MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

OPTICAL INTERCONNECTIONS
USING
COMPUTER GENERATED HOLOGRAPHIC OPTICAL ELEMENTS

by
Les Y. C. Chan, B.S.E.E.

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfilment
of the requirements for the degree of
Master of Engineering

Ottawa-Carleton Institute for Electrical Engineering
Department of Electronics
Faculty of Engineering
Carleton University

December 1990

© copyright
1990, Les Y. C. Chan

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-515-28892-0

Canada

CARLETON UNIVERSITY
FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

"OPTICAL INTERCONNECTIONS USING
COMPUTER GENERATED HOLOGRAPHIC OPTICAL ELEMENTS"

submitted by Les Y.C. Chan, B.S.E.E.
in partial fulfilment of the requirements for
the degree of Master of Engineering.

Thesis Supervisor

Thesis Supervisor

Chairman, Department of Electronics

Carleton University
January 1991

Abstract

The drive towards ever faster computers is straining the limitations of electrical interconnects. Optical interconnects are necessary to improve the performance of subsequent generations of computers.

Free space optical interconnects are able to solve some of the problems with electrical interconnects. Computer generated holographic optical elements (CGHOE) are required to implement such systems.

The procedure to design and manufacture CGHOEs is detailed in this thesis. This includes low cost amplitude CGHOEs produced by photoreduction onto holographic plates. A higher efficiency binary phase CGHOE is designed and fabricated. The experimental results for the binary phase CGHOE are $\approx 10\mu\text{m}$ diameter spots and 39% efficiency.

Acknowledgements

I wish to thank Dr. Chrostowski and Dr. Syrett for being my thesis supervisors. They have been helpful and supportive throughout.

The National Research Council of Canada deserves thanks for allowing me to perform my research at its facilities. The photonics lab available there enabled research on my thesis to become a reality. I would like to recognize the help and friendship of all the people in the Photonics and Sensors section. Special mention goes to Maurice O'Sullivan, Simon Boothroyd, Ross Misner, Alfred Dufour, and Pierre Gariépy.

Great appreciation is bestowed upon all my friends, during this long period of time, for their support, friendship, and encouragement. Particular mention goes to Craig Twardy, Kate Harris, Peter Lindstrom, and Val Van Alstine

Finally, the patience and understanding of my parents have been much appreciated.

Table of Contents

Abstract	iii
Acknowledgements	iv
Table of Figures	vii
Chapter 1 INTRODUCTION	1
1.1 Introduction	1
1.2 Thesis Objectives and Organization	4
Chapter 2 COMPARISON OF OPTICAL VS. ELECTRICAL INTERCONNECTIONS	6
2.1 Introduction	6
2.2 Electrical Interconnections	8
2.2.1 Standard Conductors	8
2.2.2 Superconductors	9
2.3 Optical Interconnections	11
2.3.1 Guided Wave Optical Interconnections	11
2.3.2 Free-space Optical Interconnections	15
2.4 Interconnection Comparison	19
Chapter 3 THEORY OF COMPUTER-GENERATED HOLOGRAPHIC ELEMENTS	20
3.1 Introduction	20
3.2 Diffraction	21
3.2.1 Fresnel-Kirchhoff Diffraction	23
3.2.2 Rayleigh-Sommerfeld Diffraction	27
3.2.3 Interference	29
3.2.4 Plane Wave Spectrum and Propagation	30

3.3 Holograms	36
3.3.1 Conventional Holograms	36
3.3.2 Fourier Transform Holograms	40
3.4 Computer Generated Holograms	43
3.4.1 Computer Generated Fourier Holograms	43
3.4.2 Dammann Gratings	47
3.4.3 Iterative Holograms	50
3.4.4 Direct Computer Generated Holograms	52
Chapter 4 DESIGN AND CHARACTERIZATION OF COMPUTER GENERATED HOLOGRAPHIC ELEMENTS	57
4.1 Zone Plates	57
4.2 Multifacet HOE	65
4.2.1 One to 16 Amplitude HOE	70
4.2.2 SLM Controlled 1 to 8 HOE	73
4.3 Binary Phase HOE	79
Chapter 5 LIMITS AND REQUIREMENTS OF HOEs	91
Chapter 6 CONCLUSIONS	95
6.1 Summary	95
6.2 Conclusions	97
6.3 Future Directions	98
References	99
Appendix A PROGRAM LISTINGS	103
A.1 FZP.BAS	103
A.2 HOE.LSP	105
A.3 PS-HOE.LSP	111
A.4 PRISM.LSP	121

Table of Figures

2-1 Fibre coupling methods	13
2-2 Cross over of rib waveguides	14
2-3 Two free space clock distribution	16
2-4 Other Interconnection schemes.	18
3-1 Kirchhoff surface of integration.	24
3-2 Fresnel-Kirchhoff geometry	25
3-3 Rayleigh-Sommerfeld geometry	27
3-4 Classical Interference pattern	29
3-5 Plane wave	31
3-6 Rayleigh-Sommerfeld diffraction geometry	32
3-7 Gabor hologram setup	38
3-8 Leith-Upatneiks hologram setup	39
3-9 Fourier transforming lens	42
3-10 Detour-phase hologram	45
3-11 1-dimensional Dammann grating cell	48
3-12 2-dimensional Dammann grating cell	49
3-13 Constrained region of a DBS hologram	51
3-14 Point source and plane wave interference	54
3-15 Direct binary CGH	55
3-16 1 to 5 Hologram	56
4-1 Path difference method	58
4-2 Efficiency vs. phase levels	59
4-3 NA as a function of feature size	60
4-4 10x picture of actual FZP	61

4-5 FZP test setup	62
4-6 Point spread function for FZP	63
4-7 First minimum location	64
4-8 Multi-facet HOE construction.	66
4-9 Circle-square intersections	69
4-10 Setup for testing 1:16 HOE	71
4-11 Intensity profile of 1:16 HOE	72
4-12 SLM controlled 1:8 HOE	75
4-13 Output image of 1:8 HOE	75
4-14 Intensity profile of 1:8 HOE output	76
4-15 Three different SLM patterns	78
4-16 Pi phase shift	80
4-17 Paths of off-axis HOE	82
4-18 AutoCAD picture of 1:4 HOE	83
4-19 Setup to evaluate 1:4 HOE	84
4-20 Composite Intensity profile for 1:4 HOE	85
4-21 Theoretical and experimental 1:4 HOE profiles	86
4-22 Fabrication steps for reflective HOE	87
4-23 Diffraction efficiency versus phase	89
4-24 Relative efficiency versus exposure	90
5-1 Determining NA for minimum feature size	92

Chapter 1

INTRODUCTION

1.1 Introduction

Since the invention of the computer, electronic circuits have been getting faster and smaller. However, limits to the operating speed of completely electronic computers are being approached [1.1]. To overcome these limits, many have envisioned all optical computers [1.2]. These computers would contain elements that interact with photons instead of electrons. The interfaces to these optical computers would be the only place where photon-electron conversions would take place. But recently, there is a realization that this goal will not be achieved until the far future, unless there is a revolutionary breakthrough. The reason is that the interaction between photons is weak -- witness the fact that two beams of light can pass through each other without interaction. This property of light, however, makes it well suited as an information carrier.

In current integrated circuits, signals between different parts of the circuit are carried by metallic and poly-crystalline (polysilicon in the case of silicon IC's) layers. Signals in poly-crystalline lines travel via a diffusion process, which makes the propagation delay proportional to the square of the distance. Thus short interconnection distances can be handled by poly-crystalline lines without suffering long propagation times. With long metallic lines, there is the problem of capacitive loading. Finite drive currents mean that unless enough power is available, there will be delays introduced proportional to the capacitance.

Table 1-1 Comparison of optical vs. electrical interconnections.

	Optical	Electrical
Bandwidth	Greater than 100 THz	Less than 10 GHz
EMI	None	Yes
Complexity	Higher	Lower
Power dissipation	Lower at high data rates	Lower at lower data rates

Another major problem in electronic circuits is power dissipation. At higher speeds, proper terminations are required because even short interconnecting lines act as transmission lines and excessive ringing will result without proper termination. This also increases the power required to the signal lines because the impedances are quite low (usually 50 ohms). Another problem is that it is difficult to maintain proper terminations for a signal line that fans out to many others. The prime example is the clock line that is distributed to most of the circuit. These are some problems that have impeded the design of faster circuits.

Certain problems are more suited to a parallel architecture than a serial architecture. Thus, there is currently much interest in parallel computers. But parallel machines place an even greater demand on interconnections.

Optical interconnections may be used to overcome the problems mentioned previously. A brief comparison of electrical versus optical interconnections is shown in Table 1-1.

Thus, in the foreseeable future, the most promising role of optics in electronic systems will be in the interconnection of high speed electronic circuits.

Freespace optical interconnections are found to be attractive for the

interconnection of high speed VLSI. A crucial part of this type of interconnection is the optical element. The optical element must redirect, refocus and provide fanout. Holographic optical elements (HOEs) are able to fulfill these numerous functions. Standard HOEs are difficult to fabricate and are limited to physically recordable setups. Computer generated HOEs (CGHOEs) are compatible with the manufacture of standard integrated circuits. Since the CGHOE pattern is created in the computer, it is not constrained by the physical recording setup.

1.2 Thesis Objectives and Organization

The goal of this thesis is to evaluate the role of optics in interconnections, specifically the use of free space interconnections versus other methods. Holographic optical elements (HOEs) are constructed and tested. Computers are used to generate the HOEs because arbitrary patterns can be created by this method. It is also not necessary to worry about multiple exposures, which would be required in the case of conventional holograms.

The role of optical interconnects in electronic systems is still uncertain. In order to be able to apply this technology effectively, the advantages and limitations must be understood. Chapter 2 provides a comparison between optical and electrical interconnection technologies. The comparison is based upon current techniques in use. Optical interconnects will not be widely applied immediately and so other emerging technologies, such as superconductors, also must be considered.

The theory of holograms is briefly reviewed in chapter 3. This chapter begins with the theory behind conventional holograms. Next, the theory of various types of computer generated holograms is explained.

The design and characteristics of several holographic elements are detailed in chapter 4. The first HOE is a computer generated Fresnel zone plate. This demonstrates the use of an ordinary laser printer to generate holograms. Two methods are used to create a HOE that focused the collimated output of a laser diode at $1.064\text{ }\mu\text{m}$ to 16 points. Finally a system that uses a spatial light modulator (SLM) to dynamically enable and disable various facets of the HOE is covered.

Theoretically, computer generated HOEs can be very complex. Computational and production limitations are discussed in chapter 5. The

necessary requirements for producing HOEs for various types of systems are also summarized in chapter 5.

Recommendations for future work and conclusions are stated in chapter 6.

Chapter 2

COMPARISON OF OPTICAL VS. ELECTRICAL INTERCONNECTIONS

2.1 Introduction

Interconnections are one of the most important parts in any system. For electronic systems, the interconnections may be classified into four main levels. The first level connects the various elements within an integrated circuit. Connections on a circuit board constitute the second level of interconnection. Backplanes which connects a collection of circuit boards is an example of the third level of interconnection. Long distances characterize the fourth level of interconnection, with examples being LANs, radio transmitters and receivers. These classifications are not absolutes and there is a nearly continuous gradation between these levels. But the adoption of these levels enables a meaningful discussion of interconnection technology to take place.

Level-four connections have seen the widest penetration of optical interconnections in recent years. The most prominent example is the replacement of copper wire with fiber optic cables in the long distance telephone system. Optical backplanes using fiber optics have recently been introduced. This signals the beginning of optical interconnection in the level-three hierarchy. The use of opto-couplers to solve certain low speed isolation problems in level-two connections has existed for many years. Currently, there does not exist a more general use of optical interconnections in level-two. The use of optical interconnections in level-one is still primarily in the research stage.

The following sections will compare optical and electrical

interconnections. The emphasis will be on level-one and level-two connections. The comparison begins with characteristics of electrical connections. This is followed by a comparison of free-space and guided wave optical connections.

2.2 Electrical Interconnections

This section will describe the characteristics of electrical interconnections. The distinguishing characteristic of this type of connection is that communication takes place by the movement of charged carriers, usually electrons or holes. The first subsection will describe standard conductors that normally have finite resistance. There has been much research into high temperature superconductors in recent years, with many potential applications. Thus, the second subsection will deal with the characteristics of superconductors for use as interconnects.

2.2.1 Standard Conductors

There are two main types of conductors in the level-one hierarchy of intra-chip interconnections. One type is the metallic interconnect (typically aluminum), while the other type is the semiconductor interconnect (typically silicides). The main characteristic of standard conductors is their finite resistivity. Typical aluminum resistivity is $2.7 \times 10^{-6} \Omega \cdot \text{cm}$ while it is much higher for semiconductors ($\approx 2 \times 10^{-3} \Omega \cdot \text{cm}$ for polysilicon and $\approx 1 \times 10^{-4} \Omega \cdot \text{cm}$ for silicides) [2.1]. For high frequency signals the effective conductor area may be reduced due to the skin effect. The skin effect is frequency dependent and so high-speed pulses will undergo dispersion as they propagate along the interconnect. This distortion of pulse shapes may limit the speed of operation.

The heating of the standard interconnections, due to its finite resistance, is usually low. This is especially true of metals. Still, a more important problem is that this resistance contributes to part of the RC time constant. In many cases this limits the maximum operating speed.

Fanouts are simple to accomplish physically, as it is a matter of tying more conductors together. But terminations on IC's are quite complex because most interconnections are not transmission lines. A similar problem occurs at the junctions where signals are split. These conditions cause reflections that degrade the signal.

2.2.2 Superconductors

The recent discovery of superconductors having transition temperatures greater than that of liquid nitrogen (77°K) has garnered much attention. The main property of this material is that of zero resistance to flow of current at 0 K. Loss occurs due to finite penetration depth with AC signals just as with ordinary conductors, although this loss is exceedingly small up to several tens of gigahertz. Another property that is even more important when superconductors are used for interconnections is its near dispersionless nature. Transmission lines made using superconductors have zero group delay from DC to greater than a terahertz [2.2].

Zero resistance means that the RC limit will only result from the source's ability to charge up the capacitances. Slew-rate sensitive devices benefit from the very short risetimes that are available in superconductors. For inter-board and even intra-board interconnects, the negligible resistance means that much higher bandwidth signals can be used without the need to consider attenuation effects.

The application of superconductors must be tempered with the fact that at 77°K, copper is a very good conductor. Superconductors currently also have a problem with low critical current densities. When the current density exceeds

this limit, they revert to normal conductivities that are usually less than that of copper. The high current densities in IC mean that, the near room temperature superconductors are not suitable in this application [2.3].

2.3 Optical Interconnections

This subsection will deal with the two types of optical interconnections. One type is the guided wave interconnection with optical fiber and planar waveguides. The second type is free-space optical interconnection.

The ability of light beams to cross without interfering with each other is one of the advantages to using optical interconnections. This property makes fanouts using optical interconnections simpler to route than their electrical counterparts.

Optical interconnections are not susceptible to, nor do they generate, electromagnetic interference (EMI). This is because light is and electromagnetic(EM) wave itself. And EM waves can cross each other without interference. As light is an EM wave, it cannot generate other EM waves.

Light has very short wavelength. The short wavelength translates into extremely high bandwidths. The frequency band between wavelengths of 800 nm and 1.5 μm is 175 THz. A change of 1 nm corresponds to roughly 250 GHz.

Optical interconnections are more complex than electrical interconnections to fabricate. The complexity arises from the need to have convertors between the electrical and optical domains. The laser diode used to convert electrical signals to optical signals require more difficult fabrication steps than electrical connections.

2.3.1 Guided Wave Optical Interconnections

There are two major categories of guided wave interconnections. One is the optical fibre and the second is the planar wave guide. Currently, optical fibre is mainly used for communications between complete electronic systems

such as telephone switches. The use of fibre optic backplanes has recently been introduced. Connections between individual ICs on a circuit board by optical fibre are still in the research stage. To the author's knowledge, there is no research on using fibre optics as interconnections on a single IC at present.

Figure 2-1(a) shows a method of aligning fibres to a die [2.4]. In this scheme, a block is micro-machined with a v-groove for each fibre. Each fibre and the block is polished at approximately 45° so that total internal reflection takes place and the light is directed onto the die.

Another method to attach fibres to a die is shown in Figure 2-1(b) [2.5]. A hole the size of the fibre core is made in the photodetector. The stripped core of the fibre is inserted into the hole and glued in place. Although, the mechanical strength of this method is questionable as is its complexity, it provides exact alignment and uses very little chip area.

Planar waveguides are attractive because of the possibility of integration with ICs and circuit boards, which are planar in nature. In order to have a sufficiently high interconnection density, the waveguides must appear much like that of copper traces on a circuit board. This conflicts with the requirement that directional changes must be very gradual to reduce loss. Some work on abrupt directional changes using total internal reflection have been made with loss in 90° bends of 0.2-0.5 dB [2.6]. Crossovers using rib waveguides of the type shown in Figure 2-2 have been fabricated [2.7]. This is not possible with electrical interconnections because the signals would simply short together. The ability to cross over should simplify signal routing compared to electrical interconnections.

(a)

(b)

Figure 2-1
Two methods of connecting optical fibres to IC dies. (a) Cleaved fibre with mounting block. (b) Bare fibre core in photodiode hole.

Figure 2-2
Diagram showing how rib waveguides can cross each other while
maintaining signal integrity.

Optical waveguide production technology is similar to IC fabrication and simple waveguide structures have been integrated with electronics to create OEIC's (Opto-Electronic Integrated circuits) [2.8]. Problems with alignment are then limited to the normal alignment problems occurring with photographic masks and patterning equipment. In addition, once manufactured the device would be robust to mechanical stress.

The signal routing using optical interconnections is constrained to two

dimensions as it is for conventional planar electrical interconnects. Also, current technology for waveguide fabrication produce waveguides with relatively high loss, so that only short interconnects are feasible.

2.3.2 Free-space Optical Interconnections

The name free-space optical interconnects is descriptive of the major characteristic of this type of interconnection. Light is not guided, but is instead allowed to propagate freely. Free-space interconnections can be further classified into focused and unfocused systems. Figure 2-3 shows a simple example of each. The unfocused system is very inefficient because only a small portion of the light falls on detectors and will not be considered further.

The focusing element can be either a lens or a hologram. In practice, if there is fanout, lenses are impractical because it would be too difficult to fabricate small multifacet lenses that can focus to arbitrary locations.

There are several types of holograms that can be used. The first is optically created, in which laser beams physically interfere to create the hologram. Further distinctions include whether these holograms are the thin type or the volume type. Thin holograms have amplitude and/or phase variations that occur in a plane. Volume holograms have amplitude and/or phase variations that occur in three dimensional space. There are several ways to record these holograms. One method is to illuminate the holographic material by replacing each detector location with a source with the desired direction and divergence. Although this method is conceptually simple, it is very difficult to implement. A second method involves making multiple exposures. The holographic material is illuminated with the source and each detector location

Figure 2-3
Simple clock distribution system using free-space interconnections. (a)
unfocused, (b) focused.

consecutively. Unfortunately this method has difficulties with partially overwriting the hologram with each exposure and thus reducing the efficiency. The third method is similar to the second except that, for each exposure, only a unique section is uncovered. This multifaceted hologram generally produces a larger spot size than the other two methods.

Computer generated holograms make up the other class of holograms. The holograms generated by computers are invariably thin. This is because it would be extremely difficult to fabricate a volume hologram. Computer generation allows multifacet holograms to be fabricated with no additional steps to that required for a single hologram.

The HOE shown in Figure 2-3(b) is of the transmission type. Alternative uses of HOE are shown in Figure 2-4(a). Here, each laser illuminates a portion of the HOE which then splits and redirects the signal to different detectors. The scheme shown in Figure 2-4(b) [2.9] is useful for covering larger distances. Here, a laser illuminates the first hologram which collimates and redirects the beam. The collimated beam then propagates freely except at glass-air interfaces, where it undergoes total internal reflection. The figure only shows one transmitter and receiver. However, the first HOE could split the incident light into multiple beams that propagate in different directions. The output HOE could also split and focus the incoming collimated beam onto multiple detectors.

Free-space interconnections invariably take advantage of the third dimension as seen in the previous examples. This could allow greater densities because there would be less space taken up by interconnections on the surface. The clock distribution system shown in Figure 2-3(b) could provide very low clock skew.

(a)

(b)

Figure 2-4 (a) General interconnection scheme on an IC using a reflective HOE. (b) Board level interconnection scheme.

2.4 Interconnection Comparison

The main advantage of standard conductors for interconnects is their simplicity. The processes for producing them are also well known. Disadvantages include susceptibility to and generation of EMI. In comparisons with optical interconnects, it has been shown that at high speeds and/or long connections lengths, electrical interconnections use more power. Feldman, et al. [2.10] showed that the break-even aluminum line length, in terms of power consumption, is approximately 1 mm. The comparison conditions are a switching frequency of 500 MHz and 1 μ m geometry compared with 9% optical system efficiency and 1 mW laser threshold.

Fanouts are an important aspect of computer systems. They are required for signal distribution in structures such as buses. The number of fanouts on a transmission line is limited (ie. 1 load per 2 cm in a polyimide strip line even at data rates less than a GHz [2.4]). The number of fanouts in optical systems, however, is only limited by the power at each receiver and thus can be much larger. Waveguides are more suited for low fanout count. This is because fanout using waveguides requires a regular structure to distribute power equally. Free-space interconnections are not constrained to two dimensions and are well suited to provide fanout to random locations.

Chapter 3

THEORY OF COMPUTER-GENERATED HOLOGRAPHIC ELEMENTS

3.1 Introduction

Optical elements may be grouped into two major categories. The first include elements that can be described by geometrical (or ray) optics; examples are lenses, mirrors, and apertures. The second category includes elements that must be described by diffraction and interference. Holograms, and thus holographic elements, fall into the second category. Moreover, in certain circumstances, diffraction theory is also necessary for the elements such as apertures and lenses.

Section 3.2 briefly covers the theory of diffraction. It begins with the Fresnel-Kirchhoff formulation. The plane-wave spectrum method then follows.

Conventional holograms will be discussed in section 3.3. The next subsection will describe computer generated holograms. The various types of computer generated Fourier transform holograms are discussed first. This is followed by computer generated holograms that do not require a Fourier transform.

3.2 Diffraction

The propagation of light is governed by the laws of diffraction. This includes the nonrectilinear behavior of light, and in the limit as wavelength approaches zero, it also describes geometric optics. The effects of certain optical elements, such as holograms, can only be described by diffraction. The following treatment briefly describes the theory of diffraction. More complete and rigorous descriptions can be found in many other sources [3.1, 3.2].

This introduction to diffraction will deal only with **scalar** diffraction and monochromatic light sources. Scalar treatment means that only the magnitude of the transverse component of either the magnetic or electric field vectors is included in the analysis. The other component is assumed to be independent and analyzed in the same manner. The limitations of the scalar treatment are: the wavelength must be small compared with the diffracting aperture, and the observation point must not be too close to the aperture. If these conditions are not met, vector diffraction theory must be used to obtain accurate field descriptions. Additionally, only a few simple configurations have been solved by the vector approach. Nonetheless, useful results can still be obtained with scalar diffraction, even though the limitations may be violated [3.3]. The choice of monochromatic sources is not a serious limitation. Non-monochromatic sources can be treated as a generalization of the monochromatic case [3.2].

The two widely used scalar diffraction formulae are the Fresnel-Kirchhoff and the Rayleigh-Sommerfeld. The subsequent derivations loosely follow that of Goodman [3.2].

The scalar wave equation,

$$\nabla^2 u = \frac{1}{c^2} \frac{\partial^2 u}{\partial t^2}, \quad (3.1)$$

must be satisfied for the field $u(P,t)$ to be a valid optical wave. One function which complies is

$$u(P,t) = U(P) \cos[2\pi\nu t + \phi(P)] , \quad (3.2)$$

which represents an optical source at point P with field magnitude $U(P)$ and phase $\phi(P)$ at the optical frequency ν . Equation (3.1) may be more conveniently written as

$$u(P,t) = \text{Re}[U(P)e^{-j2\pi\nu t}] \quad (3.3)$$

where Re denotes the real part and

$$U(P) = U(P)e^{-j\phi(P)} . \quad (3.4)$$

The substitution of $u(P,t)$, in free-space, into scalar wave equation results in the Helmholtz equation:

$$(\nabla^2 + k^2)U = 0 \quad (3.5)$$

where the wave number is

$$k = \frac{2\pi}{\lambda} = 2\pi \frac{\nu}{c} . \quad (3.6)$$

The constant λ is the wavelength and c is the velocity of light in free-space.

The solution to the Helmholtz equation is simplified by using Green's theorem. The theorem states

$$\int_V \int_V (G \nabla^2 U - U \nabla^2 G) dV = \int_S \left(G \frac{\partial U}{\partial n} - U \frac{\partial G}{\partial n} \right) dS , \quad (3.7)$$

where $G(P)$ and $U(P)$ are complex variables of position. The integrations take place over a closed volume V of surface S , and $\partial/\partial n$ denotes partial differentiation along the normal pointing outwards from dS .

If G also satisfies the Helmholtz equation, then

$$\int_V \int_V (G \nabla^2 U - U \nabla^2 G) dV = \int_V \int_V (G k^2 U - U k^2 G) dV = 0. \quad (3.8)$$

Thus right hand side of (3.7) becomes

$$\int_V \left(\mathbf{G} \frac{\partial \mathbf{U}}{\partial n} - \mathbf{U} \frac{\partial \mathbf{G}}{\partial n} \right) dS = 0 \quad (3.9)$$

by substituting (3.8) into (3.7).

3.2.1 Fresnel-Kirchhoff Diffraction

Kirchhoff chose the Green's function $\mathbf{G}$ at an arbitrary point P_s to be

$$\mathbf{G}(P_s) = \frac{e^{ikr_s}}{r_s} \quad (3.10)$$

where r_s is the radial distance from the source P_s to the field observation point P_o . This is a reasonable choice because it has the form of a point emitter; it has a sinusoidal component and a $1/r^2$ power decay.

One problem with this choice of $\mathbf{G}$ is that there is a singularity $\mathbf{G}$ at $r_s = 0$.

To get around this problem, the integration volume excludes a spherical volume of radius δ around P_s . The surface of integration becomes $S = S' + \epsilon$ as shown in Figure 3-1.

Thus the summation in equation (3.9) is split into two as follows:

$$\int_V \left(\mathbf{G} \frac{\partial \mathbf{U}}{\partial n} - \mathbf{U} \frac{\partial \mathbf{G}}{\partial n} \right) dS - \int_{\epsilon} \left(\mathbf{G} \frac{\partial \mathbf{U}}{\partial n} - \mathbf{U} \frac{\partial \mathbf{G}}{\partial n} \right) dS = 0. \quad (3.11)$$

But since $\mathbf{G}$ is known, by differentiating $\mathbf{G}$ with respect to n ,

Figure 3-1
Kirchhoff surface of integration.

$$\frac{\partial G(P_o)}{\partial n} = \cos(\vec{n}, \vec{r}_{os}) \left(jk - \frac{1}{r_{os}} \right) e^{jk r_{os}}. \quad (3.12)$$

Where $\cos(\vec{n}, \vec{r}_{os})$ is the cosine of the angle between the outward normal $\vec{n}$ and $\vec{r}_{os}$, the vector from point P_s to P_o . When P_o is on ϵ , then

$$G(P_o) = \frac{e^{jk\delta}}{\delta} \quad \text{and} \quad \frac{\partial G(P_o)}{\partial n} = G(P_o) \left(\frac{1}{\delta} - jk \right). \quad (3.13)$$

If U and its derivatives are continuous and taking the limit as δ approaches 0,

$$\begin{aligned}
 & \lim_{\delta \rightarrow 0} \iint_{\Sigma} \left(\mathbf{G} \frac{\partial \mathbf{U}}{\partial n} - \mathbf{U} \frac{\partial \mathbf{G}}{\partial n} \right) dS \\
 &= \lim_{\delta \rightarrow 0} 4\pi \left[\frac{\partial \mathbf{U}(P_0)}{\partial n} \frac{e^{ik\delta}}{\delta} - \mathbf{U}(P_0) \frac{e^{ik\delta}}{\delta} \left(\frac{1}{\delta} - ik \right) \right] = -4\pi \mathbf{U}(P_0).
 \end{aligned}
 \tag{3.14}$$

Substituting equation (3.14) into equation (3.11) results in

$$\mathbf{U}(P_0) = \frac{1}{4\pi} \iint_{\Sigma} \left[\frac{\partial \mathbf{U}}{\partial n} \left(\frac{e^{ikr}}{r} \right) - \mathbf{U} \frac{\partial}{\partial n} \left(\frac{e^{ikr}}{r} \right) \right] dS.
 \tag{3.15}$$

Equation (3.15) is known as the *integral theorem of Helmholtz and Kirchhoff* [3.2]. This equation may be solved to give the field at a point P_0 in terms of the gradient on an arbitrary closed surface surrounding this point.

Figure 3-2
Geometry for derivation of the Fresnel-Kirchhoff diffraction formula

A useful application to solve is that of a point source that illuminates an opaque screen with some aperture A as shown in figure 3-2. The solution begins by applying equation (3.15) with the surface of integration being $S = A + B + C$. However equation (3.15) cannot be solved without imposing certain boundary conditions. Kirchhoff chose the following:

1. The field distribution $\mathbf{U}$ across the aperture and its partial derivative $\partial\mathbf{U}/\partial n$ are identical with the case when the opaque screen is absent.
2. The field $\mathbf{U}$ in the geometric shadow of the screen is zero as well as its partial derivative $\partial\mathbf{U}/\partial n$.

By the preceding two assumptions, S is reduced to $A + C$. For brevity, make the assumption that the wave is not strictly monochromatic and thus does not exist for all time (ie. not an eternal sinusoid). Then by making the radius for surface C arbitrarily large, $\mathbf{U}$ and $\partial\mathbf{U}/\partial n$ will both be zero on this surface. Finally, the surface of integration reduces to just the aperture A . If $r_{oA} \gg \lambda$ then equation (3.13) can be approximated by

$$\frac{\partial G(P_o)}{\partial n} \approx jk \cos(\vec{n}, \vec{r}_{oA}) \frac{e^{jk r_{oA}}}{r_{oA}}. \quad (3.16)$$

The point source wave $\mathbf{U}(P_o) = E_o \exp(jk r_{oA})/r_{oA}$ can be approximated in a similar manner if $r_{sA} \gg \lambda$. With these substitutions

$$\mathbf{U}(P_o) = \frac{E_o}{j\lambda} \int_A \int \frac{e^{jk(r_{sA} + r_{oA})}}{r_{sA} r_{oA}} \left[\frac{\cos(\vec{n}, \vec{r}_{sA}) - \cos(\vec{n}, \vec{r}_{oA})}{2} \right] ds. \quad (3.17)$$

This equation is known as the *Fresnel-Kirchhoff diffraction formula* [3.2].

3.2.2 Rayleigh-Sommerfeld Diffraction

Figure 3-3
Geometry for Rayleigh-Sommerfeld integration.

The Fresnel-Kirchoff diffraction formula has some limitations. The major one is that it does not provide the field very close to the aperture. The boundary conditions imply that the field is zero everywhere behind the aperture. This contradicts what is known in practice.

The problems with (3.17) were resolved by Sommerfeld. He chose $\mathbf{G}$ to be a point source at P_0 as well as a source 180° out of phase at the mirror image of P_0 . This configuration is shown in Figure 3-3. Thus $\mathbf{G}$ in equation (3.10) becomes

$$G(P_s) = \frac{e^{ikr_{sa}}}{r_{sa}} - \frac{e^{ik\tilde{r}_{sa}}}{\tilde{r}_{sa}}. \quad (3.18)$$

The derivative term is given by

$$\frac{\partial G(P_s)}{\partial n} = \cos(\vec{n}, \vec{r}_{sa}) \left(jk - \frac{1}{r_{sa}} \right) \frac{e^{ikr_{sa}}}{r_{sa}} - \cos(\vec{n}, \vec{\tilde{r}}_{sa}) \left(jk - \frac{1}{\tilde{r}_{sa}} \right) \frac{e^{ik\tilde{r}_{sa}}}{\tilde{r}_{sa}}. \quad (3.19)$$

The surface of integration is $S = A + C$. When P_s is on A ,

$$r_{sa} = \tilde{r}_{sa} \quad (3.20)$$

$$\cos(\vec{n}, \vec{r}_{sa}) = -\cos(\vec{n}, \vec{\tilde{r}}_{sa}).$$

Substituting into (3.18) and (3.19) results in

$$G(P_s) = 0$$

$$\frac{\partial G(P_s)}{\partial n} = 2 \cos(\vec{n}, \vec{r}_{sa}) \left(jk - \frac{1}{r_{sa}} \right) \frac{e^{ikr_{sa}}}{r_{sa}} \quad (3.21)$$

Make the assumption that the wave is not strictly monochromatic and thus does not exist for all time. Then by making the radius for surface C arbitrarily large, U and $\partial U / \partial n$ will both be zero on this surface. Thus equation (3.15) reduces to the following with the assumption that $r_{sa} \gg \lambda$,

$$U(P_o) = \frac{1}{j\lambda} \int_A \int U(P_s) \frac{e^{ikr_{sa}}}{r_{sa}} \cos(\vec{n}, \vec{r}_{sa}) ds. \quad (3.22)$$

Equation (3.22) will be referred to as the *general Rayleigh-Sommerfeld diffraction formula*.

The *Rayleigh-Sommerfeld diffraction formula* for a point source

$$U(P_o) = \frac{E_0}{j\lambda} \int_A \int \frac{e^{ik(r_{sa} + r_{oA})}}{r_{sa} r_{oA}} \cos(\vec{n}, \vec{r}_{oA}) ds \quad (3.23)$$

is obtained by selecting point source for $U(P_s)$ of (3.22). A comparison between (3.17) and (3.23) shows that the only difference is in the last term of each equation. The terms $\cos(\vec{n}, \vec{r}_{oA})$ and $[\cos(\vec{n}, \vec{r}_{oA}) - \cos(\vec{n}, \vec{r}_{sA})]/2$ are known as

obliquity factors. When the observation point is very close to the aperture, then the Rayleigh-Sommerfeld is more accurate. When the distance is larger, the two formulae provide very nearly the same answer.

3.2.3 Interference

Certain aspects of diffraction can be more easily explained by the interference principle. Interference takes place when coherent waves intersect. A classical demonstration (Young's double-slit experiment) is illustrated in Figure 3-4.

Figure 3-4
Interference from two coherent point sources in a single plane.

The circular lines represents lines of equal phase. Where two lines cross, constructive interference occurs and the amplitude increases. When the amplitudes of source 1 and 2 are equal, then there are locations where destructive interference makes the amplitude zero and other locations where constructive interference makes the amplitude twice the original amplitude. This effect of addition and cancelation is the principle of interference.

3.2.4 Plane Wave Spectrum and Propagation

The propagation of optical fields can be treated as a linear filter. In some cases this approach may provide simpler computations than the solution of the diffraction integral. This treatment mainly follows that of Gaskill [3.4].

In a rectangular system of coordinates, a propagating plane wave is represented by

$$U(x, y, z) = Ae^{j(\alpha x + \beta y + \gamma z)} \quad (3.24)$$

The angles between the positive x , y , and z axes and the propagation vector are $\cos^{-1}(\alpha)$, $\cos^{-1}(\beta)$, and $\cos^{-1}(\gamma)$ respectively. Since α , β , and γ are the direction cosines, they also must satisfy the relationship $\alpha^2 + \beta^2 + \gamma^2 = 1$. By rearrangement, we have

$$\gamma = \sqrt{1 - (\alpha^2 + \beta^2)} = \begin{cases} \text{Real} & (\alpha^2 + \beta^2) \leq 1 \\ \text{Imaginary} & (\alpha^2 + \beta^2) > 1 \end{cases} \quad (3.25)$$

When γ is real, U represents a propagating plane wave. However, when γ is imaginary, it represents an exponentially decaying field with distance. This type of non-propagating decaying field is termed evanescent. A pictorial representation of a plane wave with its propagation in the yz plane is shown in Figure 3-5.

Figure 3-5
Plane wave with direction cosines $(0, \beta, \gamma)$.

For a constant $z = z_1$, we have $U(x, y, z_1) = A e^{ikx_1} e^{ik(\alpha x_1 + \beta y_1)}$. For a given z_1 , the constant A can absorb e^{ikx_1} , which results in

$$U(x, y, z_1) = A e^{ik(\alpha x_1 + \beta y_1)}. \quad (3.26)$$

The geometry for Rayleigh-Sommerfeld diffraction may be simplified to that shown in Figure 3-6. The distance between P_s and P_o is given by

Figure 3-6
Rayleigh-Sommerfeld diffraction: geometry.

$$r_{12} = \sqrt{(z_2 - z_1)^2 + (x - \alpha)^2 + (y - \beta)^2} = z_{12} \sqrt{1 + \left(\frac{x - \alpha}{z_{12}}\right)^2 + \left(\frac{y - \beta}{z_{12}}\right)^2}. \quad (3.27)$$

The cosine of the angle between $\vec{n}$ and $\vec{r}$ is given by

$$\cos(\vec{n}, \vec{r}_{12}) = \frac{z_{12}}{r_{12}} = \frac{z_{12}}{\sqrt{1 + \left(\frac{x - \alpha}{z_{12}}\right)^2 + \left(\frac{y - \beta}{z_{12}}\right)^2}}. \quad (3.28)$$

These two equations may be substituted in (3.23), provided $r_{12} \gg \lambda$, which results

in

$$u_2(x, y) = \iint_{\Sigma} u_1(\alpha, \beta) \frac{\exp\left(jk z_{12} \sqrt{1 + \left(\frac{x - \alpha}{z_{12}}\right)^2 + \left(\frac{y - \beta}{z_{12}}\right)^2}\right)}{j\lambda z_{12} \left[1 + \left(\frac{x - \alpha}{z_{12}}\right)^2 + \left(\frac{y - \beta}{z_{12}}\right)^2\right]} d\alpha d\beta. \quad (3.29)$$

Moreover, (3.29) is equivalent to the convolution integral of [3.4]

$$u_2(x, y) = u_1(x, y) * \frac{\exp\left(jkz_{12}\sqrt{1 + \left(\frac{x}{z_{12}}\right)^2 + \left(\frac{y}{z_{12}}\right)^2}\right)}{j\lambda z_{12}\left[1 + \left(\frac{x}{z_{12}}\right)^2 + \left(\frac{y}{z_{12}}\right)^2\right]}, \quad (3.30)$$

where * denotes convolution. In a linear shift invariant system interpretation, u_2 is the output of a system with input u_1 and impulse response

$$h_{12}(x, y) = \frac{\exp\left(jkz_{12}\sqrt{1 + \left(\frac{x}{z_{12}}\right)^2 + \left(\frac{y}{z_{12}}\right)^2}\right)}{j\lambda z_{12}\left[1 + \left(\frac{x}{z_{12}}\right)^2 + \left(\frac{y}{z_{12}}\right)^2\right]}. \quad (3.31)$$

It is well known that convolution in one Fourier domain corresponds to multiplication in the transformed Fourier domain. The corresponding equations are

$$u_2(x, y) = u_1(x, y) * h_{12}(x, y), \quad (3.32)$$

$$U_2(\xi, \eta) = U_1(\xi, \eta)H_{12}(\xi, \eta).$$

where $h_{12} \xleftrightarrow{FT} H_{12}$, $u_1 \xleftrightarrow{FT} U_1$, and $u_2 \xleftrightarrow{FT} U_2$. If the transfer function H_{12} is known,

then the output field in the frequency domain is obtained by multiplication of the input field in the frequency domain by the transformation. The problem now becomes one of finding the Fourier transform of the impulse response.

Consider a physically realizable field $u_i(x, y)$ at $z = z_i$. This field will possess a two dimensional Fourier transform $U_i = \mathcal{FT}\{u_i(x, y)\}$. Writing u_i as the inverse transform of U_i , we have

$$u_i(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} U_i(\xi, \eta) e^{j2\pi(\xi x + \eta y)} d\xi d\eta. \quad (3.33)$$

By defining

$$\xi = \frac{\alpha}{\lambda}, \quad \eta = \frac{\beta}{\lambda}, \quad \zeta = \frac{\gamma}{\lambda}. \quad (3.34)$$

the plane wave of (3.26) becomes

$$U = A e^{jk(\alpha x + \beta y)} = A e^{j2\pi(\xi x + \eta y)}. \quad (3.35)$$

The components ξ , η , and ζ are the spatial frequencies in the x , y , and z axes, respectively. The units of angular frequency are m^{-1} .

The kernel of the Fourier transform can be viewed as plane wave with direction cosines $(\lambda\xi, \lambda\eta, \sqrt{1 - (\lambda\xi)^2 - (\lambda\eta)^2})$. Thus u_i can be regarded as being composed of a sum of plane waves. From (3.24) it can be seen that the only change that a plane wave undergoes is the phase factor $e^{jk(z_2 - z_1)}$ in traversing from z_2 to z_1 . This means that each elemental plane wave at z_2 can be obtained from its z_1 counterpart as [3.4],

$$U_2(\xi, \eta) = U_1(\xi, \eta) e^{jk(z_2 - z_1)\sqrt{1 - (\lambda\xi)^2 - (\lambda\eta)^2}}. \quad (3.36)$$

Furthermore, by simple rearrangement,

$$H_{12}(\xi, \eta) = \frac{U_2(\xi, \eta)}{U_1(\xi, \eta)} = e^{jk(z_2 - z_1)\sqrt{1 - (\lambda\xi)^2 - (\lambda\eta)^2}}. \quad (3.37)$$

In many situations, numerical results will suffice. When this is the case, the discrete Fourier transform (DFT) and its inverse may be used. The answer to the question of whether to use numerical integration of the diffraction formulas or the DFT will depend on several factors. If the field at the observation plane only needs to be known at a few points, then it is probably better to use direct integration. Conversely, if the fields need to be known over the whole observation plane, then it probably would be advantageous to use the DFT (actually the fast Fourier transform, FFT). A difficulty in using the FFT is that in order to get sufficient resolution over the observation plane, the step size must be small, which leads to very large two dimensional arrays. Most

computers, even with virtual memory, find it difficult to handle sizes larger than 2048×2048 which translates to 32 megabytes of storage assuming single precision numbers. Direct integration does not suffer from this problem. Its limitation is that a large number of function evaluations (especially great for non-smooth functions) must be made for each desired observation point.

3.3 Holograms

Photodetectors, such as photographic film or the eye, respond only to the intensity of light. Recordings of intensity is sufficient to capture two dimensional information.

In the case of light of a single frequency, if the instantaneous amplitude and phase is known at a plane, then the instantaneous amplitude and phase is known everywhere else. This basic ideal is the basis of holograms, because they are able to record both amplitude and phase information.

3.3.1 Conventional Holograms

One of the earliest holograms is the in-line (Gabor) hologram [3.5]. As the name implies, the source and object are aligned with the hologram axis. The arrangement used to record and display images is shown in Figure 3-7. A subject transparency $s(x,y)$ is illuminated by a coherent collimated source with amplitude r . The subject produces a field at the hologram plane of $s(x,y)$. Photosensitive materials, such as photographic plates, respond only to the intensity (magnitude squared). Thus the intensity $I(x,y)$ is given by [3.6]

$$I(x,y) = |r + s(x,y)|^2 \quad (3.38)$$

$$= r^2 + |s(x,y)|^2 + rs(x,y) + r s^*(x,y),$$

where $s^*(x,y)$ is the complex conjugate of $s(x,y)$. The actual hologram transparency transmittance is given by

$$t(x,y) = t_0 + \gamma I(x,y). \quad (3.39)$$

The slope of the exposure versus transmittance is denoted by γ . The average exposure is given by t_0 .

The reconstructed amplitude can be written as

$$u(x, y) = r t(x, y) \quad (3.40)$$

$$= r(t_0 + \gamma r^2) + \gamma r |s(x, y)|^2 + \gamma r^2 s(x, y) + \gamma r^2 s^*(x, y).$$

The first term in equation (3.41) is simply a constant background plane wave. The second term can be neglected when $s(x, y) \ll r$. Besides the constant γr^2 , the third term is the virtual image of the original. The real image that is formed is the conjugate of the original and is given by the fourth term.

The main shortcoming of the in-line hologram is that, when viewed, the desired image is overlapped with an out-of-focus conjugate image as well as a constant background.

The more useful arrangement for recording holograms is the off-axis (Leith-Upatnieks) method, an example of which is shown in Figure 3-8. The intensity is given by I and the hologram transmittance by t .

$$\begin{aligned} I(x, y) &= |r(x, y) + s(x, y)|^2 \\ &= r^2 + |s(x, y)|^2 + 2r |s(x, y)|^2 \cos[2\pi \xi_r x + \phi(x, y)] \end{aligned} \quad (3.41)$$

$$t(x, y) = t_0 + \gamma I(x, y)$$

where

$$\begin{aligned} r(x, y) &= r e^{i2\pi \xi_r x} \\ s(x, y) &= |s(x, y)| e^{-i\phi(x, y)} \\ \xi_r &= \sin(\theta)/\lambda. \end{aligned} \quad (3.42)$$

The corresponding reconstructed amplitude is

$$\begin{aligned} u(x, y) &= r(x, y) t(x, y) \\ &= t_0 r e^{i2\pi \xi_r x} + \gamma r |s(x, y)|^2 e^{i2\pi \xi_r x} + \gamma r^2 s(x, y) + \gamma r^2 s^*(x, y) e^{i4\pi \xi_r x}. \end{aligned} \quad (3.43)$$

The terms in equation (3.44) are similar to (3.41) except that the terms are now angularly separated. The first and second terms are a constant background plus a varying halo, respectively. The desired virtual image is given by the third term, and the fourth term is the conjugate real image.

Figure 3-7

- a) Setup for recording an in-line hologram
 b) Setup for viewing the reconstructed image.

Figure 3-8

- a) Off-axis hologram formation configuration
 b) Image reconstruction for an off-axis hologram.

The preceding descriptions of hologram formation and reconstruction are basic configurations. Many configurations are possible some of which may be found in [3.3, 3.6, 3.7].

3.3.2 Fourier Transform Holograms

Fourier transform holograms take advantage of the 2-D Fourier transforming properties of a simple positive lens [3.2, 3.3]. Two configurations to perform this transformation are illustrated in Figure 3-9. The Fourier transform of the mask $t(x,y)$ is denoted by $T(\xi,\eta)$. An additional requirement is that the system should be configured so that the Fresnel approximation can be used. This is met when

$$(L_t + L_T)^2 \ll \frac{4\lambda d_{TT}}{\pi}, \quad (3.44)$$

where L_t and L_T are the maximum radial extent of t and T , respectively, and d_{TT} is the distance between t and T . For a sufficiently large lens, the field at plane y_3 is proportional to the complex spatial Fourier transform. For configuration (a), shown in Figure 3-9,

$$U(\xi, \eta) = \frac{A}{j\lambda f} e^{-j\pi\omega\left(1-\frac{d}{f}\right)(\xi^2 + \eta^2)} \iint_{-\infty}^{\infty} t(x_1, y_1) e^{-j2\pi(x_1\xi + y_1\eta)} dx_1 dy_1 \quad (3.45)$$

where

$$\xi = \frac{x_1}{\lambda f}, \quad \eta = \frac{y_1}{\lambda f}.$$

Except for a quadratic phase factor in front of the integral, $U(\xi, \eta)$ is the Fourier transform of the input. The exact transform $T(\xi, \eta)$ is obtained when $d = f$.

The field at y_3 in Figure 3-9(b) is given by [3.2]

$$U(\xi, \eta) = \frac{A}{j\lambda d} e^{j\pi\lambda d(\xi^2 + \eta^2)} \left(\frac{f}{d}\right) \iint_{-\infty}^{\infty} t(x_1, y_1) e^{-j2\pi(x_1\xi + y_1\eta)} dx_1 dy_1 \quad (3.46)$$

where

$$\xi = \frac{x_1}{\lambda d}, \quad \eta = \frac{y_1}{\lambda d}. \quad (3.47)$$

Again, there is a quadratic phase factor in front of the integral. Additionally, equation (3.46) indicates that the size of the transform in x,y coordinates is scaled by (d/f) when compared to equation (3.45).

The wavefront recorded by Fourier transform holograms is that at the Fourier transform plane. Thus in order to generate the original wavefront, a lens is required to perform the inverse transform.

Figure 3-9
Configurations illustrating the Fourier transforming property of a positive lens. T is the Fourier transform of t .

3.4 Computer Generated Holograms

There are two main categories of computer generated holograms (CGH). The computer generated Fourier holograms is one category, while the other is the direct computer generated hologram. A feature of computer generated holograms is that the amplitude and phase information is quantized. Binary quantization is used most often because of the binary nature of most computer output devices.

The types of CGHs that are discussed in this subsection include: detour-phase holograms, Dammann gratings, iterative holograms, and direct computer generated holograms. Other types that are not discussed can be found in a compendium by Lee [3.8].

3.4.1 Computer Generated Fourier Holograms

Like the normal Fourier hologram, the computer generated version attempts to duplicate the magnitude and phase of the Fourier transform of a desired image. The recreation of this image requires an inverse Fourier transformation. This may be accomplished by placing the hologram at the plane y_1 of Figure 3-9 and the resultant image is viewed at plane y_2 .

The first step in the procedure to create this type of hologram is to calculate the Fourier transform of the image. Either the discrete or continuous Fourier transformation may be used. For the discrete case, the image is sampled and the 2-D FFT is used. Alternately, if the Fourier transform is known, then it is sampled directly. In either case, the end result is a 2-D array of values which represent the Fourier transform at each location.

An important consideration in the design of Fourier CGHs, as in any sampled system, is the Nyquist sampling theorem. This theorem states that for any bandwidth limited analog input signal, the sampling rate must be at least twice the rate of the highest frequency component of the input, in order to be able to regenerate the original input exactly. The regenerating process requires that the sampled signal must be low pass filtered. If the low pass filter is not present or is inadequate, then the recovered signal will be distorted since aliased components will be present.

The objective of Fourier CGHs is to represent the value at each sample point physically. A basic technique for this representation is the detour-phase hologram [3.9]. In this method, the complex value of each Fourier transform sample point is represented by a rectangular cell. The amplitude and phase information is conveyed by varying the size and position of a transparent rectangular aperture in an otherwise opaque cell. A representation of this type of CGH is shown in Figure 3-10.

There are $n \times m$ cells which correspond to each of the sample values. The size of each cell is the same and its dimensions are δ_x by δ_y . The aperture is centered vertically in the cell. The horizontal position of the centre of the aperture with respect to the centre of the nm^{th} cell is given by p_{nm} . The width of the aperture in each cell is constant, but the height of the nm^{th} cell is given by h_{nm} .

The transmittance of the hologram shown in Figure 3-10 is given by

Figure 3-10
General cell of a detour-phase hologram.

$$t(x, y) = \sum_n \sum_{n_i} \Pi\left(\frac{x - n\delta_i - p_{nm}}{w}\right) \Pi\left(\frac{y - m\delta_i}{h_{nm}}\right). \quad (3.48)$$

$$\Pi(x) = \begin{cases} 1 & |x| \leq 0.5 \\ 0 & \text{otherwise} \end{cases}.$$

The Fourier transform of $t(x, y)$ is given by [3.8]

$$\begin{aligned} T(u, v) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} t(x, y) \exp[-j2\pi(ux + vy)] dx dy \\ &= \sum_n \sum_{n_i} G_{nm}(u, v) \exp[-j2\pi(n\delta_i u + m\delta_i v)], \end{aligned} \quad (3.49)$$

with

$$G_{nm}(u, v) = \frac{\sin(\pi w u)}{\pi u} \frac{\sin(\pi h_{nm} v)}{\pi v} \exp(-j2\pi p_{nm} u). \quad (3.50)$$

The horizontal location of the aperture p_{nm} appears in the phase term of G_{nm} . The phase of the sample can thus be set by correctly positioning the aperture horizontally. Equation (3.50) also shows that h_{nm} determines the magnitude of the sample. The amplitude of the sample value in a cell can be approximated by controlling the aperture height h_{nm} .

The advantage of detour-phase CGHs is its simplicity. The hologram is composed only of on-off sections. This means that the fabrication of such a hologram is quite simple.

A disadvantage is the very low diffraction efficiency. This results from the fact that only a small portion of each cell lets light through. The image will also suffer some aliasing because there is no explicit reconstruction filter in the standard configuration shown in Figure 3-9. Additional problems with this method are that a conjugate image and a DC term are also formed.

3.4.2 Dammann Gratings

Dammann gratings (originally called multiple phase holograms by Dammann [3.10]) are similar to Fourier holograms in that a Fourier transforming lens is required. This is because both holograms are encoded with the inverse Fourier transform of the desired image. The major difference is that Dammann gratings have identical cells that are repeated, whereas the cells of a Fourier hologram are generally different from each other. Dammann gratings are similar in structure to the structure shown in Figure 3-10, except that the lower portion of the figure is replaced by Figure 3-12.

To produce a Dammann grating, the basic idea is to find a cell structure with the desired Fourier series (one or two dimensional). To generate an array of spots, the usual criteria for selection of a cell structure are high diffraction efficiency and equal spot intensities for a given array size. Once a suitable cell structure is found, the cell is replicated. The number of replications determines the size of the resultant spots. The greater the number of replications, the smaller the spot size because the numerical aperture (NA) of the system increases.

The most common usage of Dammann gratings has been for the generation of spot arrays. There have been demonstrations of Dammann gratings that have generated a 25x25 array of spots [3.11]. Arrays with specific spot patterns have also been designed [3.12].

The 1-dimensional case is easiest to illustrate. The simplest case is that of binary phase Dammann gratings. This means that the only variables are the number and the location of transitions from 0 to 180°. The goal then is to optimize the locations of a given number of transition points. Figure 3-11 shows an example of a 1-dimensional Dammann grating with 14 transition points.

Figure 3-11
 a) A 1-dimensional Dammann grating cell.
 b) Fourier series expansion of a).

One method to extend the 1-D grating into two dimensions is illustrated in Figure 3-12. The main feature of this method is that the transition points along both axes are the same.

Assuming a constant focal length, the spacing of the various diffraction orders is determined by the size of each cell. The fundamental spatial period T_0 is the cell size. If the configuration in Figure 3-9(a) is used, then equation (3.45) applies and the image plane is the Fourier transform of the grating. Application of the array theorem of Fourier transforms results in the following intensity distribution:

$$I(\lambda) = \left(T_1(\xi) \frac{\sin(\pi N T_0 \xi)}{\sin(\pi T_0 \xi)} \right)^2, \quad \xi = \frac{x}{\lambda f}, \quad (3.51)$$

where N is the number of cells illuminated and $T_1(\xi)$ is the Fourier transform of one period. The sinusoidal term of (3.51) has period $\Delta_1 = \lambda / T_0$. As $N \rightarrow \infty$ the sinusoidal term approaches a Dirac delta function. Sampling of the Fourier transform is equivalent to performing the Fourier series expansion. The period Δ_1 is inversely proportional to the cell size T_0 .

Figure 3-12

One cell of a sample two dimensional Dammann grating derived from the 1-D grating shown in Figure 3-11. The black areas indicate π phase shift, while the white areas indicate 0 phase shift.

One advantage of Dammann gratings is their high diffraction efficiency with binary phase quantization. Measured efficiencies greater than 60% for 25x25 spots [3.11] have been achieved. These repeated cells can make use of

step and repeat cameras to generate the required number of repetitions. A major limitation is that a uniform collimated beam and a lens is required to perform the Fourier transformation.

3.4.3 Iterative Holograms

In this type of holograms, the desired field at the hologram plane is neither known nor calculated, which is unlike that of the types previously discussed.

(One type of iterative hologram is the direct binary search (DBS) hologram [3.13]. This hologram is a Fourier transform hologram and thus require a lens to reconstruct the image. The cell at the bottom of Figure 3-10 may be either completely transparent or opaque for an amplitude DBS hologram.

The procedure to determine the state of a cell is as follows:

1. Randomly set the state of each cell
2. Toggle the state of a single cell
3. Calculate the Fourier transform of the cell
4. Compare the transform with the desired result and calculate an error measure (such as MSE)
5. If the new measure is better than the previous measure, then accept the state change. Otherwise reverse then state change.
6. If the error measure is too high, continue with step 2 for another cell.

This procedure produces images that have higher quality than can be obtained by other methods. All digital methods, such as the detour phase method, introduce quantization error. The DBS may take advantage of this error to

improve the image because the search is performed on many possibilities.

As with all Fourier holograms there will be a conjugate image about the DC component. Thus an implicit carrier must be introduced, otherwise the two images would overlap. This can simply be performed by making the transform plane larger than the desired image and then performing the error measure on the area of interest as show in Figure 3-13. The number of elements is $n \times m$. The desired image is contained in a $r \times s$ element area on which the error measure is performed. Notice that the desired image is shifted from the DC order. This is to prevent the desired image from overlapping its conjugate.

Figure 3-13
Image plane for a DBS hologram showing constrained region and the Hermitian symmetric region.

The algorithm can be seen to be highly computationally intensive because many Fourier transforms must be performed. The minimum number of Fourier transforms is $m \times n$ 2-d transforms of size $m \times n$. A method [3.14] to reduce the computations by over two orders of magnitude has been found.

Another concern is that these algorithms have not been proven to converge to the global optimum. From work using simulated annealing [3.15], it appears that these algorithms find one of many solutions that are near optimum.

The image quality using this method is quite good, which comes from accurate wavefront reconstruction. The drawback is that the efficiency is very low: on the order of 1% of the incident light is in the desired image. The other characteristic is that the unconstrained region is very bright.

An alternate method to generate spot arrays, which is a hybrid between Dammann gratings and DBS holograms, is termed iterative discrete on-axis (IDO) [3.15]. This method contains cells that are replicated, like Dammann gratings. The cell is divided into a fixed number of sub-cells like the DBS method. A search similar to the DBS method is made, but the error criterion is calculated only at the desired spot locations and assuming infinite repetitions. For an array of 7×7 spots, efficiencies of 74% have been achieved by Feldman [3.15]. This compares favorable to a Dammann grating efficiency of 43%.

3.4.4 Direct Computer Generated Holograms

Unlike the Fourier transform holograms, a lens is not needed in order to create the desired image for a direct computer generated hologram. Fewer components translates to easier alignment and lower cost. This is the computer

generated equivalent of the conventional hologram in section 3.3.1

Direct CGHs quantize the interference pattern at the plane of the hologram. The field at the hologram between a point source and a plane wave is given by

$$U = \frac{A_1}{r} \exp(-jkr) + A_2 \exp[jk(\alpha x + \beta y + \gamma z)]$$

$$r = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2},$$
(3.52)

where (α, β, γ) are the direction cosines of the plane wave of amplitude A_2 and (x_0, y_0, z_0) is the location of the point source of amplitude A_1 . The preceding equation can be simplified by assuming that the plane wave is parallel to the hologram and the point source is located on the z-axis,

$$U = \frac{A_1}{r} e^{-jkr} + A_2, \quad r = \sqrt{x^2 + y^2 + z_0^2}$$
(3.53)

When $(x^2 + y^2) \ll z_0^2$ and $A_1/r = A_2$ then the field at the hologram only has phase variation $\exp(-jkr)$. Such a setup is shown in Figure 3-14.

Binary phase quantization of the phase in equation (3.53) results in

$$k\sqrt{x^2 + y^2 + z_0^2} = n\pi,$$
(3.54)

where n is an integer. Normalizing to zero phase at $R_n=0$ we have,

$$k\sqrt{x^2 + y^2 + z_0^2} - kz_0 = n\pi$$

$$R_n^2 = n\lambda z_0 + \left(\frac{n\lambda}{2}\right)^2, \quad R_n^2 = x^2 + y^2$$
(3.55)

Equation (3.55) gives the radii R_n of phase changes and a sample hologram is shown in Figure 3-15. This equation also corresponds to the transitions for a Fresnel zone plate.

Figure 3-14
Setup to record the interference pattern between a point source and plane wave.

For a more complicated configuration, such as with multiple point sources, the quantization generally does not result in an analytic solution. In this case, a pixelated approach is more appropriate. The field at each pixel location is calculated and then quantized. An example of this type of multiplexing is shown in Figure 3-16. A limitation with this approach is that a huge amount of data is generated when the required resolution is small. If each pixel is only $0.5\ \mu\text{m}$ on a side, then the data requirements are half a megabyte per mm^2 for binary quantization. These limitations can be overcome

Figure 3-15

Binary quantization of at the recording plane between a plane wave and a point source. The difference between black and white is π . If either the dark or white portions are opaque, then this Figure is also a Fresnel zone plate.

by using a high speed computer with large storage capacity. The payoff for overcoming the limitations is that these HOEs produce smaller spots for the same overall area.

Figure 3-16
A 1 to 5 hologram generated by the pixelated approach.

Chapter 4

DESIGN AND CHARACTERIZATION OF COMPUTER GENERATED HOLOGRAPHIC ELEMENTS

4.1 Zone Plates

The Fresnel zone plate (FZP) is the simplest of the general class of zone plates. As explained in section 3.3.4, the Fresnel zone plate has transitions equivalent to a holographic pattern created by the interference between a point and plane wave. A more intuitive method of determining equation (3.54) is shown in Figure 4-1. This method depends on finding path differences. The location of each zone edge is an integral number of half-wavelengths away from the point source. The radius can be found by the Pythagorean theorem,

$$R_n^2 = z^2 + \left(z + n \frac{\lambda}{2} \right)^2 \quad (4.1)$$

$$R_n^2 = n \lambda z + \left(n \frac{\lambda}{2} \right)^2.$$

If more phase levels are available, then equation (4.1) becomes

$$R_n^2 = 2n \frac{\lambda}{L} z + \left(n \frac{\lambda}{L} \right)^2. \quad (4.2)$$

Here L denotes the number of phase levels, and z is equivalent to the focal length of a normal lense. The diffraction efficiency of these holograms and any other direct binary phase holograms is given by [4.1]

$$\eta = \left(\frac{\sin(\pi/L)}{\pi/L} \right)^2. \quad (4.3)$$

For an on-off hologram, half of the incident power is transmitted. Thus the maximum efficiency is only $\left(\frac{1}{2}\right)^2 40.5\% = 10.1\%$, which is a quarter of the efficiency achievable for binary phase quantization.

Figure 4-1
Path difference method of generating FZP.

For a point source imaging to another point source, each at a distance z from the hologram, the n^{th} radius is given by

$$R_n^2 + z^2 = \left(\frac{n\lambda}{4} + z \right)^2 \quad (4.4)$$

$$R_n^2 = n\lambda F + \left(\frac{n\lambda}{4} \right)^2, F = \frac{z}{2}.$$

Here, F is equivalent to the focal length of a regular lens.

Figure 4-2
Diffraction efficiency as a function of binary phase quantization levels

The minimum feature size of the CGHOE determines the maximum numerical aperture (NA). The well know grating equation is

$$a(\sin \theta_m - \sin \theta_i) = m\lambda, \quad (4.5)$$

where a is the grating period, m is the m^{th} diffraction order, and θ_i is the incident angle. Any small portion of the HOE can be viewed as a grating. The NA in air is defined as $NA = \sin \theta$. For HOEs the diffraction order of interest is $m=1$ and for the simplest case the incident angle is zero. Thus the NA of HOEs with binary quantization is

$$NA = \frac{\lambda}{2\Delta R}, \quad (4.6)$$

where AR is the minimum feature size, and $2AR$ is equivalent to the grating period a . The graph in Figure 4-3 shows how the NA varies with various minimum feature sizes.

Figure 4-3

This graph shows the maximum NA possible for a given minimum feature size at 5 different wavelengths.

The feasibility of an ordinary laser printer to create holograms was tested by generating a FZP. The laser printer used was a Hewlett-Packard LaserJet (HPLJ) Series II. This printer has a rated resolution of 300 dots per inch (dpi), which corresponds to $85\text{ }\mu\text{m}$. The expected maximum NA for $\lambda=633\text{ nm}$ is 0.0037. This NA is too small to experiment with comfortably on most optical tables. This means that the minimum feature size must be reduced. One method of getting smaller features is by photoreduction.

The HPLJ has a bitmapped data mode in which each pixel may be turned on or off. The printer expects each data byte to represent 8 pixels, thus each bit encodes the state of a pixel. The program to generate a data file is listed in appendix A. The central portion of a FZP for $\lambda=633$ nm and focal length of 0.4m that is to be reduced 10 times is shown in Figure 4-4.

Figure 4-4
FZP for $\lambda=633$ nm and $F=400$ mm is shown at 10x magnification,
produced by a Hewlett-Packard LaserJet laser printer.

The printout shown in Figure 4-4 was photographically reduced by a factor of ten onto a glass plate coated with high-resolution emulsion. This one time reduction caused the black regions on the printout to be transparent on the plate and the white regions to be opaque. This HOE was then placed into the setup shown in Figure 4-5.

Figure 4-5
Test setup for laser printer generated FZP

The point spread function of the FZP can be calculated by adding up the contributions of each ring. This field at the observation plane may be calculated by using the Fresnel-Kirchoff diffraction formula (3.17). The field for one ring is given by:

$$U(x, y, z) = \frac{E_0 \pi i}{\lambda} \int_0^a \rho J_0 \left(\frac{2\pi r}{\lambda} \sqrt{\rho^2 + r^2 + z^2} \right) \exp(jk \sqrt{\rho^2 + r^2 + z^2}) d\rho, \quad (4.7)$$

$$r^2 = x^2 + y^2,$$

where a and b are the respective inner and outer radii of a single transparent ring and ρ is the variable of integration. This equation is valid when $z^2 \gg r^2 + b^2$. Figure 4-6 shows the point spread function of various numbers of rings calculated by integrating (4.7).

Figure 4-6
Point spread function for various numbers of transparent zones of a
FZP with $f=0.4\text{m}$ and $\lambda=632.8\text{nm}$.

The aperture in the test setup of Figure 4-5 was varied to allow different numbers of transparent rings to be illuminated. The location of the first minimum in the line spread function was compared to the radius of the largest

illuminated ring. The results are shown in Figure 4-7, which shows a good correspondence between theory and experiment. At larger apertures, the point spread function is no longer circularly symmetric. This is due to limited resolution in the laser printer. These aliasing components can be seen in Figure 4-4 as gray circular entities at the periphery.

Figure 4-7
Relationship between location of the first minimum and FZP size.

4.2 Multifacet HOE

The first use of multifacet HOEs was to perform wavefront transformations [4.2]. Kostuk [4.3] used optically recorded multifacet HOEs to perform signal distribution. Hossack [4.4] created computer generated multifacet HOEs for fanout using optical coordinate transformations.

In this thesis, computer generated multifacet HOEs are used to split one input beam into many output beams and thus provide fanout. Each facet of the HOE focuses the light onto a different location. If the input is a collimated beam then each facet will be a section of a FZP of the desired focal length. Thus, each facet is a direct computer generated hologram. Figure 4-8 shows a possible multifacet-HOE and its relation to a zone plate.

The design of a multifacet HOE begins with partitioning the hologram into separate facets. The size and shape of each facet is determined by the intensity distribution of the input beam. If the desired outputs are to have equal powers, then the power illumination of each facet must be equal. This criterion determines the size and shape of each facet. The next step is then to determine the start and end of each holographic fringe that fits in each facet. This information can then be used to fabricate the HOE.

The design of a HOE is facilitated if the pattern can be easily inspected before being committed to an actual device. This can be accomplished by using a computer aided design/drafting (CAD) program. One such CAD package available for many types of computers is AutoCAD [4.5]. It has the ability to display the entire HOE pattern or any portion thereof at any magnification. The program can also send a drawing to a printer or plotter for hard copy output.

Another feature of AutoCAD is its built-in programming language called AutoLISP [4.6]. This programming language can be used to generate and

Figure 4-8

Construction and use of a multifacet HOE.

The different regions of the zone plate that are used to create the HOE is shown in a). Part b) shows the HOE along with the focal point for each facet. c) is a side view of b).

manipulate AutoCAD entities and displays. (The elements of an AutoCAD drawing database are called entities.) This allows the drawing to be displayed as it is being created by an AutoLISP program.

The AutoLISP program used to generate multifacet HOEs is listed in the section A.2 of the appendix. This program is designed to create a transmission HOE pattern. It assumes that the input is a collimated beam. The shapes of each facet in this program is square.

The program HOE.LSP works by creating the pattern for one facet at a time. For each facet, the part of a FZP which lies within the square has to be calculated. The range of the maximum and minimum zone numbers is determined. The minimum zone number is one if the xy coordinate of the focus lies within the facet, otherwise the distances from the focus to the nearest side and to the corners of the facet are calculated. The maximum and minimum of these radial distances are used to solve equation (4.1) for the range of zone numbers. Each zone is circular, so that the intersection of a circle and a square has to be calculated. The 8 possible intersection scenarios are shown in Figure 4-9. The sections of a zone which lie within the square are determined by finding the correct scenario. The sections are drawn and the program continues to other zones and facets.

The user of this program creates a command file which contains the information for each facet. The format of this file is:

```
(defun multi ()
  (setq var "0" 1) "0"
  (drawHOE lam f (fx fy) (fx fy) 1)
  (setq var (decr var "1"))
)
```

The actual numerical values, in metres, are substituted as follows:
 lam=wavelength of light, f=focal length, fx=x coordinate of focus, fy=y coordinate

of the focus, qx =x coordinate of the lower left corner of the facet, qy =y coordinate of the lower left corner of the facet, len =length of a side of the facet. The line `(car-assif 0) . . .` is repeated for each facet.

The process to actually create the pattern begins by invoking AutoCAD. Next, the program HOE.LSP is loaded by entering `Command:(load "hoe.lsp")` at the AutoCAD command prompt. Following this, the facet command file is loaded in a similar fashion. `Command:mult` is entered at the prompt to start the drawing. The program is concluded when the command prompt reappears.

Figure 4-9

All possible type of intersections between a circle and a square in which all or parts of the circle lie within the square. The numerals in each square indicate the number of intersection points.

spots is shown in Figure 4-11.

The intensity profile at the focal plane shows that the spots do not have uniform intensities. The main reason for this is that the mylar plot was not consistent. The very fine plotter pen did not deliver a consistent line width. The facets with very low intensities resulted from the lines that were too thin. These HOE facets, negative images of the plot, were more opaque than desired.

The intensity profile does not show the intensity going to zero. This is a result of the modulation transfer function (MTF) of CCDs. The modulation depth versus spatial frequency drops off rapidly as the wavelength increases. A typical CCD (Fairchild CD142) has a MTF of 0.3 for a wavelength of 1100nm and spatial frequency of 7.7 cycles/mm.

Figure 4-10
Setup for testing 1:16 HOE.

Figure 4-11
Intensity profile through centre of focused spots

The results show that a 1:16 HOE can be fabricated by photoreducing the output from a plotter. Pen plotters have difficulty maintaining uniform line width control over large areas which may be 50% filled. HOEs produced by this method will have to control the plotting process carefully if uniform output is to be obtained. However, the advantages of this method are low cost, it is readily available, and fast turn around time.

4.2.2 SLM Controlled 1 to 8 HOE

An alternate solution to the pen plotter problem is to use high resolution phototypesetters normally used for the publishing industry. Resolutions as high as 2540 dots per inch or equivalently 10 μ m resolution are available. Most such systems take a Postscript data file for input. Postscript is a programming language specific to printers containing this interpreter.

AutoCAD has the ability to generate Postscript output. However, this output does not use the proper commands available. To appreciate the problem, consider the treatment of arcs. The Postscript file generated from AutoCAD contains commands to draw line segments to approximate arcs, but Postscript has an internal command to draw arcs at the resolution of the device. The result is that AutoCAD output will have bumpy curves. The other problem with AutoCAD output is that this approximation is not user controllable.

The solution to the AutoCAD Postscript output problem is to directly generate Postscript commands. Although Postscript has a large command set, only a small number will be necessary. The program HOE.LSP only needs a small modification to generate Postscript output, as it already calculates the arc locations.

The modifications to HOE.LSP include data file output functions and Postscript output. This necessitates a slight change to the HOE specification command file as follows:

```
(defun c:mult ()
  (setvar "mdecho" "0")
  (file_header)
  (drawHOE lam f () '(fx fy) '(qx qy) len)
  (file_footer)
  (setvar "mdecho" "1")
)
```

Additionally, when Command:plot is executed, the user is prompted for an output file name, magnification, and printer resolution. This replacement for HOE.LSP is PS-HOE.LSP.

```

; Plot file to draw a 1 to 16 HOE
(plot:prompt)
(setq val "hole no" 0)
(title:header)
(drawHOE 0.05 5.12e-9 0) * (2.894e-3 2.444e-3) * (2.794e-3 0) * (0.01:97)
(drawHOE 0.05 5.12e-9 0) * (2.894e-3 2.894e-3) * (2.794e-3 1.397e-3)
(plot:plot)
(drawHOE 0.05 5.12e-9 0) * (2.894e-3 2.894e-3) * (2.794e-3 2.794e-3)
(plot:plot)
(drawHOE 0.05 5.12e-9 0) * (2.894e-3 3.394e-3) * (2.794e-3 4.191e-3)
(plot:plot)
(drawHOE 0.05 5.12e-9 0) * (3.394e-3 2.444e-3) * (4.191e-3 0) * (0.01:97)
(drawHOE 0.05 5.12e-9 0) * (3.394e-3 2.794e-3) * (4.191e-3 1.397e-3)
(plot:plot)
(drawHOE 0.05 5.12e-9 0) * (3.394e-3 2.894e-3) * (4.191e-3 2.794e-3)
(plot:plot)
(drawHOE 0.05 5.12e-9 0) * (3.394e-3 3.394e-3) * (4.191e-3 4.191e-3)
(plot:plot)
(title:footer)
(setq val "hole no" 1)

```

The test setup for this HOE is shown in Figure 4-12. Initially, the spatial light modulator (SLM) and Fourier transform filter were not included.

The results of Figure 4-13 show that the spots are quite uniform. Spot number 4 is slightly out of shape. This is a result of the phototypeset film curling up during the photoreduction process. The reduced intensity of the two adjacent spots is also due to this problem.

Figure 4-12
Setup for SLM controlled 1:8 HOE.

Figure 4-13
Image at the focal plane showing the 8 output spots.

Figure 4-14
Horizontal intensity profile of 1:8 HOE output. The scale for the
abscissa is valid only for each group of 4 points.

In the next part of the experiment, a spatial light modulator was used to control the spots. The SLM used is a Semetex Fe: garnet model SM-48. The size of the display is 48x48 pixels with a period of 127 μm and an active area of 109 μm^2 .

An array of 11x11 pixels map to one facet of the HOE. There are gaps between each of the pixels in the SLM so that the HOE would not be illuminated evenly. The Fourier transform filter is setup as a low pass filter. The result is

that the light hitting the HOE is no longer pixelated but is uniform. Some results are shown in Figure 4-15. The left side shows the patterns sent to the SLM and the right side are the digitized output spots.

The use of a high resolution phototypesetter results in a good HOE, which results in uniform spots. A SLM can be used to control each individual facet of and HOE. A signal may also be imposed on the SLM to modulate each output spot independently.

Figure 4-15
SLM controlled HOE input and output. The input to the SLM is shown at the left. The resulting spot patterns are shown at the right.

4.3 Binary Phase HOE

The maximum efficiency of binary phase HOEs is 40.5%. This is 4 times higher than that of on-off HOEs. One method of producing this phase shift is to introduce a differential path length. Figure 4-16 shows several configurations of reflection and transmission to introduce π phase shift. The tilted reflection HOE requires a $\lambda/4$ thickness, which is the same as the perpendicular case. This is because the reflection must appear to take place at the same plane throughout. Each point on the virtual boundary acts as a Huygens wavelet. These wavelets then reflect off the back surface to arrive at the entry point but with a π phase shift.

A reflection HOE would be difficult to evaluate if the focus fell back upon the input beam. An off-axis HOE is amenable to evaluation. The off-axis HOE is required for the system shown in Figure 2-4(b), except in transmission.

The off-axis configuration means that the zones will no longer be circular but will be elliptical. The shape of the zones for the configuration shown in Figure 4-17 can be found by calculation of the path differences.

The field for a plane wave and point source is:

$$\begin{aligned} \text{plane wave} \quad E_i &= E_{i0} \exp[jk(z + y \sin \alpha)] \\ \text{point source} \quad E_p &= \frac{E_{p0}}{r} \exp(jkr), \quad r = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2} \end{aligned} \quad (4.8)$$

When the two beams interfere, the phase terms can be added together. The locus of points with integral number of half waves is:

$$n \frac{\lambda}{2} = \sqrt{x^2 + (y - y_0)^2 + z_0^2} + y \sin \alpha, \quad (4.9)$$

where n is an integer. After some algebraic manipulations, the result is the form of an ellipse:

Figure 4-16
Different methods of introducing π phase shift in a transmission and reflection HOE.

$$\begin{aligned}
 1 &= \frac{x^2}{a^2} + \frac{(y - y_0)^2}{b^2} \\
 a^2 &= \frac{(n\lambda \sin \alpha - 2y_0)^2}{4 \cos^2 \alpha} - \left[y_0^2 + z_0^2 - \left(\frac{n\lambda}{2} \right)^2 \right] \\
 b^2 &= \frac{a^2}{\cos^2 \alpha} \\
 y &= \frac{2y_0 - n\lambda \sin \alpha}{2 \cos^2 \alpha}
 \end{aligned}
 \tag{4.10}$$

This equation shows that n must be chosen such that a^2 is positive in order to have physical meaning. The critical value of n is given by the number of half wavelengths from the origin to the point:

$$n > \frac{2 - y_0}{\lambda \sin \alpha} \tag{4.11}$$

Equations (4.10) and (4.11) offer the solution for the zone locations of off-axis plane-point source HOEs.

The previous sections illustrate some of the drawbacks to using photoreduced HOEs. Each additional step in the process tends to introduce errors. An electron beam mask for the HOE reduces the accumulation of errors. It is also a very high resolution system with step sizes down to 100 nm. This permits large NA systems to be possible.

The Northern-Telecom e-beam mask facility is equipped to process data files in the GDS2 CAD format. Any shapes that are to be drawn in GDS2 are limited to polygons with a very limited number vertices. This means that any complex shape must be approximated with collections of polygons, each of which must have less than 200 vertices.

AutoCAD does not produce GDS2 output as an option. But there are programs which convert AutoCAD format to GDS2. One such program is

Figure 4-17
Off-axis HOE for a point source and a plane wave. The HOE is located in the xy plane.

available from Octal, Inc. This program takes a AutoCAD intermediary format and converts it to GDS2.

The problem then reduces to creating entities in AutoCAD which are made up of closed polygons with zero line width and less than 200 vertices. The program PRISM.LSP is used to generate an off-axis HOE which satisfies these constraints.

PRISM.LSP is currently limited to generating facets with complete zones. It is currently coded for a plane wave angle of 45 degrees. Each elliptical zone is composed of 4 quadrants. Each quadrant is approximated by a polygon with 90 vertices. This corresponds to 2 degree chords.

A single facet with a focal length of 20 mm and minor diameter of 2.5 mm was generated. This facet was made into an AutoCAD block. Complex entities that are to be repeated are made more efficient by grouping them into a block. This improves both AutoCAD performance and the GDS2 conversion speed. This block is repeated 4 times for the 1:4 splitting. Additional test gratings were added into the drawing manually. Figure 4-18 shows the 4 elliptical facets at the top and the test gratings below.

Figure 4-18
1:4 HOE AutoCAD patterns shown at 14 times magnification.

The setup shown in Figure 4-19 is used to evaluate the HOE. The focused spots are approximately 10 μm in diameter so that direct imaging on to the CCD would not provide high enough resolution. Thus each facet had to be imaged separately. Figure 4-20 shows a composite intensity profile of each spot. The theoretical intensity profile of a spot is compared with that of experimental spot number one in Figure 4-21.

Figure 4-19
Setup used to evaluate the 1:4 HOE.

Figure 4-20
Composite Intensity profile for each of the four focal spots. The spot
images are shown below the graph.

Figure 4-21
Theoretical and experimental intensity profiles of one spot of the 1:4
binary phase HOE.

The procedure that was used to fabricate a binary phase HOE is shown in Figure 4-22. The first step was to coat a silicon wafer with a quarter wavelength thick layer of SiO_2 . A layer of photoresist was spun on top of the SiO_2 . The photoresist was then exposed to the HOE pattern. The exposed photoresist was then developed. Unexposed regions of photoresist and the SiO_2 layer underneath are chemically etched away. The exposed resist was removed, leaving the HOE pattern on the wafer. A reflective metal coating was then applied to form the finished HOE. The coating was a layer of >127 nm thick

Figure 4-22
The steps used to create a binary phase HOE.

2 of/de 2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

aluminum. This thickness corresponds to 10 skin depths δ given by:

$$\delta = \frac{\lambda}{2\pi k_m}, \quad (4.12)$$

where k_m is the imaginary component of the index of refraction for the metal.

The major advantage in going from amplitude to binary phase HOEs is a 4 fold increase in efficiency from 10.1% to 40.5%. The measured efficiency for the four spots are:

Spot 1	38.9%
Spot 2	38.3%
Spot 3	37.1%
Spot 4	36.6%

These values are a few percentage points lower than the ideal case. One important factor affecting the efficiency is the accuracy of the phase shift. The effects of various phase shifts on efficiency was studied by using computer simulations. The simulation results as well as the experimental points are plotted on Figure 4-23. This shows how the efficiency varies as a function of phase shift. An 10° phase error results in approximately 1.5% drop in efficiency. Another fabrication parameter, the exposure time, affects the width of the etched regions, and thus affects the efficiency. Results from a different set of simulations are shown in Figure 4-24. Here, the relative efficiency (referenced to correct exposure) is shown as a function of exposure. It shows that correct exposure is very important in maintaining high efficiency, because there is a rapid drop off from the correct exposure. Incorrect exposure affects the HOE by change the widths of the fringes.

Figure 4-23

Effects of phase error on diffraction efficiency, where 180 degrees is ideal for a binary phase HOE. The simulation conditions are: 256x256 FFT over 5mmx5mm area, $\lambda=633\text{nm}$, $F=100\text{mm}$, Rings=20. The experimental points are shown as circles.

The results from the binary phase hologram show a good match between the experimental and theoretical efficiencies. The improved efficiency over the previous HOEs is achieved at a higher cost and fabrication complexity. The uniformity of the spots which was achieved, highly recommends the one-step process for generating an accurate mask.

Figure 4-24

The relative efficiency is affected by either over or under exposure during fabrication. The simulation conditions are: 2048x2048 FFT over 2 mm x 2 mm area, $\lambda=532\text{nm}$, $F=20\text{mm}$, Rings=30, minimum feature size of $6.7\text{ }\mu\text{m}$.

Chapter 5

LIMITS AND REQUIREMENTS OF HOEs

The discussion in this section applies to thin digital holograms such as those evaluated in chapter 4.

The minimum feature size requirement is dictated by the NA and the operating wavelength. This dependency is given by equation (4.6) which can be written as

$$\Delta R = \frac{\lambda}{2 \cdot NA} \quad (5.1)$$

The NA refers to the angle subtended from the focus to the farthest point on the HOE as illustrated in Figure 5-1. Equation (5.1) is valid for binary phase HOEs illuminated by plane waves. For other types of illumination, the grating equation (4.5) must be used. The minimum feature size for multilevel phase HOEs is obtained by dividing the binary phase value by the number of levels.

The minimum feature size in the preceding refers to a device which can place the patterns without restriction. If the output device has fixed locations for the output, then this becomes a sampled system. The sampling rate (resolution) should be less than one quarter of the minimum feature size of the HOE. The location of the holographic fringes must be mapped to the nearest pixel. This mapping of real values to integers should occur by rounding, which minimizes the generation of sidelobes due to the sampling process [5.1].

Diffraction optics, such as HOEs, depend upon interference effects. This translates to coherence requirements. The lateral spatial coherence must be greater than the area of a facet. A conservative coherence length is given by

Figure 5-1
Determination of NA for the purpose of calculating minimum feature size requirements.

the maximum distance on the surface of a facet.

The interference effects also give rise to pulse broadening. The worst case pulse broadening is determined by the maximum extent of a facet. This value is given by L/c seconds, where L is the maximum extent and c is the speed of light. For example, a 1 mm facet results in broadening of approximately 3 ps, which corresponds to a bandwidth of 300 GHz.

The HOEs are sensitive to wavelength fluctuations. This can be seen in equation (4.4). To a large degree, a change in wavelength is equivalent to changing the focal length. This results in an increase in spot size. Gaussian

beam analysis can be applied to approximate this increase. The change in wavelength is transformed to a change in focal length. The following equation can then be used [5.2],

$$w(z) = w_0 \sqrt{1 + \frac{\lambda^2 z^2}{\pi^2 n^2 w_0^4}}, \quad (5.2)$$

where w_0 is the spot size of the unchanged system, n is the index of refraction of the medium, and z is the change in image location. The value z can be calculated by using the standard lens equation $1/F = 1/d_1 + 1/d_2$, where d_1 and d_2 are the location of the source and image, respectively.

The physical mounting location also affects the size and location of the output spot. If the distance from the HOE to the source and image differs by Δh , then this difference may be substituted into equation (5.2) for z . A small tilt of the HOE by angle ϵ results a shift in the image location by [5.2]

$$\Delta x = h[\cot(\theta - 2\epsilon) - \cot\theta]. \quad (5.3)$$

The simulation results shown in Figure 4-23 and 4-24 show that both curves are quite shallow at the optimum values. This means that manufacturing tolerances should be maintained in those regions to avoid excessive efficiency degradation.

The fundamental number of fanouts F for a specified hologram area is given by Feldman [5.3]:

$$F \leq \frac{2A\Omega}{\lambda^2}. \quad (5.4)$$

where Ω is the solid angle subtended by the input beam and λ is the wavelength. Unless the area of the hologram approaches the magnitude of λ , this does not limit the number of fanouts. There is, however, a practical limit on the number

of transmitters that can be accommodated. This is simply due to diffraction of the transmit beam as it propagates to the HOE, and to limited HOE resolution. The number of transmitters N that can be accommodated is [5.3]:

$$N < \frac{w_s^2 \tan^2 \Phi}{4} (1.2)^2 \lambda^2, \quad (5.5)$$

$$\Phi = \tan^{-1} \{ \tan \sin^{-1}(2\lambda/\text{resolution} - \sin \theta/2) - \tan \theta/2 \},$$

where w_s is the source beam waist, Φ is the maximum deflection angle, *resolution* is the minimum feature size, and θ is the source divergence angle. At $0.8\mu\text{m}$ wavelength and 500nm resolution, the maximum number of sources is approximately 39, even with a laser having a low divergence of 1.5° .

Fanouts using HOEs will have some skew (in time) between each output. This skew is determined by the path difference from one output to another. The worst case skew is given by L/c seconds, where L is the maximum distance from one facet to another and c is the speed of light. Thus, the maximum skew for a 1 cm HOE is 33 ps . The value of skew between any outputs is known deterministically from the path difference and thus compensation may be made.

Chapter 6

CONCLUSIONS

6.1 Summary

There are many advantages to using optical interconnects rather than electrical interconnects. Optical interconnects have high bandwidth, low interference, and lower power. These requirements are necessary for subsequent generations of high speed computers. This is born out by the fact that many research projects, around the world, are concentrating on optical interconnects.

The three main methods for propagating light from one location to another are fibre optics, wave guides, and free-space propagation. Fibre optics is excellent for long distances in terrestrial links. A clear winner has yet to emerge in the competition between waveguides and free-space propagation for the shorter distances. Waveguides are good for integration onto one substrate. On the other hand, free-space propagation is not constrained to one or two dimensions. This utilization of the third dimension enables the creation of denser circuits.

A necessary component for free-space communications is a focusing element. Focusing controls the spread of light so that it is not wasted. Lenses are suitable in many applications, but the computer generated holographic optical elements can be more versatile. CGHOE can perform more than one function at a time, such as splitting a beam into many others and focusing at the same time.

During this work, three different types of CGHOE were created and tested. The simplest was a FZP created on an ordinary laser printer and photo-reduced. The performance was verified against theory. Although low in cost, the resolution ($\approx 8.5\mu\text{m}$) was not high enough to generate small spot sizes.

Two different multifacet HOE were produced and evaluated. The first HOE split and focused the input into 16 spots. The initial pattern was created on a drafting plotter and then photographically reduced. The problem with this method was that the pen did not remain consistent throughout the entire large D-size plot. But it did show that a multifacet CGHOE splits and focuses the input. The second HOE was printed on a high resolution phototypesetter and reduced also. The eight output spot pattern was much more uniform. This HOE was also placed in a SLM system that demonstrated the output from each facet could be modulated independently.

The third HOE type constructed was a binary phase type. The key feature is that the theoretical efficiency increased to 40.5%. The reflection binary phase HOE is very compatible with standard silicon IC fabrication processes. The pattern was made in one step on an electron beam mask making machine. This obviates the need for photoreduction because of the high resolution (100nm). The actual HOE was an etched SiO_2 on silicon wafer coated with aluminum. The HOE performed as expected with efficiencies near 40%. The uniformity between spots was very good.

It has been shown that good quality HOEs with good efficiencies can be fabricated. Lower cost methods for the production of HOEs were also demonstrated.

6.2 Conclusions

The use of a single HOE for signal distribution is limited to VLSI. The need to maintain a compact system imposes this restriction. The size of the HOE in the configuration shown in Figure 2-4(a) should be approximately the same size as the die. The height of the HOE above the surface will be approximately the same as the die size.

The longer interconnection lengths on a circuit board will require a system such as that shown in Figure 2-4(b). This method maintains a relatively compact system and still provides the desirable optical interconnections.

Free-space interconnections using Fourier HOEs can provide fanouts (ie. thousands per cm^2) that are necessary in massively parallel computers. This density of high speed interconnections would be very difficult to implement by electrical means.

Clock distribution using optical interconnections provides very low skew even for large fanouts.

The interconnection bandwidth using free-space propagation and HOEs is approximately 300 GHz (assuming 1 mm facets). This is much higher than that of current systems and thus these interconnections will be the limiting factor.

6.3 Future Directions

The diffraction efficiency of 40% for binary phase holograms can be improved by increasing the number of phase levels as shown in Figure 4-2. A two mask process can produce a three-level or four-level holograms with 70 and 80 percent efficiencies, respectively.

Computer originated HOEs (COHOE) have the high efficiency of analog holograms, but they can have the arbitrary wavefronts of computer generated holograms. The COHOEs are created by only copying the information carrying diffraction order of a regular HOE onto a volume holographic material, such as dichromated gelatin.

Diffraction-reflective systems such as those proposed by Sauer [2.8, 5.2] should be studied since they allow extended reach while maintaining a low system volume.

References

- 1.1 J. W. Goodman, F. I. Leonberger, S. Y. Kung, R. A. Atale, "Optical interconnections for VLSI," *Proceedings of the IEEE*, Vol. 72, no 7, pp 850-866 (1984).
- 1.2 J. A. Neff, "Major initiatives for optical computing," *Optical Engineering*, Vol. 26, no 1, pp 2-9 (1987).
- 2.1 P. M. Solomon, "The need for low resistance in future high-speed systems," *Interconnection of High Speed and High Frequency Devices and Systems*, Alfred P. De Fonzo, Editor, Proc. SPIE 947, pp 104-119 (1988).
- 2.2 H. Kroger, C. Hilbert, U. Ghoshal, D. Gibson and, L. Smith, "Applications of superconductivity to packaging," *IEEE Circuits and Devices Magazine*, Vol. 5, no. 3, pp 16-21 (1989).
- 2.3 R. C. Frye, "Analysis of the trade-offs between conventional and superconducting interconnections," *IEEE Circuits and Devices Magazine*, Vol. 5, no. 3, pp 27-32, (1989).
- 2.4 P. R. Haugen, S. Rychnovsky, A. Husain, and L. D. Hutcheson, "Optical interconnects for high speed computing," *Optical Engineering*, Vol. 25, no. 10, pp 1076-1084, (1986).
- 2.5 P. R. Prucnal, "VLSI optical interconnection networks," *Integration and Packaging of Optoelectronic Devices*, Proc. SPIE 703, pp 106-115 (1986).
- 2.6 C. T. Sullivan, M. C. Roth, and T. Bubzynski, "Integrated optics approach for high-density optical interconnections in high-speed multichip IC packages," *Optoelectronic Materials, Devices, Packaging, and Interconnects*, Ted E. Batchman, Richard F. Carson, Robert L. Gallawa, Henry J. Wojtunik, Editors, Proc. SPIE 836, pp 343-350 (1988).

```

; then x is a number and y is a list
(if (= (length y) 1) ;check number of numbers in list
    (list x (car y))
    (list (list x (car y)) (list x (cadr y))))
(if (= (length x) 1) ; check number of numbers in list
    (append x (list y))
    (list (list (car x) y) (list (cadr x) y))))
)
;*****
;There are 2 intersections of the circle with the square
; Parameters: centre - centre of circle
;             width - width of arc
;             p_list - list of intersection points
;
(defun two_inters (centre width p_list)
  (print "two") (print p_list) ;*-
  (if (= (length p_list) 1)
      (progn
        (if (equal (caddr p_list) (caar p_list))
            (draw_circle centre (distance centre (caar p_list))
                          width)
            (draw_arc centre width (caddr p_list) (caar p_list))))
      (progn
        (arc_180 centre width (car p_list) (cadr p_list))))
  )
;*****
; draw an arc less than 180 degrees - based upon a cross product
; Parameters: c - centre of circle
;             width - width of arc
;             p1, p2 - end points of arc (ccw from p1)
(defun arc_180 (c width p1 p2)
  (if (minusp (- (* (- (car p1) (car c)) (- (cadr p2) (cadr c)))
                  (* (- (car p2) (car c)) (- (cadr p1) (cadr c)))))
      (draw_arc c width p2 p1)
      (draw_arc c width p1 p2))
  )
;*****
;
;
(defun four_inters (centre width p_list)
  (print "four") (print p_list) ;*-
  (if (= (length p_list) 2)
      (progn
        (draw_arc centre width (caddr p_list) (caadr p_list))
        (draw_arc centre width (cadadr p_list) (caar p_list)))
      (if (atom (caar p_list))
          (if (listp (caadr p_list))
              (progn
                (draw_arc centre width (car p_list) (caadr p_list))
                (draw_arc centre width (cadadr p_list) (caddr
p_list)))
              (progn
                (draw_arc centre width (cadr (caddr p_list)) (car
p_list))
                (draw_arc centre width (cadr p_list) (caaddr
p_list))))
          (progn
            (draw_arc centre width (caddr p_list) (cadr p_list))
            (draw_arc centre width (caddr p_list) (caar p_list))))
  )

```


- 3.7 H. M. Smith, Principles of HOLOGRAPHY, Wiley-Interscience, New York, New York, 1969.
- 3.8 W. H. Lee, "Computer generated holograms: techniques and applications," *Progress in Optics*, E. Wolf Editor, Vol. XVI, pp 119-232 (1978)
- 3.9 A. W. Lohmann and D. P. Paris, "Binary Fraunhofer holograms generated by computer," *Applied Optics*, Vol. 6, pp 679 (1967).
- 3.10 H. Dammann and K. Görtler, "High-efficiency in-line multiple imaging by means of multiple phase holograms," *Optics Communications*, Vol. 3, no. 5, pp 312-515 (1971).
- 3.11 X. Lu, Y. Wang, M Wu, and G. Jin, "The fabrication of a 25x25 multiple beam splitter," *Optics Communications*, Vol. 72, no. 3,4, pp 157 (1989).
- 3.12 J. Turunen, A. Vasara, J. Westerholm and A. Salin, " Stripe-geometry two-dimensional Dammann Gratings," *Optics Communications*, Vol. 74, no. 3-4, pp 245 (1989).
- 3.13 M. A. Seldowitz, J. P. Allebach, and D. W. Sweeney, "Synthesis of digital holograms by direct binary search," *Applied Optics*, Vol. 26, no. 14, pp 2788 (1987).
- 3.14 B. K. Jennison, J. P. Allebach, and D. W. Sweeney, "Iterative approaches to computer-generated holography," *Optical Engineering*, Vol. 28, no. 6, pp 629 (1989).
- 3.15 M. R. Feldman and C. C. Guest, "Iterative encoding of high-efficiency holograms for generation of spot arrays," *Optics Letters*, Vol. 14, no. 10, pp 479 (1989).

- 4.1 J. W. Goodman, A. M. Silvestri, "Some effects of fourier-domain phase quantization," *IBM Journal of Research and Development*, Vol. 13, pp 478-484 (1970).
- 4.2 S. K. Case, P. R. Haugen, and O. J. Løkberg, "Multifacet holographic optical elements for wave front transformations," *Applied Optics*, Vol. 20, no. 15, pp 2670-2675 (1981).
- 4.3 R. K. Kostuk, J. W. Goodman, and L. Hesselink, "Optical imaging applied to microelectronic chip-to-chip interconnections," *Applied Optics*, Vol. 24, no. 17, pp 2851-2858 (1985).
- 4.4 W. J. Hossack, P. McOwan, and R. E. Burge, "Computer generated optical fan-out element," *Optics Communications*, Vol. 68, no. 2, pp 97-102 (1988).
- 4.5 Autodesk Inc., AutoCAD Release 10 Reference Manual, Autodesk Inc., 1988.
- 4.6 Autodesk Inc., AutoCAD Release 10 Programmer's Reference, Autodesk Inc., 1988.
- 5.1 C.L. Chen and T. R. Osborne, "Quantization effects on the fields of electron-beam generated cylindrical zone plates," *Applied Optics*, Vol. 26, no. 12, pp 2342-2347 (1987).
- 5.2 K. H. Brenner and F. Sauer, "Diffractive-reflective optical interconnects," *Applied Optics*, Vol. 27, no. 20, pp 4251-4254 (1988).
- 5.3 M. R. Feldman and C. C. Guest, "Interconnect density capabilities of computer generated holograms for optical interconnection of very large scale integrated circuits," *Applied Optics*, Vol. 28, no. 15, pp 3134-3137 (1989).

Appendix A

PROGRAM LISTINGS

A.1 FZP.BAS

```

' generate Fresnel zone plate
' a spherical point source located at z0 away from the hologram
'
' Note: the data file contains binary data, therefore "PRINT" may
'       not work as it can read a eof character before the actual eof
'       Use instead "Copy/B datafile PRN", notice the /B.
DECLARE FUNCTION numremblank$ (x!)
DIM mask(8)
CONST esc = 27, ff = 12, dpi = 300
INPUT "Enter size of plot (inches) : "; size
INPUT "Enter distance to point source (m) "; z0#
INPUT "Enter scale of plot (eg. 2) : "; scale
INPUT "Enter output file name : "; filename$
bytes = INT(size * dpi / 8)
dots = bytes * 3 - 1
RANDOMIZE (1234)
OPEN filename$ FOR OUTPUT AS #1
pi# = 3.141592654#
radtodeg# = 180# / pi#
lambda# = .00000063#
k# = pi# / lambda# / z0# * radtodeg#
xrange = (dots + 1) / 300 * .0254 / scale: xoffset = xrange / 2
yrange = xrange: yoffset = yrange / 2
imax = 2
FOR i = 1 TO 8
    mask(i) = 2 ^ (8 - i) ' initialize mask
NEXT i
PRINT #1, CHR$(esc); "E"; ' reset printer
PRINT #1, CHR$(esc); "p300x400Y"; ' move to 300,400
PRINT #1, CHR$(esc); "t300R"; ' set 300dpi
PRINT #1, CHR$(esc); "r1A"; ' start at current position
SCREEN 12
yc = yrange / dots
FOR xcount = 0 TO dots
    PRINT "line "; xcount
    x# = xcount * xrange / (dots) - xoffset
    y# = x# * x#
    PRINT #1, CHR$(esc); "b"; numremblank$(bytes); "W";
    FOR bytecount = 0 TO bytes - 1
        byte = 0
        bc = bytecount * 8
        FOR bitcount = 1 TO 8
            ycount = bc + bitcount - 1
            y# = ycount * yc - yoffset
            phase# = k# * (x# + y# * y#)
            phase# = phase# MOD 360
            IF phase# > 180 THEN
                byte = byte OR mask(bitcount)
            END IF
        NEXT bitcount
    NEXT bytecount
NEXT xcount

```

```

        PSET (ycount, xcount)
    END IF
NEXT
PRINT #1, CHR$(byte);
NEXT
NEXT
PRINT #1, CHR$(esc); ""rB"           ' end graphics mode
PRINT #1, : PRINT #1,
PRINT #1, "      Fresnel zone plate at "; lambda*; "m wave-length"
PRINT #1, "      point located"; z0*; "m away from hologram"
PRINT #1, "      magnification x"; scale
PRINT #1, CHR$(ff); CHR$(esc); "E";
CLOSE #1
FUNCTION numremblank$ (x) STATIC
    x$ = STR$(x)
    numremblank$ = LTRIM$(RTRIM$(x$))
END FUNCTION

```

A.2 HOE.LSP

```

; this file draws fzps in autocad assuming the database units are
; in mm
;
; Les Chan 1989 March 1
;
(defun c:lf () ; temporary function to save typing
  (load "fzp"))
(defun c:red ()
  (command "edit" "fzp.lsp"))
(setq m_to_mm 1000.0) ; global conversion variable
; get information about a fzp and draw it
(defun c:fzp ()
  (setq f (getreal "\n Enter focal Length (m):"))
  (setq L (getreal "\n Enter Wavelength (m):"))
  (setq rings (getint "\n Enter number of Rings :"))
  (setq centre (getpoint "Enter Focal point (m,m):"))
  (setq start (getpoint "Enter lower left corner of hoe (m,m):"))
  len (getdist start "Enter length of square (m): ")
  (square start len)
  (drawHOE f L () centre start len))
; draw multiple fzps
(defun c:mult ()
  (setvar "cmdecho" 0)
  (drawHOE 0.05 1.064e-6 () '(0 62.5e-6) '(-5.0e-3 0) 0.0025)
  (drawHOE 0.05 1.064e-6 () '(0 187.5e-6) '(2.5e-3 0) 0.0025)
  (drawHOE 0.05 1.064e-6 () '(0 312.5e-6) '(-2.5e-3 0) 0.0025)
  (drawHOE 0.05 1.064e-6 () '(0 437.5e-6) '(0 0) 0.0025)
  (drawHOE 0.05 1.064e-6 () '(0 562.5e-6) '(-2.5e-3 2.5e-3) 0.0025)
  (drawHOE 0.05 1.064e-6 () '(0 687.5e-6) '(0 2.5e-3) 0.0025)
  (drawHOE 0.05 1.064e-6 () '(0 812.5e-6) '(-5.0e-3 2.5e-3) 0.0025)
  (drawHOE 0.05 1.064e-6 () '(0 937.5e-6) '(2.5e-3 2.5e-3) 0.0025)
  (setvar "cmdecho" 1)
  ;
  ; actually do the drawing of the fresnel zones using the donut command
  (defun drawfzp (F L rings scale / i)
    (setq i 1)
    (while (<= i rings)
      (setq outer (Rn F L (* 2 i)))
      (setq inner (Rn F L (1- (* 2 i))))
      ; donut expects dia. not rad. therefore multiply by 2
      (command "donut" (* outer scale 2) (* inner scale 2) centre)
      (setq i (1+ i))))
  ;
  (defun drawHOE (F L rings centre start len / i)
    (setq i (find_ring_range centre start len F L))
    (if (null rings)
      (setq rings (cadr i)))
    (setq i (car i))
    (prin1 "drawhoe") (print rings)
    (while (<= i rings)
      (print i)
      (setq outer (Rn F L (* 2 i)))
      (setq inner (Rn F L (1- (* 2 i))))
      (circ_square centre inner outer start len)
      (setq i (1+ i)))
    ;
    ;
    (defun find_ring_range (centre start len F L / i)

```

```

(setq i (list start (list (+ (car start) len) (cadr start))
                (list (+ (car start) len) (+ (cadr start) len))
                (list (car start) (+ (cadr start) len))))
(setq i (mapcar '(lambda (x)
                  (/ (- (sqrt (+ (sqr F) (sqr (distance centre
x))))
                    F) L))
                i))
(setq i (list (apply 'min i) (apply 'max i)))
(setq i (mapcar 'fix i))
(if (or (zerop (car i))
        (and (between (car centre) (car start) (+ (car start) len))
              (between (cadr centre) (cadr start) (+ (cadr start)
len))))
    (cons 1 (cdr i))
    i)
)
; return the radius of the nth fresnel zone for a plane wave
; and one spherical wave with focus F and wavelength L
(defun Rn (F L n)
  (sqrt (+ (* n L F) (expt (* n L 0.5) 2))))
; test only *-
(defun c:test ()
  (setq start (getpoint "Enter starting point of square : ")
        len (getdist start "Enter length of square : "))
  (square start len)
  (setq centre (getpoint "Enter starting center of circle : ")
        radius (getdist centre "Enter radius of circle : "))
  (circ_square centre radius start len)
)
; draw the arc/s of a circle centred at centre with radius
; that are contained in a square of length with
; its lower left corner located at start
(defun circ_square (centre i_rad o_rad start len /
                  bot_side top_side left_side right_side num_inters
                  bot_top left right radius width)
  (setq width (- o_rad i_rad))
  (setq radius (+ o_rad i_rad) 0.5))
; check for intersection with bottom side, etc.
(setq bot (cadr start)) (setq top (+ bot len))
(setq left (car start)) (setq right (+ left len))
(setq bot_side (circ_intersect centre radius nil bot)
      top_side (circ_intersect centre radius nil top)
      left_side (circ_intersect centre radius left nil)
      right_side (circ_intersect centre radius right nil))
(setq bot_side (less_more (inbounds bot_side left right))
      top_side (more_less (inbounds top_side left right))
      left_side (more_less (inbounds left_side top bot))
      right_side (less_more (inbounds right_side top bot)))
(setq num_inters (+ (length bot_side) (length top_side)
                    (length left_side) (length right_side)))
(setq p_lst (make_points (list bot_side right_side top_side
left_side)
                        (list bot right top left)))
; (print "main") (print p_lst) ; *-
(cond
  ((= num_inters 0)
   (if (in_rect centre radius left right top bot)
       (draw_circle centre radius width)))
  ((= num_inters 2)
   (two_inters centre width (rotate_empty p_lst)))
  ((= num_inters 4)

```

```

        (four_inters centre width (rotate_empty p_list)))
      ((= num_inters 6)
       (six_inters centre width (rotate_empty p_list)))
      ((= num_inters 8)
       (draw_arc centre width (cadar p_list) (caadr p_list))
       (draw_arc centre width (cadadr p_list) (caaddr p_list))
       (draw_arc centre width (cadr (caddr p_list)) (car (caddr
p_list))))
        (draw_arc centre width (cadr (caddr p_list)) (caar p_list))))
    )
  ;
  (defun make_points (int_list side_list)
    ;(print "make") (print int_list) (print side_list) ;*-
    (list
      (if (car int_list)
          (pt (car int_list) (car side_list)))
      (if (cadr int_list)
          (pt (cadr side_list) (cadr int_list)))
      (if (caddr int_list)
          (pt (caddr int_list) (caddr side_list)))
      (if (caddr int_list)
          (pt (caddr int_list) (caddr side_list)))
      (if (caddr int_list)
          (pt (caddr int_list) (caddr side_list)))
    )
  ;
  (defun pt (x y)
    ;(print "pt") (print (list x y)) ;*-
    (if (atom x)
        (if (= (length y) 1)
            (list x (car y))
            (list x (car y) (list x (cadr y))))
        (if (= (length x) 1)
            (append x (list y))
            (list (list (car x) y) (list (cadr x) y))))
  )
  ;
  (defun two_inters (centre width p_list)
    ;(print "two") (print p_list) ;*-
    (if (= (length p_list) 1)
        (progn
          (if (equal (cadar p_list) (caar p_list))
              (draw_circle centre (distance centre (caar p_list))
width)
              (draw_arc centre width (cadar p_list) (caar p_list))))
        (progn
          (draw_arc centre width (cadar p_list) (caar p_list)))
          (arc_180 centre width (car p_list) (cadr p_list)))
    )
  ; draw an arc less than 180 degrees - based upon a cross product
  (defun arc_180 (c width p1 p2)
    (if (minusp (- (* (- (car p1) (car c)) (- (cadr p2) (cadr c)))
                    (* (- (car p2) (car c)) (- (cadr p1) (cadr c))))
        (draw_arc c width p2 p1)
        (draw_arc c width p1 p2))
  )
  ;
  (defun four_inters (centre width p_list)
    ;(print "four") (print p_list) ;*-
    (if (= (length p_list) 2)
        (progn
          (draw_arc centre width (cadar p_list) (caadr p_list))
          (draw_arc centre width (cadadr p_list) (caar p_list)))
        (if (atom (caar p_list))
            (if (listp (caadr p_list))

```

```

      (progn
        (draw_arc centre width (car p_list) (cadr p_list))
        (draw_arc centre width (cadadr p_list) (caddr
p_list)))
      (progn
        (draw_arc centre width (cadr (caddr p_list)) (car
p_list))
        (draw_arc centre width (cadr p_list) (caddr
p_list))))
      (progn
        (draw_arc centre width (cadar p_list) (cadr p_list))
        (draw_arc centre width (caddr p_list) (car p_list))))
    )
;
(defun num_nils (p_list / i empty)
  (setq i 0) (setq empty 0) ; find number of nils in list
  (while (< i (length p_list))
    (if (nth i p_list) ()
        (setq empty (1+ empty)))
    (setq i (1+ i)))
  empty
)
;
(defun rotate_empty (p_list / i empty)
  (setq i 0) (setq empty (num_nils p_list))
  (while (< i empty)
    (if (car p_list)
        (setq p_list (append (cdr p_list) (list (car p_list)))))
    (progn
      (setq p_list (cdr p_list))
      (setq i (1+ i))))
  p_list
)
; pairs of points in ccw fashion
(defun six_inters (centre width p_list)
  (print "six") (print p_list) ; "-"
  (if (= (length p_list) 3)
      (progn
        (draw_arc centre width (cadar p_list) (caddr p_list))
        (draw_arc centre width (cadadr p_list) (cadddr p_list))
        (draw_arc centre width (cadr (caddr p_list)) (car p_list)))
      (progn
        (while (or (atom (caar p_list)) (atom (cadadr p_list)))
          (setq p_list (append (cdr p_list) (list (car p_list)))))
        (draw_arc centre width (cadar p_list) (caddr p_list))
        (draw_arc centre width (cadadr p_list) (caddr p_list))
        (draw_arc centre width (cadddr p_list) (caar p_list))))
  )
;
(defun in_rect (centre radius left right top bot)
  (if (between (car centre) left right)
      (if (between (cadr centre) top bot)
          (if (> (* radius 2) (abs (- right left)))
              ()
              T)))
  )
; return a list of values in points list that are between a and b
; where points is a list of numbers with a and b being single numbers
(defun inbounds (points a b)
  (if (null points)
      ()

```



```

      (append (between (car points) a b) (inbounds (cdr points) a
      ))
    )
    ; return inbet as a list if it is between a and b else nil
    (defun between (inbet a b)
      (if (<= (max a b) inbet)
        (if (<= (min a b) inbet)
          (list inbet)
          ())
        ())
    )

    ; return a list containing the intersection locations (if any) between
    ; a circle of radius with either a line at position x
    ; or a line at position y
    ; either x or y must be nil
    (defun circ_intersect (centre radius x y / radical ipoints)
      (if (eval x)
        (progn
          (setq radical (- (sqr radius) (sqr (- x (car centre)))))
          (if (minusp radical)
            () ;then return nil because there is no intersection
            ;else return list of intersection points
            (if (zerop radical)
              ;then there is only one intersection point
              (list (cadr centre) (cadr centre))
              (progn ;else
                (setq radical (sqrt radical))
                (list (+ (cadr centre) radical)
                      (- (cadr centre) radical))))))
          (progn
            (setq radical (- (sqr radius) (sqr (- y (cadr centre)))))
            (if (minusp radical)
              () ;then return nil because there is no intersection
              ;else return list of intersection points
              (if (zerop radical)
                ;then there is only one intersection point
                (list (car centre) (car centre))
                (progn ;else
                  (setq radical (sqrt radical))
                  (list (+ (car centre) radical)
                        (- (car centre) radical))))))
          )
        )
      )
    (defun sqr (arg) ; return the square of the argument
      (* arg arg)
    )

    ; draw a square of length starting at start and going ccw
    (defun square (start len / pt2 pt3 pt4)
      (setq start (convert_mm start)
            len (* len m_to_mm))
      (setq pt2 (polar start 0.0 len)
            pt3 (polar pt2 (/ pi 2) len)
            pt4 (polar pt3 pi len))
      (command "line" start pt2 pt3 pt4 start "")
    )

    ;
    (defun draw_circle (centre radius width)
      (setq centre (convert_mm centre)
            radius (* radius m_to_mm)
            width (* width m_to_mm))
      (if (null width)
        (command "circle" centre radius)
        (command "donut" (* (- radius (* width 0.5)) 2)

```

```

)
(* (+ radius (* width 0.5)) " centre ""))
; Draw and arc with centre cew from point pt1 to point pt2
(defun draw_arc (centre width pt1 pt2)
  (setq centre (convert_mm centre)
        width (* width m_to_mm)
        pt1 (convert_mm pt1)
        pt2 (convert_mm pt2))
  ;(print "arc") (print (list pt1 pt2)) ;"--"
  (if (equal pt1 pt2) ()
      (if (zerop width)
          (command "arc" "c" centre pt1 pt2)
          (command "pline" pt1 "w" width "" "a" "ce" centre pt2 "")))
)
;
(defun convert_mm (pt)
  (list (* 1000 (car pt)) (* 1000 (cadr pt)))
)
;
(defun more_less (l)
  (if (car l)
      (if (cadr l)
          (list (apply 'max l) (apply 'min l))
          l),
      l)
)
(defun less_more (l)
  (if (car l)
      (if (cadr l)
          (list (apply 'min l) (apply 'max l))
          l))
)

```

A.3 PS-HOE.LSP

```

; Autolisp program to generate CGH's in autocad
; as well as a postscript file.
; The calculation units are in metres while the drawing is in mm
;
; Les Chan 1990 June
;
(defun c:fz () ; temporary function to save typing
  (load "psfzp"))
(defun c:ed ()
  (command "edit" "psfzp.lsp"))
;.....
; global constants
;
; (setq m_to_mm 1000.0) ; global conversion variable
; get information about a fzp and draw it
(defun c:fzp ()
  (setvar "cmdecho" 0)
  (file_header)
  (setq f (getreal "\n Enter focal Length (mm):"))
  (setq f (/ f m_to_mm))
  (setq L (getreal "\n Enter Wavelength (nm):"))
  (setq L (* L 1e-9))
  (setq centre (getpoint "Enter Focal point (mm,mm):"))
  (setq centre (convert_m centre))
  (setq start (getpoint "Enter lower left corner of hoe (mm,mm):"))
  (setq len (getdist start "Enter length of square (mm): "))
  (setq start (convert_m start))
  (setq len (/ len m_to_mm))
  (drawHOE f L () centre start len)
  (file_footer)
  (setvar "cmdecho" 1)
)
; draw multiple fzs
(defun c:mult ()
  (setvar "cmdecho" 0)
  (file_header)
  ; (drawHOE 0.05 1.064e-6 () '(0 62.5e-6) '(-5.0e-3 0) 0.0025)
  ; (drawHOE 0.05 1.064e-6 () '(0 187.5e-6) '(2.5e-3 0) 0.0025)
  ; (drawHOE 0.05 1.064e-6 () '(0 312.5e-6) '(-2.5e-3 0) 0.0025)
  (drawHOE 0.05 532e-9 () '(4.8895e-3 3.694e-3) '(2.794e-3 2.794e-3)
0.001397)
  ; (drawHOE 0.05 1.064e-6 () '(0 562.5e-6) '(-2.5e-3 2.5e-3) 0.0025)
  ; (drawHOE 0.05 1.064e-6 () '(0 687.5e-6) '(0 2.5e-3) 0.0025)
  ; (drawHOE 0.05 1.064e-6 () '(0 812.5e-6) '(-5.0e-3 2.5e-3) 0.0025)
  ; (drawHOE 0.05 1.064e-6 () '(0 937.5e-6) '(2.5e-3 2.5e-3) 0.0025)
  (file_footer)
  (setvar "cmdecho" 1)
)
; actually do the drawing of the fresnel zones using the donut command
(defun drawfzp (F L rings scale / i)
  (setq i 1)
  (while (<= i rings)
    (setq outer (Rn F L (* 2 i)))
    (setq inner (Rn F L (1- (* 2 i))))
    ; donut expects dia. not rad. therefore multiply by 2
    (command "donut" (* outer scale 2) (* inner scale 2) centre
**)
    (setq i (1+ i)))
)
;.....
; all lengths and distances are in meters

```

```

; Parameters are:
; F - focal length
; L - lambda (wavelength)
; rings - maximum number of rings to draw; 0 : program calculated
; centre - coordinate of the centre of the zones
; start - lower left hand corner of square
; len - length of a side of the square
(defun drawHOE (F L rings centre start len 1)
  (setq i (find_ring_range centre start len F L))
  (if (null rings)
      (setq rings (cadr i)))
  (setq i (car i))
  (prin1 "drawhoe") (print rings) ; print to show program working
  (write_center centre)
  (while (<= i rings)
    (print i) ; print to show program working
    (setq outer (Rn F L (* 2 i))) ; find outer radius
    (setq inner (Rn F L (1- (* 2 i)))) ; find inner radius
    (circ_square centre inner outer start len) ; calculate and draw
    arc
    (setq i (1+ i))) ; increment counter
  )
; ****
; function to calculate the maximum and minimum zones in the square
; input : centre = centre point of zones (x y)
;         start = lower left corner of square (x y)
;         len = length of a side of the square (m)
;         F = focal length of zones (m)
;         L = design wavelength (m)
; return format : (min max)
(defun find_ring_range (centre start len F L 1 1)
  ; i = list of corner points
  (setq i (list start (list (+ (car start) len) (cadr start))
                (list (+ (car start) len) (+ (cadr start) len))
                (list (car start) (+ (cadr start) len))))
  ; j = list of distance to sides
  (setq j (sides centre i))
  ; k = list of distance to corners
  (setq k (mapcar '(lambda (x) (distance centre x)) i))
  ; combine i and j if centre could be nearer to sides than corners
  (if (not (and (null (between (car centre) (car start) (+ (car start)
len)))
                (null (between (cadr centre) (cadr start) (+ (cadr start)
len)))))
      (setq i (append i j))
  )
  ; i = list of zone numbers to corners
  (setq i (mapcar '(lambda (x)
                    (/ (- (sqrt (+ (sqr F) (sqr (- x))
F) L))
                    i))
  ; i = (min max) in real numbers
  (setq i (list (apply 'min i) (apply 'max i)))
  ; i = (min max) integers
  (setq i (mapcar 'fix i))
  ; %check for focus in square or min=0
  (if (or (zerop (car i))
          (and (between (car centre) (car start) (+ (car start) len))
               (between (cadr centre) (cadr start) (+ (cadr start)
len)))))
      (cons 1 (cdr i))
      i)

```

```

(defun sides (centre corners / temp templ
  (setq temp (if (< (cadr centre) (cadr corners))
    (pointtoline centre (car corners) (cadr corners))
    ))
  (if temp (setq temp (list temp)))
  (setq templ (if (> (car centre) (caadr corners))
    (pointtoline centre (cadr corners) (caddr corners))
    ))
  (if templ (setq templ (list templ)))
  (setq temp (append temp templ))
  (setq templ (if (< (cadr centre) (car (caddr corners)))
    (pointtoline centre (caddr corners) (caddr corners))
    ))
  (if templ (setq templ (list templ)))
  (setq temp (append temp templ))
  (setq templ (if (< (car centre) (caadr corners))
    (pointtoline centre (caddr corners) (car corners))
    ))
  (if templ (setq templ (list templ)))
  (setq temp (append temp templ))
)

(defun pointtoline (centre pt1 pt2 / a b c d)
  (setq a (- (cadr pt2) (cadr pt1)))
  (setq b (- (car pt2) (car pt1)))
  (setq c (+ (* (cadr pt1) b) (* (car pt1) a) ))
  (setq d (abs (/ (+ (* a (car centre)) (* b (cadr centre)) (- 0 c))
    (sqrt (+ (* a a) (* b b))))))
)

;.....
; return the radius of the nth fresnel zone for a plane wave
; and one spherical wave with focus F and wavelength L
(defun Fn (F L n)
  (sqrt (+ (* n L F) (expt (* n L 0.5) 2))))
;.....
; test only *-
(defun c:test ()
  (setq start (getpoint "Enter starting point of square : "))
  (len (getdist start "Enter length of square : "))
  (square start len)
  (setq centre (getpoint "Enter starting center of circle : "))
  (radius (getdist centre "Enter radius of circle : "))
  (circ_square centre radius start len)
)

;.....
; draw the arcs of a circle centred at centre with radius
; that are contained in a square of length with
; its lower left corner located at start
;
(defun circ_square (centre i_rad o_rad start len
  bot_side top_side left_side right_side num_inters
  bot_top left radius width)
  (setq width (- o_rad i_rad)) ; width of arc is equal to outer-inner
  (setq radius (+ o_rad i_rad 0.5)) ; center of line is
  (innet+outer) 2
  ; check for intersection with bottom side, etc.
  ; bot= y coord of bot, left = x coord of left side, etc.
  (setq bot (cadr start)) (setq top (+ bot len))

```

```

(setq left (car start)) (setq right (+ left len))
;find intersection between circle and lines at bot, top, etc.
(setq bot_side (circ_intersect centre radius nil bot)
      top_side (circ_intersect centre radius nil top)
      left_side (circ_intersect centre radius left nil)
      right_side (circ_intersect centre radius right nil))
;check that intersections are in the square and
;order them ccw starting at the bottom
(setq bot_side (less_more (inbounds bot_side left right))
      top_side (more_less (inbounds top_side left right))
      left_side (more_less (inbounds left_side top bot))
      right_side (less_more (inbounds right_side top bot)))
;calculate the number of intersection of the circle and square
(setq num_inters (+ (length bot_side) (length top_side)
                   (length left_side) (length right_side)))
;make list of intersection points ccw from the bottom
(setq p_list (make_points (list bot_side right_side top_side
                                left_side)
                        (list bot right top left)))
;(print "main") (print p_list) ;"--
;go to appropriate routine for the number of intersections
(cond
  ((= num_inters 0) ; case of circle completely inside or outside
   (if (in_rect centre radius left right top bot)
       (draw_circle centre radius width)))
  ((= num_inters 2)
   (two_inters centre width (rotate_empty p_list)))
  ((= num_inters 4)
   (four_inters centre width (rotate_empty p_list)))
  ((= num_inters 6)
   (six_inters centre width (rotate_empty p_list)))
  ((= num_inters 8) ;all points in p_list are used
   (draw_arc centre width (caddr p_list) (caaddr p_list))
   (draw_arc centre width (cadadr p_list) (caaddr p_list))
   (draw_arc centre width (cadr (caddr p_list)) (car (caddr
p_list))))
   (draw_arc centre width (cadr (caddr p_list)) (car p_list))))
) ;end defun circ_square
;*****
;convert list of intersection coordinates and list of side coordinates
;to a list of intersection points. The lists start at the bottom then
;right side, top, left.
;
(defun make_points (int_list side_list)
  (print "make") (print int_list) (print side_list) ;"--
  (list
    (if (car int_list) ; check that bottom has intersection
        (pt (car int_list) (car side_list)))
    (if (cadr int_list) ; check right side
        (pt (cadr side_list) (cadr int_list)))
    (if (caddr int_list) ; check top side
        (pt (caddr int_list) (caddr side_list)))
    (if (caddr int_list) ; check left side
        (pt (caddr side_list) (caddr int_list))))
  )
;*****
;convert a number and a list to a list of points.
;only the 1st two elements of the list are used
;
(defun pt (x y)
  (print "pt") (print (list x y)) ;"--
  (if (atom x) ; check if x is a number or a list

```

```

; then x is a number and y is a list
(if (= (length y) 1) ; check number of numbers in list
    (list x (car y))
    (list (list x (car y)) (list x (cadr y))))
(if (= (length x) 1) ; check number of numbers in list
    (append x (list y))
    (list (list (car x) y) (list (cadr x) y))))
)
;*****
; There are 2 intersections of the circle with the square
; Parameters: centre - centre of circle
;             width - width of arc
;             p_list - list of intersection points
;
(defun two_inters (centre width p_list)
  (print "two") (print p_list) ;*-
  (if (= (length p_list) 1)
      (progn
        (if (equal (cadar p_list) (caar p_list))
            (draw_circle centre (distance centre (caar p_list))
                          width)
            (draw_arc centre width (cadar p_list) (caar p_list))))
      (progn
        (arc_180 centre width (car p_list) (cadr p_list))))
  )
;*****
; draw an arc less than 180 degrees - based upon a cross product
; Parameters: c - centre of circle
;             width - width of arc
;             p1, p2 - end points of arc (ccw from p1)
(defun arc_180 (c width p1 p2)
  (if (minusp (- (* (- (car p1) (car c)) (- (cadr p2) (cadr c)))
                  (* (- (car p2) (car c)) (- (cadr p1) (cadr c)))))
      (draw_arc c width p2 p1)
      (draw_arc c width p1 p2))
  )
;*****
;
;
(defun four_inters (centre width p_list)
  (print "four") (print p_list) ;*-
  (if (= (length p_list) 2)
      (progn
        (draw_arc centre width (cadar p_list) (caadr p_list))
        (draw_arc centre width (cadadr p_list) (caar p_list)))
      (if (atom (caar p_list))
          (if (listp (caadr p_list))
              (progn
                (draw_arc centre width (car p_list) (caadr p_list))
                (draw_arc centre width (cadadr p_list) (caddr
p_list))))
              (progn
                (draw_arc centre width (cadr (caddr p_list)) (car
p_list))
                (draw_arc centre width (cadr p_list) (caaddr
p_list))))
          (progn
            (draw_arc centre width (cadar p_list) (cadr p_list))
            (draw_arc centre width (caddr p_list) (caar p_list))))
  )

```

```

;*****
; this function returns the number of nils in p_list
;
(defun num_nils (p_list / i empty)
  (setq i 0) (setq empty 0)      ; find number of nils in list
  (while (< i (length p_list))
    (if (nth i p_list) ()
        (setq empty (1+ empty)))
        (setq i (1+ i)))
  empty      ;Return the number of nils
)
;*****
; return list with the nulls removed
;
(defun rotate_empty (p_list / i empty)
  (setq i 0) (setq empty (num_nils p_list))
  (while (< i empty)
    (if (car p_list)
        (setq p_list (append (cdr p_list) (list (car p_list)))))
    (progn
      (setq p_list (cdr p_list))
      (setq i (1+ i))))
  p_list
)
;*****
; pairs of points in ccw fashion
(defun six_inters (centre width p_list)
  (print "six") (prin1 p_list) ;*-
  (if (= (length p_list) 3)
      (progn
        (draw_arc centre width (caddr p_list) (caadr p_list))
        (draw_arc centre width (cadadr p_list) (caaddr p_list))
        (draw_arc centre width (cadr (caddr p_list)) (caar p_list)))
      (progn
        (while (or (atom (caar p_list)) (atom (caadr p_list)))
          (setq p_list (append (cdr p_list) (list (car p_list)))))
        (draw_arc centre width (caddr p_list) (caadr p_list))
        (draw_arc centre width (cadadr p_list) (caaddr p_list))
        (draw_arc centre width (caddr p_list) (caar p_list)))
      )
  )
;*****
; determine if a circle is within a rectangle of side coordinates
; left, right, top, bot
;
(defun in_rect (centre radius left right top bot)
  (if (between (car centre) left right)
      (if (between (cadr centre) top bot)
          (if (> (* radius 2) (abs (- right left)))
              ()
              T)))
  )
;*****
; return a list of values in points list that are between a and b
; where points is a list of numbers with a and b being single numbers
(defun inbounds (points a b)
  (if (null points)
      ()
      (append (between (car points) a b) (inbounds (cdr points) a b)))
  )
;*****
; return inbet as a list if it is between a and b else nil.
(defun between (inbet a b)

```



```

    (if (>= (max a b) inbet)
        (if (<= (min a b) inbet)
            (list inbet)
            ())
        ())
)
;*****
; return a list containing the intersection locations (if any) between
; a circle of radius with either a line at position x
; or a line at position y
; either x or y must be nil
(defun circ_intersect (centre radius x y / radical ipoints)
  (if (eval x)
      (progn
        (setq radical (- (sqr radius) (sqr (- x (car centre)))))
        (if (minusp radical)
            () ;then return nil because there is no intersection
            ;else return list of intersection points
            (if (zerop radical)
                ;then there is only one intersection point
                (list (cadr centre) (cadr centre))
                (progn ;else
                  (setq radical (sqrt radical))
                  (list (+ (cadr centre) radical)
                        (- (cadr centre) radical)))))))
      (progn
        (setq radical (- (sqr radius) (sqr (- y (cadr centre)))))
        (if (minusp radical)
            () ;then return nil because there is no intersection
            ;else return list of intersection points
            (if (zerop radical)
                ;then there is only one intersection point
                (list (car centre) (car centre))
                (progn ;else
                  (setq radical (sqrt radical))
                  (list (+ (car centre) radical)
                        (- (car centre) radical)))))))
  )
;*****
(defun sqr (arg) ; return the square of the argument
  (* arg arg)
)
;*****
; draw a square of length starting at start and going ccw
(defun square (start len / pt2 pt3 pt4)
  (setq start (convert_mm start)
        len (* len m_to_mm))
  (setq pt2 (polar start 0.0 len)
        pt3 (polar pt2 (/ pi 2) len)
        pt4 (polar pt3 pi len))
  (command "line" start pt2 pt3 pt4 start ""))
;*****
; draw a circle on the screen as well as in in file
;
(defun draw_circle (centre radius width)
  (write_donut centre radius width)
  (setq centre (convert_mm centre)
        radius (* radius m_to_mm)
        width (* width m_to_mm))
  (if (null width)
      (command "circle" centre radius)

```

```

      (command "donut" (* (- radius (* width 0.5)) 2)
                  (* (+ radius (* width 0.5)) 2) centre **))
    )
;*****
; Draw and arc with centre ccw from point pt1 to point pt2
(defun draw_arc (centre width pt1 pt2)
  (write_arc centre width pt1 pt2)
  (setq centre (convert_mm centre)
        width (* width m_to_mm)
        pt1 (convert_mm pt1)
        pt2 (convert_mm pt2))
  (print "arc") (print (list pt1 pt2)) ;*-
  (if (equal pt1 pt2)
      ()
      (if (zerop width)
          (command "arc" "c" centre pt1 pt2)
          (command "pline" pt1 "w" width "" "a" "ce" centre pt2 **)))
)
;*****
; convert the x and y coordinates of a point from m to mm
;
(defun convert_mm (pt)
  (list (* 1000 (car pt)) (* 1000 (cadr pt))))
;
; converts the x and y coordinates of a point to m from mm
;
(defun convert_m (pt)
  (list (/ (car pt) m_to_mm) (/ (cadr pt) m_to_mm)))
;
;*****
; return the max and min (in that order) values of a numeric list
;
(defun more_less (l)
  (if (car l)
      (if (cadr l)
          (list (apply 'max l) (apply 'min l))
          1))
  )
(defun less_more (l)
  (if (car l)
      (if (cadr l)
          (list (apply 'min l) (apply 'max l))
          1))
  )
;
;*** procedures used for the generation of a PostScript data file ***
;
(defun write_donut (c r w / r1 r2)
  (setq w (* 0.5 w))
  (setq r1 (- (+ r w) pen_width))
  (setq r2 (+ (- r w) pen_width))
  (write_point_c) (princ (* scale r1) outf) (princ " 9 180 a " outf)
  (write_point_c) (princ (* scale r2) outf) (princ " 180 9 a f"
outf)(newline)
  (write_point_c) (princ (* scale r1) outf) (princ " 9 180 c " outf)
  (write_point_c) (princ (* scale r2) outf) (princ " 180 9 a f"
outf)(newline)
)
(defun write_arc (c w p1 p2 / r r1 r2 a1 a2)
  (setq r (distance c p1))
  (setq w (* 0.5 w))
  (setq r1 (- (+ r w) pen_width)) (setq r2 (+ (- r w) pen_width))
  (setq a1 (angle c p1)) (setq a2 (angle c p2))

```

```

    (setq a1 (angtos a1 0 4)) (setq a2 (angtos a2 0 4))
    (write_point_c) (princ (* scale r1) outf) (space) (princ a1 outf)
  (space)
    (princ a2 outf) (princ " a " outf)
    (write_point_c) (princ (* scale r2) outf) (space) (princ a2 outf)
  (space)
    (princ a1 outf) (princ " c f" outf) (newline)
)
(defun space ()
  (princ " " outf)
)
(defun write_point (p)
  (princ (* scale (car p)) outf) (space) (princ (* scale (cadr p))
outf) (space)
)
(defun write_point_c ()
  (princ "x y " outf)
)
(defun write_centre (p)
  (princ "/x " outf)
  (princ (* scale (car p)) outf) (princ " def" outf) (newline)
  (princ "/y " outf) (princ (* scale (cadr p)) outf)
  (princ " def" outf) (newline)
)
(defun file_header ()
  (setq filename (getstring "Enter PostScript output file name : "))
  (setq pixelsize (getreal "Enter the resolution of the printer in
pixels per inch: "))
  (setq pixelsize (/ 0.0254 pixelsize)) ; calculate pixel diameter in
m
  (setq pixelsize (/ pixelsize 2)) ; convert to radius
  (setq mac (getstring "Mac wants <cr> only: mac format [y or n] ? "))
  (if (= mac "Y") (setq mac "y"))
  (if (= mac "y")
    (defun newline () (princ (chr 13) outf))
    (defun newline () (princ (chr 10) outf))
  )
  (setq magnification (getreal "Enter the magnification factor (eg.
10) :"))
  ; there are 1e6 microns in 1 metre
  (setq scale (* 1e6 magnification))
  ; PostScript devices use measures in 1/72 of an inch
  ; there are 25.4e3 microns in one inch
  (setq ps_unit (/ 72 25.4e3)) ; this is the scale factorto convert PS
units
  ; for a linotype 300 printer, pixel size = 10 um, radius = 5 um
  (setq pen_width (/ pixelsize magnification))
  (setq outf (open filename "w"))
  (princ "!PS-Adobe-2.0" outf) (newline)
  (princ "/n (newpath) def" outf) (newline)
  (princ "/f (fill) def" outf) (newline)
  (princ "/a (arc) def" outf) (newline)
  (princ "/c (arcn) def" outf) (newline)
  (princ "300 400 translate" outf) (newline)
  (princ ps_unit outf) (space) (princ ps_unit outf) (princ " scale"
outf) (newline)
  (princ "% new units are in microns" outf) (newline)
  (princ "n" outf) (newline)
  ; print values in fixed format
  (setq lunitslsp (getvar "lunits"))
  (setq lupreclsp (getvar "luprec"))
  (setvar "lunits" 2) ; decimal mode

```

```
(setvar "luprec" 1) ; 1 digit after the decimal point
)
(defun file_footer ()
  (princ "showpage" outf) (newline)
  (princ (chr 4) outf) ; finish file with EOT character
  (close outf)
  ;restore current units
  (setvar "lunits" lunitslsp)
  (setvar "luprec" lupreclsp)
)
```

A.4 PRISM.LSP

```

; *this file draws elliptical fzps in autocad assuming the database
; units are in mm
;
; Les Chan 1990 June
;
; Global Constants
(setq m_to_mm 1000.0) ; constant for converting metres to mm
(setq root2 (sqrt 2)) ; square root of 2
(setq sina (/ 1 root2)) ; sin(alpha)=sin(45)

(defun c:lf () ; temporary function to save typing
  (load "prism"))
(defun c:ed ()
  (command "edit" "prism.lsp"))
(defun sqr (arg) ; return the square of the argument
  (* arg arg)
)

; return inbet as a list if it is between a and b else nil
(defun between (inbet a b)
  (if (>= (max a b) inbet)
    (if (<= (min a b) inbet)
      (list inbet)
      ())
    ())
)

; draw a square of length starting at start and going ccw
(defun square (start len / pt2 pt3 pt4)
  (setq start (convert_mm start)
        len (* len m_to_mm))
  (setq pt2 (polar start 0.0 len)
        pt3 (polar pt2 (/ pi 2) len)
        pt4 (polar pt3 pi len))
  (command "line" start pt2 pt3 pt4 start "")
)

;
(defun draw_circle (centre radius width)
  (setq centre (convert_mm centre)
        radius (* radius m_to_mm)
        width (* width m_to_mm))
  (if (null width)
    (command "circle" centre radius)
    (command "donut" (* (- radius (* width 0.5)) 2)
                  (* (+ radius (* width 0.5)) 2) centre ""))
)

; Draw an arc with centre ccw from point pt1 to point pt2
(defun draw_arc (centre width pt1 pt2)
  (setq centre (convert_mm centre)
        width (* width m_to_mm)
        pt1 (convert_mm pt1)
        pt2 (convert_mm pt2))
  (print "arc") (print (list pt1 pt2)) ;*-
  (if (equal pt1 pt2) ()
    (if (zerop width)
      (command "arc" "c" centre pt1 pt2)
      (command "pline" pt1 "w" width "" "a" "ce" centre pt2 "")))
)

; *** convert the x and y coordinates of a point from m to mm ***
;

```

```

(defun convert_mm (pt)
  (list (* m_to_mm (car pt)) (* m_to_mm (cadr pt))))
)
;*** return first 2 elements of list in descending order ***
;
(defun more_less (l)
  (if (car l) ; check first element is not nil
      (if (cadr l) ; check second element is not nil
          (list (apply 'max l) (apply 'min l))
          1))
      1)
)
;*** return first 2 elements of list in ascending order ***
;
(defun less_more (l)
  (if (car l)
      (if (cadr l)
          (list (apply 'min l) (apply 'max l))
          1))
      1)
)

; get information about a fzp and draw it
(defun c:ptest ()
  (setvar "cmdecho" 0)
  (initellipse)
  (setq f (getreal "\n Enter distance (m):"))
  (setq f (/ f root2))
  (setq L (getreal "\n Enter Wavelength (m):"))
  ; (setq rings (getint "\n Enter number of Rings :"))
  (setq centre (getpoint "Enter Focal point (m,m):"))
  (setq start (getpoint "Enter lower left corner of box (m,m):"))
  len (getdist start "Enter length of square (m): ")
  (square start len)
  (drawELLIP f L () centre start len)
  (setvar "cmdecho" 1)
)
;*** return the radius of the minor axis ***
; uses global constant root2
;
(defun Rminor (y L n)
  (sqrt (- (* (sqrt (* n L)) 0.5) (* n L y root2))))
;***
;Return the zone number given point p with y0 and centre c
; uses global constant twool
; uses global constant sina
(defun zone (p y0 c / y x)
  (setq y (- (cadr p) (cadr c)))
  (setq x (- (car p) (car c)))
  (* (+ (sqrt (+ (sqr x) (sqr (- y y0)) (* y0 y0))) (* y0 sina))
  twool)
)
;
;
(defun c:tt ()
  (setvar "cmdecho" 0) ; turn off command echo
  (initellipse)
  ;make sure pline with is 0
  (command "pline" '(0 0) "w" 0 0 '(1 1) "" "erase" "l" "")
  (drawELLIP (/ 0.02 root2) 532e-9 () '(0 0) '(-0.001 -0.001) 0.002)
  ; (drawELLIP (/ 0.02 root2) 532e-9 () '(0.0025 0) '(0.0015 -0.001)
  0.002)
  ; (drawELLIP (/ 0.02 root2) 532e-9 () '(0.005 0) '(0.004 -0.001)
  0.002)
  ; (drawELLIP (/ 0.02 root2) 532e-9 () '(0.0075 0) '(0.0065 -0.001)

```

```

; 1992)
  (setvar "cmdecho" 1)          ; restore command echo
;
; sets Global variable twoool
(defun drawELLIP (y0 L auto centre start len / i)
  (setq twoool (/ 2 L))        ; 2/Lambda global variable
  (setq i (find_ellip_range centre start len y0 L))
  (print i)
  (if (null auto)
    (setq auto (cadr i)))
  (setq i (car i))
  (print i)
  (princ "drawELLIP") (print auto)
  (while (<= i auto)
    (setq inner (Rminor y0 L i))
    (setq outer (Rminor y0 L (1+ i)))
    (print i); (princ centre) (princ outer) (princ inner) (princ "while")
    (ellip (ec y0 L i centre) inner outer)
    (setq i (+ i 2)))
  )
;
(defun ec (y L n c)
  (list (car c) (+ (- (* 2 y) (/ (* n L) root2)) (cadr c)))
)
;
(defun find_ellip_range (centre start len F L / i)
  ; i = list of corner points
  (setq i (list start (list (+ (car start) len) (cadr start))
    (list (+ (car start) len) (+ (cadr start) len))
    (list (car start) (+ (cadr start) len))))
  ; i =
  (setq i (mapcar '(lambda (x) (zone x F centre)) i))
  (setq i (list (1+ (fix (* root2 F twoool))) (apply 'max i)))
  (print i) (princ "find")
  (setq i (mapcar 'fix i))
  )
;
(defun ellip(c in out / minor major width)
  (setq c (convert_mm c)
    in (* in m_to_mm)
    out (* out m_to_mm)
  )
  (drawellipse c in out)
)
;
; *** Create a list of points for 1/4 of an ellipse (with eccentricity of
; 2)
; in ascending order (up) and in descending order (down) ***
;
(defun initellipse (/ r2 heights i y inc)
  (setq r2 (sqrt 2))
  (setq heights (list 0))
  (setq inc 2)          ; size of approximation step in degrees
  (setq i inc)
  (while (<= i (- 90 inc))
    (setq y (* (sin (* i (/ pi 180))) r2))
    (setq heights
      (append heights
        (list (* (sin (* i (/ pi 180))) r2)))
      )
    (setq i (+ i inc))    ; increment counter i
  )
  (setq up (mapcar '(lambda (y)

```

```

                                (cons (sqrt (- 1 (* y y 0.5))) (list y))
                                )
                                heights)
                                )
                                (setq up (append up (list (cons 0 (list 12)))))
                                (setq heights nil)
                                (setq i (- 90 inc))
                                (while (>= i 0)
                                  (setq heights (append heights (list (* (sin (* 1 ( pi 180)))
r2))))
                                  (setq i (- i inc))
                                  )
                                  (setq down (mapcar '(lambda (y)
                                                         (cons (sqrt (- 1 (* y y 0.5))) (list y))
                                                         )
                                                         heights)
                                  )
                                  (setq down (append (list (cons 0 (list r2))) down))
                                )
                                (defun drawellipse (center scale1 scale2)
                                  (command "ucs" "o" center)
                                  (command "pline")
                                  (mapcar '(lambda (x) (command (scalepoint1 x scale1))) up)
                                  (mapcar '(lambda (x) (command (scalepoint1 x scale2))) down)
                                  (command "C")
                                  (command "pline")
                                  (mapcar '(lambda (x) (command (scalepoint2 x scale1))) up)
                                  (mapcar '(lambda (x) (command (scalepoint2 x scale2))) down)
                                  (command "C")
                                  (command "pline")
                                  (mapcar '(lambda (x) (command (scalepoint3 x scale1))) up)
                                  (mapcar '(lambda (x) (command (scalepoint3 x scale2))) down)
                                  (command "C")
                                  (command "pline")
                                  (mapcar '(lambda (x) (command (scalepoint4 x scale1))) up)
                                  (mapcar '(lambda (x) (command (scalepoint4 x scale2))) down)
                                  (command "C")
                                  (command "ucs" "w")
                                )
                                (defun scalepoint1 (p sc)
                                  (list (* sc (car p)) (* sc (cadr p)))
                                )
                                (defun scalepoint2 (p sc / t)
                                  (setq t (- 0 sc))
                                  (list (* t (car p)) (* sc (cadr p)))
                                )
                                (defun scalepoint3 (p sc / t)
                                  (setq t (- 0 sc))
                                  (list (* t (car p)) (* t (cadr p)))
                                )
                                (defun scalepoint4 (p sc / t)
                                  (setq t (- 0 sc))
                                  (list (* sc (car p)) (* t (cadr p)))
                                )

```


END

03|03|9|2

FIN