A cyclic shearing interferometer for collimating beams with short coherence times
by Timothy Don Henning

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Physics
Montana State University

Abstract:
Until now there has not been an accurate method for measuring the radius of curvature, R, of a short coherence length light source such as a short-pulse or broad-band laser. We show that the easily aligned, Cyclic Shearing Interferometer (CSI) solves this problem. The CSI produces a stable fringe pattern from which R can be determined and can be used on beams with short coherence times down to 300fsec, since the two beams in the interferometer follow nearly the same path. Comparison with data from a broadband, XeCl laser (30psec coherence time), confirms that the CSI performs as theory predicts.

A CYCLIC SHEARING INTERFEROMETER FOR COLLIMATING BEAMS WITH SHORT COHERENCE TIMES

by

Timothy Don Henning

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Physics

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 1991

APPROVAL

of a thesis submitted by

Timothy Don Henning

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

5-22-91
Date

Chairperson, Graduate Committee

Approved for the Major Department

5-22-91
Date

Head, Major Department

Approved for the College of Graduate Studies

June 14, 1991
Date

Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under the rules of the Library. Brief quotations from this thesis are allowed without special permission, provided that accurate acknowledgment of source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Dean of Libraries when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature 

Date 3-22-91

ACKNOWLEDGEMENTS

This author would like to thank a number of people for helping me complete my project and this thesis. I send a warm thanks to Doug McGarry and TMA for their help researching and assembling the video system which was used for data acquisition. I am also thankful to the secretarial staff at the Physics Department for their constant help on the use of word processors. I am also grateful for the constant source of support, intellectual discussions, and help provided by the other members in Carlsten's labs. A special thanks goes to one of those members, Phillip Battle, who not only helped with a lot of programming problems, but also presented a lot of insightful thoughts. This author also thanks his graduate committee, Dr. George Tuthill and Dr. R.T. Robiscoe, who carefully read this thesis and offered many suggestions on how to improve it.

I offer my deepest gratitude to my advisor, Dr. John Carlsten, who taught me much more than physics. Some of the physics that I have learned from him forms the firm basis which this work was built upon. Whereas the practical knowledge that I have learned from him, such as communication skills and a holistic view to problem solving, cannot easily be measured.

I am indebted to all of my friends who have supported me and made my stay in Bozeman a memorable one. Finally, I am indebted for life to Diane Mathews for her constant source of love, support, and smiles.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES -- continued

ABSTRACT

Until now there has not been an accurate method for measuring the radius of curvature, R, of a short coherence length light source such as a short-pulse or broad-band laser. We show that the easily aligned, Cyclic Shearing Interferometer (CSI) solves this problem. The CSI produces a stable fringe pattern from which R can be determined and can be used on beams with short coherence times down to 300*fsec*, since the two beams in the interferometer follow nearly the same path. Comparison with data from a broad-band, XeCl laser (30*psec* coherence time), confirms that the CSI performs as theory predicts.

CHAPTER 1

INTRODUCTION

In many experimental situations it is often necessary to collimate or, more specifically, measure the radius of curvature (R) of a light beam. To this end there are a number of methods, both geometric[1] and interferometric[2-11], that have been developed which can be used to check the degree of collimation of a light beam. However not all of these methods are appropriate for collimating sources with short coherence times. A short synopsis of each method and its applicability to short coherence length sources is given below.

To collimate a beam using the geometric approach one simply needs to compare the size of the beam at two different points along the beam's axis. While this method can be applied to sources with short coherence times, it is cumbersome because it requires a large distance to obtain moderate accuracy. Bass and Whittier[1] have improved the geometrical method using a technique with corner cube backreflection. However, this improved technique still requires a large distance ~$15m$ to obtain collimation within the range $360m \leq R \leq \infty$.

A more compact method to check the degree of collimation of a light beam can be realized by using interferometric techniques. The interferometers that have been used can be put into two categories; those using beam splitters or shearing plates[2-7], and those using diffraction gratings[8-11]. The first category of interferometers includes a parallel shearing plate[2], a modified Michelson interferometer[3], a wedged, shearing plate interferometer[4,5] (often referred to as a collimation tester), a cyclic shearing interferometer[6], and a modified wedged, shearing plate interferometer[7]. The second category, interferometers utilizing diffraction gratings, are all interferometers of the Talbot type[8-11] . They all use the Moiré technique to produce intensity patterns in which collimation can be determined. Unfortunately, all of the interferometers except the modified Michelson and Cyclic Shearing Interferometer have an inherent pathlength difference between the two interfering beams which renders them unusable for short coherence length sources. While the modified Michelson interferometer can be operated at a near-zero pathlength difference, it is susceptible to small mechanical vibrations and air currents. However, the Cyclic Shearing Interferometer (CSI) has inherent stability as well as a near-zero pathlength difference making it ideal for collimating short coherence length sources.

A survey of the early beginnings of cyclic interferometers is given by Hariharan[12]. Four other papers[13-16] have dealt with the Cyclic Shearing Interferometer (CSI) with little or no interest in the radius of curvature. The most recent of these papers[16] talks about the stability of a CSI. The article by Cordero-Davila *et. al*[6]. discusses a method to measure the radius of curvature of a converging wavefront. Wenzel[17] did some initial calculations

for a vertical shearing interferometer and discussed the possibility of using the interferometer for measuring the radius of curvature of a short coherence length source.

Chapter 2 will describe the CSI and its operation for measuring the radius of curvature (or the collimation) of a short coherence length source. Before discussing the detailed analysis of CSI, a novel ray tracing method is described. As a demonstration of the ray tracing method it is used to analyze the wedged, shearing plate interferometer (Collimation Tester) to rederive the results of Riley and Gusinow[5] in Appendix A. Next, a general fringe pattern will be derived for a shearing interferometer and it will be shown how the radius of curvature can be determined from various parameters measured from the fringe pattern. The CSI will then be analyzed in detail using the ray tracing method.

Chapter 3 will discuss an experimental test of the CSI using a broadband XeCl laser and show that it does behave as expected. The fringe patterns are acquired with the use of a video system which is described in Appendix C. Once the fringe patterns are acquired they are analyzed to find the incident Radius of Curvature by using a procedure outlined in Appendix B which uses a least-squares fitting program for which the source code is given in Appendix D.

Chapter 4 discusses some special considerations when using the CSI as well as one design for a compact CSI.

CHAPTER 2

THEORY

The Cyclic Shearing Interferometer (CSI), shown in Fig. 1, is a single-pass cyclic interferometer in which a single incident light beam is split into two beams which follow nearly the same path in the interferometer, but in different directions. The two exiting beams interfere and produce a fringe pattern on a screen from which several parameters can be measured to determine the radius of curvature of the incident beam. The fact that the beams follow nearly the same path provides the CSI with two distinct advantages. The first advantage is that the CSI is much more stable to mechanical vibrations than other multi-optic interferometers such as a Mach-Zehnder interferometer[16]. The second advantage is that the two counter-rotating beams' pathlengths will be nearly the same. A pathlength difference very close to zero means that the CSI could be used with sources that have short coherence lengths (short coherence times) such as broadband lasers or short pulsed lasers.

Ray Tracing Method

A vector ray tracing method was used to evaluate how the various optical surfaces of an interferometer can split an incident beam to produce the two beams that form a useable fringe pattern. The method, described by Herzberger[18], is easy to use and easy to implement on a computer. Herzberger showed that both Snell's law and the law of

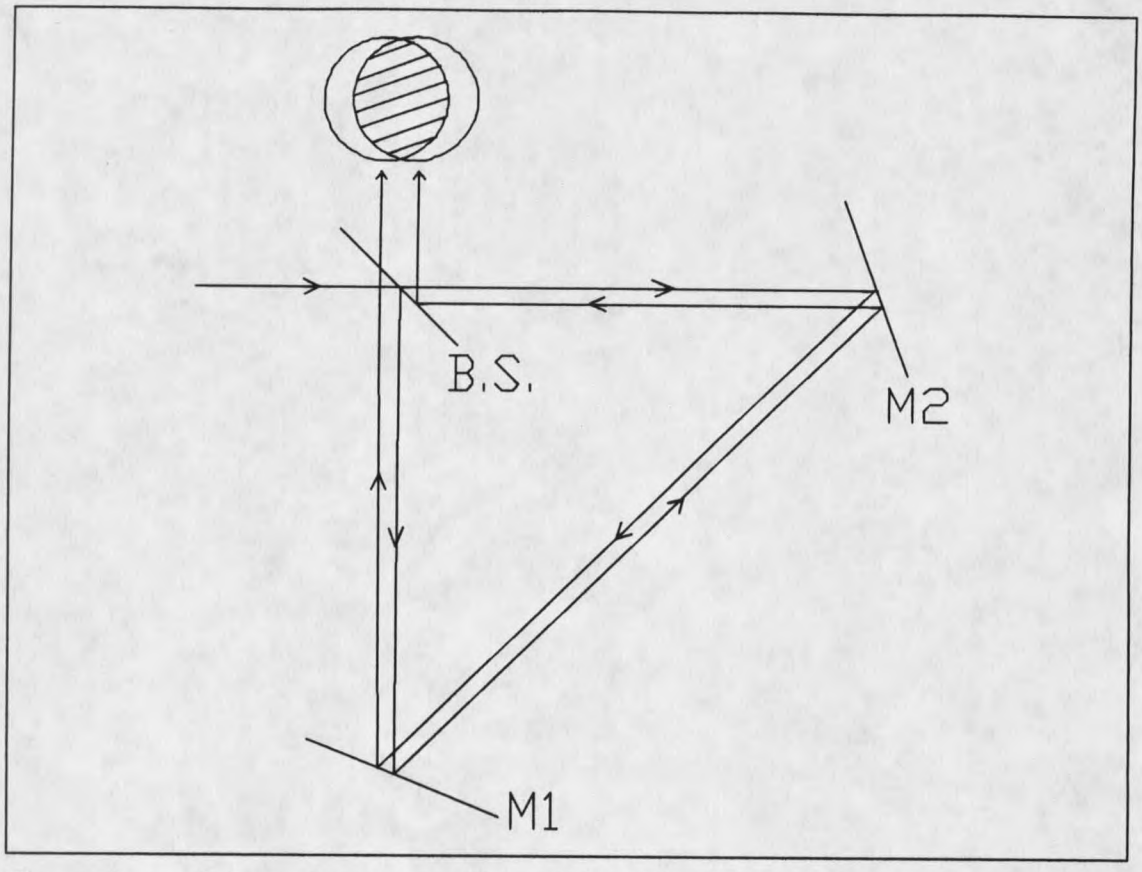

Figure 1. Schematic representation of the CSI. M1 and M2 are mirrors and BS is a 50% beam splitter. Note that the two counter-rotating rays shown represent the middle of the two beams whose diameters are about the same as the diameters of the optics. A typical fringe pattern is shown at the CSI output.

reflection can be combined and written compactly in vector form given by

$$S' = S + \Gamma N . \tag{1}$$

S is the direction vector associated with the beam incident on the optical surface, S' is the beam direction vector associated with the transmitted or reflected beam, and N is

the optical surface unit normal, which is on the opposite side of the interface from the incident beam. These are shown graphically in Fig. 2. The magnitude of S (or S') is given by the index of refraction of the medium in which the beam propagates. Γ is defined for the refraction or reflection off an optical surface and is given by

$$\begin{aligned} \Gamma_{refract} &= \left[(n_2^2 - n_1^2 + (N \cdot S)^2)\right]^{1/2} - N \cdot S \\ \Gamma_{reflect} &= -2(N \cdot S), \end{aligned} \tag{2}$$

where the n's are the indices of refraction for the two media.

The following section will describe how to trace a beam through an optical system to find the exact paths of the beams once the beam direction vectors are known (Eqs. (1) and (2)). This is useful when finding the pathlength difference introduced in an interferometer or when finding the shear introduced by an interferometer. A graphical representation of the vector tracing formula is shown in Fig. 3. Note that the starting and ending points a and a' are specified by the vectors, $\boldsymbol{a} = x\hat{x} + y\hat{y} + z\hat{z}$ and $\boldsymbol{a}' = x'\hat{x} + y'\hat{y} + z'\hat{z}$, with their tails at a fixed origin. P_o is a point known to be on the optic plane in which the initial beam S will be reflected (or refracted). The vector tracing equation is simply

$$\boldsymbol{a}' = \boldsymbol{a} + D * \frac{S}{n}, \tag{3}$$

where D is the distance traveled from the starting point a and n is the magnitude of S.

The ending point a' cannot be found until the distance traveled, D, is known. One can find D by observing that when the point a' lies on a given optic plane a line drawn from the point a' to a known point on the plane, P_0, will be perpendicular to the plane's normal. In mathematical terms this can be expressed as

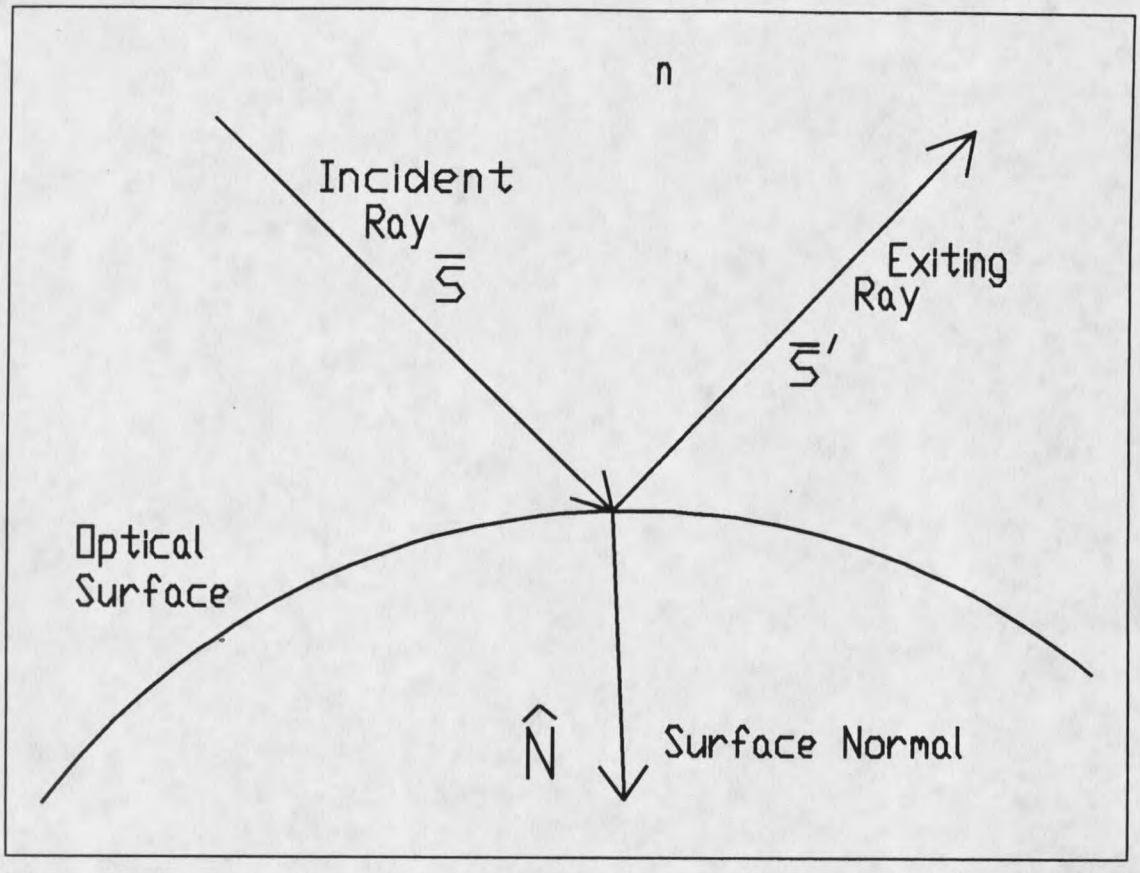

Figure 2. A diagram of a beam being reflected off of an optical surface. The incident ray, S, being reflected off of an optical surface with normal N to form the reflected beam S'. The direction of S' can be found from $S' = S + \Gamma N$ (Eqs. (1) and (2)).

$$\overline{P_0 a'} \cdot N = 0, \tag{4}$$

or, in more explicit terms

$$(x' - x_0) N_x + (y' - y_0) N_y + (z' - z_0) N_z = 0. \tag{5}$$

If the explicit expressions for the components of a', from Eq. (3), are put into Eq. (5) one can solve for D, the distance between the points a and a', which yields the

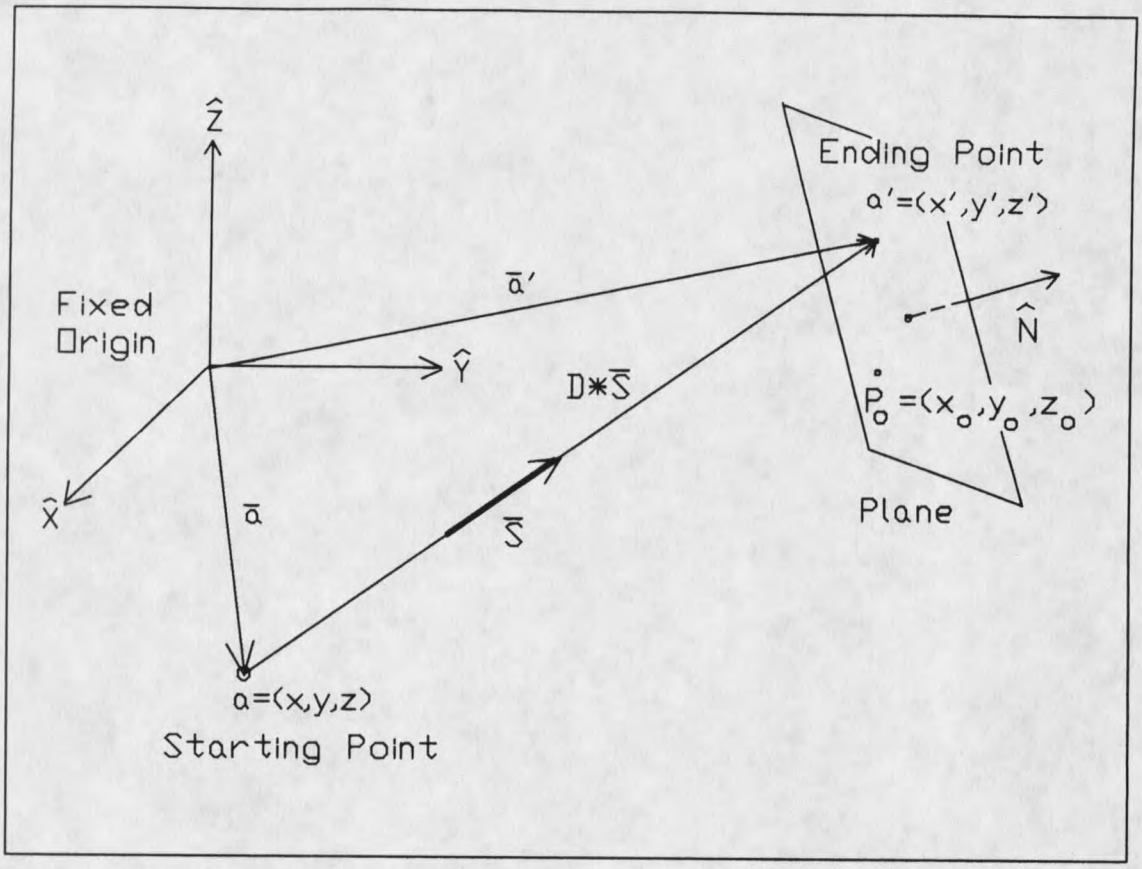

Figure 3. Schematic of a beam traveling a distance D in the direction S from point a to point a' which lies on the plane defined by the plane's normal N and a known point P_o. The vectors a and a' are the vectors that point to the points a and a' respectively.

expression

$$D = \frac{(x_0 - x)N_x + (y_0 - y)N_y + (z_0 - z)N_z}{S_x N_x + S_y N_y + S_z N_z}. \qquad (6)$$

Note that Eq. (6) contains only components that are known.

Once D has been found one can put it into Eq. (3) to get a', which specifies the point at which the beam reflects or refracts from the optic's surface, a'. To proceed, one takes the vector a' to be the new vector a in which the beam proceeds from along the new direction vector found from Eq. (1). It is easy to see that this technique can be

applied over and over until the beam exits the interferometer.

Derivation of Fringe Pattern

Any wavefront splitting interferometer, such as the Collimation Tester described in Appendix A, produces an interference pattern by generating both shear and a tilt between the exiting beams. For convenience the tilt and shear are broken into horizontal and vertical components labeled by θ, vertical tilt; ξ, horizontal tilt; *vs*, vertical shear; and *s*, horizontal shear. The derivation of the interference fringe pattern is done in the same manner that Riley and Gusinow derived their fringe pattern except that this thesis includes two more parameters, ξ and *vs*, which will be needed for the discussion of the CSI in the next section.

Fig. 4 shows the two interfering beams exiting an interferometer; each is labeled with its own coordinate system. The two beams have scalar electric fields for a uniform spherical wavefront given by

$$\begin{aligned} E_1 &= E_0 \exp\left(-\frac{i}{2R(z)}(x^2+y^2) - ikz\right) \\ E_2 &= E_0 \exp\left(-\frac{i}{2R(z')}(x'^2+y'^2) - ikz' - i\beta\right), \end{aligned} \tag{7}$$

where $R(z)$ is the radius of curvature, k is the propagation number $2\pi/\lambda$, and β is a constant phase shift. It will be shown later that β is π for the CSI because one beam undergoes an internal reflection whereas the counter-rotating beam does not have any internal reflections. $R(z)$ can be treated as a constant in the typical case when the incident radius of curvature is much larger than the path dimensions in the interferometer. The transformation between coordinate systems is given by

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\xi & 0 & \sin\xi \\ 0 & 1 & 0 \\ -\sin\xi & 0 & \cos\xi \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} s \\ vs \\ (-)D \end{pmatrix}, \tag{8}$$

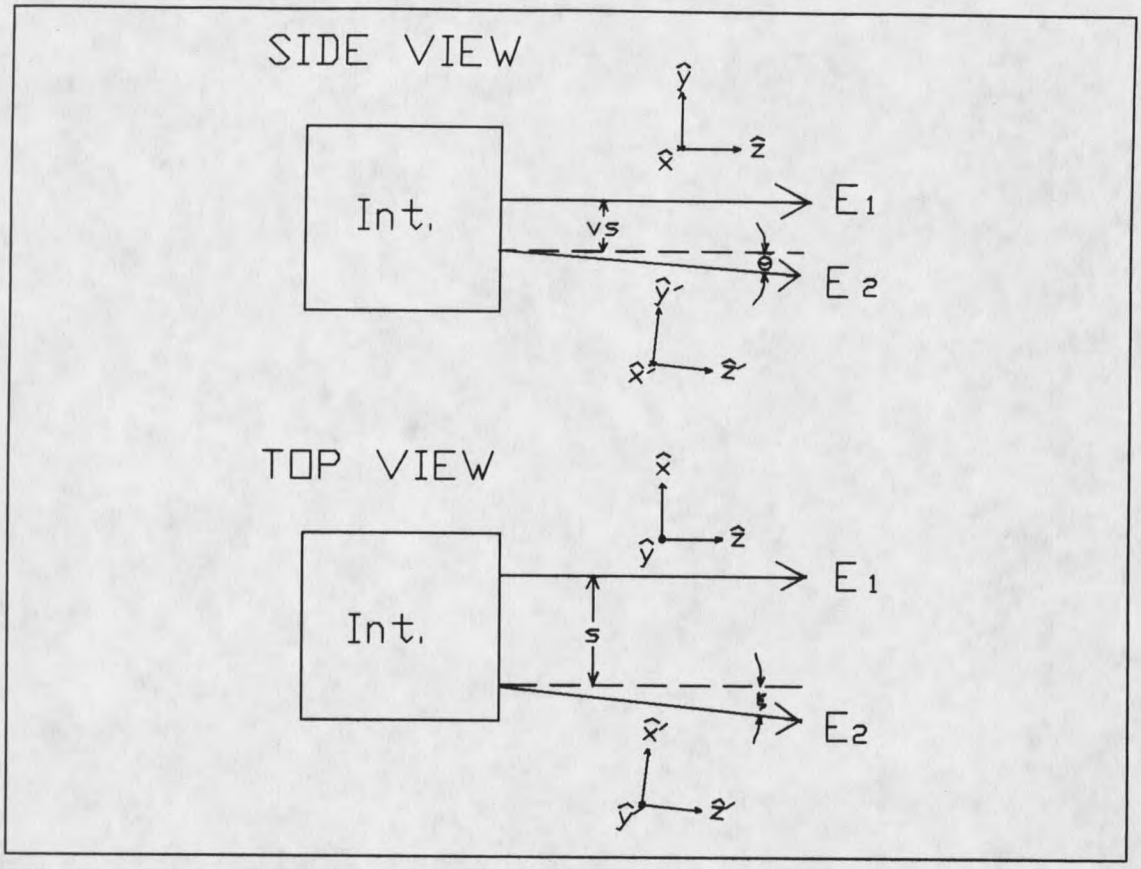

Figure 4. The side and top view of two rays exiting a generic interferometer with a coordinate system associated with each ray. E_1 and E_2 are the electric fields associated with the spherical wavefronts of the two rays. θ and ξ are the vertical and horizontal tilt components between the two rays, and vs and s are the vertical and horizontal shear components between the two rays.

where D is the extra distance traveled in one path of the interferometer. For small angles this becomes

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & \xi \\ 0 & 1 & 0 \\ -\xi & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \theta \\ 0 & -\theta & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} s \\ vs \\ (-)D \end{pmatrix}. \tag{9}$$

By keeping only linear, small angle terms, the derivation of the fringe pattern is easier and eliminates the worry of the transformation matrix order.

Since the total interfering electric field is given by

$$E_t = E_1 + E_2, \tag{10}$$

the field intensity will be

$$I = E_t^* E_t . \tag{11}$$

An equation for the intensity pattern is found by changing the primed coordinates in E_2 to unprimed coordinates given by Eq. (9) then plugging E_1 and E_2 into Eq. (11). Neglecting the multiplicative constants, the intensity pattern can be simplified to

$$I = \sin^2\left(\frac{k}{2R}\left[x(s - \xi L + \xi R) + y(vs - \theta L + \theta R)\right] + \frac{\Phi_0}{2}\right), \tag{12}$$

where the interference intensity pattern was evaluated at a distance $z = L$ from the interferometer. Again for large radius of curvature

$$R(z = L) = R(L + D) = R, \tag{13}$$

where R is essentially a constant through the path of the interferometer. Φ_0 is a constant phase and has no effect on the fringe pattern.

A typical fringe pattern is shown in Fig. 5. The vertical distance between fringes, d, is found by finding points on the y-axis which have a phase difference of π. Using Eq. (12) this can be shown to be

$$d = \frac{\lambda}{\theta\left(1 - \frac{L}{R}\right) + \frac{vs}{R}}. \tag{14}$$

The slope of the fringes, ϕ, are the lines of constant phase of the intensity formula Eq. (12), that is, $y/x = \tan\phi =$ constant. Using the definition for d, the slope of the fringes can be found from Eq. (12) yielding

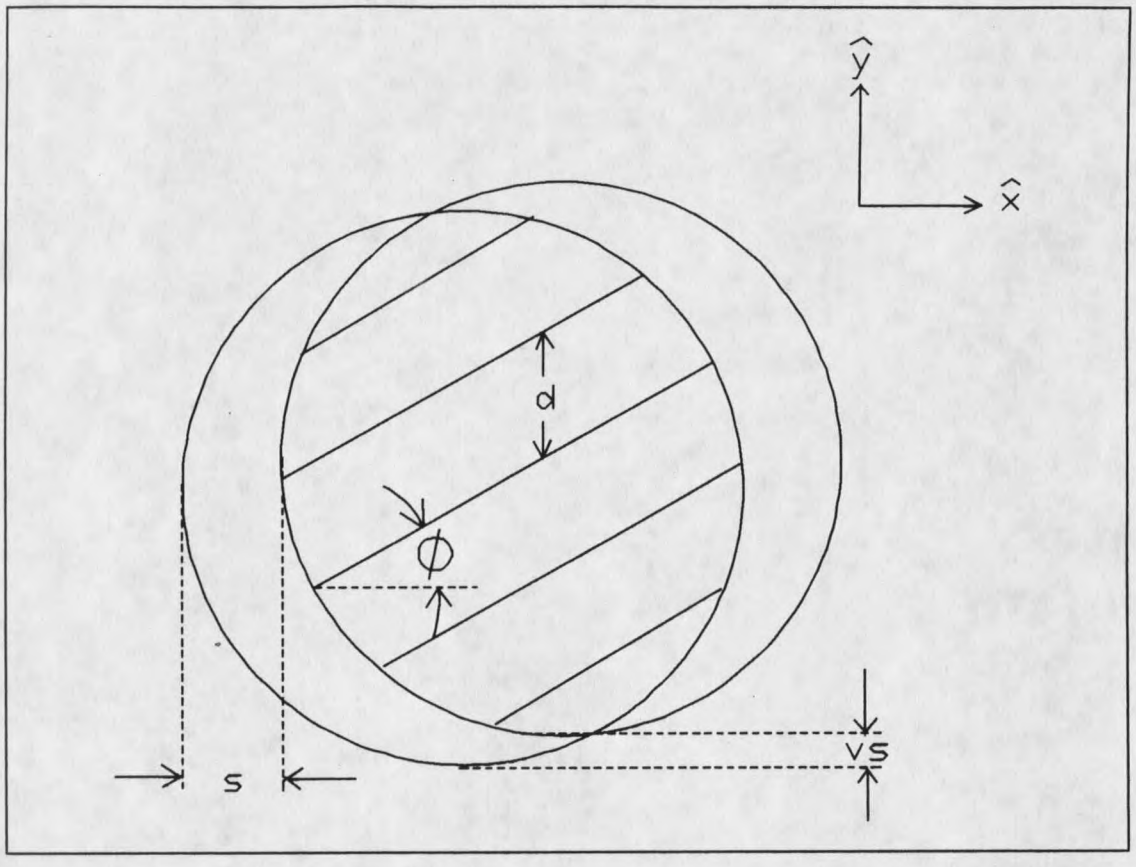

Figure 5. A schematic of a typical fringe pattern showing the various important physical parameters. *s* and *vs* are the horizontal and vertical shears and *d* is the vertical distance between the fringes. ϕ is the horizontal tilt of the fringes.

$$\tan\phi = \frac{sd}{\lambda R} + \frac{d\xi}{\lambda}\left(1 - \frac{L}{R}\right). \tag{15}$$

Note that the radius of curvature, R, could not be easily found, because of the existence of the horizontal tilt, ξ . This small angle cannot be easily measured with an acceptable amount of accuracy, but can be ignored when $\xi \ll s/R$. For the collimation tester this is the usual case, so ξ is assumed to be zero in Eq. (15) and the radius of curvature can be solved for yielding

$$R = \frac{sd}{\lambda \tan\phi}. \tag{16}$$

Typically, one measures ϕ, s, and d experimentally so that the radius of curvature can be determined from Eq. (16) for a known central wavelength. It will be shown later that ξ will not always be negligible for the CSI so a scheme will be devised to eliminate ξ so that the CSI can be used with greater accuracy.

Cyclic Shearing Interferometer

The CSI is shown in full detail with the beam direction vectors in Fig. 6, with the exception of the infinitely thin beam splitter. The assumption of a thin beam splitter makes the calculation of the beam direction vectors much easier and leads to more readable equations. The effect of a real beam splitter will be examined in the discussion. The ray tracing method described at the first part of this chapter was used to find the following beam direction vectors.

$$\begin{aligned}
S_0 &= \hat{x} \\
S_{1c} &= \hat{x} \\
S_{2c} &= (1-2\cos^2\gamma\cos^2\frac{\pi}{8})\hat{x} - \cos^2\gamma\sin\frac{\pi}{4}\hat{y} - \sin 2\gamma\cos\frac{\pi}{8}\hat{z} \\
S_{3c} &= \sin^2\gamma\sin\frac{\pi}{4}\hat{x} + (\cos^2\gamma - \sin^2\gamma\sin\frac{\pi}{4})\hat{y} - \sin 2\gamma\cos\frac{\pi}{8}\hat{z} \\
S_{4c} &= S_{3c} \\
S_{1cc} &= \sin^2\psi\hat{x} - \cos^2\psi\hat{y} - \sin 2\psi\cos\frac{\pi}{4}\hat{z}
\end{aligned} \tag{17}$$

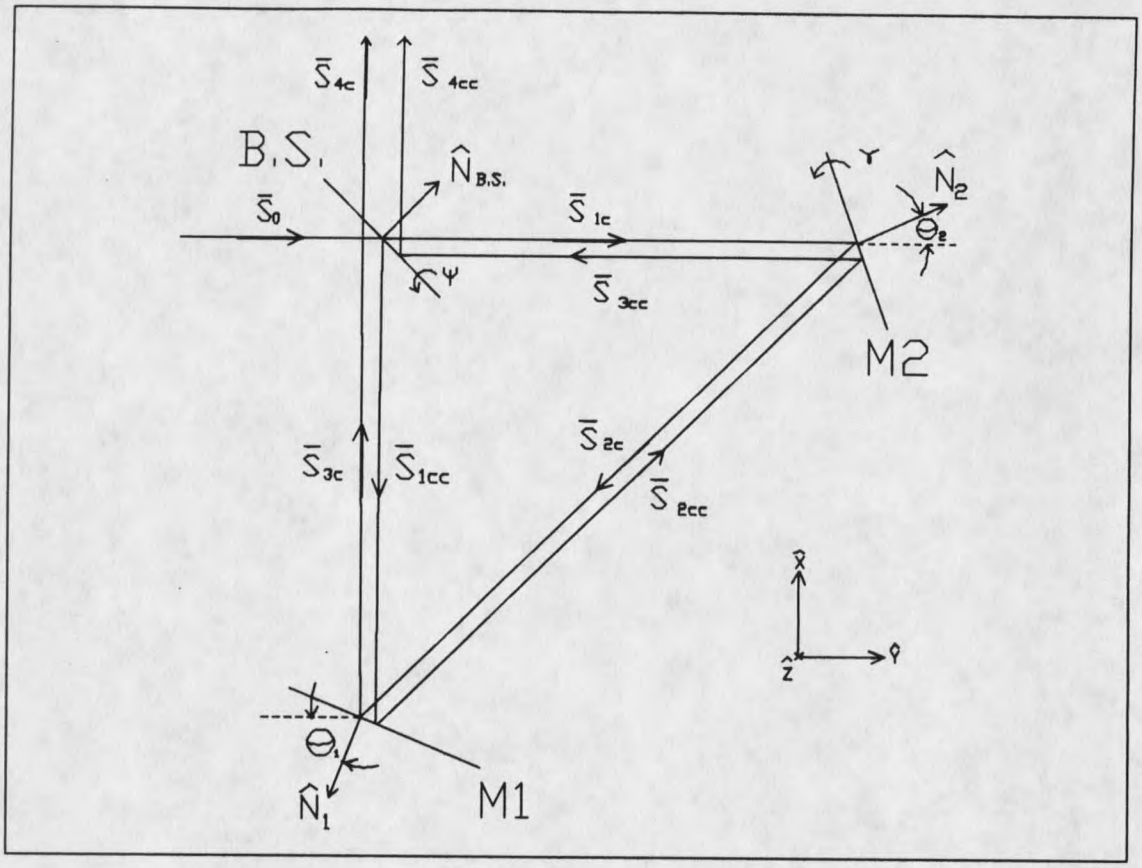

Figure 6. CSI in full detail showing all of the beam direction vectors, optical surface normals, and important alignment angles.

$$\boldsymbol{S}_{2cc} = \frac{\sqrt{2}}{2}\hat{x} + \frac{\sqrt{2}}{2}\cos 2\psi\,\hat{y} - \frac{\sqrt{2}}{2}\sin 2\psi\,\hat{z}$$

$$\begin{aligned}\boldsymbol{S}_{3cc} = {} & (-1+(1+\frac{\sqrt{2}}{2})\gamma^2 + 2.61313\gamma\psi + \psi^2)\hat{x} + \\ & (\frac{\sqrt{2}}{2}\gamma^2 + 1.08239\gamma\psi - \psi^2)\hat{y} + \\ & (-1.84776\gamma - \sqrt{2}\psi + 2.82843\gamma^2\psi + \\ & 1.08239\gamma\psi^2 + 1.23184\gamma^3 + .942809\psi^3)\hat{z}\end{aligned}$$

$$\begin{aligned}\boldsymbol{S}_{4cc} = {} & (-\frac{\sqrt{2}}{2}\gamma^2 + 1.53073\gamma\psi + 2\psi^2)\hat{x} + \\ & (1 - (1+\frac{\sqrt{2}}{2})\gamma^2)\hat{y} + \\ & (-1.84776\gamma + 1.23184\gamma^3 - .585786\gamma^2\psi - \\ & .448342\gamma\psi^2 + 2.82843\psi^3)\hat{z}\end{aligned} \qquad (17_{cont})$$

In Eq. (17), S_{Ncc} refers to the Nth counter-clockwise direction vector, and S_{Nc} refers to the Nth clockwise direction vector. Some of the direction vectors were reduced to third order in small angles in order to have manageable equations in which the dependence on various angles is manifest.

The vertical tilt, θ, between the two exiting beams is found by subtracting the angles in which the two exiting beams make with the x-y plane. These angles can be found by comparing the z components of the two exiting beams. Assuming small angles, the vertical tilt is found to be

$$\theta \approx \sin\theta = 2\sqrt{2}\,\psi^3 + (-2+\sqrt{2})\gamma^2\psi + \gamma\psi^2\left((2\sqrt{2}-4)\cos\frac{\pi}{8} + (4\sqrt{2}-4)\sin\frac{\pi}{8}\right). \tag{18}$$

The horizontal tilt, ξ, between the two exiting beams is found by computing the difference between the angles the beams make with the y-axis in the x-y (horizontal) plane. Note that both beams lie very close to the y axis which allows us to make small angle approximations. The horizontal tilt is

$$\xi = \frac{\gamma^2}{\sqrt{2}} \Bigg/ \left(1 - \left(1+\frac{\sqrt{2}}{2}\right)\gamma^2\right) - \left((-)\frac{\sqrt{2}}{2}\gamma^2 + 2\psi^2 + \gamma\psi\left(2\cos\frac{\pi}{8} + 2\sin\frac{\pi}{8}\left(1-\frac{\sqrt{2}}{2}\right)\right)\right) \div \left(1 - \left(1+\frac{\sqrt{2}}{2}\right)\gamma^2 + \gamma\psi\left(2\cos\frac{\pi}{8}\left(1-\frac{\sqrt{2}}{2}\right) + 2\sin\frac{\pi}{8}\right)\right). \tag{19}$$

When deriving the fringe pattern, specifically Eq. (15), it was shown that the horizontal tilt, ξ, must be eliminated in order to use a shearing interferometer to measure the radius

of curvature. ξ can be eliminated by using a relationship between ψ, the tilt on the beam splitter, and γ the tilt on mirror #2. This relationship is established by setting $\xi = 0$; the two roots are

$$\psi = \gamma\left[\sin\frac{\pi}{8}\left(\frac{\sqrt{2}}{2} - \frac{1}{2}\right) - \frac{1}{2}\cos\frac{\pi}{8} \pm \frac{1}{2}(2+\sqrt{2})^{1/2}\right]. \tag{20}$$

The positive root corresponds to the condition of $\theta = 0$, which gives the unusable setup with fringes that have an infinite distance between them.

The negative root, which corresponds to $\xi = 0$ and $\theta \neq 0$ is the desired root which gives a usable fringe pattern. The negative root is

$$\psi = -\gamma(\sin\frac{\pi}{8} + \frac{1}{2}(2+\sqrt{2})^{1/2}) \approx -1.306\gamma. \tag{21}$$

There will not be any horizontal tilt introduced as long as the interferometer is operated utilizing the relationship given above. An experimental method for insuring that Eq (21) is satisfied is presented in the "Alignment of the CSI" section of Chapter 3. Since normal operation will require that the angles of the CSI will follow the above relationship, it can be substituted into the equation for the vertical tilt, Eq. (18), to give the simple relationship

$$\theta = 2\sqrt{2}\,\psi^3. \tag{22}$$

The vertical shear, *vs*, is naturally created in the CSI and does not play a role in determining the radius of curvature of a beam; however, it does pose a problem when it is about the same size as the beam's diameter. This problem will be examined in "Special Considerations" section of Chapter 4.

The shear, *s*, can be produced by three different methods. The first two methods

involve translating an optic element and the last involves rotating both mirrors. All methods work equally well; it is a matter of convenience for the user which one to choose.

The first method produces shear by translating the beam splitter perpendicular to its optic face as shown in Fig. 7. Simple trigonometry shows that

$$s = \sqrt{2}\,P, \tag{23}$$

where P is the amount of translation.

The second method is similar to the first except that M1 is translated perpendicular to its optic face as shown in Fig. 8. Again, simple trigonometry shows that

$$s = 4P'\sin\frac{\pi}{8}. \tag{24}$$

where P' is the translation of the mirror.

The third method involves rotating mirrors #1 and #2 about the vertical as shown in Fig. 9. The shear is found by tracing the beams around the CSI and finding their horizontal displacement on the x-axis. The shear, s, is found to be

$$s = (2+\sqrt{2})L\left(\sin\left(\frac{\pi}{4}+2\beta\right)\left(1+\tan\left(\frac{\pi}{8}+\beta\right)\right)-\cos\left(\frac{\pi}{4}+2\beta\right)\left(1+\cot\left(\frac{3\pi}{8}+\beta\right)\right)\right), \tag{25}$$

where β is the rotation angle and L is the length of the short leg. This method requires care to make sure both mirrors are rotated the same amount, otherwise horizontal tilt would be introduced.

One very important consideration is the pathlength difference inherent in the CSI. The ability to measure R of a broadband beam requires that the pathlength difference

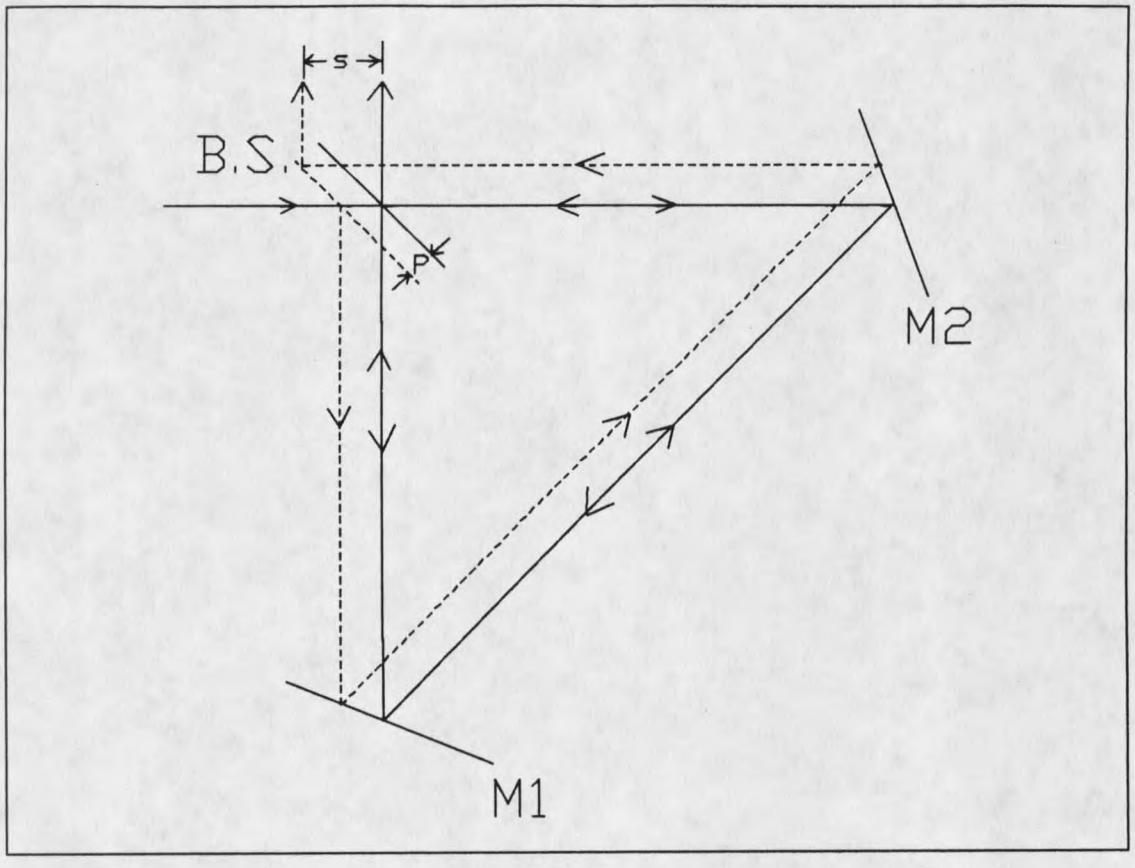

Figure 7. A method of producing horizontal shear, s, is to translate the Beam Splitter perpendicular to its optical face a distance P.

between the two exiting beams must be smaller than the coherence length of the light source. The pathlength difference between the two exiting beams is found by using the vector tracing method described in the "Ray Tracing Method" section of Chapter 2. The calculation was done assuming that the CSI was set up in the right triangle configuration shown in Fig. 6. The origin, (0,0,0), was chosen to be where the incident beam strikes the beam splitter. Mirror #1 has one fixed point at $P_{0_1} = (0,-L,0)$ and Mirror #2 has a fixed point at $P_{0_2} = (L,0,0)$. The beam direction vectors are given in Eq. (17). By tracing a beam through the CSI one can find the distance it travels to intercept the viewing plane. Once the pathlength of each beam is known, subtracting the two will

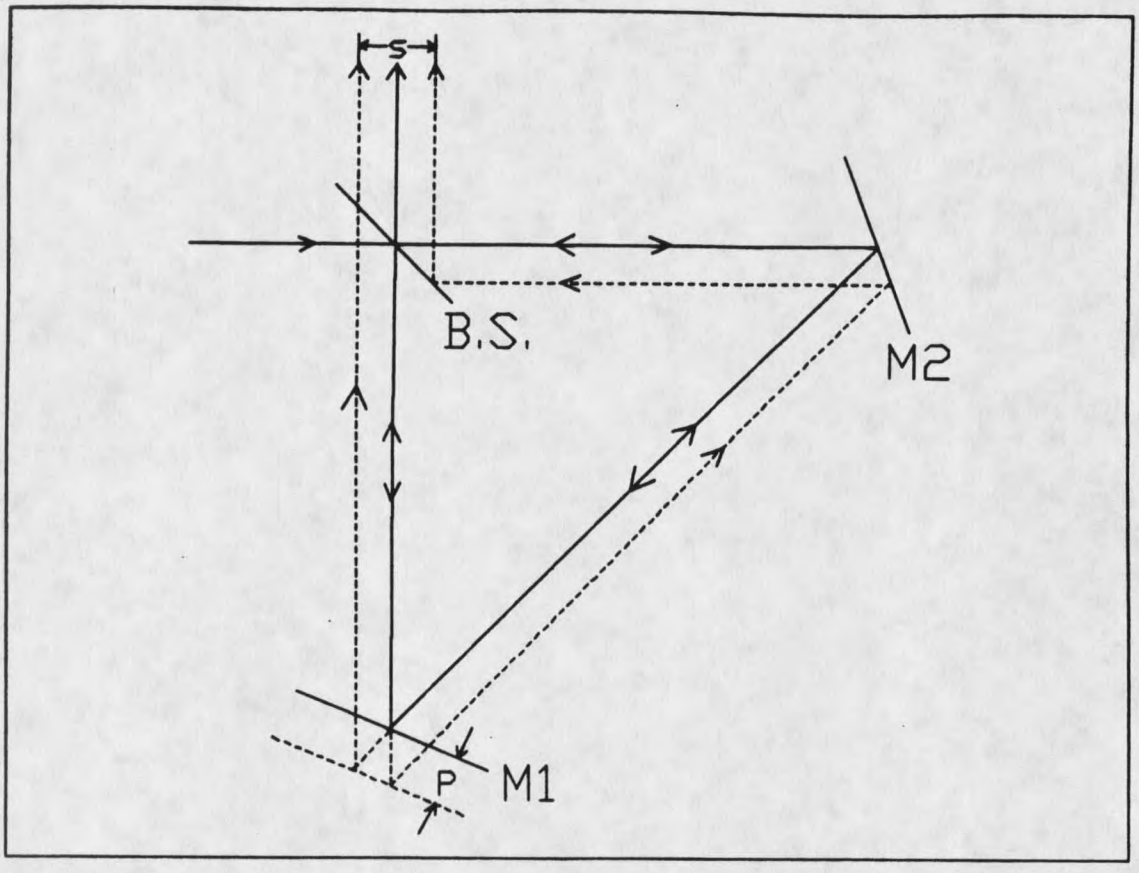

Figure 8. A method of producing horizontal shear, s, by translating one of the mirrors perpendicular to its optical face a distance P'.

yield the pathlength difference, ΔL . The alignment of the viewing screen for the purposes of this calculation is shown in Fig. 10. Note that the screen has its normal parallel to S_{4c} and a known point, $P_{0_{SCREEN}}$. $P_{0_{SCREEN}}$ is where the counterclockwise beam strikes the beam splitter immediately before it exits the CSI. Due to the lengthy calculations the work was done on a computer to give the numerical relationship

$$\Delta L = 2.6678 L \psi^2. \tag{26}$$

γ was eliminated by using the relation given in Eq. (21) which is necessary to operate

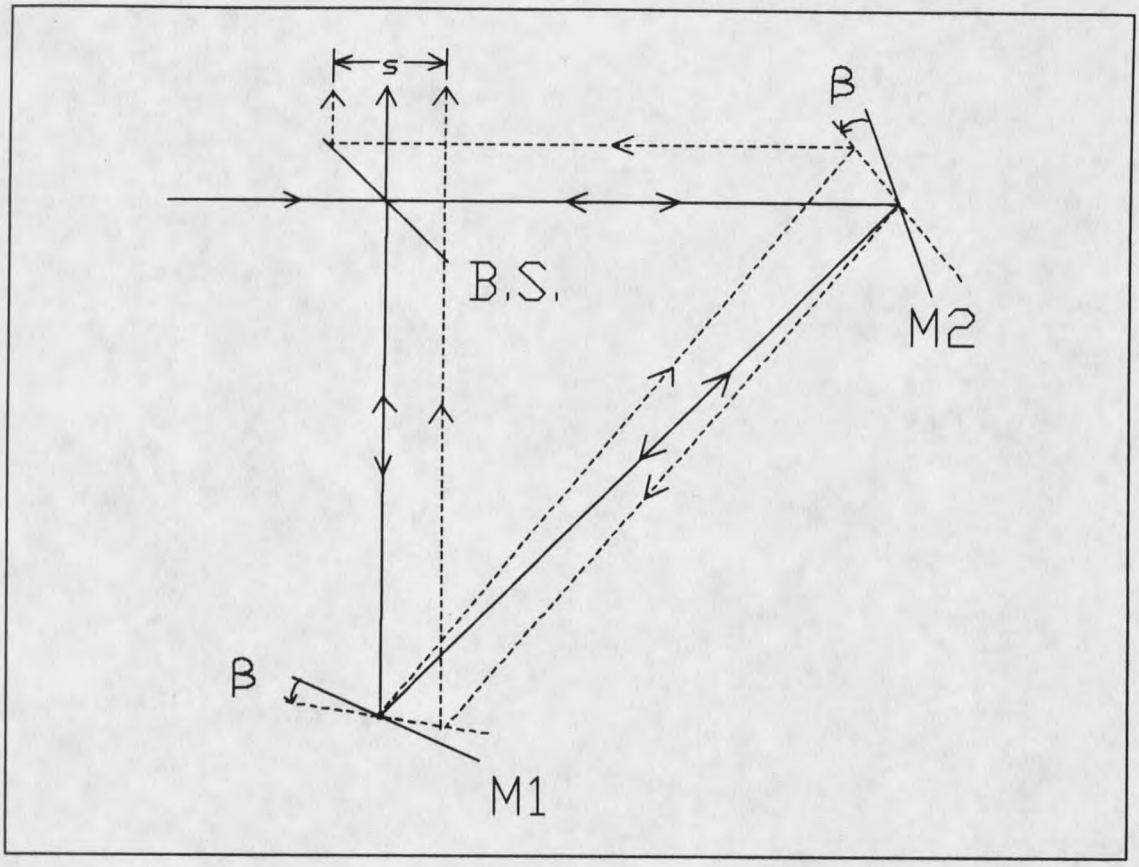

Figure 9. A method of producing horizontal shear, *s*, by rotating both mirrors an amount β in the direction shown above.

the CSI without horizontal tilt.

It will be shown in Chapter 3 that the pathlength difference introduced in the CSI is about 170 times less than the pathlength difference introduced in the Collimation Tester which is described in Appendix A. This big difference will allow a user to measure the radius of curvature for beams that have coherence lengths 170 times smaller than the smallest coherence length beam which will produce a useable fringe pattern from the Collimation Tester.

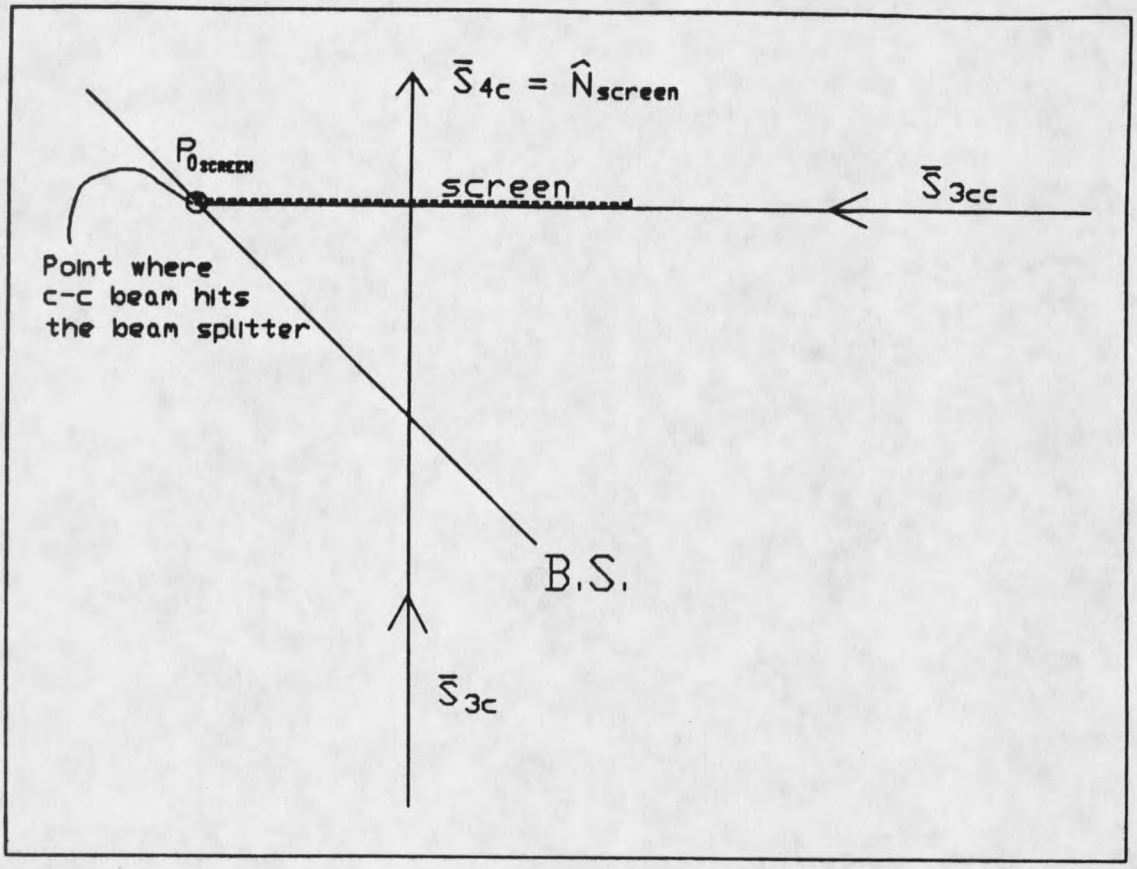

Figure 10. Alignment of the viewing screen for the purpose of computing the pathlength difference between the two exiting beams. The screen's normal, N_{screen}, is parallel to S_{4c} and one point on the screen's plane, $P_{0\,screen}$, is the point where the counter-clockwise beam strikes the Beam Splitter.

CHAPTER 3

EXPERIMENTAL

The CSI must be calibrated with a beam known to be collimated since the CSI has several parameters, such as tilts and positions of the optics, that must be properly set for its operation. The reference beam that was used was a He-Ne beam which was collimated after it was expanded from a spatial filter. The reference beam's collimation was checked with a Collimation Tester (see Appendix A). After the CSI is aligned with the reference beam, described in detail in the next section, it will operate the same with any other incident beam, with the exception of the fringe spacing which depends upon the wavelength (see Eq. (14)).

Alignment of CSI

The alignment procedure is fulfilled in two segments. The first part is to align the CSI with all optics perpendicular to the table as shown in Fig. 1. When the optics are aligned so that the collimated, counterrotating, reference beams lie on top of each other, an interference pattern that does not have any fringes is produced. This pattern without

fringes is called a null pattern which implies that the two collimated beams exiting the CSI have no tilt between them. The operator must be sure that there is a null pattern not just very finely spaced fringes.

The second part of the alignment scheme is to adjust the interferometer so that the appropriate fringe pattern will be formed. Using a collimated He-Ne beam, the tilt between the exiting beams, and thus the number of fringes visible, is set by simply varying ψ. However to avoid horizontal tilt, γ must be varied at the same time to satisfy Eq. (21). This is easy to do experimentally by varying γ to keep the fringes horizontal for the collimated reference beam. The sensitivity, the amount that the fringes tilt when the radius of curvature is changed, can be adjusted by changing the amount of shear by any of the techniques discussed in the "Cyclic Shearing Interferometer" section of Chapter 2 (Eq. (23)-(25)).

Experimental Setup

The experimental setup used to test the CSI is shown in Fig. 11. The distance L between the beam splitter and the two mirrors was $8cm$. The typical vertical distance d between the fringes produced with the collimated, He-Ne test beam is $1cm$. A simple calculation using Eq. (14), Eq. (22), and d given above tells us that $\psi = 28mr$. Using Eq. (26), and ψ and L given above, we find that the pathlength difference between the two exiting beams was $\Delta L = 170\mu m$. This corresponds to a temporal difference of $\Delta t = \Delta L/c = 564fsec$ which implies that a beam with a coherence time greater than $564fsec$ will produce a fringe pattern with good visibility. It is interesting to note that a standard

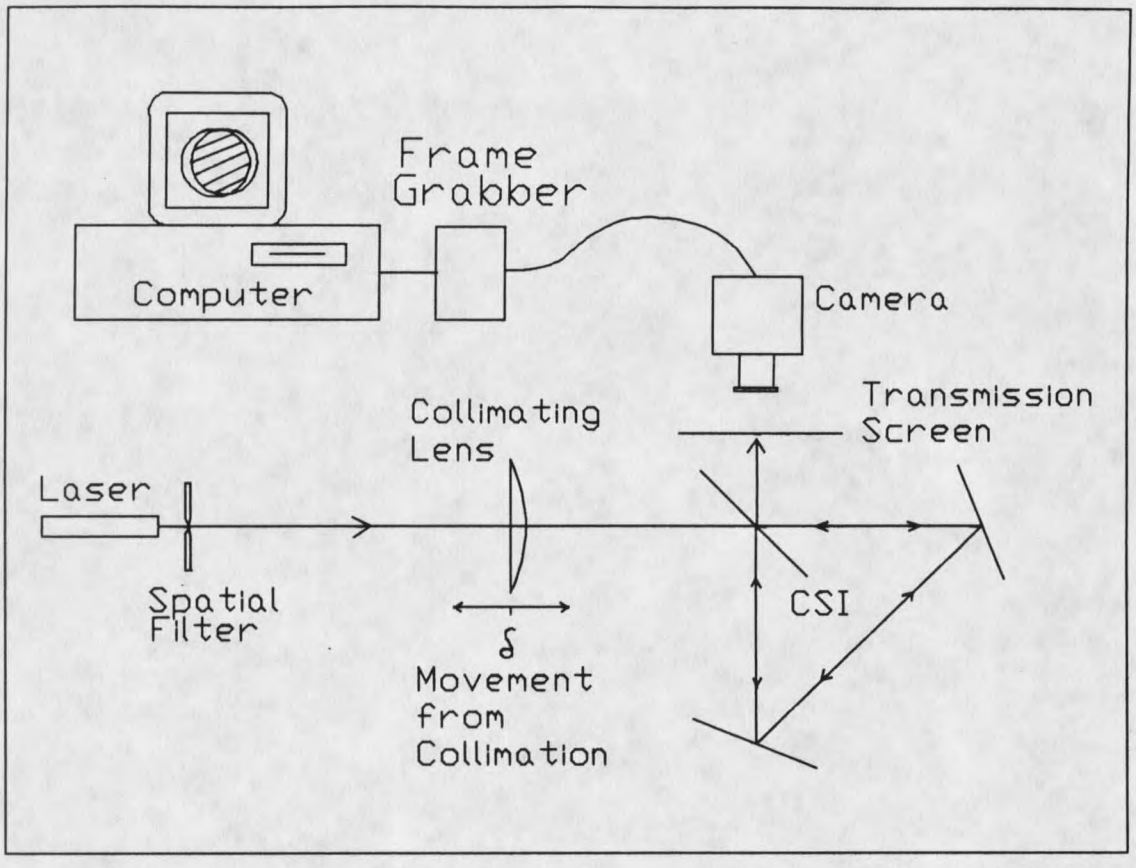

Figure 11. Experimental setup used to test the CSI. The Radius of Curvature of the incident wavefront was changed by moving the collimating lens a distance γ. The camera, frame grabber, and computer were used to acquire, store, and analyze the fringe patterns.

Collimation Tester operated at 45° has a much larger pathlength difference at $\Delta L = 2.9cm$. It will be shown in Chapter 4 that a compact CSI will have a pathlength difference of only $61\mu m$.

The radius of curvature of the beam incident on the CSI is changed by moving the collimating lens along the beam's axis. The theoretical radius of curvature is found from the Gaussian Lens Formula, $1/f = 1/s_0 + 1/s_i$, with $s_o = f+\delta$, where δ is the displacement

of the collimating lens from collimation along the beam's axis and f is the focal length of the lens. The radius of curvature is simply the image distance, s_i, minus the distance between the lens and the screen where the fringe pattern is produced. The latter distance is usually very small compared to R, therefore it can be ignored in most experimental situations. R is found by using a binomial expansion and solving for s_i. We find that

$$R_{theory} \approx s_i \approx \frac{f^2}{\delta}. \tag{27}$$

The laser shown in Fig. 11 was a XeCl Eximer which lases at $308nm$ with a linewidth[19] of 1 cm^{-1}. A linewidth of one wave number corresponds to a coherence time of $15psec$ (FWHM). The spatial filter produces a gaussian beam which is then passed through a collimating lens with a focal length measured to be $2028 \pm 4mm$ using a method described by Malacara[20]. After the beam passed through the CSI it produced a visible pattern on a thin sheet UV converter. The visible fringe pattern was then stored in computer memory for analysis by using a video camera and frame-grabber. The stored frames were analyzed with the help of the computer and a least-squares fitting program to find the parameters s, d, and $tan(\phi)$, R_{exp} can be found from Eq. (16) once these parameters are determined. The method used to analyze the fringe patterns to find d and $tan\phi$ is discussed in Appendix B; the next section of this chapter will discuss how to find s.

The visibility of the fringes was very nearly 1 signifying that the coherence length of the laser was much longer than the temporal difference produced by the CSI between

the two exiting beams. This result was expected because the coherence length of the laser was about $15psec$ which is much longer than the temporal difference between the two exiting beams $\Delta t = 565fsec$. In contrast, no fringes were produced by a Collimation Tester because its pathlength difference ($\Delta t \geq 96ps$) is larger than the coherence time of the laser.

Data Taking Procedure

The three phases of taking data, calibration, acquiring fringes, and analyzing the fringes, could begin only after the experiment was assembled as shown in Fig. 11. The calibration was needed to determine the horizontal and vertical calibration constants, the aspect ratio, and the shear produced in the CSI. The horizontal and vertical calibration constants, C_H and C_V, were needed to determine actual distances, like d, from the stored frames, thus they have units of meters per pixel (m / Pix). They are determined by simply acquiring a picture of a piece of graph paper of known rulings and dividing the distance between lines by the number of computer pixels separating the lines. This is done by using the calibration option in the frame grabber / camera controlling program "MAIN_CNTL" (See Appendix C). The aspect ratio, AR, is easily determined from the calibration constants by $AR = C_V/C_H$. The aspect ratio is a multiplicative constant used to multiply the apparent slope of the captured fringes to find the actual slope. Finally, the shear was determined by acquiring a frame of the two images produced by placing a fine wire in the incident collimated beam. The shear is simply the number of pixels separating the two images of the wire multiplied by the horizontal calibration constant C_V.

The actual data taking consisted of changing δ by a known amount and capturing the corresponding fringe pattern into computer memory with the video system. The actual video snapping was coordinated with the firing of the pulsed XeCl laser by controlling the frame grabber with a trigger from the laser (See Appendix C). Ten frames of different $\delta's$, from $2.5mm$ to $25mm$, were acquired to compare to theory.

The several step analysis used to determine the radius of curvature and its uncertainty from the acquired frames is described in Appendix B.

Results

The theoretically computed R_{theory} (Eq. (27)) is plotted with the experimentally determined R_{exp} points (Eq. (16)) as a function of delta in Fig. 12. The error bars for the experimental points come from the uncertainty in measuring s, d, and $tan(\phi)$ (See Appendix B). As seen in Fig. 12, all of the points lie within their experimental uncertainty of the theoretical curve which indicate that the agreement between theory and experiment is excellent.

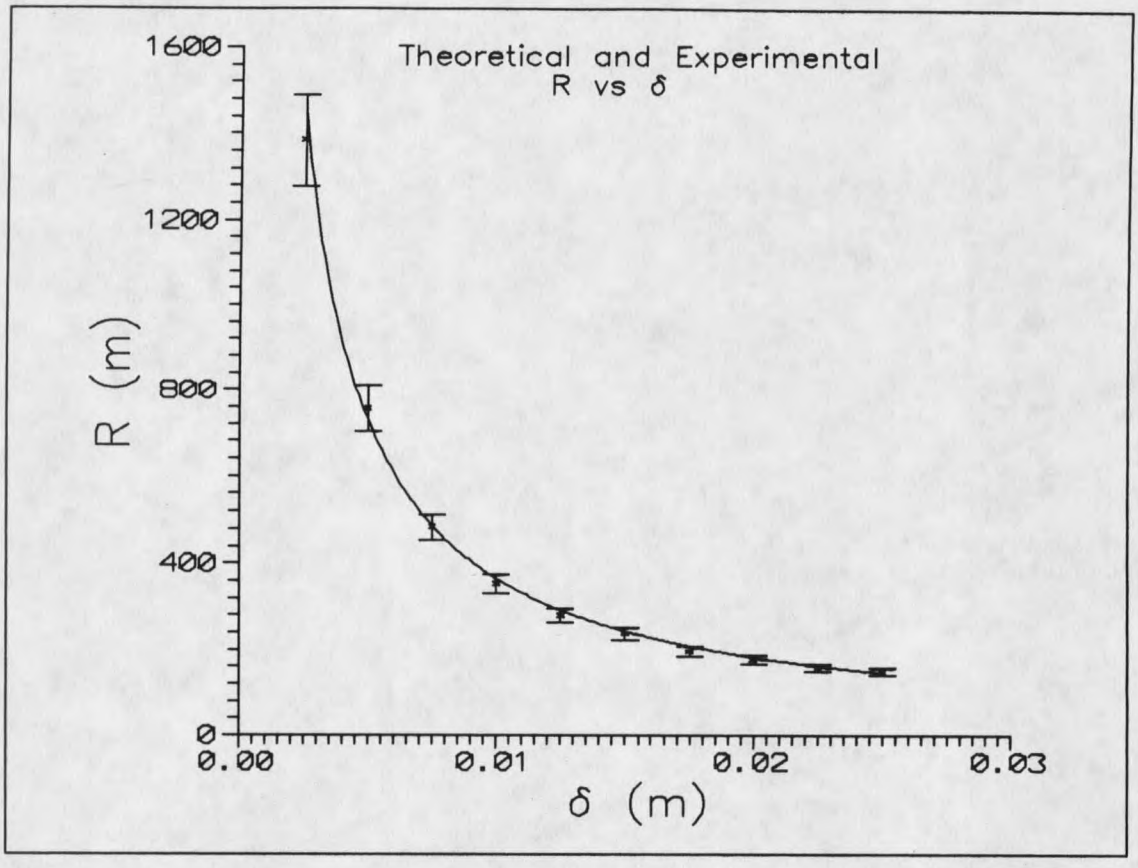

Figure 12. Experimental and theoretical Radius of Curvature results plotted against the displacement, δ, of the collimating lens from its collimation point.

CHAPTER 4

DISCUSSION

Special Considerations

The theoretical analysis of the CSI was done assuming that the beam splitter was infinitely thin. In practice this is not true since real beam splitters have both thickness and a wedge angle between the two surfaces. The nonzero thickness of the beam splitter in itself does not introduce any new tilts on the interfering beams. However, the effect of the nonzero thickness coupled with a wedge does produce some tilts. A wedge that is orientated so that it deflects a beam vertically will simply affect the vertical tilt between the two exiting beams and hence the vertical shear. A wedge that is orientated in the horizontal plane will produce a horizontal tilt that will have to be counteracted by adjustments to γ or ψ. The effect of a wedge orientated in the horizontal plane is small for typical amounts of wedge since the horizontal tilt goes like the wedge angle squared. It should be noted that the effects of the wedge are small in any orientation and a small amount of adjustment on γ and ψ can counteract their effects.

Another effect of a real beam splitter is a small secondary reflection off its back

surface which will produce lighter, secondary fringes that may confuse the operator when setting up the CSI or hamper reading the fringe pattern. The secondary fringes can be distinguished from the desired fringes since they will rotate if the beam splitter is rotated. These fringes can be eliminated by having either a very good anti-reflection coating on the back surface, or having a beam splitter with a wedge less than 1/2 fringe.

We found that one can measure the collimation of a laser beam that has a diameter down to $1.5cm$ without special care being taken. However, when measuring the colimation of a beam with a diameter between $1.5cm$ and $.5cm$ one must reduce the amount of shear, which is usually about $1cm$, so that the beams will overlap to form a fringe pattern. For small diameter beams one may also want to increase the tilt between the exiting beams to reduce the spacing between the fringes. However, the effect of reducing the shear and the spacing between the fringes will decrease the sensitivity of the interferometer. The relation between the slope of the fringes and the radius of curvature can be seen in Eq. (16). When s and d are made smaller for a given R the slope of the fringes, $tan\phi$, will be smaller. This reduced sensitivity will increase the uncertainty in R.

Vertical shear may be of concern if it is about the same size as the incident beam's diameter. This might happen if the cyclic pathlength in the CSI is large, or if beams of small diameters are used. There are three ways to reduce or eliminate the vertical shear. The first method is to simply reduce the size of the legs of the CSI which will reduce the beams' pathlength, in turn reducing the vertical shear. This will reduce but not eliminate *vs*. The next method is to use a beam splitter with a wedge orientated

so that it deflects a beam vertically which would counteract the *vs* produced by the normal operation of the CSI. The final method that can be used is to place a window in the long leg of the CSI as shown in Fig. 13. Vertical shear can be varied if the window is rotated about an axis that is in the horizontal plane perpendicular to the beam's axis. All of the methods described above can be used separately or together to reduce or eliminate the vertical shear.

Compact Design

A compact one-piece design of the CSI, such as the one shown in Fig. 14, would provide the user with two distinct advantages over a CSI constructed out of several pieces of optics. The first advantage is the smaller size of the compact unit. A smaller CSI implies a smaller pathlength difference between the two exiting beams (Eq. (26)) which means that the smaller CSI would produce usable fringe patterns with beams that have a shorter coherence length. A compact unit that is built to accommodate a $.75cm$ beam will have a pathlength difference of $\Delta L = 90\mu m$ ($\Delta t = 300fsec$). This implies that a beam with a coherence time greater than $300fsec$ will produce a usable fringe pattern with two fringes.

The second advantage is one of convenience. A one piece CSI would allow the user to simply place the CSI into a beam line to measure the radius of curvature. The sensitivity of the CSI could be adjusted by simply rotating it. The sensitivity changes with a rotation about a vertical axis with changes of shear.

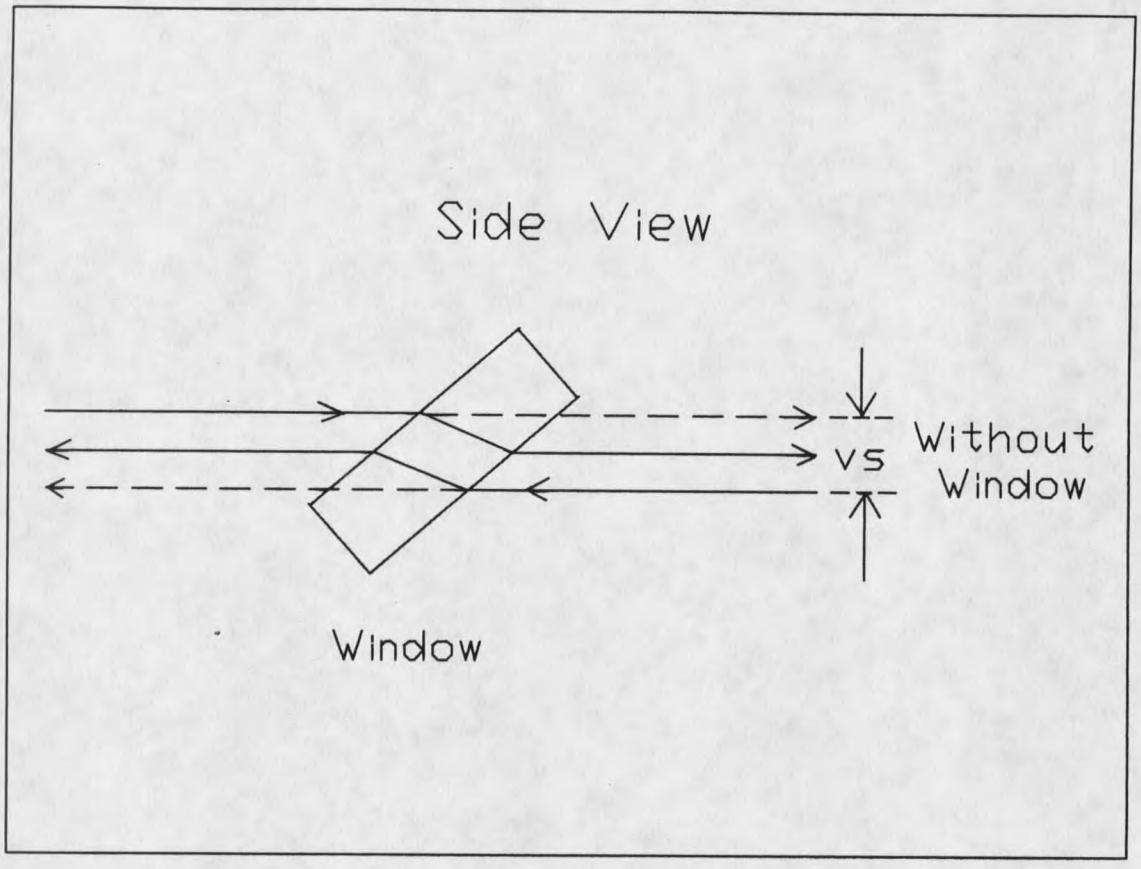

Figure 13. Use of a window in the long leg of the CSI to eliminate *vs*.

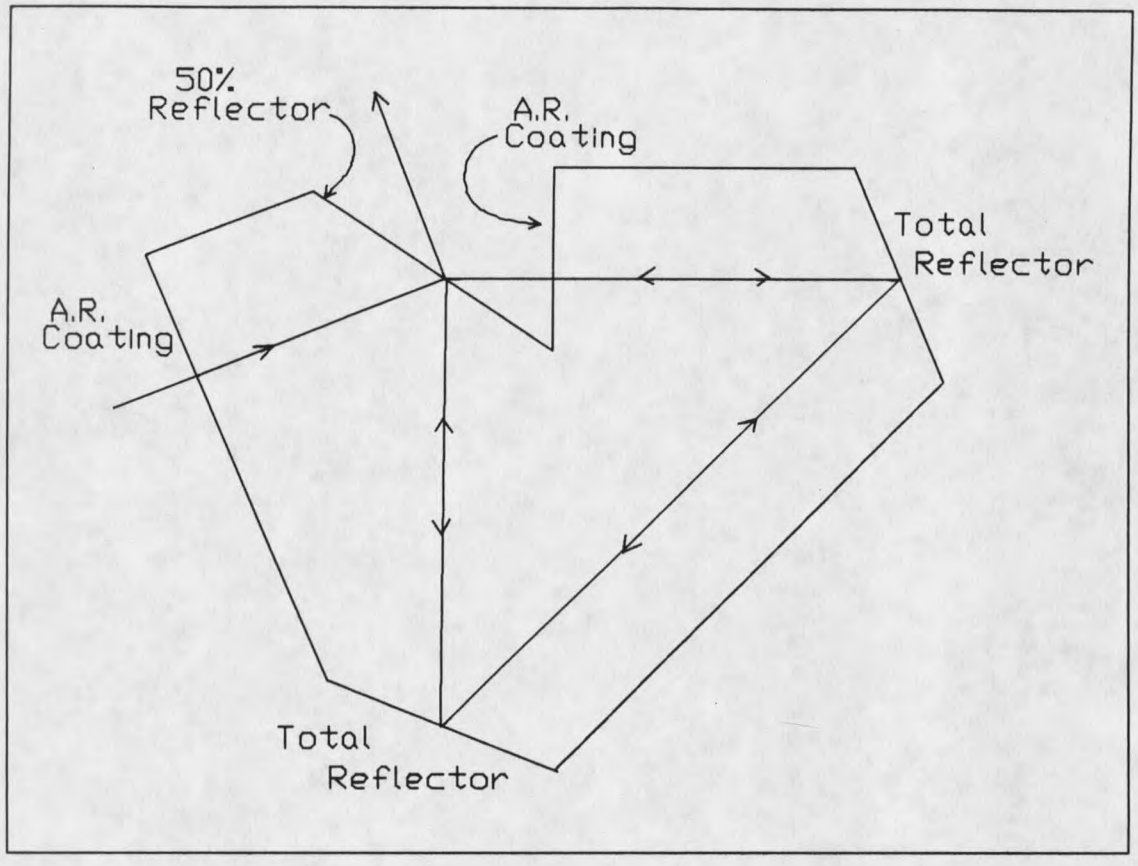

Figure 14. A compact, one-piece design of the CSI. The tilts of the of the reflecting surfaces must be ground to resemble a CSI made from separate pieces.

CHAPTER 5

CONCLUSION

A survey of interferometers showed that, up until now, there was not an accurate method that could be used to collimate or, more specifically, measure the radius of curvature of a short-pulsed or broad-band laser. After deriving the fringe pattern from a generic wavefront splitting interferometer, a novel ray tracing method was applied to analyze the CSI. It was found that when the CSI is operated without horizontal tilt the radius of curvature can be easily found from parameters measured from the fringe pattern. After determining the pathlengths of the counter-rotating beams, it was determined that the CSI has a very small pathlength difference, as small as $90\mu m$, which enables it to be used with a broad-band or short-pulsed laser source. On the other hand, other interferometers, such as the Collimation Tester, have a relatively large pathlength difference between the exiting beams which make these interferometers unusable for short coherence length sources. Experimental results showed that the CSI did behave as predicted; the comparison between the theoretical and experimental results was excellent. It was shown in Chapter 4 that the CSI could be built as a compact, one-piece unit for the shortest pathlength difference possible, and convenience.

REFERENCES CITED

REFERENCES CITED

1. M. Bass, J.S. Whittier, "Beam Divergence Determination and Collimation Using Retroreflectors," Appl.Opt. 23, 2674 (1984).

2. D. Malacara, *Optical Shop Testing* (Wiley and Sons, N.Y., 1978), p.133-6.

3. P. Langenbeck, "Improved Collimation Test," Appl.Opt. 9, 2590 (1970).

4. F.M. Dickey, T.M. Harder, "Shearing Plate Optical Alignment," Opt.Eng. 17, 295 (1978).

5. M.E. Riley, M.A. Gusinow, "Laser Beam Divergence Utilizing a Lateral Shearing Interferometer," Appl. Opt. 16, 2753 (1977).

6. A. Cordero-Davila, J. Pedraza-Contreras, O. Cardona-Nunez, A.Cornejo-Rodriguez, "Cyclic Interferometers for Optical Testing," Appl.Opt. 22, 2478 (1983).

7. R.S. Sirohi, M.P. Kothiyal, "Double Wedged Shearing Interferometer for Collimation Test," Appl.Opt. 26, 4054 (1987).

8. D.E. Silva, "A Simple Interferometric Method of Beam Collimation," Appl.Opt. 10, 1980 (1971).

9. J.C. Fouéré, D. Malacara, "Focusing Errors in a Collimating Lens or Mirror: Use of a Moire' Technique," Appl.Opt. 13, 1322 (1974).

10. S. Yokozeki, K. Patorski, K. Ohnishi, "Collimation Method Using Fourier Imaging and Moire Techniques," Opt.Comm. 14, 410 (1975).

11. K. Patorski, S. Yokozeki, T. Suzuki, "Collimation Test by Double Grating Shearing Interferometer," Appl.Opt. 15, 1234 (1976).

12. P. Hariharan, "Sagnac or Michelson-Sagnac Interferometer?," Appl.Opt. 14, 2319 (1975).

13. P. Hariharan, D. Sen, "Cyclic Shearing Interferometer," J.Sci.Instrum. 37, 374 (1960).

14. A.J. Montgomery, "Anaysis of Two-Tilt Compensating Interferometers," J.Opt.Soc.Am. 57, 1121 (1967).

15. O.D.D. Soares, "Analysis and Alignment of Cyclic Interferometers," J.Phys.E:Sci.Instrum., 11, 773 (1978).

16. Y. Li, G. Eichmann, R.R. Alfano, "Pulsed-Mode Laser Sagnac Interferometry with Applications in Nonlinear Optics and Optical Switching," Appl.Opt. 25, 209 (1986).

17. R.J. Wenzel, "Tilt in the Triangular Shearing Interferometer," Paper Presented at the OSA 1986 Annual Meeting in Seattle WA.

18. M. Herzberger, *Modern Geometrical Optics*, (Interscience Publishers,N.Y.,1958), p.3-12.

19. J.L. Carlsten, J. Rifkin, D.C. MacPherson, "Spatial Mode Structure of Stimulated Stokes Emission from a Raman Generator," J.Opt.Soc.Am.B, Vol.3,No.10, 1476 (1986).

20. D. Malacara, *Optical Shop Testing* (Wiley and Sons,N.Y.,1978), p.464-5.

APPENDICES

APPENDIX A

REVIEW OF THE WEDGED, SHEARING PLATE INTERFEROMETER

APPENDIX A

REVIEW OF WEDGED, SHEARING PLATE INTERFEROMETER

The wedged, shearing plate interferometer[5] can be used to measure the radius of curvature of a coherent light beam. This interferometer, also called a collimation tester, is often used to collimate a beam by checking that the radius of curvature of the beam is infinite. Although Riley and Gusinow[5] have investigated the collimation tester and its fringe pattern, the details of their method of analysis of the two exiting beams was not given. Therefore, for completeness and as a simple illustrative example of the use of the direction vectors described in Chapter 2, the formulas for the shear and tilt components between the two exiting beams of a collimation tester will be rederived exactly in this appendix. We will see that the formulas that Riley and Gusinow derived rely on small angle approximations but are appropriate for most experimental situations.

Fig. 15 shows a schematic of the collimation tester with its surface normals and the incident beam direction vector. Note that the incident beam is in the x-y plane. The tracing formula given by Eq. (1) and Eq. (2) of Chapter 2 is used to trace the two beams through the interferometer. A top view of the beam direction vectors are shown in Fig. 16. Note that S_1 is simply reflected off the front surface and remains in the x-y plane.

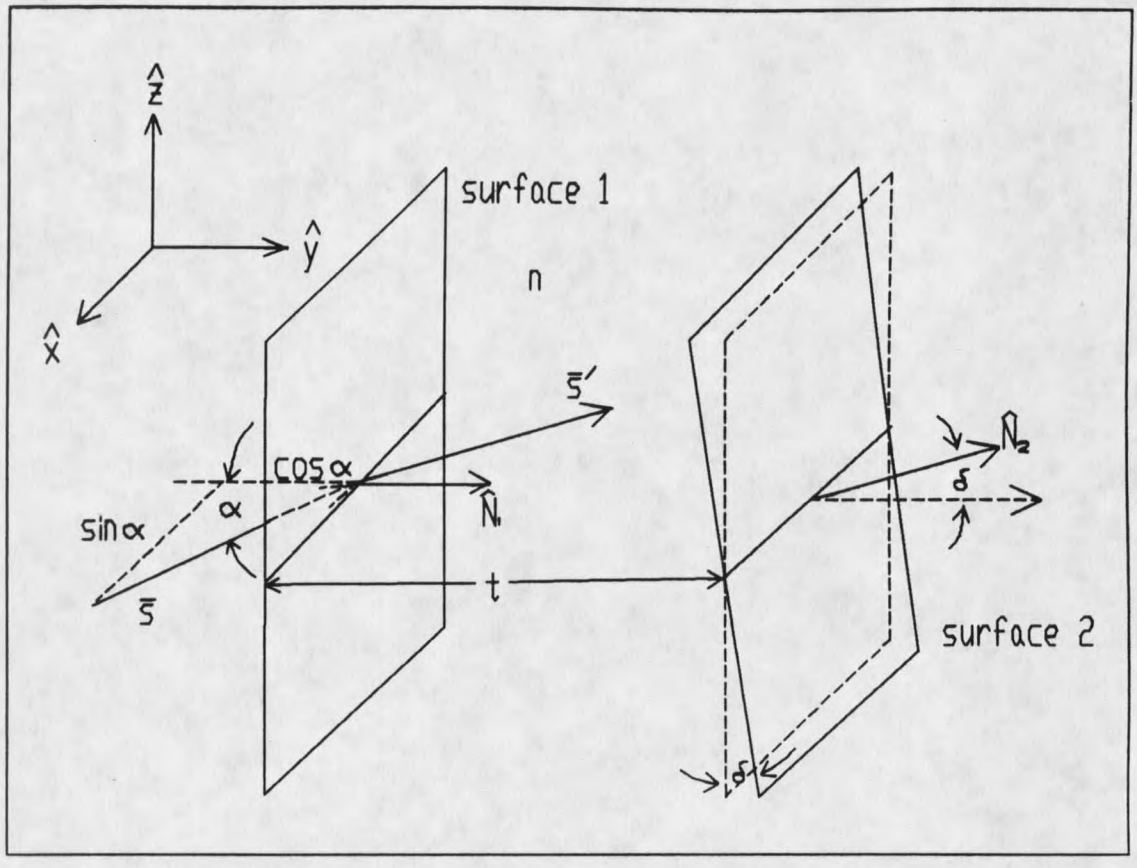

Figure 15. A schematic of the two surfaces of a collimation tester. Surface 1 and surface 2 are separated by a distance t with normals N_1 and N_2 respectively. The interferometer is made out of a material with an index of refraction n. The origin of the fixed coordinate system was conveniently chosen to be at the point where the incident ray, S, hits the first surface at an angle α with respect to the normal.

However the ray S''' will not lie in the x-y plane due to the reflection from surface 2 which has tilt about the x axis. The results are given by

$$S = -\sin\alpha\hat{\ell} + \cos\alpha\hat{y}$$

$$S_1 = -\sin\alpha\hat{\ell} - \cos\alpha\hat{y} \qquad \text{(A-1)}$$

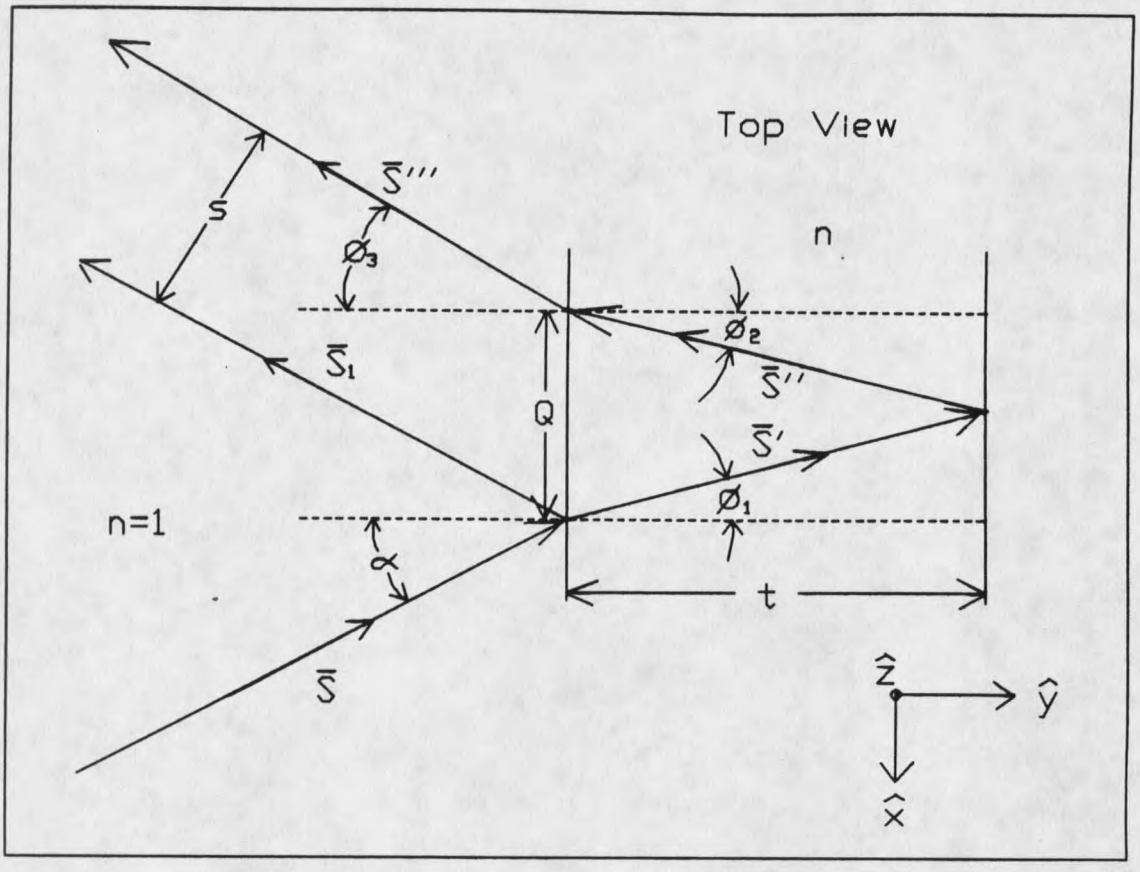

Figure 16. A top view of the Collimation Tester showing the incident beam, s, being traced through the interferometer (S', S'', S''', S_1) so that the shear, s, may be found. The beams associated with S_1 and S''' will interfere to form the fringe pattern on a screen.

$$S' = -\sin\alpha\hat{x} + (n^2 - \sin^2\alpha)^{1/2}\hat{y}$$

$$S'' = -\sin\alpha\hat{x} - (n^2 - \sin^2\alpha)^{1/2}\cos(2\delta)\hat{y} \qquad (A\text{-}1)_{cont.}$$
$$-(n^2 - \sin^2\alpha)^{1/2}\sin(2\delta)\hat{z}$$

$$S''' = -\sin\alpha\hat{x} - (1 - n^2 + (n^2 - \sin^2\alpha)\cos^2(2\delta))^{1/2}\hat{y}$$
$$-(n^2 - \sin^2\alpha)^{1/2}\sin(2\delta)\hat{z} \,,$$

where the index of refraction of air was taken to be 1 and the index of refraction of the

optic material is denoted as n. The vertical tilt, θ, between the exiting beams is found by comparing the z components of S''' and S_I. The relationship between the tilt and the radius of curvature is derived in Chapter 2. Since S_I has no z component, we find that

$$\sin\theta = (n^2 - \sin^2\alpha)^{1/2} \sin 2\delta , \tag{A-2}$$

or for small angles as Riley and Gusinow assumed,

$$\theta \approx (n^2 - \sin^2\alpha)^{1/2} 2\delta . \tag{A-3}$$

The horizontal tilt, ξ, between the two exiting beams is found by subtracting the angle which S_I makes with the y axis from the angle that S''' makes with the y axis. This angle is found to be

$$\xi = \alpha - \arctan\left(\frac{\sin\alpha}{\left(1 - n^2 + (n^2 + \sin^2\alpha)\cos^2(2\delta)\right)^{1/2}}\right) \tag{A-4}$$

The above expression can be simplified by using a Taylor expansion of the cosine term and then an expansion of the arctangent. These simplifications are possible because δ is typically very small. After expanding and keeping the first interesting term, the simplified expression is found to be

$$\xi \approx 2\delta^2 \tan\alpha\left(\sin^2\alpha - n^2\right) . \tag{A-5}$$

We see that ξ can be ignored because it is a function of δ^2, whereas θ cannot be ignored because it is a function of δ. It was shown in Chapter 2 that ξ must be small compared to θ if the radius of curvature of the incident beam is to be determined accurately. The horizontal shear, s, which is needed to produce a usable fringe pattern is found by

projecting the direction vectors onto the horizontal plane and finding Q as shown in Fig. 16. After simple geometry Q is found to be

$$Q = \frac{t \sin\alpha}{(n^2 - \sin^2\alpha)^{1/2}} \frac{\cos 2\delta + 1}{\cos 2\delta}, \tag{A-6}$$

where t is the average thickness of the interferometer. The horizontal shear is then given by the perpendicular distance between the beams,

$$s = Q \cos\alpha = \frac{t \sin(2\alpha)}{(n^2 - \sin^2\alpha)^{1/2}} \left(\frac{\cos 2\delta + 1}{2 \cos 2\delta} \right), \tag{A-7}$$

which can be reduced for small δ to give the solution of Riley and Gusinow,

$$s \approx \frac{t \sin 2\alpha}{(n^2 - \sin^2\alpha)^{1/2}}. \tag{A-8}$$

In most experimental situations the equation for the horizontal shear is never used because the shear is generally measured experimentally.

Vertical shear, *vs*, is the displacement of the two exiting beams along the vertical and only needs to be considered when it is about the same size as the beam's diameter. In that case the exiting beams will simply not interfere because one beam is above the other. The vertical shear can be found by projecting the direction vectors onto the vertical plane. One finds

$$vs \approx t(n^2 - \sin^2\alpha)^{1/2} \sin 2\delta, \tag{A-9}$$

where again, t is the thickness of the interferometer where the incident beam strikes.

In order for the two exiting beams to interfere, the coherence length of the incident

source must be less than, or comparable to the pathlength difference introduced in the interferometer. A $1cm$ thick collimation tester with an incident beam striking at 45° will have a pathlength difference, ΔL, of $2.9cm$. This spatial difference corresponds to a temporal difference of $\Delta t = \Delta L/c = 96psec$. Thus the Collimation Tester will not produce a usable fringe pattern from beams that have coherence times less than the $96psec$ listed above.

APPENDIX B

DETERMINING THE RADIUS OF CURVATURE FROM A FRINGE PATTERN

APPENDIX B

DETERMINING THE RADIUS OF CURVATURE FROM A FRINGE PATTERN

Before a fringe pattern is analyzed to find the Radius of Curvature R, one must determine the aspect ratio AR, the vertical and horizontal calibration constants C_V and C_H of the frame grabber (FG) / camera system, and the horizontal shear s produced in the Cyclic Shearing Interferometer. These Constants are found as described in Chapter 3. The analysis of the fringe pattern can continue only after the fringe pattern is viewed by the video camera and then captured by the FG and then stored in computer memory as a two dimensional array.

The analysis of a fringe pattern is essentially a two-dimensional least-squares fit to the two dimensional sine squared pattern of the fringes (Described in Chapter 2). The parameters which need to be determined from the fringe pattern, shown in Fig. 17, are the vertical fringe spacing, d, and the slope of the fringes from the horizontal, $tan\phi$. From these parameters and the previously found shear, s, the Radius of Curvature, R, can be found from Eq. (16) of Chapter 2

$$R = \frac{sd}{\lambda \tan\phi} . \qquad \text{(B-1)}$$

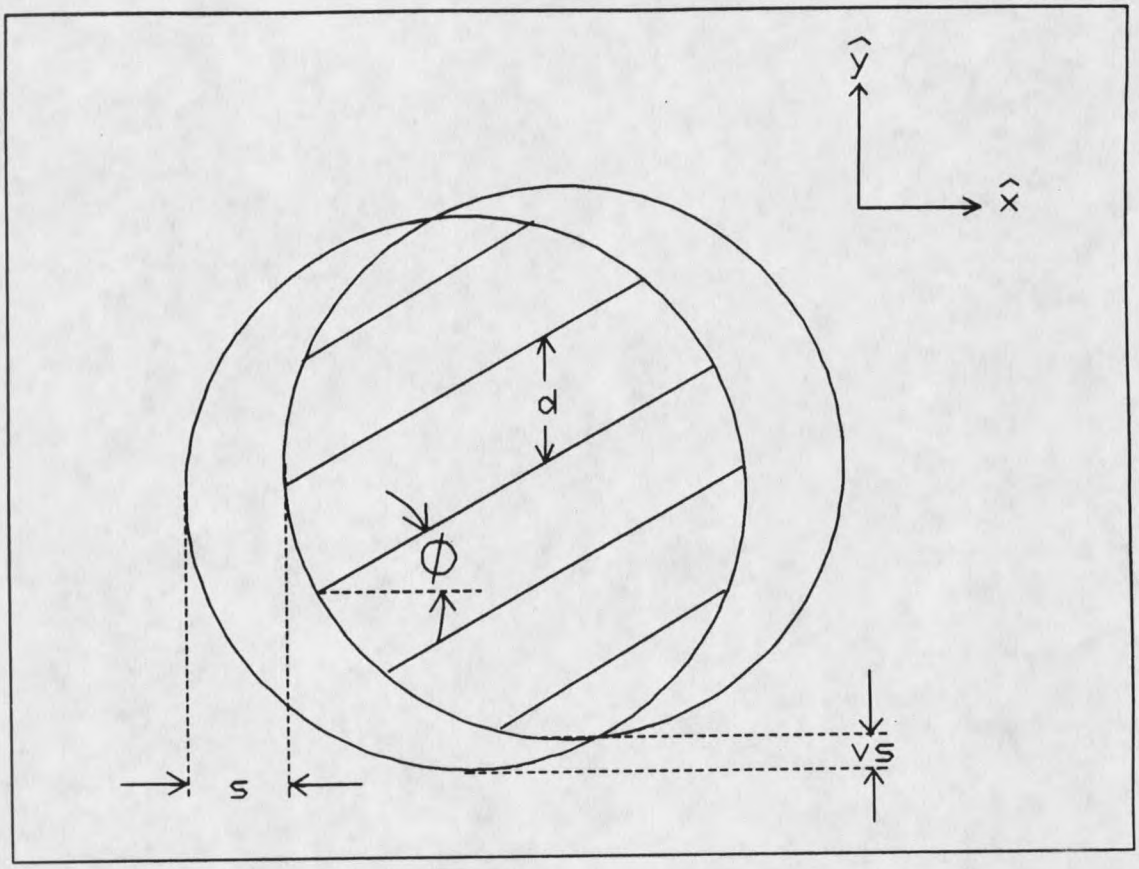

Figure 17. A schematic of a typical fringe pattern showing all of the measurable quantities. The quantities that are needed to compute the Radius of Curvature, R, are the shear, s, the distance between the fringes, d, and the slope of the fringes, $tan\phi$. The vertical shear, vs, is not needed to compute R.

The parameter d is essentially a scaled spatial period of the fringes along the y-axis whereas the slope of the fringes is associated with the phase shift between different vertical intensity profiles taken from the fringe pattern. The procedure used to determine the parameters d and $tan\phi$ and ultimately R from a captured fringe is given in the following five steps.

The first step is to acquire four vertical intensity profiles of the fringe pattern as shown in Fig. 18. More vertical profiles may be needed to ensure a good fit if the fringe

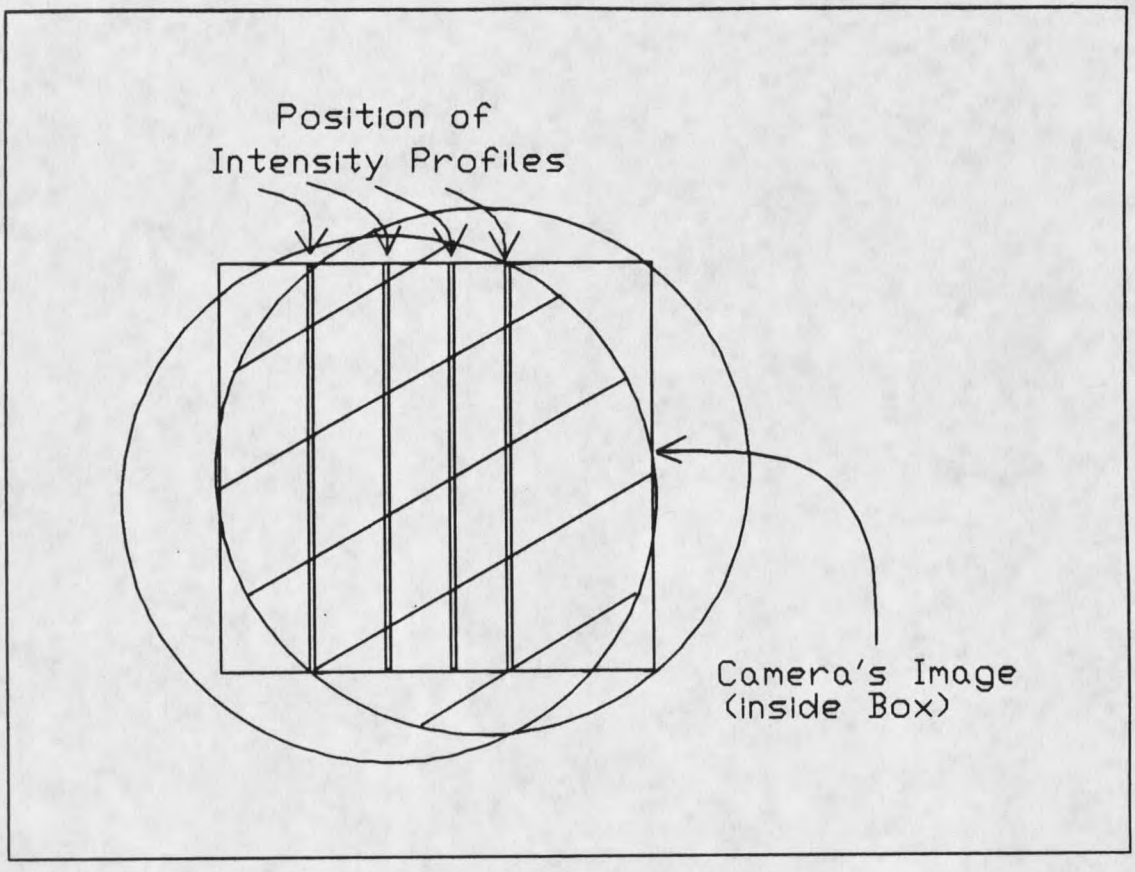

Figure 18. Schematic showing where four intensity profiles are taken from the fringe pattern inside the camera's view for analysis. The vertical fringe spacing and the slope of the fringes can be found from analyzing the profiles.

pattern contains a lot of noise. Capturing and storing profiles is done by using an option that is in the "Profile" subroutine which is in the program "Main_cntl" (See Appendix C). The user has the ability to choose which vertical profiles to extract which allows him/her to avoid portions of the fringe pattern which show defects in the optics. Each of these vertical profiles is stored as a data file that have names which indicate which column the intensity profile was taken from. The column information is important when determining the slope of the fringes. One such intensity profile is shown in Fig. 19.

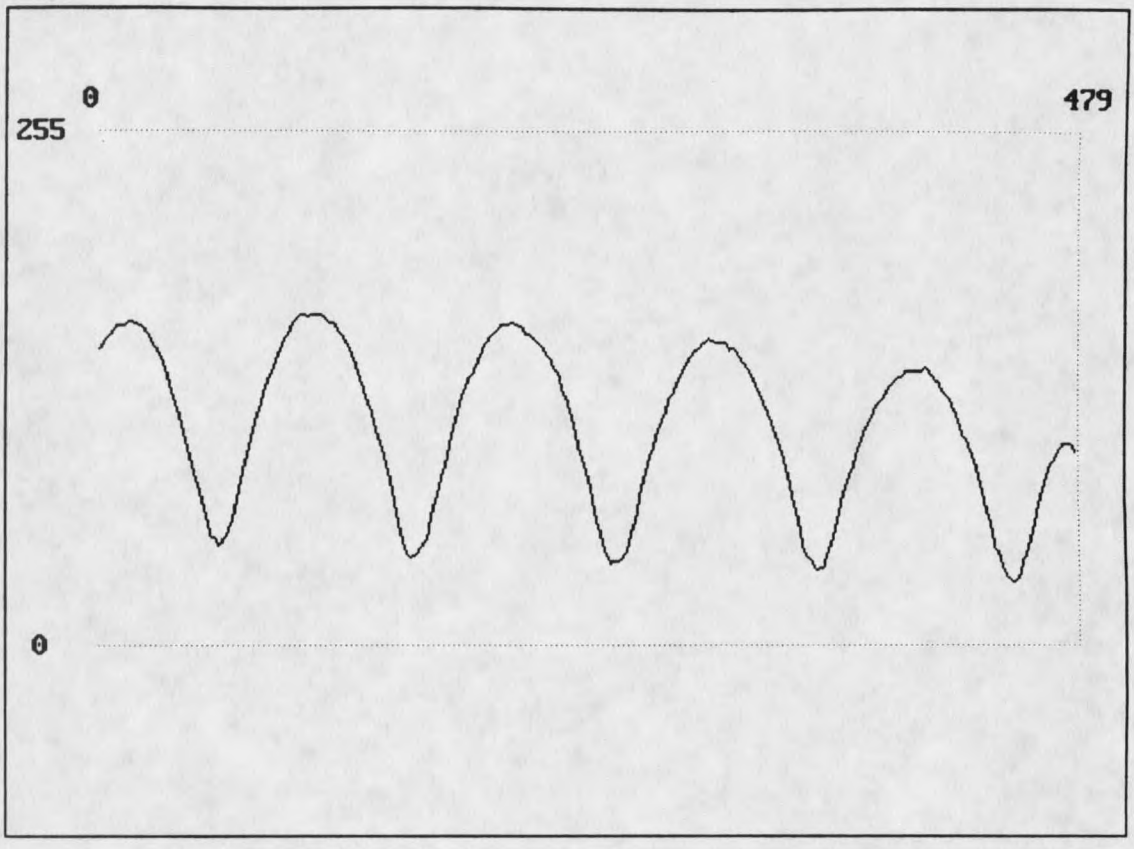

Figure 19. Vertical intensity profile data taken from an actual fringe pattern. This pattern could be saved as a data file in which the phase and the period could be found with the help of a fitting program.

The second step of the procedure is to determine the period of the vertical profiles using a least-squares fitting program (See Appendix D). The phase information for each of the profiles is also given from the fitting program, however, it will be shown in the next step that the phase information from this first round of fitting is not accurate, and therefore it is ignored. The period, T, for each profile is found by fitting each profile to

$$I = A\sin^2(ky+\theta) \ , \qquad \text{(B-2)}$$

where I is the intensity, A is the amplitude, $k = \pi/T$ (T is the period of the sine squared function), and θ is the phase of the function assuming that the top of the captured fringe pattern (or the left side of the profile shown in Fig. 19) is $y = 0$. In general one will obtain four slightly different values of k from this calculation. The period is obtained from the average value of k, or k_{ave}. The uncertainty in k_{ave}, Δk, is simply one standard deviation which is computed from the four k's.

The third step uses a least-squares fit to the function in Eq. B-2 to find the accepted phases of the individual profiles by keeping k fixed at k_{ave}. The reason we can not use the θ values obtained in the first least-squares fitting is that θ and k are not independent of each other for a periodic function that is not infinite. Therefore by forcing k to stay at its expected (average) value we force the phases of the profiles, θ, to fit to a fringe pattern that has a particular, constant fringe spacing, k_{ave}. The uncertainty in the fitted phases was computed in the least-squares fitting program.

Step four is to compute the actual slope of the fringes from the phases obtained from step two. To do this we use the definition $\theta = \beta k_{ave}$, where β has units of video pixels and $k_{ave} = \pi/T_{ave}$. When the β's are plotted against x (the column from which β was computed from) one will observe a straight line with a slope m_1. A linear fitting routine (See Appendix D) was used to find m_1 and its uncertainty, Δm_1. Now, m_1 is not the slope of the actual fringes because of some distortion introduced when acquiring and capturing the fringe pattern. The true slope of the fringes can be found by adjusting m_1 with the aspect ratio, AR. Therefore the slope of the fringes is

$$\tan\phi = m = (AR)m_1 \ , \qquad \text{(B-3)}$$

where the uncertainty in the slope is

$$\frac{\Delta \tan\phi}{\tan\phi} = \frac{\Delta AR}{AR} + \frac{\Delta m_1}{m_1} \quad . \tag{B-4}$$

Step five is to compute the vertical distance, d, between the fringes. This distance is simply the spatial period of the intensity pattern described in Eq. B-2. One simply needs to multiply the period, T (which has units of video pixels), by the calibration constant, C_v, to find d which has units of distance. So,

$$d = C_v T_{ave} \quad , \tag{B-5}$$

where the uncertainty in d is given by

$$\frac{\Delta d}{d} = \frac{\Delta C_v}{C_v} + \frac{\Delta T_{ave}}{T_{ave}} = \frac{\Delta C_v}{C_v} + \frac{\Delta k_{ave}}{k_{ave}} \quad . \tag{B-6}$$

Finally, step six is to compute the Radius of Curvature from Eq. B-1. So,

$$R = \frac{sd}{\lambda \tan\phi} \quad , \tag{B-7}$$

where the uncertainty in R is given by

$$\frac{\Delta R}{R} = \frac{\Delta s}{s} + \frac{\Delta d}{d} + \frac{\Delta \tan\phi}{\tan\phi} . \qquad \text{(B-8)}$$

APPENDIX C

DESCRIPTION OF VIDEO SYSTEM

APPENDIX C

DESCRIPTION OF VIDEO SYSTEM

The frame grabber/camera system was the chief component in the data collection which was needed to verify that the CSI operated as expected. The system includes the software to operate the system as well as the frame grabber and camera. Before the use of the controlling software for the video system is described, a brief review of the hardware will be presented.

Hardware

The hardware for the frame grabber/camera system consists of a "Data Translations" DT2853 frame grabber board, a "Pulnix" 745 CCD camera (or a "RCA" 2/3*in* VIDICON camera), and a control box. The control box contains a power supply for the camera, a trigger input, and cable connections which come from the computer to control the "Pulnix" camera's integration. Each of the hardware components will be described in more detail in the following paragraphs.

The DT2853 frame grabber (FG) is a computer card that fits on one of the computer's expansion slots which contains a set of A-to-D's and D-to-A's, two lookup tables, enough memory to store two video images, and memory moving controllers. All

of the FG's functions are controlled by writing (and reading) a series of 16 bit controlling registers. It is the job of the software to do the appropriate bit twiddling to control the various functions of the FG (and camera).

The basic operation of the FG is shown in a block diagram shown in Fig. 20. The operation is as follows: (1) The FG digitizes the input video signal with an A-to-D. (2) The signal that goes to one of the frame memories is established by comparing the digitized input video signal with the elements in the Input Lookup Table (Input LUT). The input LUT is generally linear, ie. a small input signal will be stored in the FG's memory as a small value. (3) The information that will eventually be sent to the video monitor comes from the output LUT's conversion of one of the frame memories. (4) The output LUT sends a digital signal to the output D-to-A which is simply the signal that goes to the monitor. The whole process transfers 512 X 480 video bits at full video speed which is $(1/30)s$ per frame. The FG also has the ability to be controlled by an external trigger source. One must refer to the FG's manual to fully understand its complicated operation or its numerous possible configurations.

The camera used in this system is the "Pulnix" TM745 CCD camera. It is a high resolution (768 X 493 *pixels*), $2/3in$ CCD shutter camera. The camera and FG are compatable because it communicates to the outside world using the RS-170 video communication standard. The user has the ability to either electronically shutter the camera or integrate the camera for any length of time. The integration allows the user to capture dim signals by integrating the CCD for a period of time. The user should refer to the camera's literature for specific operation information.

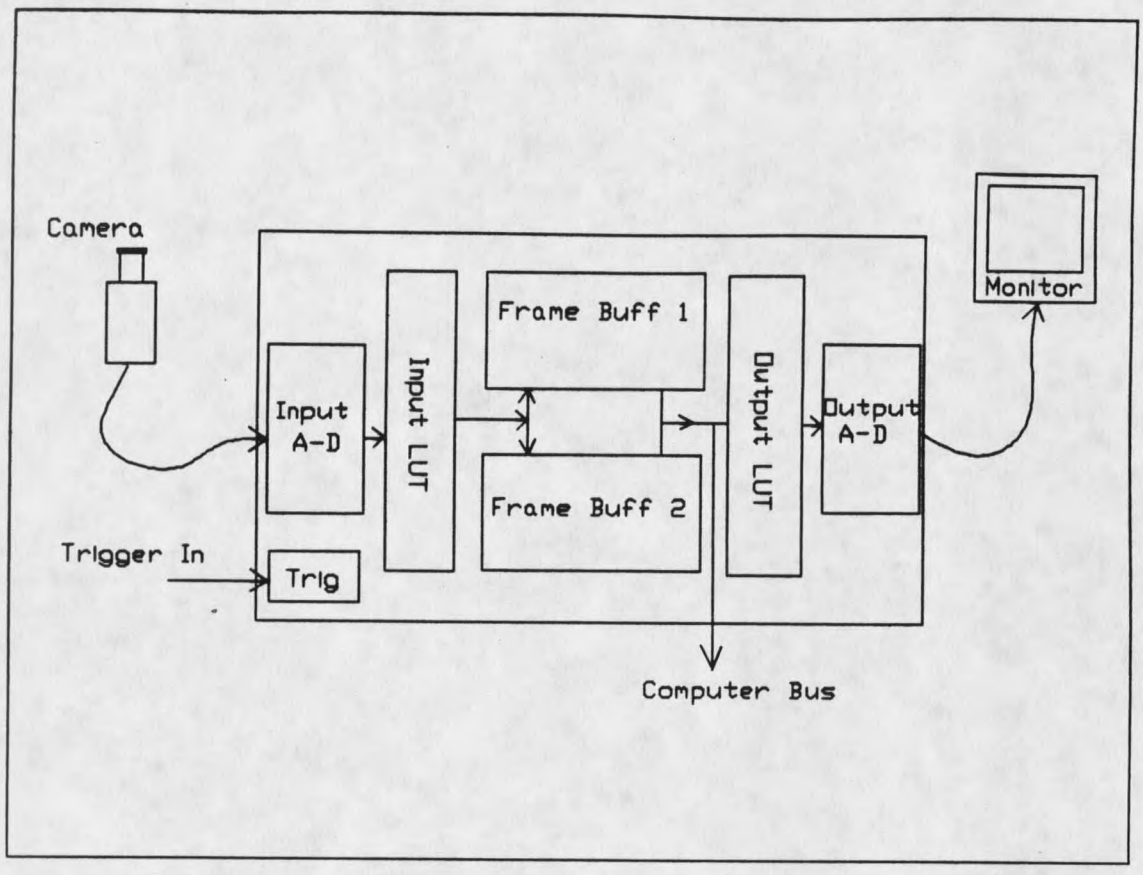

Figure 20. Block diagram of the frame grabber board.

The control box has three purposes; the first is to supply power to the camera, the second is to condition and invert the input trigger, and finally it has the proper connections to allow the computer to control the camera's integration (shuttering has not yet been configured). The trigger input allows an operation to start when the trigger goes from low to high (leading edge sensitive). It is useful when timing a frame capture to an event such as the firing of a pulsed laser. The controller connects to the computer via a parallel cable to the parallel port and a BNC cable to the FG board. Note, the controller box must be connected to the computer by either the parallel cable or the trigger BNC cable to ensure the camera's ground does not float. A floating ground will

not damage the camera, it will simply cause the camera to not communicate with the outside world.

Frame Grabber / Camera Operation

One program was written which has routines that control both the camera and the FG to make the system easy to operate. This program is called "Main_cntl". It is beyond the scope of this appendix to describe how the many subroutines work, but the general operation of the program will be described as a terse user manual for the operator.

Before "main_cntl" can be run one needs to make sure that the FG board is properly installed, the camera controller and camera are properly installed, and that all system interconnections are properly made. One can refer to the FG's instruction manual for proper board installation and most of the system interconnections. After all the system parts are properly installed and connected one can run the program by typing "MAIN_CNTL" while in the directory that the program is.

After starting the program the main menu will appear as shown in Fig. 21. At the bottom of the main menu screen is the status bar which displays buffer, camera, and cursor information. The following sections will describe each of the functions listed in the FG operation menu.

(1) Display Buffer 0/1: This determines which frame buffer will be displayed on the video monitor. The displayed buffer is toggled when this choice is selected. The current buffer being displayed is shown on the first entry of the status bar at the bottom of Fig. 20.

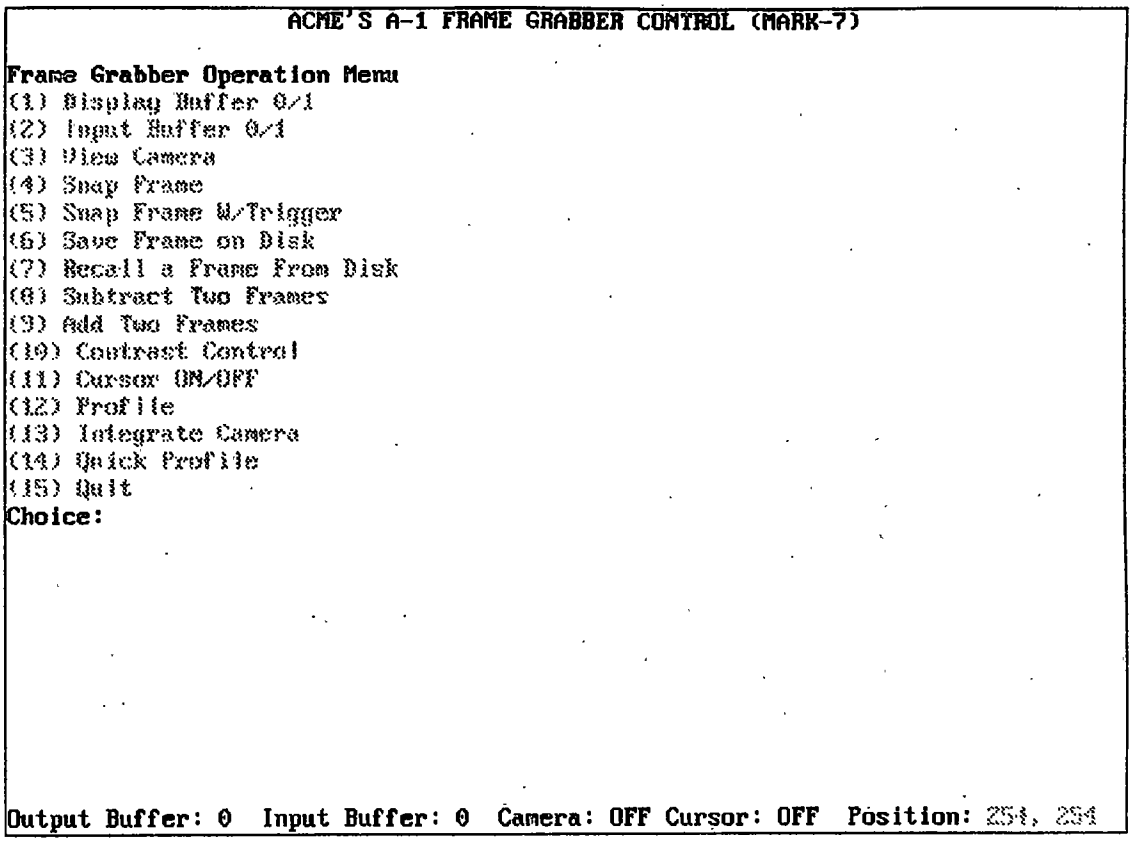

Figure 21. Main menu of the Frame Grabber / Camera controlling program called "MAIN_CNTL". The status bar at the bottom displays input and output buffers selected, whether the camera is being viewed, and the status and position of the cursor.

(2) Input Buffer 0/1: This will determine which buffer will receive the information from the camera or other source. The input buffer is toggled when this choice is selected. One cannot view the camera, or a frame from another source, on the monitor unless the input and output buffers are the same. The current input buffer is displayed on the status bar.

(3) View Camera: When selected it allows the information from the camera to pass to the input buffer and be refreshed at regular video speed. The status of the camera is

displayed in the status bar. (Note, a camera status of OFF does not imply that the camera does not have power, it means that the information from the camera is not passed to the input buffer.) To stop the FG from accepting information from the camera one needs to snap a frame (option 4).

(4) Snap Frame: When selected the camera will be turned off and the last frame accepted will be stored in the input frame buffer. If the camera is already off and this choice is selected, the FG will accept one frame from the camera and store it in the input frame memory.

(5) Snap Frame W/Trigger: Same as (4) except it waits for a trigger before capturing a frame. The 'snap frame' process is activated on the rising edge of the external trigger.

(6) Save Frame on Disk: Allows the user to store the viewed frame on a computer disk with a user specified name. The image is stored as XXXXXXXX.img, the XXXXXXXX is the user supplied name. Each frame takes about .25 Mbytes of memory.

(7) Recall Frame From Disk: Allows the user to recall a frame from the disk that was previously stored using (6). The image is recalled by typing in the name of the image that was saved under, but not the ".img" extension.

(8) Subtract Two Frames: Subtracts the frame not viewed from the currently viewed frame pixel by pixel. The resulting frame replaces the currently viewed frame. This function is useful when comparing two frames. Unexpected results will occur if the camera is ON (see (3)).

(9) Add Two Frames: Adds the frame viewed with the frame not viewed pixel by pixel. The resulting frame replaces the currently viewed frame. Unexpected results will occur if the camera is ON (see (3)).

(10) Contrast Control: Allows the user to enhance or decrease the contrast of a captured frame. When selected the user is asked for the amount of contrast enhancement between .001 and 1. A contrast enhancement of 1 will use the entire intensity scale. This is useful when viewing frames with very low intensities.

(11) Cursor ON/OFF: Toggles the monitor's cursor ON or OFF. When ON the user can move the cross-hairs around on the screen with the computer's mouse until a key is hit on the keyboard, then the cursor is frozen. Selecting this choice again will turn the cursor OFF which will make the cursor disappear. The Cursor's position is displayed on the status bar as x-y coordinates. Note that the top left corner of the frame is position 0,0.

(12) Profile: The profile subroutine allows the user to do several tasks related to the analysis of the frame. Several such things, described in detail below, are to view frames and spatially calibrate the video system, save individual intensity profiles as data files, and artificially colorize a frame so that it can be printed via a graphics grabber package. When *Profile* is selected a menu appears with the following options: *(1) Column Profile, (2) Row Profile, (3) Load New Frame, (4) Print Frame, (5) Quit*, these options are described in detail below.

(1) Column Profile / (2) Row Profile: When selected, the user will be asked for several things. The user will first be asked to analyze the *(1) Even, (2) odd, or (3)*

Both fields of the video display. The second question asked is whether to *(1) View Profile, (2) Calibrate the FG, or (3) Find the Intensity/Position Information With the Mouse*. Finally, the user will be asked which column/row to display. An example of a user viewing the odd field of an image to find the intensity/position information of line 250 is shown in Fig. 22. If one selected to *View Profile* no cursor or cursor information would appear. To *Calibrate the FG* (find the number of FG pixels per inch) one must first capture a frame of an object of known dimensions; a piece of graph paper works well. Once the image is captured and "Profile" is selected the calibration procedure can begin. To find the vertical calibration constant, C_v, one must first select the *Column profile* option and then select the *Calibrate FG* option. The rest of the procedure is directed by the software. The horizontal calibration constant, C_H, is found through the same procedure except that one needs to start with the *Row Profile* option.

When exiting this option (by hitting the spacebar) the user is asked whether to save the profile as a data file, if the user wishes to save a profile he/she will be asked for a file name.

(3) Load New File: Allows the user to load a new frame into *Profile* working memory without actually viewing it.

(4) Print Frame: Creates and prints an artificial colorized image on the computer's monitor which can be grabbed and printed with the use of a graphics grabber program. One such program is "Grafplus" by Jewell Technologies. This program can grab and store computer images in a variety of formats; the figures in this appendix were captured with "Grafplus".

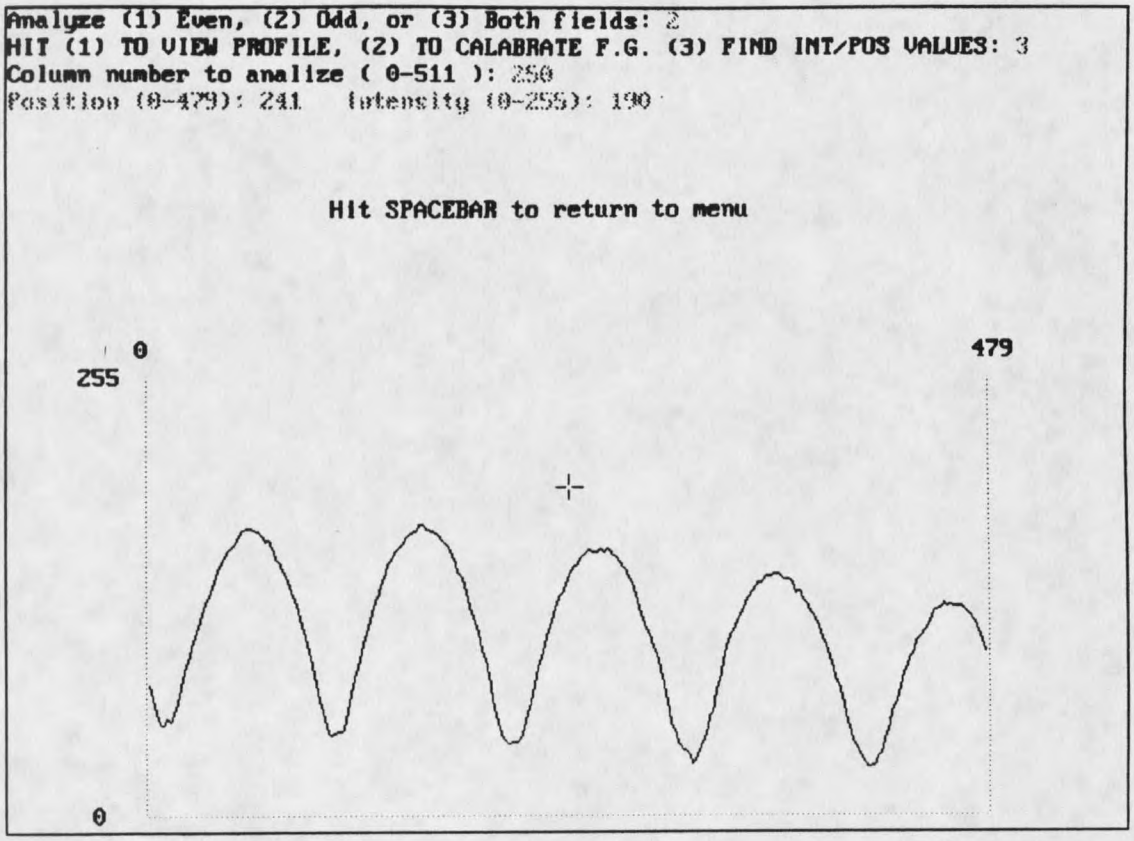

Figure 22. A typical screen in the "Profile" subroutine inside of the "MAIN_CNTL" program. A typical intensity profile is displayed at the bottom box, the mouse driven cursor is also shown (cross hairs). The cursor's intensity and position are displayed at the bottom of the text.

(5) Quit: Return to the main menu.

(13) Integrate Camera: Allows the user to control the time that the camera integrates to acquire a frame. The integration time, supplied by the user, is in terms of the camera's normal video integration period (1/30 *s*). Therefore an integration period of 3 will be approximately 3 times as bright as normal video. An integration period of more than 300 should be avoided because the accumulation of electronic noise will begin

to wash out the image. This choice is very useful in low light conditions.

(14) Quick Profile: Displays the cursor selected, horizontal intensity profile of the frame viewed on the video monitor on the computer's monitor. The cursor can be moved by moving the computer's mouse.

(15) Quit: End Program.

APPENDIX D

SOURCE CODE FOR LEAST-SQUARES FITTING PROGRAM

APPENDIX D

SOURCE CODE FOR THE LEAST-SQUARES FITTING PROGRAM

Figure 23. Source code written in 'C' for the Least-Squares curve fitting program.

```
/*****************************************************************************
*
*         PROGRAM NAME: LSQ_FIT
*            FILE NAME: Lsq_fit.c
*        CREATION DATE:
*      CREATION AUTHOR: Timothy Henning
*        REVISION DATE:
*      REVISION AUTHOR:
*           DESCRIPTION: This program can fit an array of data stored in a data
                          file to several functions: Sin^2, Sin^2 W/fixed Period,
                          Sin^2 W/Gaussian Envelope, and a Line.  Presently it is
                          taylored to data files created from the "Profile" sub-
                          routine created in the Frame Grabber controling program
                          called "MAIN_CNTL".
*          DATA INPUTS: Reads 480 data pairs from inputed data file ( 240 data
                          pairs for odd/even field data files ).  User is asked
                          which function to fit to.  Sigma is assumed same for
                          data value (set to 1).
*        RETURN VALUES: (not configured) yfit[512]--best fit function.
*       CONSTANTS USED:
*         GLOBALS USED:
*     GLOBALS MODIFIED:
*     FUNCTIONS CALLED:
* OS FUNCTIONS CALLED:
*         INCLUDE FILES:
*             LINK WITH: curve.obj, gridls.obj
*****************************************************************************/


extern void sin_sq1(), line_fit(), sin_sq2(), sin_sq3();
float x[512], y[512], sig[512], yfit[512];      /* x[] and y[]: holds data pairs
                                                   sig[]: holds errors in data
                                                   yfit[]: return best fit data
                                                */
main()

{
int choice, npts, field;
char dummy;

npts = 480;   /* number of data pairs */

      /* determines how many points to pull from data file */

printf("Data points come from (1) full frame, or (2) odd or even field : ");
```

Figure 23. continued.

```
scanf("%d%c",&field,&dummy);
if (field == 2)
       {
       npts = npts/2;
       }

       /* select which function to fit to */

printf("\nFit Data to: (1) line, (2) sin^2, (3) sin^2 (fixed period),");
printf(" (4) sin^2*gauss : ");
scanf("%d%c",&choice,&dummy);

switch(choice)              /* select which function to fit to */

   {
       case 1:
       {
       line_fit(x,y,yfit,sig,npts);      /* linear fit */
       }
       break;

       case 2:
       {
       sin_sq2(x,y,yfit,sig,npts);       /* sin^2 function */
       }
       break;

       case 3:
       {
       sin_sq3(x,y,yfit,sig,npts);       /* sin^2 function (fixed period) */
       }
       break;

       case 4:
       {
       sin_sq1(x,y,yfit,sig,npts);    /* sin^2 function (gaussian envelope) */
       }
       break;

   }
}            /* end of program */
```

Figure 23. continued.

```
/*************************************************************************
*       FUNCTION NAME: sin_sq1()
*           FILE NAME: curve.c
*       CREATION DATE:
*     CREATION AUTHOR: Timothy Henning
*       REVISION DATE:
*     REVISION AUTHOR:
*         DESCRIPTION: controls a sin^2*gaussian fitting subroutine.
*              INPUTS: x[]:x points of curve to be fit, y[]: y points
                       coorisponding to the x points, yfit[](blank), sig[]:
                       errors in y values, npts: number of points
*       RETURN VALUES:
*      CONSTANTS USED:
*        GLOBALS USED:
*    GLOBALS MODIFIED:
*    FUNCTIONS CALLED: gridls(), sinsq()
* OS FUNCTIONS CALLED:
*       INCLUDE FILES: stdio.h
*         LINK WITH: gridls.obj
*************************************************************************/
#include <stdio.h>
extern void gridls();
extern float sinsq();

void sin_sq1(x,y,yfit,sig,npts)
float x[], y[], yfit[], sig[];
int npts;

{
FILE *fopen(), *out_file;
int opt, i, j, numread, fpar, par, ncvm, ma, mfit, lista[3], mode, numwr;
char filename[30], dummy, change, save;
float a[6],deltaa[6],sigmaa[6],*chisq, chisq_old, co, ch, p[5];

printf("\nPull data from (1) Data File or (2) Frame Currently In Memory
");
scanf("%d%c",&opt,&dummy);     /* option 2 not programed */
if (opt==1)
      {
      do
         {
         printf("\nFile name to pull data from: ");
         gets(filename);

         out_file = fopen(filename,"r");
         if (out_file==NULL)
            {
            printf("\nCAN NOT READ FILE->TRY AGAIN\n");
            }
         }
        while(out_file==NULL);              /* keep asking for name until ok
*/

      for (i = 0;i<npts;i++)  /* Read in data pairs into working arrays */
            {
            fscanf(out_file,"%f %f",&x[i],&y[i]);
            }
      fclose(out_file);        /* close file */
      }
```

Figure 23. continued.

```
for (i=0;i<npts;i++)
      {
      sig[i] = 1;        /* y point's uncertainties = 1 */
      }
AGAIN:;
do
{                  /* Ask for parameters */
 fpar = 2;
 par = 4;
 printf("\nform:   p[0]   *   Sin^2(a[0]x   +   a[1])*exp((a[2]-x)/a[3])^2
+p[1]\n");
 for (i=0;i<fpar;i++)
      {
      printf("Input guess for fixed parameter: p[%d]: ",i);
      scanf("%f%c",&p[i],&dummy);
      }
 for (i=0;i<par;i++)
      {
      printf("Input guess for parameter: a[%d]: ",i);
      scanf("%f%c",&a[i],&dummy);
      }
 printf("\n\nThe parameters you guessed are: ");
 for (i=0;i<fpar;i++)
      {
      printf("p[%d]=%f ",i,p[i]);
      }
 printf("\n");
 for (i=0;i<par;i++)
      {
      printf("a[%d]=%f ",i,a[i]);
      }
 printf("\n DO YOU WISH TO CHANGE ANY OF THESE PARAMETERS ? (y/n): ");
 change = getch();
}
while (change == 'y');        /* Loop back up if any need to be changed */

chisq = &ch;
mode = 0;
            /* searching steps for parameters */
deltaa[0] = .001;
deltaa[1] = .05;
deltaa[2] = 1;
deltaa[3] = .5;
deltaa[4] = 5;
deltaa[5] = 5;
printf("\nThe searching step sizes for the parameters are: ");
for (i=0;i<par;i++)
      {
      printf("\ndeltaa[%d] = %f",i,deltaa[i]);
      }
printf("\nDo you wish to change the step sizes? (y/n): ");
change = getch();
if (change == 'y')
      {
      for (i=0;i<par;i++)
            {
            printf("deltaa[%d]: ",i);
            scanf("%f%c",&deltaa[i],&dummy);
            }
```

Figure 23. continued.

```
      }
printf("\nCalculating......\n");
chisq_old = 10000000;
j = 0;
for (i = 1;i<20;i++)     /* Main Iteration Loop */
                  /* Loop until chisq changes only 1% */

      {
      gridls(x,y,sig,npts,par,mode,a,deltaa,sigmaa,p,yfit,chisq,sinsq);
      printf("\nparameters    are:\na[0]=%f    +-    %f\na[1]=%f    +-
%f",a[0],sigmaa[0],a[1],sigmaa[1]);
      printf("\na[2]    =    %f    +-    %f\na[3]    =    %f    +-%f",a[2],
sigmaa[2],a[3],sigmaa[3]);
      printf("\na[4]    =    %f    +-    %f\na[5]    =    %f    +-
%f\n",a[4],sigmaa[4],a[5],sigmaa[5]);
      printf("chi^2 = %f\n",*chisq);
      if (((chisq_old - *chisq)/chisq_old)<=.01) i = 20;  /* end if change

                                                = 1%          */
      chisq_old = *chisq;
      j = j+1;
      }
if (j >=19)  /* If parameters not found within 20 iterations  */
      {
      printf("\nParameters not found within 20 iterations !!");
      printf("\nDo you wish to try again? (y/n): ");
      scanf("%c%c",&change,&dummy);
      if (change == 'y') goto AGAIN;
      }
else              /* Option to save best fit data as sata file */
      {
      printf("\nDo you wish to save the best fit curve? (y/n): ");
      scanf("%c%c",&save,&dummy);
      if (save=='y')
            {
            printf("\n\nFilename to store best fit data: ");
            gets(filename);
            out_file = fopen(filename,"w");
            for (i=0;i<npts;i++) fprintf(out_file,"%f %f\n",x[i],yfit[i]);
            fclose(out_file);
            }
      }
}


/*************************************************************************
*
*       FUNCTION NAME: sin_sq2()
*           FILE NAME: curve.c
*       CREATION DATE:
*     CREATION AUTHOR:
*       REVISION DATE:
*     REVISION AUTHOR:
*           DESCRIPTION: Fit to sin^2, fit phase and k. (see above for
program
                  comments)
*               INPUTS: x[], y[], yfit[](blank), sig[], npts
*       RETURN VALUES:
```

Figure 23. continued.

```
*       CONSTANTS USED:
*         GLOBALS USED:
*     GLOBALS MODIFIED:
*     FUNCTIONS CALLED:
* OS FUNCTIONS CALLED:
*        INCLUDE FILES: stdio.h
*
*****************************************************************************/
#include <stdio.h>
extern void gridls();
extern float sinsq2();

void sin_sq2(x,y,yfit,sig,npts)
float x[], y[], yfit[], sig[];
int npts;

{
FILE *fopen(), *out_file;
int opt, i, j, numread, fpar, par, ncvm, ma, mfit, lista[3], mode, numwr;
char filename[30], dummy, change, save;
float a[6],deltaa[6],sigmaa[6],*chisq, chisq_old, co, ch, p[5];

printf("\nPull data from (1) Data File or (2) Frame Currently In Memory
");
scanf("%d%c",&opt,&dummy);
if (opt==1)
      {
      do
         {
         printf("\nFile name to pull data from: ");
         gets(filename);

         out_file = fopen(filename,"r");
         if (out_file==NULL)
            {
            printf("\nCAN NOT READ FILE->TRY AGAIN\n");
            }
         }
        while(out_file==NULL);

      for (i = 0;i<npts;i++)
            {
            fscanf(out_file,"%f %f",&x[i],&y[i]);
            }
      fclose(out_file);
      }
for (i=0;i<npts;i++)
      {
      sig[i] = 1;
      }
AGAIN:;
do
{
 fpar = 2;
 par = 2;
 printf("\nform: p[0] * Sin(a[0]x + a[1])^2 +p[1]\n");
 for (i=0;i<fpar;i++)
      {
      printf("Input guess for fixed parameter: p[%d]: ",i);
```

Figure 23. continued.

```
      scanf("%f%c",&p[i],&dummy);
      }
 for (i=0;i<par;i++)
      {
      printf("Input guess for parameter: a[%d]: ",i);
      scanf("%f%c",&a[i],&dummy);
      }
 printf("\n\nThe parameters you guessed are: ");
 for (i=0;i<fpar;i++)
      {
      printf("p[%d]=%f ",i,p[i]);
      }
 printf("\n");
 for (i=0;i<par;i++)
      {
      printf("a[%d]=%f ",i,a[i]);
      }
 printf("\n DO YOU WISH TO CHANGE ANY OF THESE PARAMETERS ? (y/n): ");
 change = getch();
 printf("\n");

}
while (change == 'y');

chisq = &ch;
mode = 0;
deltaa[0] = .001;
deltaa[1] = .05;
deltaa[2] = 1;
deltaa[3] = .5;
deltaa[4] = 5;
deltaa[5] = 5;
printf("\nThe searching step sizes for the parameters are: ");
for (i=0;i<par;i++)
      {
      printf("\ndeltaa[%d] = %f",i,deltaa[i]);
      }
printf("\nDo you wish to change the step sizes? (y/n): ");
change = getch();
if (change == 'y')
      {
      printf("\n");
      for (i=0;i<par;i++)
            {
            printf("deltaa[%d]: ",i);
            scanf("%f%c",&deltaa[i],&dummy);
            }
      }
printf("\nCalculating......\n");
chisq_old = 10000000;
j = 0;
for (i = 1;i<20;i++)
      {
      gridls(x,y,sig,npts,par,mode,a,deltaa,sigmaa,p,yfit,chisq,sinsq2);
      printf("\nparameters    are:\na[0]=%f    +-    %f\na[1]=%f    +-
%f",a[0],sigmaa[0],a[1],sigmaa[1]);
      printf("\nchi^2 = %f\n",*chisq);
      if (((chisq_old - *chisq)/chisq_old)<=.01) i = 20;
```

Figure 23. continued.

```
        chisq_old = *chisq;
        j = j+1;
        }
if (j >=19)
        {
        printf("\nParameters not found within 20 iterations !!");
        printf("\nDo you wish to try again? (y/n): ");
        scanf("%c%c",&change,&dummy);
        if (change == 'y') goto AGAIN;
        }
else
        {
        printf("\nDo you wish to save the best fit curve? (y/n): ");
        scanf("%c%c",&save,&dummy);
        if (save=='y')
                {
                printf("\n\nFilename to store best fit data: ");
                gets(filename);
                out_file = fopen(filename,"w");
                for (i=0;i<npts;i++) fprintf(out_file,"%f %f\n",x[i],yfit[i]);
                fclose(out_file);
                }
        }
}


/*************************************************************************
*
*       FUNCTION NAME: sin_sq3()
*           FILE NAME: curve.c
*       CREATION DATE:
*     CREATION AUTHOR:
*       REVISION DATE:
*     REVISION AUTHOR:
*         DESCRIPTION: fit to sin^2 k held constant, phase varies
                       (see sinsq1() for program comments).
*              INPUTS: x[], y[], yfit[](blank), sig[], npts
*       RETURN VALUES:
*      CONSTANTS USED:
*        GLOBALS USED:
*    GLOBALS MODIFIED:
*    FUNCTIONS CALLED:
* OS FUNCTIONS CALLED:
*       INCLUDE FILES: stdio.h
*
*************************************************************************/
#include <stdio.h>
extern void gridls();
extern float sinsq3();

void sin_sq3(x,y,yfit,sig,npts)
float x[], y[], yfit[], sig[];
int npts;

{
FILE *fopen(), *out_file;
int opt, i, j, numread, fpar, par, ncvm, ma, mfit, lista[3], mode, numwr;
```

Figure 23. continued.

```
char filename[30], dummy, change, save;
float a[6],deltaa[6],sigmaa[6],*chisq, chisq_old, co, ch, p[5];

printf("\nPull data from (1) Data File or (2) Frame Currently In Memory
");
scanf("%d%c",&opt,&dummy);
if (opt==1)
      {
      do
         {
         printf("\nFile name to pull data from: ");
         gets(filename);

         out_file = fopen(filename,"r");
         if (out_file==NULL)
            {
            printf("\nCAN NOT READ FILE->TRY AGAIN\n");
            }
         }
        while(out_file==NULL);

      for (i = 0;i<npts;i++)
            {
            fscanf(out_file,"%f %f",&x[i],&y[i]);
            }
      fclose(out_file);
      }
for (i=0;i<npts;i++)
      {
      sig[i] = 1;
      }
AGAIN:;
do
{
 fpar = 3;
 par = 1;
 printf("\nform: p[0] * Sin(p[1]x + a[0])^2 +p[2]\n");
 for (i=0;i<fpar;i++)
      {
      printf("Input guess for fixed parameter: p[%d]: ",i);
      scanf("%f%c",&p[i],&dummy);
      }
 for (i=0;i<par;i++)
      {
      printf("Input guess for parameter: a[%d]: ",i);
      scanf("%f%c",&a[i],&dummy);
      }
 printf("\n\nThe parameters you guessed are: ");
 for (i=0;i<fpar;i++)
      {
      printf("p[%d]=%f ",i,p[i]);
      }
 printf("\n");
 for (i=0;i<par;i++)
      {
      printf("a[%d]=%f ",i,a[i]);
      }
 printf("\n DO YOU WISH TO CHANGE ANY OF THESE PARAMETERS ? (y/n): ");
```

Figure 23. continued.

```
 change = getch();
 printf("\n");

}
while (change == 'y');

chisq = &ch;
mode = 0;
deltaa[0] = .001;
deltaa[1] = .05;
deltaa[2] = 1;
deltaa[3] = .5;
deltaa[4] = 5;
deltaa[5] = 5;
printf("\nThe searching step sizes for the parameters are: ");
for (i=0;i<par;i++)
      {
      printf("\ndeltaa[%d] = %f",i,deltaa[i]);
      }
printf("\nDo you wish to change the step sizes? (y/n): ");
change = getch();
if (change == 'y')
      {
      printf("\n");
      for (i=0;i<par;i++)
            {
            printf("deltaa[%d]: ",i);
            scanf("%f%c",&deltaa[i],&dummy);
            }
      }
printf("\nCalculating......\n");
chisq_old = 10000000;
j = 0;
for (i = 1;i<20;i++)
      {
      gridls(x,y,sig,npts,par,mode,a,deltaa,sigmaa,p,yfit,chisq,sinsq3);
      printf("\nparameters are:\na[0]=%f +- %f",a[0],sigmaa[0]);
      printf("\nchi^2 = %f\n",*chisq);
      if (((chisq_old - *chisq)/chisq_old)<=.01) i = 20;
      chisq_old = *chisq;
      j = j+1;
      }
if (j >=19)
      {
      printf("\nParameters not found within 20 iterations !!");
      printf("\nDo you wish to try again? (y/n): ");
      scanf("%c%c",&change,&dummy);
      if (change == 'y') goto AGAIN;
      }
else
      {
      printf("\nDo you wish to save the best fit curve? (y/n): ");
      scanf("%c%c",&save,&dummy);
      if (save=='y')
            {
            printf("\n\nFilename to store best fit data: ");
            gets(filename);
            out_file = fopen(filename,"w");
```

Figure 23. continued.

```
            for (i=0;i<npts;i++) fprintf(out_file,"%f %f\n",x[i],yfit[i]);
            fclose(out_file);
            }
      }
}

/*************************************************************************
*
*       FUNCTION NAME: line_fit()
*           FILE NAME: curve.c
*       CREATION DATE:
*     CREATION AUTHOR:
*       REVISION DATE:
*     REVISION AUTHOR:
*         DESCRIPTION: fits data to a line. Finds slope and y intercept
*              INPUTS: x[], y[], yfit[](blank), sig[], npts
*       RETURN VALUES:
*      CONSTANTS USED:
*        GLOBALS USED:
*    GLOBALS MODIFIED:
*    FUNCTIONS CALLED:
* OS FUNCTIONS CALLED:
*       INCLUDE FILES: stdio.h
*
*************************************************************************/
#include <stdio.h>

extern void gridls();
extern float line();

void line_fit(x,y,yfit,sig,npts)
float x[], y[], yfit[], sig[];
int npts;

{
FILE *fopen(), *out_file;
int opt, i, j, numread, par, ncvm, ma, mfit, nterms, lista[3], mode,
numwr;
char filename[30], dummy, change, save;
float p[5],a[6],deltaa[6],sigmaa[6],*chisq, chisq_old, co, ch;

par = 2;
nterms = par;
do
   {
   printf("\nFile name to pull data from: ");
   gets(filename);
   out_file = fopen(filename,"r");
   if (out_file==NULL)
      {
      printf("\nCAN NOT READ FILE->TRY AGAIN\n");
      }
   }
while(out_file==NULL);

for (i = 0;i<npts;i++)
      {
```

Figure 23. continued.

```
      fscanf(out_file,"%f %f",&x[i],&y[i]);
      }
fclose(out_file);
for (i=0;i<npts;i++)
      {
      sig[i] = 1;
      }
AGAIN:;
do
{
par = 2;
printf("\nform: y = a[1]x + a[0]");
for (i=0;i<par;i++)
      {
      printf("Input guess for parameter: a[%d]: ",i);
      scanf("%f%c",&a[i],&dummy);
      }
 printf("\n\nThe parameters you guessed are: ");
 for (i=0;i<par;i++)
      {
      printf("a[%d]=%f ",i,a[i]);
      }
 printf("\n DO YOU WISH TO CHANGE ANY OF THESE PARAMETERS ? (y/n): ");
 change = getch();
}
while (change == 'y');
printf("\nNumber of Data Pairs in Data File: ");
scanf("%d%c",&npts,&dummy);
chisq = &ch;
mode = 0;
deltaa[0] = .001;
deltaa[1] = .05;
deltaa[2] = 1;
deltaa[3] = .5;
deltaa[4] = 5;
deltaa[5] = 5;
printf("\nThe searching step sizes for the parameters are: ");
for (i=0;i<par;i++)
      {
      printf("\ndeltaa[%d] = %f",i,deltaa[i]);
      }
printf("\nDo you wish to change the step sizes? (y/n): ");
change = getch();
if (change == 'y')
      {
      for (i=0;i<par;i++)
            {
            printf("deltaa[%d]: ",i);
            scanf("%f%c",&deltaa[i],&dummy);
            }
      }
printf("\nCalculating......\n");
chisq_old = 10000000;
j = 0;
for (i = 1;i<20;i++)
      {
      gridls(x,y,sig,npts,nterms,mode,a,deltaa,sigmaa,p,yfit,chisq,line);
      printf("\nparameters are:\na[0]=%f +- %f\na[1]=%f +-
```

Figure 23. continued.

```
%f",a[0],sigmaa[0],a[1],sigmaa[1]);
      printf("\nchi^2 = %f",*chisq);
      if (((chisq_old - *chisq)/chisq_old)<=.01) i = 20;
      chisq_old = *chisq;
      j = j+1;
      }
if (j >=19)
      {
      printf("\nParameters not found within 20 iterations !!");
      printf("\nDo you wish to try again? (y/n): ");
      scanf("%c%c",&change,&dummy);
      if (change == 'y') goto AGAIN;
      }
else
      {
      printf("\nDo you wish to save the best fit curve? (y/n): ");
      scanf("%c%c",&save,&dummy);
      if (save=='y')
            {
            printf("\n\nFilename to store best fit data: ");
            gets(filename);
            out_file = fopen(filename,"w");
            for (i=0;i<npts;i++) fprintf(out_file,"%f %f\n",x[i],yfit[i]);
            fclose(out_file);
            }
      }
}
```

Figure 23. continued.

```
/*************************************************************************
*
*       FUNCTION NAME: gridls()
*           FILE NAME: gridls.c
*       CREATION DATE: 1990
*      CREATION AUTHOR: Tim Henning (addapted from "Data Reduction and
Error
                  Analysis for the Physical Sciences", P. Bevington.)
                  p.212
*       REVISION DATE:
*     REVISION AUTHOR:
*        DESCRIPTION: A grid search least-squares fitting program which
can
                  fit to a function with non-linear coefficients.
                  several such functions are listed below.  The program
                  uses a 'parabolic' root search to find the minimum
                  chisq from three different values of chisq.

*                   INPUTS: x[], y[]-->data pairs, sigmay[]-->standard
deviations
                  for y points, npts--> number of data pairs,
                  a[]-->array of parameters to fit, contains initial
                  quesses for parameters, mode--> method of
                  weighting fit (dissabled), deltaa[]--> grid increments
                  for a[], sigmaa[]-->standard deviations for a[],
                  p[]-->constant parameters

*      RETURN VALUES: a[], sigma[], yfit[]-->best fit data pts, chisqr-->
                  chi squared value of fit.
*      CONSTANTS USED:
*        GLOBALS USED:
*    GLOBALS MODIFIED:
*    FUNCTIONS CALLED: all functions contained in this file
* OS FUNCTIONS CALLED:
*       INCLUDE FILES: math.h
*
*************************************************************************/

#include<math.h>

v                       o                       i                       d
gridls(x,y,sigmay,npts,nterms,mode,a,deltaa,sigmaa,p,yfit,chisqr,functn)

float x[], y[], sigmay[], a[], deltaa[], sigmaa[], yfit[],p[],*chisqr;
int npts, nterms, mode;
float (*functn)();

{
int i, j, nfree;
float chisq1, chisq2, chisq3, save, fn, delta;
float sinsq(), line(), fchisq(), free;

nfree = npts - nterms;
free = (float)nfree;
*chisqr = 0;
if (nfree<=0) goto DONE;   /* Not enough points to find a fit for vars  */

for (j=0;j<nterms;j++)  /* finds initial chisq for initial quesses */
```

Figure 23. continued.

```
      {
      for (i=0;i<npts;i++) yfit[i] = (*functn)(x,i,a,p);
      chisq1 = fchisq(y,sigmay,npts,nfree,mode,yfit);
      fn = 0;
      delta = deltaa[j];

    AGAIN:a[j] = a[j] + delta;              /* steps parameter by delta and
                                  computes a new chisq2 value   */

      for (i=0;i<npts;i++) yfit[i] = (*functn)(x,i,a,p);
      chisq2 = fchisq(y,sigmay,npts,nfree,mode,yfit);  /* compute second
                                                 chisq  */
      if (chisq1>chisq2) goto HERE; /* stepped in right direction  */

      else if (chisq1 == chisq2) goto AGAIN;   /* if no change do it again
*/

      else        /* steped in wrong direction, change delta to -delta */
      delta = -delta;
      a[j] = a[j] + delta;
      for (i=0;i<npts;i++) yfit[i] = (*functn)(x,i,a,p);
      save = chisq1;
      chisq1 = chisq2;
      chisq2 = save;

     HERE:fn = fn+1;
           /* compute third chisq with adjusted parameter a[j] */
      a[j] = a[j] + delta;
      for (i=0;i<npts;i++) yfit[i] = (*functn)(x,i,a,p);
      chisq3 = fchisq(y,sigmay,npts,nfree,mode,yfit);
      if (chisq3<chisq2)       /* trying to brachet bottom of parabola */
                        /* chisq2 must be the smallest value    */
           {
           chisq1 = chisq2;
           chisq2 = chisq3;
           goto HERE;
           }
           /* using parabola fit to find correct step to use  */
      delta = delta * (1./(1.+(chisq1 - chisq2)/(chisq3 - chisq2)) +.5);
      a[j] = a[j] - delta;
           /* compute standard deviation for a[j]  */
      sigmaa[j]= deltaa[j]  *  sqrt(2./(free  *  (chisq3-2.  *  chisq2  +
chisq1)));
      deltaa[j] = deltaa[j] * fn/3;  /* adjusts step size for next time */
      }

for (i=0;i<npts;i++) yfit[i] = (*functn)(x,i,a,p);
*chisqr = fchisq(y,sigmay,npts, nfree, mode, yfit);  /* comput final chisq
*/
DONE:;
}


/*************************************************************************
*
*       FUNCTION NAME: sinsq()
*           FILE NAME: gridls.c
*       CREATION DATE:
```

Figure 23. continued.

```
*      CREATION AUTHOR: Timothy Henning
*        REVISION DATE:
*      REVISION AUTHOR:
*           DESCRIPTION: Sin^2*gaussian function for fitting. Adjustable
                          parameters are in the array a[], const parameters
                          are in p[].  x[] is the data's x value.
*                INPUTS: x[] array of x data values, a[] array of fitting
                          parameters, p[] array of fixed parameters, i: data
                          value.
*         RETURN VALUES:
*        CONSTANTS USED:
*          GLOBALS USED:
*      GLOBALS MODIFIED:
*      FUNCTIONS CALLED:
*  OS FUNCTIONS CALLED:
*         INCLUDE FILES:
*
*************************************************************************/

float sinsq(x,i,a,p)
float x[], a[], p[];
int i;

{
float hold;

hold = sin(a[0] * x[i] +a[1]);
hold=exp(-(x[i]-a[2]) * (x[i]-a[2])/(a[3] * a[3])) * hold * hold * p[0] +
p[1];

return(hold);
}



/*************************************************************************
*
*         FUNCTION NAME: sinsq2()
*             FILE NAME: gridls.c
*         CREATION DATE:
*       CREATION AUTHOR: Timothy Henning
*         REVISION DATE:
*       REVISION AUTHOR:
*            DESCRIPTION: Function for sin^2 fixed amplitude and offset.
                          Fitting with phase and k.
*                 INPUTS: see above function
*         RETURN VALUES:
*        CONSTANTS USED:
*          GLOBALS USED:
*      GLOBALS MODIFIED:
*      FUNCTIONS CALLED:
*  OS FUNCTIONS CALLED:
*         INCLUDE FILES:
*
*************************************************************************/

float sinsq2(x,i,a,p)
float x[], a[], p[];
```

Figure 23. continued.

```
int i;

{
float hold;

hold = sin(a[0] * x[i] +a[1]);
hold= hold * hold * p[0] + p[1];

return(hold);
}


/*************************************************************************
*
*       FUNCTION NAME: sinsq3()
*           FILE NAME: gridls.c
*       CREATION DATE:
*     CREATION AUTHOR: Timothy Henning
*       REVISION DATE:
*     REVISION AUTHOR:
*          DESCRIPTION: Fit Sin^2 with phase only.
*               INPUTS:
*       RETURN VALUES:
*      CONSTANTS USED:
*        GLOBALS USED:
*    GLOBALS MODIFIED:
*    FUNCTIONS CALLED:
* OS FUNCTIONS CALLED:
*       INCLUDE FILES:
*
*************************************************************************/

float sinsq3(x,i,a,p)
float x[], a[], p[];
int i;

{
float hold;

hold = sin(p[1] * x[i] +a[0]);
hold= hold * hold * p[0] + p[2];

return(hold);
}


/*************************************************************************
*
*       FUNCTION NAME: line()
*           FILE NAME: gridls.c
*       CREATION DATE:
*     CREATION AUTHOR: Timothy Henning
*       REVISION DATE:
*     REVISION AUTHOR:
*          DESCRIPTION:  Line fitting function, parameters to fit Slope
                  (a[1]) and y intercept (a[0])
```

Figure 23. continued.

```
*              INPUTS:
*       RETURN VALUES:
*      CONSTANTS USED:
*        GLOBALS USED:
*    GLOBALS MODIFIED:
*    FUNCTIONS CALLED:
* OS FUNCTIONS CALLED:
*       INCLUDE FILES:
*
*****************************************************************************/

float line(x,i,a,p)
float x[], a[], p[];
int i;

{
float hold;

hold = a[1] * x[i] + a[0];

return(hold);
}


/*****************************************************************************
*
*       FUNCTION NAME: fchisq(): float
*           FILE NAME: gridls.c
*       CREATION DATE:
*     CREATION AUTHOR: Timothy Henning (adapted from Data Reduction.....,
                       Phillip R. Bevington)
*       REVISION DATE:
*      REVISION AUTHOR:
*         DESCRIPTION: Computes a chisq value for a set of data fitted to
                       a set of computed y-points which coorispond to
                       the x data points.
*              INPUTS: y[]: set of y data points, sigmay[] uncertainty in
                         y
                       points, npts: number of data points, nfree: free data
                       points (not tied to parameters), mode: weighting of
                       uncertainties, yfit[]: theoritical data points.
*       RETURN VALUES:
*      CONSTANTS USED:
*        GLOBALS USED:
*    GLOBALS MODIFIED:
*    FUNCTIONS CALLED:
* OS FUNCTIONS CALLED:
*       INCLUDE FILES:
*
*****************************************************************************/

float fchisq(y,sigmay,npts,nfree,mode,yfit)

float y[], sigmay[], yfit[];
int npts, nfree, mode;
                        /* mode not used, sigma[i]=1 (equal weight) */
{
float chisq, hold, weight;
```

Figure 23. continued.

```
int i;

chisq = 0;
if (nfree<=0)      /* must have more data points than parameters */
      {
      goto DONE;
      }
for (i=0;i<npts;i++)
      {
      weight = 1;         /* no weighting */
      hold = (y[i] - yfit[i]);
      chisq = chisq + weight * (hold * hold);
      }
chisq = chisq/(float)nfree;
DONE: return(chisq);
}
```