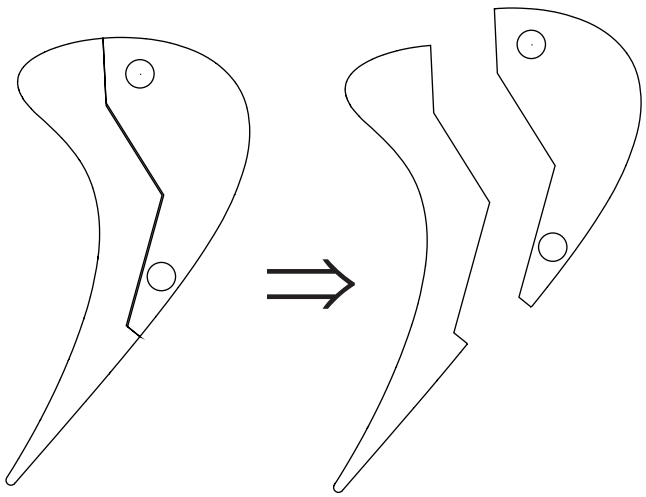

Figure 6.6: Model blades

6.1.4 Cooling Design

This experiment's primary focus is on the shower head film cooling which is typical in the first stage turbine. The film cooling consists of six rows of film holes. Each

hole has a nominal diameter of 0.041 inches and are spaced 0.36 inches apart. All of the holes are fed from the same plenum. The plenum is insulated with a nylon sheath to minimize heat transfer between the blade and the cooling flow.

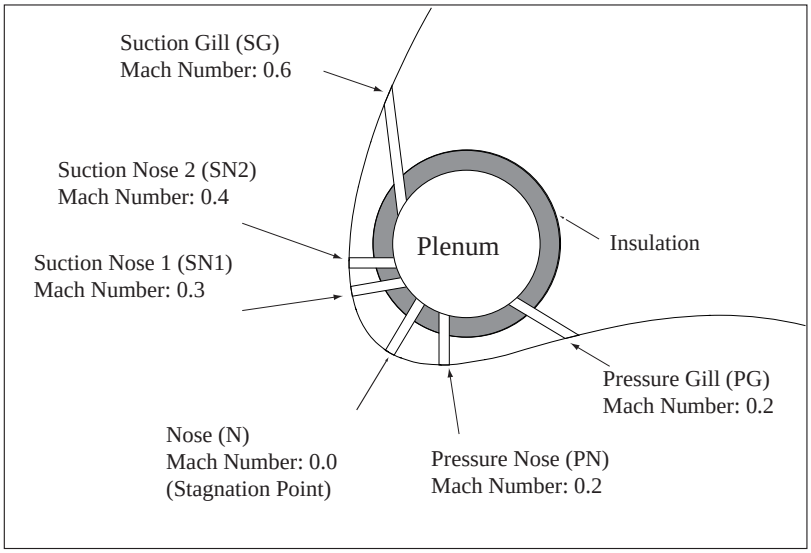

Figure 6.7: Illustration of the film cooling configuration

The cooling design is also supplied by General Electric. Figure 6.7 shows the hole labeling and the surface Mach number at that location. Each row of holes has a thermocouple and pressure transducer mounted in one of the far holes. This allows for an accurate measure of the local static pressure and temperature. The pressure measurements show agreement between the predicted surface Mach number and the actual Mach Number.

There are four rows of nose holes, one row on the stagnation point, two rows on the suction surface and one row on the pressure surface. These holes are oriented normal to the surface, but inclined 30° in the spanwise direction, see Figure 6.8. There are two rows of gill holes, one on both sides of the blade. These are inclined 45° with the surface but have no spanwise angle, see Figures 6.8 and 6.9. This geometry corresponds with a typical engine configuration. For a detailed description of the cooling design see Popp[6].

In order to ensure that an adequate supply of air is available for the film cooling, a coolant supply system was designed. A schematic of this system can be

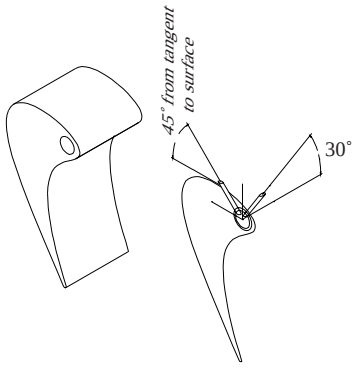

Figure 6.8: Cooling Hole Orientation Relative to Surface and Span

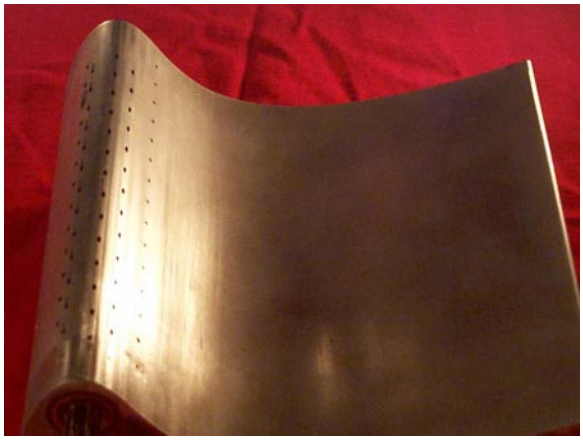

Figure 6.9: Photograph of the showerhead film cooling scheme

seen in Figure 6.10. The tank is supplied with compressed air from a two-stage Ingersoll-Rand compressor. The tank is pumped up to a pressure of 160 psi, which is controlled by a sensor on the compressor. A manually operated on-off valve is used to start the flow system. An integral feedback control valve is used to control the feed system stagnation pressure. The feedback system maintains a prescribed ratio of tunnel total pressure to supply total pressure. This is set manually and then checked during the run with a differential pressure transducer. To measure an exact mass flow rate, the system has a mass flow meter.

To maintain the coolant at a low temperature, a simple heat exchanger was built. The cooling line is coiled up and placed in a bath of liquid nitrogen which is held in a styrofoam container. By adjusting the height of the nitrogen we are able to

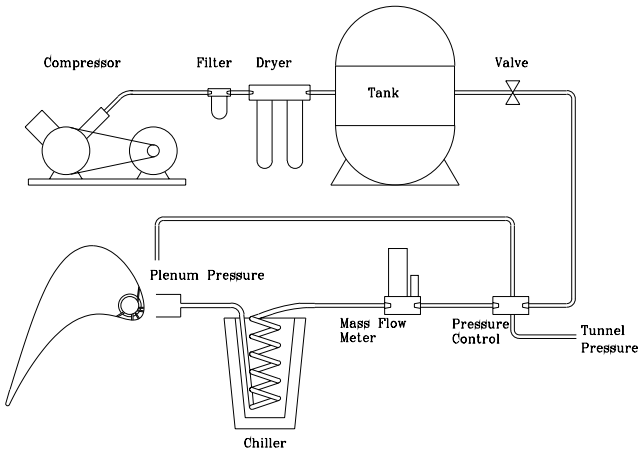

Figure 6.10: Diagram of the film cooling supply setup

control the coolant temperature, down to a temperature of -100°C .

6.1.5 Tunnel Validation

Validation of the tunnel flow conditions was performed to ensure that the correct flowfield is established in our test section. There are several particular issues which were crucial to cascade experiments.

The first issue was flow periodicity. In a cascade, it is important that all surrounding blades see a similar flow geometry. This accurately models the flowfield in an engine. This does not occur near the last and first row of holes, because of the influence of the tunnel walls. Far from these boundaries, however the flow should be periodic. This was checked with a wall oil flow visualization. The results are shown in Figure 6.11. The poor flow through the last passage is clearly seen as a large fan-tail like flow pattern on the window. Again, this is caused by the influence of the wall, since the last passage is made up of one blade and one half blade. The oil flow visualization also shows that the other passages are functioning normally. The flow around the two blades surrounding the film cooled blade are similar in profile.

The second issue was boundary layer interference. The tunnel walls also have a boundary layer. This boundary layer grows as it moves down the blade surface. If the boundary layer becomes too thick, there will not be a sufficient region of clean flow in which to take experiments. The boundary layer effect will also make the flow

Figure 6.11: Oil flow visualization of the endwall

three dimensional. Figure 6.12 shows the boundary layer growth on the blade surface. From these pictures it is clear that the boundary layer does not consume the entire blade surface. If testing is performed near the center of the span, then there will be little boundary layer effect from the endwalls.

For the film cooling experiments, the precise location of the stagnation point was also critical. Before manufacturing of the film holes, the location of the stagnation point was checked with the idealized inviscid profile, see Figure 6.13. This was done through oil flow visualization again. A line of red oil was placed in the stagnation region. At the stagnation point, a line will form, since there will be no flow through that line. This is the stagnation point. The results of that test can be see in Figure 6.14. The results correspond roughly with the predicted stagnation point.

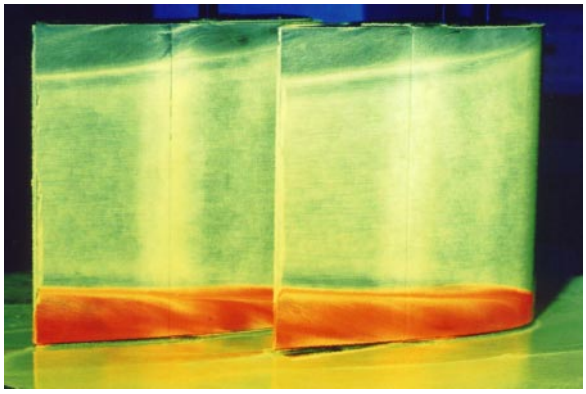

Figure 6.12: Oil flow visualization of the blade surface

6.2 Interferometer Program Initialization

The initialization files for the test program can be found in the Appendix. Here are presented the initial interferometer pictures and reference density field.

The reference density field was calculated by first measuring the static pressure on the tunnel wall. An illustration of the location of this field relative to the blade in question can be seen in Figure 6.15. The dimension of the box is 0.25 in x 0.5 in. The pressure taps are staggered; the center holes are offset 0.125 in from the other holes. During each run the entire field was sampled simultaneously using the PSI System. The upstream total temperature and pressure were also recorded. To create the reference density field, the upstream total density was calculated. This was related to the local Mach number using standard isentropic relations, and then the resulting field was loaded into *Tecplot* where a Krige interpolation on the points was performed. The interpolated field was then smoothed with a standard smoothing function to get rid of any artifacts from the interpolation, since there was a small degree of variability in the field. The final field can be seen in Figure 6.16. This field is placed in the location of the reference field in the given interferograms to begin the solution.

For this study, it was desired to analyze the whole field in a blade passage using both cardinal directions. Analyzing the region of the coolant film is particularly difficult with an interferometer, because of the high density gradients present due

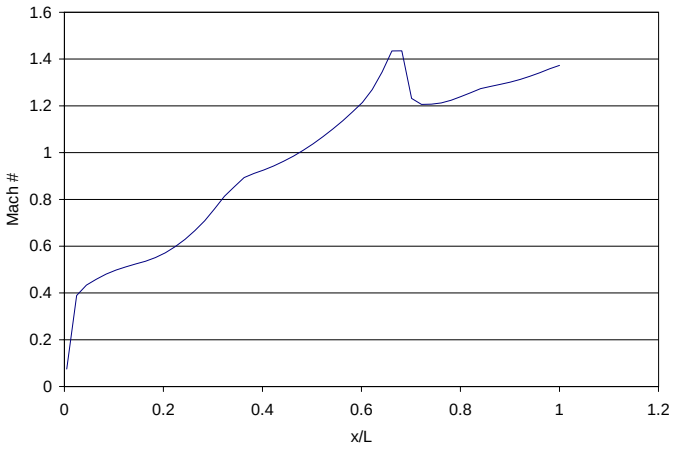

Figure 6.13: Ideal Suction Surface Mach Number Profile

to strong temperature changes. To reduce this problem, the tunnel was run with a temperature ratio (freestream stagnation temperature versus coolant stagnation temperature) lower than normal. The nominal value is something close to 2.0; in our case the tests were run with a ratio of close to 1.1. This change is not significant in regards to the main flowfield as shown in Popp [6]. Even with this less severe temperature change, measuring gradients normal to the surface will create fringe shifts which are too large. It is, therefore, necessary to step across the layer, which is a direction without such large density gradients. This needs to be done in different directions as one traverses the pressure surface of the blade, since the surface normal is not in a single general direction.

Figures 6.17 and 6.18 show the interferograms taken of the relevant portion of the flow field. There are two sets of interferograms–horizontal and vertical. The horizontal interferograms are evident by the fringes being vertical and the double image shift being along a horizontal line. These images measure the gradients in the horizontal direction. The opposite is true for the vertical interferograms. These interferograms can be seen in Figures 6.19 and 6.20.

The no-flow pictures are presented for a comparison to the pictures with an

Figure 6.14: Oil flow visualization showing the positioning of the stagnation point on the blades.

established flowfield. In the no-flow pictures, it is clear that there is a small amount of distortion to the fringes. These distortions are due to material stresses in the Plexiglas generated during manufacturing. These distortions have been minimized by selecting a large region of minimum fringe distortion to machine, and then following the machining procedures listed in previous chapters.

The flow pictures clearly show the established film layer flowing over the blade surface. This is represented by the shifted fringe pattern directly off the surface. The fringes in the inviscid region are also shifted slightly by the changing density gradients induced by the flow acceleration.

Uncertainty analysis for the single-plate interferometer for similar tests at comparable conditions in the same facility was preformed by Wesner [9]. In her analysis, the uncertainties in pressure measurement, fringe counting and frequency shifting in the laser were all taken into account using a standard jitter analysis. Her analysis also took into account errors due to uncertainty in her triggering device, since she was studying an unsteady flowfield with a passing shockwave present. From her analysis, she determined the uncertainty in the final density calculations to be $\pm 3.3\%$. Those results apply directly here.

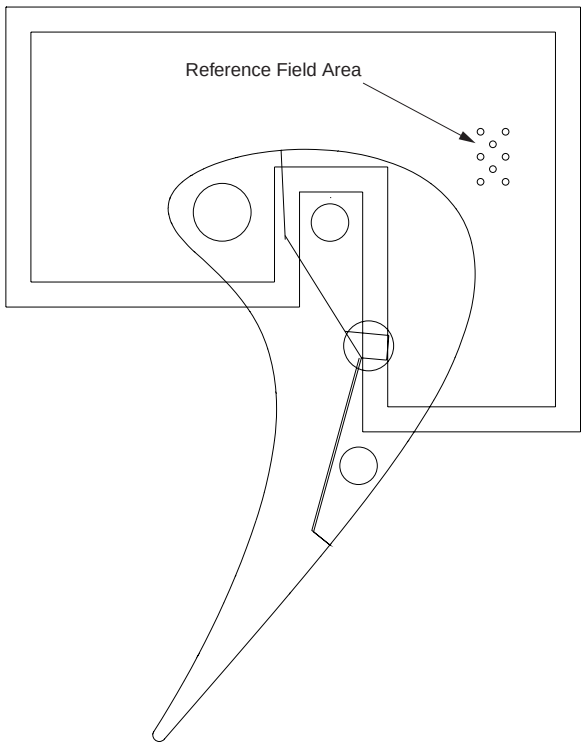

Figure 6.15: Illustration of the location of the reference field relative to the blade

6.3 Results

The reference field in Figure 6.15 was the starting point of the interferogram analysis. This field is approximately one image separation distance wide and two image separation distances high. The height was longer in order to capture the blade surface without stepping down, but also to make it small enough as to be reasonable to sample the pressures in the whole field at once.

This reference density field is not in the ideal location because of the relatively quick changes in density. These changes occur because of the acceleration of the flow through the blades. While the location of the reference field is not ideal for sampling purposes, it is ideal for studying the state of the film layer. The density gradient through the film layer is heavily affected by the temperature gradient through the layer. Since the fringe count is a function of the density gradient, fringe interpretation became a problem when measuring density changes normal to the surface. This is because the density changes very quickly. Resolving individual fringe lines in this type

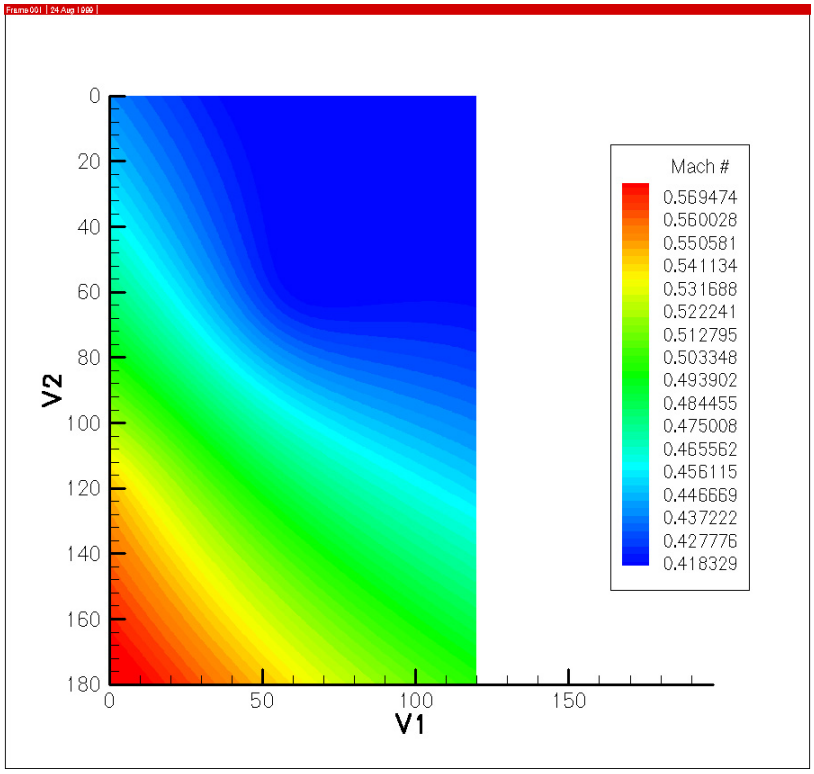

Figure 6.16: Reference Density Field

of field is difficult. To reduce this problem, the program steps through the boundary layer nearly parallel to the surface. By measuring gradients in this direction, the fringe count becomes more manageable, but still difficult to analyze for realistic temperature ratios.

Because of the curved contour of the blade, the surface normal varies considerably, relative to a fixed coordinate system. Consequently, both vertical and horizontal interferograms are necessary. At the point of peak curvature of the blade, it is apparent that for both the horizontal and vertical interferograms, the surface normal is nearly 45° relative to the direction in which the gradients are measured. While it would be ideal to step along this line, as well as the two cardinal directions, this ability does not exist in the current system.

The program used the following prescribed stepping procedure to fill the entire field. First, the field was filled along the horizontal direction. This took two steps to the right and eight steps to the left. A series of seven vertical steps were then taken

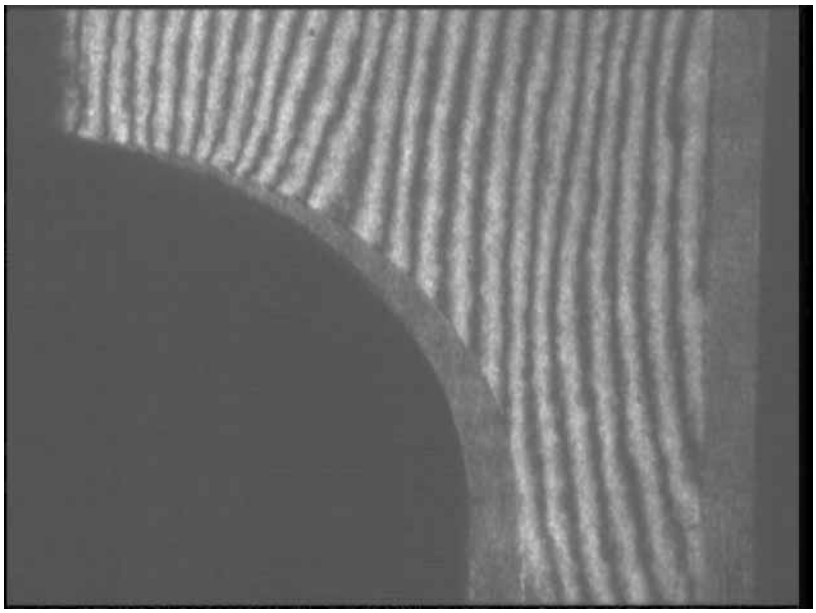

Figure 6.17: Horizontally oriented interferogram without flow

to fill the field down through the passage. The resultant field can be seen in Figure 6.21. In this image, we see the density contour plot from the calculation based on the interferograms. These density values represent the entire calculable region available with the presented interferograms. If it were necessary, the area encompassed by the entire window could have been sampled by tiling the interferograms. Since the region of concern on the blade is already in this image, this was not done.

To check the veracity of the calculation, the implied surface isentropic Mach number variation was compared to isentropic Mach number predictions developed by General Electric. The surface isentropic Mach number was found by converting the entire flowfield from a density distribution into a Mach number distribution using isentropic relations. These results are invalid in the boundary layer and they are therefore only used when outside of it. The surface isentropic Mach number at the edge of the film layer should correspond to the surface isentropic Mach number in the predictions. A plot comparing the two is presented in Figure 6.22. The solid line represents the numerical prediction provided by GE, and the squares represent the measured Mach number given by the interferometer. From this plot it is clear that there is a high degree of correlation between the predicted and actual values. A

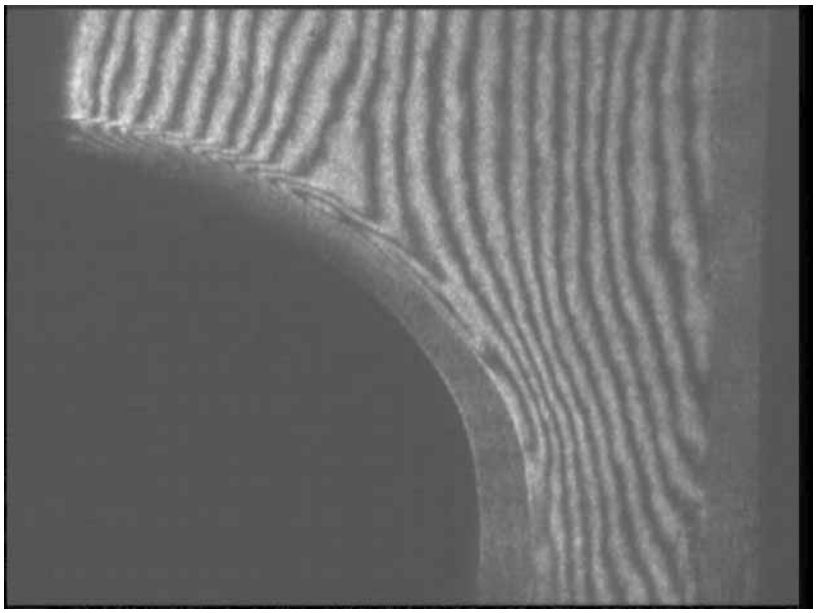

Figure 6.18: Horizontally oriented interferogram with flow

deviation in results begins to grow near the last sampled points in the interferogram results. Although it maintains the correct shape, the levels are further off than the other points.

It may seem surprising that the calculations of density for the throat region show a continued pattern of relatively high densities, which correspond to Mach numbers of approximately 0.5, a small distance from the surface as one proceeds across the passage. To determine the probability of this type of flowfield, a similar flowfield from a numerical prediction was studied. The numerical prediction was performed by GE for a similar profile. This prediction is presented in Figure 6.23. This plot presents the density distribution for the flowfield around a similar turbine blade geometry. The prediction provides some important qualitative information when compared to the measurements made with the interferometer. The most important aspect of both flowfields is the extension of high density values near the blade surface in the passage. But, since the Mach number along the suction surface quickly approaches the speed of sound, it may be expected that the all the flow near this point in the passage would also be following this pattern. Because the surface on the opposite side of the passage is at a relatively low Mach number, the Mach number varies quickly across

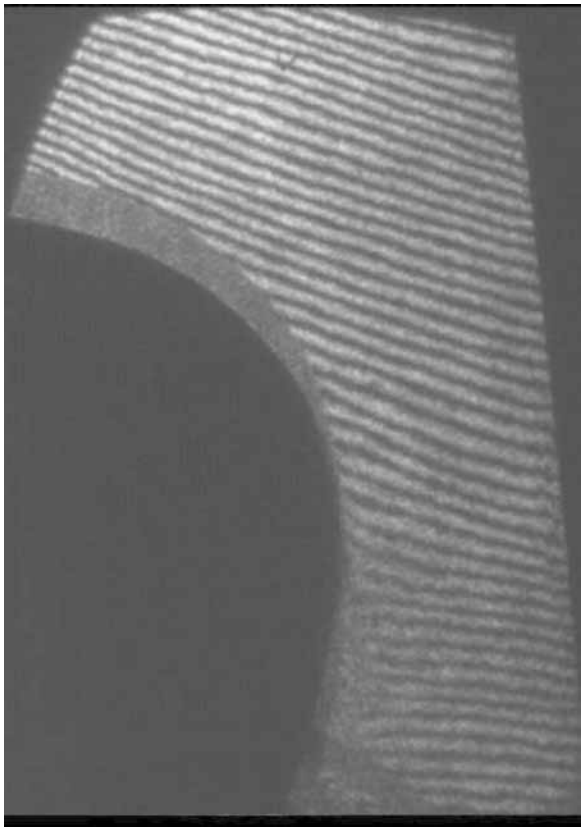

Figure 6.19: Vertically oriented interferogram without flow

the passage. This imposes a lower Mach number distribution for the flow further from the suction surface. This is readily apparent in both the measurements and the calculation. Therefore, the numerical prediction shows the probability of having high density values far into the passage, as was measured.

A plot of theortical density distributions was generated to provide a comparison for our measurements. This plot is based on a crude assumption of linear variation of Mach number across the passage at a selected station in the passage. The slope of this line is determined by looking at the density on the pressure surface and suction surface for the geometry used during the experiment. A simple slope is then calculated. For viewing purposes, the plot was normalized. The densities were normalized on the initial surface density, and the length scale was normalize by the width of the passage along the line normal to the surface. This simplified prediction and the corresponding measured profile are shown in Figure 6.24. As is clear from the plots, the slopes of

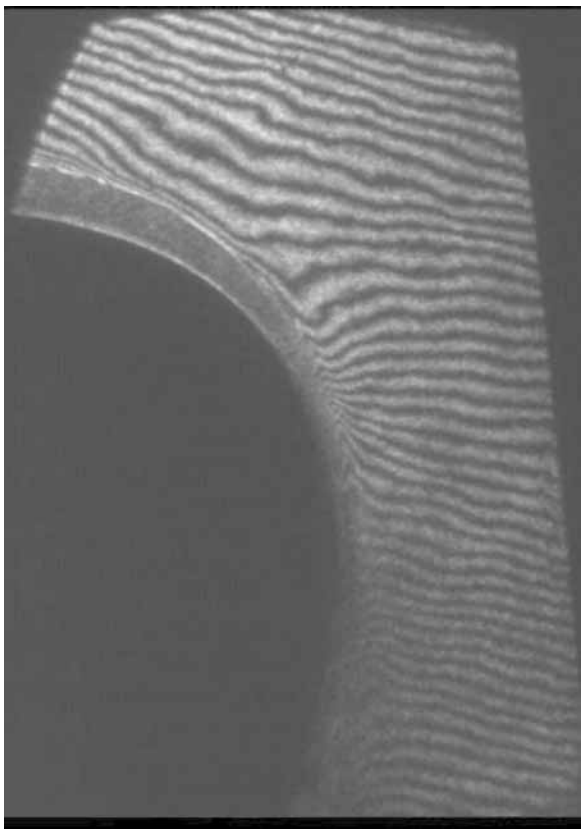

Figure 6.20: Vertically oriented interferogram with flow

these lines are different. Because of the actual non-linear variation of density across the passage, this can be expected. Near the surface the density varies drastically. The slope of the linear line will not capture this detail, because it simply looks at the beginning and ending points across the passage. The non-linear nature of the flow is therefore lost in the analysis. Because of this, the assumed variation predicted by the linear analysis will be less severe than the actual variation. Nonetheless, this simple exercise shows that strong density variations across the passage are to be expected.

Figure 6.25 shows the measured density profiles normal to the surface for several points along the suction side. Each line is labeled at the endpoint with the corresponding normalized distance along the surface. The majority of the data points along each line are outside the coolant film layer. For each line, the surface normal was followed from the approximate surface location up to the end of the calculated region. The points which were invalid because of fringe counting problems were then

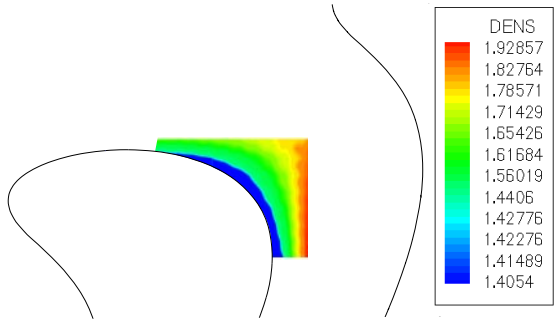

Figure 6.21: Contour plot of the density distribution calculated with the single plate interferometer

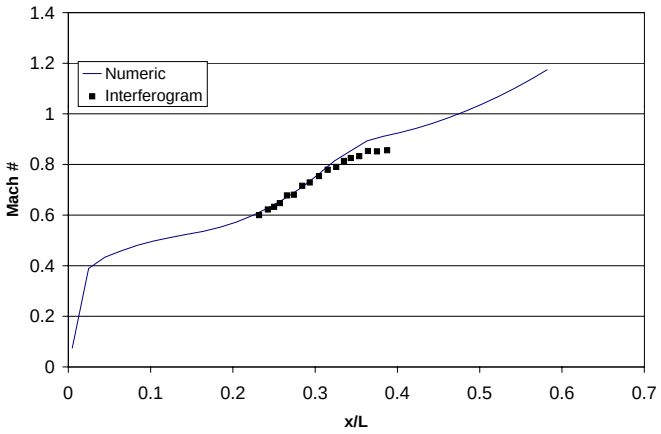

Figure 6.22: Comparison of theoretical and measured surface Mach number

deleted. The plot shown is the final result of this analysis. From this plot, it seems that the film layer does not appear clearly in the density distribution. While this may seem odd, there are two very important considerations to keep in mind. The temperature ratio is low, which means that the density changes created by the temperature gradient are also low. At the same time, the density gradient due to pressure changes in the flow are high. The pressure gradients are, therefore, the dominating parameter in the density gradient. The fact that the slope of the density profiles is relatively constant is also expected, since the length of the area traversed above the surface is

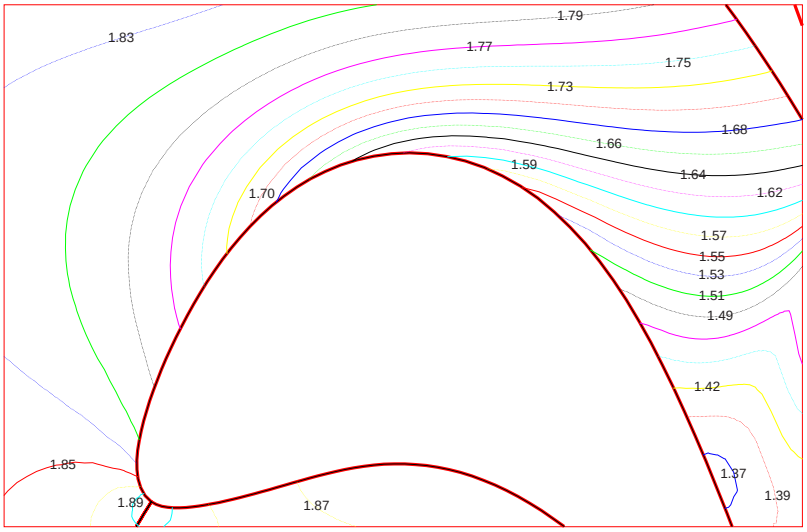

Figure 6.23: Numerically calculated flow field provided by GE (presented values are density in $\frac{kg}{m^3}$) *Courtesy General Electric*

small.

Although the density information is interesting, the temperature distribution through the boundary layer is more important for heat transfer studies. Originally, the temperature profiles through the boundary layer were going to be investigated by converting the directly measured densities into temperatures. Through the film layer, this calculation would have been based on the Ideal Gas Law and the assumption that the static pressure is constant normal to the surface. This assumption is based on standard boundary layer theory (see Schetz [7]). Because of the extreme surface curvature and pressure gradients in the present flow, the pressure variation normal to the surface becomes significant. Therefore, the constant pressure assumption would introduce errors in the temperature profiles. Instead of using a constant pressure assumption, a numerical prediction of the actual pressure profile could be performed. This pressure solution could be used in conjunction with the density measurements to calculate the temperature profiles using Ideal Gas Law.

Another problem occurs because of the double image of the surface, created by the wedge plate. Near any surface, there is a region of interference between the

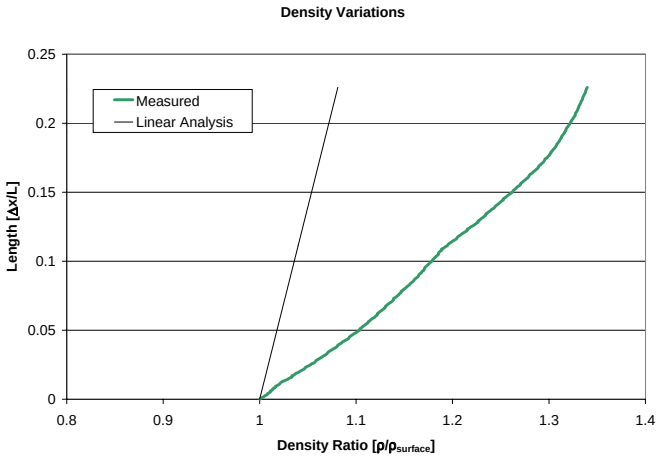

Figure 6.24: Comparison of the density profile normal to the surface

two images. This makes the determination of the actual local surface contours very difficult. Without being able to accurately see the surface contours, it is not possible to accurately tell exactly where the surface is, or what the exact surface normal is. This further complicates the use of the single-plate interferometer through the boundary layer in our setup.

One last problem associated with taking these boundary layer measurements develops from surface interference. Near the surface, there is fringe formation which does not go on beyond the region of interest. For example, in Figure 6.18 we see the fringes in the layer end at the surface. To accurately measure points through the layer, there needs to be fringe information completely surrounding the region of calculation. This is required to sample an accurate fringe count. Near the surface in this picture, rather than having a fringe to the left, there is a solid boundary. This boundary does not correspond to a fringe line, in general. Therefore, only areas near the surface which have a fringe line between it and the surface, can be calculated. This severely limits the ability of the single-plate interferometer to measure surface density values.

Additional error was introduced in the measurement because of alignment

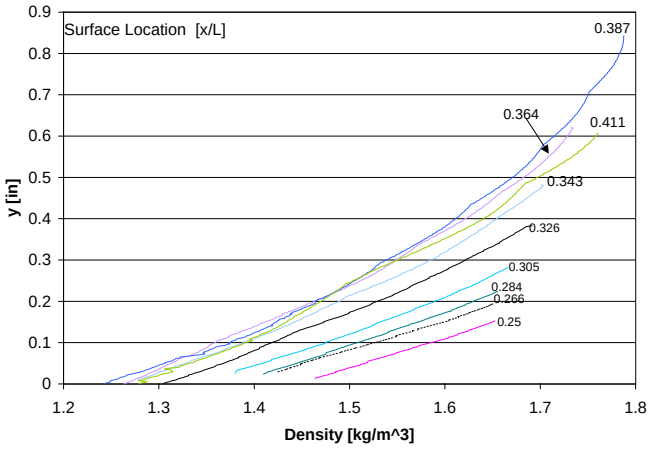

Figure 6.25: Plot of density profiles normal to the boundary layer surface. Line labels at the end of the lines represent the starting surface location as normalized by the total surface distance on the suction side

problems between the horizontal and vertical interferograms. By using the template, it was possible to accurately scale the images; however, due to unsymmetric stretching introduced by the optical setup, it was not possible to accurately position all the fields precisely. The unsymmetric scaling could not be taken out by traditional stretching algorithms, which assume that the entire field is distorted in a uniform way. This scaling problem can be minimized in future setups by more carefully positioning the optics.

These problems highlight some of the inherent problems of the single-plate interferometer compared to the Mach-Zehnder interferometer. The problem created by the double image is absent in the Mach-Zehnder setup. This allows for a much more distinct resolution of the surface. Because the entire flow field is analyzed using one image, scaling issues also don't become a factor in the data reduction process for the Mach-Zehnder interferometer. Furthermore, because it is possible to adjust the sensitivity of the Mach-Zehnder system by adjusting the angular orientation of the mirrors and prisms, it is possible to step normal to the surface through the boundary layer. This removes the problem of counting fringes normal to the surface. Therefore,

the Mach-Zehnder interferometer could be used to calculate density values to the surface.

Theoretically, the sensitivity of the singe-plate interferometer can be changed by adjusting the wedge angle as well. To reduce the sensitivity in this manner is impractical, since this would require a reduction of the wedge angle. The current wedge angle is as low as is possible by use of reasonable manufacturing techniques. The current angle is simply the error in surface parallelism which occurs when attempting to grind a perfectly parallel optical flat. Further reduction of the angle would require custom manufacturing which would be cost prohibitive.

Chapter 7

Conclusions

This paper represents the culmination of several years of work directed at turning a theoretically possible instrument into a practical and usable system. The acquisition of precise positioning hardware and a suitable digital camera allows for an easier, more accurate interferometer setup. The conversion of all the programs into *Java* code makes the program universally available. The improvements in the algorithms used inside the program make it possible to study flow fields which require large numbers of interferograms. The final program, which encapsulates all these changes, provides a software tool for common analysis. Although this last phase required the least amount of theoretical development, it represents the first time that a single-plate interferometer data reduction system has become available for general use.

The test case presented shows both the power and limitations of the single-plate interferometer and the developed data reduction system. It is clear that studying flowfields away from surfaces and solid boundaries is easy with the single-plate interferometer and the developed reduction software. This system can essentially replace the cumbersome Mach-Zehnder interferometer for applications with this type of work. Near surfaces with large density gradients, however the single plate interferometer possesses some limitations which the Mach-Zehnder interferometer does not have. Without being able to readily control the sensitivity of the interferometer, as with the Mach-Zehnder interferometer, it may not be possible to study these flows. As in the test case, the density gradient needs to be reduced before quantitative analysis is

possible.

To carry out the analysis of the temperature profile of the film layer, several improvements will be needed. The omni-directional stepping needs to be developed to allow stepping along arbitrary directions. Furthermore, the wedge angle could be reduced to provide something close to the infinite fringe Mach-Zehnder arrangement. This may provided fringe formations in the film layer which are clearer. The required wedge angle would have to be reduced from 15 arcseconds down to something on the order of 0.1 arcseconds. This would provide density information for the whole layer, however the pressure variation through the layer would have to be calculated using a numerical procedure.

The test case also showed the problems with using Plexiglas as a window material. There is no question that qualitative analysis of the flowfield is possible with Plexiglas. The problems of fringe deformation can be reduced by minimizing manufacturing stresses. Quantitative analysis of these images, however, has proven to be very fickle. Reducing the material stresses created during manufacturing was a trial-and-error process. The current windows were the product of eight sets of windows being manufactured. The combination with the smallest amount of fringe deformation were chosen. It proved difficult to consistently machine the Plexiglas. Although Plexiglas is inherently easier to machine than glass, window manufacturing for interferometry still proves difficult. Therefore, unless window geometry requires the use of Plexiglas, it is recommended that $\frac{\lambda}{4}$ crown glass be used.

Although the software presented here does represent the first truly black-box implementation of the data reduction algorithms, there is still room for improvement. The image processing algorithms should be revamped to allow for completely autonomous fringe detection. This will most likely require a new approach to the image processing. It would also be important to allow for arbitrary stepping directions. The basic reduction procedure for this type of stepping is already in place. The only problem occurs when transferring densities to and from the main calculation domain. The solution to this problem is straight-forward, but because this was not necessary for our analysis, it was not implemented.

Besides these fundamental changes, the interface to this program can further be enhanced. With the current setup, the software uses the now outdated Advanced Windowing Toolkit (AWT). This has been superseded with the much more robust *Swing* GUI classes. By re-implementing the interface using these classes, it will be possible to create a much more user friendly environment. To further help future users, it would also be advantageous to add a graphic front end to the initialization procedures. This would allow a much more intuitive way of generating the stepping procedures for the data reduction.

Bibliography

[1] Bush, Loretta J. *Evaluating the Accuracy of Line Thinning Algorithms After Processing Scanned Line Data*, Virginia Tech

[2] Hariharan, P. *Basics of Interferometry*,Academic Press, 1992

[3] Holder, D.W.; North, R.J. *Optical Methods for Examining the Flow in High Speed Wind Tunnels, Part One*, NATO-AGARD, 1956

[4] Kiss, Tibor *Experimental and Numerical Investigation of Transonic Turbine Cascade Flow*, Virginia Tech, 1992

[5] Merzkirch, Wolfgang *Flow Visualization*,Academic Press ,1974

[6] Popp, Oliver *Steady & Unsteady Heat Transfer in a Film Cooled Transonic Turbine Cascade*,Virginia Tech, 1999

[7] Schetz, Joseph A., *Boundary Layer Analysis*, Prentic-Hall, 1993

[8] Siegman, Anthony *Lasers*,University Science Books, 1986

[9] Wesner, Angela, *An Unsteady Interferometric Study of Passing Shock Flow in Turbine Cascade*, Virginia Tech, 1996

[10] Wesner, A.; Schetz J.A.; Grabowski, H.C. *Interferometric Study of Passing Shock Flow in a Turbine Cascade*, AIAA Journal of Propulsion and Power, Volume 15, Number 4, July/August 1999

[11] http://www.geocities.com/Area51/9851/INIFiles.html

Appendix A

loader.ini

The loader.ini file is the file which is used to initialize the program jbips. The structure is of a standard windows initialization file. The case and the name of the entries, as well as what is expected, must match exactly. If there is any deviation, the program will abort and give you an error message telling you which line is in error. Below is a description of the entire structure, with an overview of the file. An example is provided for the test case in this appendix.

```
[GENERAL]
KnownFieldType="mach" or "density"
NumberSets=<int>
```

The *[GENERAL]* section contains information of general information for initializing the program. The field *KnownFieldType* tells the program whether to expect a reference field given as Mach numbers or as densities. The two options listed above for this field are the only possible values of this field. The field *NumberSets* tells the program how many sets of interferograms to expect. Each flow/noflow combination represents a set. For each set, there must be a corresponding *[INTERFEROGRAM]* entry below.

```
[CONSTANTS]
ImageSeparationDistance=<int>
```

```
L=<double>
k=<double>
n0=<double>
rho_ref=<double>
lambda0=<double>
```

This section tells the program important constants information. The field *ImageSeparationDistance* tells the program the image separation distance in pixels for all the interferograms. The remaining constants are used in calculating the proportionality constant for the stepping procedures. L is the span of the test section, or the area over which the flow is being integrated **in meters**. k is an optical proportionality constant which varies between 0.000290 and 0.000298. *n0* is the ambient index of refraction, which is somewhere around 1. *rho_ref* is the density at sea level, $1.225\frac{kg}{m^3}$. *lambda0* is the wavelength of the laser light **in meters**. For an Ar^+ laser it is $514.5x10^{-9}$m.

```
[INTERFEROGRAM_i]
NoFlowFilename=<String>
FlowFilename=<String>
Orientation="h" or "v"
```

The "i" in the header name corresponds to the interferogram set number which this initialization list corresponds to. This index is zero based, so if there is only one interferogram the header would be `[INTERFEROGRAM_0]`. The two filenames correspond to the filename which each interferogram is related to. The field *Orientation* is used to tell the program whether the given set is a horizontal or vertical interferogram. The only two possible options for this are "h" or "v".

Appendix B

Test Case loader.ini File

This file corresponds to the loader.ini file for the test case. It is called `filml.ini`.

```
[GENERAL]

KnownFieldType=mach

NumberSets=2



[CONSTANTS]

ImageSeparationDistance=101

L=0.1524

k=0.000295

n0=1.0002506
```

```
rho_ref=1.225

lambda0=515.5E-9



[INTERFEROGRAM_0]

NoFlowFilename=horizontal_film_noflow.thn.gif

FlowFilename=horizontal_film_flow.thn.gif

Orientation=h



[INTERFEROGRAM_1]

NoFlowFilename=vertical_film_noflow.thn.gif

FlowFilename=vertical_film_flow.thn.gif

Orientation=v
```

Appendix C

runner.ini

This file is the initialization file to run a particular case. It has been designed to allow on-the-fly editing. In other words, the program doesn't have to be restarted to try different options, or to addremove various steps. Like loader.ini, this file is a standard Windows initialization file. Therefore, case, wording and options must meet these exact specifications. The program will also abort if there are errors in this file.

```
[GENERAL]
KnownDataFieldName=<string>
ReloadDataField="true" or "false"
NumberOfCommands=<int>
DefaultMachNumber=<double>
StagnationDensity=<double>;
```

This section provides general information for the running process. The field *KnownDataFieldName* provides the filename for the initial reference field. *ReloadDataField* determines whether the program reloads the reference field each time it runs. This is helpful if the program is being tweeked, since it removes the need to load the file off the disk, which can take time. *NumberOfCommands* determines how many separate command sequences there are. For each command sequence there needs to be a corresponding entry below. *DefaultMachNumber* gives the default Mach number to be used for the average field calculation, if the average field is used for the reference

field. *StagnationDensity* gives the value of the stagnation density in $\frac{kg}{m^3}$.

```
[DOMAIN_SIZE]
width=<int>
height=<int>
```

This field is used to tell the program how large the computational region is in pixels. The computational region is where all the density calculations are stored.

```
[COMMAND_i]
Multiplier=+/- 1
SetNumber=<int>
StepDirection="right","left","up","down"
NumberOfSteps=<int>
nx=<int>
ny=<int>
fx=<int>
fy=<int>
ox=<int>
oy=<int>
CalcHeight=<int>
```

For each step this section tells the program what it will be performing. The "i" corresponds to which step. This too is zero-indexed, so the first step's header would be `[COMMAND_0]`. The term *Multiplier* tells the program what the sign of the proportionality constant in the reduction equation should be. This value can only be ± 1. The term *SetNumber* is used to tell the program which interferogram set that the calculations will be made with during this command sequence.

StepDirection is used to tell the program which direction the stepping will be done in. The four values listed are the only possible values for this term. The program checks the step direction against the interferogram set, to make sure that the two are consistent. For example, if the command is to step "right", but the interferogram set

which is being used is a vertically oriented interferogram, then the program will give an error.

The next series of options corresponding to the starting point of the calculation in terms of the x and y coordinates in the noflow image, flow image and the output field, respectively.

CalcHeight tells the program how long the analysis field is for this step, normal to the image separation distance. For horizontal steps, this will correspond to the height of the field. For vertical steps, this will correspond to the width of the field.

Appendix D

Test Case runner.ini File

Below is the text of the `runner.ini` file for the test case. This file is titled `filmr.ini`.

```
[GENERAL]

KnownDataFieldName=film.plt

ReloadDataField=false

NumberOfCommands=3

DefaultMachNumber=0.48

StagnationDensity=1.91306



[DOMAIN_SIZE]

width=1339
```

```
height=1000



[COMMAND_1]

Multiplier=-1

SetNumber=0

StepDirection=right

NumberOfSteps=2

nx=840

ny=152

fx=840

fy=152

ox=840

oy=152

CalcHeight=180
```

```
[COMMAND_0]

Multiplier=-1

SetNumber=0

StepDirection=left

NumberOfSteps=8

nx=840

ny=152

fx=840

fy=152

ox=840

oy=152

CalcHeight=180



[COMMAND_2]

Multiplier=-1
```

```
SetNumber=1

StepDirection=down

NumberOfSteps=7

nx=256

ny=312

fx=256

fy=312

ox=502

oy=294

CalcHeight=585
```

Appendix E

Java Primer

A detailed set of information on Java is readily available on the internet, or in the bookstore. There is some basic information that is necessary for the novice or uninitiated Java programmer to known before running or modifying this code. These basics are presented below in different sections, divided by their functionality.

It is possible to run the program without any real programming knowledge. Modifying the program will require some programming knowledge. It would be best if the programmer has Java experience. It is still possible to do without having a full understanding of Java programming. The programmer needs to have at least a qualitative understanding of object oriented programming however. If someone is familiar with C or C++ then there will be few problems with changing this code. A Pascal programmer would have a bit more difficult of time, since Java syntax is heavily based on C++. FORTRAN coders will have some further difficulty.

E.1 Running Java Programs

Java achieves its portability by **not** compiling the programs into machine-specific binary. Instead, the java compiler outputs what is known as *Java bytecode*. This is a machine independent binary format. To run this code, the computer must interpret the java bytecode. This is done by something called a *virtual machine*. Most operating systems have a java virtual machine JVM). The commands presented here are for the

Java2 compatible JVM's. This is not necessarily universally applicable, but it is mostly consistent. Each JVM should provide its own directions.

The command structure for running a java program will be:

```
java -<options> classname <program options>
```

There are two useful commands for running the program. The first is `Xmx`. This command sets the maximum amount of memory available to the program. The argument of this option is an integer for the number of megabytes followed by "M" (ie `128M`). If the program tries to store more than this amount of memory, it will abort with an `OutOfMemory` error. If the *HotSpot* engine is installed, which is highly recommended on Windows/Solaris systems, then an option `Xincgc` should be used as well. This option incrementally provides garbage collection. This significantly reduces the memory footprint of the program, since it more frequently deletes objects.

The program options list is where the specific class' options would go, such as filenames. A full fledged execution of the program *jbips* under Windows or Solaris with a HotSpot engine would be:

```
java -Xincgc -Xmx128M jbips loader.ini
```

E.2 Java Quick Guide

This section is for the programmer who feels comfortable editing the code, has a good programmer's base, and needs some information on common syntax and problems in Java.

E.2.1 Memory Management

Unlike C and FORTRAN, all memory in Java is dynamically allocated. Also unlike both of these languages, there is no need to perform garbage collection manually. Garbage collection is the process of explicitly deleting dynamically created variables. This is all handled by the JVM. There are several basic classes: byte, char, int, long,

float, double, String. These are first class objects. To create a new variable of these types simply follow C syntax:

```
int x=0;
double a,b,c;
String name="Joe Jones";
```

For defining and allocating objects, follow C++ syntax:

```
Object x;
x=new Object();
Frame f;
f=new Frame("Main Frame");
Label l=new Label("Run");
```

where the name following the new operator has the arguments for the given constructor.

Arrays are declared like dynamic arrays in C++. If there are an array of objects, you follow the same syntax. However, even though the array of objects is allocated, it will still be necessary to manually initialize each one using their constructors, examples of both are provided below:

```
double[] x;
x=new double[100];
interferogram[] test;
test=new interferogram[100];
for(int i=0;i<100;i++){
   test[i]=new interferogram("test.gif",'h');
}
```

Control statements work exactly like they do in C. Above is an example of the **for** loop.

E.2.2 Object Structure in Java

Objects in Java are much simpler than in C++. The basic structure of a Java class definition would be something like:

```
public class helloWorld{
   public double x;
   private Frame f;

   public helloWorld(String name){
      f=new Frame();
      f.setTitle(name);
      f.resize(300,300);
      f.show();
   }

   public double value(){
      return(x);
   }
}
```

In this basic example, there is a constructor, a member function and two variables. This should provide a very crude look at how things are done in Java. More information can be found at the below sights.

E.3 Java Information

By far the best place for information on programming in java is Sun's Java web page. From this site you can download introduction programming guides, API manuals and examples. This is also the location to download Java development kits supported by Sun, or to find Java development kits that are available for other platforms. Currently the only platforms which are officially support by Sun are Windows and Solaris. At

this moment they are about to have official support for Linux. An example of the myriad of operating systems that have Java runtimes: FreeBSD, NetBSD, OpenBSD, Linux, MacOS, BeOS,SCO Unix, NeXTStep, IRIX, AmigaOS, HP-UX, etc.

Sun's Java web page: `http://www.javasoft.com`
Resources: `http://www.gamelan.com`
Helpful Online Magazine: `http://www.javaworld.com`

Appendix F

ContourPlot.java

The contour plot class provides a way of internally creating contour plots for the program. It provides a very crude way of performing this task. The system works on a Hue-Saturation-Brightness (HSB) color model. It was hypothesized that it would be possible to create a continuous gradation with a maximum of 256 levels by simply changing the hue and holding the saturation and brightness constant. The class has a 2D double precision floating point data structure. To save memory, this simply points to the data structure used to create the array. Although this isn't very safe from a memory management point of view, the garbage collector shouldn't delete it until the array in *ContourPlot* stops pointing to it.

F.1 API Guide

public ContourPlot(double[][] temp,float minimum, float maximum,float cutoff1)

This is a constructor for the *ContourPlot* class. The first argument is the array that the contour plot should be based on. The second and third arguments set the lowest and highest contour level. The final parameter selects the cutoff value. Any value below the cutoff value is automatically assigned a white pixel.

public ContourPlot(double[][] temp)

This is another constructor for the *ContourPlot* class. The sole argument for this class is a 2D, double precision array. With this constructor the object automatically selects the minimum and maximum contour levels. This is based on the lowest and highest values in the matrix. The cutoff is automatically set to the lowest possible value of a double precision number.

public boolean mouseDown(Event evt,int x,int y)

This method is activated by the Java Runtime Environment (JRE) whenever the left mouse button is pressed while the cursor is over the ContourPlot object. It will print the local contour plot value to the standard output. A future version will store the current value in a string to be presented in a popup message.

public void setPlot(double[][] temp)

This method is used by the program to setup the contour plot. It can be used by the user, however there have been some problems recreating the contour plot after the initial object is made. It is better, although less efficient, to simply create a new *ContourPlot* object if the information in the plot has changed.

public void setPlot(double[][] temp,float minimum,float maximum)

This method is similar to the other *setPlot* method, but allows the programmer to select the minimum and maximum contour level.

public void listValues(String filename)

This method outputs the contour levels to the given filename. This is used for the person who needs to see a corresponding hue-to-density information. The output is a two column list, with the first column being the hue index and the second being the hue value.

public void paint(Graphics g)

This method is used be the JRE to paint the image to the screen.

public Dimension minimumSize()

This method is used by the JRE as a hint to the preferred minimum size for the object.

public Dimension preferredSize()

This method is used by the JRE as a hint to the preferred size for this object.

public Rectangle getBounds()

This method will return the size of the object in the form of a *Rectangle* object. This again is used mostly by the JRE

Appendix G

grayImage.java

The *grayImage* class is the class definition used to work with grayscale images. It was important for all the algorithms to have a grayscale image with an 8-bit index. This index has 0 for black and 255 for white. A *grayImage* object can read in GIF,JPEG or RAW images. All of these are done in a slightly different way. The only way to read in a GIF or JPEG file is through constructing a new *grayImage* object. RAW images must be read in explicitly, since none of the size information is inherent in a RAW image. When the object writes its values back to the disk, it writes it in the form of an 8-bit RAW image. The storage device used is a one dimensional array of 16 bit integers. This was required since there are no unsigned 8-bit data types.

G.1 API Guide

public grayImage(int r,int c)

This is the basic constructor for the *grayImage* class. This is the constructor used if a RAW image is going to be read into the object. The constructor sets up an array large enough to hold the image specified by the rows and columns arguments above.

public grayImage(String filename)

This constructor is used if the desired image is stored in a GIF or JPEG file. Reading in either image should be done with caution however. The system is expecting a

grayscale image to be stored in either of the above file types. In a grayscale image, all three of the basic colors (red, green and blue) are going to have the same values. For this reason, the algorithm solely stores the value of the red color. If a color image is read in instead of a grayscale, the object will not convert it to grayscale. Instead the image's red component only will be read in. This will most likely produce undesirable results.

public int get(int r,int c)

This method gets the pixel value at the point (r,c). It should be remember that the image is stored in row major form, therefore the rows are down and the columns are across.

public void put(int r,int c,int value)

This method will put the integer value at point (r,c). The method will not check if the number is greater than 255. If the value is higher than 255 then unexpected results can arise, especially when writing the file to disk

public void put(int r,int c,byte value)

This method writes an 8-bit value to point (r,c). **byte** values can only be between -128 and 128.

public void readImage(String filename)

This method is used to read in a RAW image file. The first thing which should be done is create a new *grayImage* with the proper size, using the first constructor. Then the specified image should be read in with this method.

public void writeImage(String filename)

This method writes the given *grayImage* object to the disk in a RAW file format.

This will be a string of 8-bit values.

public int getRows()

This method returns the number of rows in the image.

public int getColumns()

This method returns the number of columns in the image.

Appendix H

imageGetter.java

This small class provides the mechanism from which a JPEG or GIF image can be read in. These mechanisms are part of the standard Java media API's. Although simple, it was used to encapsulate the necessary methods in order to insulate the programmer from the various toolkit semantics

H.1 API Guide

public imageGetter(String filename)

This is the constructor for this class. Any valid image format will be accessed from this method. Currently that only includes GIF and JPEG images. In future releases of the Java APIs that will include BMP,TIFF,and PNG images.

Appendix I

interferogram.java

This class encapsulates all the necessary methods and structures necessary to operate with reduced interferograms from within the program. The fringes are stored in two 2D arrays, which are not directly accessible by the user. The first array stores the fringe location along a counting line, the second stores the fringe count value.

I.1 API Guide

public interferogram(String name,char type)
Standard constructor. The first argument is the name of the image file that the image reduced interferogram is stored in. This needs to be either a GIF or JPEG images. As stated before, to reduce the problems that the lossy compression used in JPEGs produces, the image should really be stored in a GIF image.

interferogram(String name,char type,int tile)
This is the same as the above constructor but allows you to store the tile number of the interferogram. This comes in handy when working with multiple interferogram sets. The tile number for a given flow/noflow interferogram set should be the same.

public double getCount(int row,int column)
This method returns the fringe count at the given (row,column) location. The images

are stored in row major form.

public String getType()

This method returns the interferogram type. The string will be one of three values:

- "Vertical Interferogram"
- "Horizontal Interferogram"
- "Not Properly Initialized"

public int getRows()

This method returns the number of rows in the interferogram

public int getColumns()

This method returns the number of columns in the interferogram

public int getTileNumber()

This method returns the tile number of the interferogram

Appendix J

interproc.java

The *interproc* class encapsulates all the required methods for reducing interferograms. The coordinate systems used in the various stepping algorithms can be found in the thesis. The method arguments however are listed below.

J.1 API Guide

public void HTakeDensity(double[][] field,double[][] densArray,Point start)
This method samples the density from the array *field* and stores it in *densArray*. The starting point in *field* is passed in through *start*.

public void VTakeDensity(double[][] field, double[][] densArray,Point start
This method provides the same functionality as the above method for when the processing involves vertical samples. The separate method is needed to make it easier to do consistent indexing with the *densArray* object

void LoadDensity(String filename,double[][] densArray)
This method loads a density field from the disk stored in *filename*. The size of the known field is gotten from the dimensions of *densArray* before it is passed in. Any points which are outside of this boundary will not be read in from the file. For example, if the size of *densArray* is 50x150, and if the file has an array stored 100x200,

then only the points from 0..50x0..150 will be read. The rest will be thrown out. The file should be an ASCII file without any header. The structure should be a series of rows with the following format: x,y,value. All three values can be ints or doubles. However, if they are doubles, then they **cannot** be in scientific notation. This is a limitation of the class used to read in the ASCII files.

void HStudyStepRight(interferogram NF,interferogram FF,double[][] DF, double[][] densArray,Point nStarts,Point fStarts,Point startO, int max_x,int max_y)

This method is used to step right from the known field into an unknown area. The first interferogram is the noflow interferogram, the second is the flow interferogram. *DF* is the array which the final calculations are written to. *densArray* is the known field information. This area will be written to *DF* as well as the calculated region. The series of *Point* objects are the starting points in the noflow interferogram, flow interferogram, and output array, respectively. *max_x* defines the number of columns in *densArray* and the area to be calculated. This **must** be equal to the image separation distance. *max_y* is the height of the calculated region.

The algorithm assumes that this method will be used in a loop. Therefore, it updates the starting location at the end. For this reason, if one was going to step right several times, all they have to do is run this algorithm multiple times, with a *HTakeDensity* before each iteration to sample a new reference density field. *jbips* provides an illustration of this.

void HStudyStepLeft(interferogram NF, interferogram FF,double[][] DF, double[][] DensArray, Point NStarts,Point FStarts, Point Starto,int max_x,int max_y)

This performs the same functions, and has the same inputs as *HStudyStepRight*, but steps from a known field to the left, into an unknown region.

void VStudyStepDown(interferogram NF, interferogram FF,double[][] D-

F, double[][] DensArray, Point NStarts,Point FStarts, Point Starto,int max_x,int max_y)

This performs the same function, and has the same inputs as *HStudyStepRight* but steps down from a known area.

void VStudyStepUp(interferogram NF,interferogram FF,double[][] DF, double[][] densArray,Point nStarts,Point fStarts,Point startO, int max_x,int max_y)

This performs the same function, and has the same inputs as *HStudyStepRight* but steps up from a known area.

Appendix K

jbips.java

This class is the main program for the interferogram processing. No direct interaction with the code itself is necessary for the standard user. This is all done with *.ini files. This structure is found elsewhere in the appendix

K.1 API Guide

public static void Version()
This method outputs the current version number to the standard output.

public jbips(String filename)
This method is the constructor for this object type. The filename in this section is the loader input file.

public void setDensity(double[][] x)
This method takes the array x and converts all the values in that matrix to densities. It is assumed that this method is a matrix of Mach numbers. The conversion parameters were entered in the initialization file.

public double[][] densityToMachNumber(double[][] Area)
This method takes the array Array and converts it from densities to Mach numbers.

This assumes isentropic flow for every point. It will provide invalid calculations in viscous areas or areas with heat transfer.

public static void Output(String filename,double[][] area)

This method outputs the given array to the file given in *filename*. The format is as a standard Tecplot7 ASCII input file. It can be read without modification into Tecplot.

public void loadData(String filename)

This method loads the initial data series, which includes all constants needed to initialize the *interproc* object and all the interferograms. Once this method has been completed successfully, it will set a variable which will allow the program to run its analysis method.

public void runAnalysis()

This method runs the analysis system. It will not run unless the object has finished being initialized. This is determined by the object itself. It runs based on it's runner initialization file.

Appendix L

linepick.java

This class provides all the algorithms necessary to implement the line following method. It is a static class, which means that it is not necessary to create a *linepick* object.

L.1 API Guide

public static char[][] convertToCharMap(grayImage image)
This method takes a grayImage and converts it into a character array. The method assumes that it is looking at a binary image, with black having a value of 0. It assigns character flags based on the internal convention: 0 is default,1 is black,2 is white and greater values are reserved for flags.

public static boolean checkPoint(char[][] image,int i,int j)
This method checks to see if the point is actually part of a line or if it is a stray point, or a point in a solid

public static void setPoint(char[][] image, int i, int j)
This method sets the flag for a certain point based on the algorithm presented in this

paper

public static void setUpper(char[][] image,int i,int j)

During algorithm implementation, it turned out that a slight variation of the algorithm was necessary when stepping up. This was to maintain a consistent flag for the entire line. This method is used to set the upper three points of the matrix surrounding a black pixel.

public static void analyze(char[][] image

This method is the main means of interacting with the line following algorithm. The input is a character array which conforms to the sign convention presented above

public static void setLabel(char[][] image,int i,int j,char index)

This method sets the labels in the 3x3 matrix.

public static grayImage convertToImage(char[][] image)

This method will convert the matrix to a 1-bit grayscale image. Any flag will be output as a black pixel.

public static void writeTable(String filename,char[][] image)

This method will output a table with the values of the matrix *image*. The table will simply be a string of numbers, with carriage returns at the end of each row.

Appendix M

MathRead.java

This class allows for the reading of integers and floating point numbers from an ASCII file. This feature is not available in standard Java, although later implementations have it available. This class should probably be replaced or made more efficient.

M.1 API Guide

public static boolean readDouble(DataInputStream din,double[] temp)
This method accepts a standard input stream and will fill the array with the necessary values. If the stream runs out of numbers before it fills the array, then an error will be thrown.

Vita

Henry C. Grabowski III was born in Wilmington, Delaware on June 4, 1975 to Henry C. Grabowski Jr. and Mary Grabowski. He grew up in the Wilmington area his entire life and attended Archmere Academy for high school. After a brief stint at the United States Air Force Academy, he attended VPI from January 1994 to September 1999. In May 1998 he received his BS degree in Aerospace Engineering from VPI. In December 1999 he received his MS in Aerospace Engineering with a specialization in fluid mechanics. He has now gone on to work in the aerospace software industry near his home town.