The Design and Implementation of a Structured Programming Language for the Description of Optical Architectures

by

Adel Othman Mohammed Lafi

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

January, 1992

Order Number 1354051

The design and implementation of a structured programming language for the description of optical architectures

Lafi, Adel Othman Mohammed, M.S.

King Fahd University of Petroleum and Minerals (Saudi Arabia), 1992

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

THE DESIGN AND IMPLEMENTATION OF A STRUCTURED PROGRAMMING LANGUAGE FOR THE DESCRIPTION OF OPTICAL ARCHITECTURES

BY

ADEL OTHMAN MOHAMMED LAFI

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

JANUARY 1992

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

This thesis, written by **ADEL OTHMAN MOHAMMED LAFI** under the direction of his thesis committee, and approved by all the members, has been presented to and accepted by the Dean, College of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Thesis committee

Chairman (*Dr. SUBARAO GHANTA*)

Member (*Dr. MOHSIN GUIZANI*)

Member (*Dr. BASSEL ARAFEH*)

Member (*Dr. SULAIMAN AL-BASSAM*)

Dr. MOHAMMED AL-TAYYEB
Department Chairman

Dr. ALA H. AL-RABEII
Dean College of Graduate Studies

Date: 19.1.92


TO MY PARENTS

ACKNOWLEDGMENT

Acknowledgment is due to the King Fahd University of Petroleum and Minerals for providing the opportunity to carry out this research work.

I would like to express my appreciation to Dr. Subbarao Ghanta, the chairman of my thesis committee, whose valuable directions and helpful guidance made this work possible. I sincerely thank the other members of the committee, Dr. Mohsen Guizani, Dr. Bassel Arafeh, and Dr. Sulaiman Al-Bassam for their valuable comments and directions.

I would like, also, to express my gratitude for Dr. Yilmaz Akyildiz for giving me access to his wonderful NEXT station.

I thank Dr. Fida Al-Adel and Dr. Ihsan Khawaja from the Energy Research Laboratory at KFUPM for giving me access to valuable information for my work. Special thanks are due to Mr. Joseph Mastromorino for sharing his valuable knowledge with me.

I would like, at the end, to express my deepest appreciation for Mr. Husni Al-Muhtasib, Mr. Yahya Garout, Mr. Majed Al-Ashram, Mr. Yahya Zaidan, Mr. Mahmood Hussain, Mr. Jaweed Yazdani, and Mr. Ayman Nazzal for the time they spent to help me get this work done. I also thank Mr. Salah Adam and Mr. Abed Abdul_Azzeem for their help.

Finally, I thank Mr. Hakeem Nazzal, Mr. Khaleel Khaleel, Mr. Ahmad Awad, Mr. Mohammed AL-Daous, Mr. Ziad Ma, Mr. Azzam Ibraheem, and Mr. Ahmad Ghazal for their continuous support and encouregment.

THESIS ABSTRACT

NAME OF STUDENT : ADEL OTHMAN MOHAMMED LAFI

TITLE OF STUDY : THE DESIGN AND IMPLEMENTATION OF A STRUCTURED PROGRAMMING LANGUAGE FOR THE DESCRIPTION OF OPTICAL ARCHITECTURES

MAJOR FIELD : COMPUTER SCIENCE

DATE OF DEGREE : JANUARY 1992

Significant amount of time and effort is spent in the process of debugging optical architectures (mainly aligning/adjusting, removing, or/and adding of components within the architecture). The debugging is needed to detect the sources of errors in the results of the debugged architecture, under study/development.

We design a system to simulate the behaviour of any optical architecture provided the architecture is described using **PLOADS**: a Programming Language for Optical Architectures Description / Specification. Therefore, the debugging could be performed on the simulated architecture which is much easier, time and cost effective. This system is a basis for *Design Automation* tools for **OCAD** (Optical Computer Aided Design).

To simulate the behaviour of any optical architecture, the simulation of the behaviour of the individual components is necessary. For that we provide models for the heavily used optical components. These models are for computer representation. We then provide a procedure for each component. The procedure uses the model of the component to simulate its behaviour as part of any architecture. The procedures are part of a *components library* designed for the sake of simulating optical architectures.

We design **PLOADS**, a language for optical architectures description. Using **PLOADS**, the designer (user) can describe any optical architecture. The system, which we develope is based on the *language* and the *components library*, simulates the architecture and provides the information required by the user about it.

MASTER OF SCIENCE DEGREE

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

DHAHRAN, SAUDI ARABIA

JANUARY 1992

خلاصة الرسالة

اسـم الطالـب : عادل عثمان محمد لافي
عنوان الدراسة: تصميم و تنفيذ لغة برمجة هيكلية لوصف الأبنية الضوئية
التخصـــص : علوم الحاسب الآلي
تاريخ الدرجـة: يناير (كانون الثاني) ١٩٩٢

يصرف الكثير من الوقت و الجهد في عمليـة إزالة الشوائـب من التصاميم الأولية للأبنية الضوئية (تشمل هذه العملية تصفيف، ضبط، تعديل، إزالة، أو/و إضافة وحدات ضوئية للبناء). عملية إزالة الشوائب ضرورية لتحديد مصادر الأخطاء في نتائج البناءالضوئي قيد البحث و التطوير.

من خلال هذه الدراسة، نقوم بتصميم نظام متكامل يقوم بمحاكاة أي بناء ضوئي استخدمت لوصفه ل. ب. و. أ. ض. : لغة برمجة لوصف الأبنية الضوئية. إنتاج البناء المحاكا يمكن المصمم من إزالة الشوائب من النسخة المحاكا مما يوفر الوقت و الجهد و يوفر أيضا من تكلفة هذه العملية. هذا النظام يشكل قاعدة لأدوات التصميم الأوتوماتيكى للتصاميم المساعدة بالحاسب الآلي.

لمحاكاة التصرف العام لأي بناء ضوئي، محاكاة تصرف الوحدات المكونة للبناء يكون ضروريا. لهذا فنحن نقدم نماذج للوحدات كثيفة الإستخدام مصممة خصيصا للتقديم من خلال أجهزة الحاسب الآلي. بناء على هذه النماذج، نقوم بتقديم إجراءلكل وحدة يستخذم نموذجها لمحاكاة تصرفها. تشكل هذه الإجراءات مجتمعة ما أطلقنا عليه إسم مكتبة الوحدات.

بعد ذلك نقوم بتصميم ل. ب. و. أ. ض.، لغة لوصف الأبنية الضوئية. بإستخدام ل. ب. و. أ. ض.،يستطيع المصمم (المستخدم) وصف أي بناءضوئي. من خلال هذا الوصف، يقوم النظام المطور على أساس اللغة و مكتبة الوحدات بمحاكاة البناء و توفير المعلومات المطلوبة من المستخدم بخصوصه.

درجة الماجستير في العلوم
جامعة الملك فهد للبترول والمعادن
الظهران - المملكة العربية السعودية
كانون الثاني (يناير) ١٩٩٢م

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

In this chapter, we introduce the concept of Optical Computing in terms of its evolution, achievments, drawbacks and basic trends in its research. The objectives and motivation of this study are then discussed, in addition to exploring the problems it is expected to overcome. Related previous work is reviewed.

1.1 Optical Computing

Optical Computing has evoked significant interest during the last decade. This interest was in part due to the inherent advantages possessed by linear and non-linear optical components [WHER87].

Optical Computing was first discussed seriously in the 1960's, right after the invention of laser. However, it had to wait till 1983 when the first universal logic gate was demonstrated using the non-linear characteristics of optical devices [WHER87]. Still it took a back seat with respect to the electronics primarily due to the technological problems for miniaturization [FEIT88].

One main reason for the domination of electronics over optics, so far, was the

success of conventional Von Neuman computers that:

- satisfied the world's computational needs.
- were flexible and easy to program.
- gave results that were accurate to any desired degree.
- were evolving tremendously in terms of speed and efficiency, due to advances in VLSI.

This reduced emphasis of optical computers lasted till the late 1980's when the merits of optical systems were put to use [FEIT88].

1. In direct image processing, images are actually thought of in terms of optical signals and light beams. Image processing can be carried out by specific optical systems directly on the image with no need for sampling, quantization and the $\Omega(N^2)$ conventional algorithms that process it. These optical systems operate faster than their electronic counterparts.

2. The movement towards parallel computing had special requirements that VLSI technology could not fully support. These requirements place a burden on the communications lines and interconnections between nodes of parallel machines. The *Von Neuman bottleneck* represents a fundamental problem in the architectures of serial and parallel computers which have a common memory. This occurs when information is constantly moving to and from memory as it is needed by the processor(s) [NEFF87]. Such a problem is easily overcome by optical systems, because of the inherent parallelism that characterizes them.

3. In optical communication, millions of data channels may operate in parallel, each with a bandwidth several orders of magnitude greater than that of any electronic link.

4. Optical devices have proven to be faster than their conventional digital electronic counterparts.

5. The electrons circulating in a conventional computer are susceptible to electromagnetic interferences and field-disturbances (EMI) [WHIT85], and are likely to fail operating in radiation zones (such as nuclear reactors and nuclear battlefields). Optical computers, on the other hand, would continue to function undisturbed.

1.1.1 Trends of ongoing research on Optical Computing

Optical systems fall into either of the following two classes.

1. Special purpose analog systems for
   - image processing
   - signal processing.
2. General purpose digital optical computing systems.

A good portion of ongoing research is towards mimicking existing electronic computing elements (logic gates, memories, switches) using optical components. However, emphasis recently is being given to the special features of optical systems/devices. These devices allow the realization of functionalities which are very difficult to obtain with digital electronics, and/or computationally intractable in

the digital domain. In other words, trends are emerging with shifted focus from *technology-driven* to *requirements-driven* methodologies.

Side by side, another ongoing research has been done by physicists, who are trying to develop, enhance, and invent optical components to be used in optical architectures. That trend is extremely useful, in the sense that many of these new optical componenets are found to suit the computational part of optical systems. Suitability here is in terms of space, and potential for miniaturization. These components have ensured minimal power losses and hardly noticed beam deviation within the experiments they constructed.

Having that in mind, a more detailed classification of previous research on optical computing has been done. The following nine major fields of research seem to classify the activities in optical computing field:

1. Image processing, which involves both pattern recognition and character recognition ([CAI 90], [FERR90], [JAHN90a]).

2. Optical implementations of neural networks and expert systems ([OITA90], [BIAN90], [YEE90]).

3. Optical implementation of storage devices, random access or associative memories ([LALA87], [GIND88], [LIN 89]).

4. Optical mimicking of logical/electronic components, such as logical gates and flip-flops ([HARA90], [ISLA90], [FATE84]).

5. Optical implementation of computational systems, such as subtraction/ addition units, look-up tables, numerical base conversion systems, residue arithmetic systems ([MUKH90a], [MIRS90], [DATT89]).

6. Optical implementation of different matrix operations which are given a considerable attention ([HONG90], [MURT88], [DEJA87]).

7. Optical implementation of linear algebraic problem solving systems ([JOHN90a], [ANJA87]).

8. Optical implementation of different interconnection algorithms, such as the perfect shuffle, banyan network, crossover network ([JAHN90b], [JAHN90c], [KARL88]).

9. A fair amount of work related to numerical computing and optical transform actions in general ([WHER87], [NEFF87], [FLAV90], [PERI87], [ZHAN90], [JOHN90b], [SENI90]).

1.1.2 General problems with Optical Computing

The major problem in optical systems is, that they tend to be large, crude, and sensitive to movement in any direction. The following represents the main sources of the problems [FEIT88]:

- Large physical space is required to provide for the bulky optical components.
- Large systems are hard to produce in terms of feasibility in time and cost, and in terms of convenience at the time of installation.
- Large systems are sensitive to thermal and vibrational influences from the environment. Which makes them very unpredictable.
- Alignment and adjustment problems which arise at the time of setting and operating the optical system. These problems are major factors that contribute to increasing the cost of designing an optical system.

The first three problems were partially solved by the integrated optics technology, very similar to the revolution of electronic integrated circuit [FEIT88]. At the same time, physicists have been working on overcoming the problem of sensitivity of optical components to thermal noise and other influencing parameters, which might cause real damage to the results expected from any optical system. The last problem, on the other hand, is the one that needs the attention and contribution of disciplines from computer science, such as *OPCAD* (Optical Computer Aided Design), a new terminology which might very much be the first step on the way to ***automating*** the process of designing and installing optical setups/systems/devices/architectures. This issue is the core of our work, to which we will come back later.

Other kinds of problems arise when comparing optical computers with digital electronic computers, such as:

1. In optical computers, decisions are hard to realize. This could be easily figured out when it is realized that signals in optical computers are nothing but light beams, which cannot disappear all of a sudden by a decision making block of the system. Therefore, in order to effectively use optical computers/computing, multi-level decision making has to be flattened as much as possible.

2. The operations of optical systems are basically analog by nature, and therefore are inaccurate. Henceforth, attention must be paid to scaling and normalization issues when analog computations are carried out.

1.2 Motivation of the study

It has been found that one of the most time and effort consuming phases of optical system design is the phase of setting up the experiment. The experiment here is the optical system in the phase of testing. The goal of experiments is to establish the feasibility and to prove the correctness of the design as a whole. However, in practice this is not a simple task, and the difficulties arise out of several sources. These are:

1. *The manufacturing (practical) limitations of the different optical components*: Such limitations force designers to use whichever component are available, regardless of the efficiency problems this might cause to the whole optical system.

2. *The deviation defects of the light beam used as input to the experiment which might be caused by different factors such as thermal noise*: This makes it necessary for the experimenter to keep changing the positions (alignment) of the components of the experiment according to the direction of the incident beam. A process that is appearently annoying.

3. *Technical problems due to failures of any component that is part of the experiment*: Locating a bad/malfunctioning component is a time consuming task.

4. *The serious problem of alignment/adjustment of the components constructing the experimented design*: This takes an enormous amount of time and effort spent on experimenting any design. Simply, it is very difficult to be sure that the current alignment/adjustment of components will give the desired

results. Instead, what all is known is that using these components at the specified positions gives an expected result very close to the desired one. This expected result has to be enhanced by continuous alignment/adjustment till it reaches the desired level of correctness and accuracy. We call this process as *debugging/tuning* the experimental setup.

5. *The conceptual problems that might be caused when testing a non-theoretically proven design*: Usually, the intention of the experimenter is to prove a hypothesis. In these cases, optical components need to be heavily altered rather than just aligned.

The aim of this study is to ease up the *debugging* process. This could be achieved by developing a Software tool that will:

- Save significant time and effort needed in debugging the design.
- Mature the process of realizing the experimental setups, faster.
- Encourage researchers to go for large experimental architectures.
- Provide of an environment for researchers and experimenters to tune up the parameters of their experiments before even going into actual installation issues.
- Give researchers the ability to customize any needed optical component. This is just like the researcher being able to imagine the existence of any optical component with any functional specifications and then use it within his experiment description.
- provide a basis for optical computer aided design (OPCAD) for optical experiments.

The method used to achieve these objectives is through *simulation*.

In this thesis, we have designed a structured programming language, with syntax and semantics selected to suit the description of optical experiments/systems/devices. The main purpose of the language is to describe any optical setup. A tool that will enable any experimenter to tune any parameter within the experiment and check the performance within a short time. To accomplish this, the experimenter has to use the designed language to write a program that describes the experiment. We made sure that all necessary constructs, and structures were available within the language.

The language and hence the system is so flexible that the user can change or add any parameter within his describing program. This could be done in a straight forward manner. What all the user has to do is to change partially or fully, as required, the components or values within the describing program. The selection of syntax and semantics of the language is done after going through the following two steps:

1. Studying a fair number of existing and verified optical architectures. This enabled us to extract all necessary components and constructs for our language.

2. Simulating all the known optical components by describing their functions and characteristics. This information is saved in a library, referred to as the *component library*. The component library is designed to be flexible and extendable to hold information about any new optical components that are continuously being improved and discovered. The information in this library will be used by the system (language) to simulate the functionality of the components used in the optical setup.

The language is designed to be rich in its descriptive power and to allow compact and modular description of optical systems. This language could then be used to *debug* any of the setups by changing the descriptions of the architectures. It can also be used to *probe* for any piece of information necessary for the user (experimenter) at any time/position within the experiment. The language allows translation of any description/specification of any optical experimental setup into a low-level description. Actual three-dimensional positioning of components, and their exact functional specifications are major information supplied by the user of the language. Using this information, allows the system to detect the actual positions and orientations of optical components within the described optical system. It also allows the system to trace the light beam throughout any set-up. Tracing in the sense that its characteristics could be detected whenever it is needed. This gives the basis to automate the process of design of optical systems. Furthermore, intelligent manufacturing systems could be developed to make use of this information in order to automate the process of building the described optical system.

In the current practice, optical experiments are conducted by the following basic steps:

1. A diagram is initially produced. It represents what the experimenter believes out of the results of the designed system.

2. Placing the optical components from the diagram in positions that are as accurate as possible.

3. A series of alignments and adjustments of components are performed on the components. These are necessary before ensuring the correctness and accuracy of the system. The time and effort put into this process is significant

and unpredictable. However, it dramatically increases with the size/scale of the experiment and the diversity of component characteristics.

1.3 Outline of the Thesis

In Chapter 2, we introduce the reader to the necessary background with respect to optics and optical components. Chapter 3 discusses the modeling of selected optical components in terms of mathematical models will be discussed. Chapter 4 presents the design of the component library, which will contain information about available components. The functionality of these components will be described in *pseudo* code. In Chapter 5, we design the language and give the formal language specification, together with illustrative examples. Chapter 6 describes selected experiments using the designed language. Chapter 7 covers some implementation aspects of the interpreter. In Chapter 8, some experimental results produced by the system in it's current form is given. Finally, Chapter 9 concludes the thesis. It also discusses the future work.

Chapter 2

OPTICS REVIEW

This chapter is intended to provide a quick review of the required basics about optics. We will discuss general issues such as the propagation of light beams, ray optics, wave theory, electromagnetic representation of rays, and different related principles such as diffraction, interfernce, coherence, and others. A brief description of the functionalities of a number of selected optical components will be given as well.For further details, the reader is refered to ([FEIT88], [HECH87], [NEWP90], [CASA77], [ORIE85], [MELL85], [LIA81], [SIEG71],[HEAV91], [GIAN84]).

In what follows, the terms rays and light beams will mean the same. Rays are light beams in both visible and invisible spectra. Light beams, on the other hand, are rays in the visible spectrum only. In optical computing, however, we only use the visible spectrum.

2.1 Light Beams

In this section, we present the reader two methodoligies to formally describe any light beam.

2.1.1 Light as an Electromagnetic Wave

This description is used for the mathemetical computations involving different phases of analysis of a light beam propagating into any optical system.

Light is a transverse electromagnetic wave. The electric (E) and magnetic (H) fields are perpendicular to each other and to the propagation vector. The thumb rule could be used to detect the directions of the three vectors. Therefore, any light beam could be described as a wave propagating in space.

Let us now consider the mathematics of wave motion: precisely, the mathematical representation of the harmonic wave. To represent any harmonic wave that is moving in space along a certain line, for simplicity, assume it was the $x - axis$, with a velocity v, we use the following equation:

$$\Psi(x,t) = A\ sin\ k(x \pm vt) \tag{2.1}$$

$-v$, means that the wave is moving in the direction of positive x. k, is a positive constant known as the **propagation number.** The maximum disturbance of the wave is known as the amplitude of the wave and referenced by A. This wave is periodic in both space and time. The **spatial period** is known as the **wavelength** and denoted by λ . Any increase or decrease in x by the amount λ or 2π should leave Ψ unaltered. Therefore,

$$sin\ k(x - vt) = sin\ k((x \pm \lambda) - vt) = sin(k(x - vt) \pm 2\pi) \tag{2.2}$$

and so,

$$k\lambda = 2\pi \tag{2.3}$$

and,

$$k = 2\pi/\lambda \tag{2.4}$$

Another important quantity related to harmonic waves is the **temporal period**, τ. This is the amount of time it takes for one complete wave to pass a stationary observer. It is the number of units of time per wave, the inverse of **frequency** ν

$$\tau = \frac{\lambda}{v} \tag{2.5}$$

$$\nu = \frac{1}{\tau} = \frac{v}{\lambda} \tag{2.6}$$

There are two other quantities often used in the literature of wave motion, these are **angular frequency** ω and **wave number** χ ,

$$\omega = \frac{2\pi}{\tau} \tag{2.7}$$

$$\chi = \frac{1}{\lambda} \tag{2.8}$$

The phase φ of the wave, which takes values from 0 to 2π, detects where the wave is, in units of its wave period. Hence, if one takes a cross-section of the propagating wave, he should see the wave at a position within its period which goes from 0 to 2π, again.

The whole argument of the sine function of Equation 2.1 could be replaced by the phase φ , a function of x and t:

$$\varphi = (kx - \omega t) \tag{2.9}$$

Let us now introduce a new term ε called the initial phase angle. It is the constant contribution to the phase arising at the generator and is independent of how far in space, or how long in time, the wave has travelled. Now, the general equation to represent harmonic waves becomes:

$$\Psi(x,t) = A\ sin\ (kx \pm \omega t + \varepsilon) \tag{2.10}$$

Having in mind that the light beam is an electromagnetic wave propagating in space, one can now represent the light beam mathematically as a harmonic wave with amplitude E_0, which is the magnitude of the electric field, and mathematical formula:

$$E(x,t) = E_0 \; sin \; (\omega t - (kx + \varepsilon)) \tag{2.11}$$

Having had an idea about the mathematical representation of light beams, we can now discuss the parameters and the quantities that affect propagation and distinguish light beams from each other.

The speed of light in vacuum is $c = 3.0 \times 10^8$ (km/s). In other media, the light velocity is affected by the **index of refraction** of the medium.The index of refraction is:

$$n = \frac{c}{v} \tag{2.12}$$

where v is the velocity of the propagating light beam in the medium with refractive index n. Therefore, v is one important parameter of any light beam. Other previously mentioned parameters are:

k propagation number or wave vector [radians/m].

ν frequency [Hertz].

ω angular frequency [radians/sec].

λ wavelength [m].

τ temporal period [$Hertz^{-1}$].

χ wave number [m^{-1}].

The dependence of velocity on the refractive index is a result of the dependence of wavelength on the index. The relation

$$v = \nu\lambda \tag{2.13}$$

explains that.

λ_0, like c, is the wavelength of the light beam in vacuum.

Light **intensity I**, defined as, the average energy per unit time crossing a unit area, or as, the radiant flux density, sometimes called **irradiance**, is another very important parameter computed using formulae that are analogous to **Ohm's Laws.**

$$\begin{aligned} I &= \frac{EH}{2} = \frac{E^2}{2\eta} \\ &= \frac{\eta H^2}{2} && (2.14) \\ E &= \eta H = \sqrt[2]{2\eta I} && (2.15) \\ H &= \frac{E}{\eta} = \sqrt[2]{\frac{2I}{\eta}} && (2.16) \\ \eta &= \frac{E}{H} = \frac{\eta_0}{n} && (2.17) \\ \eta_0 &= 377 ohms && (2.18) \end{aligned}$$

where,

I intensity $[watts/m^2]$.

E magnitude of the electric field, and the amplitude in Equation 2.11 [volts/m].

H magnitude of the magnetic field [amperes/m].

η_0 the wave impedance of vacuum [ohms].

η the wave impedance of a medium of refractive index n [ohms].

The intensity could also be computed using a different set of formulae:

$$I = cvE^2 = \frac{c}{\mu}H^2 \tag{2.19}$$

$$\epsilon = K_e c_0 \tag{2.20}$$

$$\mu = K_m \mu_0 \tag{2.21}$$

$$\mu_0 = 4\pi \times 10^{-7} \tag{2.22}$$

where,

K_m relative permeability.

K_e relative permittivity.

ϵ permitivity $[s^2C^2/m^3kg]$

μ permeability $[mkg/C^2]$

The velocity of a light beam propagating in a medium of permeability μ and permitivity ϵ is:

$$v = \frac{1}{\sqrt{\epsilon\mu}} \tag{2.23}$$

The intensity was found to decrease along the propagation vector. Its decrease is due to loosing partial energy to the medium of propagation. This relation was formulated in the form: $\frac{I_2}{I_1} = \frac{r_1^2}{r_2^2}$. The distances r_1 and r_2 are on the same propagation vector.

The intensity of light beams is commonly computed as $I = power/area$. The power is in units of *watts* and the area is the area of the cresssection of the light beam. Therefore, the intensity is usually defined as the distribution of energy over some area.

Polarization of light is an important characteristic and therefore will be given special attention.

2.1.1.1 Polarization

It has already been mentioned that light may be treated as a transverse electromagnetic wave. The direction of the electric field is what determines the polarization of the beam. The magnitude and sign of the electric field varies along each wave and repeats itself in each of the following waves, provided the polarization is not changed and the wave did not lose part of its energy.

2.1.1.1.1 Linear Polarization

The electric field in an electromagnetic wave could be analysed into two orthogonal optical disturbances $\mathbf{E}_x(z,t)$ and $\mathbf{E}_y(z,t)$. The resultant optical disturbance is the vector sum of them,

$$\mathbf{E}(z,t) = \mathbf{E}_x(z,t) + \mathbf{E}_y(z,t) \tag{2.24}$$

The two orthogonal disturbances could have a phase difference between them. That is, they could have originated at different times and therefore have a different initial phase. The phase difference, due to whatever reason, between the two is what determines the type of polarization of the resultant optical disturbance.

If the phase difference $\varepsilon = 0(in-phase)$ or $180°(out-of-phase)$, then the beam is linearly polarized. In the case of 180° phase difference, the plane of vibration is rotated from that of the 0 phase difference by 90°. Figure 2.1 illustrate these ideas. In linearly polarized light beams, the electric field vector propagates in a single plane.

2.1.1.1.2 Circular Polarization

Another case of interest arises when both $\mathbf{E}_x$ and $\mathbf{E}_y$ have equal amplitudes (i.e., $E_{0x} = E_{oy} = E_0$), and relative phase difference $\varepsilon = -\pi/2 + 2m\pi$, where

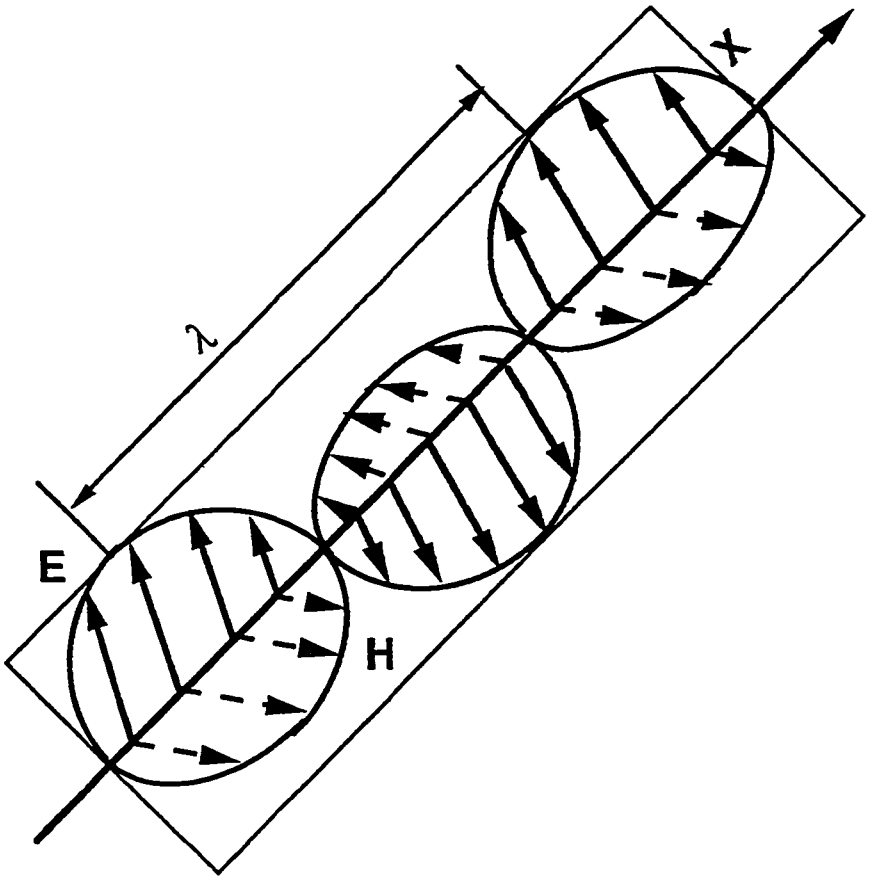


Figure 2.1 Plane (linearly) polarized light

$m = 0, \pm 1, \pm 2, \ldots$. This causes the amplitude of the resultant wave to be a constant. But the direction of **E** to be time varying, and not restricted to a single plane, see Figure 2.2.

This is called circularly polarized light, and more precisly **right-circularly polarized** because the electric field vector **E** rotates clockwize.

On the other hand, if $\epsilon = \frac{\pi}{2} + 2m\pi$, where $m = 0, \pm 1, \pm 2, \ldots$, **E** will rotate counterclockwize this time. Henceforth, the light beam is called **left-circularly polarized.**

2.1.1.1.3 Elliptical Polarization

In this case, $\epsilon = \pm\pi/2, \pm 3\pi/2, \pm 5\pi/2, \ldots$, the resultant electric field vector **E** will rotate changing its direction as well as magnitude. See Figure 2.3 for more information.

2.1.1.1.4 Unpolarized Light

Natural light is unpolarized. A fair number of artificially emitted light beams are unpolarized as well. This means that the electric field vector is distributed, in terms of magnitude, all over its propagation vector.

2.1.2 Gaussian Beam Optics

In order to gain an appreciation of the principles and limitations of beam optics, it is necessary to understand the nature of the light beam. The light wave propagates in three-dimensional space in the form of a plane, with ∞ radius of curvature. As it propagates, this plane wave begins to curve into a spherical wave with R radius of curvature.

When we take into account the wave nature of light, we find ourselves lead to the very basic and important concept of a *Gaussian spherical light beam/wave.*

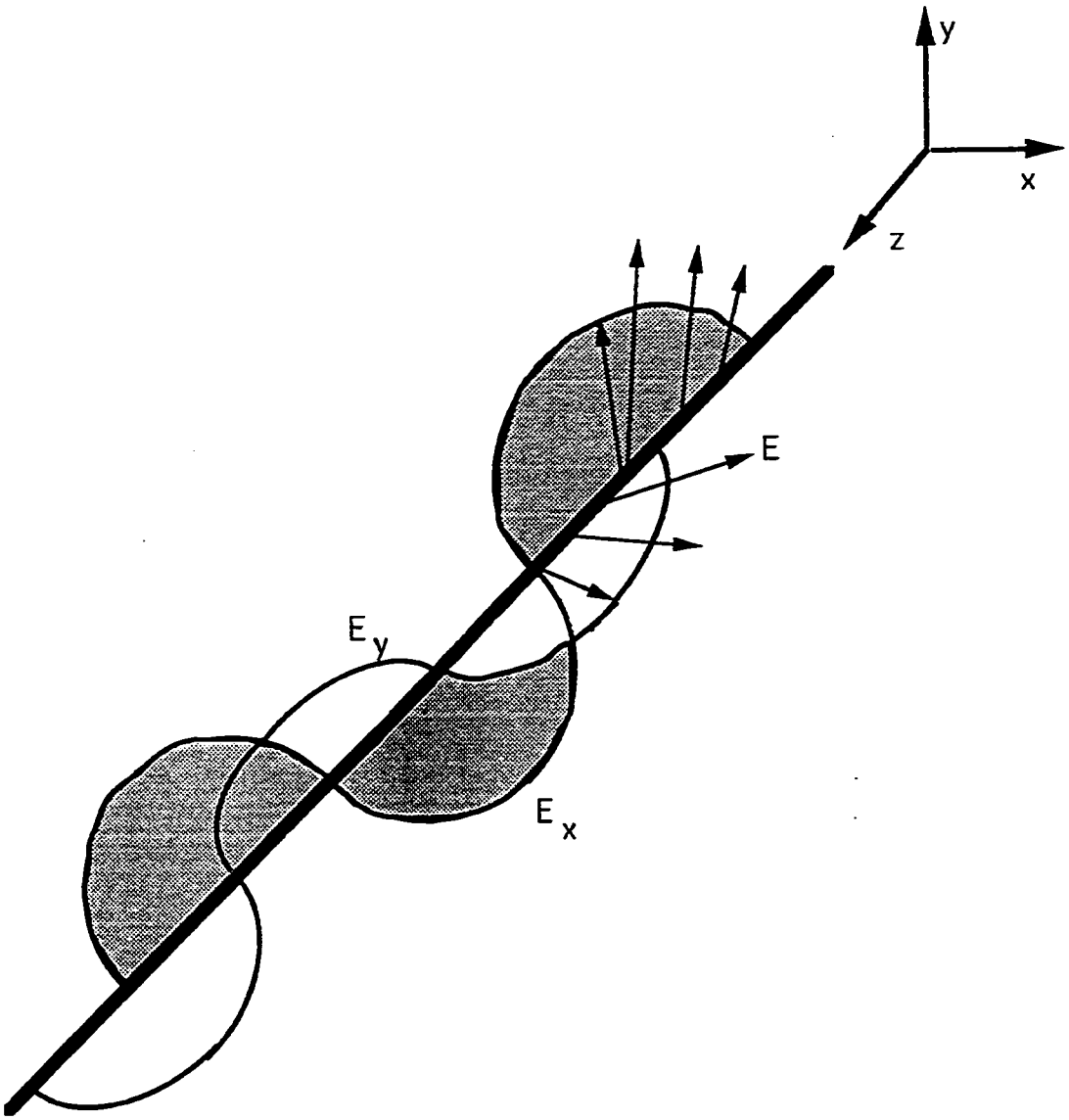


Figure 2.2 Left-circularly polarized light

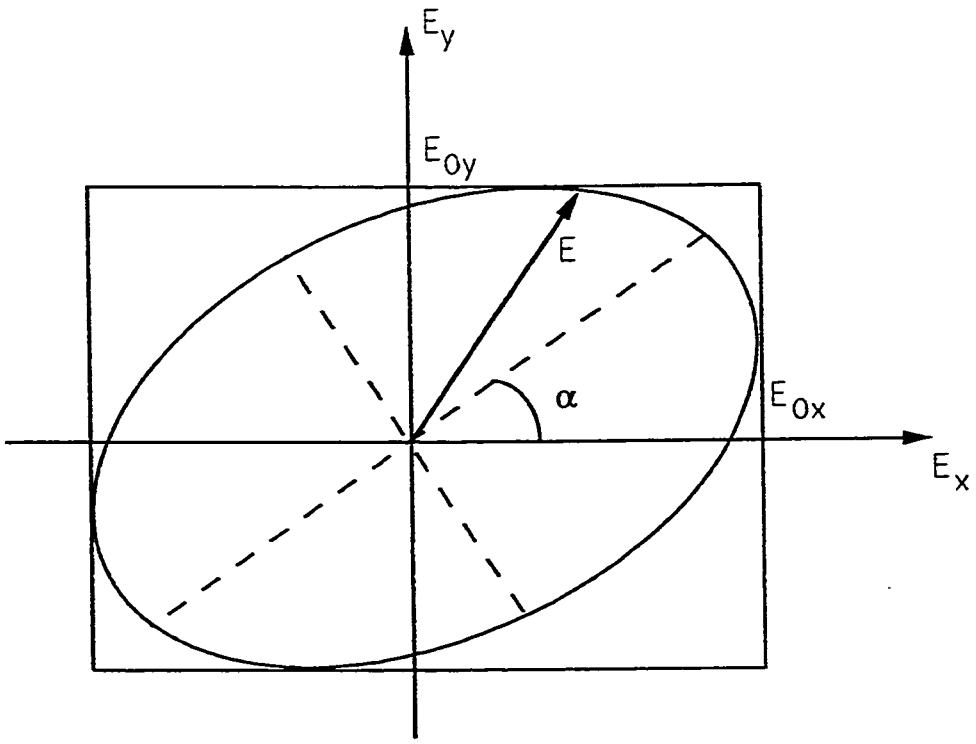


Figure 2.3 Eliptically polarized light

The beams emitted from lasers and other artificial sources, the kind of beams widely used in optical computing, have perfect wavefronts and Gaussian transverse irradiance profile. The Gaussian intensity distribution is taken at a cross-section of the light beam. This radially symmetric distribution whose electric field variation is given by:

$$E_s = E_0 \; exp \left(-\frac{r^2}{w_0^2}\right) \tag{2.25}$$

has the interesting mathematical property that its Fourier transform is also a Gaussian distribution.

It has been determined that a Gaussian source distribution remains Gaussian at every point along its path of propagation through the optical system. Of course, its size will change as it is focused by lenses , but the intensity remains Gaussian. This makes it particularly easy to visualize the distribution of the fields at any point in the optical system. The Gaussian distribution has no obvious boundaries to give it a characteristic dimension like the diameter of the circular aperture, for example. The definition of the size of the Gaussian is somewhat arbitrary. One could define the radius of the Gaussian as the distance from the axis at which the intensity has decreased to some fraction of the value on the axis. Figure 2.4 shows the Gaussian intensity distribution:

$$I(r) = I_0 \; exp \left[-\frac{2r^2}{w_0^2}\right] \tag{2.26}$$

where $r = a$. This distribution might be observed at the output of a laser. The parameter w_0, usually called the Gaussian beam radius, is the radius at which the intensity has reduced to $1/e^2$ or 0.135 of its value at the axis. Note that the region near the axis over which the Gaussian is reasonably constant is rather small. Nearly 100% of the power of the beam is contained in a radius $r = 2w_0$. One half the power is contained within 0.59 w_0, and only 10% of the power is

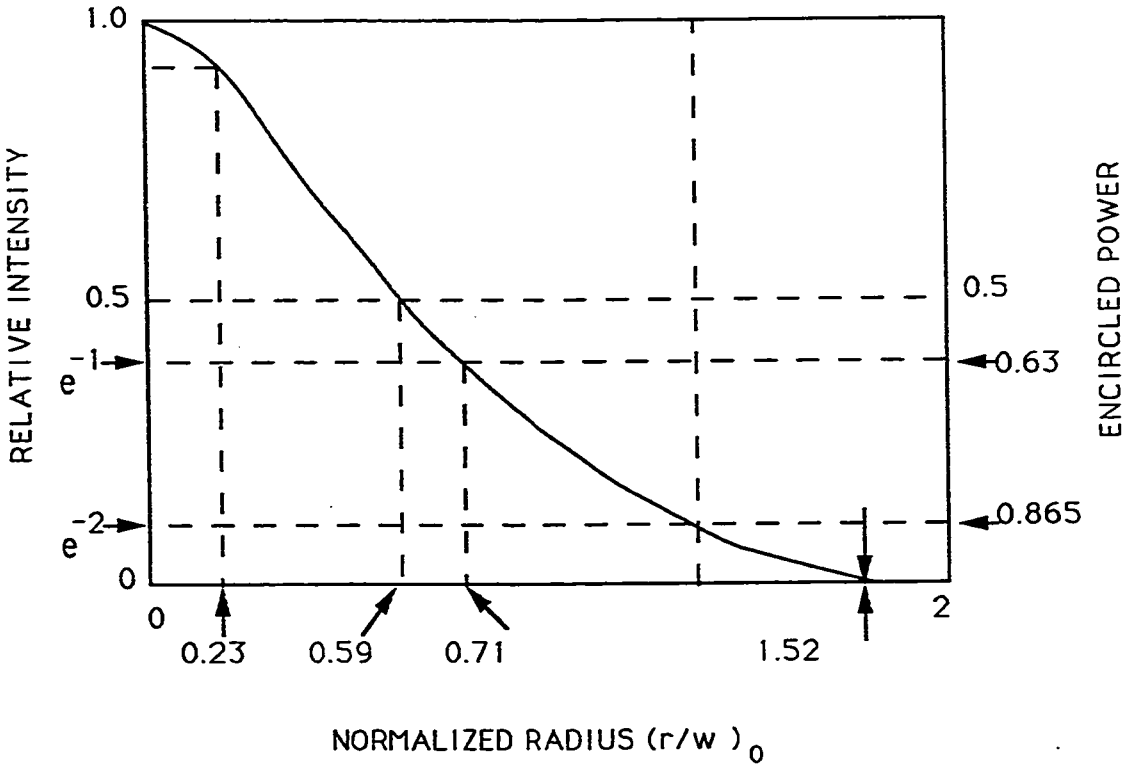


Figure 2.4 Gaussian intensity distribution

contained within 0.23 w_0, the radius at which the intensity has only decreased by 10%. The total power, p(∞)(Watts) is the quantity typicallly advertised by the manufacturers. It is related to the axis intensity, I(0) ($watts/m^2$), by:

$$P(\infty) = \left(\frac{\pi w_0^2}{2}\right) I(0) \tag{2.27}$$

$$I(0) = P(\infty)\left(\frac{2}{\pi w_0^2}\right) \tag{2.28}$$

Care should be taken in cutting off the Gaussian distribution with a very small aperture to make the beam more uniform over its extent. The source distribution would no longer be Gaussian, and the far-field intensity distribution would develope zeros and other non-Gaussian features. However, if the apperture is at least $3w_0$ or $4w_0$ in diameter, then these effects would be negligible. We will make this a requirement within our design among the used components.

Propagation of Gaussian beams through an optical system can be treated almost as simply as geometric optics. Because of the unique self-Fourier Transform of the Gaussian, we do not need an integral to describe the evolution of the intensity profile with distance; the transverse distribution of intensity remains Gaussian at every point in the system; only the radius of the Gaussian and the radius of curvature of the wavefront change. Imagine that we somehow create a coherent light beam with a Gaussian distribution and a plane wavefront at a position x=0. The beam size and wavefront will then vary with x as shown in Figure 2.5.

The equations describing the Gaussian beam radius w(x) and wavefront radius of curvature R(x) are:

$$w^2(x) = w_0^2\left[1 + \left(\frac{\lambda x}{\pi w_0^2}\right)^2\right] \tag{2.29}$$

$$R(x) = x\left[1 + \left(\frac{\pi w_0^2}{\lambda x}\right)^2\right] \tag{2.30}$$

where w_0 is the beam radius at $x = 0$ and λ is the wavelength.

These parameters could be merged into one parameter, x_R, the Rayleigh range:

$$x_R = \frac{\pi w_0^2}{\lambda} \tag{2.31}$$

In fact, it is at $x = x_R$ that R has its minimum value.

Note that these equations are also valid for negative values of x. We only imagined that the source of the beam was at $x = 0$; we could have created the same beam by creating a larger Gaussian beam with a negative wavefront curvature at some $x < 0$. This we can easily do with a lens, as illustrated in Figure 2.6.

The input to the lens is a Gaussian with diameter D and a wavefront radius of curvature which, when modified by the lens, will be $R(x)$ given by the equation above with the lens located at $-x$ from the beam waist at $x = 0$. That input Gaussian will also have a beam waist position and size (or Rayleigh range) associated with it. Thus we can generalize the law of propagation of a Gaussian through even a complicated optical system: In the free space between lenses, mirrors, etc., a Gaussian beam is specified in diameter and wavefront radius of curvature by the equations given above; the position of the beam waist and the waist diameter (or Rayleigh range) completely determine the beam. When a beam passes through a lens, mirror, or dielectric interface the diameter is unchanged, but the wavefront curvature is changed, resulting in new values of waist position and waist diameter (or Rayleigh range) on the output side of the interface.

These laws with input values of w and R will allow us to trace a Gaussian beam through any optical system. Of course, some restrictions must apply: optical surfaces need to be spherical and with not too short a focal length, so that beams do not change diameter too fast.

For weakly focusing systems, systems with high focal length, the beam waist

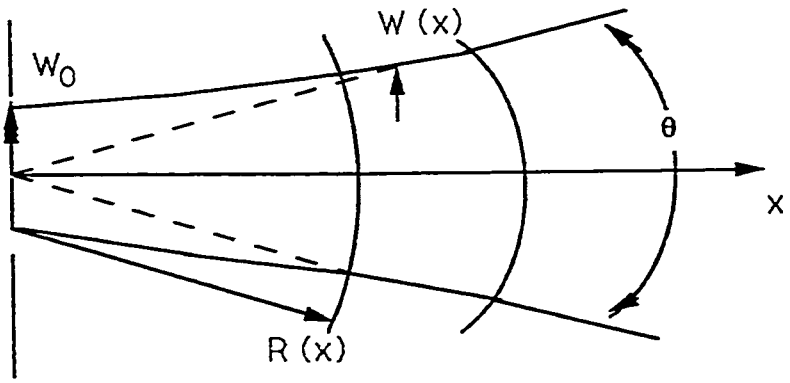


Figure 2.5 Beam size and wavefront curveture as they vary with distance

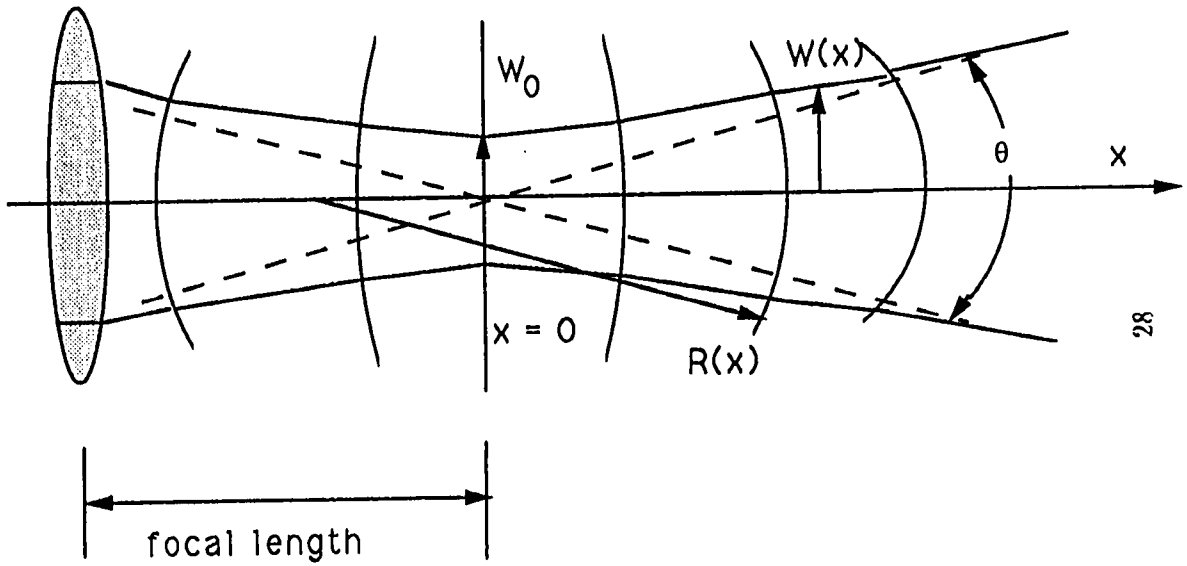


Figure 2.6 Focusing a collimated Gaussian beam using a lens

does not occur at the focal length. The waist moves toward the lens as the focal length of the lens is increased. However, we can easily believe the limiting case of this behavior by noting that a lens of infinite focal length such as a flat piece of glass, placed at the beam waist of a collimated beam will produce a new beam waist not at infinity but at the position of the glass itself.

2.2 Reflection/Transmission, Absorption, and Refraction

As the light beam propagates in space, it is most expected that it will collide with some object that might reside on the direction of propagation of the beam. Upon incidence, the beam power gets partially reflected in a certain direction, transmitted with a certain deviation or displacement, and partially absorbed by the object. This process is controlled by a set of rules. The derivation of these rules is not discussed here. For further information, the reader is advised to check the list of references mentioned at the begining of the chapter.

2.2.1 Reflection/Transmission and Refraction

To simplify reflection and transmission calculations, the incident electric field is broken into two plane polarized components. The p-polarized component, where the electric field vector is parallel to the **plane of incidence** (see Figure 2.7) and the s-polarized component, where the electric field vector is perpendicular to the plane of incidence (see Figure 2.8).

Fresnel equations give the *amplitude reflection coefficient* (r) as well as the

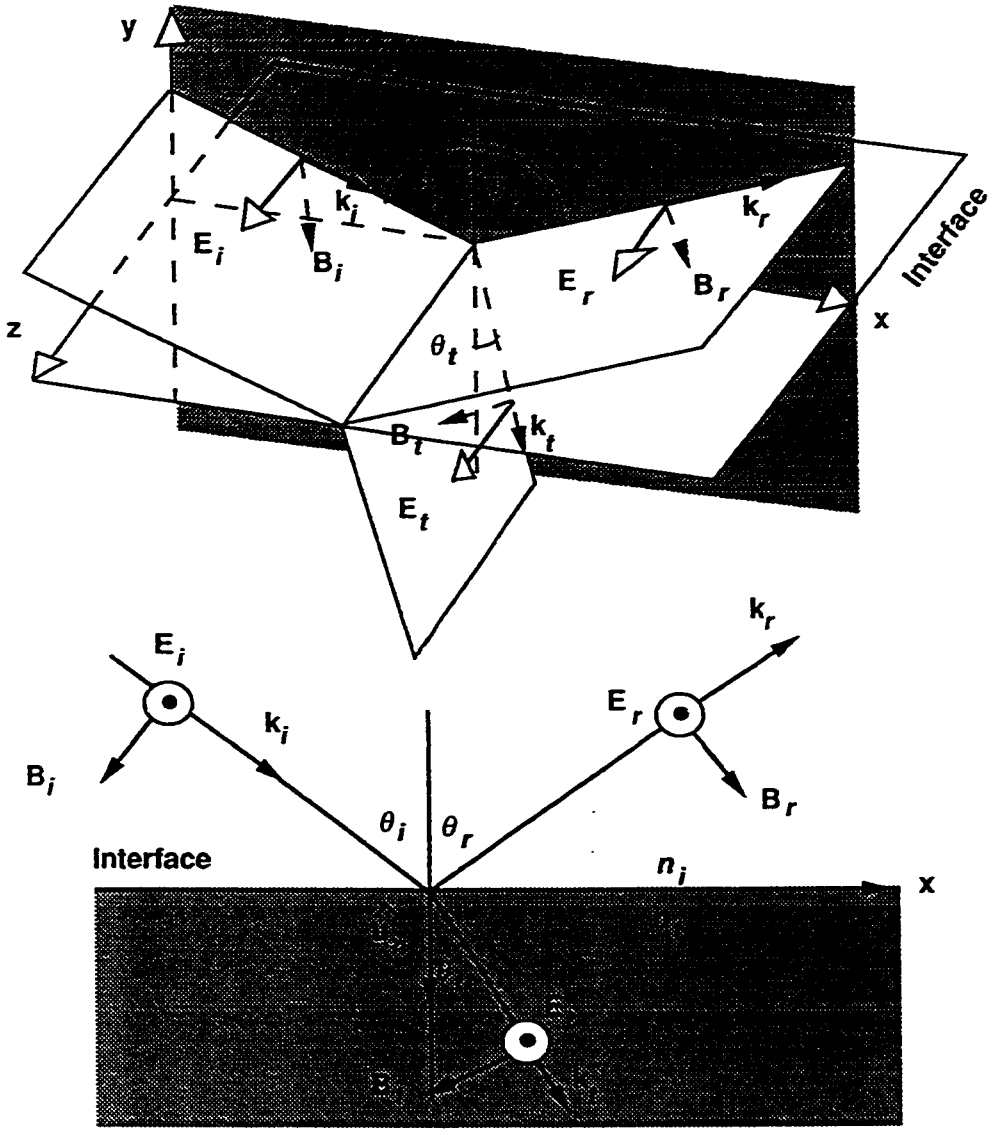


Figure 2.7 Incident light beam with *E* field normal to the plane of incidence

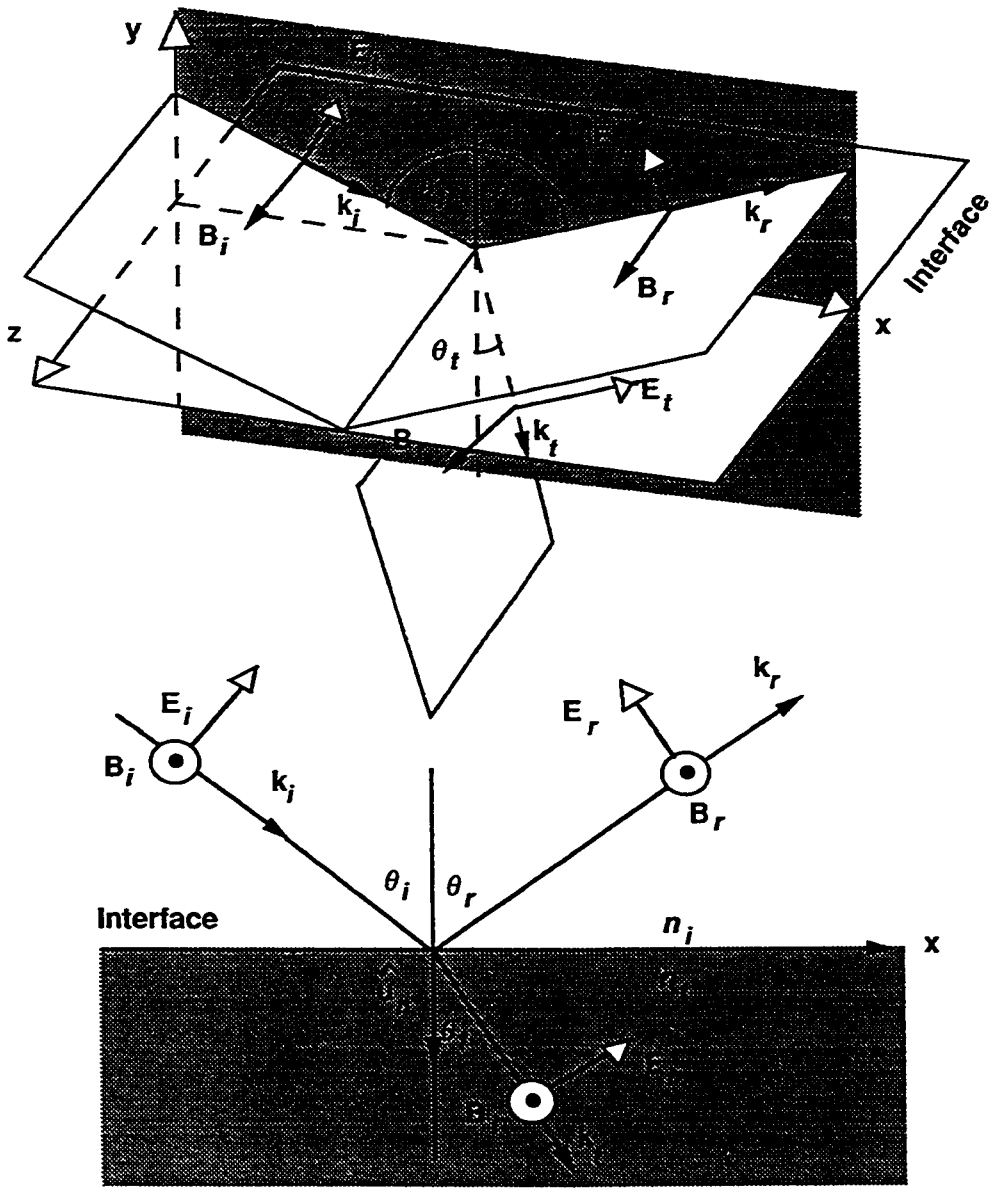


Figure 2.8 Incident light beam with *E* field parallel to the plane of incidence

amplitude transmission coefficient (t) for both components as follows:

$$r_s = \left(\frac{E_{0r}}{E_{0i}}\right)_s = \frac{\frac{n_i}{\mu_i}cos\theta_i - \frac{n_t}{\mu_t}cos\theta_t}{\frac{n_i}{\mu_i}cos\theta_i + \frac{n_t}{\mu_t}cos\theta_t} \tag{2.32}$$

$$r_p = \left(\frac{E_{0r}}{E_{0i}}\right)_p = \frac{\frac{n_t}{\mu_t}cos\theta_i - \frac{n_i}{\mu_i}cos\theta_t}{\frac{n_i}{\mu_i}cos\theta_t + \frac{n_t}{\mu_t}cos\theta_i} \tag{2.33}$$

$$t_s = \left(\frac{E_{0t}}{E_{0i}}\right)_s = \frac{2\frac{n_i}{\mu_i}cos\theta_i}{\frac{n_i}{\mu_i}cos\theta_i + \frac{n_t}{\mu_t}cos\theta_t} \tag{2.34}$$

$$t_p = \left(\frac{E_{0t}}{E_{0i}}\right)_p = \frac{2\frac{n_i}{\mu_i}cos\theta_i}{\frac{n_i}{\mu_i}cos\theta_t + \frac{n_t}{\mu_t}cos\theta_i} \tag{2.35}$$

where, the symbols used have the following meaning.

i incident medium.

t transmission medium.

θ_i incident angle.

θ_r reflection angle. According to the law of reflection $\theta_r = \theta_i$.

θ_t transmission angle, computed using **Snell's Law** of refraction.

$n_i \; sin \; \theta_i = n_t \; sin \; \theta_t$.

n_i refractive index of the incident medium.

n_t refractive index of the transmission medium.

μ_i permeability of the incident medium.

μ_t permeability of the transmission medium.

If both media forming the interface are dielectric, the equations above could be used without the permeability coefficients.

The sum of r_s and r_p is the total reflection coefficient of the incident light beam. And so is the sum of t_s and t_p.

2.2.1.1 Interpretation of Fresnel equations

Because of the great importance of **Fresnel** equations, they were interpreted and the following interesting results were found:

1. When $n_t > n_i$ which implies that $\theta_i > \theta_t$, it is found that r_s is negative for all θ_i's.

2. r_p is positive when $\theta_i = 0$. Moreover, $r_p = 0$ when $(\theta_i + \theta_t) = 90°$, in which case only the s-polarized component is reflected. θ_i in this case is the special **Brewster's Angle**. To compute it, we use the following:

$$\theta_\beta = arctan(n_t/n_i) \tag{2.36}$$

   This special angle is also called the **polarization angle**, because it transmits one plane-polarization component and reflects the other. For $\theta_i > \theta_\beta$, r_p becomes negative.

3. $t_s + (-r_s) = 1$, a fact for all θ_i's.

4. $r_p + t_p = 1$, only for $\theta_i = 90°$.

5. When $n_i > n_t$, which implies $\theta_t > \theta_i$, r_s is positive. r_s increases till it reaches $+1$ at θ_c, the **critical angle** when $\theta_c = \theta_t = 90°$. A situation called **total internal reflection**. After θ_c, all incoming light energy is reflected back. $r_p = +1$ at $\theta_i = \theta_c$ as well. Fresnel equations could be reformulated as follows:

$$r_s = \frac{cos\theta_i - (n_{ti}^2 - sin^2\theta_i)^{\frac{1}{2}}}{cos\theta_i + (n_{ti}^2 - sin^2\theta_i)^{\frac{1}{2}}} \tag{2.37}$$

(2.38)

$$r_p = \frac{n_{ti}^2 cos\theta_i - (n_{ti}^2 - sin^2\theta_i)^{\frac{1}{2}}}{n_{ti}^2 cos\theta_i + (n_{ti}^2 - sin^2\theta_i)^{\frac{1}{2}}} \quad (2.39)$$

6. r_p starts negative and reaches the 0 value at θ_β.

7. The sign of r_s could be positive if the direction of the incident electric field was reversed (i.e., rotated by 180°).

8. The component of the electric field normal to the plane of incidence undergoes a phase shift of π radians ($\Delta\varphi_s = \pi$). That happens upon reflection when $n_i > n_t$.

9. When $n_i > n_t$, no phase shift occurs in the normal component on reflection.

10. $\Delta\varphi_s = 0$ as long as $\theta_i < \theta_c$. Afterwards, there will be no transmittance and all of the beam is reflected.

11. r_p is positive and $\Delta\varphi_p = 0$ as long as:

$$n_t cos\theta_i - n_i cos\theta_t > 0 \quad (2.40)$$

which means that,

$$sin\theta_i cos\theta_i - cos\theta_t sin\theta_t > 0 \quad (2.41)$$

or equivelantly,

$$sin(\theta_i - \theta_t)cos(\theta_i + \theta_t) > 0 \quad (2.42)$$

This will be the case for $n_i < n_t$ if $(\theta_i + \theta_t) < \pi/2$, and for $n_i > n_t$ when $(\theta_i + \theta_t) > \pi/2$.

12. When $n_i < n_t$, $\Delta\varphi_p = 0$, until $\theta_i = \theta_\beta$. Afterwards $\Delta\varphi_p = \pi$, when $\theta_i > \theta_\beta$.

13. When $n_i > n_t, r_p$ is negative until $\theta_i = 90° - \theta_\beta$. This means that $\Delta\varphi_p = \pi$. For $90° - \theta_\beta \leq \theta_i \leq \theta_c$, r_p will be positive until $\Delta\varphi_p = 0$. Beyond θ_c, r_p becomes complex, and $\Delta\varphi_p$ gradually increases to π at $\theta_i = 90°$.

14. For normal incidence ($\theta_i = 0$), the following apply:

$$r = \frac{\frac{n_i}{\mu_i} - \frac{n_t}{\mu_t}}{\frac{n_i}{\mu_i} + \frac{n_t}{\mu_t}} \tag{2.43}$$

$$t = \frac{2\frac{n_i}{\mu_i}}{\frac{n_i}{\mu_i} + \frac{n_t}{\mu_t}} \tag{2.44}$$

2.2.1.2 Reflectance and Transmittance

Intensity of the light or its irradiance (I) was mentioned in the first section.

The intensity of the incident light beam is affected and Fresnel gave the basic concepts of these effects. How much light is transmitted and how much of it is reflected are determined by r and t. For this purpose, we define the terms reflectance (R), the ratio of the reflected power (or flux) to the incident power

$$R = \frac{I_r cos\theta_r}{I_i cos\theta_i} = \frac{I_r}{I_i} \tag{2.45}$$

We also define the transmittance (T) to be the ratio of the transmitted power to the incident one

$$T = \frac{I_t cos\theta_t}{I_i cos\theta_i} \tag{2.46}$$

In relation to r and t, these ratios are given by:

$$R = r^2 \tag{2.47}$$

$$T = \left(\frac{\frac{n_t}{\mu_t} cos\theta_t}{\frac{n_i}{\mu_i} cos\theta_i}\right) t^2 \tag{2.48}$$

$$R + T = 1 \tag{2.49}$$

$$R_{s,p} + T_{s,p} = 1 \tag{2.50}$$

2.2.2 Absorption

As the light beam passes through an object, part of its energy gets absorbed by the material of the object. This is described by the **exponential law of absorption**, sometimes called the **Bear-Lembert Law**, which states that:

$$I = I_0 e^{-K(\nu)x} \tag{2.51}$$

where I_0 is the intensity of the incident light, x is the thickness of the material and I is the intensity of the transmitted light. K(ν) is called the **absorption coefficient** with units of (cm^{-1}). The absorption coefficient is a function of the frequency/wavelength of the incident light. It is numerically equal to the inverse of distance (measured in centimeters) that a light beam must travel in a given medium, in order for the intensity to decrease to 1/e of the incident light intensity.

2.2.3 Refraction

In general, refraction is expressed by Snell's Law as described earlier. However, special cases for refraction are worth considering. These are displacement and deviation.

2.2.3.1 Displacement

A flat piece of glass can be used to *displace* a light ray laterally without changing its direction. The displacement varies with the angle of incidence, it is zero at normal incidence and equals the thickness of the flat at grazing incidence ($\theta_i = 90°$). The shape of the curve depends on the refractive index of the glass (flat), as shown in the following equation. See Figure 2.9.

$$d = h\ sin\theta_1 \left[1 - \frac{cos\theta_1}{\sqrt{\left(\frac{n_2}{n_1}\right)^2 - sin^2\theta_1}} \right] \tag{2.52}$$

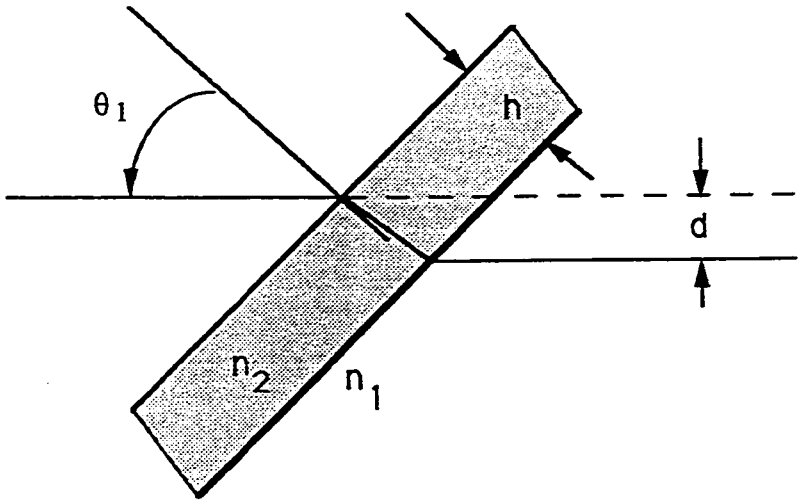


Figure 2.9 Displacement

The permeability μ might be needed if any of the media is not a dielectric.

2.2.3.2 Deviation

Additional deviation to the incident beam occurs if the media on the two sides of the flat are different (i.e., have different refractive indicies) (Figure 2.10). The displacement is the same, but the angular deviation δ is given by the formula:

$$\delta = \theta_1 - sin^{-1}\left(\frac{n_1}{n_3} sin\theta_1\right) \tag{2.53}$$

The permeability factors have to be added if any of the media is not a dielectric.

2.3 The Superposition of Waves

There might be cases where light waves share a common constant plane of vibration. Those beams or wavesmight all propagate on the same line, or they might just intersect and share a particular volume in space. The later case will be discussed when we address the issue of interference.

The former, however, is called superpositioning of waves, where light waves add up to each other. Adding up in the sense of adding their consecutive amplitudes. Care has to be given for the **frequency** of each of them. That is the case because adding two harmonic waves with different frequencies will produce an **anharmonic** wave as a resultant.

2.3.1 The Addition of Waves of the Same Frequency

Superposition of any number of coherent harmonic waves having a given frequency and traveling in the same direction, leads to a harmonic wave of the same frequency. The amplitudes of these waves do not simply add up, phase differences between them have to be taken into consideration. The phase difference occures because

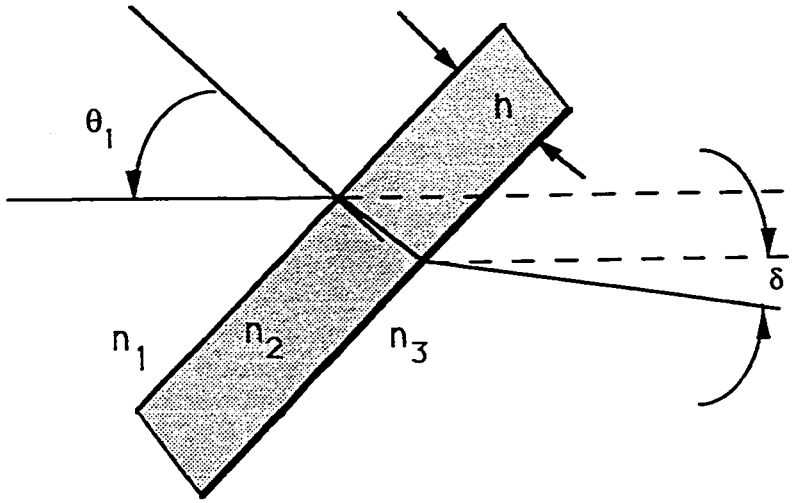


Figure 2.10 Deviation

the waves have propagated different distances, which creates the so called **optical path difference**. This difference directly causes the phase difference. In general, the sum of N such waves is,

$$E = \sum_{i=1}^{N} E_{0i} cos(\alpha_i \pm \omega t) \tag{2.54}$$

is given by

$$E = E_0 cos(\alpha \pm \omega t) \tag{2.55}$$

where

$$E_0^2 = \sum_{i=1}^{N} E_{0i}^2 + 2 \sum_{j>i}^{N} \sum_{i=1}^{N} E_{0i} E_{0j} cos(\alpha_i - \alpha_j) \tag{2.56}$$

and

$$tan\ \alpha = \frac{\sum_{i=1}^{N} E_{0i}\ sin\ \alpha_i}{\sum_{i=1}^{N} E_{0i}\ cos\ \alpha_i} \tag{2.57}$$

See Figure 2.11.

2.3.2 The Addition of Waves of Different Frequencies

This type of addition will not be of much concern to us, because in optical computing, we care about monochromatic light (light with a single frequency) rather than polychromatic light. That is, in any optical system, an observer can only see one frequency of the light beams propagating within the system.

The sum of waves of different frequencies keeps the repetition feature of harmonic waves. But not in the level of one wavelength repeating itself. Instead, a number of waves seem to repeat themselves over the propagation line. This group of waves is commonly called the *modulation envelope*, because its amplitude will be modulating as it propagates further. As an example, let us take the following two waves and sum them up.

$$E_1 = E_{01} cos(k_1 x - \omega_1 t) \tag{2.58}$$

$$E_2 = E_{02}cos(k_2x - \omega_2 t) \tag{2.59}$$

The resultant sum will be of the form:

$$E = E_{01}cos(\overline{k}x - \overline{\omega}t) \tag{2.60}$$

$$E_0(x,t) = (E_{01} + E_{02})cos(k_m x - \omega_m t) \tag{2.61}$$

where,

$\overline{\omega}$ average angular frequency $= 1/2(\omega_1 + \omega_2)$.

$\overline{k}$ average propagation number $= 1/2(k_1 + k_2)$

ω_m modulation frequency $= 1/2(\omega_1 - \omega_2)$.

k_m modulation propagation number $= 1/2(k_1 - k_2)$.

The total disturbance is a travelling wave of frequency $\overline{\omega}$ having a time- varying or modulated amplitude $E_0(x,t)$. The disturbance consists of a high frequency $(\overline{\omega})$ *carrier wave*, **amplitude-modulated** by a cosine function. The phase velocity of this carrier wave is:

$$v = \overline{\omega}/\overline{k} \tag{2.62}$$

Wherease, the rate at which the modulation envelope advances is known as the group velocity v_g.

$$v_g = \frac{\omega_m}{k_m} \tag{2.63}$$

Recall that the refractive index of the medium of propagation is related to the velocity of light in that medium $(v_g = \nu_g \lambda_g)$. Therefore, we can also define a group index of refraction n_g.

$$n_g = c/v_g \tag{2.64}$$

See Figure 2.12.

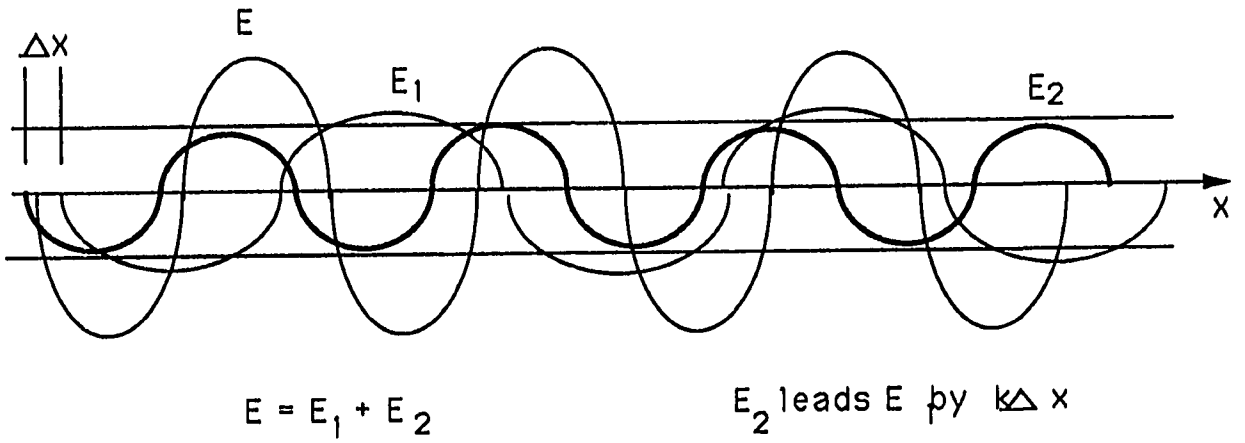


Figure 2.11 Light waves out of phase

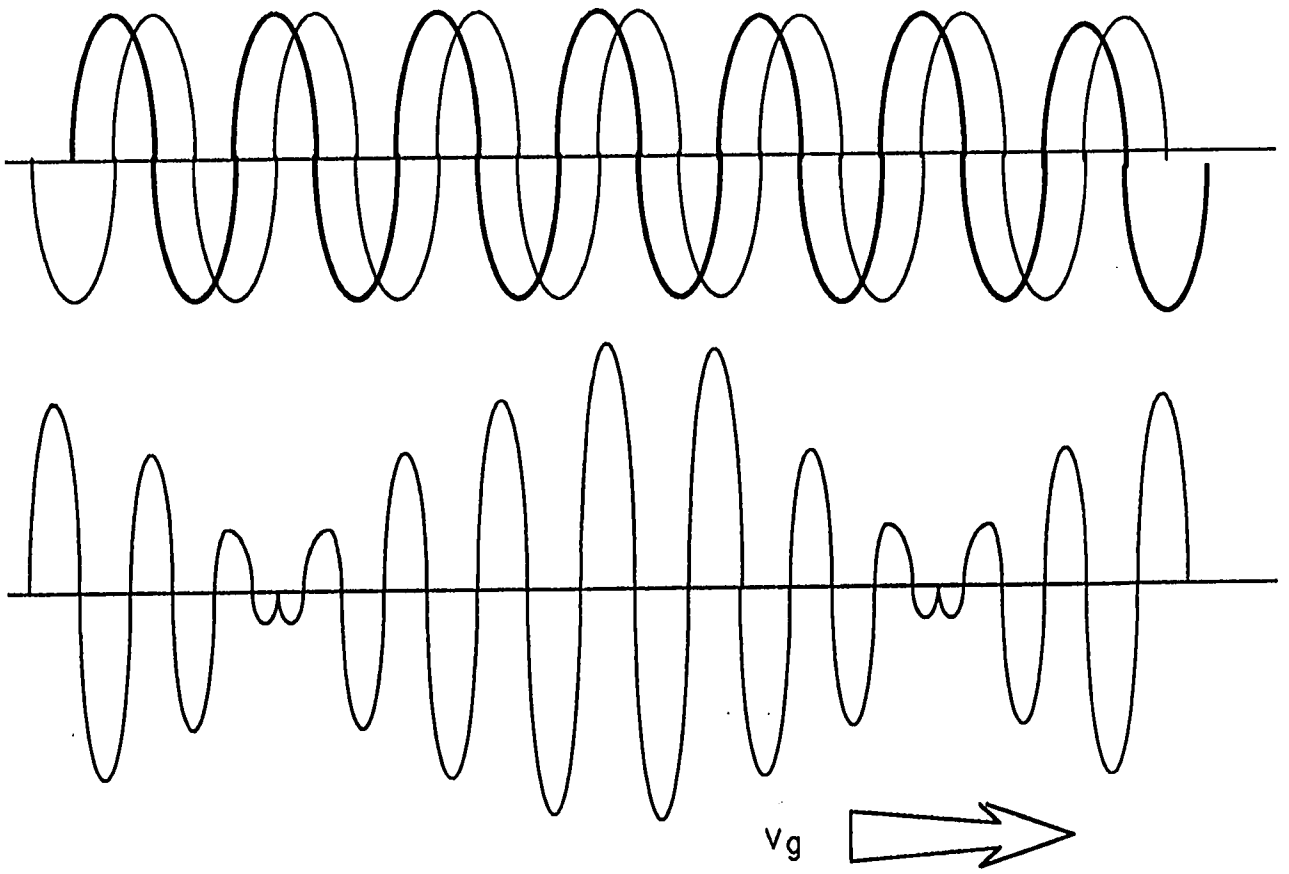


Figure 2.12 Group velocity

2.4 Interference

Interference could be defined as, the interaction of two or more light waves yielding a resultant irradiance (intensity) that deviates the sum of components irradiances. In Section 2.3, we considered the case when two or more light waves were propagating along the same line. Here, the situation is slightly different, since these waves might just intersect (i.e share a common volume in space for a certain period of time, see Figure 2.13). According to this, there will be an angle between the interfering waves. This angle will make the addition of these waves different from their sum.

Since in optical computing no light beams with different frequencies propagate in the optical system, care has to be given only to phase differences between interfering beams. No care will be given to beams frequencies. Let us now consider the interference of the two light waves:

$$\mathbf{E}_1(\mathbf{r},t) = \mathbf{E}_{01}\, cos(\mathbf{k}_1.\mathbf{r} - \omega t + \varepsilon_1) \tag{2.65}$$

$$\mathbf{E}_2(\mathbf{r},t) = \mathbf{E}_{02}\, cos(\mathbf{k}_2.\mathbf{r} - \omega t + \varepsilon_2) \tag{2.66}$$

The interference of these two waves will result in a sum of irradiances:

$$I = I_1 + I_2 + I_{12} \tag{2.67}$$

$$I_1 = <\mathbf{E}_1^2> \tag{2.68}$$

$$I_2 = <\mathbf{E}_2> \tag{2.69}$$

$$I_{12} = 2<\mathbf{E}_1.\mathbf{E}_2> \tag{2.70}$$

I_{12} is known as the *interference term*. To evaluate it, we form:

$$\mathbf{E}_1.\mathbf{E}_2 = \mathbf{E}_{01}.\mathbf{E}_{02} cos(\mathbf{k}_1.\mathbf{r} - \omega t + \varepsilon_1) \times cos(\mathbf{k}_2.\mathbf{r} - \omega t + \varepsilon_2) \tag{2.71}$$

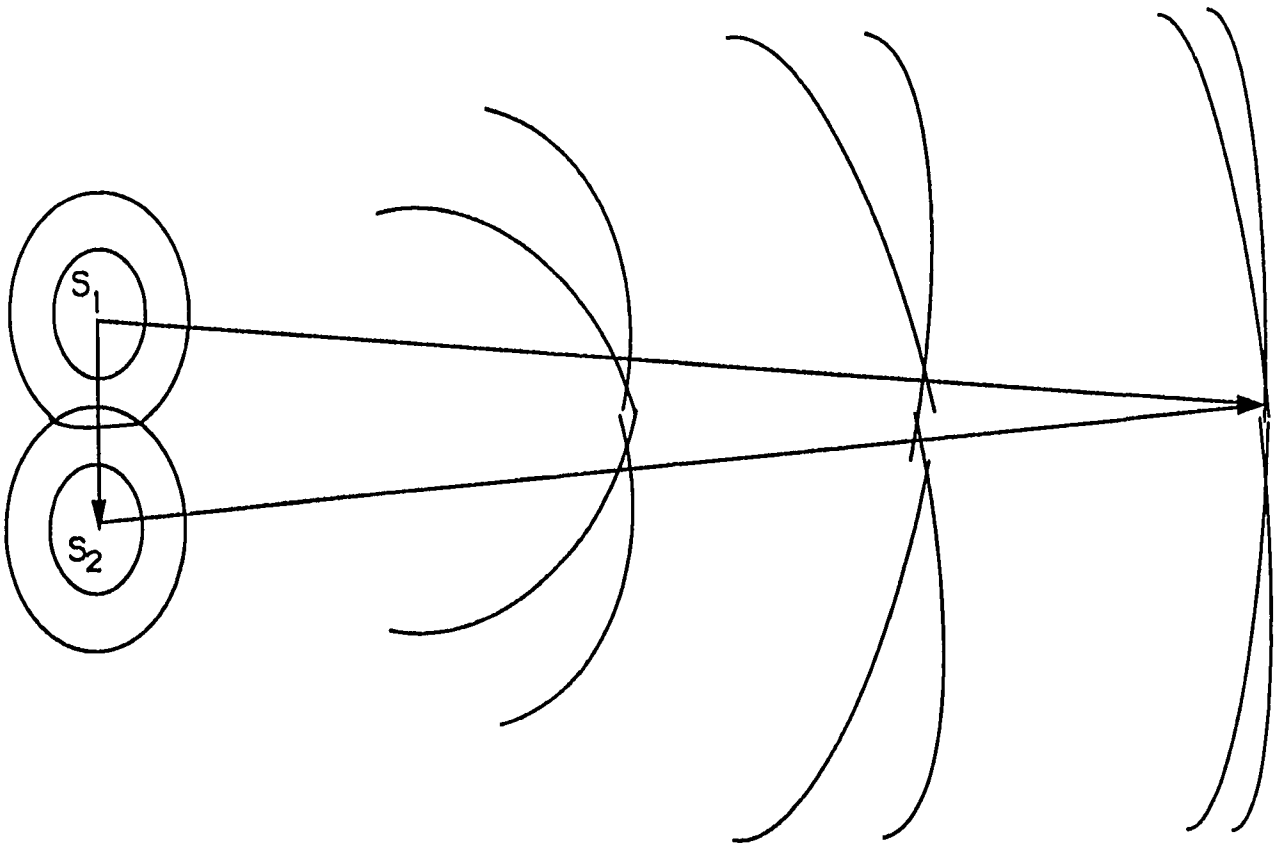


Figure 2.13 Waves from two point sources overlapping (interfering) in space

After a sequence of derivations, this term evaluates to:

$$I_{12} = \mathbf{E}_{01}.\mathbf{E}_{02} cos\delta \tag{2.72}$$

$$\delta = (\mathbf{k}_1.\mathbf{r} - \mathbf{k}_2.\mathbf{r} + \varepsilon_1 - \varepsilon_2) \tag{2.73}$$

Where δ is the *phase difference* resulting from a combined optical path length and initial phase-angle differences. Therefore,

$$I = I_1 + I_2 + 2\sqrt{I_1 I_2}\ cos\ \delta \tag{2.74}$$

which is the resultant irradiance of interference.

2.4.1 General Results

From what has been said, the following could be concluded:

1. *Total constructive interference* takes place when $\delta = 0, \pm 2\pi, \pm 4\pi, \ldots$, where $cos\ \delta = 1$. The irradiance takes its maximum value.

2. *Constructive interference* takes place when $0 < cos\ \delta < 1$.

3. *Total destructive interference* takes place when $\delta = \pm\pi, \pm 3\pi, \pm 5\pi, \ldots$. That is when $cos\delta = -1$. The irradiance takes its minimum value.

4. *Destructive interference*, on the other hand, takes place when $-1 < cos\delta < 0$.

5. For $\delta = \pm\pi/2, \pm 3\pi/2, \pm 5\pi/2, \ldots$.The interference term will disappear and the irradiance will take its medium value ($I = I_1 + I_2$). In the absence of I_{12}, there is no actual interference. Instead, the waves are said to have superpositioned or added up.

2.4.2 Interference of Polarized Light

Fresnel and Argo made an extensive study of the conditions under which the interference of polarized light occurs. The **Fresnel-Argo Laws** are as follows (Figure 2.14):

1. Two orthogonal light waves cannot interfere. Orthogonal here means that the angle between the electric field vectors of the two beams equals 90°. No interference in the sense that $I_{12} = 0$ (i.e., the interference term disappears).

2. Two light waves with parallel electric field vectors will interfere in the same way as will natural light.

3. The two constituent orthogonal electric field vectors within natural light cannot interfere.

2.5 Diffraction

Diffraction of light could be defined as, the deviation of light from rectilinear propagation. Diffraction occurs whenever a portion of a wavefront is obstructed.

There is no significant physical distinction between *interference* and *diffraction*. However, it has become somewhat customery, if not always, to speak of interference when considering the superposition of only a few waves and diffraction when treating a large number of waves.

The most famous scientists who studied diffraction were, *Fresnel* and *Fraunhofer*. Fresnel said, if the plane of observation is moved away from the

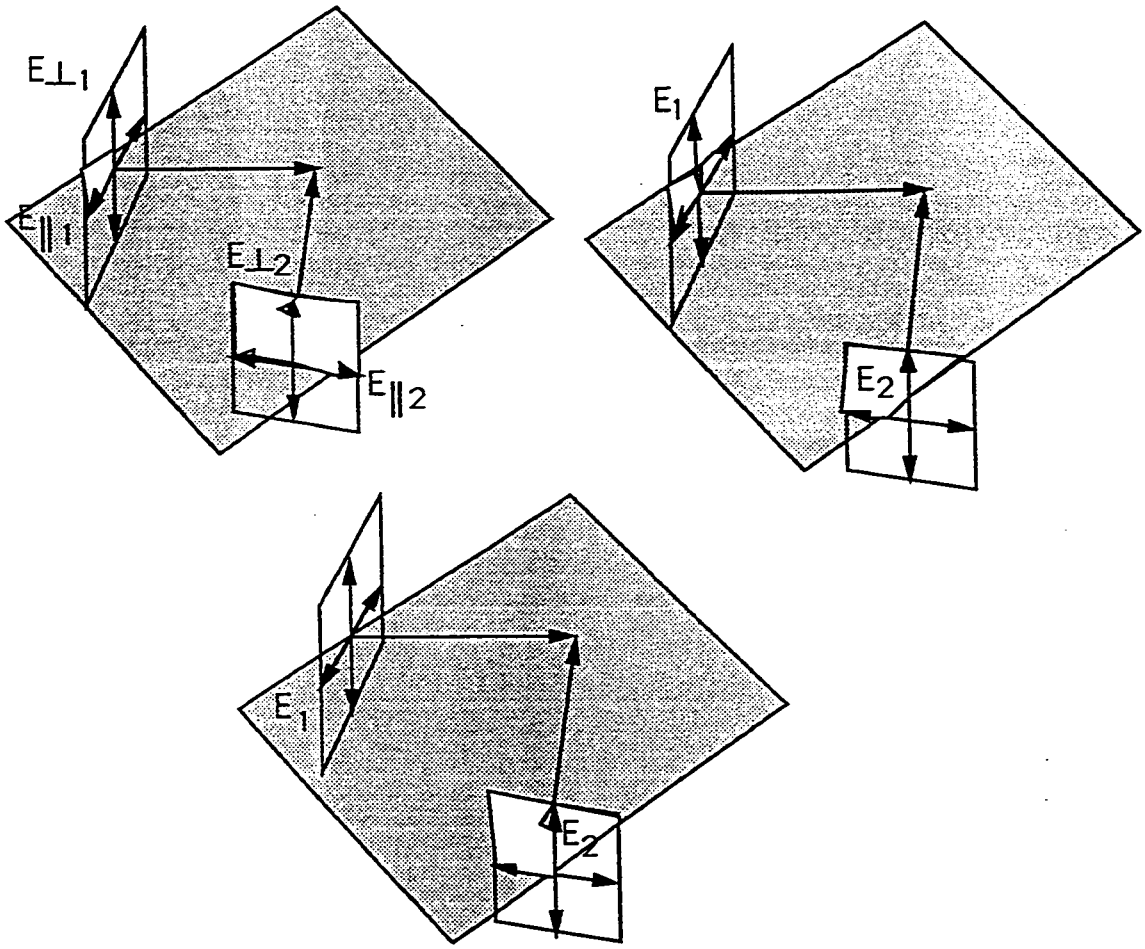


Figure 2.14 Interference of polarized light

obstructor, the image of the obstructor, although still easily recognizable, becomes increasingly more structured as the diffraction pattern becomes more prominent. Fraunhofer, on the other hand, said that moving the plane of observation far away from the obstructor changes only the size of the pattern and not its shape.

Fraunhofer and Fresnel diffractions are discussed thoroughly in a number of optics books. Our intentions here are far away from discussing any more details. However, we will be giving *diffraction gratings* special attention when discussed as one of the modeled components for our designs.

2.6 Coherence

Any light beam, although mathematically represented as a single wave, is not just that. A light wave is composed of a number of small waves, sometimes called wavelets, that combine to construct the beam. If these wavelets were all in phase, the light beam is said to be *coherent*. Unfortunatly, this is not always the case. In many situations, the phases of these wavelets and thus of the light wave, change in a random manner. When this happens, we say that the light is *incoherent*.

The most important difference between coherent and incoherent light is that mutually incoherent beams ***do not interfere***. When two incoherent beams combine, their intensities (rather than their complex amplitudes) add up.

We distinguish between two types of coherence in light:

Spatial coherence the phase relation between different parts of the cross-section of the beam are constant; therfore, different parts of the cross-section can be made to interfere with each other.

Temporal coherence the phase relations between consecutive wavefronts are

constant; therefore, a beam can interfere with a delayed version of itself. Temporal coherence is actually equivalent to monochromaticity of the light: it means that the frequency of the light beam is stable and confined to a narrow band.

The boundary between coherent and incoherent light is not clear-cut.

A measure of coherence is the *coherence length* of the light. This is the width of beam over which spatial coherence is maintained, or the distance the light travels before temporal coherence is lost. The path length in a system must be shorter than the coherence length for the system to be coherent.

2.7 Basic Elements of Optical Systems

In this section, a brief description of selected optical components will be given. The selection of these components was based on their usage in a number of architectures. We studied a large number of optical systems from which we extracted a number of components that are heavily used.

2.7.1 Mirrors

Mirrors are used to reflect and hence, change the direction of light beams. Upon reflection, the polarization of the reflected light beam is affected: it tends to be parallel to the plane of incidence with the mirror.

2.7.2 Continuous Laser Sources

These are the major sources of light beams/signals in any optical system. The description of their action is beyond the scope of this thesis.

Laser beams are intense, monochromatic, coherent, and directional. Their output is tunable, in terms of both frequency and intensity. The wavefronts of laser beams are planar at the laser source.

2.7.3 Pulsed Laser Sources

The output of these components is a laser beam in the form of pulses. These pulses are generated at a certain pulse rate and for a constant pulse duration.

The pulsed laser technology has improved significantly in the last decade. The output of the new models is stable, in terms of frequency, intensity, as well as pulse duration.

Their output is intense, monochromatic, coherent, and directional.

Intensity, frequency, pulse rate, and pulse duration can be tuned manually.

2.7.4 Beam Splitters

Beam Splitters are basically used to split any input light beam into two output beams. However, they can also join two input beams into one output beam.

Their function is simply to transmit a portion of the incident beam and reflect the other. The output beams will have partial characteristics of the input beam, such as partial intensity.

To join beams, the two input beams are directed in such a way that the reflected part of one of the beams and the transmitted part of the other come out together.

In either one of the two uses of beam splitters, half of the energy of the input beams is lost.

Some beam splitters are sensetive to the state of polarization of the incident beam. Therefore, care should be taken when dealing with this type of beam splitters.

Another type of beam splitters, the *polarizing beam splitters* are going to be discussed as a type of polarizers.

2.7.5 Windows

Windows are used to pass light beams from one medium to the other, such as passing a laser beam from air to a box containing some liquid.

2.7.6 Prisms

Prisms act as mirrors, in the sense that they change the direction of the incident beam. However, prisms refract light beams rather than reflecting them.

2.7.7 Retardation Plates

The main purpose of using retardation plates is to change the polarization of the input light beam.

Two types of retardation plates are in use, **half-wave plates** and **quarter-wave plates.**

Half-wave plates rotate the polarization plane of a plane-polarized light by 90°. If the input beam was circularly polarized, they change that from left-circular to right-circular and visa versa.

Quarter-wave plates, on the other hand, change the plane polarized input into a circularly polarized output and visa versa.

2.7.8 Lenses

Lenses are optical components that can form images. This is what makes them so important.

The famous type of lenses are spherical *convex* and *concave* lenses. Convex lenses have positive focal lengths; converge incident light; and form both real images (as might be focused on a piece of paper) and virtual images (as are seen through the lenses if they are used as magnifiers). They also collimate light coming from a point source. They are widely used in telescopes, collimators, optical transivers, magnifiers, and radiometers.

Concave lenses have negative focal lengths, diverge collimated incident light, and form only virtual images which are seen through the lens. They are often used to expand light beams. They also can convert a convergent beam into a collimated one.

Bi-convex, planar convex, bi-concave, planar concave, meniscus convex, and meniscus concave lenses are diversities of the spherical lenses.

Cylindrical lenses are another type of lenses. They are used in applications requiring magnification in one dimension only, such as transforming a point image into a line image or changing the height of an image without changing its width and visa versa.

2.7.9 Polarizers

Polarizers are components that alter the polarization state of the incident light. In what follows, we will discuss three different types of polarizers with different effects on light polarization.

2.7.9.1 Birefringent Polarizers

A birefringent crystal, will divide an entering beam of monochromatic light into two beams having orthogonal plane polarizations. The beams will usually propagate in different directios and both have differnt propagation speeds.

Depending on whether the crystal is uniaxial or biaxial, there will be one or two directions (the optic axis direction) within the crystal along which the beam will remain collinear and continue to propagate at the same speed.

The point here is that these crystals have two refractive indices. The converging beams will follow these indices and propagate at speeds and directions relative to them. The two beams are called *ordinary* and *extraordinary*. The ordinary beam is the s-polarized one and the extraordinary is the one with p-polarization. The refractive indices are therefore called n_e (extraordinary) and n_o (ordinary).

If $n_e > n_o$, then the velocity of the s-polarized ray is higher than that of the p-polarized and visa versa.

One last point is that the ordinary ray will propagate through the crystal undeviated only when $n_e > n_o$, while the extraordinary ray deviates by a specific angle. The converse occurs when $n_o > n_e$.

2.7.9.2 Polarizing Beam Splitters

The rule of splitting one input beam into two and joining two input beams into one, is still valid here as it was with ordinary beam splitters. The difference is that this component splits the input beam into two orthogonally polarized output beams.

One of the output two beams will have the direction of the input (i.e., it will be transmitted through the splitter. The other, on the other hand, will be reflected by 90°. The transmitted one will be p-polarized while the other will be s-polarized.

2.7.9.3 Dichroic Sheet Polarizers

Sheet type or dichroic polarizers are made of dichroic materials.

Dichroic materials are birefringent materials in which one of the two orthogonal polarizations (either ordinary or extraordinary) is subject to strong absorption. At the same time, the other polarization is not.

Henceforth, these components pass beams with polarization planes parallel to their optic axis only. The other polarization plane is completely absorbed by the sheet.

2.7.10 Diffraction Gratings

A diffraction grating can be classified into two types. The first type has a number of fine, straight, parallel, and transparent slits one a piece of metal. The other consists of a substrate usually of an óptical material; with a number of parallel grooves ruled or replicated in its surface, overcoated with a reflecting material.

The spacing of the slits/grooves is extremely crucial. With a grating, the incident beam is split (transmitted as in the case of slits or reflected as in the case of grooves) into a number of beams that propagate in different directions. These directions are determined by the number of slits/grooves that get impigned by the incident beam, the spacing between these slits/grooves, as well as the wavelength of the incident beam.

The diffraction grating, therefore, is a repetitive array of diffracting elements, either apertures (slits) or obstacles (grooves), that has the effect of producing periodic alterations in the phase, amplitude, or both of an emergent beam/wave.

The main use of gratings in optical systems is to divide the amplitude of the incident beam in many directions with different portions of amplitude. One other use of it is in *spectroscopy*, wher white light gets analysed into its composing frequencies.

2.7.11 Filters

A filter passes selected harmonics of the incident wave through while heavily absorbing/reflecting the rest of the harmonics. A number of filters and their

uses will be introduced in this section.

2.7.11.1 Attenuation/Density Filters

They are basically used to reduce the intensity of a light beam. This is done by absorption and/or reflection of a specific (depending on the used model) amount of the input light energy (intensity).

2.7.11.2 Wavelength Selective Filters (Color Filters)

These filters are used to produce or select a specific color (frequency) or band of color (band of frequencies) from a polychromatic (white, for example) light source. This is achieved by isolating the needed frequency(ies) or rejecting the unneeded frequency(ies). One classification of this type of filters is as follows:

2.7.11.2.1 Cut-off Filters

In this kind of filters, there will be an abrupt division between the regions (frequency regions) of higher and lower transmission. If the filter transmits higher frequencies and rejects lower ones, it is called a *short-wave-pass* (or high-pass) filter. The exact opposite is called a *long-wave-pass* (or low-pass) filter. The cutoff is defined to be the wavelength with 37% transmission.

2.7.11.2.2 Bandpass Filters

The same discussion is applied here, but the curve of transmittance versus wavelength tends to look more normal than that of cut-off filters. In this case, a specific wavelength (frequency) will have a maximum transmittance. There will be however a certain passband which includes all wavelengths that could be passed/transmitted. The rest of the wavelength band will be rejected.

Another classification of wavelength selective filters is the one that classifies them to:

Absorption Filters Where a certain band of wavelengths is absorbed by the filter and the the rest are passed.

Reflection Filters Also called " optical interference filters " where a very narrow wavelength band is allowed to pass, and the rest is reflected rather than absorbed.

A combination of Absorption and Reflection Filters In these filters, part of the wavelength is absorbed, another is reflected, and the rest is transmitted.

2.7.12 Spatial Light Modulators (SLMs)

A spatial light modulator is a device that creates some sort of modulation on the cross section of a beam of light. Usually a uniform beam of light impinges upon the SLM, and a modulated beam of light results. The resultant beam could be modulated in terms of its phase, amplitude, polarization, or any combination of these. The basic use of SLMs in optical computing is as storage devices. A character S, for example, can be imaged on a SLM which will cause different modulator than that of imaging another character, say F, for example.

There are good number of SLMs available at this time. The main criteria for comparing them and evaluating their performance are:

Method of writing on (exposing) the SLM There are two basic options:

Addressing each point separately Achieved by a scanning electron or by individual addressing electrodes.

Casting of the image Here, an image of the whole data is casted optically on the surface of the SLM.

Light Any requirement of the light. For example, if it must have a specific wavelength.

Modulation What type of modulation produced?. What is the modulation efficiency?.

Time How long is the right-use-erase cycle?. How long can the SLM keep the data stored in it?. Can it serve as a memory device?.

Operating conditions The power consumption, sensitivity size, temperature rang, etc.

Table 2.1 shows the type of modulation produced by different SLMs.

modulation	*realized by*	*SLM types*
amplitude	change of transmittance	photographic film photodichroic materials
phase	change of shape	thermoplastic micromechanical deformable mirror
	change of density	acoustooptic cell
polarization	physical effects in crystals	liquid crystal electrooptic crystals magnetooptic materials

Table 2.1 Underlying techniques for spatial light modulation

Chapter 3

MODELING OF OPTICAL COMPONENTS

The first step in describing optical architectures/systems is to explore the basic optical components. Therefore, the functionality of each component has to be simulated. Consequently, emphasis is given to the simulation of components, and in this chapter models of various basic optical components are described.

Light beams propagate within optical systems carrying data and transferring it from one component to the other as required by the design. As light propagates in media, its characteristics and parameters continuously change.

The characteristics and parameters are the ones discussed in Chapter 2 and they are:

- Direction in the three-dimensional space
- Velocity
- Wavelength/Frequency

- Polarization
- Phase
- Beam diameter

These characteristics are affected by the media it propagates through, and the optical components it passes through along its propagation vector.

For instance, the direction of the light beam is altered by components such as *mirrors*; the intensity, direction, and number of beams propagating within a system are changed by *beam splitters*; and the polarization of light can be changed from one state to another by *retardation plates*.

To conduct simulation, we developed models for almost every basic optical component. Each model makes use of a number of *attributes* of the component it describes. Attributes are used as part of mathematical formulae that simulate the functionality of the corresponding component. Thus mathematical models simulating the effects (functions) of their components on the characteristics and parameters of input light beams, are produced.

In this Chapter we define the attributes of each optical component. In addition to that, attributes of different media through which light propagates are also defined.

Based on the discussions with the users of optical components and the information documented in the optics manufacturers' catalogues, we infered that

- A large number of models of different optical components are provided by a number of manufacturers. The difference between these models is in the values of the components attributes

- Manufacturers' claims about the efficiency and correctness of their products are not taken for granted by users. Calibration is usually performed by users to confirm these claims
- Users can order components from manufacturers with specifications according to their needs. These specifications might not match those of manufacturers' lists
- Optics catalogues are more realistic in the sense that what they provide is usually what could be provided. For example, a user can imagine a spherical lens with a diameter of one meter but this lens could never be realized in reality

Therefore, we took the information provided by manufacturers and calibration processes into consideration. An example of such information is when manufacturers say that a certain mirror reflects 99% of its incident energy, this will be taken for granted until proven otherwise by calibration. Notice that we found no need to compute the reflectance by Fresnel equations. This saves both time and effort affecting neither accuracy nor efficiency of the system.

The *position*, *face of incidence*, and *orientation* of each optical component in the system are very important and have to be provided as part of the attributes of components.

The position of a component is a point in the three-dimensional space where a specific reference point in the component is placed. Each component will be given a reference point as an attribute. There might be components with two reference points and hence, two positions. For example, laser sources have a point at its center and another at its head to enable us to detect the direction of the output light. A position has three values for the three axes (x, y, z).

The face of incidence is the only face of the component that can take incident light as input to the component. Flat mirrors, for example, cannot reflect light that is incident on its back face. Only the front face, the face of incidence, can reflect incident light. Not all components have this feature. Cube beam splitters, for example, have six faces of incidence from which can enter them. This attribute takes one or a combination of the following values:

1. Up-z or Down-z corresponding to the positive or negative $z - axis$, respectively

2. Right-x or Left-x which correspond to the positive or negative $x - axis$

3. Back-y or Forth-y corresponding to the positive or negative $y - axis$

The orientation is the direction of the component in space. It is determined by two angles (θ, ϕ). θ is the angle between the projection of the normal to the face of incidence on the $x - y$ plane and the positive $x - axis$. It is taken negative $y - axis$ part of the plane. ϕ, on the other hand, is the angle between the normal to the face of incidence of the component and the positive $z - axis$ (Figure 3.1). Therefore, an axis for each component has to be determined.

One more note to make is that the *permeability* of media or components will only be needed if the media or components are dielectrics.

In order to fully detect the direction of the face of incidence, we have to give the orientation of its normal.This is done only when more than one of the above values are selected for defining the direction of face of incidence. For example, if the face was directed Up and Right.

In addition to that, the dimensions of the components have to be given. This is important because when placing these components, the system has to make sure

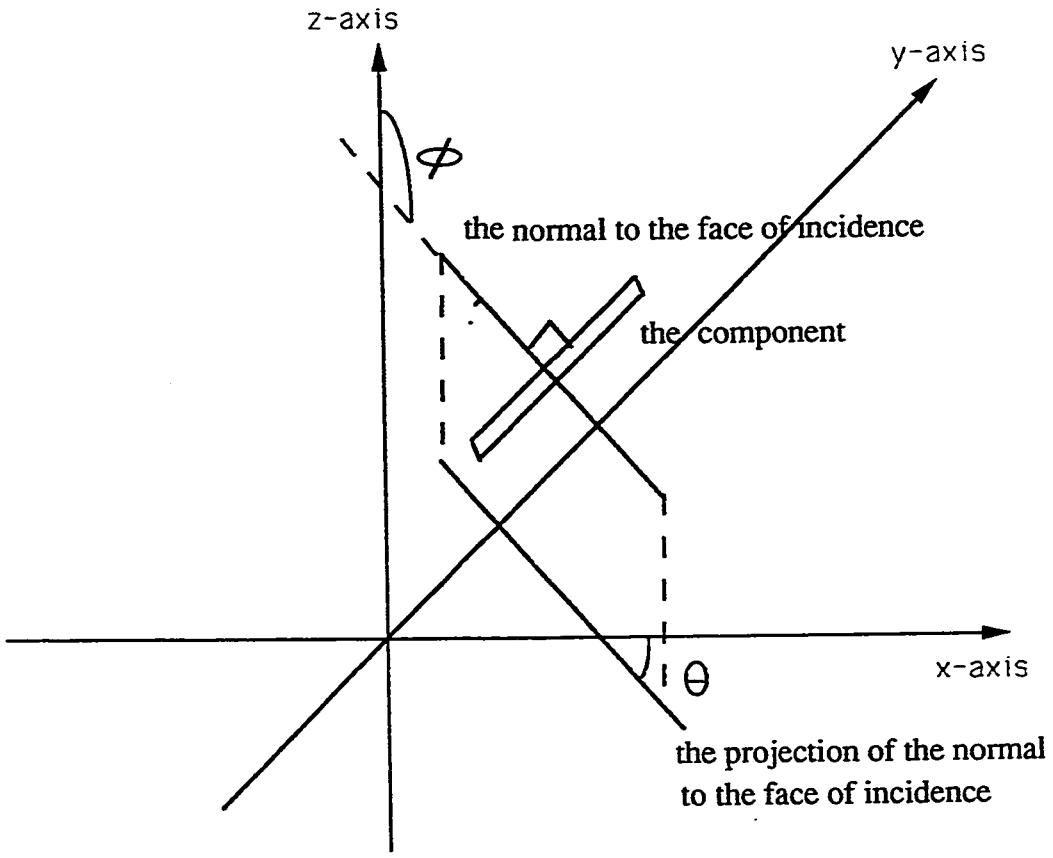


Figure 3.1 Orientation of optical components

that no two components occupy the same space/volume.

3.1 Light Propagation

Any light beam should have an origin. This origin (the light source) determines the initial characteristics and parameters of the light beam. These characteristics are also affected by the propagation media. The media might be a volume filled with air, gas, or liquid. In this section, we will not discuss the case of light propagating through optical components. Free space, however, is the commonly used medium and this is the case of most concern.

Any medium has a **refractive index** n and a **permeability** μ that determine the changes on the characteristics of the light beam that propagates through it. Some points to note are:

- The velocity of the light in the medium is computed by: $v = \frac{c}{n}$, c being the speed of light in vacuum
- The wavelength is computed by: $\lambda = \frac{\lambda_0}{n}$, where λ_0 is the wavelength of the beam as its source
- The frequency ν is determined by both velocity and wavelength: $\nu = \frac{v}{\lambda}$
- The polarization is determined by the source of the light beam, whether that was the original light source (polarized light from its origin) or an optical component that changed the original polarization
- *The phase can be taken to have a* **zero** *value at any time with no loss of generality.* The reason is that the wavelength band is never higher than units of *nanometers.* Furthermore, the level of accuracy the user can get

for a position is in units of *millimeters*, much larger than nanometers. Now let us assume that the user wants to probe for the phase of the light wave at a certain position, he will then be probing for a value that should not exceed units of nanometers at a position that cannot be in units less than millimeters. Therefore, one can answer this request giving a zero value for the phase without any loss of accuracy

- The direction of light is not changed as it propagates through homogeneous media
- The beam diameter changes according to Gaussian optics by: $w^2(x) = w_0^2\left[1+\left(\frac{\lambda x}{\pi w_0^2}\right)^2\right]$, where the diameter equals $2w$

Therefore, the necessary attributes for light propagating in a medium are:

1. The **refractive index** of the medium
2. The **initial polarization** with which the light beam enters the medium
3. The **initial phase** that the light beam enters the medium with (zero at the beginning)
4. The **wavelength** or **frequency** of the entering light beam
5. The **direction** of propagation at the point where the light enters the medium. The orientation is that of the light propagation vector
6. The **initial beam diameter** with which the beam enters the medium
7. The **permeability** of the medium if it was not a dielectric

3.2 Continuous Laser Sources

We have selected laser sources to be the origins of light beams within the optical systems we consider. Therefore, initial characteristics and parameters of light are determined by these sources.

The following attributes are needed for continuous lasers:

1. **Positions P1 and P2,** the reference points (Figure 3.2) of the component
2. **Direction** of the base of the component if the component was box shaped (Figure 3.2)
3. **Orientation** of the normal to the base of the component if the component was box shaped.
4. **Beam diameter** at the origin ($2w_0$ in (mm))
5. **Power** at the laser in watts
6. **Frequency** of the emitted beam in Hertz. Wavelength (nm) could be computed using the frequency
7. **Longitudinal mode spacing** of the laser source. It determines the discrete frequencies that the device can generate.
8. **Dimensions** of the source (cm)
    - Length, width, and height for box shaped sources
    - Diameter and length for cylindrical sources
9. The **plane polarization ratio** is needed if the output beam is *plane polarized.* Some manufactured laser sources produce plane polarized light. The

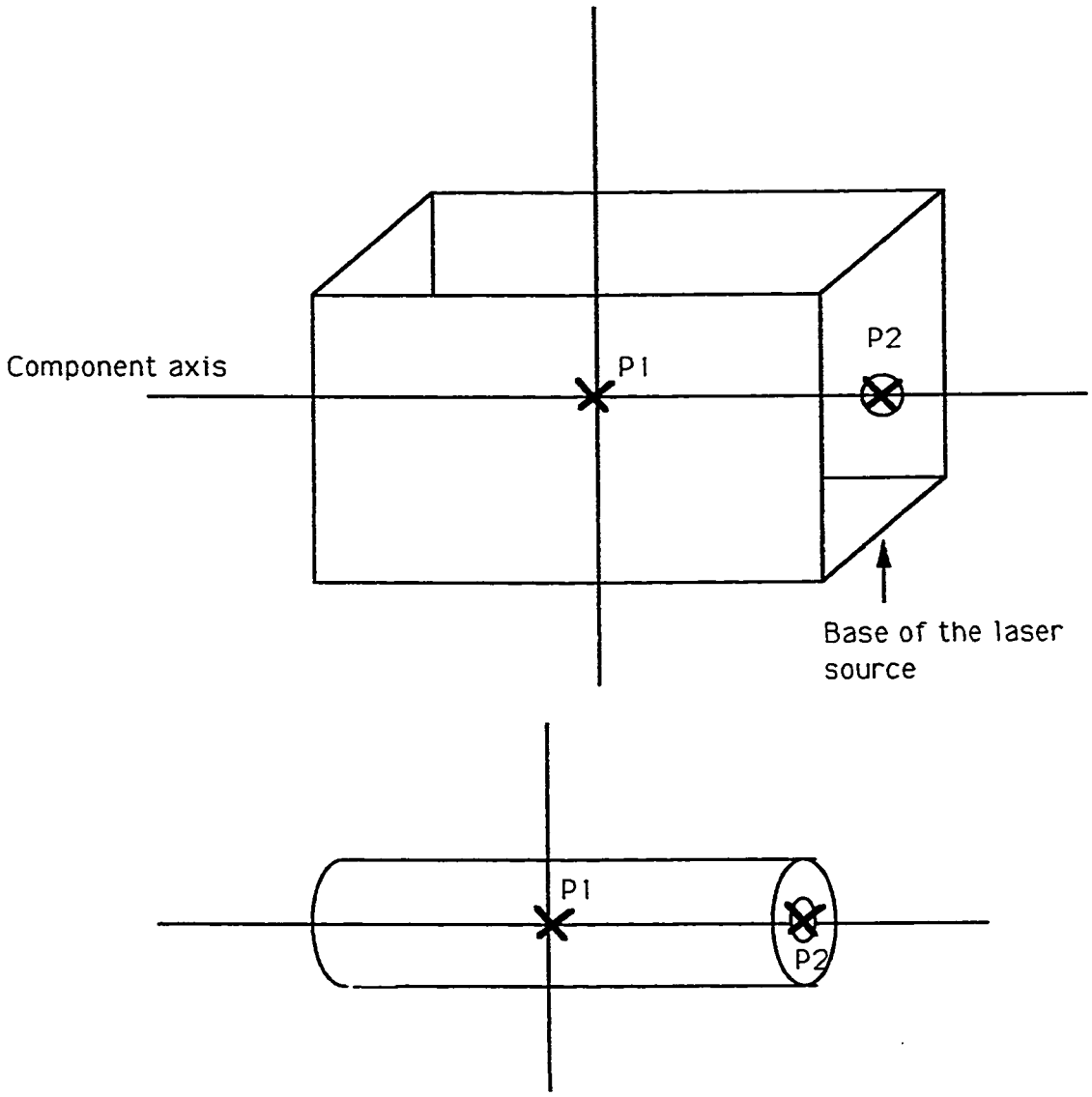


Figure 3.2 Reference points and axis of continuous laser sources

plane of polarization is parallel to the base of the component. Manufacturers give that in terms of *polarization ratio*. If the beam is polarized, this ratio will be 1 : 500 usually. The ratio is that between the two orthogonal polarization components. The 500 according to this definition is for the component parallel to the base of the laser source

3.3 Pulsed Laser Sources

The output of these components is light pulses with a rate and a duration. Thus, there is no continuity of laser emission. The pulsed laser sources tend to have larger dimensions than those of their continuous counterparts. One other difference between them is that the output beam is not circular, it is typically rectangular in cross section.

The beam diverges as it propagates. The width-divergence (θ_{width}), is the angle of divergence in the direction of width. The length-divergence (θ_{length}), is the angle of divergence in the direction of length (Figure 3.3).

Henceforth, the following attributes are needed:

1. **Positions P1 and P2** of the reference points (Figure 3.4)
2. **Direction** of the component base (Figure 3.4)
3. **Orientation** of the normal to the base of the component
4. **Beam cross section dimensions** at the source (length and width) in (mm)
5. **Power** in (watts) at the source
6. **Frequency** in (Hertz) at the source

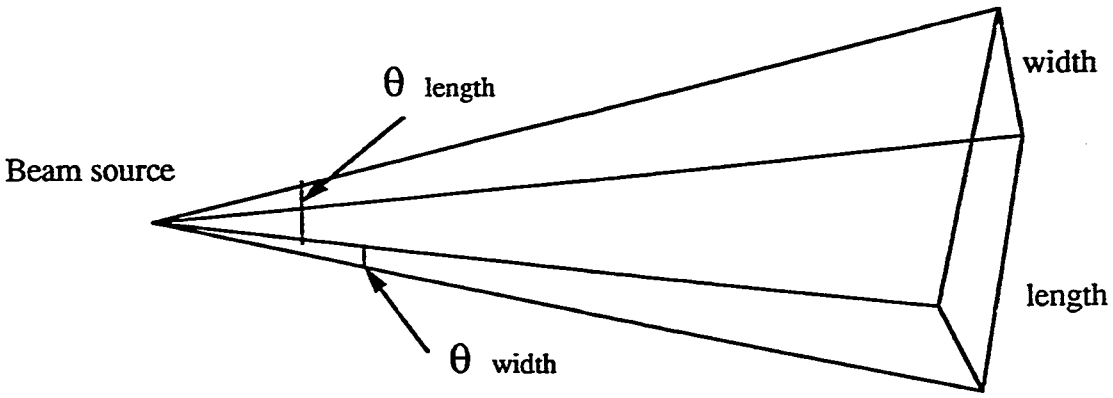


Figure 3.3 The beam divergence in pulsed lasers

7. **Dimensions** of the source (length, width, and height) in (cm)

8. **Plane polarization ratio**, if any, as described in the previous section

9. **Pulse rate** in (pulses per second (pps))

10. **Pulse duration** in (ns)

3.4 Beam Splitters

Beam splitters, as discussed in Chapter 2, split the input beam into two output beams propagating in two different directions. In the following we discuss models for two types of beam splitters.

3.4.1 Plate Beam Splitters

With this component, part of the incident light beam is reflected while the other part is transmitted (Figure 3.5). We take the information about the ratios of reflectance and transmittance as they are provided by manufacturers or by calibration reports. The reflectance and transmittance should be given for all possible angles of incidence.

Reflection under certain conditions causes phase shifts. Therefore, if the beam splitter was coated for stronger reflection or anti-reflection, the index of refraction of the coating material or its permeability have to be provided.

The attributes necessary to describe the function of this component are:

1. **Position P** of its center, regardless of its shape (They could be circularly or squarely shaped

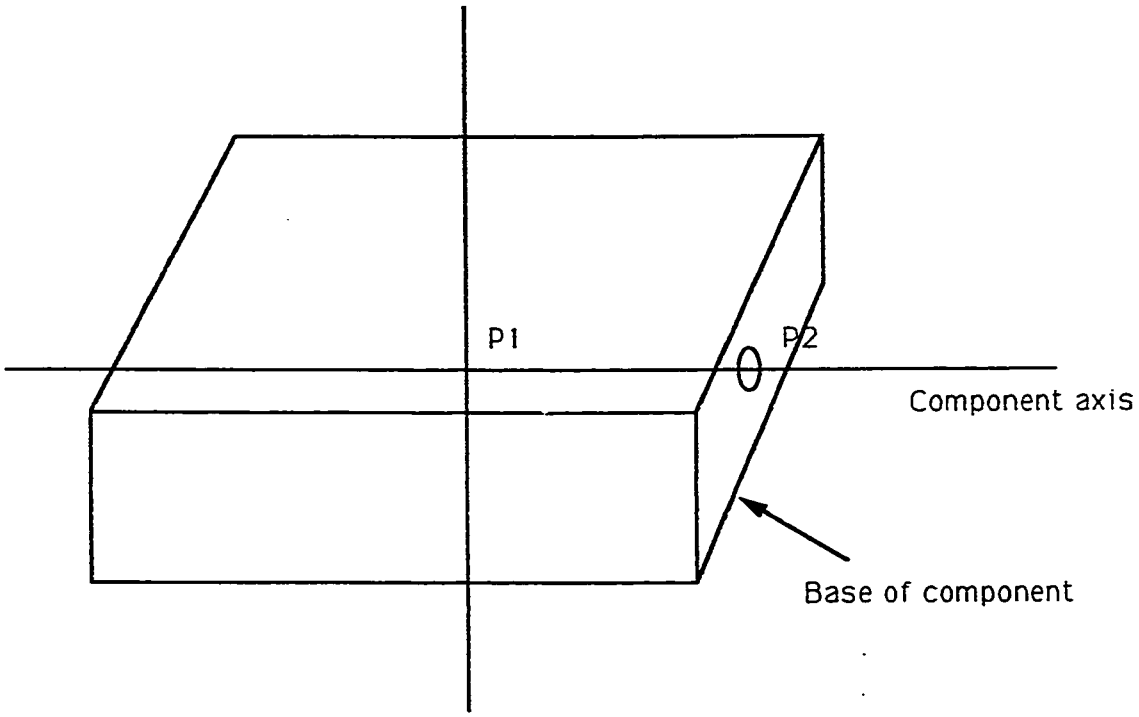


Figure 3.4 Reference points and axis of pulsed laser sources

2. **Face of incidence** direction

3. **Orientation** of the normal to the face of incidence

4. **Dimensions** (side or diameter) of the plate in (cm)

5. **Refractive index of the plate**, used for the computation of displacement of the incident beam, and **permeability** of the substrate of the component

6. **Refractive index of the coating layer on the incidence face.** Its **permeability** is also needed

7. **Thickness** of the plate, used for the displacement computation

8. The output beams **ratios of reflectance (R)** and **transmittance (T)** powers to the incident beam power. These ratios are taken from calibration reports. Usually, diagrams like the ones in Figure 3.6 show these ratios

3.4.2 Cube Beam Splitters

The rule of partial reflection-transmission still applies for this component (Figure 3.7). Again diagrams like the ones in Figure 3.8 are provided for each product or model.

The attributes for this component are:

1. **Position P** of its center

2. **Direction** of the inner face of incidence. The face which reflects the incident beam.

3. **Orientation** of the normal to the inner face of incidence

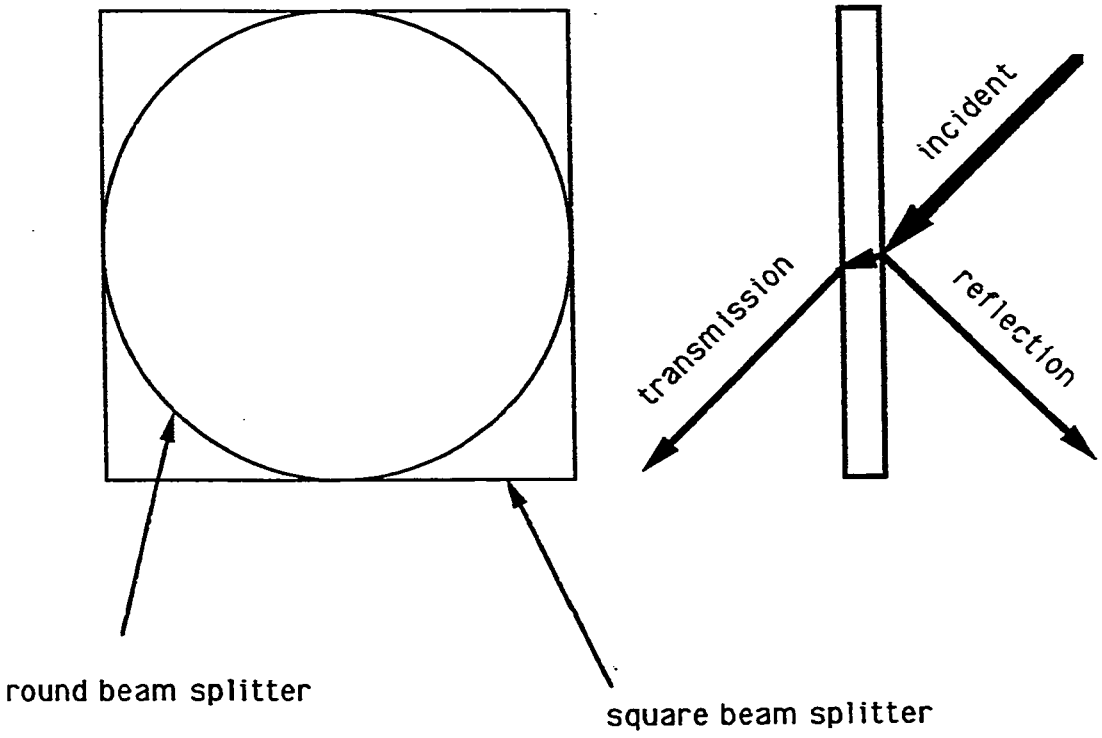


Figure 3.5 Plate beam splitter

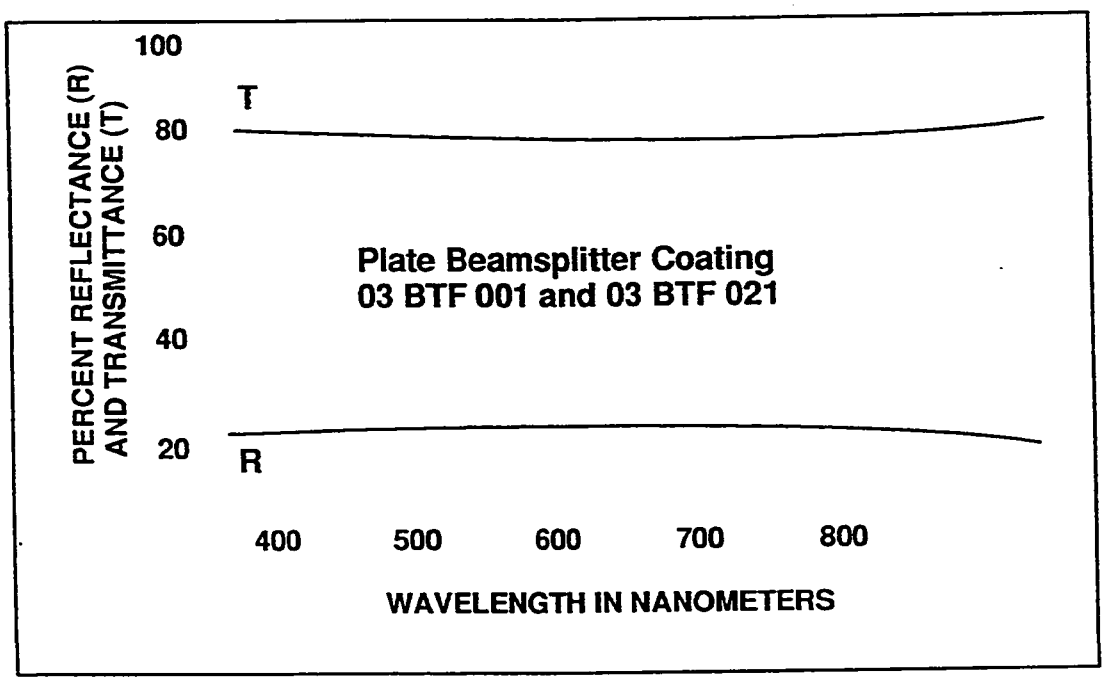


FIG. 3.6 (a) Transmition T and reflectance R of a plate beam splitter

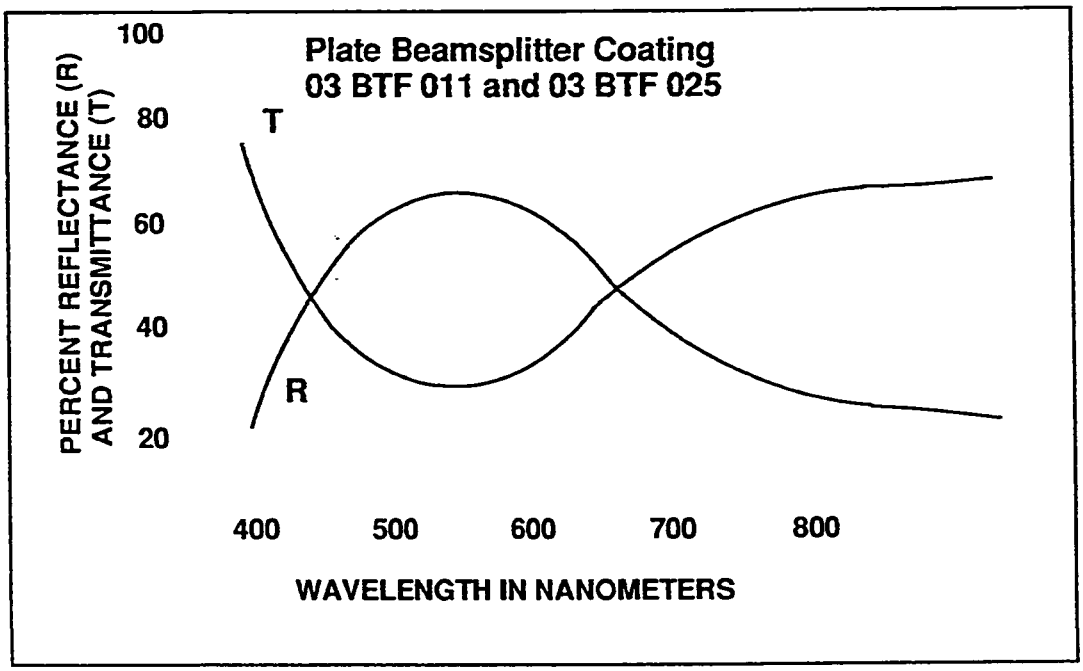


FIG. 3.6 (b) Transmition T and reflectance R for a plate beam splitter

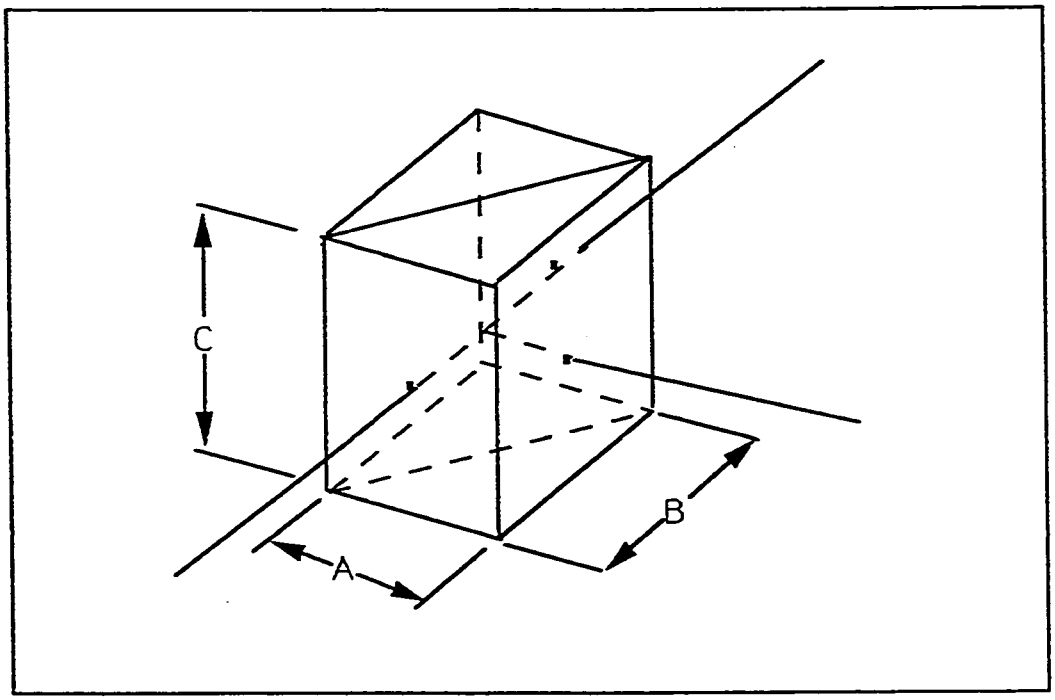


Figure 3.7 Cube beam splitter

4. **Dimensions** A, B, and C in (cm), see Figure 3.7

5. **Refractive index of the cube and permeability** if the material of the cube was not a dielectric

6. **Refractive index of the coating layer at the incident face.** Its permeability is also needed

7. In the case of polarization-insensitive beam splitters, the **reflectance** (R_{avg}) and **transmittance** (T_{avg}) **ratios**

8. **Reflectance R1** of the outer face of incidence, used for non-normal incidence

3.5 Flat Mirrors

Mirrors are basically used to reflect light. Old models used to suffer from losses of part of the incident power in the form of transmission or absorption. To overcome these losses, many reflection coatings are being used. We now have mirrors with 99% reflectance ratio. This is the type of mirror we will be dealing with in this section (Figure 3.9).

Diagrams such as the ones shown in Figure 3.10 are usually provided by manufacturers or calibration reports. Each curve represents the relation between the wavelength of the incident beam and the reflectance ratio for different coating materials. The attributes for these components are:

1. **Position P** at the center of the mirror (circular or square)

2. **Direction** of the face of incidence

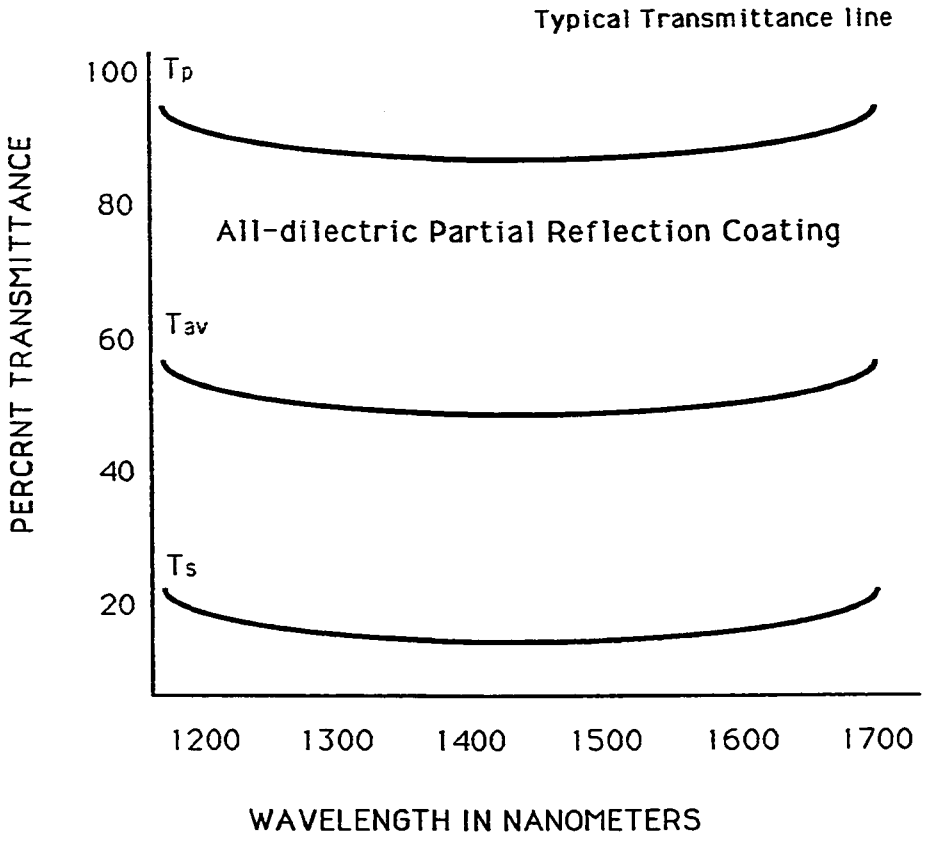


Figure 3.8 Transmittance of a cube beam splitter

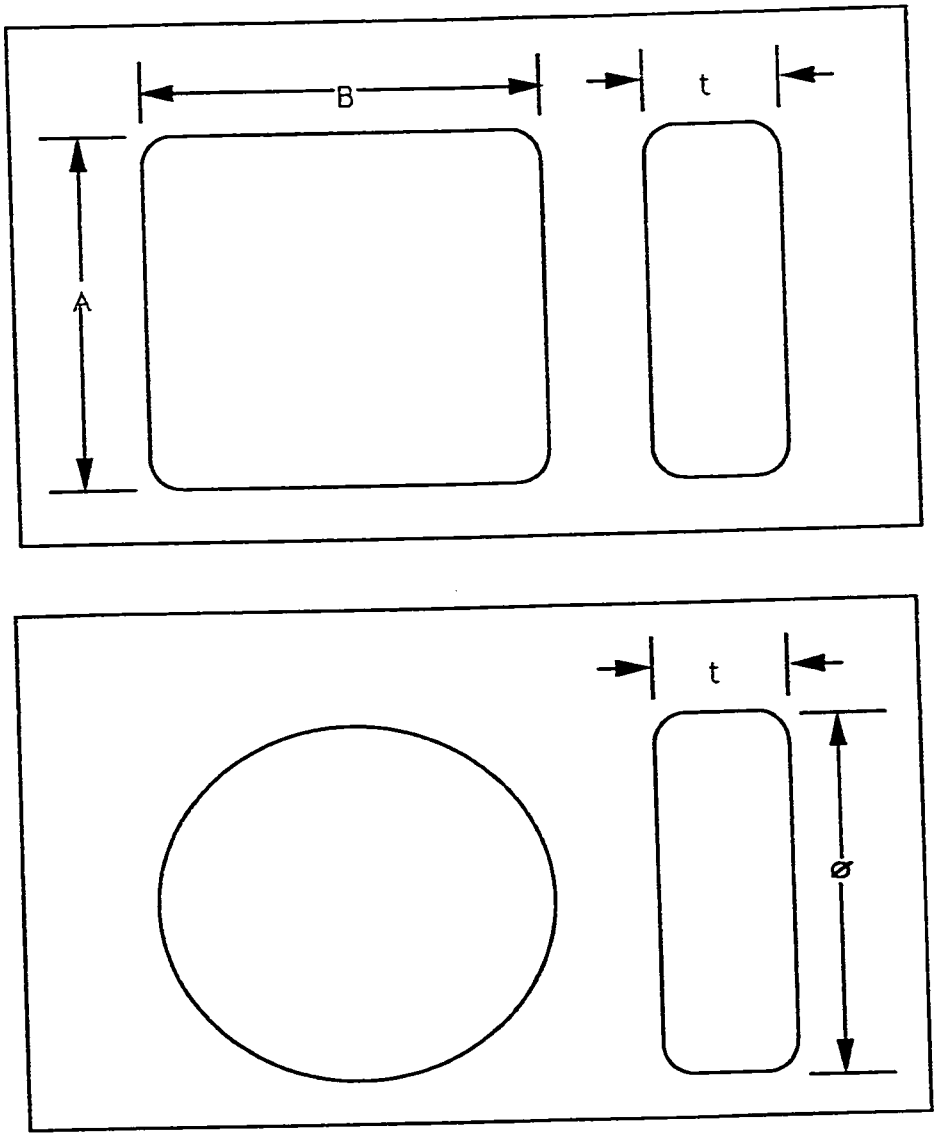


Figure 3.9 Flat mirrors

3. **Orientation** of the normal to the face of incidence

4. **Index of refraction of the first layer of coating.** Its permeability is also needed

5. **Dimensions** (side/diameter) of the component

6. **Reflectance R** of the mirror

7. **Thickness** of the mirror for the displacement computations of the transmitted part, if any

3.6 Windows

Windows are used to allow optical radiations to pass from one environment to another without allowing other elements of these environments to interfere. Different models with different dimensions and material are provided by manufacturers.

Diagrams that show the transmittance of these components in relation with the wavelength of the incident beam are similar to the one in Figure 3.11. Part of the incident energy is either reflected or absorbed.

The attributes of this component are:

1. **Position P** of the center of the window

2. **Direction** of the face of incidence

3. **Orientation** of the normal to the face of incidence

4. **Index of refraction of the window substrate** and its **permeability**

5. **Diameter** of the window

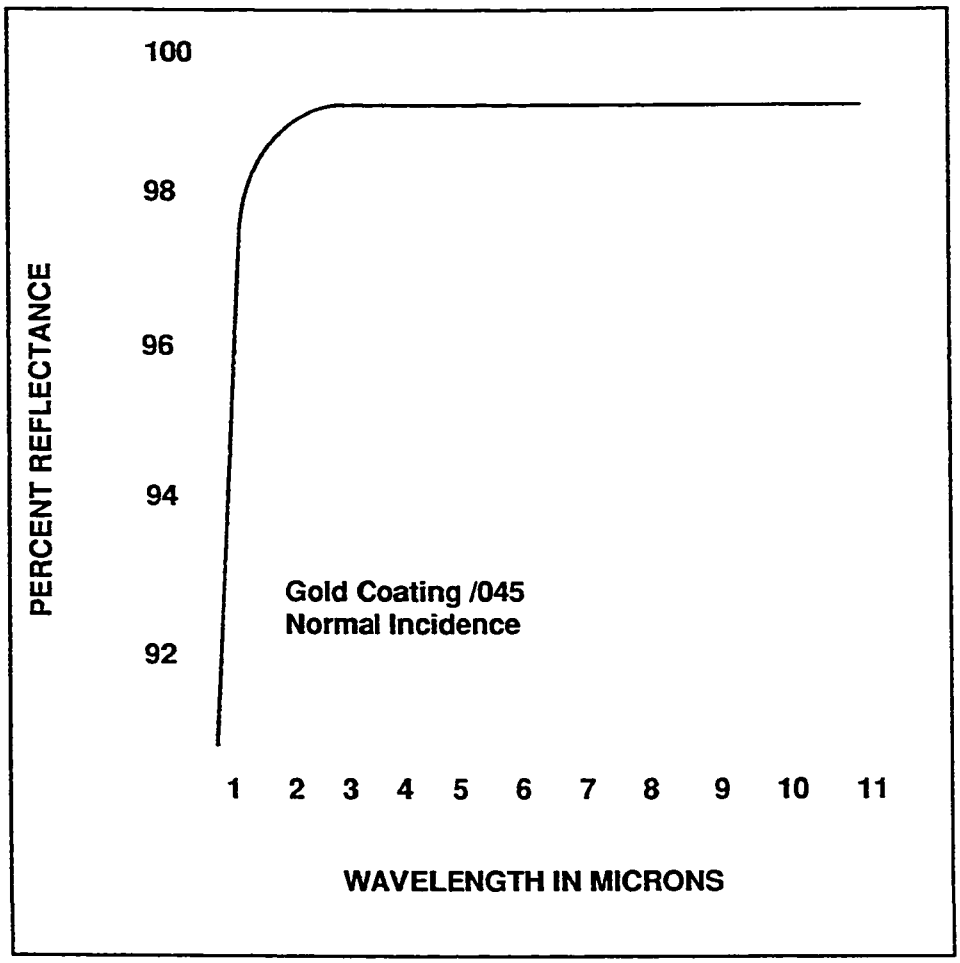


FIG. 3.10 Reflection of flat mirrors with gold coat

6. **Thickness** of the window

7. **The transmittance** T taken from diagrams such as the one in Figure 3.11

3.7 Prisms

A large number of prisms of different shapes and types appear in literature and in manufacturers' catalogues. In optical computing, however, we noticed that only one type is heavily used. That is the **right-angle prism** (Figure 3.12).

This type of prisms has three faces dedicated for incidence. The hypotenuse is used to totally reverse the direction of the incident beam, see Figure 3.12. The other two faces cause the reflection of the incident beam by a 90° angle each (Figure 3.13).

Anti-reflection coating could be used to reduce the reflectance at the incidence faces of the prism.

Attributes of this component are:

1. **Position P** of the component (Figure 3.13)

2. **Direction** of the hypotenuse face

3. **Orientation** of the normal to the hypotenuse face

4. **Dimensions A, B, C** of the component (Figure 3.12) in (cm)

5. **Refractive index of the substrate of the prism** and its **permeability**

6. **Refractive index and permeability of the coating material.**

7. **T1, T2,** and **T3,** transmittances of the three incidence faces corresponding to A, B, and C

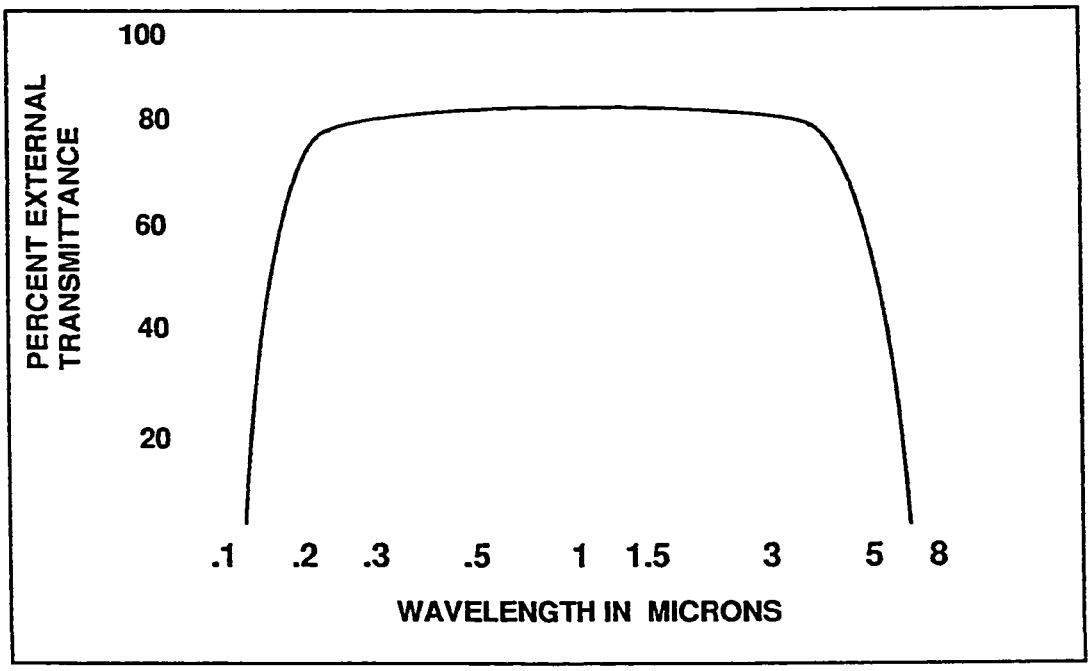


FIG. 3.11 Transmittance of windows

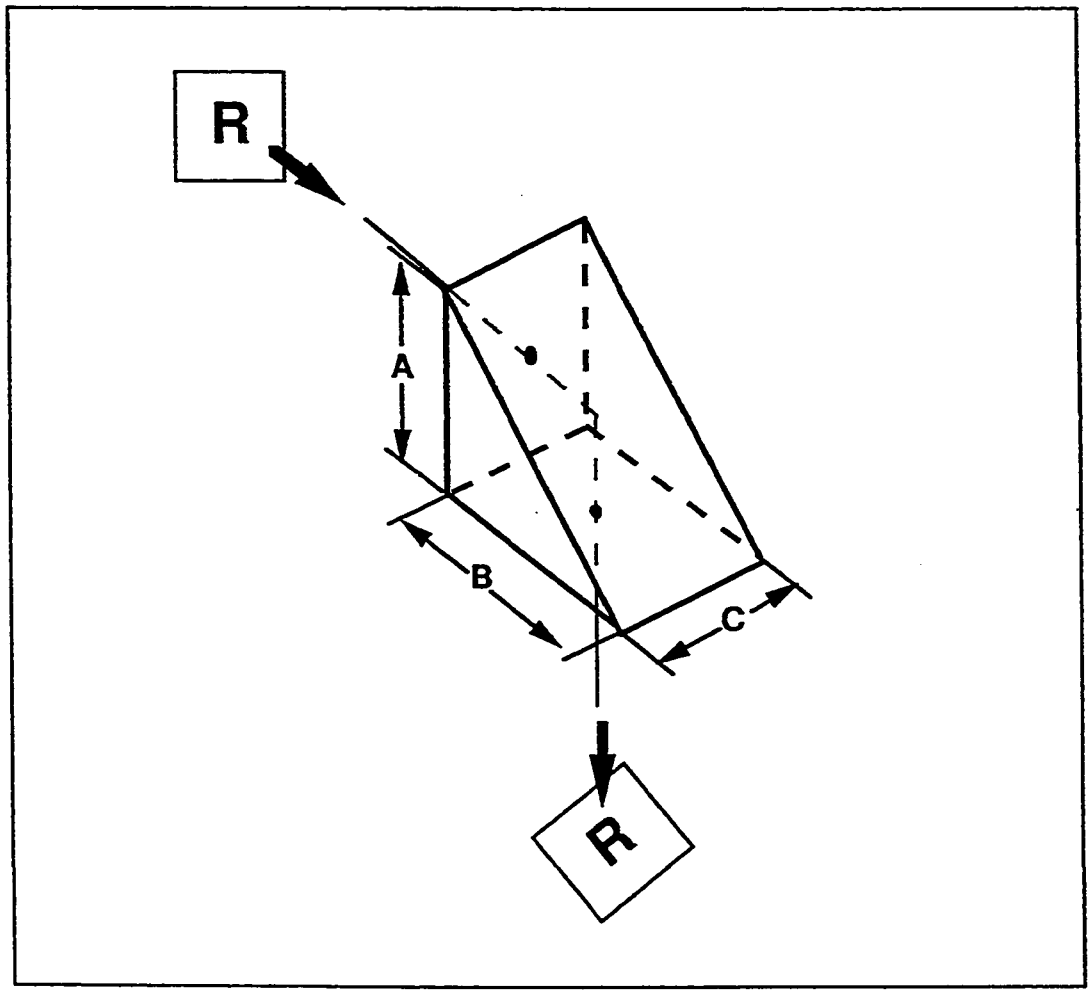


FIG. 3.12 The right-angle prism

8. **R1, R2, and R3,** reflectances of the three faces of the prism

3.8 Retardation Plates

Retardation plates were discussed in Chapter 2. They are of two major types, half-wave plates and quarter-wave plates. Their main function is to alter either the plane or state of polarization of the incident light beams (See Figure 3.14). Anti-reflection coats could be added to their faces of incidence.

The attributes of these components are common and they are:

1. **Position** of the center of the circularly shaped component
2. **Direction** of the face of incidence
3. **Orientation** of the normal to the face of incidence
4. **Orientation** of the optic axis of the plate
5. **Diameter** of the plate
6. **Refractive index of the material of the plate** and its **permeability**
7. **Type of the plate,** quarter-wave or half-wave plate
8. **Refractive index and permeability of the coating material,**
9. **Thickness** of the plate
10. **The ratio of transmittance T to the incident power.** This ratio is related to the incident beam wavelength. It is available in calibration reports

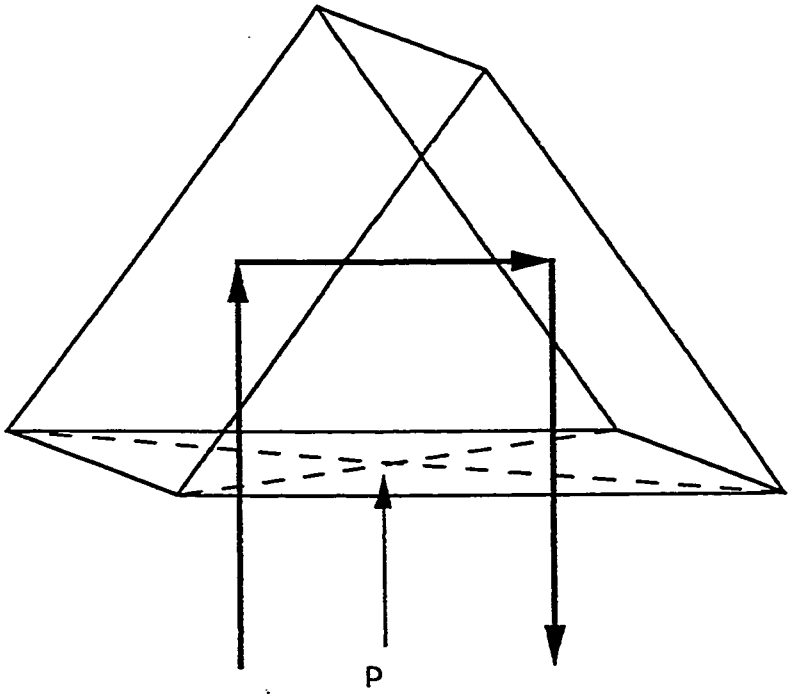


Figure 3.13 Total internal reflection in right-angle prisms

3.9 Polarizers

3.9.1 Birefringent Polarizers

The concept of birefringence was discussed in Chapter 2. Many crystalline materials were found to have this feature. However, in reality, three crystals were formally developed and they are discussed below.

Glan-Taylor Polarizing Prism

The output of this component is the *extraordinary ray*, one of the two rays that result from birefringence. The extraordinary ray will have a plane of polarization normal to the plane of incidence in the case of this component (Figure 3.15). The *ordinary ray* is reflected at the hypotenuse and totally absorbed by the mounting material. The output extraordinary beam does not deviate in direction. Normal incidence is required for best results.

Glan-Thompson Polarizing Prisms

In these prisms, only the output extraordinary beam will have a polarization plane parallel to the plane of incidence. The ordinary ray is again totally absorbed. The direction of the output beam does not deviate from the direction of the incident one (See Figure 3.16). Normal incidence is again required.

For the previous two components, the following attributes are defined:

1. **Position P** of the center of the component axis under consideration here has a similar direction to the incident beams in Figures 3.15 and 3.16

2. **Dimensions L, ϕ, x, and y** as shown in Figures 3.15 and 3.16

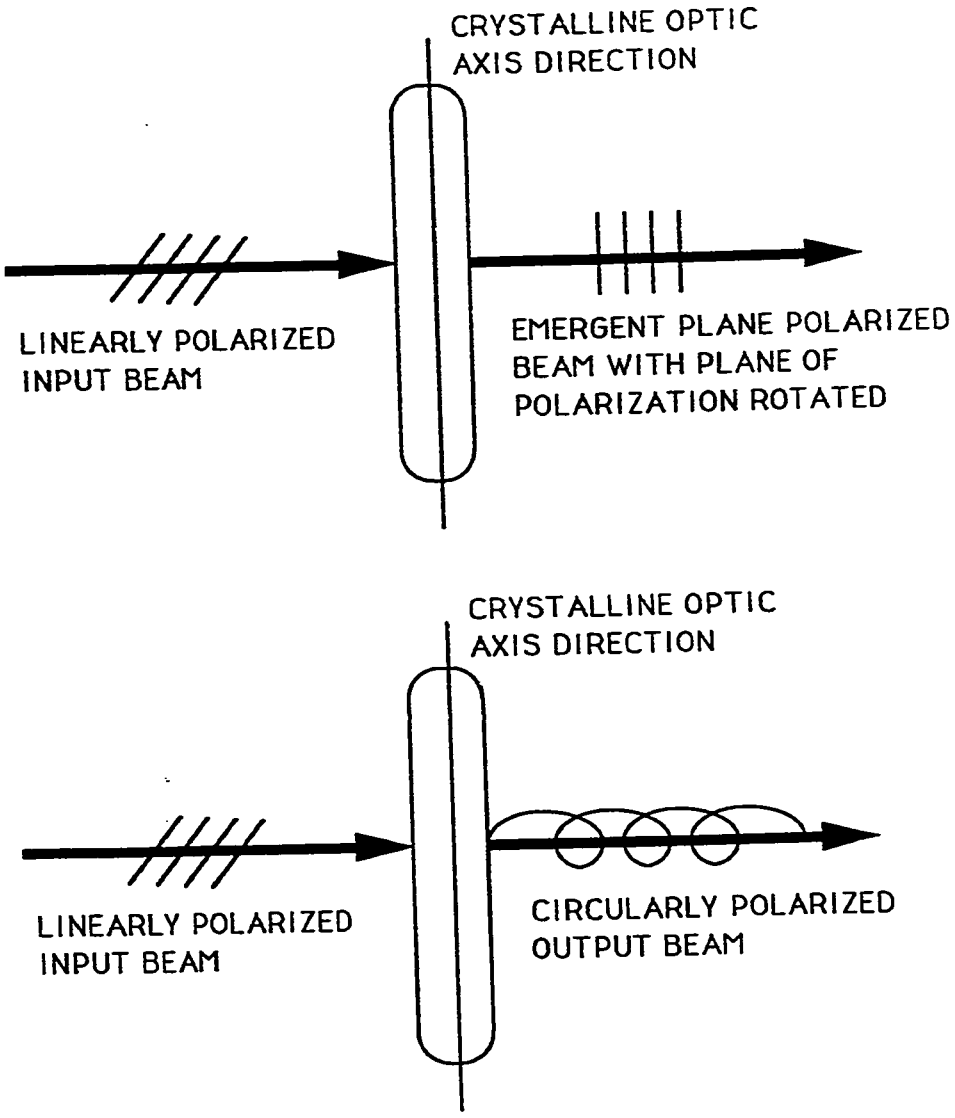


Figure 3.14 (a) Half-wave plates
(b) Quarter-wave plates

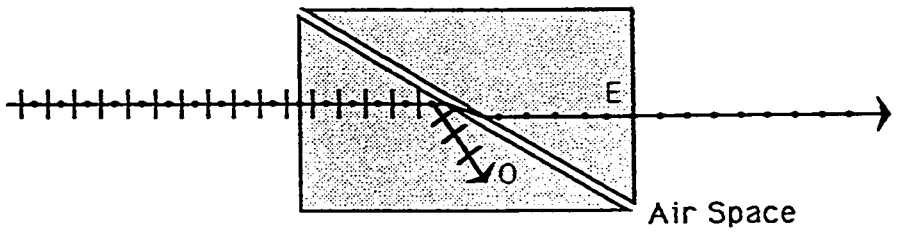


Figure 3.15 Glan polarizing prism

3. **Direction** of the face of incidence

4. **Orientation** of the normal to the face of incidence

5. **Type of the component,** Glan-Taylor or Glan-Thompson

6. **The refractive index n_e, for the extraordinary ray**

7. **Transmittance T** of the component taken from the results of the calibration process

8. If the component was coated, then the **reflectance R** at the planes of incidence is needed. The **refractive index and permeability of the coating material** is also required

3.9.2 Polarizing Beam Splitters

These components are commonly used in optical computing. They are very efficient in the sense that they do not loose much energy as was the case with the previously discussed polarizers. Figure 3.17 shows the way they operate. Notice what happens if plane polarized light was the input of such a component.

The attributes are:

1. **Position P** of the center of the component

2. **Direction** of the inner face of incidence

3. **Orientation** of the normal to the inner face of incidence

4. **Dimensions** (length, width, and height) of the component

5. **Refractive index of the substrate of the component**

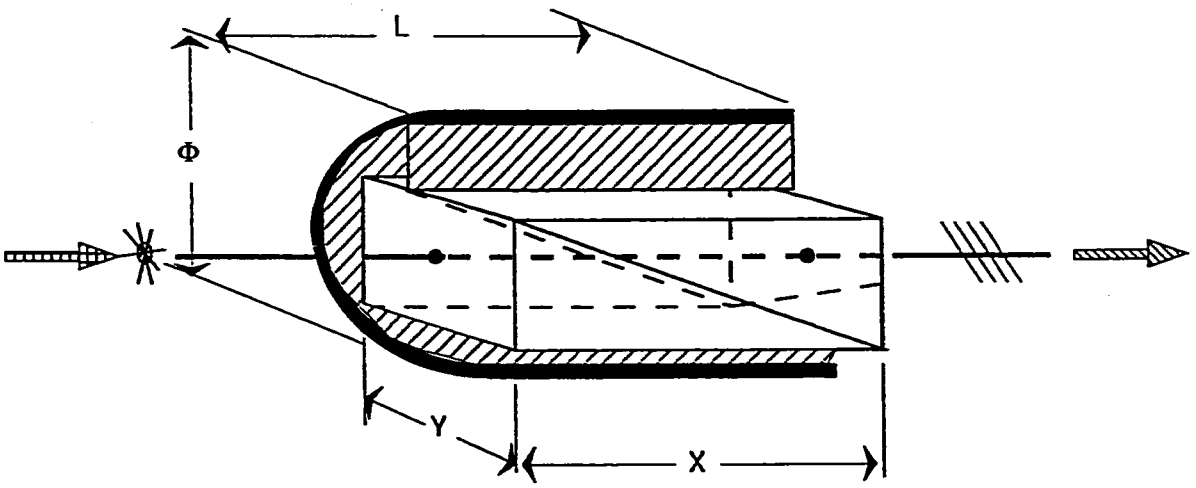


Figure 3.16 Glan-Thompson prism

6. **Transmittances T1 and T2** of both the ordinary and extraordinary, consequently, output beams

7. **Reflectance R** over the outer face of incidence, for non-normal incidence

8. If the component was coated, the **refractive index of the coating material** and its **permeability** are needed

3.9.3 Dichroic Sheet Polarizers

With this polarizer, the optic axis, also called the *Polarization axis*, of the sheet determines the orientation of the plane of polarization of the output beam. It will be parallel to the optic axis of the sheet (Figure 3.18a). Only the part of the incident beam with polarization plane parallel to the sheet optic axis gets transmitted. The other parts are totally absorbed by the sheet's material.

If the input beam was plane polarized, the output beam will have a polarization plane parallel to the optic axis as well. The transmittance of the sheet is computed using the following formula:

$$T = K_1 \cos^2(\theta) + K_2 \sin^2(\theta) \tag{3.1}$$

where, K_1 and K_2 are the principal transmittances of the sheet. They are functions of wavelength given by diagrams such as the one in Figure 3.18b. The angle between the plane of polarization of the incident beam and the optic axis of the sheet, is θ.

The attributes of the component are:

1. **Position P** of the center of the circularly shaped polarizer

2. **Orientation** of the optic axis (polarizing axis) of the component

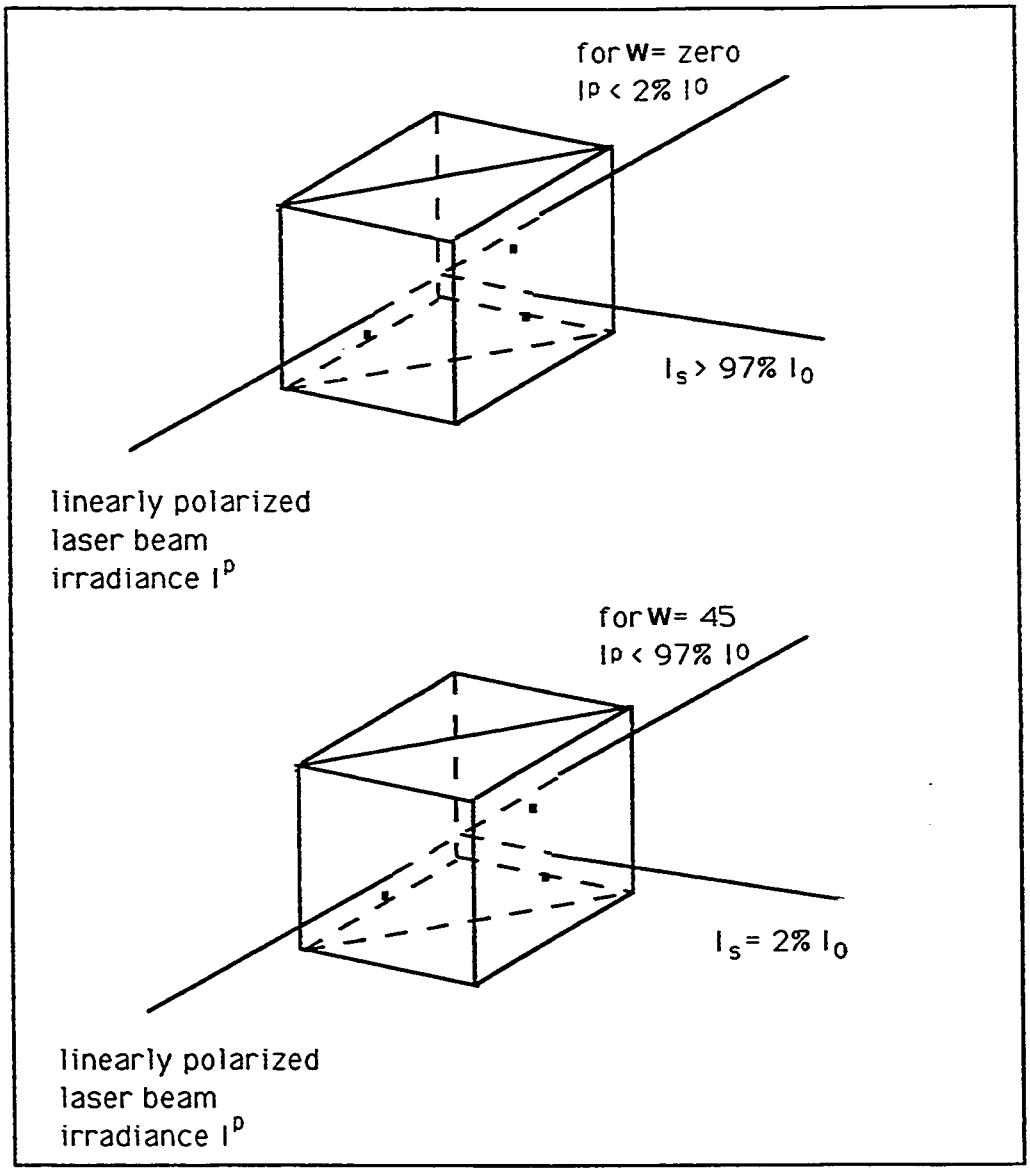


Figure 3.17 Polarizing beam splitters

3. **Direction** of the face of incidence

4. **Orientation** of the normal to the face of incidence

5. **Diameter** of the sheet

6. **Thickness** of the sheet

7. **Refractive index of the substrate of the component**

8. **The principal transmittances** K_1 **and** K_2 of the sheet

9. **Transmittance T** of the sheet if the input light is not polarized

10. If the input beam was not plane polarized, **transmittance T** is required

11. If the sheet is coated, then the **refractive index of the coating material** and its **permeability** are needed

3.10 Spherical Lenses

Different types of spherical lenses are provided by different manufacturers. They are mainly classified as **convex** and **concave** lenses. See Figure 3.19.

In order to trace the light beam through lenses, we use **John's matrices**. The John's matrices for lenses look as follows:

$$\begin{pmatrix} r_{out} \\ u_{out} \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} r_{in} \\ u_{in} \end{pmatrix} \tag{3.2}$$

In the above, r is the distance between the point where the beam enters the lens and the center of the lens (it is negative when talking about the lower half of the lens). u is the slope of the beam with respect to the normal to the lens axis from

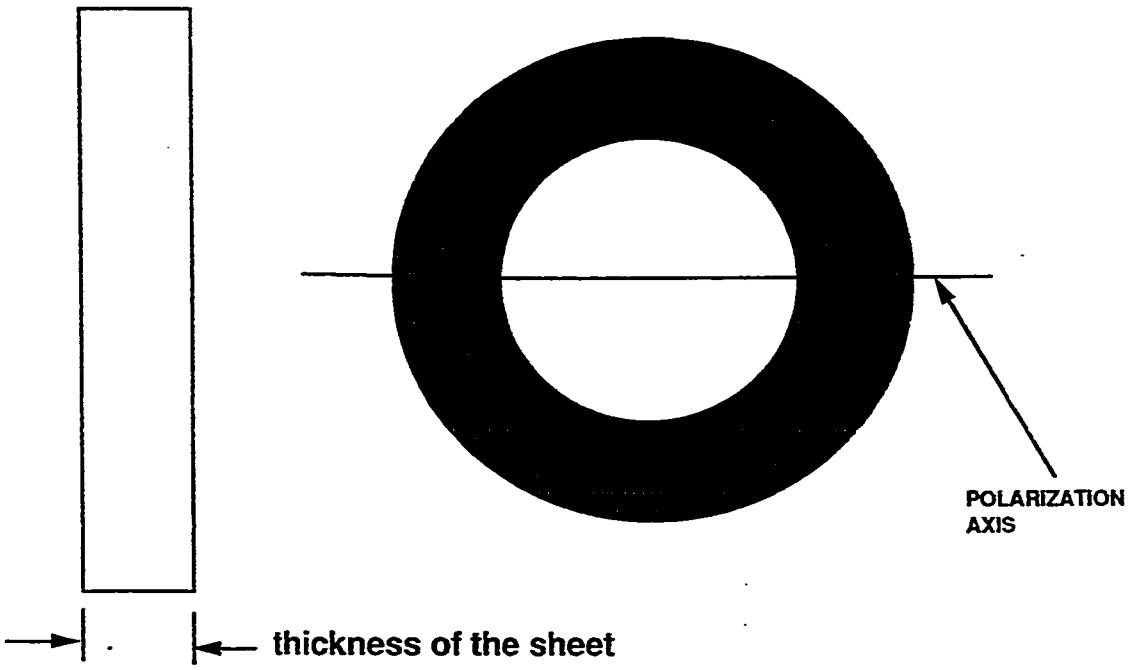


FIG. 3.18: (a) Dichroic Sheet polarizer

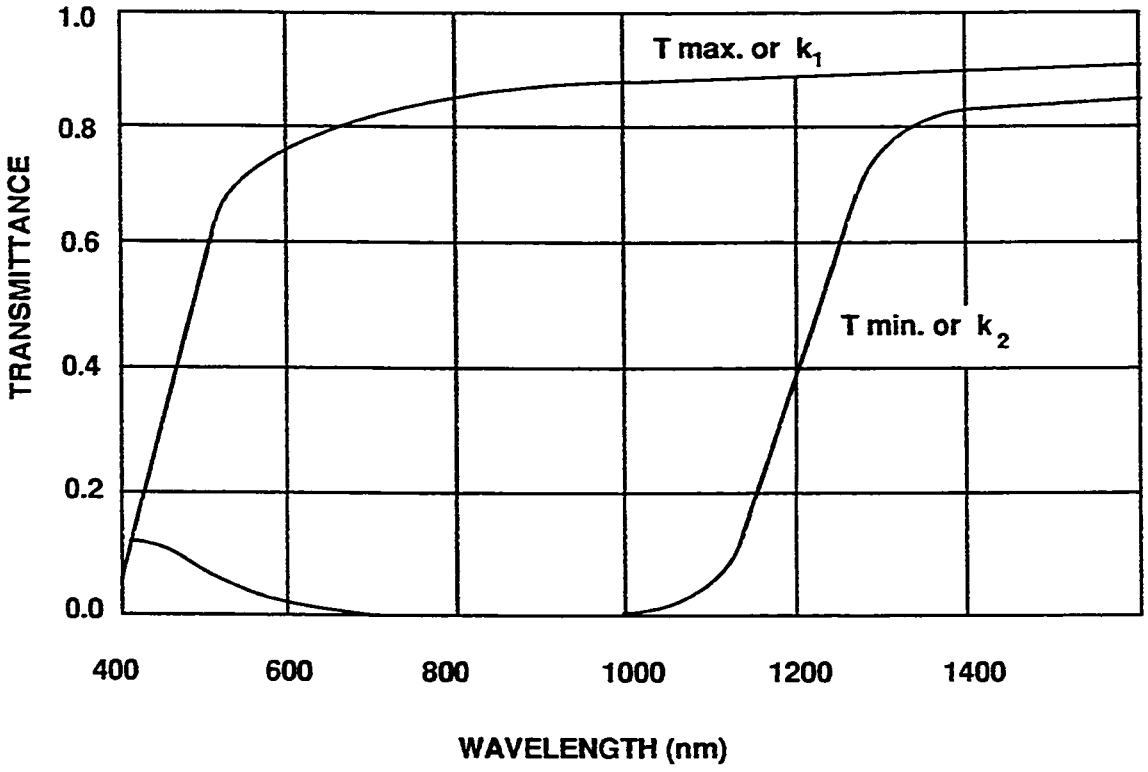


Figure 3.18 (b) The principal transmittances K_1 and K_2 as functions of wavelength

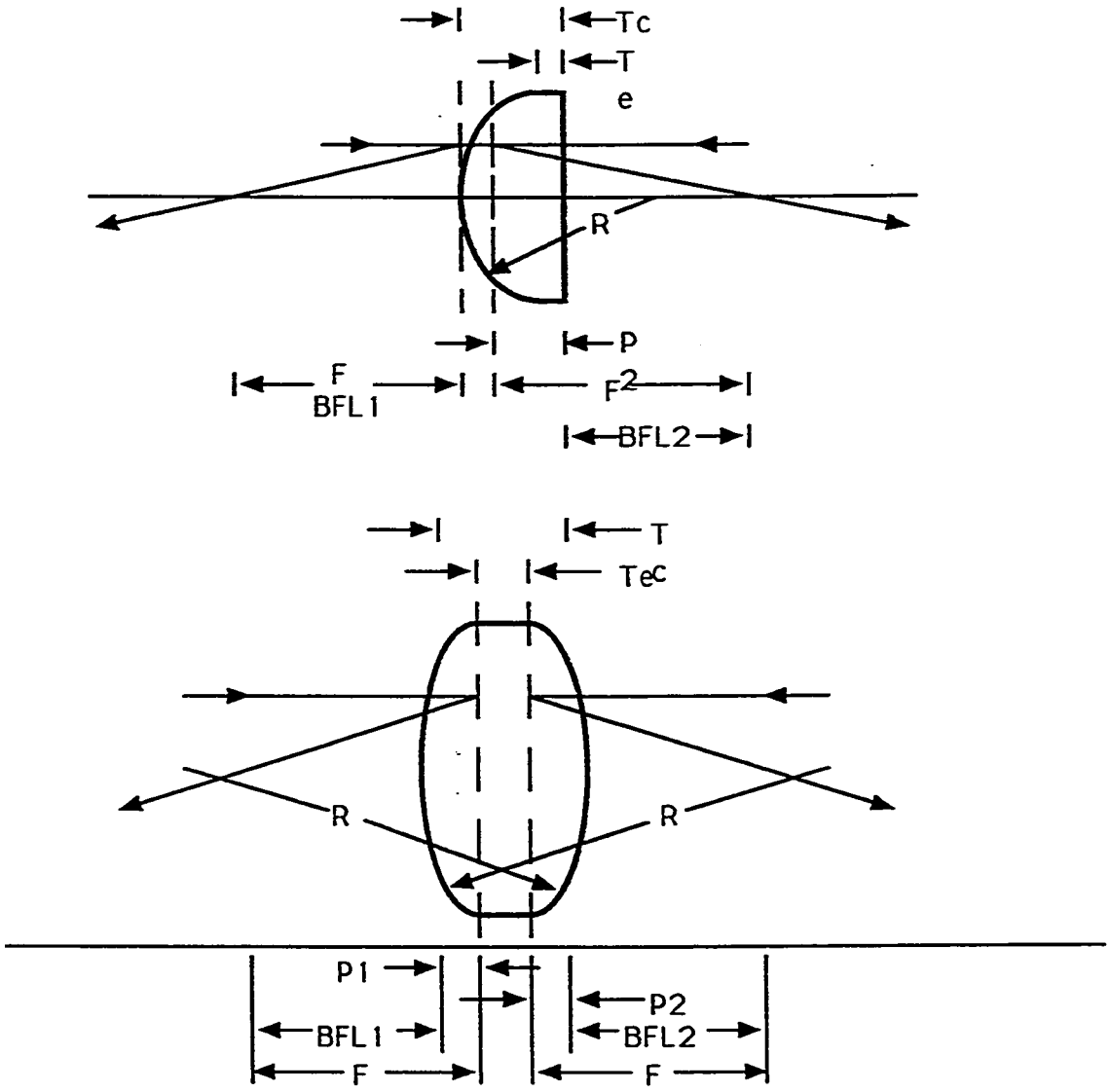


Figure 3.19 (a) Plano-convex and bi-convex spherical lenses

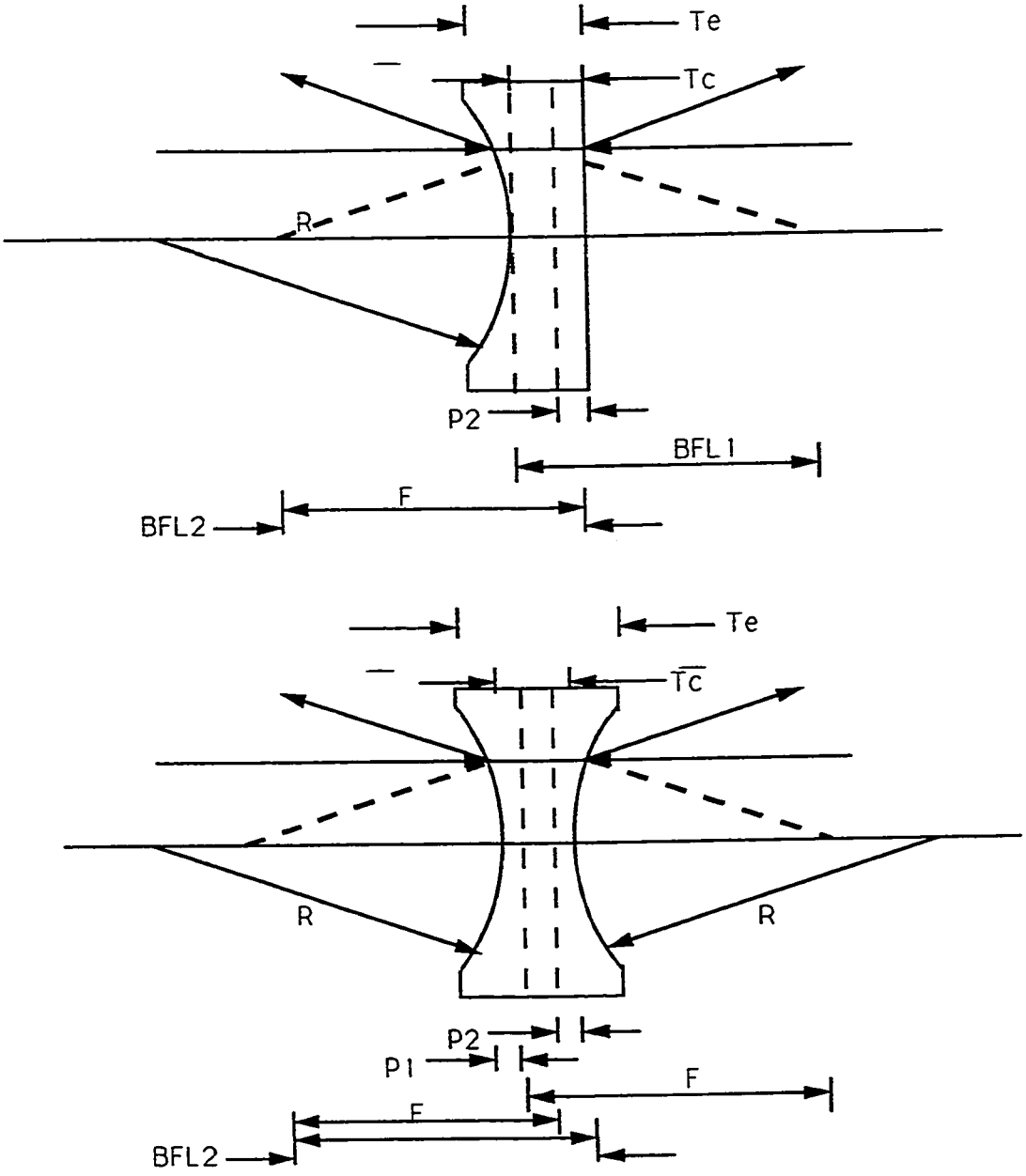


Figure 3.19 (b) plano-concave and bi-concave spherical lenses

the incidence face side. The subscripts **in** and **out** are used to refer to the input and output beams, consequently (Figure 3.20).

In the case of bi-convex lenses, the values for A,B,C, and D are computed as follows:

$$A = 1 - \frac{t(n-1)}{R_1 n} \tag{3.3}$$

$$B = \frac{t}{n} \tag{3.4}$$

$$C = -(n-1)\left(\frac{1}{R_1} - \frac{1}{R_2} + \frac{t(n-1)}{R_1 R_2 n}\right) \tag{3.5}$$

$$D = 1 + t(n-1) \tag{3.6}$$

where n is the refractive index of the substrate of the lens. In the case of plano-convex lenses, R_2 is ∞ and hence, C will be:

$$C = -(n-1)\left(\frac{1}{R_1}\right) = \frac{1-n}{R_1} \tag{3.7}$$

When considering bi-concave lenses, R_1 and R_2 will have opposite signs and A and C will have different values as follows:

$$A = 1 + \frac{t(n-1)}{R_1 n} \tag{3.8}$$

$$C = -(n-1)\left(\frac{1}{R_2} - \frac{1}{R_1} + \frac{t(n-1)}{R_1 R_2 n}\right) \tag{3.9}$$

For the case of plano-concave lenses, R_2 is ∞ and this gives a new formula for the computation of C:

$$C = -(n-1)\left(-\frac{1}{R_1}\right) = \frac{n-1}{R_1} \tag{3.10}$$

The attributes of these components are:

1. **The type of the lens:** plano-convex, bi-convex, plano-concave, or bi-concave

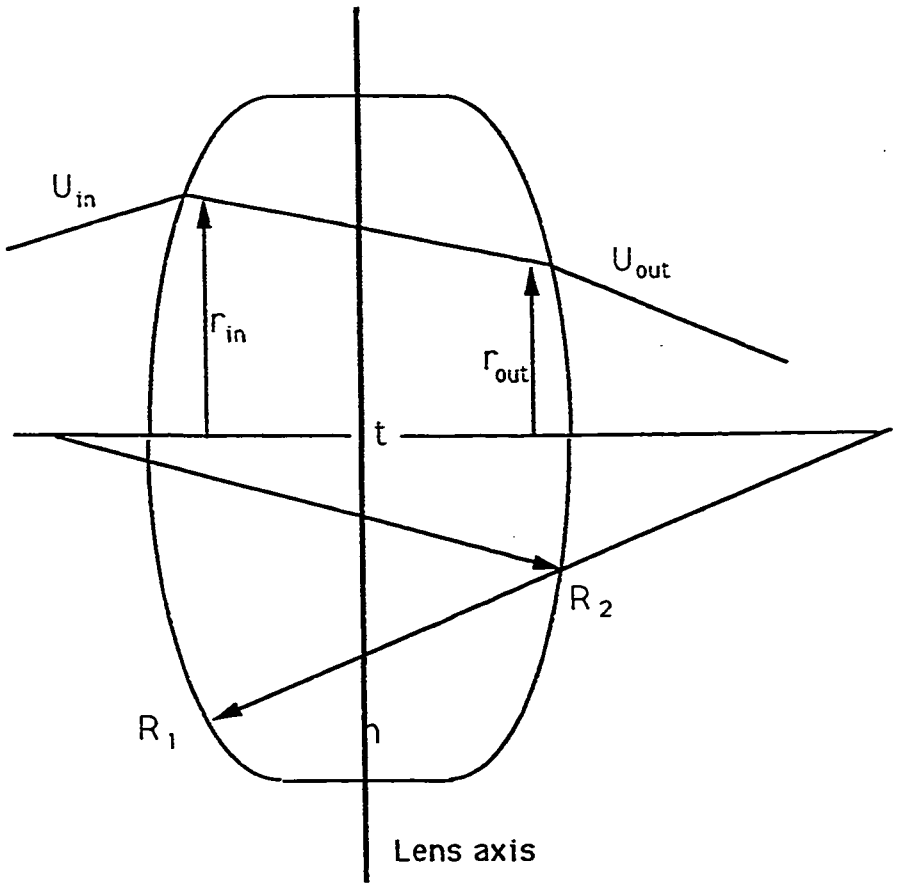


Figure 3.20 Ray tracing in bi-convex lenses

2. **Position P** of the center of the lens

3. **Diameter** of the lens and its **radius of curvature R** or **radie of curveture** R_1 **and** R_2 if the lens is not symmetric

4. **Direction** of the face of incidence

5. **Orientation** of the face of incidence

6. **Thicknesses** t_c **and** t_e of the lens (Figure 3.19)

7. **Refractive index of the lens**

8. **Focal length F** and **back focal lengths BFL1** and **BFL2**

9. **Transmittance T** of the lens

10. If the lens was coated, the **refractive index of the coating material** and its **permeability** are required

3.11 Cylindrical Lenses

Cylindrical lenses have exactly the same attributes as those of spherical lenses had, except that they are rectangularly shaped. Therefore, they have two dimensions, x and y (Figure 3.21). There are neither bi-convex nor bi-concave cylindrical lenses.

Equations 3.1,2,3,5, and 6 apply for ray tracing in plano-convex cylindrical lenses.

Equations 3.1,3,5,7, and 9, on the other hand, apply for ray tracing in plano-concave cylindrical lenses.

Hence, the same attributes of spherical lenses are required for cylindrical ones except the fourth attribute which becomes:

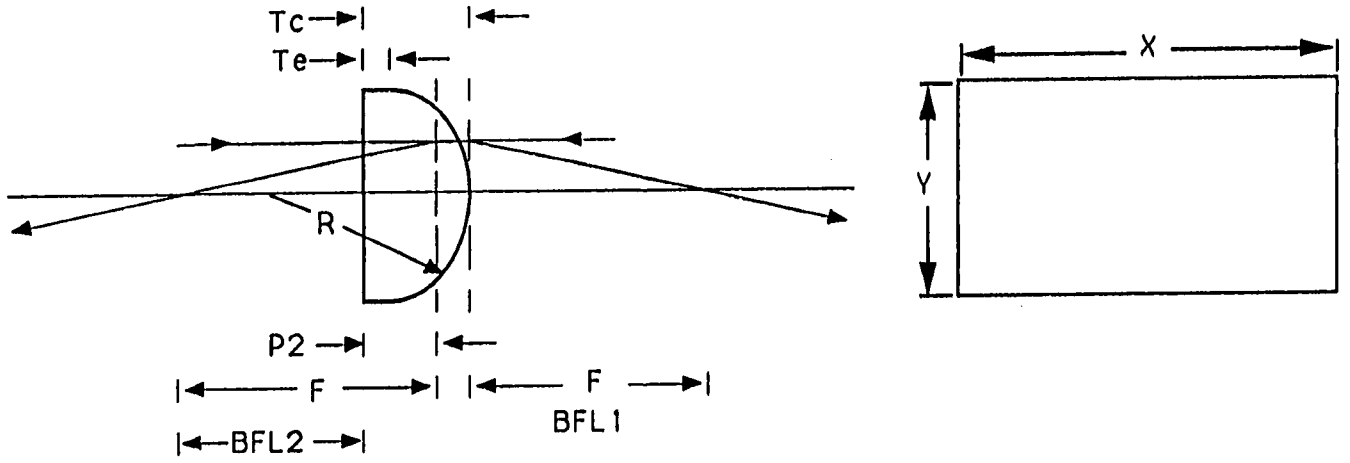


FIG. 3.21 (a) Plano-convex cylindrical lense

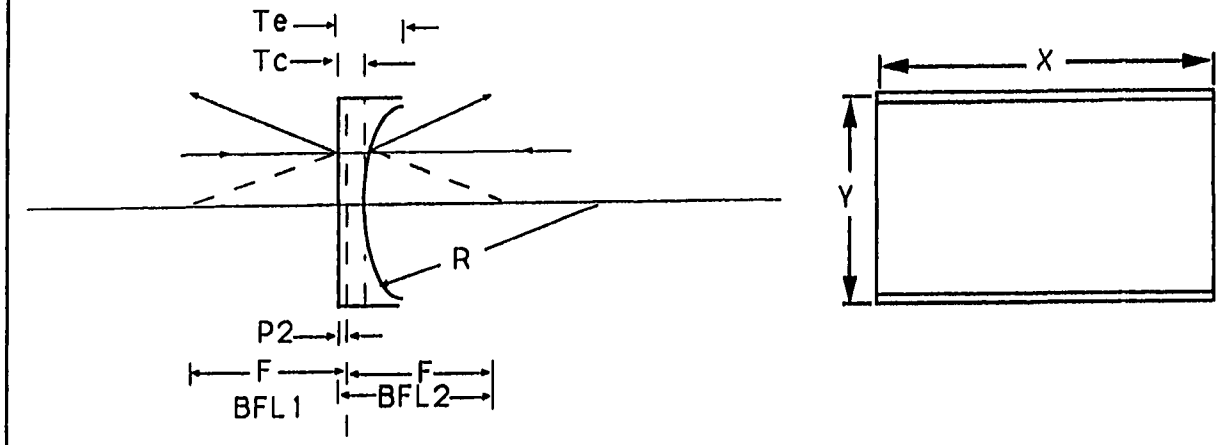


FIG. 3.21 (b) Plano-concave cylindrical lense

4. **Dimensions x and y of the cylinder** and the **radius of curvature R or radie of curveture R_1 and R_2** if the lens is not symmetric

3.12 Diffraction Gratings

Gratings analyse incident light beams to a number of output beams each with different directions and intensities. There are two types of gratings, the *transmission gratings*, with slits to transmit light beams, and *reflection gratings*, with grooves to reflect light beams. See Figures 3.22, 3.23, 3.24, and 3.25.

Some of the output beams interfere constructively in some directions with specific intensities. The other beams interfere destructively causing light beams to have zero or weak intensities.

The grating diffraction equation is:

$$a\ (sin\ \theta_m - sin\ \theta_i) = m\lambda \tag{3.11}$$

m represents the orders for constructive interference, where the *zero* order has the highest intensity, orders ± 1 have similar and weaker intensities, orders ± 2 have similar and further weaker intensities, and so on.

The intensities of the different orders (beams) are computed using the following formula:

$$I(\theta_m) = I_0 \left(\frac{sin\beta}{\beta}\right)^2 \left(\frac{sinN\alpha}{sin\alpha}\right)^2 \tag{3.12}$$

where, $\beta = (kb/2)\ sin\theta_m$ and $\alpha = (ka/2)\ sin\theta_m$. N is the number of slits/grooves that contribute to the flux diffraction (i.e., the slits that got incident with the input light beam). The propagation number, $k = 2\pi/\lambda$.

The reflection gratings have to be dealt with care. That is so because, some reflections might cause phase shifts.

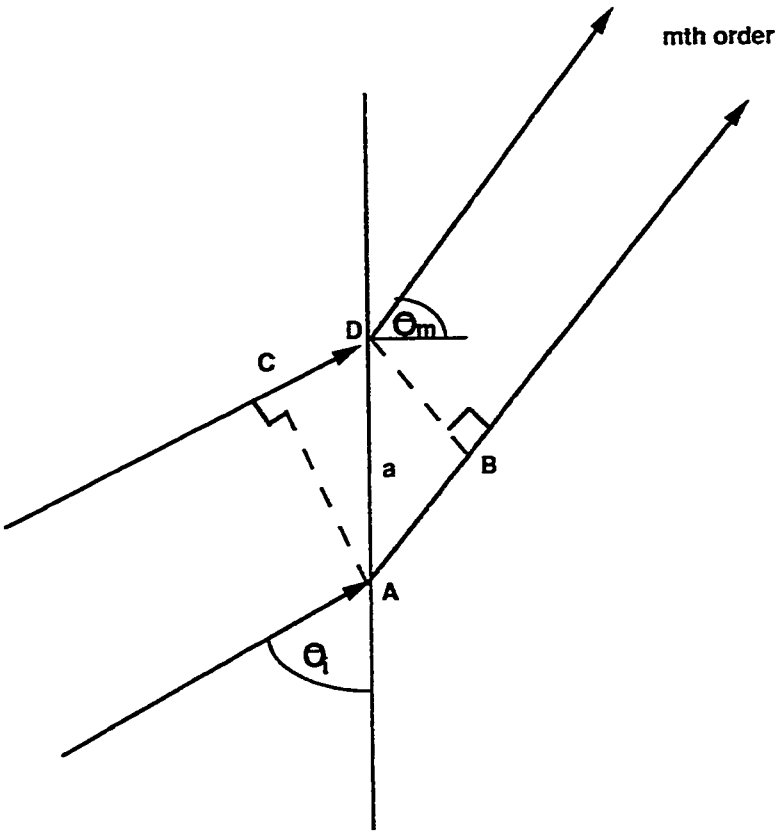
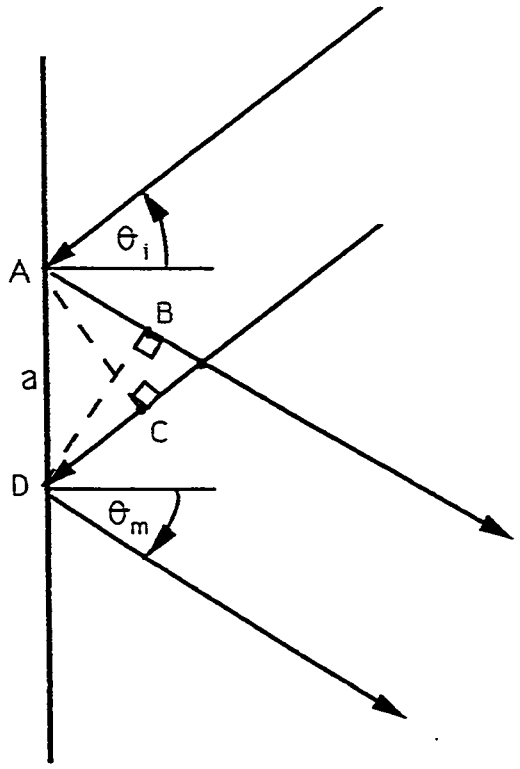


FIG. 3.22 A transmission grating



$$AB - CD = a\,(\sin\theta_m - \sin\theta_i)$$

Figure 3.23 A reflection grating

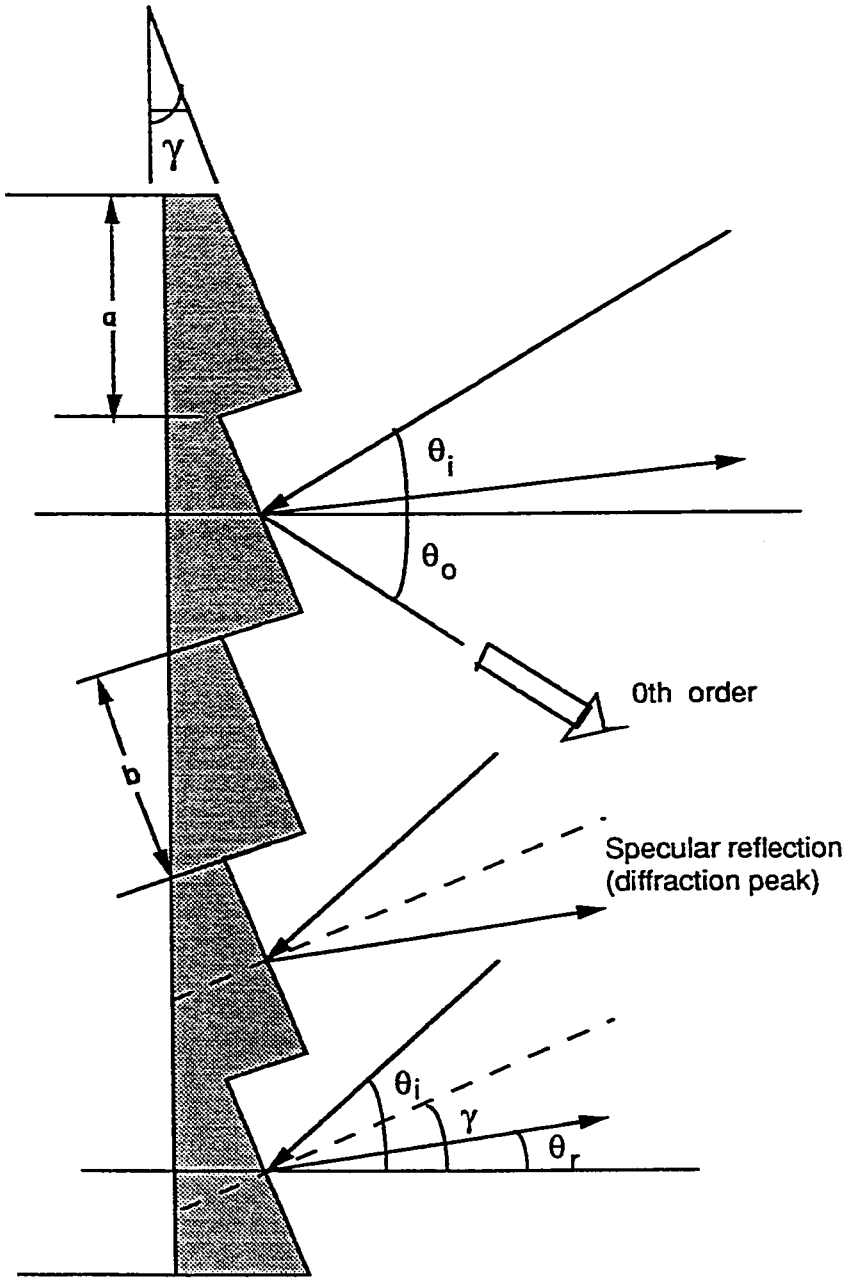


Figure 3.24 Analysis of the output of a reflection grating.

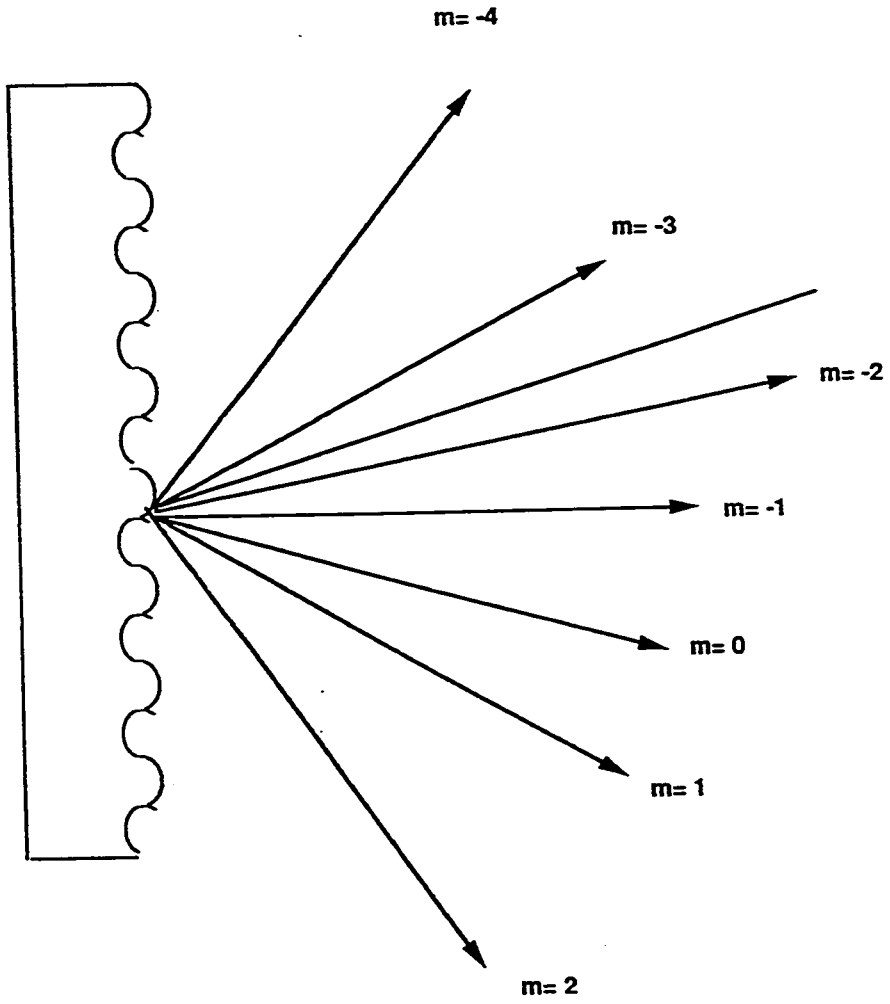


FIG. 3.25 Orders of output beams from a reflection grating

The attributes of this component are:

1. **Position P** of the center of the component
2. **Direction** of the face of incidence, if the grating was a reflection diffraction grating
3. **orientation** of the normal to the plane of incidence
4. **The type of the grating,** transmission or reflection
5. **Dimensions** (length and width) of the grating in (cm)
6. M, the number of slits/grooves in the grating
7. **a** and **b**, the distance between slits/grooves and the width of the slit/groove, consequently (See Figures 3.22, 3.23, and 3.24)
8. If the component is coated, then the **index of refraction of the coating material** and its **permeability** are needed
9. **Thickness** of the grating if it was a transmission grating
10. **Refractive index of the substrate of the grating** and its **permeability**

3.13 Filters

Four different types of filters, are considered below.

3.13.1 Neutral-Density Filters (NDFs)

The basic function of these filters is to reduce the amount of incident energy/power. This is usually achieved by *absorption*. Diagrams like the one shown in Figure 3.27 give the optical density of the filter as a function of wavelength. The transmittance of the filter depends on its optical density by the following formula:

$$T = 10^{-D} \tag{3.13}$$

The attributes of this filter are:

1. **Position P** of the center of the filter. The filter could have the shape of a square or a circle
2. **Direction** of the face of incidence
3. **Orientation** of the normal to the face of incidence
4. **Dimension** (side or diameter) of the filter (Figure 3.26)
5. **Thickness** of the filter
6. **Refractive index of the material of the filter**
7. **Transmittance T** of the filter or its **optical density D** curve
8. If the filter is coated, then the **refractive index of the coating material** and its **permeability** are needed

3.13.2 Interference Filters

Interference filters selectively reflect light beams who have wavelengths within the **reflected wavelength band** of the filter. This band can be narrow or wide

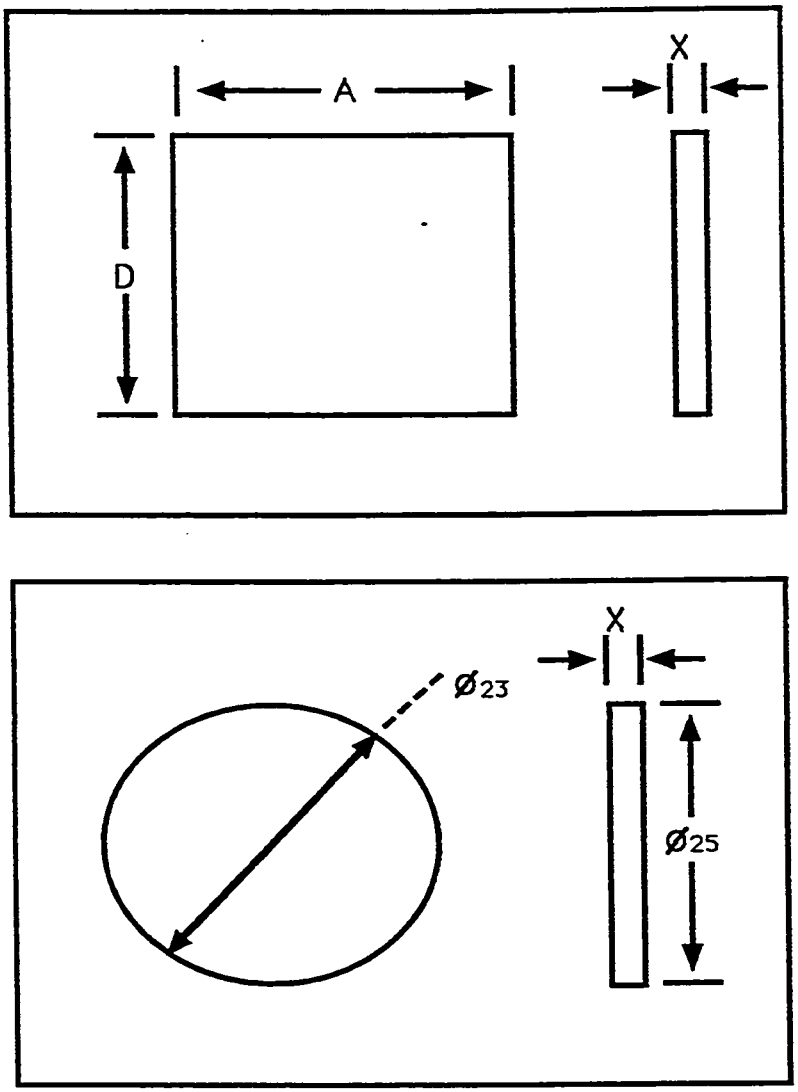


Figure 3.26 A neutral-density filter

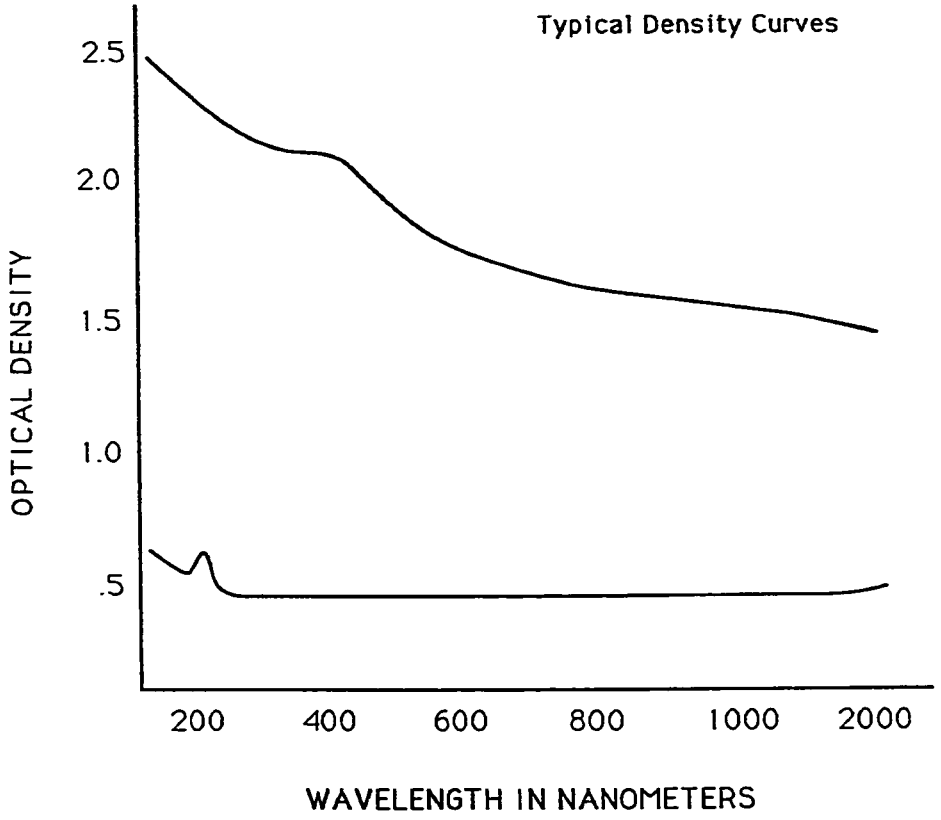


Figure 3.27 Optical density of neutral-density filters

depending on the filter itself. Figures 3.28 and 3.29 show the relation between wavelength of the incident beam and the transmittance of the filter. Each curve corresponds to a different filter. The attributes of these filters are:

1. **Position** P of the center of the filter
2. **Direction** of the face of incidence
3. **Orientation** of the normal to the plane of incidence
4. **Side** of the filter in (cm)
5. **Thickness** of the filter
6. **Refractive index of the material of the filter**
7. **Transmittance** T and **reflectance** R of the filter taken from the appropriate curve in the shown diagrams
8. If the filter is coated, then the **refractive index of the coating material** and its **permeability** are needed
9. A **table or diagram** that gives the delay forced by the filter over the incident light beam. This delay is due to the bouncing of the back and forth inside the filter. It is partially related to the wavelength of the beam. This is the main reason behinde needing a table or a diagram that describe this relation.

3.14 Spatial Light Modulators (SLM)

Spatial Light Modulators are used for storage purposes. Data is written on them by *imaging*. Initially, the SLM is in the *free state*. If a lens is used to image a

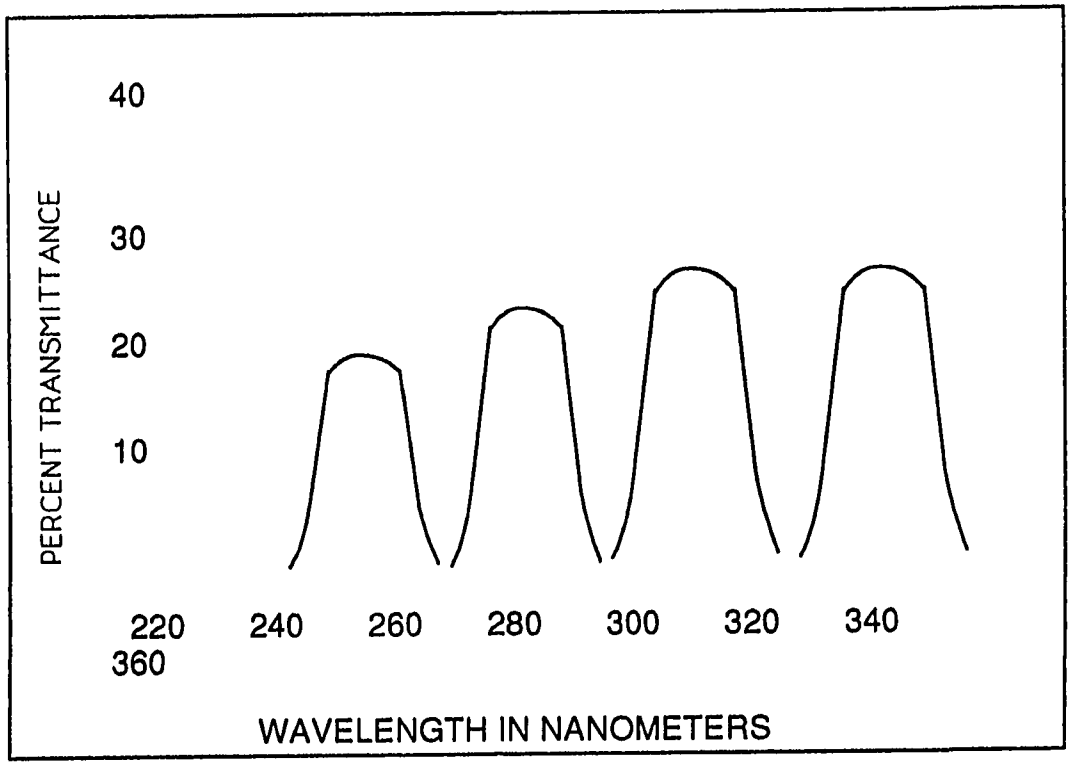


Figure 3.28 Transmittance of interference filters related to wavelength of incident light (narrow wavelength bands)

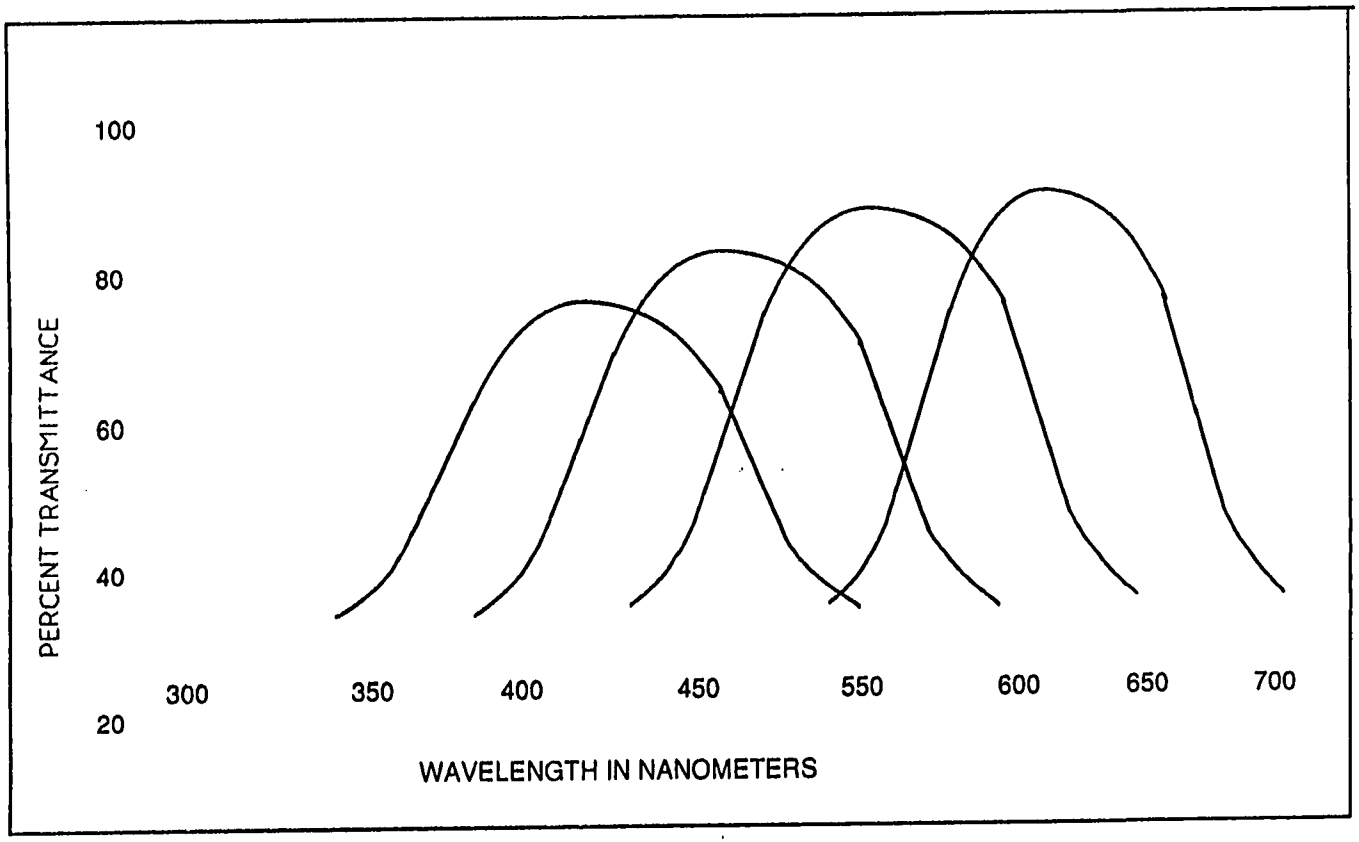


Figure 3.29 Transmittance of interference filters related to wave length of incident light (wide wavelength bands)

specific character on the SLM, the state of the SLM changes to *occupied state.* The occupation is in the sense that a certain characteristic of the SLM is changed by the light beam that wrote (imaged) the character on it.

We studied the operations of Liquid Crystal SLM's as discussed below [CASA77].

Liquid Crystal SLMs

They consist of a liquid crystal (LC) *photoconductor (PC)* sandwiched by transparent electrodes on top and bottom of it (Figure 3.30). An electric field is applied between the electrodes. Surface orientation effects tend to align the long axes of the LC molecules normal to the surface of the electrodes. These sources compete with the tendency of the molecules to align at angles or parallel to the electric field.

Upon writing on the LCSLM, the incident light distribution will change the resistance of the PC, thus producing a spatially varying electric-field distribution across the LC.

Upon reading from the LCSLM, if the reading light is linearly polarized, it will be rotated according to the twist angle of the molecules of the LC as they align depending on the value of the applied electric field.

Since the read beam can affect the electric field distribution, and thus affect the data stored in the LC, a *read face* was assigned for the read operation different from the *write face.* Therefore, the LCSLM is divided into two parts separated by a mirror that reflects both beams and stops them from reaching each other's ranges/portions.

Henceforth, the attributes of the LCSLM are:

1. **Position P** of the center of the SLM

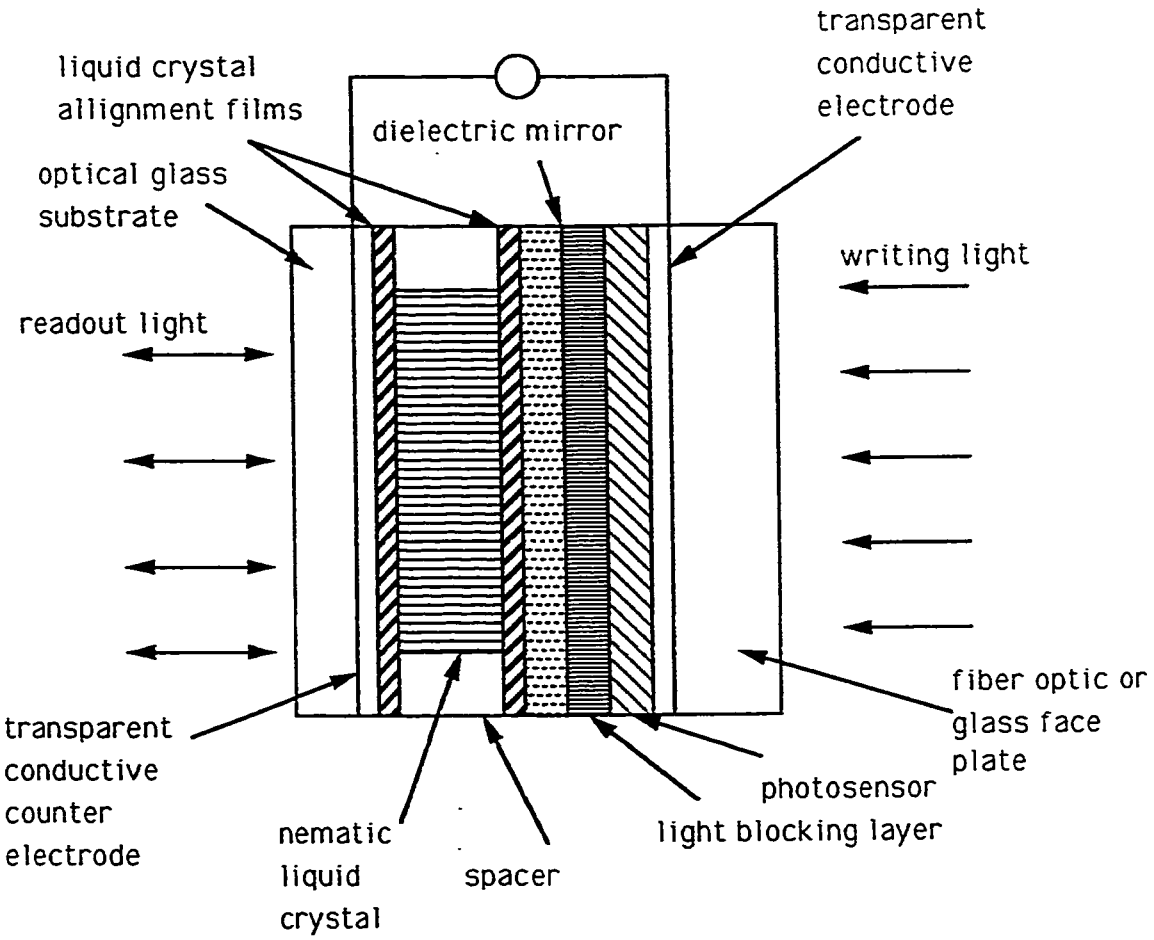


Figure 3.30 Liquid crystal spatial light modulator

2. **Orientation** of the axis of the SLM. The axis is the line that crosses both the top and bottom surfaces of the SLM and is normal to both.

3. **The directions of both the read face of the SLM.**

4. **Orientation** of the normal to the read face

5. **Thicknesses** $thick_{write}$ **and** $thick_{read}$ of the write and read sides (since they are usually seperated by a mirror)

6. **The initial twist angle** (θ_{twist}) of the molecules of the LCSLM with respect to the direction of the applied electric field

7. **The formula that explains the effect of the value of the applied electric field on the twist angle**

8. **The formula that describes the effects of the write beam on the value of the electric field**

9. **The formula that computes the rotation angle of the polarized read light beam** (θ_{rotate}). The twist angle of the direction of alignment of the LC molecules is the basic variable of the equation

10. $\mathbf{T_{read}}$ and $\mathbf{T_{write}}$, transmittances of both th read and write beam as they pass back and forth in the LCSLM

Chapter 4

COMPONENTS LIBRARY

The components library is a collection of procedures/algorithms. Each optical component is represented by one procedure. The system uses this library quite heavily when simulating programs that describe optical architectures.

The user of the system will use the language to write a program describing an experimented optical architecture. Within the program, the user specifies the components that make up the architecture and provides their attributes values. The system is then to simulate the effects of the architecture by simulating the effects of each individual component. This is achieved by calling the procedure of the corresponding component from the library.

The procedures of the library use the attributes values, described in Chapter 3, to mathematically simulate the functions and physical effects of the component.

Upon calling the procedure, the system passes the following data to it:

1. The attributes values of the medium/component given by the user in the description program (for detecting the effect of the component/medium on the inputs)

2. The direction(s) and characteristics of the input beam(s) (to use it for the computation of the direction and characteristics of the output beam(s))

3. Refractive index and permeability if the component/medium from which the input is coming from (for refraction computations)

In return, our system expects the procedure to return:

- The direction(s) and characteristics of the output beam(s)

In this chapter, we provide a procedure in pseudo code for each optical component discussed in Chapters 2 and 3. There will be a number of common inputs to the procedures and these are:

L_{d_0} the list of directions of the input beams, if there is only one input, the list will have only one term

L_{d_1} the directions of the output beam(s)

L_{in} the list of characteristics of input beam(s). If there is only one output beam, the list will be having a single term

L_{out} the list of characteristics of the output beam(s).

Direction of light beams

Before we move any further, the direction of propagation of a light beam has to be explained. Any beam propagating in free space could be considered as a vector with magnitudes x, y, and z and direction(Figure 4.1). We define the direction to be a **quintaple** of the values of:

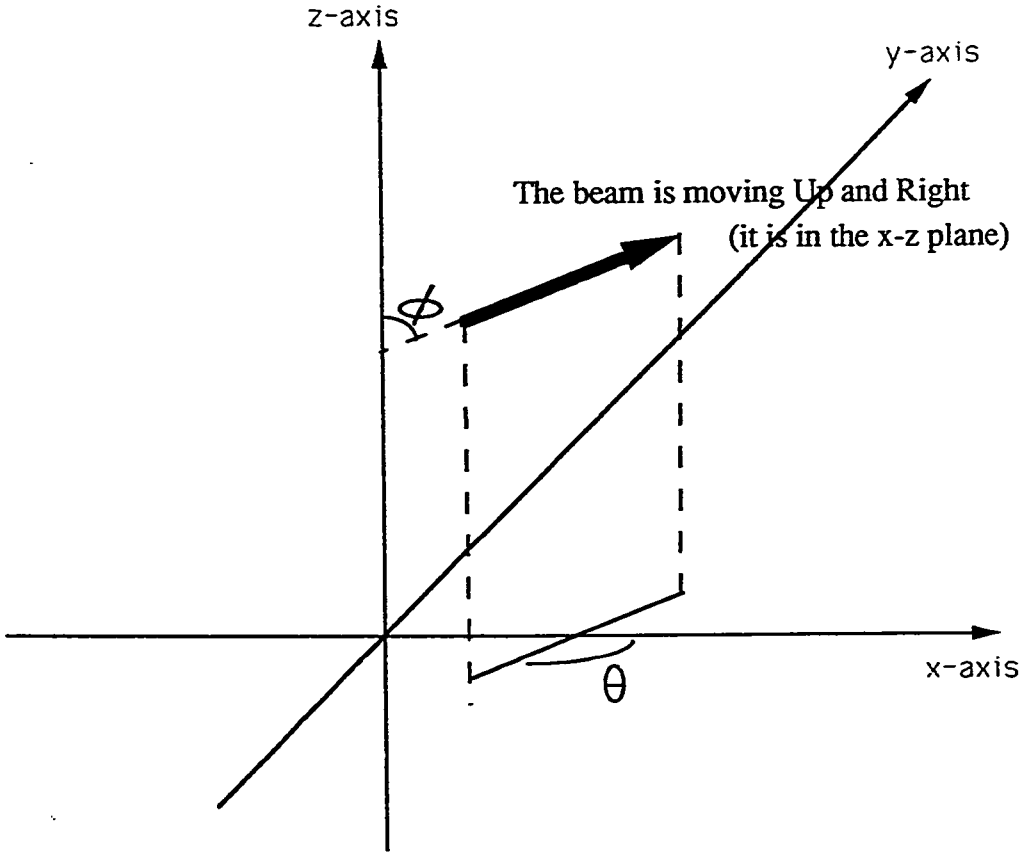


Figure 4.1 Light beam as a vector propagating in space

- θ and ϕ, which detect the orientation of the beam as explained earlier in Chapter 3

- **Up-z** if the beam was moving in the direction of positive $z-axis$, **Down-z** if the beam is moving in the direction of negative $z-axis$, or **Undef** if the beam is moving in a direction perpindicular to the $z-axis$

- **Right-x** if the beam was moving in the direction of positive $x-axis$, **Left-x** if the beam is moving in the direction of negative $x-axis$, or **Undef** if the beam is moving in a direction perpindicular to the $x-axis$

- **Forth-y** if the beam was moving in the direction of positive $y-axis$, **Back-y** if the beam is moving in the direction of negative $y-axis$, or **Undef** if the beam is moving in a direction perpindicular to the $y-axis$

One last point to consider before we start the discussion of procedures is that a number of functions made available for the use of the components' procedures. These are functions called frequently and hence, we found it more appropriate to provide them seperately as part of the overall system.

4.1 Light Propagation in Media

The procedure for this case needs to know the number of input beams to the medium it describes. Information about each beam are also required and they are:

- The direction of each beam in a list $L_{d_0} \equiv (d_{10}, d_{20}, d_{30}, \ldots)$

- The positions where each of the beams entered the medium in a list $L_{pos} \equiv (pos_1, pos_2, pos_3, \ldots)$

- The characteristics of each beam as it enters the medium. Each characteristic is given as a member of the list L_{in} as follows:
  - Velocity $(v_{10}, v_{20}, \ldots)$
  - Phase $(\varepsilon_{10}, \varepsilon_{20}, \ldots)$
  - Polarization $(pol_{10}, pol_{20}, \ldots)$
  - Beam diameter $(bd_{10}, bd_{20}, \ldots)$
  - Wavelength or frequency $(\lambda_{10}/\nu_{10}, \lambda_{20}/\nu_{20}, \ldots)$
  - Intensity $(I_{10}, I_{20}, \ldots)$

The procedure uses these data to compute and return the following information:

- The direction of the output beam(s) in the list $L_{d_1} \equiv (d_{11}, d_{12}, d_{13}, \ldots)$, they do not vary in media. Therefore, they will stay the same. They are needed for interference and superpositioning calculations
- The positions where the beams interfer inside the medium in a list, $L_{inter_{pos}} \equiv (pos_{ij}, pos_{jl}, pos_{ki}, \ldots)$, where i, j, k, and l are integers
- The characteristics of the resulting flux at the interference positions of $L_{inter_{pos}}$. These are given in a list $L_{inter_{out}}$
- The characteristics of each beam as it leaves the medium except the phase and beam diameter. They have to be computed by the system in this case because we need to know the position where these beams leave the medium. Remember that both phase and beam diameter change with distance. Each of the other characteristics is given as a member of the list L_{out} as follows:
  - Velocity $(v_{11}, v_{21}, \ldots)$

- Polarization ($pol_{11}, pol_{21}, \ldots$)
- Beam diameter ($bd_{11}, bd_{21}, \ldots$)
- Wavelength or frequency ($\lambda_{11}/\nu_{11}, \lambda_{21}/\nu_{21}, \ldots$)
- Intensity ($I_{11}, I_{21}, \ldots$)

- For each beam i, there will be an **interference list** ($L_{interfer_i}$). It contains the beams with which beam i interfers. Another list called the **superposition list** is also given (L_{super_i}). It contains the beams that superposition with beam i

The procedure in pseudo code is as follows:

algorithm $light - propagation(L_{d_0}, L_{d_1}, L_{in}, L_{out}, L_{pos}, L_{inter_{pos}}, L_{inter_{out}}$, attributes)

$L_{interfer} = \phi$

begin

N = the number of members of the list L_{d_0}

for $i = 1$ to N **do**

$L_{super_i} = \phi$

$M_i = \phi$

end loop i

$i = 1$

while ($i <= N - 1$) **do**

for $j = i + 1$ to N **do**

let θ_{ij} be the angle between directions of beams i and j

if ($\theta_{ij} \neq 0$) **then**

add the pair (i, θ_{ij}) to $L_{interfer}$

if ($\theta_{ij} = 0$ and $pos_i = pos_j$) **then**

```
                add j to L_super_i
                add j to M_i
/* M is the list of beams that should not be checked for interference or superposition.
That is because they already were found to interfer with beami */
            endif
        j = j + 1
        end loop j
    i = i + 1
    while (i ∈ M_i) do
        i = i + 1
    end while i
    for i = 1 to N
        if (L_super_i ≠ φ) then
            let k be the number of members of L_super_i
            j = 1 /* first member of L_super_i */
            for l = 2 to k /* the following members */
                if (beam_j and beam_l are plane polarized) then
                  let θ_jl be the angle between the planes of polarization
                  I_j = I_j × cos² θ_jl
                  I_l = I_l × cos² θ_jl
                  /* these produce new values for E_01 and E_02 */
                endif
                if (ν_j = ν_l) then
                  apply Equations 2.54, 55, 56, and 57 of Section 2.3.1 to
                  compute ν and E of the resultant
```

```
                else
                  apply Equations 2.60 and 2.61 to compute ν and E of the resultant
                endif
                I_j = εvE^2
/* The characteristics of the superposition resultant beam are given to the first member
of the list L_super_i */
                remove beam_l from L_interfer
/* Only beam_i stays in the interference list as a representative of L_super_i */
              end loop l
            endif
          end loop i
          if (L_interfer ≠ φ) then
            let k be the number of members of L_interfer
            for j = 1 to k − 1 do
              for l = j + 1 to k do
                let pos_jl = (x_jl, y_jl, z_jl) be the position where beam_j and beam_l interfere
                add pos_jl to L_inter_pos
                if (beam_j and beam_l are plane polarized) then
                  I_j = I_j × cos^2 θ_jl
                  I_l = I_l × cos^2 θ_jl
                endif
                apply Equations 2.73 and 74 to compute the
                intensity of the resulting flux at pos_jl
              end loop l
            end loop j
```

endif

let N be the number of beams left in the media after checking for superpositioned beams

for $i = 1$ **to** N

if ($beam_i$ is coming from a lens) **then**

if (the lens was convex) **then**

$bd_{i1} = w_{0_1}$ at a distance equal to the **focal length** of the lens from pos_i (Section 2.1.2)

else if (lens is concave) **then**

beam diameter is computed according to Equation 2.29 with the value of w_0 taken at the focal point of the lens after tracing the beam using Equation 3.2

endif

else

$w_0 = bd_{i0}$

endif

end loop i

end

4.2 Continuouis Laser Sources

These are the sources of light beams in optical systems. Therefore the only inputs the procedure, which describes this component, needs are the attributes values provided by the user in the description program. The procedure computes the direction of the output beam d_1. Notice that we only have one output beam from the component.

The procedure is as follows:

algorithm *continuous* − *laser*(d_l, attributes)

/* the orientation of the output beam is parallel to the face of incidence.

Thus, normal to the normal to the face of incidence */

begin

if ($\theta_{normal} > 90°$) **then**

$\theta_l = \theta_{normal} - 90°$

else if ($\theta_{normal} < 90°$) **then**

$\theta_l = 90° + \theta_{normal}$

else if ($\theta_{normal} = 90°$) **then**

$\theta_l = 0°$

else if ($\theta_{normal} = 0$) **then**

$\theta_l = 90°$

endif

if ($\phi_{normal} > 90°$) **then**

$\phi_l = \phi_{normal} - 90°$

else if ($\phi_{normal} < 90°$) **then**

$\theta_l = 90° + \phi_{normal}$

else if ($\phi_{normal} = 90°$) **then**

$\phi_l = 0°$

else if ($\phi_{normal} = 0$) **then**

$\phi_l = 90°$

endif

/* This previous computes the orientation of the propagating output light beam */

/* Remember that two of the attributes of continuous lasers were two reference points,

P1 at the center of the component and P2 at its head. These two points are sufficient to determine the direction of propagation of the output beam, since the beam will come out from the head of the component (P2). */

if $(P2.x - coordinate > P1.x - coordinate)$ **then**
  light is propagating **Right-x**
else
  light is propagating **Left-x**
endif
if $(P2.y - coordinate > P1.y - coordinate)$ **then**
  light is propagating **Forth-y**
else
  light is propagating **Back-y**
endif
if $(P2.z - coordinate > P1.z - coordinate)$ **then**
  light is propagating **Up-z**
else
  light is propagating **Down-z**
endif

end

4.3 Pulsed Laser Sources

The procedure for these components is the same one used for description of continuous laser sources. Our system takes care of other issues such as the **pulse rate** and **pulse duration** with respect to time. Another input has to be given to the procedure. That is the time of the call to the procedure from the system. It

is important to decide whether a pulse is generated at that specific time or not.

4.4 Plate Beam Splitters

We expect one input and two outputs. For this procedure we require the refractive index of the medium from which the light beam is coming n_{in} to be given as an input to the procedure. In the procedure, we consider the reflected beam to be beam number 1 and the transmitted to be beam number 2.

The position where the input beam enters the plate is taken as an input in the list L_{pos}. The positions where the beams leave the component are given in the same list L_{pos}.

The procedure is as follows:

---

algorithm $plate - splitter(L_{in}, L_{out}, L_{d_0}, L_{d_1}, n_{in}, L_{pos}$, attributes)
begin

/* $reflected - direction$ is a function in the system that computes the direction of the reflected beam from a surface. It is one of the functions available in the system for the interpretation of description programs */

$d_{11} = reflected - direction(d_{10}$, direction of the face of incidence of the plate, orientation of the normal to the face of incidence)

/* $transmitted - direction$ is a function in the system that computes the direction of the transmitted beam from a surface.
of the functions available in the system for the interpretation of programs */

$d_{21} = transmitted - direction(d_{10}$, direction of the face of incidence, orientation of the normal to the face of incidence, thickness of the plate,

the refractive index of the plate, the refractive index

of the medium of propagation)

/* $reflected - polarization$ is a function in the system that computes the polarization of the reflected beam from a surface.

of the functions available in the system for the interpretation of description programs */

$pol_{11} = reflected - polarization(pol_{10}$, direction of the face of incidence of

the plate, orientation of the normal to the face of incidence,

the refractive index of the coating layer or the refractive index

of the substrate of the plate if the plate is not coated,

the refractive index of the medium of propagation)

/* $transmitted - polarization$ is a function in the system that computes the polarization of the transmitted beam from a surface. It is one of the

functions available in the system for the interpretation of description programs */

$pol_{21} = transmitted - polarization(pol_{10}$, direction of the face of incidence of

the plate, orientation of the normal to the face of incidence,

the refractive index of the coating layer or the refractive index

of the substrate of the plate if the plate is not coated,

the refractive index of the medium of propagation)

$\varepsilon_{11} = \varepsilon_{10}$

/* $distance; n_p late$ computes the distance the beam overgoes inside the plate beam splitter */

$distance = distance - in - plate(pos_{10}$, d_{10} refractive index of the plate,

refractive index of the medium from which the beam is coming)

$\varepsilon_{21} = \varepsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

/* 2π could be replaced by the actual wavelength in meters. In this case, however, the phase will have values in the interval $[0, \lambda]$ */

$pos_{11} = pos_{10}$

$pos_{21} = out - pos - non - normal(pos_{10}$, thickness of the plate, refractive index of the plate, refractive index of the medium of propagation, direction of the face of incidence, orientation of the normal to the face of incidence)

/* $out - pos - non - normal$ is a function that computes the position at which the transmitted beam leaves the component.

non-normal means that the incidence is not normal */

$I_{11} = I_{10} \times R$

$I_{21} = I_{10} \times T$

end

4.5 Cube Beam Splitters

With this component, we assume normal incidence. That is because the best results found in manufacturers catalogues as well as in calibration reports were taken at incidence angle equal to zero (normal). These components can take either one or two inputs.

In the procedure, we refer to the reflected beam as beam number 1. The transmitted beam is given the number 2.

The procedure is:

---

algorithm $cube - splitter(I_{in}, I_{out}, I_{d_0}, I_{d_1}, I_{pos}$, attributes)

begin

if $(\theta_{incidence} \neq 0)$ **then**

$I_{10} = I_{10} \times R1$

endif

let N be the number of input beams to the splitter

if $(N = 1)$ **then**

/* $reflected - direction$ is the same function discussed for the plate beam splitter */

$d_{11} = reflected - direction(d_{10}$, direction of the inner face of incidence of the cube, orientation of the normal to the inner face of incidence)

$d_{21} = d_{10}$

/* $reflect - prism - dis$ is a system function that computes the distance travelled by the reflected beam inside the right-angle prism */

$distance = reflect - prism - dis(pos_{10}, d_{10}$, orientation of the normal to hypotenuse, direction of the hypotenuse)

$\epsilon_{11} = \epsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

/* $side$ is the side of the cube dimension in meters */

$distance = side$

/* 2π could be replaced by the actual wavelength in meters. In this case, however, the phase will have values in the interval $[0, \lambda]$ */

$\epsilon_{21} = \epsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

$I_{11} = I_{10} \times R_{avg}$

$I_{21} = I_{10} \times T_{avg}$

/* $reflect - out - pos - normal$ is a function that computes the position at which the reflected beam leaves the component. normal means that the incidence is normal */

$pos_{11} = reflect - out - pos - normal(pos_{10}$, side of the cube, refractive index of the cube, refractive index of the medium where the beam comes from, direction of the inner face of incidence, orientation of the normal to the inner face of incidence)

/* *transmit − out − pos − normal* is a function that computes the position at which the transmitted beam leaves the component. **normal** means that the incidence is normal */

pos_{21} = *transmit − out − pos − normal*(pos_{10}, side of the cube, refractive index of the cube, refractive index of the medium where the beam comes from, direction of the face of incidence, orientation of the normal to the face of incidence)

else

$d_{11} = d_{10}$

$d_{21} = d_{10}$

let $position_{super}$ be the position where the two beams meet in order to superposition and produce two output beams

let ε_1 and ε_2 be the initial phases of beams 1 and 2 at $position_{super}$

$I_{10_{trans}} = I_{10} \times T_{avg}$

$I_{10_{ref}} = I_{10} \times R_{avg}$

$I_{20_{trans}} = I_{20} \times T_{avg}$

$I_{20_{ref}} = I_{20} \times R_{avg}$

/* $I_{10_{trans}}$ and $I_{20_{ref}}$ will make I_{11}. On the other hand, $I_{20_{trans}}$ and $I_{10_{ref}}$ will make I_{21}. */

for $i = 1$ **to** 2 **do**

if $(\nu_1 = \nu_2)$ **then**

apply Equations 2.54, 55, 56, and 57 of Section 2.3.1 to compute ν and E_{i1} of the resultant

else

apply Equations 2.60 and 2.61 to compute ν and E_{i1} of the resultant

endif

$I_{i1} = c\nu E_{i1}^2$

end loop i

let *distance* be the distance from $position_{super}$ and the output plane of the cube beam splitter

$\epsilon_{21} = \epsilon_{20} + ([(distance)/\lambda_{20}] - INTEGER([(distance)/\lambda_{20}])) \times 2\pi$

/* 2π could be replaced by the actual wavelength in meters. In this case, however, the phase will have values in the interval $[0, \lambda]$ */

$\epsilon_{11} = \epsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

/* *reflect − out − pos − normal* is a function that computes the position at which the reflected beam leaves the component.
normal means that the incidence is normal */

pos_{11} = *reflect − out − pos − normal*(pos_{10}, side of the cube, refractive index of the cube, refractive index of the medium where the beam comes from, direction of the inner face of incidence, orientation of the normal to the inner face of incidence)

/* *transmit − out − pos − normal* is a function that computes the position at which the transmitted beam leaves the component.
normal means that the incidence is normal */

pos_{21} = *transmit − out − pos − normal*(pos_{10}, side of the cube, refractive index of the cube, refractive index of the medium where the beam comes from, direction of the face of incidence, orientation of the normal to the face of incidence)

endif

end

4.6 Flat Mirrors

With mirrors, one might expect one output beam resulting from the reflection of an incident beam over a mirror. However, this is not the case always because part of the incident power gets transmitted by the mirror. Therefore, we have to take care of that part unless its power is so small and could be neglected. We say that the light beam could be neglected if its power as it leaves the mirror $I \leq 0.3\mu Watts$.

The procedure for mirrors is:

algorithm $flat-mirror(I_{in},\ I_{out},\ I_{d_0},\ I_{d_1}\ I_{pos},\ n_{in},$ attributes)
begin

/* $reflected-direction$ is a function in the system that computes the direction of the reflected beam from a surface. It is one of the functions available in the system for the interpretation of programs */

$d_{11} = reflected-direction(d_{10},$ direction of the face of incidence
of the mirror, orientation of the normal to the face of incidence)

/* $reflected-polarization$ is a function in the system that computes the polarization of the reflected beam from a surface */

$pol_{11} = reflected-polarization(pol_{10},$ direction of the face of incidence
of the mirror, orientation of the normal to the face of incidence,
the refractive index of the coating layer
or the refractive index of the substrate of the plate if the plate is not coated,
the refractive index of the medium of propagation)

$\epsilon_{11} = \epsilon_{10}$

$pos_{11} = pos_{10}$

$I_{11} = I_{10} \times R$

if $(I_{10} - I_{11} \geq (0.3/w_{10}^2))$ **then**

/* $transmitted - direction$ is a function in the system that computes the direction of the transmitted beam from a surface */

$d_{21} = transmitted - direction(d_{10}$, direction of the face of incidence, orientation of the normal to the face of incidence, thickness of the mirror, the refractive index of the mirror, the refractive index of the medium the light was coming from)

/* $transmitted - polarization$ is a function in the system that computes the polarization of the transmitted beam from a surface */

$pol_{21} = transmitted - polarization(pol_{10}$, direction of the face of incidence of the mirror, orientation of the normal to the face of incidence, the refractive index of the coating layer or the refractive index of the substrate of the mirror if it is not coated, the refractive index of the medium of propagation)

/* $distance - in - plate$ computes the distance the beam overgoes inside the mirror */

$distance = distance - in - plate(pos_{10}$, d_{10} refractive index of the mirror, refractive index of the medium from which the beam is coming, thickness)

/* 2π could be replaced by the actual wavelength in meters. In this case, however, the phase will have values in the interval $[0, \lambda]$ */

$\epsilon_{21} = \epsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

/* $out - pos - non - normal$ is a function that computes the position at which the transmitted beam leaves the component. $non - normal$ means that incidence is not normal */

$pos_{21} = out - pos - non - normal(pos_{10}$, thickness of the mirror, refractive index of the mirror, refractive index of the medium where the beam comes from,

direction of the face of incidence, orientation of the normal to the face of incide

$I_{21} = I_{10} \times T$

endif

end

4.7 Windows

Windows pass light from one medium to another. In the procedure to follow, n_1 is the refractive index of the medium from which light is coming, n is the refractive index of the window, and n_2 is the refractive index of the medium on the rearside of the face of incidence.

The procedure is:

algorithm *window*(L_{in}, L_{out}, L_{d_0}, L_{d_1}, n_1, n_2, attributes)

begin

$I_{11} = I_{10} \times T$

/* *transmit − out − pos − normal* is the same in cube beam splitters */

pos_{11} = *transmit − out − pos − normal*(pos_{10}, thickness of the window,

refractive index of the window, refractive index of the medium of propagation,

direction of the face of incidence, orientation of the normal to the face of incide

$d_{11} = d_{10}$

/* *thickness* is that of the window in meters */

distance = *thickness*

/* 2π could be replaced by the actual wavelength in meters. In this case, however, the phase will have values in the interval $[0, \lambda]$ */

$\varepsilon_{11} = \varepsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

end

4.8 Right-angle Prism

As was described in Section 3.7 and as shown in Figure 3.12, there are three different faces of incidence for this component. We call them A, B, and the hypotenuse C.

The procedure is:

algorithm $right-angle-prism(L_{in}, L_{out}, L_{d_0}, L_{d_1}, face-of-incidence,$ attributes)
begin

if ($face-of-incidence$ is A or B) **then**

/* $reflected-direction$ is the same function discussed for the plate beam splitter */

$d_{11} = reflected-direction(d_{10},$ direction of the hypotenuse of the prism, orientation of the normal to the hypotenuse)

if ($face-of-incidence$ is A) **then**

$I_{11} = I_{10} \times T1$

else

$I_{11} = I_{10} \times T2$

endif

/* $reflect_out_pos_normal$ is a function that computes the position at which the reflected beam leaves the component. **normal** means that the incidence is normal */

$pos_{11} = reflect-out-pos-normal(pos_{10},$ face-of-incidence, dimensions of the prism, refractive index of the prism, refractive index of the medium of propagation, direction of the hypotenuse, orientation of the normal to the hypotenuse)

/* $reflect-prism-dis$ is a system function that computes the distance travelled by

the reflected beam inside the right-angle prism */

$distance = reflect - prism - dis(pos_{10}, d_{10},$ orientation of the normal to
hypotenuse, direction of the hypotenuse)

$\epsilon_{11} = \epsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

else

/* $total - internal - reflection$ is a system function that computes the direction of a totaly reflected beam from the hypotenuse of a right-angle prism */

$d_{11} = total - internal - reflection(d_{10},$ direction of the hypotenuse
of the prism, orientation of the normal to the hypotenuse)

$I_{11} = I_{10} \times T3$

/* $total - internal - reflect - out$ is a function that computes the position at which th reflected beam leaves the component */

$pos_{11} = total - internal - reflect - out$ (pos_{10}, dimensions of prism,
refractive index of the prism, refractive index of the medium of propagation,
direction of the hypotenuse, orientation of the normal to the hypotenuse)

/* $reflect - prism - dis$ is a system function that computes the distance travelled by the reflected beam inside the right-angle prism */

$distance = reflect - prism - dis(pos_{10}, d_{10},$ orientation of the normal to
hypotenuse, direction of the hypotenuse)

$\epsilon_{11} = \epsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

endif

end

4.9 Retardation Plates

Retardation plates affect polarization. They are of two types, half and quarter-wave plates. They were discussed in the previous chapters. Normal incidence is expected for these components.

Their procedure is:

---

algorithm *retardation − plate*(I_{in}, I_{out}, I_{d_0}, I_{d_1}, n_1, type-of-plate, attributes)

begin

$I_{11} = I_{10} \times T$

pos_{11} = *transmit − out − pos − normal*(pos_{10}, thickness of the plate, refractive index of the plate, refractive index of the medium where the beam comes from, direction of the face of incidence, orientation of the normal to the face of incidence)

/* *transmit − out − pos − normal* is the same in cube beam splitters */

$d_{11} = d_{10}$

distance = *thickness*

/* *thickness* is that of the window in meters */

$\varepsilon_{11} = \varepsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

/* 2π could be replaced by the actual wavelength in meters. In this case, however, the phase will have values in the interval $[0, \lambda]$ */

if (*type − of − plate* is *quarter − wave − plate*) **then**

if (pol_{10} = *plane*) **then**

pol_{11} = *circular*

else if (pol_{10} = *circular*) **then**

pol_{11} = *plane*

$pol_{11_{dir}}$ = *optic − axis − orientation*

/* $pol_{11_{dir}}$ is the direction of the output plane polarized beam which is

normal to the plane of incidence and normal to the normal to the optic axis (parallel to the optic axis)

endif

else

if ($pol_{10} = plane$) **then**

$pol_{11_{dir}} = pol_{10_{dir}}$ rotated by 90°

else if ($pol_{10} = left - circular$) **then**

$pol_{11} = right - circular$

else if ($pol_{10} = right - circular$) **then**

$pol_{11} = left - circular$

endif

endif

end

4.10 Glan-Taylor and Glan-Thompson Prisms

These components are more like cubes than prisms. Hence, they have procedures similar to those of cube beam splitters. Except here the reflected part is fully absorbed by the components' mount.

The procedure is:

---

algorithm $taylor - thompson - prism(L_{in}, L_{out}, L_{d_0}, L_{d_1}, n_1,$ attributes)

begin

$d_{21} = d_{10}$

$distance = side$

/* *side* is the length of the cube in meters */

$\epsilon_{11} = \epsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

/* 2π could be replaced by the actual wavelength in **meters**. In this case, however, the phase will have values in the interval $[0, \lambda]$ */

$pos_{21} = transmit - out - pos - normal(pos_{10}$, length of the cube, refractive index of the cube, refractive index of the medium where the beam comes from, direction of the face of incidence, orientation of the normal to the face of incidence)

/* *transmit − out − pos − normal* is a function that computes the position at where the transmitted beam leaves the component. **normal** means that the incidence is normal */

if $(pol_{10} = unpolarized)$ **then**

$pol_{11} = plane$

$pol_{11_{dir}} = normal - to - plane - of - incidence$

/* The output beam is plane polarized normal to the plane of incidence */

$I_{11} = I_{10} \times T$

else if $(pol_{10} = plane)$ **then**

let θ_{pol} be the angle between $pol_{10_{dir}}$ and the plane of incidence

$I_{11} = (I_{10} \times cos^2\ (90° - \theta_{pol})) \times T$

endif

end

4.11 Polarizing Beam Splitters

With these components, we expect one input light beam normal to the face of incidence of the splitter. This light beam could be either plane polarized or of any other state of polarization.

Polarizing beam splitters are cube beam splitters with a composing substrate

that has a birefringent feature. This feature allows them to separate the orthogonal components of any unpolarized light beam.

Their procedure is:

algorithm $polarizing - beam - splitter(I_{in},\ I_{out},\ I_{d_0},\ I_{d_1},$ attributes)

begin

if $(\theta_{incidence} \neq 0)$ **then**

$I_{10} = I_{10} \times R1$

endif

let N be the number of input beams to the splitter

if $(N = 1)$ **then**

$d_{11} = reflected - direction(d_{10},$ direction of the inner face of incidence of the cube, orientation of the normal to the inner face of incidence)

/* $reflected - direction$ is the same function discussed for the plate beam splitter */

$d_{21} = d_{10}$

$distance = reflect - prism - dis(pos_{10},\ d_{10},$ orientation of the normal to hypotenuse, direction of the hypotenuse)

/* $reflect - prism - dis$ is a system function that computes the distance travelled by the reflected beam inside the right-angle prism (the part where the reflected beam travells in) */

$\varepsilon_{11} = \varepsilon_{10} + ([[(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

$distance = side$

/* $side$ is the side of the cube dimension in meters */

$\varepsilon_{21} = \varepsilon_{10} + ([[(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

/* 2π could be replaced by the actual wavelength in meters. In this case, however, the phase will have values in the interval $[0, \lambda]$ */

$I_{11} = I_{10} \times R_{avg}$

$I_{21} = I_{10} \times T_{avg}$

$pos_{11} = reflect - out - pos - normal(pos_{10}$, side of the cube, refractive index of the cube, refractive index of the medium where the beam comes from, direction of the inner face of incidence, orientation of the normal to the inner face of incidence)

/* $reflect - out - pos - normal$ is a function that computes the position at which the reflected beam leaves the component. **normal** means that the incidence is normal */

$pos_{21} = transmit - out - pos - normal(pos_{10}$, side of the cube, refractive index of the cube, refractive index of the medium where the beam comes from, direction of the face of incidence, orientation of the normal to the face of incidence)

/* $transmit - out - pos - normal$ is a function that computes the position at which the transmitted beam leaves the component. **normal** means that the incidence is normal */

if $(pol_{10} = unpolarized)$ **then**

$I_{11} = I_{10} \times T1$

$pol_{11} = plane$

$pol_{11_{dir}} = normal - to - face - of - incidence$

$I_{21} = I_{10} \times T2$

$pol_{21} = plane$

$pol_{21_{dir}} = parallel - to - face - of - incidence$

/* Beam number 2 is the transmitted (extraordinary) ray and the reflected (ordinary) ray is beam number 1

else if $(pol_{10} = plane)$ **then**

let θ_{pol} be the angle between the plane of polarization of the incident beam and the face of incidence

$I_{11} = (I_{10} \times cos^2\ \theta_{pol}) \times T1$

$pol_{11} = plane$

$pol_{11_{dir}} = normal - to - face - of - incidence$

$I_{21} = (I_{10} \times cos^2\ (90°\ \theta_{pol}) \times T2$

$pol_{21} = plane$

$pol_{21_{dir}} = parallel - to - face - of - incidence$

endif

else

$d_{11} = d_{10}$

$d_{21} = d_{10}$

let $position_{super}$ be the position where the two beams meet in order to superposition and produce two output beams

let ε_1 and ε_2 be the initial phases of beams 1 and 2 at $position_{super}$

$I_{10_{trans}} = I_{10} \times T_{avg}$

$I_{10_{ref}} = I_{10} \times R_{avg}$

$I_{20_{trans}} = I_{20} \times T_{avg}$

$I_{20_{ref}} = I_{20} \times R_{avg}$

/* $I_{10_{trans}}$ and $I_{20_{ref}}$ will make I_{11}. On the other hand, $I_{20_{trans}}$ and $I_{10_{ref}}$ will make I_{21}. */

for $i = 1$ to 2

if $(\nu_1 = \nu_2)$ **then**

apply Equations 2.54, 55, 56, and 57 of Section 2.3.1 to compute ν and E_{i1} of the resultant

else

apply Equations 2.60 and 2.61 to compute ν and E_{i1} of the resultant

endif

$I_{i1} = \epsilon v E_{i1}^2$

end loop i

let *distance* be the distance from $position_{super}$ and the output plane of the cube beam splitter

$\epsilon_{21} = \epsilon_{20} + ([(distance)/\lambda_{20}] - INTEGER([(distance)/\lambda_{20}])) \times 2\pi$

$\epsilon_{11} = \epsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

/* 2π could be replaced by the actual wavelength in **meters**. In this case, however, the phase will have values in the interval $[0, \lambda]$ */

$pos_{11} = reflect - out - pos - normal(pos_{10}$, side of the cube, refractive index of the cube, refractive index of the medium where the beam comes from, direction of the inner face of incidence, orientation of the normal to the inner face of incidence)

/* *reflect − out − pos − normal* is a function that computes the position at which the reflected beam leaves the component.
normal means that the incidence is normal */

$pos_{21} = transmit - out - pos - normal(pos_{10}$, side of the cube, refractive index of the cube, refractive index of the medium where the beam comes from, direction of the face of incidence, orientation of the normal to the face of incidence)

/* *transmit − out − pos − normal* is a function that computes the position at which the transmitted beam leaves the component.
normal means that the incidence is normal */

endif

end

4.12 Dichroic Sheet Polarizers

These component produce only plane polarized light with a plane of polarization parallel to their optic axis. The other component of polarization, if any, gets fully absorbed.

The procedure is:

algorithm $dicroic - sheet(I_{in}, I_{out}, I_{d_0}, I_{d_1}, n_{in},$ attributes)

begin

$pos_{11} = transmit - out - pos - normal(pos_{10},$ thickness of the sheet, refractive index of the sheet, refractive index of the medium where the beam comes from, direction of the face of incidence, orientation of the normal to the face of incidence)

/* $transmit - out - pos - normal$ is the same in cube beam splitters */

$d_{11} = d_{10}$

$distance = thickness$

/* $thickness$ is that of the sheet in meters */

$\epsilon_{11} = \epsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

/* 2π could be replaced by the actual wavelength in meters. In this case, however, the phase will have values in the interval $[0, \lambda]$ */

$pol_{11} = plane$

$pol_{11_{dir}} = parallel - to - optic - axis$

if $(pol_{10} = unpolarized)$ **then**

$I_{11} = I_{10} \times T$

else if $(pol_{10} = plane)$ **then**

let θ_{pol} be the angle between the optic axis and the plane of polarization of the incident light

$I_{11} = I_{10} \times (K_1\ cos^2\ \theta_{pol} + K_2\ sin^2\ \theta_{pol})$

endif

end

4.13 Spherical and Cylindrical Lenses

The type of the lens determines the output beam diameter as was mentioned in the procedure for light propagation. Therefore, it will be given by the user as an input to the lens' procedure.

The procedure is:

---

algorithm *lens*(L_{in}, L_{out}, L_{d_0}, L_{d_1}, type, attributes)

begin

$d_{11} = d_{10}$ /* for all types of lenses */

apply Equations 3.2-10 to trace the beam as it leaves the lens

/* These equations trace the light beam inside the lens and give the slope of the upper and lower edges of it as it leaves the lens */

$pos_{11} = transmit - out - pos - normal(pos_{10}$, t_c, t_e of the lens, refractive index of the lens, refractive index of the medium where the beam comes from, direction of the face of incidence, orientation of the normal to the face of incidence)

/* $transmit - out - pos - normal$ is the same in cube beam splitters */

$distance = t_c + t_e$

/* t_c and t_e were discussed in Chapter 3 */

$\epsilon_{11} = \epsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

/* 2π could be replaced by the actual wavelength in meters. In this case, however,

the phase will have values in the interval $[0, \lambda]$ */

$I_{11} = I_{10} \times T$

end

4.14 Diffraction Gratings

With gratings, as was shown in Section 3.12, unlimited number of output beams are produced. Each of these output beams makes a different angle with the normal to the face of incidence of the grating. Each beam will have a different intensity at different directions. Therefore, it is left to the system to decide which one of these output beams it needs to have information about.

What the system does is that it detects the angle that the desired beam makes with the normal to the face of incidence and passes that to the procedure. The angle is detected by detecting the position where information about light is required. After knowing the position, the line starting from position of incidence and ending at the position where information is required is taken as the required beam orientation. The angle between this line (beam) and the normal to the face of incidence is then easily computed and passed as one of the parameters to the following procedure.

The type of the grating (transmission/reflection) has to be given as a parameter to the procedure. The procedure is:

---

algorithm *grating*(I_{in}, I_{out}, I_{d_0}, I_{d_1}, *type*, θ_m, attributes)

begin

if (*type* = *transmission*) **then**

distance = *thickness*

$\epsilon_{11} = \epsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

$pos_{21} = out - pos - non - normal(pos_{10}$, thickness of the grating, refractive index of the grating, refractive index of the medium where the beam comes from, direction of the face of incidence, orientation of the normal to the face of incidence)

/* $out - pos - non - normal$ is the system function used with plate beam splitters */

else

$pos_{11} = pos_{10}$

$\epsilon_{11} = \epsilon_{10}$

endif

$N = bd_{10}/(M/length)$

/* to compute the number of slits/grooves incident by the input light beam */

$I_{11} = I_{10} \times (sin\beta/\beta)^2 \times (sinN\alpha/sin\alpha)^2$

/* d_{11} is determined by the system prior to calling this procedure */

end

4.15 Neutral-Density Filters NDFs

These components absorb, partially, the energy of the light that passes through them. The amount of energy transmitted is given in curves produced by calibration reports. These curves display the relation between the wavelength of the incident light and either the transmittance or optical density of the filter. The system gets these values from the curves and passes them to the procedure.

We consider only the case of normal incidence because it gives the optimum results.

The procedure is:

algorithm $density - filter(I_{in}, I_{out}, I_{d_0}, I_{d_1}, n_1,$ attributes)

begin

$I_{11} = I_{10} \times T = I_{10} \times 10^{-D}$

/* D is the density of the filter used only if T was not given */

$pos_{11} = transmit - out - pos - normal(pos_{10},$ thickness of the filter, refractive index of the filter, refractive index of the medium where the beam comes from, direction of the face of incidence, orientation of the normal to the face of incidence)

$d_{11} = d_{10}$

$distance = thickness$

$\varepsilon_{11} = \varepsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

end

4.16 Interference Filters

Depending on the wavelength of the incident light beam, the filter either transmits or reflects the incident light. This is again decided by looking at the corresponding curve of the filter under consideration. The system checks that and passes the appropriate action to the procedure as a parameter called **action**. Action is either *reflect* or *transmit*.

The procedure is:

---

algorithm $interference - filter(I_{in}, I_{out}, I_{d_0}, I_{d_1}, n_1, action,$ attributes)

begin

if $(action = transmit)$ **then**

$I_{11} = I_{10} \times T$

$pos_{11} = transmit - out - pos - normal(pos_{10}$, thickness of the filter, refractive index of the filter, refractive index of the medium where the beam comes from, direction of the face of incidence, orientation of the normal to the face of incidence)

$d_{11} = d_{10}$

$distance = thickness$

$\varepsilon_{11} = \varepsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

else

$I_{11} = I_{10} \times R$

$pos_{11} = pos_{10}$

$d_{10} = -d_{11}$

/* the negating sign means that the exact opposite direction is taken by the output. This means that we have total reflection */

$\varepsilon_{11} = \varepsilon_{10}$

endif

end

4.17 Liquid Crystal SLMs

The different actions within this component are taken depending on whether the input beam is a *read* or *write* one. This is detected by the system and passed to the procedure as a parameter ($beam - type$). The beam is assumed to have a normal incidence on the face of incidence.

The procedure is:

---

algorithm $liquid - crystal(L_{in}, L_{out}, L_{d_0}, L_{d_1}, beam - type$, attributes)

begin

if $(beam - type = write)$ **then**

let θ_{twist} be the angle of allignment of the LC molecules after applying the write beam

/* the write beam affects the electric field which affects the θ_{twist} */

$I_{11} = I_{10} \times T_{write}$

$pos_{11} = pos_{10}$

$distance = 2 \times thick_{write}$

$\varepsilon_{11} = \varepsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

else

let θ_{rotate} be the angle of rotation of the polarization plane of the input beam depending on θ_{twist}

$pol_{11} = pol_{10}$ rotated by 90°

$I_{11} = I_{10} \times T_{read}$

$pos_{11} = pos_{10}$

$distance = 2 \times thick_{read}$

$\varepsilon_{11} = \varepsilon_{10} + ([(distance)/\lambda_{10}] - INTEGER([(distance)/\lambda_{10}])) \times 2\pi$

endif

end

4.18 Intensity Degredation with distance

As the light beams propagate in media, they loose part of their energy (power) to the medium of propagation. This loss of energy is formulated, as discussed in Chapter 2, as follows:

$$\frac{I_2}{I_1} = \frac{r_1^2}{r_2^2} \tag{4.1}$$

where, I_2 is the intensity of the beam at distance r_2 from the source. I_1 is, likewise, the intensity of the same beam at distance r_1 from the source.

The system takes the intensity of the light beam at a distance equal to 1 millimeter from the source to be:

$$I_1 = \frac{Power}{2\pi w_0^2} \tag{4.2}$$

where *Power* is the power of the output beam from the source and w_0 is the radius of the beam at the source. Afterwards, and as the beam travels, the distance it travels is computed till there exists a need to compute the intensity of the beam again. When such a need arises, the system uses Equation 4.1 to compute the intensity at the position of computation. This equation gives the intensity of the light beam after deducting the energy it lost to the medium. If the need to compute the intensity arised from the presence of a component that effects the intensity, the computed value at the new distance is used to compute the effect of the component on the intensity.

Chapter 5

PLOADS

PLOADS (a Programming Language for Optical Architectures Description or Specification) is a programming language with a top-down hirarchical design. **PLOADS** is a tool for optical architecture designers. It can be used to verify the correctness of the optical architecture design. Though, it does not assure it.

PLOADS is a simulation system that simulates the behaviour of optical architectures. It simulates the whole architecture by simulating individual components.

The components library of **PLOADS** contains a number of widely used components. Tthe library is open ended in the sense that it could be expanded to contain additional components.

For these purposes, adequate constructs and data structures are made available for the **PLOADS** users. In the following we shall investigate the language and its syntax and semantics.

The highest level of specification of any architecture is of the form:

<program> ::= MAIN ['{'<component_modification>'}'] '{'<declaration>'}'

'{'<description>'}' '{'<action>'}' **END** ';'

The four sections or parts of a **PLOADS** program are discussed in the rest of the chapter. We found it more suitable to leave the discussion of the *components modification* part to the end of this chapter.

5.1 The Declaration Part

PLOADS is a **strongly-typed** language. No variable could be used in programs unless declared first at the declaration part. Functions called from other parts of the program have to be defined at this part as well. The constructs (statements) allowed in this part are:

1. The ***declaration statement***, where a number of variables could be declared a certain type and initialized to certain values. For example,

   G : FLOAT;
   i, I, KLM : INT = 5;
   M : **ARRAY**[6] **OF** CHAR;
   S, y : **ARRAY**[1..3, 2] **OF** LOGICAL = ((TRUE, FALSE),
   (FALSE, FALSE), (TRUE, TRUE));

   The initialization of arrays is done row by row.

   We will discuss the declaration types later in this section.

2. The ***type statement***, where new types could be created. These new types could be used locally only (within the declaring program). For example,

TYPEDEF

{

x, F = INT;

MONTH = (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP,

OCT, NOV, DEC);

}

y : x;

MON : MONTH;

In the above, x, F, and MONTH have become new types and they could be used to declare other variables/identifiers.

3. The ***function definition.*** This construct has the following structure:

FUNCTION <function_name> '('[<parameter_list>')']

RETURN <declaration_type> ['{'<declaration>'}']

'{'<function_body> '}'

The basic declaration types that could be used are:

INT for integers

FLOAT for floating point

LOGICAL for logicals

CHAR for characters. A string is considered to be an array of characters

POSITION for position variables. Its a tripple of real values. Any position is taken in the three-dimensional space. Its unit is the centimeter

FACE_DIRECTION for face direction variables

ORIENTATION for the orientation variables. Its unit is the degrees

POLARIZATION for the polarization variables. They take values that represent the state of polarization (unpolarized, plane polarized, circularlly polarized, ..., etc.)

TIME for time variables. It is usually taken in seconds

POWER for the variables representing the power of a certain light beam. The unit for power is milliwatt

WAVELENGTH for variables that represent wavelengths of light beams. Its unit is nanometer

FREQUENCY for variables representing frequencies of light beams. Hertz is the unit for this type

ANGLE for variables representing angles. The angle values $\in [0, 180]$

PHASE for the phase variables. It's values $\in [0, 2\pi]$ or $[0, \lambda]$, where λ is the wavelength of the light beam

ARRAYS of one or two dimensions of any other declared type

component types is a set of types given to variables used to hold components identifications. These types are:

- CONTINUOUS_LASER
- PULSED_LASER

- BEAM_SPLITTER
- FLAT_MIRROR
- WINDOW
- LENS
- PRISM
- GRATING
- FILTER
- BIREFRINGENT_POLARIZER
- POLARIZING_BEAM_SPLITTER
- RETARDATION_PLATE
- DICHROIC_SHEET
- SPATIAL_LIGHT_MODULATOR
- PRISM
- GRATING

LASER_LIGHT for variables holding information that is passed as input to assemblies or as output from them

5.2 The Description Part

In the description part of the program, the programmer is concerned mainly in placing components of the architecture being described. It is the main part of the program and usually the longest. The model of this part is:

<description> ::= '{' { <all_placements> } '}' ['{'<constraint>'}']
<all_placements> ::= { <placement> }
| { <assembly_definition> }

The description part is divided into two main subparts discussed in the following subsections.

5.2.1 Placement

In this subpart, the programmer assigns the values of attributes of the components that make up the architecture being described. The program has to specify fully the functionality of the described component by specifying the values of its attributes. For example, in order to describe a lens placed at the three dimensional position (0, 5, 0), with its face of incidence towards the negative side of the $x - axis$, the programmer should follow these steps:

1. In the declaration part the following has to be typed:

   LEN1 : LENS;

2. In the description part, the following should appear:

   LEN1.position := (0, 5, 0);
   LEN1.face_direction := (LEFTX, UNDEF, UNDEF);
   LEN1.orientation := (0,90);

   In the above, it should be noted that $LEN1.orientation$ is not necessary to be given, since the face of incidence is perpendicular to the $xy - plane$. This implies that the normal to the face of incidence has to be parallel to

the $xy-plane$ (0, 90). The above segment of the description part is not yet enough. All of the attributes of LEN1 have to be specified (assigned values). This could be accomplished using the *modify attribute* statement, which has the following structure: (A note first before we get into that is that we use the term *object_id* to mean component_id, which is what we called $LEN1$)

<modify_attribute_statement> ::= <object_id> '.' <identifier>
':=' <expression> ';'
| <object_id> '.' <identifier> ':=' <object_id> '.' <identifier>
| <object_id> '.' <identifier> ':=' <identifier> ';'

The following are the **PLOADS** statements that should appear in the placement part to describe a plano-convex spherical lens.

LEN1.type := spherical_plano_convex;
LEN1.diameter := 15.0;
LEN1.r1 := 10.3;
LEN1.tc := 1.5;
LEN1.te := 2.0;
LEN1.index := 1.310;
LEN1.fl := 20.0;
LEN1.bfl1 := 22.0;
LEN1.bfl2 := 23.0;
LEN1.transmittance := 96.3;
LEN1.coat_index := 1.67;
LEN1.permeability := 1.0;

Thinking of all the pain the programmer has to go through to assign values for individual attributes of individual components, and keeping in mind that optics researchers and experimenters get components from manufacturers, we designed a database called the *component database*. How the user benefits out of this database is going to be cleared in what follows.

Component Database

This database contains information about the components available for the use of experimenters and researchers. As mentioned earlier, researchers get optical components from manufacturers. They rarely use customized components i.e., those are components that researchers order with special characteristics that are not available in catalogues). This does not mean that customized components cannot be dealt with in the scope of the designed database.

The database is made up of a number of files, one for each component. The attributes of each file are the attributes of the corresponding component (see Chapter 3). Each record in the file is for a different model of the component and it may have different values for its attributes.

For example, a file for continuous laser sources is given in Table 5.1. In the table, attributes such as position and orientation have to be adjusted by the user (programmer) and hence are not present in the file. Their presence with dummy values would unnecessarily waste space.

Therefore, all what the programmer has to do is to select a model from the components file and use its index/reference number. This implies that all of the attributes values of that model will be assigned to the component/object_id used. This saves the time and effort of the programmer considerably. It also saves a lot

model number	beam diameter (mm)	power or power range	frequency or longitudinal mode spacing	dimensions	polarization ratio
CL001	5.0	3.0	500	11, 20, 30	0
CL002	4.3	15.0	1800	50, 20	1:500
CL003	7.5	50.5	480	11, 20, 40	0

Table 5.1 The continuous laser source file in the components database.

of coding on the part of the programmer/designer.

The interface with the components database is conducted via the *object manager*. It obeys the commands of the system with respect to the components database. Therefore, it could also be called as the *database manager*. In the following, we discuss a number of constructs via which the user/programmer can use and manipulate the database. The system interprets these constructs and passes the requests to the object manager to be satisfied.

The constructs that are provided for the users to enable them access the data base are discussed in what follows.

These constructs are a number of functions that are specially used for the components database. These functions are:

- The **get_object** function. When used, it fetches a record from the database and assigns all of its attributes to the assigned *object_id*. For example, take the case of the spherical lens specified above. We can specify it, instead, using the function get_object as follows:

  LEN1 := **GET_OBJECT**(LEN054);
  LEN1.position := (0, 5, 0);
  LEN1.face_direction := (LEFTX, UNDEF, UNDEF);
  LEN1.orientation := (0,90);

  LEN054 in the above is the index of the appropriate record of the lens's file within the components database. The get_object fetched that record and assigned all of its attributes values to the object_id LEN1. The three other modify attribute statements specified the place and orientation of the LENS LEN1 in the architecture.

As noticed, using this function saved *eleven* lines of code, but at the same time, required that the designer/programmer take a look into the database to select the model that suits his needs. The programmer can still assign his own values to the component he has created by the declaration statement. To do that, he has to use the modify attribute statement.

- The **copy_object** function. This function copies the attributes values of one component into another. It was included to relieve the system from unnecessary calls to the components database. Therefore, if two components are of the same type and have similar characteristics (attribute values), the user has to first use the *get_object* function for the first one and the *copy_object* for the second. The *copy_object* does not call the components database. This understanding could be expanded to cover for *arrays* of components. In that case, one can *get* the first one and *copy* the rest using a loop of *copy_object* function calls. For example, an array of six lenses might be needed in a certain architecture. For that, and in the declaration part, the following has to be show:

```
LEN : ARRAY[6] OF LENS;
```

In the description-placement part, on the other hand, the following statements are necessary:

```
LEN[1] := GET_OBJECT(LEN045);
FOR i = 2, 6
      {
        LEN[i] := COPY_OBJECT(LEN[1]);
      }
```

- The **get_default** function. It might be the case that no model in the database fully satisfies the needs of the experimenter /programmer. Still, one of those has some attributes values that the programmer finds appropriate for his needs. In this case, and instead of fetching that corresponding record all together, he can selectively fetch individual attributes using the *get_default* function. For example, one can say:

  LEN2 : **GET_DEFAULT**(LEN024,fl, bfl1, r1);

- The **get_attribute** function. This is a function that is called by the system when the following construct is found in any **PLOADS** statement:

  <object_id> '.' <identifier>

  For example, the statement:

  focal := LEN1.fl;

  calls the *get_attribute* function and passes it the object_id, which is the identifier declared to be of certain component type, and the attribute name (LEN1 and fl in the above). The function then returns the value of the attribute for the specific referenced object.

Two different statements are used to modify the components database. They are:

- The **add_default** statement. It is used to add a record to any file in the database. For example, if a lens was provided by a manufacturer with new specifications, the user might want to add it to the database. For that the following model has to be used:

ADD_DEFAULT '(' <object_id> ')' ';'

This statement cannot be used directly. The programmer has to go through two steps before it. The first is the declaration of an identifier/variable of type LENS (to go with the example above). The second step is the assignment of the attributes values of the new lens using the *modify attribute* statement. Then the *add_default* statement could be used. The following is the detailed process. In the declaration part:

LEN : LENS;

In the description-placement part:
LEN1.type := spherical_plano_convex;
LEN1.diameter := 15.0;
LEN1.r1 := 10.3;
LEN1.tc := 1.5;
LEN1.te := 2.0;
LEN1.index := 1.310;
LEN1.fl := 20.0;
LEN1.bfl1 := 22.0;
LEN1.bfl2 := 23.0;
LEN1.transmittance := 96.3;
LEN1.coat_index := 1.67;
LEN1.permeability := 1.0;

The new record will get an index number which is greater than the index number of the last record in the file by one. For example, if the last record

in the LENS files above had an index number *LEN*178, the new record will have *LEN*179 as its index number.

- The **add_attribute** statement. This statement adds to the flexibility of the language, and hence, of the whole system. It allows the programmer to add an attribute to individual declared objects (object_id's) according to his needs. For example, if the programmer at some point needed to add an attribute called *number_of_coats* to one of the mirrors declared in the program, the following has to show in the program: In the declaration part:

  M1, M2 : LENS;

  In the description-placement part, on the other hand the following two statements have to be present:

  M1 := **GET_OBJECT**(FM032);
  ADD_ATTRIBUTE(M1, number_of_coats, INT, 20);

  INT in the above declares the new attribute to be of integer type. On the other hand, 20 is the default value of the new attribute.

That concludes the talk about the components database, where function and constructs that use and manipulate it were introduced. We now move to other constructs that could be used within the placement part.

- The **assignment** statement. It has the following construct:

  <assignment_statement> ::= { <identifier> ':=' } <non_action_expression> ';'
  | { <identifier> ':=' } <object_built_in_function> ';'

| { <identifier> ':=' } <identifier> ';'

| { <identifier> ':=' } '(' <parameter_list> ')' ';'

The parameter list in the above is just a set of identifiers, each two of which are separated by a comma. Let us now give a detailed description of what *non_action_expressions* are:

1. Arithmetic expressions. They are similar to their counterparts in other languages. They use the usual operations such as, +, −, *, /,' and DIV. The fifth operation is for exponentiation and DIV is for integer division. Arithmetic expressions could have function calls as well.

2. Logical expressions. They have comparison operations such as, <, >, <=, >=, ==, !=. The last operation is for non-equality. Logical expressions could also contain *connectors* such as, && (AND), || (OR), ! (NOT).

3. String expressions. They have two operations: the concatenation operation (//) and the substring operation of the form:

   <identifier> '['<arithmetic_expression> ':' <arithmetic_expression> ']'

4. Face expressions. They are either face_direction constants or their negations. **FACE_MIN** has the effect of reversing all of the elements of the face_direction it negates. For example the

   FACEMIN (LEFTX, BACKY, DOWNZ)

   is

   (RIGHTX, FORTHY, UPZ)

   This expression has the following construct:

<face_expression> ::= <face_direction>

| <face_declarator>

| <identifier>

| FACE_MIN <face_expression>

<face_declarator> ::= <object_id> '.' face_direction

5. Position expressions. They have the followig construct:

<position_expression> ::= <position_declarator>

| <position_constant>

| <identifier>

| POSUNMIN <position_expression>

| <position_expression> POSADD <displacement>

| <position_constant> POSMUL <position_expression>

| <arithmetic_expression> POSMUL <position_expression>

<position_declarator> ::= <object_id> '.' position

The operation POSUNMIN negates all of the elements of the position expression on its right hand side. For example,

POSUNMIN (-20, 30, 0) is evaluated to (20, -30, 0)

The operation POSADD, on the other hand, adds a certain displacement to the position expression on its right hand side. For example,

(-20, 30, 0) POSADD (RIGHT 20, BACK 20, UP 10) is evaluated to (0, 10, 10)

The displacement takes the following construct:

<displacement> ::= '(' <x-direction> <displace> ','
<y-direction> <displace> ','
<z-direction> <displace> ')'

<displace> ::= /* nothing */
| <arithmetic_expression>

The x-direction can be either RIGHT, LEFT, or NONE. The y-direction, on the other hand, is either FORTH, BACK, or NONE. The z-direction, finally, is either UP, DOWN, or NONE. The POSMUL operation multiplies a position expression by an arithmetic expression, which implies that each of the coordinates of the expression is multiplied by the same value. The multiplier can also be a position constant, which means that each coordinate of the expression gets multiplied by a different value from the constant.

6. Orientation expressions. The following are their constructs:

<orientation_expression> ::= <orientation_declarator>
| <orientation_constant>
| <identifier>
| ORUNMIN <orientation_expression>
| <orientation_expression> ORADD
<angular_displacement>

<orientation_declarator> ::= <object_id> '.' orien

<angular_displacement> ::= '(' <arithmetic_expression> ','

<arithmetic_expression> ')'

The operation ORUNMIN will add the complements of the angles of the orientation to their corresponding angles. The complement is that of the angle 90°). For example,

A := ORUNMIN (0, 90) will give A the value (90, 0)

The ORADD operation, on the other hand, adds the corresponding components of the orientation. For example,

B := (50, 30) ORADD (-20, 10) will give B the value (30, 40)

We shall talk about the parameter list assignment statement when we discuss the *assembling* of optical architectures.

- The **with_object** statement. It is used to assign values to attributes of a certain component. It has the following construct:

<with_object_statement> ::= **WITH** <object_id> '{' <parameter_list> '}'
'{' <expression_list> '}'

<parameter_list> ::= <identifier>
| <parameter_list> ',' <identifier>

<expression_list> ::= <non_action_expression>
| <expression_list> ',' <non_action_expression>

This is quite compact when compared with the equivelant set of assignment statements that could be used. For example, the following attributes of the component LEN1 can be assigned as follows:

WITH LEN1 { fl, tc, position } { 14.7, 5.4, (0, 90, 10) }

- The **compound** statement. It is simply a group of statements. It has the syntax:

  <compound_statement> ::= '{' [<declaration>] '}' '{' <placement> '}'

  It is usually used as part of other statements as will be shown later.

- The **control** statement(s). A number of those are supported by **PLOADS**. The following are their formal constructsthem.

  1. We start with the *conditional IF* statement, which can take one of the following two forms:

     IF '(' <logical_expression> ')' <compound_statement>

     or

     IF '(' <logical_expression> ')' **THEN**

     <compound_statement>

     ELSE

     <compound_statement>

     One condition is to be necessarily met here is, that the condition of the IF statement has to be statically evaluated, if the compound statement contains statements related to placement of components. Therefore, the condition cannot wait till actual simulation of the architecture to be evaluated. That is, it cannot relate to the characteristics of light beams after any component other than their source. Hence, the conditions of IF statements have to be realized before execution time *if their compound statements contain components placement statements.*

  2. For iterative processing, three different constructs are presented.

(a) **WHILE** '(' <logical_expression> ')' <compound_statement>

(b) **DO** <compound_statement> **WHILE** '(' <logical_expression> ')' ';'

(c) **FOR** <identifier> '=' <arithmetic_expression> ','
<arithmetic_expression> [',' <arithmetic_expression>]
<compound_statement>

The third arithmetic expression is the increment for the value of the identifier and is optional. In case of its absence, the default increment of1 is taken.

- The **break** statement. It could be found anywhere within the program. When it is executed, it stops the execution of the program. It looks as follows:

  BREAK ';'

- The **goto** statement. It transfers control to another statement within the same program. It gives programmers the freedom of moving back and forth within the program. It has the following form:

  GOTO <label> ';'

  The label is an identifier that labels another statement.

- The **labeled** statement. It has the form:

  <label> '::' <labeled_statement>

The labeled statement could be any statement other than the compound statement.

- The **modify attribute** statement. It was discussed earlier.

- The **medium** statement. It declares two of the characteristics of the medium of the optical setup. It has the form:

  MEDIUM '(' <arithmetic_expression> ',' <arithmetic_expression> ')' ';'

  The first arithmetic expression represents the refractive index of the medium. The second represents its permeability. The statement is used whenever a medium other than air is used within the architecture. For example, a box of liquid might be part of the architecture. This box has entrance and exit windows for light to pass through. The medium as a whole has to first be defined by the *media_definition* construct (discussed later). Then, within the description part of the *media_definition*, the *medium* statement of the above form has to be used. If the *medium* statement is absent, the medium is taken to be air.

- The **media_definition** program segment. Let us consider the following case for example. Assume that the light beam at some position within the architecture has to enter a box of water. The entrance to and exit from the box is via windows. There is a possibility of components residing inside the medium (box of water). The program segment equivalent to the above is as follows:

  Within the <placement> part:

```
{
  MEDIA water (BOX, (50.5, 32.0, 15.0), (135, 47.2, 39),
  (LEFTX, FORTHY, UNDEF), (45, 90))
  {
    W_IN, W_OUT : WINDOW;
/* declaration of other components within the medium is given here
as well */
  }
  {
    {
      MEDIUM(1.5, 1.22);
/* description of the windows and the other components should be given here */
    } /* end of the placement part */
  } /* end of the description part */
} /* end of the media definition segment */
```

In the previous, *water* is the media name, *BOX* is the media type (it could be a CYLINDER as well), the first triple represents the dimensions of the box (it would have been a pair if the media type was a cylinder), the second tripple is the position of the center of the box (or cylinder), the third is the direction of the base of the box (it is not needed in the case of a cylinder), the last pair is the orientation of the normal to the box's base (in the case of a cylinder it is the normal to both the ends of the cylinder).

- The wait statement. It enforces a delay of a certain amount of time at a certain position. It has the following construct:

WAIT_FOR <arithmetic_expression> **AT** <position_expression> ';'

These are the basic constructs available for use in the placement part. Now we move to the concept of *assembling* optical architectures.

The Assembly construct

If the user finds the architecture to be large or repetitive to describe, he can use the modularization facility provided by the language. He can divide the architecture into a number of assemblies. Assemblies are either called from the placement part of the main program or from other assemblies in the same program. *Recursion is not allowed in assemblies.*

Calling the assembly causes the realization of the subarchitecture, that the assembly describes, as part of the overall architecture.

The assembly definition of some part of an architecture might look as follows:

Within the <placement> part:

```
{
  ASSEMBLY interference ( delta, INPUT :
  BS1, M1, OUTPUT : BS8, M3, M5)
  {
    /* the declaration part of the assembly */
  }
  { /* the description part */
    { /* the placement part */
      /* description of the components should be given here */
    } /* end of the placement part of the description */
```

```
/* a constraints part (discussed later) could be added */
  } /* end of the description part */
} /* end of the assembly definition */
```

In the above, *interference* is the name of the assembly. Any assembly that exists can be imagined to reside in a box. To place the assembly at different positions and to give it different directions, we give the direction of the base of the box (Figure 5.1) and the orientation of the normal to it. We also give the position of the corner of the box. All of that is given in the calling program or assembly.

To understand exactly what goes on, the reader has to think of the assembly as an architecture that is realized. The programmer might want to include this assembly in the architecture he is describing. To do that the above parameters have to be identified in the calling program before calling the assembly (calling it is just realizing it). To identify these parameters which are similar to those of any component, we use the ***modify attribute statement*** to set the position, direction and orientation of the assembly within the architecture. With this respect, the assembly could be thought of as a component with the shape of a box. This component has three attributes and a detailes description PLOADS program.

Now, no matter what the direction of this box is, the components of the assembly should be placed within the box irrespectively. For that, we consider the box to reside in the positive three-dimensional space, as shown in Figure 5.1. One of the components of the assembly **at least** has to have a position relative to the position (0, 0, 0) (Figure 5.1).

For example, in the assembly *interference* of Section 6.5, M1 could be given

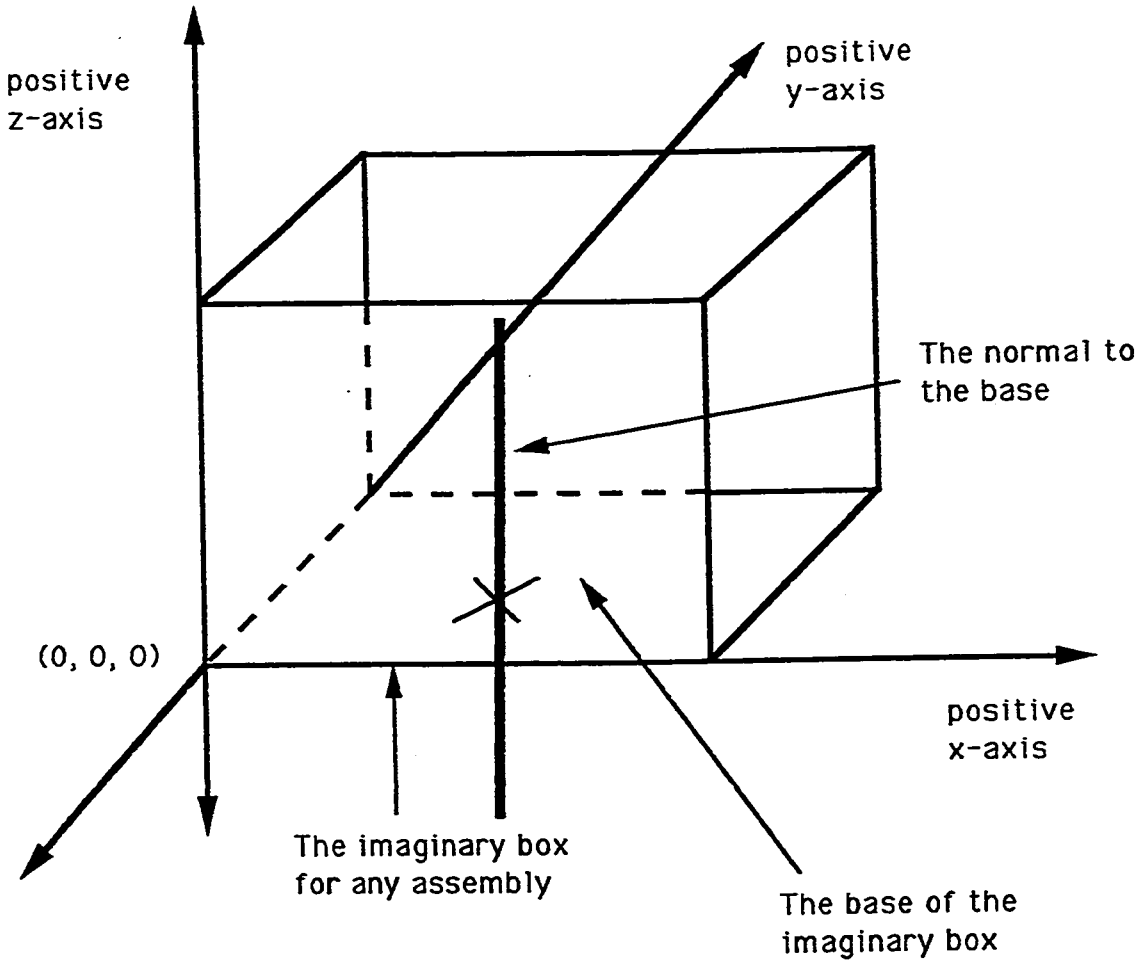


Figure 5.1 The assembly's direction. The assembly's components are assumed to reside in the shown box.

a position relative to (0, 0, 0). Since M1 is the bottom left most element, it could be given the position (0, 0, 0) itself. The direction of M1 does not depend on the direction of the overall assembly (box).

Input parameters such as *delta* are optional and their number, type and order is left totally to the programmer.

Next, two classifications of parameters/arguments should appear. The first one is *INPUT* parameters. These follow the word **INPUT** and they are components names declared within the assembly, where input light beams to the assembly are expected. The second is *OUTPUT* parameters. They are as well components names declared in the assembly that come after the word **OUTPUT**, where the beams coming from which go as output from the assembly to the outer world.

To call an assembly from the placement subpart of any other assembly or from the main program, the following statements could be used:

interference.position := (20, -40 0);
interference.face_direction := (UNDEF, BACKY, UNDEF);
interference.orientation := (90, 90);
ASSEMBLY interference(delta, inter_in1, inter_in2,
inter_out1, inter_out2, inter_out3);

delta in the same as that of the assembly definition. The reader can notice that before any attempt was made to call the assembly, its attributes were set. The other arguments such as inter_in1 have to be of type **LASER_LIGHT**. A variable of this type can be assigned variables that are declared to be of a laser source type. It could also be assigned other assemblies' outputs. For example, for the inputs of the assembly *interference*, the following statements could be found in its calling program or assembly :

```
{
  /* a segment of the declaration part of the calling program */
  LSX, LSY : CONTINUOUS_LASER;
  inter_in1, inter_in2, inter_out1, inter_out2 : LASER_LIGHT;
  inter_out3, comp_in1, comp_in2, comp_out1 : LASER_LIGHT;
}
{ /* the description part */
  { /* the placement subpart */
    /* after some placement statements or at the beginning
    pos1 := (0, 30, 5);
    dir1 := (UNDEF, BACKY, UNDEF);
    orient1 := (90, 90);
    LSX := LSY := GET_OBJECT(CL023);
    inter_in1 := LSX;
    inter_in2 := LSY;
    ASSEMBLY interference(pos1, dir1, orient1, delta, inter_in1,
    inter_in2, inter_out1, inter_out2, inter_out3);
    pos2 := (67, 30, 10);
    dir2 := (UNDEF, FORTHY, UNDEF);
    orient2 := (90, 90);
    comp_in1 := inter_out1;
    comp_in2 := inter_out3;
    ASSEMBLY compare(pos2, dir2, orient2 comp_in1, comp_in2,
    comp_out1);
    /* the rest of the placement part, if any */
```

```
  } /* end of the placement part */
} /* end of the description part */
```

5.2.2 The Constraint Subpart

It is usually hard to keep track of the positions of individual components in big architectures. Therefore, we provided the *relative placement* facility as an optional part of any **PLOADS** program. For example, instead of knowing the exact position of a certain component, the programmer can just detect the distance between it and some other component of known position. This also applies on both orientation and face direction.

The constraint subpart places constraints on components positioning and direction issues as well as constraints on the space occupied by the described architecture. Within this subpart, the following statements are allowed:

```
{ /* the constraints subpart */
  LEN2.position := BS.position POSADD (RIGHT 10, NONE, NONE);
  LEN1.position := (0, 80, -30);
  LEN3.position := POSUNMIN LEN1.position;
  M1.orientation := LEN2.orientation;
  LEN1.orientation := ORUNMIN M1.orientation;
  LEN2.face_direction := (RIGHTX, UNDEF, UNDEF);
  M2.face_direction := FACEMIN M1.face_direction;
  ARCHITECTURE_DIMENSION((0, 0, 0), (300, 400, 100));
} /* the end of the constraints subpart */
```

The last statement in the above places a constraint on the dimensions of the architecture of which it is a part (it could have been an assembly). The two triples

in the statement are positions of the corners of the diagonal of the *imaginary* box containing/enclosing the architecture.

5.3 The Action Part

In this part, the programmer can probe for information resulting from simulating the described architecture. The user is allowed to ask about the properties/characteristics of the light beam at any position at any time. He is also allowed to get the value of any attribute of any component. The **print** statement (discussed later) can then be used to print any of the probed/sampled values.

The characteristics of the light beam are:

POW for power

POLAR for polarization

BEAM_DIAMETER

PHAS for phase

WAVE for wavelength

FREQ for frequency

Therefore, the following statements could be used to print some of the characteristics of a light beam:

```
{ /* the action part */
  pos3 := (0, 0, 0);
  p := POLAR AT (0, 30, -20) AT 30.5;
```

```
  t := 25.3;
  i := POW AT (20, 0, 10);
  il := POW AT LEN1.position POSADD (RIGHT 10, NONE, UP 5);
  J := BEAM_DIAMETER AT pos3 AT t;
  PRINT, p, "polarization with ", i, "and" il, "intensity at: ", pos3;
} /* the end of the action part */
```

The formal model for these action assignment statements is:

{ <identifier> ':=' } <characteristics>
AT <position_expression> [AT <arithmetic_expression>] ';'

The arithmetic expression at the end represents the time at which the probing should take place.

In the action part, also, the user can ask the system to draw a diagram of the overall architecture, or the assembly's architecture, by using the *draw_diagram* statement which has the construct:

DRAW_DIAGRAM <identifier> ';'

The identifier is the name of the main program or assembly for which the diagram is to be drawn.

In the *action* part, the user cannot use the components database exscept via the *get_attribute* function. Other constructs such as the *compound* statement and the *control* statements are allowed in this part.

In the *action* part the designer/programmer can also select the time unit that the system should use for its computations. The system is designed by default to compute different characteristics of light beams every one second. The

programmer can change that to any other value. For example, the programmer can select the time_unit for the computations of his architecture using the *time_unit_statement*, which has the model:

<time_unit_statement> ::= TIME_UNIT '=' <arithmetic_expression> ';'

Therefore, to make the time_unit to 1 minute, the programmer has to write:

TIME_UNIT = 60;

which implies that the computations will be conducted every 60 seconds. Let us now consider one of the most important features of **PLOADS** , the **components modification** capability.

5.4 The Components Modification Part

The components library described in Chapter 4 contained procedures for a number of heavily used optical components. This alone does not make it complete. A feature or construct that enables users to add new components has to be provided. These constructs are discussed in this section.

In Chapter 3, a model of each component was developed. Modeling of such components was accomplished by specifying the attributes that fully describe their behaviour. These attributes might change due to any technical or physical reason such as changing the shape of the component all together. This asks for adding new attribute(s) to cover for the changes on the model of the component. Adding components or components attributes can be done within this part.

Let us start first with adding an attribute to a component. For example, if the **weight** and **aperture area** of beam splitters are attributes that should be added to their model, then the following statement has to appear in this part:

LIB_ADD_ATTRIBUTE(BEAM_SPLITTER, weight, FLOAT, 0.23,
apper_area, FLOAT, 25.0);

which implies that the attribute called *weight*, of type FLOAT and default value of 230 grams, is added to the attributes of the beam splitters. Another attribute called the *apearture area* is also added, with type FLOAT and default value 25 squared centimeters. These attributes and their values are added to all the records of the beam splitter's file in the components database.

If the addition of the attributes enforces a change on the procedure that describes the component's behaviour in the components library, a new procedure is given by the user right after the above statement.

To add a new component to the components library and of course to the components database, we need to use the following construct:

ADD_COMPONENT '(' <identifier> ')' ';'

where the *identifier* represents the name of the type of the new added component. After that, the attributes of the component have to be defined by using a number of the *LIB_ADD_ATTRIBUTE* statements. At the end, the procedure to be added to the components library has to be given by the programmer. This is the modification routine present in the formal description of the language. In the modification routine, constructs such as the *compound* statement and the *control* statements could be used to build up the routine. The routine has the construct:

<modification_routine> ::= '{' <routine_declaration> '}' '{' <routine_body> '}'

Adding an attribute to some existing component will, automatically add this attribute to all the records in the corresponding file of the component in the components database. Adding a new component, on the other hand will create a

new file in the database for the new component. create a file in the components database for the new componnent This concludes the tour in **PLOADS**. A formal description of **PLOADS** in the **BNF** form is presented in the Appendix.

Now let us talk about general constructs, constructs that could be found in more than a part in the **PLOADS** program.

5.5 General Constructs and Considerations

Constructs such as the following could be found in any part of the **PLOADS** program:

1. The **comment** statement. It is an unexecutable statement that is enclosed by /* and */.

2. The **print** statement. It is an output capability that we offer. It has the following construct:

   PRINT ',' <printed_list> ';'

   The printed list could be either variables, expressions, constants, and/or strings enclosed by quotations.

One aspect of the language is that it is *case sensitive*. This could be noticed clearly in the examples given throughout this chapter.

Chapter 6

DESCRIPTION OF OPTICAL ARCHITECTURES

In order to verify the expressive power and suitability of **PLOADS**, we investigate a number of optical architectures and describe them using PLOADS. We chose architectures that fully utilize the power of **PLOADS**. Several were described as will be shown in the rest of the chapter.

In the following sections, we will present the selected optical architectures together with the description programs in **PLOADS**.

6.1 APPLICATION I: A Synchronous Trip-Flop

In this section we describe the optical architecture proposed in [GUIZ90] using **PLOADS**. The architecture (Figure 6.1) represents the optical design of a **trip-flop**. In what follows, we present the reader with the program that describes the architecture in Figure 6.1.

In the figure, we give the assumed distances between the different components of the architecture. This will give the reader a feeling of what should be done as a first step before getting to the writing of actual code.

```
MAIN trip_flop
/* we start with the declaration part. All components and other variables
needed within the architect ure are declared */
{
  LS1 : CONTINUOUS_LASER;
  PL1, PL2, PL3 : PULSED_LASER;
  IF14, IF15, IF16 : FILTER;
  M1, M2 : FLAT_MIRROR;
  PBS1, PBS2, PBS3 : POLARIZING_BEAM_SPLITTER;
  BS : BEAM_SPLITTER;
  p : POLARIZATION;
/* p is a polarization variable, it will be used later to
save the polarization state of the light beam at a certain time
and position */
  t1 : TIME;
  i : POWER;
}
```

```
/* end of the declration part and the beginning of the
description part */
/* the description part is divided to two subparts,
the placement and action subpart */
/* first the placement subsection */
  {
    {
      LS1 := GET_OBJECT (CL023);
      LS1.power := 5.0;
      LS1.wavelength := 632.0;
      LS1.position := (0, 0, 0);
      LS1.face_direction := (RIGHTX, UNDEF, UNDEF);
  /* the orientation of the normal to the base of the source is
  not needed because the face of incidence looks right only,
  which means that the normal to it is parallel to the x - axis */
      LS1.laser_head := (0, -15, 0);
      IF14 := GET_OBJECT (F034);
      IF14.position := (0, -20, 0);
      IF14.face_direction := (UNDEF, FORTHY, UNDEF);
      PL1 := GET_OBJECT (PL007);
      WITH PL1 { power, wavelength, position, face_direction,
      laser_head, pulse_rate, pulse_duration }
      { 2.5, 632.0, (20, -30, 0), (UNDEF, BACKY, UNDEF),
      (-5, 15, 0), 60, 4.0 }
      PBS1 := GET_OBJECT (PBS005);
```

```
PBS1.position := (0, -30, 0);
PBS1.face_direction := (RIGHTX, FORTHY, UNDEF);
PBS1.orientation := (135, 90);
M1 := GET_OBJECT (FM003);
M1.position := (-20, -30, 0);
M1.face_direction := (RIGHTX, BACKY, UNDEF);
M1.orientation := (135, 90);
IF15 := GET_OBJECT (F004);
IF15.position := (0, -40, 0);
IF15.face_direction := IF14.face_direction;
IF16 := GET_OBJECT (F001);
IF16.position := (-20, -40, 0);
IF16.face_direction := IF14.face_direction;
PL2 := GET_OBJECT (PL001);
PL2.power := 3.0;
PL2.wavelength := 632.0;
PL2.position := (20, -50, 0);
PL2.face_direction := PL1.face_direction;
PL2.laser_head := (15, -50, 0);
PL2.pulse_rate := 30;
PL2.pulse_duration := 2.0;
PL2.polarization := P_PLANE;
PBS2 := GET_OBJECT (PBS009);
PBS2.position := (0, -50, 0);
PBS2.face_direction := PBS1.face_direction;
```

```
    PBS2.orientation := PBS1.orientation;
    PL3 := COPY_OBJECT (PL2);
/* this instruction copies all the attributes values of PL2 into
the object PL2 */
    PL3.position := (-40, -50, 0);
    PL3.polarization := S_PLANE;
    PBS3 := COPY_OBJECT (PBS2);
    PBS3.position := (-20, -50, 0);
    PBS3.face_direction := (LEFTX, FORTHY, UNDEF);
    PBS3.orientation := (45,90); M2 := COPY_OBJECT (M1);
    M2.position := (-20, -60, 0);
    M2.face_direction := FACEMIN M1.face_direction;
/* this instruction gives the object M2 a face of incidence
directed exactly in the opposite way from that of object M1 */
    M2.orientation := (45, 90);
  }
}
/* there is no constraint subpart. Therefore, we move to the
action part */
{
  t1 := 0.0;
  WHILE (t1 < 36);
            {
              p := POLARIZATION AT (0, -65, 0) AT t1;
              i := POWER AT (0, -65, 0) AT t1;
```

```
            PRINT "polarization of output at", t1, "nanosecond is", p;
            PRINT "power of output at", t1, "nanosecond is", i;
            t1 := t1 + 2;
          }
} END;
```

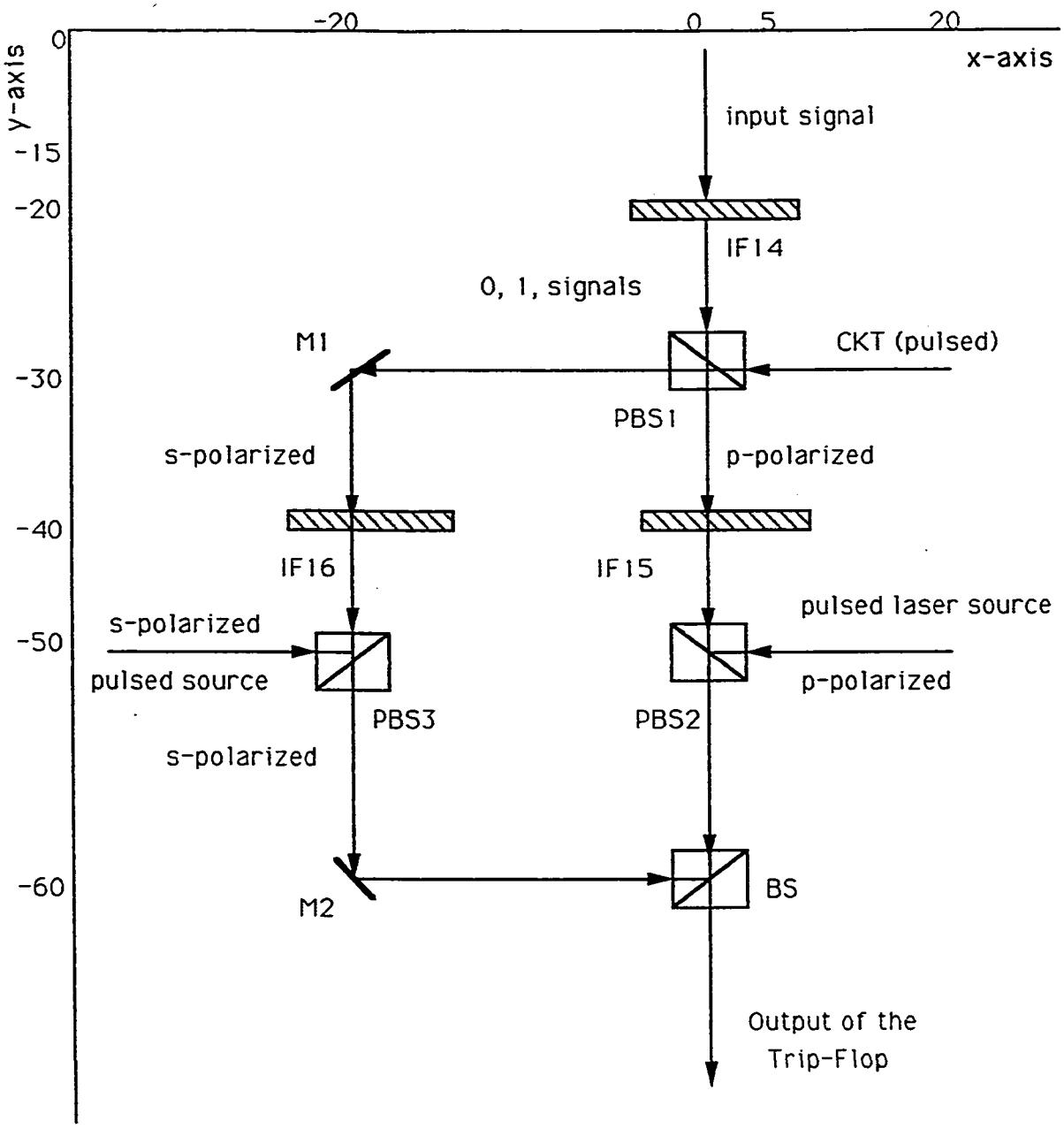


Figure 6.1 Optical design of a trip-flop

6.2 APPLICATION II: Banyan Network

We have studied the optical architecture proposed and tested by *Jahns* in [JAHN90c] (Figure 6.2). The following program is a description of this architecture using **PLOADS.**

In the program, as you will notice, we use the *constraints* part. One other important construct appears in the program which is the array of lenses (array of components or array of objects).

```
MAIN michelson_banyan_network
{
  LS : CONTINUOUS_LASER;
  LEN : ARRAY [4] OF LENS;
  QWP, HWP : RETARDATION_PLATE;
  M1, M2 : FLAT_MIRROR;
  PBS1, PBS2 : POLARIZING_BEAM_SPLITTER;
  M1, M2, M3 : FLAT_MIRROR;
  pos : POSITION;
  i : POWER;
  p : POLARIZATION;
  t : TIME;
  fl : FLOAT;
}
{
  {
    LS := GET_OBJECT (CL021);
    LS.position := (0, 0, 0);
```

```
    LS.face_direction := (UNDEF, UNDEF, DOWNZ);
    LS.power := 18.0;
    LS.wavelength := 633.0;
    LEN[1] := GET_OBJECT (LEN053);
    fl := LEN1.focal_length;
    LEN[1].position := (0, -fl, 0);
    LEN[1].face_direction := (UNDEF, FORTHY, UNDEF);
    PBS1 := GET_OBJECT (PBS020);
    PBS1.position := LS.position POSADD (NONE, DOWN fl, NONE);
    PBS1.face_direction := (RIGHTX, FORTHY, UNDEF);
    PBS1.orientation := (135, 90);
    LEN[2] := COPY_OBJECT (LEN[1]);
    LEN[2].face_direction := (LEFTX, UNDEF, UNDEF);
    LEN[3] := COPY_OBJECT (LEN[2]);
/* note here that LEN[2] takes all the attributes values of LEN[1]
and so does LEN[3]. Still they cannot take the same position.
Therefore, the system checks the constraints part before taking
any action regarding this situation */
    HWP := GET_OBJECT (RP004);
    LEN[4] := COPY_OBJECT (LEN[2]);
    PBS2 := COPY_OBJECT (PBS1);
    M1 := GET_OBJECT (FM007);
    M1.orientation := (150, 90);
    M2 := COPY_OBJECT (M1);
    QWP := GET_OBJECT (RP035);
```

```
    M3 := COPY_OBJECT (M2);
    pos := LEN[2].position;
    }
  /* now starts the constraints subpart */
  {
    LEN[2].position := PBS1.position POSADD (LEFT fl, NONE, NONE);
    LEN[3].position := PBS1.position POSADD (RIGHT fl, NONE, NONE);
    HWP.position := LEN[3].position POSADD (RIGHT fl, NONE, NONE);
    HWP.orientation := LEN[3].orientation;
    LEN[4].position := HWP.position POSADD (RIGHT fl, NONE, NONE);
    PBS2.position := LEN[4].position POSADD (RIGHT fl, NONE, NONE);
    M1.position := PBS2.position POSADD (RIGHT fl, NONE, NONE);
    M2.position := PBS2.position POSADD (NONE, DOWN fl, NONE);
    M2.orientation := M1.orientation ORADD (-30, 0);
    QWP.position := PBS1.position POSADD (NONE, DOWN fl, NONE);
    QWP.orientation := LEN[1].orientation;
    M3.position := QWP.position POSADD (NONE, DOWN fl, NONE);
  }
}
/* the end of description part and the beginning of the action
part */
{
  t := 100;
  pos := pos POSADD (LEFT 10, NONE, NONE);
  i := POWER AT pos AT t;
```

```
  p := POLARIZATION AT pos AT t;
  PRINT i, p;
}
END;
```

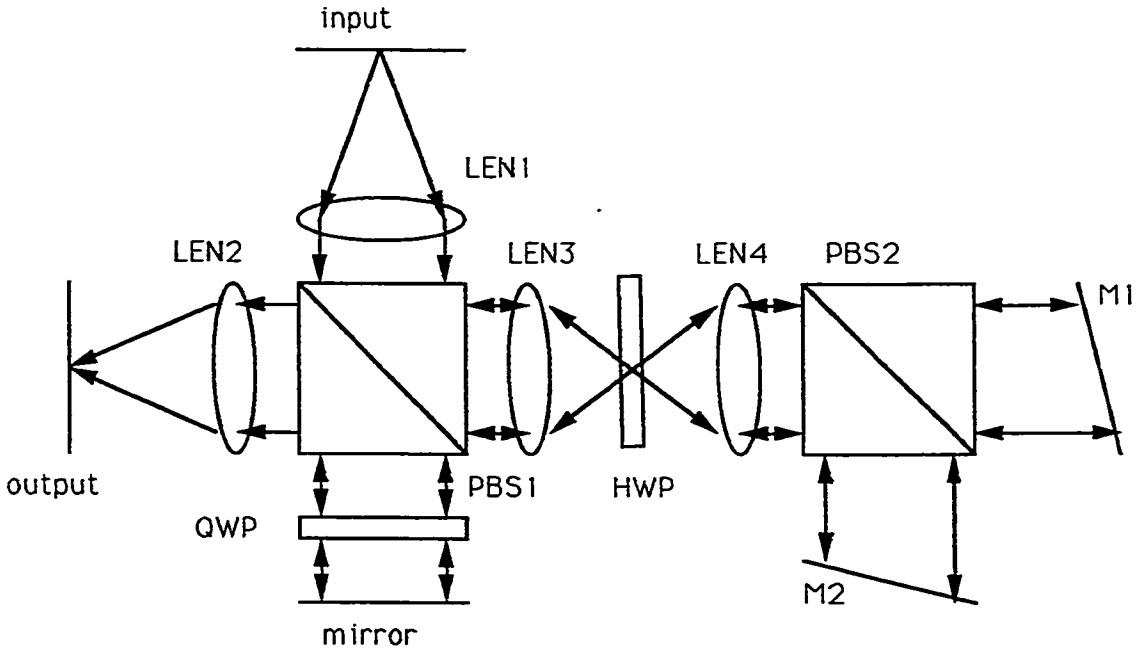


Figure 6.2 Michelson setup for the Banyan Network

6.3 APPLICATION III:Optical Processor for Addition and Subtraction

An optical architecture was proposed in [FUKU90] for the implementation of a parallel digital adder and subtracter. By digital we mean that the system is binary. Figure 6.3 shows the diagram for the architecture. The designer used *liquid crystal* spatial light modulators. The modulation of this type of SLMs is polarization.

The following is the description program for the architecture and there, we used arrays of different components. In the diagram, the reader can see an array of sensors (CCD) which is not necessary to be described since we can sense the output signals in the action part of the program. All the placement issues will be dealt with in the *constraints* part.

```
MAIN parallel_adder_subtracter
{
  LS : ARRAY [5] OF CONTINUOUS_LASER;
  PBS : ARRAY [4] OF POLARIZING_BEAM_SPLITTER;
  BS : ARRAY [7] OF CUBE_BEAM_SPLITTER;
  LC : ARRAY [6] OF SPATIAL_LIGHT_MODULATOR;
  PR : ARRAY [4] OF PRISM;
  CL1, CL2 : LENS;
  HWP1,HWP2 : RETARDATION_PLATE;
  DSP : DICHROIC_SHEET;
  pos1, pos2 : POSITION;
  t : TIME;
  i : POWER;
```

```
  pol : POLARIZATION;
}
{
  {
    LS[1] := BET_OBJECT (CL020);
    LS[1].position := (0, -30, 0);
    LS[1].face_direction := (UNDEF, UNDEF, DOWNZ);
    LS[1].power := 25.0;
    LS[1].wavelength := 633.0;
    LS[1].polarization := P_PLANE;
    LS[1].laser_head := (5, -30, 0);
    LS[2] := COPY_OBJECT (LS[1]);
    LS[2].laser_head := (5, -60, 0);
    LS[3] := COPY_OBJECT (LS[1]);
    LS[3].laser_head := (100, 15, 0);
    LS[4] := COPY_OBJECT (LS[1]);
    LS[4].laser_head := (110, 15, 0);
    LS[4].polarization := S_PLANE; LS[5] := COPY_OBJECT (LS[4]);
    LS[5].laser_head := (150, 15, 0);
    PBS[1] := GET_OBJECT (PBS015);
    PBS[2] := COPY_OBJECT (PBS[1]);
    PBS[3] := COPY_OBJECT (PBS[1]);
    PBS[4] := COPY_OBJECT (PBS[1]);
    BS[1] := GET_OBJECT (BS005);
    BS[2] := COPY_OBJECT (BS[1]);
```

```
  BS[3] := COPY_OBJECT (BS[1]);
  BS[4] := COPY_OBJECT (BS[1]);
  BS[5] := COPY_OBJECT (BS[1]);
  BS[6] := COPY_OBJECT (BS[1]);
  BS[7] := COPY_OBJECT (BS[1]);
  LC[1] := GET_OBJECT (SLM002);
  LC[2] := COPY_OBJECT (LC[1]);
  LC[3] := COPY_OBJECT (LC[1]);
  LC[4] := COPY_OBJECT (LC[1]);
  LC[5] := COPY_OBJECT (LC[1]);
  LC[6] := COPY_OBJECT (LC[1]);
  CL1 := GET_OBJECT (LEN033);
  CL2 := COPY_OBJECT (CL1);
  DSP := GET_OBJECT (DSP004);
  DSP.polarization := P_PLANE;
  HWP1 := GET_OBJECT (HWP003);
  HWP2 := COPY_OBJECT (HWP1);
  PR[1] := GET_OBJECT (PR003);
  PR[2] := COPY_OBJECT (PR[1]);
  PR[3] := COPY_OBJECT (PR[1]);
  PR[4] := COPY_OBJECT (PR[1]);
}
{
  LC[1].position := LS[1].position POSADD (RIGHT 20, NONE, NONE);
  LC[1].face_direction := (LEFTX, UNDEF, UNDEF);
```

```
PBS[1].position := LC[1].position POSADD (RIGHT 30, NONE, NONE);
PBS[1].face_direction := (LEFTX, BACKY, UNDEF);
PBS[1].orientation := (135,90);
PR[2].position := PBS[1].position POSADD (NONE, FORTH 30, NONE);
PR[2].face_direction := (LEFTX, FORTHY, UNDEF);
PR[2].orientation := (55,90);
LC[3].position := PBS[1].position POSADD (RIGHT 30, NONE, NONE);
LC[3].face_direction := FACEMIN LC[1].face_direction;
HWP1.position := PR[2].position POSADD (RIGHT 20, NONE, NONE);
HWP1.face_direction := LC[1].face_direction;
BS[1].position := LC[3].position POSADD (RIGHT 30, NONE, NONE);
BS[1].face_direction := PR[2].face_direction;
BS[1].orientation := (45,90);
BS[7].position := HWP1.position POSADD (RIGHT 30, NONE, NONE);
BS[7].face_direction := PBS[1].face_direction;
BS[7].orientation := PBS[1].orientation;
BS[2].position := BS[1].position POSADD (RIGHT 30, NONE, NONE);
BS[2].face_direction := (LEFTX, BACKY, UNDEF);
BS[2].orientation := BS[7].orientation;
LC[5].position := BS[7].position POSADD (RIGHT 30, NONE, NONE);
LC[5].face_direction := (LEFTX, UNDEF, UNDEF);
PR[3].position := LC[5].position POSADD (RIGHT 10, NONE, NONE);
PR[3].face_direction := (RIGHTX, FORTHY, UNDEF);
PR[3].orientation := (135,90);
PBS[3].position := BS[2].position POSADD (RIGHT 30, NONE, NONE);
```

```
PBS[3].face_direction := BS[1].face_direction;
PBS[3].orientation := BS[1].orientation;
LS[3].position := BS[7].position POSADD (NONE, FORTH 20, NONE);
LS[3].face_direction := LS[1].face_direction;
LS[4].position := BS[1].position POSADD (NONE, FORTH 50, NONE);
LS[4].face_direction := (RIGHTX, UNDEF, UNDEF);
LS[5].position := LC[5].position POSADD (RIGHT 20,FORTH 20, NONE);
CL1.position := BS[2].position POSADD (NONE, BACK 15, NONE);
CL1.face_direction := (UNDEF, FORTHY, UNDEF);
LS[2].position := LS[1].position POSADD (NONE, BACK 50, NONE);
LS[2].face_direction := LS[1].face_direction;
LC[2].position := LS[2].position POSADD (RIGHT 20, NONE, NONE);
LC[2].face_direction := (LEFTX, UNDEF, UNDEF);
PBS[2].position := LC[2].position POSADD (RIGHT 30, NONE, NONE);
PBS[2].face_direction := BS[1].face_direction;
PBS[2].orientation := BS[1].orientation;
LC[4].position := PBS[2].position POSADD (RIGHT 30, NONE, NONE);
LC[4].face_direction := LC[3].face_direction;
HWP2.position := PBS[2].position POSADD (NONE, BACK 20, NONE);
HWP2.face_direction := (UNDEF, FORTHY, UNDEF);
PR[1].position := HWP2.position POSADD (NONE, BACK 20, NONE);
PR[1].face_direction := FACEMIN PR[3].face_direction;
PR[1].orientation := (135, 90);
BS[4].position := LC[4].position POSADD (RIGHT 60, NONE, NONE);
BS[4].face_direction := BS[1].face_direction;
```

```
BS[4].orientation := BS[1].orientation;
CL2.position := BS[4].position POSADD (NONE, FORTH 20, NONE);
CL2.face_direction := FACEMIN CL1.face_direction;
PBS[4].position := PR[1].position POSADD (RIGHT 100, NONE, NONE);
PBS[4].face_direction := BS[2].face_direction;
PBS[4].orientation := BS[2].orientation;
BS[5].position := BS[4].position POSADD (RIGHT 30, NONE, NONE);
BS[5].face_direction := BS[4].face_direction;
BS[5].orientation := BS[4].orientation;
DSP.position := BS[5].position POSADD (RIGHT 10, NONE, NONE);
DSP.face_direction := LC[2].face_direction;
DSP.orientation := LC[2].orientation;
LC[6].position := PBS[4].position POSADD (RIGHT 30, NONE, NONE);
LC[6].face_direction := LC[5].face_direction;
BS[6].position := BS[5].position POSADD (RIGHT 30, NONE, NONE);
BS[6].face_direction := BS[2].face_direction;
BS[6].orientation := BS[2].orientation;
PR[4].position := LC[6].position POSADD (RIGHT 20, NONE, NONE);
PR[4].face_direction := FACEMIN PR[2].face_direction;
PR[4].orientation := (45, 90);
/* the placement issues are done here. next we are going to place
a constraint on the space taken by the architecture. Two points
representing the two corners of the diagonal of the imaginary box
where we wish to place the components of the architecture in are
given */
```

```
    ARCHITECTURE_DIMENSION((0, 0, 20), (200, 200, 0));
  }
}
/* the beginning of the action part */
{
  t := 27.4;
  pos1 := (110, -80, 0);
  i := POWER AT pos1 AT t;
  PRINT i;
  t := t + 12; pos2 := (220, -80, 0);
  pol := POLARIZATION AT pos2 AT t;
  PRINT pol;
}
END;
```

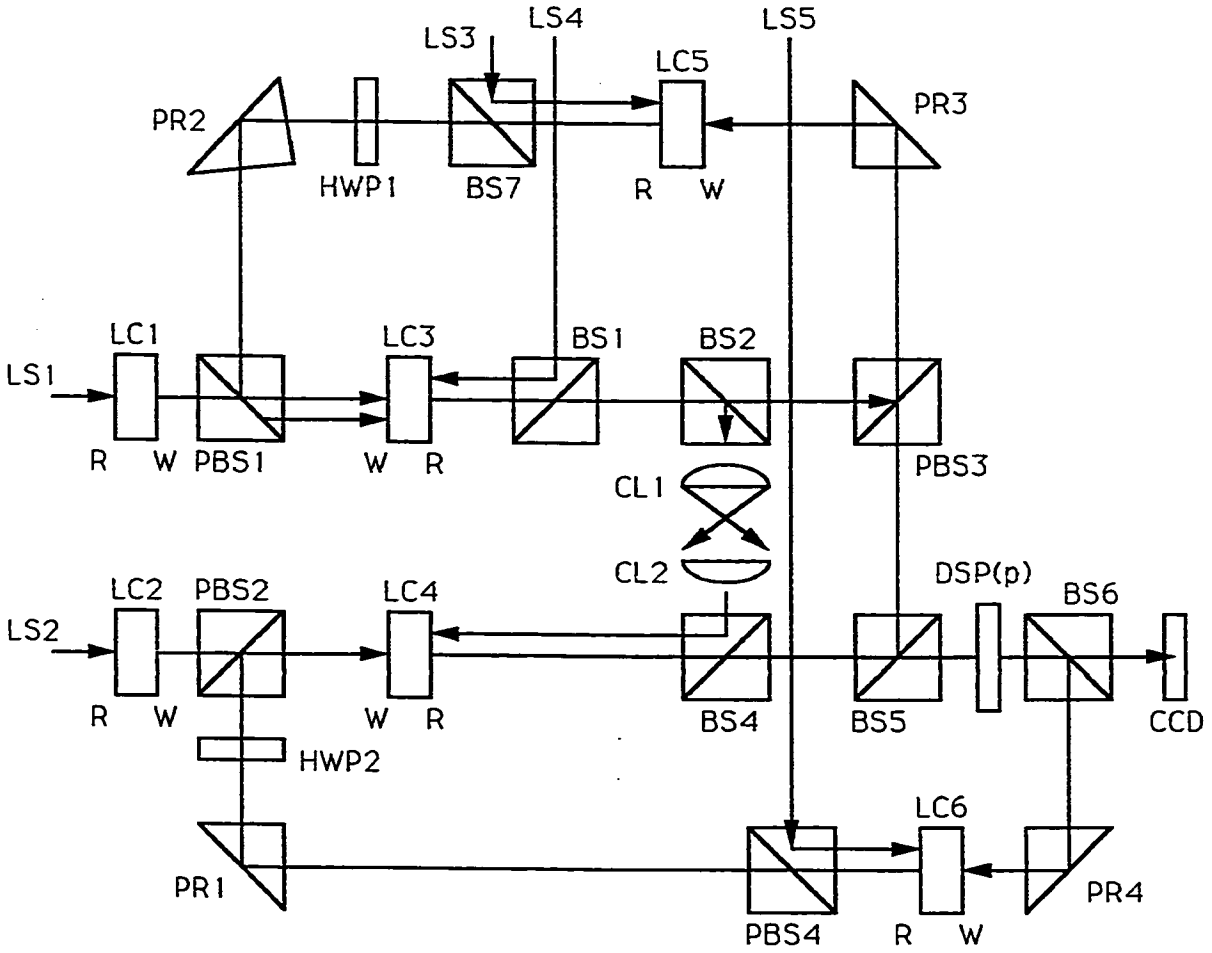


Figure 6.3 Experimental setup of an optical processor for addition and subtraction

6.4 APPLICATION IV: Optical Implementation of a Crossover Interconnection Network

An optical implementation of a *half-crossover network* for N input ports is found in [JAHN88]. The half-crossover network for (N=8) ports is given in Figure 6.4. Each stage of the (M=log_2 N) of the network can be opticaly implemented by using the architecture of Figure 6.5. The only difference in the architecture for different stages is the number of prisms in the *prism array*.

The number of prisms is evaluated by the formula $prism\_num = 2^{stage\_number}$. In the following program, we describe the setup in Figure 6.5 by the assembly *single_stage*.

For the sake of completeness and to make sure that assemblies do not conflict in position, we added a mirror at the output at 45° incidence to reflect the light parallel to the $x - axis$. This makes the different stages continue parallel to the $x - axis$ rather than making a loop and after 4 stages repeat themselves on the same positions.

```
MAIN crossover_network
{
  LS : CONTINUOUS_LASER;
  in, out : ARRAY [0..3] OF LASER_LIGHT;
/* in and out are going to be the inputs and outputs to and from assemblies */
  LEN : LENS;
  t : TIME;
  p : POLARIZATION;
  pos : POSITION;
```

```
  focal : FLOAT;
  i, M,stage : INT;
  N: INT = 8; /* 8 ports to be interconnected */
  orient :ORIENTATION;
}
{ /* description placement part */
  {
    { /* assembly beginning */
      ASSEMBLY single_stage( stage_num, INPUT : LEN[1], OUTPUT : LEN[4]);
      {
        LEN : ARRAY [4] OF LENS;
        BS : BEAM_SPLITTER;
        PR : ARRAY [2*stage_num] OF PRISM;
        M1, M2 : FLAT_MIRROR;
        i : INT;
        hypot, fl : FLOAT;
      }
      {
        {
          i := 1;
          WHILE (i <= 4)
                  {
                    LEN[i] := GET_OBJECT(LEN110);
                    i := i + 1;
                  }
```

```
        LEN[1].position := (0, 0, 0) POSADD (NONE, FORTH 20, NONE);
/* LEN[1] is the first component within the assembly and it has a position
relative to (0, 0, 0)
        LEN[1].orientation := (0, 90);
        fl := LEN[1].focal_length;
        BS := GET_OBJECT(BS049);
        BS.position := LEN[1].position POSADD (RIGHT fl, NONE, NONE);
        BS.orientation := (45, 90);
        BS.face_direction := (LEFTX, FORTHY, UNDEF);
        LEN[2].position := BS.position POSADD (NONE, FORTH fl, NONE);
        LEN[2].orientation := ORUNMIN LEN[1].orientation;
        i := 1; j :=2*stage_num/2-1;
        WHILE(i <= 2 *stage_num)
                {
                  PR[i] := GET_OBJECT(PR022);
                  hypot := PR[i].hypotenuse_dim;
                   IF (i <= 2*stage_num/2) THEN
                      {
                        PR[i].position := LEN[2].position POSADD
                        (LEFT (j+0.5)*hypot, FORTH fl, NONE);
                      }
                  ELSE
                      {
                        PR[i].position := LEN[2].position POSADD
                        (RIGHT (j+0.5)*hypot, FORTH fl, NONE);
```

```
                    }
                PR[i].face_direction := (UNDEF, BACKY, UNDEF);
                i := i + 1;
                j := j - 1;
              }
        LEN[3].position := BS.position POSADD (RIGHT fl, NONE, NONE);
        LEN[3].face_direction := LEN[1].face_direction;
        M1 := GET_OBJECT(FM020);
        M1.position := LEN[3].position POSADD (RIGHT fl, NONE, NONE);
        M1.face_direction := LEN[1].face_direction;
        LEN[4].position := BS.position POSADD (RIGHT fl, NONE, NONE);
        LEN[4].face_direction := LEN[1].face_direction;
        M2 := COPY_OBJECT(M1);
        M2.position := LEN[3].position POSADD (NONE, BACK fl, NONE);
        M2.face_direction := ORUNMIN LEN[1].orientation;
      }
    }
  }
  LS := GET_OBJECT(CL002);
  LS.position := (-20, 0, 0);
  LS.face_direction := (UNDEF, UNDEF, DOWNZ);
  LS.power := 18.0;
  LS.wavelength := 632.0;
  LS.laser_head := (-15, 0, 0);
/* note that the lens is of the same model chosen in the assmbly */
```

```
    LEN := GET_OBJECT(LEN110);
    focal := LEN.focal_length;
    in[0] := LS;
    M := LOG2(N);
    pos := (0, 0, 0);
    single_stage.face_direction := (UNDEF, UNDEF, DOWNZ);
    single_stage.orientation := (90, 0);
    FOR i = 0, M-1
          {
       single_stage.position := pos; ASSEMBLY single_stage(i, in[i], out[i]);
       in[i+1] := out[i]
       pos := pos POSADD (RIGHT 3*focal, NONE, NONE);
          }
  }
} /* end of description part, begin the action part */
{
  t := 12.0 ;
  p := POLARIZATION AT LEN[4].position POSADD (NONE, BACK 0.5 * fl, NONE);
  PRINT p;
}
END;
```

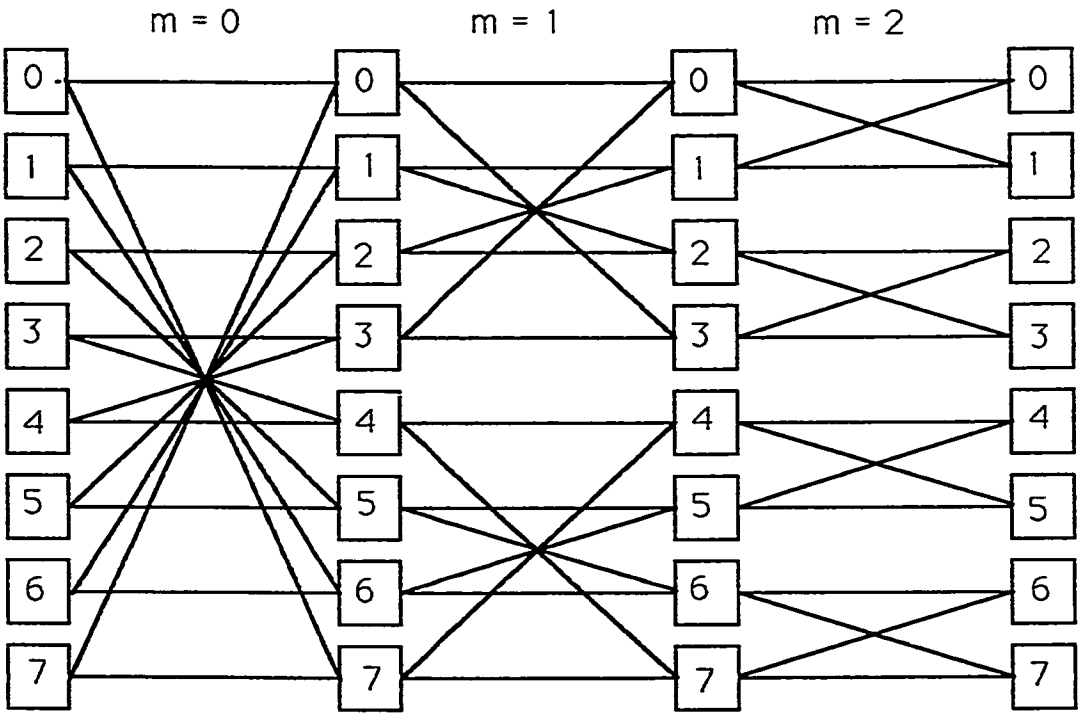


Figure 6.4 Half-crossover network for eight input ports. The index m indicates the number of a specific stage

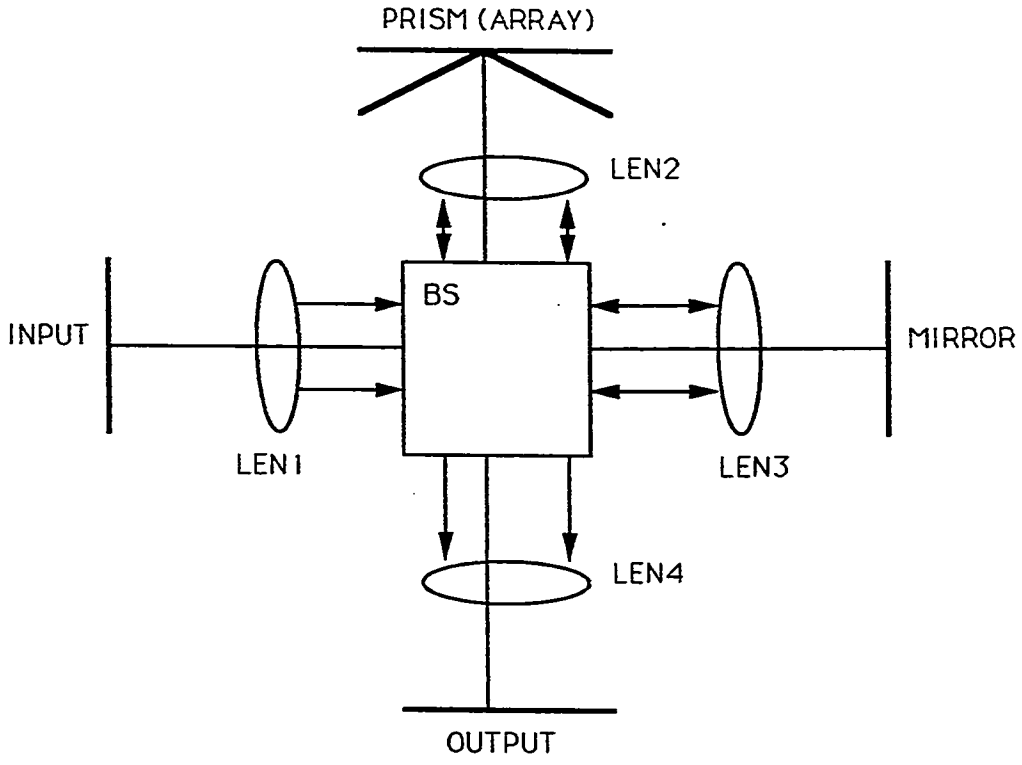


Figure 6.5 Optical setup for the implementation of one stage

6.5 APPLICATION V : An All-Optical Circuit-Switch

The optical architecture of a circuit-switch is proposed in [GHAF90]. The design consists of a number of units: the *interference* unit, the *compare and contention* unit, the *exchange* unit, the *backward router* unit and the *pseudo signal generator* unit.

We described the architecture as a set of assemblies within the describing program. Each of the units is described by a seperate assembly.

Figure 6.6 shows the architecture, In the figure, the reader can detect the units mentioned above. The reader will notice components with same numbers such as **BS1** in all of the units. This occurs because we treat each unit as a seperate assembly. Dividing the architecture into assemblies gives the **PLOADS** user the advantages of *modularization.*

The program in **PLOADS** for this architecture is the following:

```
MAIN circuit_switch
{
  LSX, LSY, LSS , B1, B2 : CONTINUOUS_LASER;
  QWP1, QWP2 : RETARDATION_PLATE;
  INF : FILTER;
  M1, M2 : FLAT_MIRROR;
  BS : BEAM_SPLITTER;
  delta : FLOAT;
  power1, power2 : POWER;
  dir : FACE_DIRECTION;
```

```
  orient : ORIENTATION;
  inter_in1, inter_in2, inter_in3 : LASER_LIGHT;
  inter_out1, inter_out2, inter_out3, inter_out4 : LASER_LIGHT;
  inter_out5, inter_out6, inter_out7, inter_out8 : LASER_LIGHT;
  comp_in1, comp_in2, comp_out1, comp_out2 : LASER_LIGHT;
  comp_out3, exch_in1, exch_in2, exch_in3 : LASER_LIGHT;
  back_in1, back_in2, back_in3, back_out1 : LASER_LIGHT;
  back_out2, sig_in1, sig_in2, sig_in3, sig_out : LASER_LIGHT;
}
{
/* at the beginning of the description part, we start with the interference unit
description assembly */
  {
    {
      ASSEMBLY interference (delta, INPUT : BS[1], BS[4], INF[6],
      OUTPUT : M[5], BS[8], M[1], M[11], INF[6], M[14], BS[11], M[8]);
/* note that M[5], M[12], M[14], M[8] produce output light that is not used
by other assemblies or by the main program. However, we can still describe them
since they could have some effect on the architecture if their output is not stopped
before it reaches other components in the system */
      {
        i : INT;
        BS : ARRAY[11] OF BEAM_SPLITTER;
        INF : ARRAY[6] OF FILTER;
        CK : PULSED_LASER;
```

```
  M : ARRAY[14] OF FLAT_MIRROR;
}
{
  {
    BS[1] := GET_OBJECT(BS003);
    BS[1].position := (0, 0, 0);
    BS[1].face_direction := (LEFTX, BACKY, UNDEF);
    BS[1].orientation := (45, 90);
    FOR i = 2, 11
          {
            BS[i] := COPY_OBJECT(BS[1]);
          }
    INF[1] := GET_OBJECT(INF021);
    INF[1].position := BS[1].position POSADD (NONE, FORTH 30, NONE);
    INF[1].face_direction := (UNDEF, BACKY, UNDEF);
    FOR i = 2, 11
          {
            INF[i] := COPY_OBJECT(INF[1]);
          }
    BS[2].position := BS[1].position POSADD (RIGHT 30, NONE, NONE);
    BS[2].face_direction := (LEFTX, FORTHY, UNDEF);
    BS[1].orientation := (135, 90);
    M[1] := GET_OBJECT(FM030);
    FOR i = 2, 11
          {
```

```
        INF[i] := COPY_OBJECT(INF[1]);
      }
M[3].position := BS[2].position POSADD (RIGHT 40, NONE, NONE);
M[3].face_direction := (LEFTX, FORTHY, UNDEF);
M[3].orientation:= (45, 90);
INF[2].position := M[3].position POSADD (RIGHT 10, FORTH 30, NONE);
INF[2].face_direction := (UNDEF, BACKY, UNDEF);
M[2].position := INF[2].position POSADD (NONE, BACK 50, NONE);
M[2].face_direction := M[3].face_dorection;
M[2].orientation:= M[3].orientation;
M[1].position := M[2].position POSADD (LEFT 100, BACK 50, NONE);
M[1].face_direction := (RIGHTX, FORTHY, UNDEF);
M[1].orientation:= M[3].orientation ORADD (90, 0);
M[4].position := M[3].position POSADD (RIGHT 20, NONE, NONE);
M[4].face_direction := M[1].face_direction;
M[4].orientation:= M[1].orientation;
BS[3].position := M[4].position POSADD (RIGHT 20, NONE, NONE);
BS[3].face_direction := (LEFTX, BACKY, UNDEF);
BS[3].orientation := (135, 90);
BS[4].position := BS[3].position POSADD (RIGHT 40, NONE, NONE);
BS[4].face_direction := (RIGHTX, BACKY, UNDEF);
BS[4].orientation := (45, 90);
INF[3].position := BS[4].position POSADD (NONE, FORTH 30, NONE);
INF[3].face_direction := INF[2].face_direction;
CK := GET_OBJECT(PL009);
```

```
CK.position := BS[2].position POSADD (RIGHT 20, FORTH 50, NONE);
CK.face_direction := (UNDEF, UNDEF, DOWNZ);
CK.laser_head := CK.position POSADD (RIGHT 5, NONE, NONE);
CK.wavelength :=633.0;
WAIT_FOR delta AT CK.position POSADD (RIGHT 5, NONE, NONE);
M[6].position := CK.position POSADD (RIGHT 20, NONE, NONE);
M[6].face_direction := (LEFTX, BACKY, UNDEF);
M[6].orientation:= (135, 90);
INF[4].position := BS[3].position POSADD (NONE, FORTH 50, NONE);
INF[4].face_direction := INF[2].face_direction;
BS[6].position := INF[2].position POSADD (NONE, FORTH 40, NONE);
BS[6].face_direction := (RIGHTX, BACKY, UNDEF);
BS[6].orientation := (45, 90);
M[7].position := CK.position POSADD (NONE, FORTH 20, NONE);
M[7].face_direction := M[6].face_direction;
M[7].orientation:= M[6].orientation;
M[5].position := M[7].position POSADD (LEFT 95, NONE, NONE);
M[5].face_direction := FACEMIN M[6].face_direction;
M[5].orientation := (135, 90);
WAIT_FOR delta AT M[5].position POSADD (NONE, FORTH 90, NONE);
M[8].position := M[7].position POSADD (RIGHT 90, NONE, NONE);
M[8].face_direction := (LEFTX, FORTHY, UNDEF);
WAIT_FOR delta AT M[8].position POSADD (NONE, FORTH 90, NONE);
BS[7].position := INF[4].position POSADD (NONE, FORTH 40, NONE);
BS[7].face_direction := (LEFTX, BACKY, UNDEF);
```

```
BS[7].orientation := (135, 90);
BS[5].position := INF[1].position POSADD (NONE, FORTH 40, NONE);
BS[5].face_direction := (RIGHTX, BACKY, UNDEF);
BS[5].orientation := BS[5].orientation ORADD (-90, 0);
INF[5].position := BS[2].position POSADD (NONE, FORTH 105, NONE);
INF[5].face_direction := INF[2].face_direction;
BS[9].position := INF[5].position POSADD (NONE, FORTH 15, NONE);
BS[9].face_direction := (RIGHTX, BACKY, UNDEF);
BS[9].orientation := (45, 90);
BS[10].position := INF[3].position POSADD (NONE, FORTH 90, NONE);
BS[10].face_direction := (LEFTX, BACKY, UNDEF);
BS[10].orientation := BS[5].orientation ORADD (90, 0);
BS[8].position := BS[5].position POSADD (NONE, FORTH 50, NONE);
BS[8].face_direction := BS[5].face_direction;
BS[8].orientation := BS[5].orientation;
WAIT_FOR 3*delta AT BS[8].position POSADD (RIGHT 30, NONE, NONE);
WAIT_FOR delta AT BS[8].position POSADD (NONE, FORTH 20, NONE);
BS[11].position := BS[10].position POSADD (NONE, FORTH 20, NONE);
BS[11].face_direction := BS[10].face_direction;
BS[11].orientation := BS[10].orientation;
WAIT_FOR 3*delta AT BS[8].position POSADD (LEFT 20, NONE, NONE);
WAIT_FOR delta AT BS[8].position POSADD (NONE, FORTH 10, NONE);
M[10].position := BS[11].position POSADD (LEFT 60, NONE, NONE);
M[10].face_direction := FACEMIN M[6].face_direction;
M[10].orientation := (135, 90);
```

```
        M[9].position := BS[8].position POSADD (RIGHT 80, NONE, NONE);
        M[9].face_direction := M[8].face_direction;
        M[9].orientation:= M[8].orientation;
        M[12].position := BS[9].position POSADD (NONE, FORTH 60, NONE);
        M[12].face_direction := FACEMIN M[7].face_direction;
        M[12].orientation := (135, 90);
        M[11].position := M[12].position POSADD (LEFT 15, NONE, NONE);
        M[11].face_direction := M[5].face_direction;
        M[11].orientation:= M[5].orientation;
        INF[6].position := M[9].position POSADD (RIGHT 10, FORTH 40, NONE);
        INF[6].face_direction := INF[2].face_direction;
        M[13].position := BS[7].position POSADD (NONE, FORTH 90, NONE);
        M[13].face_direction := FACEMIN M[7].face_direction;
        M[13].orientation := ORUNMIN M[7].orientation;
        M[14].position := M[13].position POSADD (RIGHT 20, NONE, NONE);
        M[14].face_direction := M[8].face_direction;
        M[14].orientation:= M[8].orientation;
      }
    }
/* the following is the compare and contention unit's assembly */


  {
    ASSEMBLY compare_contention(delta, INPUT : QWP[1], QWP[2],
```

```
OUTPUT : BS[1], BS[2], BS[3])
{
  i : INT;
  BS : ARRAY[5] OF BEAM_SPLITTER;
  QWP : ARRAY[6] OF RETARDATION_PLATE;
  M : ARRAY[4] OF FLAT_MIRROR;
  PBS1, PBS2 : POLARIZING_BEAM_SPLITTER;
  INF1, INF2 : FILTER;
  CK : PULSED_LASER;
}
{
  {
    QWP[1] := GET_OBJECT(RP030);
    QWP[1].position := (0, 0, 0);
    QWP[1].face_direction := (UNDEF, BACKY, UNDEF);
    FOR i = 2, 6
        {
          QWP[i] := COPY_OBJECT(QWP[1]);
        }
/* the copy command in the previous statement copies all of the attributes values
of the QWP[1] into all of the other quarter wave plates in the architecture */
    BS[1] := GET_OBJECT(BS056);
    BS[1].position := QWP[1].position POSADD (NONE, FORTH 40, NONE);
    BS[1].face_direction := (RIGHTX, BACKY, UNDEF);
    BS[1].orientation := (45, 90);
```

```
FOR i = 2, 5
      {
        BS[i] := COPY_OBJECT(BS[1]);
      }
PBS1 := PBS2 := GET_OBJECT(PBS003);
PBS1.position := BS[1].position POSADD (RIGHT 30, NONE, NONE);
PBS1.face_direction := (LEFTX, BACKY, UNDEF);
PBS1.orientation := (135, 90);
QWP[3].position := PBS1.position POSADD (RIGHT 30, NONE, NONE);
QWP[3].face_direction := (LEFTX, UNDEF, UNDEF);
INF1 := GET_OBJECT(F020);
INF1.position := QWP[3].position POSADD (RIGHT 10, NONE, NONE);
INF1.face_direction := QWP[3].face_direction;
INF1.thickness := 0.1; /* of a meter */
QWP[4].position := INF1.position POSADD (RIGHT 10, NONE, NONE);
QWP[4].face_direction := QWP[3].face_direction;
PBS2.position := QWP[4].position POSADD (RIGHT 90, NONE, NONE);
PBS2.face_direction := (LEFTX, BACKY, UNDEF);
PBS2.orientation := PBS1.orientation;
BS[2].position := PBS2.position POSADD (NONE, BACK 40, NONE);
BS[2].face_direction := (RIGHTX, BACKY, UNDEF);
BS[2].orientation := (45, 90);
QWP[2].position := BS[2].position POSADD (NONE, BACK 20, NONE);
/* it has the same face_direction as that of QWP[1] by the copy statement */
M[1] := GET_OBJECT(FM024);
```

```
M[1].position := PBS2.position POSADD (RIGHT 20, NONE, NONE);
M[1].face_direction := (LEFTX, FORTHY, UNDEF);
M[1].orientation := (45, 90);
M[1].reflectance := 95.6; /* reflectance percentage */
FOR i = 2, 4
      {
        M[i] := COPY_OBJECT(M[1]);
      }
M[2].position := PBS1.position POSADD (NONE, FORTH 120, NONE);
M[2].face_direction := FACEMIN M[1].face_direction;
/* no need to specify the orientation sinc the direction is the exact opposite */
BS[5].position := M[2].position POSADD (RIGHT 100, NONE, NONE);
BS[5].face_direction := (LEFTX, BACKY, UNDEF);
BS[5].orientation := BS[1].orientation ORADD (90, 0);
INF2 := GET_OBJECT(F017); /* different than INF1 */
INF2.position := BS[5].position POSADD (NONE, BACK 20, NONE);
INF2.face_direction := (UNDEF, BACKY, UNDEF);
INF2.side := 0.16; /* of a meter */
QWP[6].position := INF[2].position POSADD (NONE, BACK 35, NONE);
BS[4].position := QWP[6].position POSADD (NONE, BACK 15, NONE);
BS[4].face_direction := (LEFTX, FORTHY, UNDEF);
BS[4].orientation := BS[1].orientation;
CK := GET_OBJECT(PL011);
CK.position := BS[4].position POSADD (LEFT 70, BACK 4, NONE);
/* the CK center and head position are not on the same Y-line */
```

```
      CK.laser_head := CK.position POSADD (RIGHT 15, FORTH 4, NONE);
      CK.face_direction := (UNDEF, UNDEF, DOWNZ);
      CK.wavelength := 633.0;
      WAIT_FOR 3*delta AT CK.position POSADD (RIGHT 10, NONE, NONE);
      QWP[5].position := BS[4].position POSADD (NONE, BACK 15, NONE);
      BS[3].position := QWP[5].position POSADD (NONE, BACK 15, NONE);
      BS[3].face_direction := (RIGHTX, FORTHY, UNDEF);
      BS[3].orientation := (135, 90);
      M[4].position := M[1].position POSADD (NONE, FORTH 140, NONE);
      M[4].face_direction := FACEMIN M[2].face_direction;
      M[3].position := M[4].position POSADD (LEFT 80, NONE, NONE);
      M[3].face_direction := M[2].face_direction;
    }
  }
} /* end of the assembly for the compare and contention unit */



{ /* the beginning of the exchange unit assembly */
  ASSEMBLY exchange(delta, INPUT : BS1, PBS3, OUTPUT :
  M[2], PBS2, M[4])
  {
    BS1, BS2, BS3 : BEAM_SPLITTER;
    M : ARRAY[4] OF FLAT_MIRROR;
    INF1, INF2 : FILTER;
```

```
  PBS1, PBS2, PBS3 : POLARIZING_BEAM_SPLITTER;
  QWP1, QWP2, QWP3, HWP1, HWP2 : RETARDATION_PLATE;
  CK : PULSED_LASER;
  LS : CONTINUOUS_LASER;
  j : INT;
}
{
  {
    BS1 := GET_OBJECT(BS013);
    BS1.position := (0, 0, 0);
    BS1.face_direction := (RIGHTX, BACKY, UNDEF);
    BS1.orientation := (45, 90);
    WAIT_FOR 4*delta AT BS1.position POSADD (NONE, FORTH 20, NONE);
    M[1] := GET_OBJECT(FM002);
    M[1].position := BS1.position POSADD (RIGHT 50, NONE, NONE);
    M[1].face_direction := FACEMIN BS1.face_direction;
    M[1].orientation := BS1.orientation;
    FOR i = 2, 4
        {
          M[i] := COPY_OBJECT(M[1]);
        }
    BS2 := COPY_OBJECT(BS1);
    BS2.position := M[1].position POSADD (NONE, FORTH 15, NONE);
    BS2.face_direction := BS1.face_direction;
    BS2.orientation := BS1.orientation;
```

```
CK := GET_OBJECT(PL021);
CK.position := BS2.position POSADD (LEFT 30, NONE, NONE);
CK.laser_head := CK.position POSADD (RIGHT 10, NONE, NONE);
CK.face_direction := (UNDEF, UNDEF, DOWNZ);
CK.wavelength := 633.0;
WAIT_FOR 2*delta AT CK.position POSADD (RIGHT 10, NONE, NONE);
INF1 := GET_OBJECT(F019);
INF1.position := BS2.position POSADD (NONE, FORTH 15, NONE);
INF1.face_direction := (UNDEF, BACKY, UNDEF);
HWP1 := GET_OBJECT(RP009);
HWP1.position := INF1.position POSADD (NONE, FORTH 15, NONE);
HWP1.face_direction := (UNDEF, BACKY, UNDEF);
BS3 := COPY_OBJECT(BS1)
BS3.position := HWP1.position POSADD (NONE, FORTH 20, NONE);
BS3.face_direction := (LEFTX, BACKY, UNDEF);
BS3.orientation := BS1.orientation ORADD (90, 0);
LS := GET_OBJECT(CL011);
LS.position := BS3.position POSADD (LEFT 30, NONE, NONE);
LS.laser_head := LS.position POSADD (RIGHT 10, NONE, NONE);
LS.face_direction := CK.face_direction;
LS.wavelength : 633.0;
HWP2 := COPY_OBJECT(HWP1);
HWP2.position := BS3.position POSADD (NONE, FORTH 20, NONE);
WAIT_FOR 3*delta AT HWP2.position POSADD (NONE, FORTH 5, NONE);
QWP1 := GET_OBJECT(RP039);
```

QWP1.position := BS1.position POSADD (NONE, FORTH 85, NONE);

QWP1.face_direction := (UNDEF, BACKY, UNDEF);

PBS1 := **GET_OBJECT**(PBS003);

PBS1.position := QWP1.position POSADD (NONE, FORTH 15, NONE);

PBS1.face_direction := (RIGHTX, BACKY, UNDEF);

PBS1.orientation := (45, 90);

M[3].position := PBS1.position POSADD (NONE, FORTH 20, NONE);

M[3].face_direction := (LEFTX, BACKY, UNDEF);

M[3].orientation := (135, 90);

M[3].reflectance := 92.0;

M[2].position := M[3].position POSADD (LEFT 20, NONE, NONE);

M[2].face_direction := FACEMIN M[3].face_direction;

M[2].orientation := (45, 90);

M[2].reflectance := 94.0;

PBS2 := **COPY_OBJECT**(PBS1);

PBS2.position := HWP2.position POSADD (NONE, FORTH 15, NONE);

PBS2.face_direction := (RIGHTX, BACKY, UNDEF);

PBS2.orientation := (45, 90);

QWP2 := **COPY_OBJECT**(QWP1);

QWP2.position := PBS2.position POSADD (RIGHT 40, NONE, NONE);

QWP2.face_direction := (LEFT, UNDEF, UNDEF);

INF2 := **COPY_OBJECT**(INF1);

INF2.position := QWP2.position POSADD (RIGHT 15, NONE, NONE);

INF2.face_direction := (LEFTX, UNDEF, UNDEF);

QWP3 := **COPY_OBJECT**(QWP2);

```
      QWP3.position := INF2.position POSADD (RIGHT 15, NONE, NONE);
      PBS2 := COPY_OBJECT(PBS1)
      PBS2.position := QWP2.position POSADD (RIGHT 50, NONE, NONE);
      PBS2.face_direction := (LEFTX, BACKY, UNDEF);
      PBS2.orientation := (135, 90);
      M[4].position := PBS3.position POSADD (NONE, FORTH 20, NONE);
      M[4].face_direction := (RIGHTX, BACKY, UNDEF);
      M[4].orientation := (45, 90);
    }
  }
} /* the end of the exchange unit assembly description */



{ /* the beginning of the backward router unit assembly */
  ASSEMBLY back_router(delta, INPUT : M1, BS1, M2, OUTPUT :
  PBS1, PBS2)
  {
    BS1 : BEAM_SPLITTER;
    M1, M2 : FLAT_MIRROR;
    PBS1, PBS2 : POLARIZING_BEAM_SPLITTER;
    QWP1, QWP2 : RETARDATION_PLATE;
    INF : FILTER;
  }
  {
```

```
  {
    M1 := M2 := GET_OBJECT(FM010);
    M1.position := (0, 0, 0);
    BS1 := GET_OBJECT(BS033);
    PBS1 := PBS2 := GET_OBJECT(PBS009);
    QWP1 := QWP2 := GET_OBJECT(RD003);
    INF := GET_OBJECT(F020);
  }
}
{ /* the constraints part */
  M1.position := (0, 0, 0);
  M1.face_direction := (RIGHTX, FORTHY, UNDEF);
  M1. orientation := (135, 90);
  BS1.position := M1.position POSADD (RIGHT 20, NONE, NONE);
  BS1.face_direction := (LEFTX, FORTHY, UNDEF);
  BS1.orientation := (45, 90);
  PBS1.position := BS1.position POSADD (RIGHT 20, NONE, NONE);
  PBS1.face_direction := BS1.face_direction;
  PBS1.orientation := (45, 90);
  QWP1.position := PBS1.position POSADD (RIGHT 30, NONE, NONE);
  QWP1.face_direction := (LEFTX, UNDEF, UNDEF);
  INF.position := QWP1.position POSADD (RIGHT 15, NONE, NONE);
  INF.face_direction := QWP1.face_direction;
  QWP2.position := INF.position POSADD (RIGHT 15, NONE, NONE);
  QWP2.face_direction := QWP1.face_direction;
```

```
    PBS2.position := QWP2.position POSADD (RIGHT 20, NONE, NONE);
    PBS2.face_direction := (LEFTX, BACKY, UNDEF);
    PBS2.orientation := (135, 90);
    M2.position := PBS2.position POSADD (RIGHT 30, NONE, NONE);
    M2.face_direction := (LEFTX, FORTHY, UNDEF);
    M2. orientation := (45, 90);
  }
}
{ /* the beginning of the signal_generator assembly */
  ASSEMBLY signal(delta, INPUT : M1, M2, BS1, OUTPUT : BS3)
  {
    BS : ARRAY[4] OF BEAM_SPLITTER;
    M1, M2, M3 : FLAT_MIRROR;
    QWP : RETARDATION_PLATE;
    INF1, INF2 : FILTER;
    i : INT;
  }
  {
    {
      M1 := GET_OBJECT(FM006);
      M1.position := (20, 0, 0); /* it is relative to (0, 0, 0) */
      M1.orientation := (45, 90);
      M1.face_direction := (LEFTX, FORTHY, UNDEF);
      WAIT_FOR 2*delta AT M1.position POSADD (LEFT 10, NONE, NONE);
      INF1 := INF2 := GET_OBJECT(F036);
```

```
INF1.position := M1.position POSADD (NONE, FORTH 10, NONE);
INF1.face_direction := (UNDEF, BACKY, UNDEF);
BS[4] := GET_OBJECT(BS026);
BS[4].position := INF1.position POSADD (RIGHT 5, FORTH 100, NONE);
BS[4].orientation := (135, 90);
BS[4].face_direction := (LEFTX, BACKY, UNDEF);
FOR i = 1, 3
      {
        BS[i] := COPY_OBJECT(BS[4]);
      }
M3 := M2 :=COPY_OBJECT(M1);
M3.position := BS[4].position POSADD (NONE, FORTH 20, NONE);
M3.orientation := (135, 90);
M3.face_direction := (LEFTX, BACKY, UNDEF);
BS[3].position := M3.position POSADD (LEFT 25, NONE, NONE);
BS[3].orientation := (135, 90);
BS[3].face_direction := M3.face_direction;
WAIT_FOR delta AT BS3.position POSADD (NONE, BACK 5, NONE);
BS[2].position := BS[3].position POSADD (NONE, BACK 20, NONE);
BS[2].orientation := (45, 90);
BS[2].face_direction := (RIGHTX, BACKY, UNDEF);
BS[1].position := BS[2].position POSADD (NONE, BACK 20, NONE);
BS[1].orientation := BS[3].orientation;
BS[1].face_direction := (LEFTX, BACKY, UNDEF);
QWP := GET_OBJECT(RP011);
```

```
        QWP.position := BS[1].position POSADD (NONE, BACK 15, NONE);
        QWP.face_direction := (UNDEF, FORTHY, UNDEF);
        INF2.position := QWP.position POSADD (NONE, BACK 15, NONE);
        INF2.face_direction := (UNDEF, BACKY, UNDEF);
        M2.position := INF2.position POSADD (NONE, BACK 40, NONE);
        M2.orientation := (45, 90);
        M2.face_direction := (LEFTX, FORTHY, UNDEF);
      }
    }
  }
/* the description part of the main program */
  dir := (UNDEF, UNDEF, DOWNZ);
  orient := (90, 0);
  INF := GET_OBJECT(F024);
  delta := INF.delay;
  LSX := LSY := LSS := GET_OBJECT(CL021);
  LSX.wavelength := LSY.wavelength := LSS.wavelength := 632.0;
  LSX.position := (0, 0, 0);
  LSX.face_direction := (UNDEF, UNDEF, DOWNZ);
  LSX.laser_head := (0, 10, 0);
  LSY.position := LSX.position POSADD (RIGHT 150, NONE, NONE);
  LSY.face_direction := LSX.face_direction;
  LSY.laser_head := (150, 10, 0);
  inter_in1 := LSX; /* the first input to the interference assembly */
  inter_in2 := LSY;
```

```
inter_in3 := comp_out3; /* note that comp_out3 is not yet given. The system
will take from the assembly compare_contention once it is executed */
interference.position := (0, 40, 0);
interference.face_direction := dir;
interference.orientation := orient;
ASSEMBLY interference(delta, inter_in1, inter_in2, inter_in3, inter_out1,
inter_out2, inter_out3, inter_out4, inter_out5, inter_out6, inter_out7, inter_out8);
comp_in1 := (inter_out2, inter_out1, inter_out4);
comp_in2 := (inter_out7, inter_out6, inter_out8);
compare_contention.position := (0, 260, 0);
compare_contention.face_direction := dir;
compare_contention.orientation := orient;
ASSEMBLY compare_contention( delta, comp_in1, comp_in2,
comp_out1, comp_out2, comp_out3);
exch_in1 := comp_out1;
exch_in2 := sig_out;
exchange.position := (0, 460, 0);
exchange.face_direction := dir;
exchange.orientation := orient;
ASSEMBLY exchange(delta, exch_in1, exch_in2, exch_out1,
exch_out2, exch_out3);
back_in1 := B1;
back_in2 := exch_out2;
back_in3 := B2;
back_router.position := (50, 590, 0);
```

```
back_router.face_direction := dir;
back_router.orientation := orient;
ASSEMBLY back_router(delta, back_in1, back_in2, back_in3,
back_out1, back_out2);
sig_in1 := comp_out2;
sig_in2 := comp_out3;
sig_in3 := LSS;
signal.position := (220, 280, 0);
signal.face_direction := dir;
signal.orientation := orient;
ASSEMBLY signal(delta, sig_in1, sig_in2, sig_in3, sig_out1);
M1 := M2 := GET_OBJECT(FM010);
M1.position := (-40, 610, 0);
M1.face_direction := (RIGHTX, BACKY, UNDEF);
M1. orientation := (45, 90);
WAIT_FOR 5*delta AT M1.position POSADD (NONE, BACK 30, NONE);
BS := GET_OBJECT(BS011);
BS.position := M1.position POSADD (RIGHT 20, NONE, NONE);
BS.face_direction := (LEFTX, BACKY, UNDEF);
BS. orientation := (135, 90);
QWP1 := QWP2 := GET_OBJECT(RP021);
QWP1.position := BS.position POSADD (NONE, BACK 20, NONE);
QWP1.face_direction := (UNDEF, FORTHY, UNDEF);
INF.position := QWP.position POSADD (NONE, BACK 20, NONE);
INF.face_direction := (UNDEF, FORTHY, UNDEF);
```

```
    LSS := B1 := B2 := GET_OBJECT(CL003);
    LSS.wavelength := B1.wavelength := B2.wavelength := 632.0;
    B1.position := (30, 630, 0);
    B1.face_direction := (UNDEF, UNDEF, DOWNZ);
    B1.laser_head := (30, 620, 0);
    B2.position := (270, 630, 0);
    B2.face_direction := B1.face_direction;
    B2.laser_head := (270, 620, 0);
    LSS.position := (220, 480, 0);
    LSS.face_direction := (UNDEF, UNDEF, DOWNZ);
    LSS.laser_head := (230, 480, 0);
    M2.position := (290, 560, 0);
    M2.face_direction := (LEFTX, FORTHY, UNDEF);
    M2. orientation := (45, 90);
    QWP2.position := M2.position POSADD (NONE, FORTH 10, NONE);
    QWP2.face_direction := (UNDEF, BACKY, UNDEF);
    WAIT_FOR 5*delta AT QWP2.position POSADD (NONE, FORTH 5, NONE);
  }
}
{ /* the beginning of the action part of the main program */
  power1 := POWER AT BS.position POSADD (NONE, FORTH 5, NONE);
  power2 := POWER AT QWP.position POSADD (NONE, FORTH 5, NONE);
  PRINT "the power at H is ", power1;
  PRINT "the power at L is ", power2;
}
```

```
END;
```

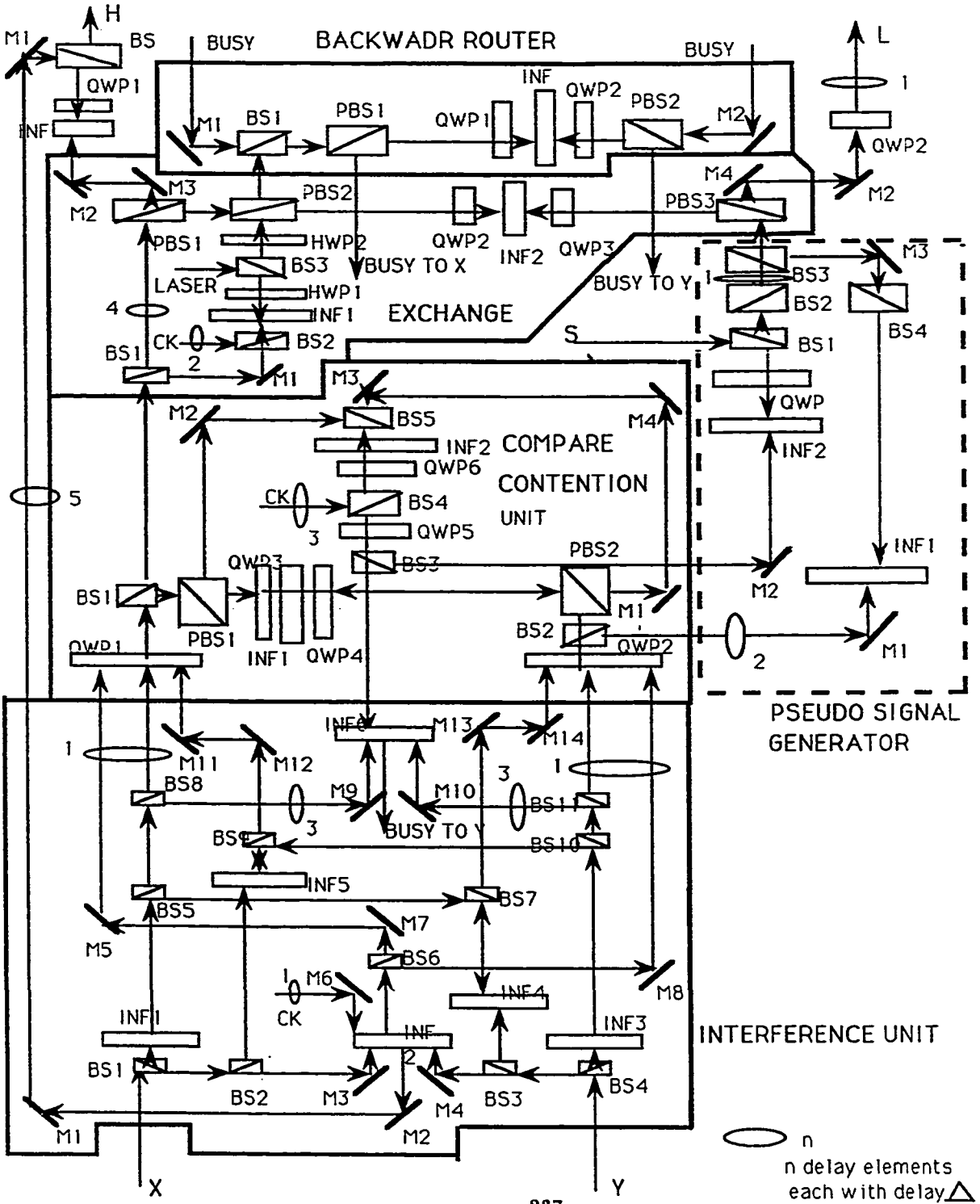


Figure 6.6 Layout of all the units of the switch

Chapter 7

PLOADS INTERPRETER AND SIMULATOR

In this chapter, the design of the overall system of PLOADS is discussed. The different operations and stages of the system are described.

The system consists of mainly two parts. These parts are not yet fully realized in terms of implementation. The first two stages of the first part are implemented. Nevertheless, the detailed design of the system was developed and is discussed in what follows.

We start with Figure 7.1 showing the overall system. The figure shows the transformation stages of a PLOADS program inside this system. The stages of the process are described thoroughly as we move ahead in the chapter.

The two main parts of the system, as shown in Figure 7.1, are the **interpreter** and the **simulator**.

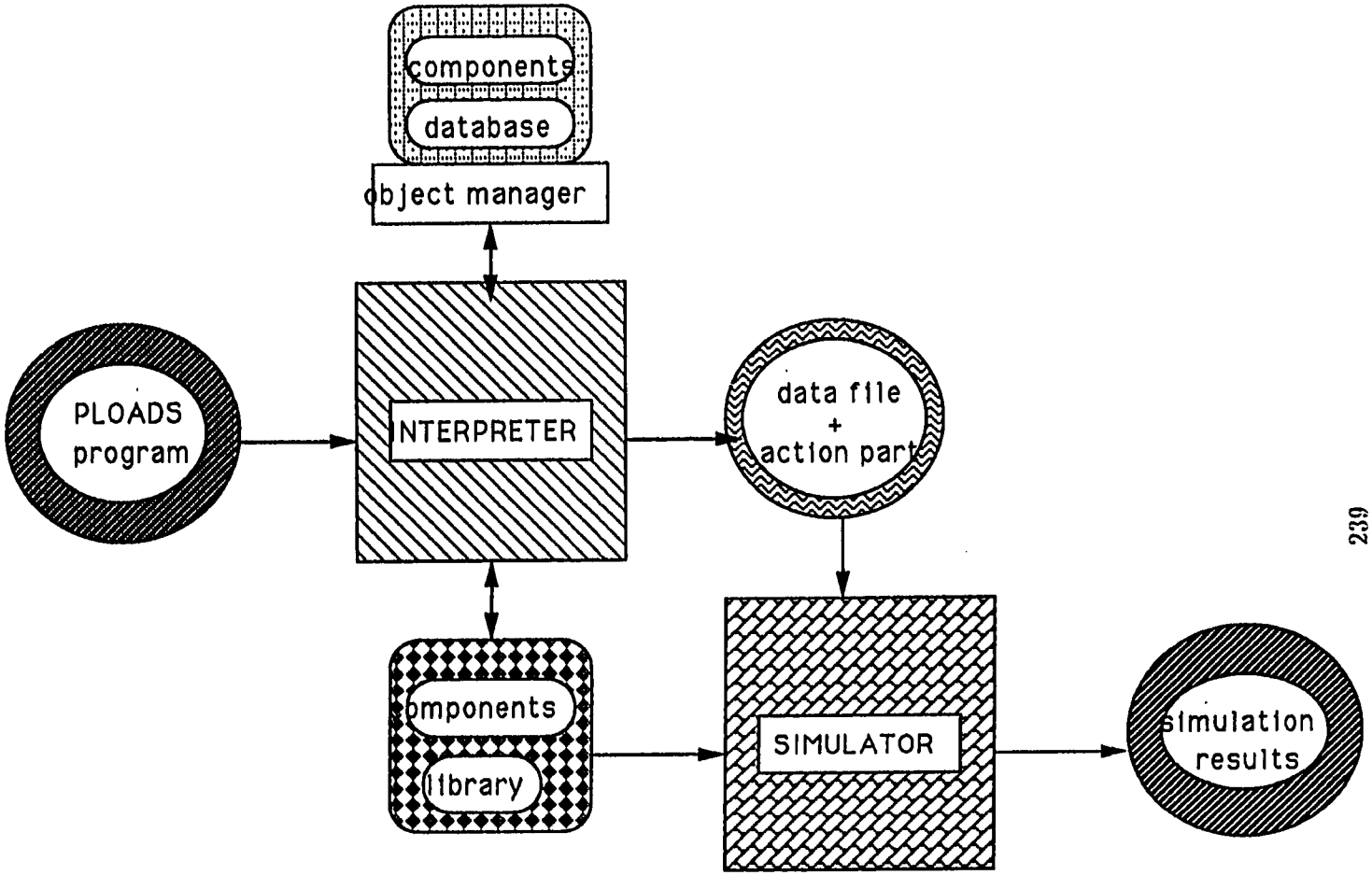


Figure 7.1 The overall system for PLOADS programs execution.

7.1 The PLOADS Interpreter

The first function of the interpreter is to check the syntax of the input program and produce proper messages if any error occurs. It is, also, to produce a detailed description of the optical architecture described by the input program. The input programs might contain assemblies or some constraints. This implies that relative placement of components within the architecture must have occurred. Relative placement makes the simulation of the architecture somewhat difficult. This, in turn, means that all these aspects of relative placement and constraint checking have to be fully resolved before the simulator starts functioning.

The different functions of the interpreter are done over a number of stages. Each of which is described in the following subsections in their execution order.

Figure 7.2 shows the interpreter's operational/computational stages.

One important note has to be made here and before we go into details of the system. That is, when any component is created by declaring a name or identification number for it in the declaration part, a record for that component is created and made part of a *data file*. The record has attributes that correspond to the attributes of the component model. The data file at the end of the interpretation will contain records with different sizes. Each of these records will correspond to different components of the file. The records are called *description records* and the data file is called the *description data file*.

7.1.1 The PLOADS Lexical Analyzer

To lexically analyze PLOADS programs, a function *yylex* was implemented using C on the Unix operating system.

The function *yylex* recognizes PLOADS *regular expressions*. The expressions

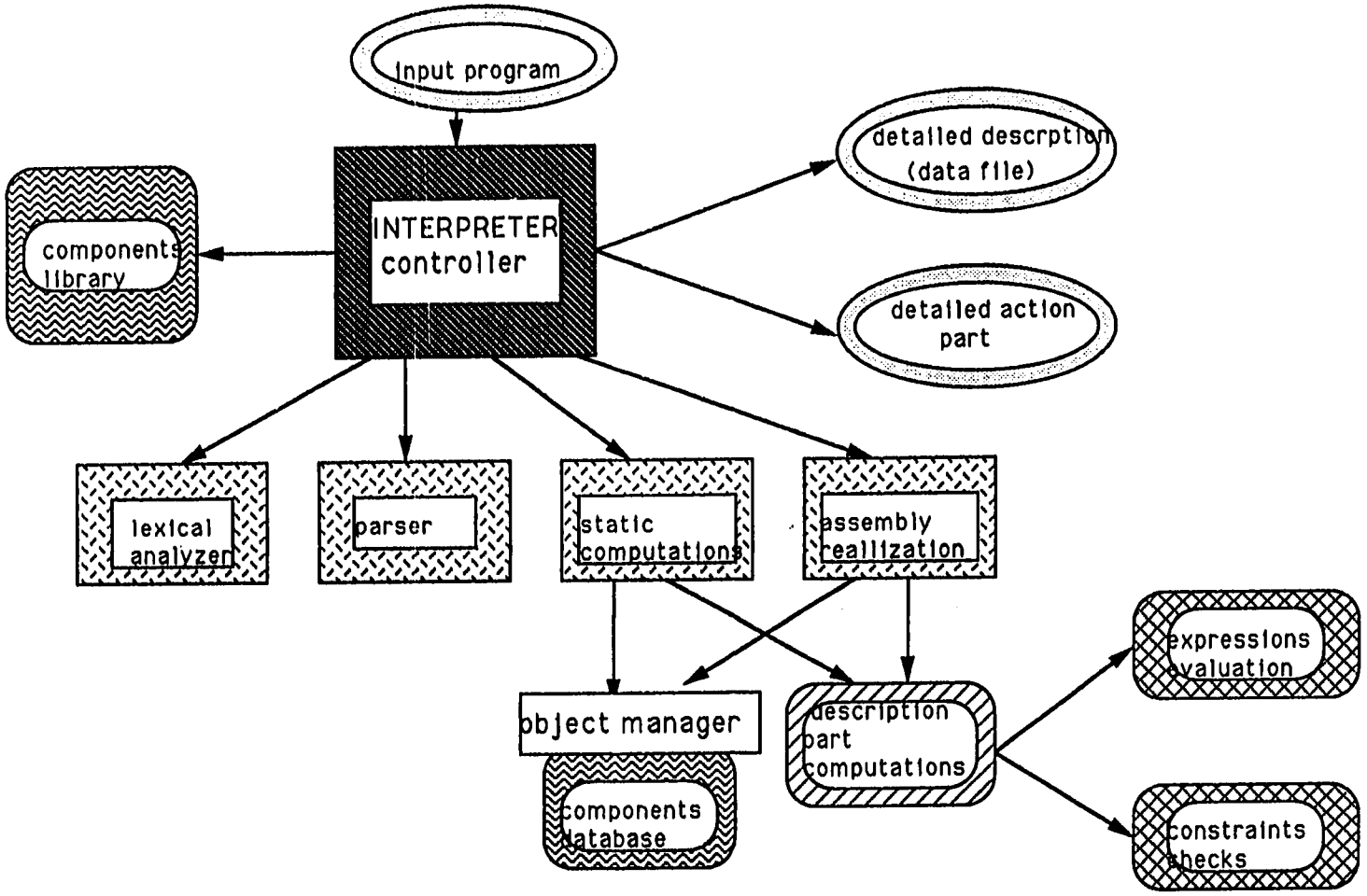


Figure 7.2 The PLOADS interpreter.

are recognized in a data stream (input), which is the PLOADS program. Upon recognition of expressions, and proper tokens, token numbers are returned.

This function was used together with *yacc*, (a parser generator) on the *Unix*, to perform the lexical analysis stage of PLOADS programs execution.

7.1.2 The PLOADS Parser

The syntax of the constructs used in the input PLOADS program are checked for correctness using this parser. Yacc was used to develop the parser. The parser makes use of the outputs of the function *yylex* described earlier.

The yacc program provides a general tool for imposing structure on the input to a computer program. We gave the yacc the grammar rules of PLOADS. The yacc, upon execution, generates a function to control the input process. This function, called the *parser*, calls the function *yylex* (the lexical analyzer) to pick the basic items (called *tokens*) from the input stream. These tokens are organized according to the input structure rules, called *grammar* rules. When one of these rules has been recognized, an action that we have given to that rule, if any, gets invoked. Actions help the interpreter in later stages for data preparation and code generation. The yacc program is written in a portable dialect of the C language. Figure 7.3 shows both the lexical analysis and parsing stages.

7.1.3 Static Computations

After assuring the correctness of the input program, the interpreter starts some computations, such as, evaluation of expressions or fetching records from the components database. We call these computations static because they do not change with time. They are unlike computations of the characteristics of light beams

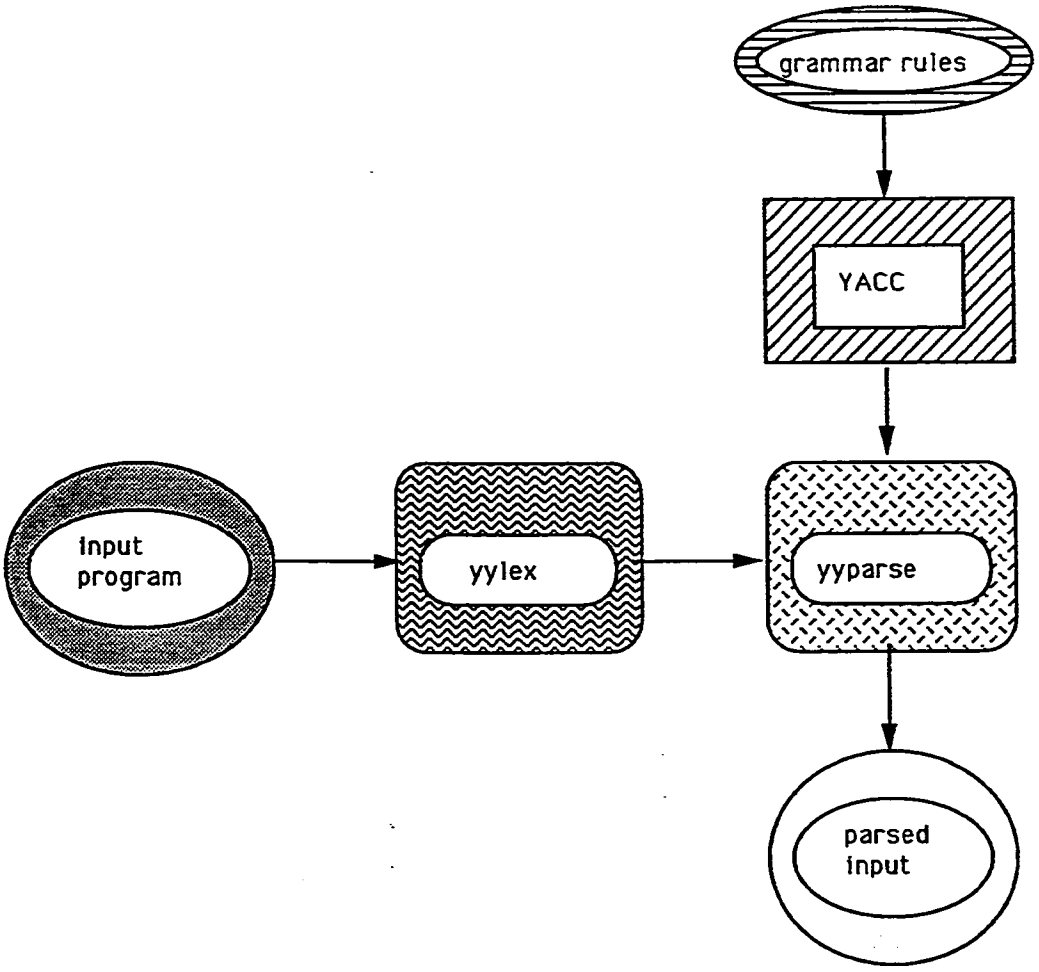


Figure 7.3 The lexical analysis and parsing of PLOADS programs.

within architectures, which change with time and position. Such computations are performed by the **simulator**.

While computations are being performed, the interpreter checks the semantics of the program. Examples of such checks are:

- The check that the type of the left hand side of an assignment statement is the same of the result of its right hand side.
- The check of the inputs to an assembly.

Other semantics have to be checked as well.

There are two types of static computations, these are discussed in what follows. In addition to these two types, the constraints enforced by the programmer have to be checked and applied.

In the constraints subpart of the program, relative placement computations have to be performed. This implies modifications over the records of the corresponding components. Other constraints are checked as well, these will be explained later on.

Non-database Calls

These computations are basically done to the description part of the program. There, two basic constructs could be found: The assignment statement, where expressions need to be evaluated. The others are control structures, such as the loop structures and the conditional structures.

These constructs have to be interpreted and computations needed for their interpretation have to be performed.

Realizations of assemblies are left for a later stage.

The Database Calls

Calls to the components database are passed to the object manager. It manages the database and performs the requirements implied in the calls.

The object manager gets the interpreted calls and executes them. It also creates records for components of the described architecture. These records are made available for the use of the system in later stages. Thereafter, the system works on these records rather than the records of the original components database.

Each component of the architecture is given a record indexed by its assigned component/object-id. The record contains all the attributes values of the component. Any modifications on these attributes are performed on the same record. Therefore, some link between the static computations part and the components database has to exist via the object manager. This is illustrated in Figure 7.2.

Constraint Checks and Applications

The constraints that are checked at this stage are:

- The dimensions of the overall architecture. This check is delayed if assemblies still need to be realized.
- The placement and positioning checks: The system makes sure that no two components take the same volume in space, or that no two components interfere in space.
- The system, in this stage, checks whether all the attributes of the components of the architecture have been assigned values.
- All relative placement statements present in the constraints part are evaluated and executed properly.

7.1.4 The Realization of Assemblies

This stage involves repetitive application of the *static computations* stage. The previous stage prepares the inputs that should be passed to the assemblies. Other computations within the assembly's body are conducted in exactly the same manner as in the *static computations* stage.

The system takes care of the placement computations of components of the assemblies. That is because their positions, directions, and orientations are all relative to those of the assembly's imaginary box. Hence, relative placement is what takes place in assemblies. The system handles those without concerning the programmer (user).

Again constraints have to be checked when realizing the assemblies in the same manner as before.

Inputs to and outputs from the assemblies are added as attributes to the description records of the corresponding components. For example, if a component called BS1 of some assembly gets input from the outer world, the system specifies that in the BS1's description record.

7.1.5 The Components Library Interface

This interface is needed if the program contains any component modification code.

The user of the PLOADS system is restricted to use only those components that have been modeled, as discussed in Chapter 3. In Chapter 4, we described procedures that simulate the behaviour of these components. This restriction over the usable components is imposed because the system needs to simulate the effects of the used components when simulating the behavior of the overall architecture. Simulating the behavior of the components is performed by calling the simulat-

ing procedures in the components library. Therefore, the system will not allow the usage of a component that is not modeled, and hence, that does not have a procedure that simulates its effects in the components library.

We find this restriction over the usable components unrealistic. That is because new components are being produced and put to use every now and then. In addition to that, optical components might be enhanced at any time. Having restrictions on the used components means that the system will become obsolete in a short time. This, in turn, means that new versions of the system will have to be developed in short periods. Newer versions should contain new components or enhanced old components. We solved this problem by introducing the *components modification* part of the **PLOADS** programs. There, the user can add new attributes to existing components. This will simulate the enhancement of the components. The user can also add new components with new attributes as well. For both cases, the user can write his own simulatin procedure that reflects the changes imposed by these modifications.

The simulation procedure provided by the user will then have to be part of the components library. Here is where we need the interface with the components library at this stage of interpretation.

7.1.6 Producing a Detailed Action Part

Producing a detailed action part implies that two basic operations have to be done. The first is the evaluation of any expression or computational control structure in the action part. The second is the expansion of action expression statements. For example, if some action statement contained a probe for some characteristic at a relative position inside some assembly, the expansion of such a probe means that

the exact position where probing is requested has to be evaluated. The evaluated position replaces the old relative position in the detailed action part. Another example is: assume that one of the action statements probes for a characteristic at a position inside an assembly that is called for more than one time. This implies that the probed for characteristic has to be computed at different positions within the architecture. This is done by expanding the probe statement into a number of similar statements at the different evaluated positions. A similar situation occurred in the example of Section 6.4.

7.1.7 Producing a Detailed Description of the Optical Architecture

At this stage, a description data file is created. It contains records for every component of the architecture with detailed attributes values. The values of these attributes are received from components database calls or resulted from static computations.

In addition to the description data file, the action part is now detailed and ready to be passed to the simulator for actual and detailed computations.

Let us now consider some examples to get a feeling of what the interpreter does for input programs.

7.2 EXAMPLE I : The Banyan Network

Since the interpreter is not fully implemented, we will discuss part of the interpretation of the program in Section 6.2. We will concentrate on important and special cases that are worth considering. The interpretation stages of this example

are discussed one in each of the following subsections.

7.2.1 Lexical Analysis

In the example (Section 6.2), the stream of input starts with the word MAIN. The function *yylex* will analyse that to be an **identifier** and will return a token value for it. As we move ahead, LS, LEN, QWP, and others are all considered to be identifiers. The 18.0, 633.0, 1 ,135, 90, and other values are analysed as **constants.**

When the function finds a /*, which indicates the beginning of a comment statement, it neglects all what follows till it reaches the */, which marks the end of the comment.

The GET_OBJECT, COPY_OBJECT, POSADD, and others are given tokens as if they were identifiers.

7.2.2 The Parser

The results of the lexical analysis in addition to the grammar rules of PLOADS are used to detect any syntax error in the input (program).

7.2.3 Static Computations

Expressions such as the following:

LS.position := (0, 0, 0);
fl := LEN1.focal_length;
PBS1.position := LS.position POSADD (NONE, DOWN fl, NONE);

are evaluated and their values are assigned to proper attributes or variables.

Database functions or **object** functions such as the following:

PBS1 := **GET_OBJECT**(PBS020);
LEN[4] := **COPY_OBJECT**(LEN[2]);

are interpreted as: the first is a call to the components database to get the proper record from it and store the values of its attributes in the proper attributes of the components description record. The second statement (function) copies all the attributes values of the record of LEN[2] in the attributes of the record of LEN[4].

Once we are done with the *placement* subpart, we move to the *constraint* subpart. Only position, face, and orientation expressions are expected in this subpart. These expressions are evaluated and results are assigned to the proper attributes or variables.

No dimension constraint was found. This leaves us with all components realized in the form of description records and all constraints satisfied.

Since there are no assemblies in the architecture's description, the stage of realizing assemblies will be trivial. And since there is no component modification part, its corresponding stage will be trivial too.

7.2.4 Producing a Detailed Action Part

The expressions:

pos := pos POSADD (LEFT 10, NONE, NONE);
t := 100;

are evaluated first. In the action part of this example, the system finds no need to detail any of the action expressions. They remain as two action expressions and a *print* statement.

After the accomplishment of the above stages, the description data file is now ready and so is the action part. This information is then passed to the simulator.

7.3 EXAMPLE II : Optical Implementation of a Crossover Interconnection Network

In this example, lexical analysis and parsing is done as in the previous example.

Numerical expressions are found in the placement part. These are evaluated as part of the static computations stage.

Constructs such as the **FOR** loop are executed in the static computations stage as well. Within the **FOR** loop, an assembly is being called. Its realization is left to the assemblies' realization stage.

7.3.1 Reallization of Assemblies

The assembly realization is accomplished by a static computations stage similar to the one for the main program. For each call to the assembly, the same assembly is realized at different positions, directions, and orientations.

7.3.2 Producing a Detailed Action Part

In the action part, the action expression probes for polarization at positions dependent on the position of a component in the assembly. This implies that information about the polarization of the light beam in each instance of the assembly is requested. The assembly in Section 6.4 was called 3 times and this expands the one action expression into three similar ones as follows:

p1 := POLAR AT instance1.LEN[4].position POSADD (NONE, DOWNfl, NONE);
p2 := POLAR AT instance2.LEN[4].position POSADD (NONE, DOWNfl, NONE);
p3 := POLAR AT instance3.LEN[4].position POSADD (NONE, DOWNfl, NONE);
PRINT p1, p2, p3;

Next, the three positions are evaluated and the result replaces the positions in the action expressions above.

Afterwards, the description data file together with the above detailed action statements is passed to the simulator.

7.4 The PLOADS Simulator

The seconds part of the overall system, as mentioned earlier, is the **simulator**. The simulator takes the ***description data file*** and the ***detailed action part*** as inputs from the interpreter.

Our simulator has three stages of execution. Figure 7.4 shows these stages. In what follows, we explain each stage in some details.

7.4.1 The Directed Graph Generator

Since light beams in optical architectures travel from one component to the other, and since light beams travel in straight lines between components, since all of that is true, any optical architecture could be described as a *directed graph*. The nodes of the directed graph are the components constituting the architecture. The directed edges of the graph, on the other hand, will represent the light beams travelling from one node (component) to the other.

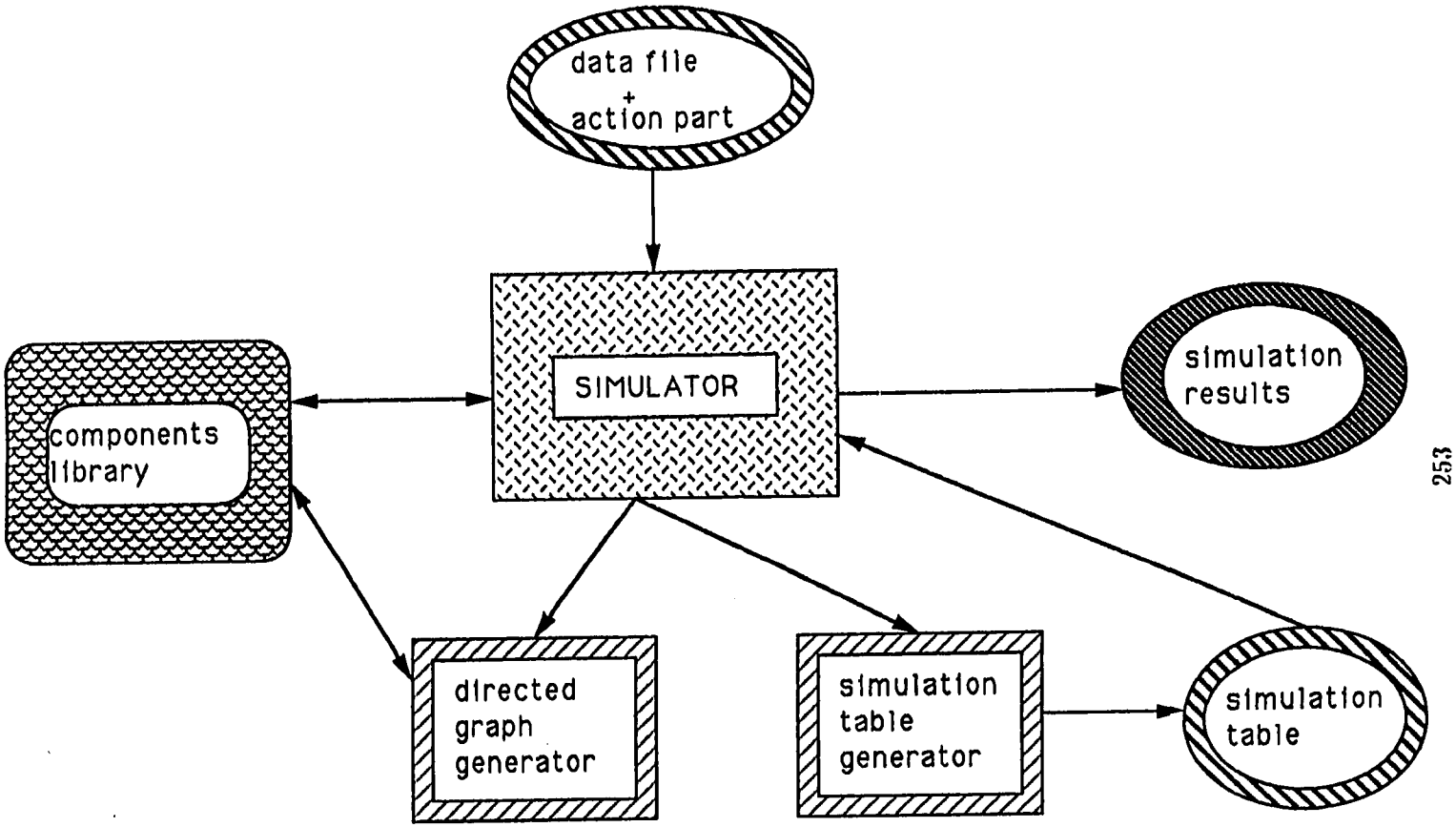


Figure 7.4 The basic stages of the PLOADS simulator.

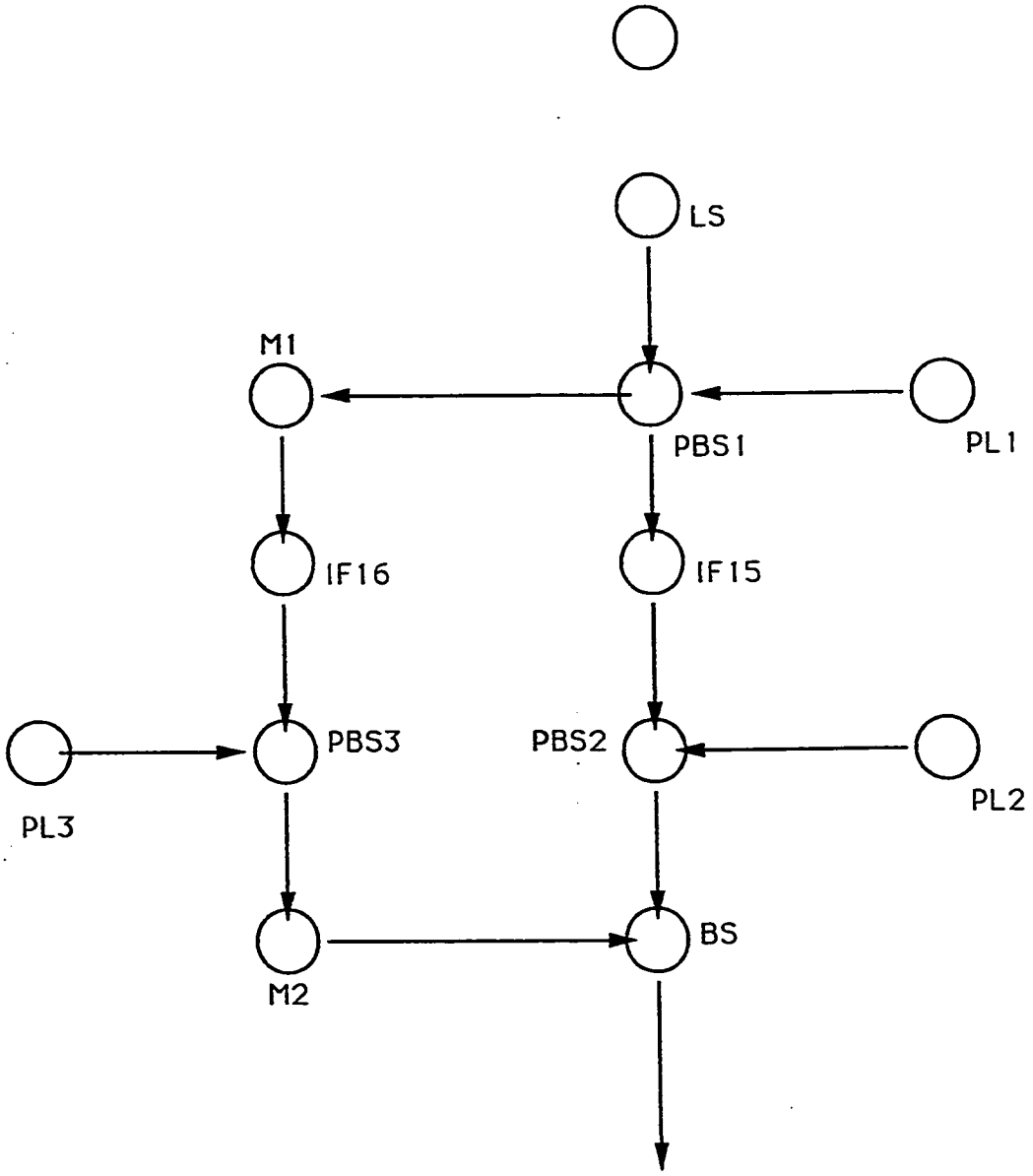


Figure 7.5 The directed graph of the architecture of Section 6.1.

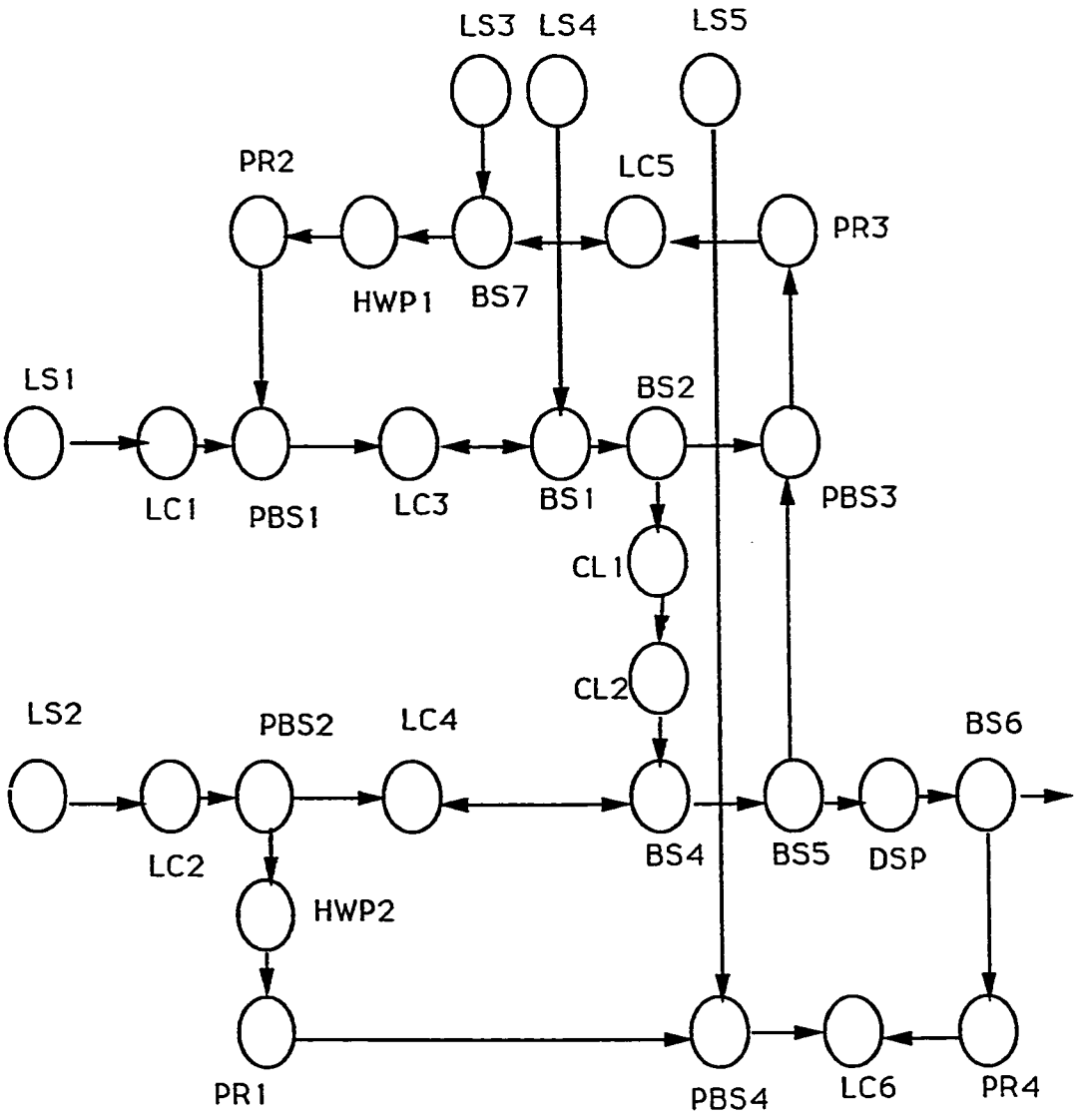


Figure 7.6 The directed graph for the architecture in Section 6.3.

Figures 7.5, 7.6 are the directed graphs for the optical architectures of Sections 6.1, 6.3.

The algorithm that generates these graphs is simple and in the following, we give it in pseudo code.

To each component (node), we assign two *lists* called the *input* and *output* lists. For laser sources, there is only an *output* list. Each member of the list is a doublet that consists of two values. The first is an *object-id* and the other value is a *direction*. For the input lists, the object-id refers to the component from which the input is coming, and the direction is the direction of the input beam. For the output lists, on the other hand, the object-id refers to the component to which the output is going, and the direction is the direction of the output beam.

The algorithm is:

```
directed_graph_generator(data_file, directed_graph, input_list, output_list)
begin
      loopfor all components in data_file
          if (the component is a laser source) then
            get the direction of its output
/* the output beam could be considered as a vector and the following
procedure is called to find out which plane (face of incidence) would it
intersect with first (i.e., which component does it hit first) */
            intersect(vector_dir, data_file, output_list, input_list)
          else
            if (the input_list of the component is not empty) then
              loop for each member of the list
                    call the procedure in the components
```

```
                    library that corresponds to the component
                    under consideration to get the directions
                    of the output beams from the component
                    according to the input
                    loop for each of the outputs
                        intersect(vector_dir, data_file,
                        input_list, output_list)
                    end loop outputs
                end loop members of the list
              endif
            endif
        end loop
end
intersect(vector_dir, data_file, input_list, output_list)
begin
        loop for all non-source components in data_file
            if (the output vector of the source intersects with any
              of the planes of incidence of the component) then
              add the object_id of the component to output_list
              together with the direction of the beam
              add the object_id of the source component to
              input_list of the intersected component together
              with the direction of the beam
/* the output_list is for the source of the beam and input_list
is for the component that the beam intersects with */
```

```
        endif
    end loop non-source
    loop for all the members of output_list
        -the nearest component to the source stays and the rest are
        omitted from the list because the beam will not reach them
        -as a result the object_id of the source component is
        omitted from the input_list of the component omitted from
        the output_list of the source
    end loop members
end
```

Using the *input* and *output* lists of each component, the directed graph can be easily drawn.

The graph of optical architectures is a three-dimensional graph.

7.4.2 The Simulation Table Generator

The action part of any PLOADS program contains requests for information about light beams within the architecture. Probing for such information has to be done at certain positions, and these positions have to be on one of the edges of the directed graph of the architecture.

Keeping that in mind, and to satisfy such requests of the user, the system generates a table called the *simulation table*. The rows of the table are the edges of the graph. The columns, on the other hand, are points in time starting with the value Δt and incremented by Δt. Hence, column 1 will give the characteristics of the light beam at the beginning of the corresponding edge (row).

Δt is by default 1 second. However, the user/programmer can change that to any other value (60 seconds, 0.001 second, 0.000001 second, ..., etc.) of his choice.

The edge is represented in the table by the names (object_ids) of the two nodes connected by the edge. Table 7.1 shows the simulation table for the graph in Figure 7.5.

The characteristics of the light beams are computed at the beginning of the edge. This means that they are computed as soon as the beam leaves the source component of the edge. If the position, where probing is requested, is on the edge and not at the beginning of it, these characteristics at the beginning of the edge (from the simulation table) are used to compute those at the specified position on the edge.

The table is not infinite in terms of columns. It depends on the availability of memory, the implementer's specifications, and the programmer's action requests.

7.4.3 The Action Part Computations

The needs of the user can now be satisfied by *table look-up* from the *simulation table.*

Any new actions that the user adds without changing the description part of the program are satisfied by table look-ups from the simulation table. This reduces the number of runs or interpretations of the program. All of the interpreter's stages can be skipped except the lexical analysis, parsing, and detailing the action part.

That was a brief description of the overall PLOADS system. Its full implementation will be suggested as future work.

characteristics at time units / edges of the directed graph	characteristics of the light beam at the specific edge after1 time unit	characteristics of the light beam at the specific edge after2 time units	characteristics of the light beam at the specific edge after3 time units	— — —
(LS, IF14)	.....			
(IF14, PBS1)	.....			
(PL1,PBS1)	.....			
| | |				

Table 7.1 The simulation table of the directed graph (architecture) of Figure 7.5.

Chapter 8

CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

The process of design and implementation of optical architectures was studied thoroughly. Problems such as the alignment and adjustment of these architectures were analysed. Our analysis went further to include the problem of *debugging* within the optical architectures.

We found out that one approach was taken by experimenters to tackle these problems. That approach was the *trial and error* approach of continuous alignment, adjustment, and replacement of the components of the architecture.

This study lead us to a very important conclusion: *debugging a simulated architecture is much easier and time and cost effective than debugging a real architecture. This conclusion was the driving force behind our work. Producing a simulated optical architecture was the challenge that we took and met successfully.*

To simulate optical architectures that could consist of a variety of optical components, a ***methodology*** had to exist that enables the experimenter/researcher to describe his/her architecture without any significant losses. In order to describe architectures, the experimenter has to describe each and every component of the architecture. The description of any components has to identify a number of aspects:

1. The location of the component within the architecture
2. The orientation and direction of the component with respect to the other components
3. The type of the component
4. The physical characteristics of the component

The physical characteristics of a component describe its effects on the light beam that passes through the component.

Therefore, the design and implementation of a *software tool* that helps researchers/experimenters simulate their designed architectures is achieved. To fully realize this tool, we

- **studied** a number of existing optical architectures and extracted the most commonly used components
- **developed a model** for each one of these components. Models that described the functionality and behaviour of the component when it is used within optical architectures
- **designed a library** that consists of a number of *procedures*. Each of these procedures simulates an optical component in terms of its physical effects.

The procedures are called by the system whenever the corresponding component apears in the simulated architecture. The library is *expandable* such that the addition of new procedures for new components is allowed and easily achieved

- **designed PLOADS:** ***a programming language for optical architectures description or specification.*** PLOADS programmers can describe any optical architectures. Adequate constructs and structures are available in PLOADS

- **designed** and partially **implemented** the *PLOADS system.* The system consists of two main parts that interpret PLOADS programs. These parts or stages of the system are the **interpreter** and the **simulator.**

- **designed** a database called the **components database.** The system accesses the database upon requests from the PLOADS user/programmer. The database contains files for optical components. Each record of each file presents the necessary information about the model of the corresponding optical components. Each file corresponds to a certain component. The database is managed by an *object manager.* This manager manipulates the database according to calls made from the system in response to requests of the users/programmers.

This tool represents the preparatory step to designing an **optical CAD** system. A system that automates the whole optical design process.

8.2 Future Work

A number of technical reports are being prepared for near future publication. These reports are on the following aspects of optical architectures usage:

1. PLOADS, a description language
2. Modeling of optical components for computer representation
3. The design of a library for optical components simulation
4. PLOADS interpreter and simulator, design and implementation aspects

For future work, we suggest the full implementation of the interpreter and simulator of PLOADS.

APPENDIX : THE PLOADS BNF GRAMMER

In this appendix, we give the *Beckus Nor Form* grammer of **PLOADS.**

```
<program>                 ::= MAIN <program_name>
                              ["{"<component_modification>"}"]
                              "{"<declaration>"}"  "{"<description>"}"  "{"<action>"}"
                              END ";"

<program_name>            ::= <identifier>

<declaration>             ::=  <declaration_statement>
                              | <declaration> <declaration_statement>
                              | <type_statement>
                              | <declaration> <type_statement>
                              | <function_definition>
                              | <declaration> <function_definition>

<declaration_statement>   ::=  <declared_variable> ":" <declaration_type> ";"
                              | <declared_variable> ":" <declaration_type> "="
                              <initializer> ";"
                              | <declared_variable> ":" <array> ";"
                              | <declared_variable> ":" <array> "="
                              <array_initializer> ";"
```

<declared_variable> ::= { <identifier> }

<declaration_type> ::= INT | FLOAT | LOGICAL | CHAR | POSITION
| ORIENTATION | FACE_DIRECTION
| POLARIZATION | TIME | POWER
| WAVELENGTH | FREQUENCY | ANGLE | PHASE
| <object_type> | LASER_LIGHT

<array> ::= <one_dimensional_array> |
<two_dimensional_array>

<one_dimensional_array> ::=ARRAY "[" <constant> "]" OF <declaration_type>

<two_dimensional_array> ::= ARRAY "[" <constant> "," <constant> "]" OF
<declaration_type>

<initializer> ::= <constant> | <string_constant>
| <logical_constant> | <position_constant> |
| <orientation_constant> | <polarization_constant>

<type_statement> ::= TYPEDEF "{"<type_definition>"}"

<type_definition> ::= { <declared_variable> "="
<declaration_type> ";" }
| { <user_defined_type> "="
"("<enumeration_value>")" ";" }

<enumeration_value> ::= { <declared_value> }

<declared_value> ::= IDENTIFIER | { <initializer }

<object_type> ::= CONTINUOUS_LASER | PULSED_LASER
| BEAM_SPLITTER | <identifier>

```
                                | FLAT_MIRROR | WINDOW | LENS | PRISM | GRATING
FILTER | DICHROIC_SHEET
                                | BIERFRINGENT_POLARIZER
                                | POLARIZING_BEAM_SPLITTER
                                | RETARDATION_PLATE | SPATIAL_LIGHT_MODULATOR

 <constant>                     ::= NUMBER

  <string_constant>             ::= STRING

  <logical_constant>            ::= "." TRUE "." | "." FALSE "."

  <position_constant>           ::= "("<arith_id> "," <arith_id> ","
                                <arith_id> ")"
  <orientation_constant>        ::= "("<arith_id> "," <arith_id> ")"

  <polarization_constant>       ::= UNPOLARIZED | CIRCULAR | ELLIPTICAL
                                | <plane_polarized>

 <plane_polarized>              ::= S_PLANE | P_PLANE | ANGLE_PLANE

  <array_initializer>           ::= "(" <detailed_array_initializer> ")"

  <detailed_array_initializer>::= { "(" <initializer_list> ")" }

  <initializer_list>            ::= { <initializer> }

  <function_definition>         ::= FUNCTION <function_name> "("
                                [<parameter_list>]")"
                                RETURN <declaration_type> [ "{" <declaration> "}" ]
                                "{"<function_body>"}"

  <function_name>               ::= <identifier>

  <parameter_list>              ::= { <identifier> }
```

```
<identifier>            ::= IDENTIFIER

<function_body>         ::=  { <action_assignment_statement> }
                        | { <comment_statement> }
                        | { <action_compound_statement> }
                        | { <action_control_statement> }
                        | { <print_statement> }
                        | { <return_statement> }

<comment_statement>     ::= COMMENT

<return_statement>      ::= RETURN ";"
                        | RETURN <safe_non_action_expression> ";"
                        | RETURN <action_expression> ";"
                        | RETURN <identifier> ";"

<print_statement>       ::= PRINT <printed_list> ";"

<printed_list>          ::= { <polarization_constant> }
                        | { <action_expression> }
                        | { <safe_non_action_expression> }

<description>           ::= "{"<all_placement>"}" ["{"<constraint>"}"]

<all_placement>         ::= {<placement> } | { <assembly_definition> }

<placement>             ::= <assignment_statement>
                        | <comment_statement>
                        | <compound_statement>
                        | <control_statement>
                        | <assembly_call>
                        | <media_definition>
                        | <wait_statement>
                        | <modify_attribute_statement>
                        | <medium_statement>
```

```
                        | <media_call>
                        | <add_to_database_statement>

<assignment_statement>  ::= <left_hand_side> ":="
                        <safe_non_action_expression> ";"
                            | <left_hand_side> ":=" <object_built_in_function> ";"
                        | <left_hand_side> ":=" <identifier>
                        | <left_hand_side> ":=" "(" <parameter_list> ")" ";"

<left_hand_side>        ::= { <identifier> }

<safe_non_action_expression>::= <arithmetic_expression
                        | <logical_expression>
                        | <string_expression>
                        | <position_expression>
                        | <orientation_expression>
                        | <face_expression>

<arithmetic_|expression> ::= <function|_call>
                        | <constant>
                        | <unary_expression>
                        | <arithmetic_expression> "+" <arithmeic_expression>
                        | <arithmetic_expression> "-" <arithmeic_expression>
                        | <arithmetic_expression> "*" <arithmeic_expression>
                        | <arithmetic_expression> "/" <arithmeic_expression>
                        | <arithmetic_expression> "DIV"
                        <arithmetic_expression>
                        | <arithmetic_expression> "^" <arithmeic_expression>
                        | "(" <arithmetic_expression> ")"

<function_call>         ::= <function_name> "("[<input_parameter_list>]")"

<input_parameter_list>  ::= <parameter_list>
                        | <arith_id>

<unary_expression>      ::= "-" <arith_id>
```

```
<arith_id>                  ::= <arithmetic_expression> | <identifier>

<logical_expression>        ::=  <arith_id> <comparison_op> <arith_id>
                            | "!" "(" <logic_id> ")"
                            | <logical_constant>
                            | <binary_logical_expression>

<binary_logical_expression>::= <bin_logic_id>  "&&"  <bin_logic_id>
                            | <bin_logic_id>  "| |"  <bin_logic_id>

<bin_logic_id>              ::= <binary_logical_expression>
                            | <logical_constant>
                            | <logic_id>

<comparison_op>             ::= "<" | ">" | "<=" | ">=" | "==" | "!="

<string_expression>         ::= <string_constant>
                            | <binary_string>
                            | <identifier> "[" <arith_id> ":" <arith_id> "]"

<binary_string>             ::= <string_expression> "//" <identifier>
                            | <string_expression> "//" <string_expression>
                            | <identifier> "//" <identifier>
                            | <identifier> "//" <string_expression>

<face_expression>           ::= <face_direction>
                            | FACEMIN <face_id>

<face_id>                   ::= <identifier>
                            | <face_declarator>
                            | <face_expression>
<face_declarator>           ::= <object_id> "." <face_direction>

<face_direction>            ::= "(" <direction_x> "," <direction_y>
                            "," <direction_z> ")" ";"
```

```
<direction_x>              ::= RIGHTX | LEFTX | UNDEF

<direction_y>              ::= FORTHY | BACKY | UNDEF

<direction_z>              ::= UPZ | DOWNZ | UNDEF

<position_expression>      ::=  <position_constant>
                           | POSUNMIN <posit_id>
                           | <posit_id> POSADD <displacement>
                           | <position> "*" <orientation_vector>
                           | <position_expression> "*" <posit_id>
                           | <arith_id> POSMUL <posit_id>

<position_declarator>      ::= <object_id> "." <position>

<orient_id>                ::= <orientation_expression>
                           | <identifier>
                           | <orientation_declarator>

<orientation_declarator>   ::= <object_id> "." orientation

<posit_id>                 ::= <position_expression>
                           | <identifier>
                           | <position_declarator>

<displacement>             ::= "(" <x_direction> [ <displace> ] ","
                           <y_direction> [ <displace> ] ","
                           <z_direction> [ <displace> ] ")"

<displace>                 ::= <arith_id>

<x_direction>              ::= RIGHT | LEFT | NON

<y_direction>              ::= BACK | FORTH | NON
```

```
<z_direction>                ::= UP | DOWN | NON

<orientation_expression>     ::=  <orientation_constant>
                             | ORUNMIN <orientation>
                             | <orient_id> ORADD <angular_displacement>

<angular_displacement>       ::= "(" <arith_id> "," <arith_id> ")"

<object_built_in_function>::=  <get_object>
                             | <copy_object>
                             |  <get_attribute>
                             | <get_default>

<add_to_database_statement>::=  <add_default>
                             | <add_attribute>

<get_object>                 ::= GET_OBJECT "(" <identifier> ")"

<copy_object>                ::= COPY_OBJECT "(" <object_id> ")"

<get_attribute>              ::= <object_id> "." <identifier>

<get_default>                ::= GET_DEFAULT "(" <identifier> ","
                             <attribute_identifier>  ")"

 <attribute_identifier>      ::=    <identifier> | <attribute_identifier> ","  <identifie

<add_default>                ::= ADD_DEFAULT "(" <object_id> ")"

<add_attribute>              ::= ADD_ATTRIBUTE "("<object_id> "," <identifier> ","
                             <identifier> "," <identifier> ")"

<object_id>                  ::= <identifier>

<compound_statement>         ::= "{" [ "{"<declaration>"}" ] <placement>"}"
```

<control_statement> ::= <conditional>
| <while>
| <do>
| <for>
| <break>
| <goto>
| <labeled>

<conditional> ::= IF <logic_id> <compound_statement>
| IF <logic_id> THEN <compound_statement>
ELSE <compound_statement>

<while> ::= WHILE <logic_id> <compound_statement>

<do> ::= DO <compound_statement> WHILE <logic_id> ";"

<for> ::= FOR <identifier> "=" <arith_id> "," <arith_id>
["," <arith_id>] <compound_statement>

<break> ::= BREAK ";"

<continue> ::= CONTINUE ";"

<goto> ::= GOTO <label> ";"

<label> ::= <identifier>

<labeled> ::= <label> ": :" <labeled_statement>

<labeled_statement> ::= <assignment_statement>
| <while>
| <do>
| <for>
| <break>
| <modify_attribute_statement>

```
<modify_attribute_statement>::= <object_id> "." <identifier> "="
                                <safe_non_action_expression> ";"
                                | <object_id> "." <identifier> "=" <object_id> "."
                                <idfentifier> ";"
                                | <object_id> "." <identifier> "=" <identifier> ";"

 <medium_statement>             ::= MEDIUM "(" <arith_id> "," <arith_id> ")" ";"

 <media_definition>             ::= "{" MEDIA <media_name> "(" <media_type> ","
                                <dimensions> "," <posit_id> [ "," <face_direction> ]
                                [ "," <orient_id> ] ")"
                                [ "{" <declaration> "}" ] "{" <description> "}"

<media_type>                    ::= CYLINDER
                                | BOX

<dimensions>                    ::= "(" <arith_id> "," <arith _id> "," <arith_id> ")"
                                | "(" <arith_id> "," <arith_id> ")"

 <assembly_definition>          ::= ASSEMBLY <asembly_name> <assembly_declarator>
                                [ "{"<declaration>"}" ]  "{"<description>"}"

<assembly_name>                 ::= <identifier>

 <assembly_declarator>          ::= "(" [ <parameter_list> "," ] INPUT ":"
                                <parameter_list> "," OUTPUT ":" <parameter_list> ")"

 <assembly_call_declarator>::= "(" <parameter_list> ")"

 <assembly_call>                ::= ASSEMBLY <assembly_name>
                                <assembly_call_declarator> ";"

 <media_call>                   ::= MEDIA <media_name> ";"

 <media_name>                   ::= <identifier>
```

```
<wait_statement>          ::= WAIT_FOR <arith_id> AT <posit_id> ";"

<constraint>              ::=  { <constraint_statement> }
                          | { <dimension_statement> }

<constraint_statement>    ::=  <position_declarator> ":=" <posit_id> ";"
                          | <orientation_declarator> ":=" <orient_id> ";"
                          | <face_declarator> ":=" <face_id> ";"

<dimension_statement>     ::= ARCHITECTURE_DIMENSION "(" <posit_id> ","
                          <posit_id> ")" ";"

<action>                  ::=  { <action_assignment_statement> }
                          | { <comment_statement> }
                          | { <action_compound_statement> }
                          | { <action_control_statement> }
                          | { <draw_diagram> }
                          | { <print_statement> }
                          | { <non_parameter_function_call> }
                          | { <time_unit_statement> }

<non_parameter_function_call>::= <function_name> "("  ")" ";"

<action_assignment_statement::=  <left_hand_side> ":="
                          <safe_non_action_expression> ";"
                          | <left_hand_side> ":=" <action_expression> ";"
                          | <left_hand_side> ":=" <identifier> ";"
                          | <left_hand_side> ":=" <get_attribute> ";"

<action_expression>       ::=  <characteristics> AT <position_expression> [ AT
                          <arithmetic_expression> ]

<characteristics>         ::= POWER | POLARIZATION | BEAM_SPLITTER | PHASE
                          | WAVELENGTH | FREQUENCY

<action_compound_statement>::= "{" [ "{"<declaration>"}" ] <action>"}"
```

```
<action_control_statement>::=   <action_conditional>
                              | <action_while>
                              | <action_do>
                              | <action_for>
                              | <break>
                              | <goto>
                              | <action_labeled>

<action_conditional>          ::=  IF <logic_id> <action_compound_statement>
                              | IF <logic_id> THEN <action_compound_statement>
                              ELSE <action_compound_statement>

<action_while>                ::= WHILE <logic_id> <action_compound_statement>

<action_do>                   ::= DO <action_compound_statement> WHILE
                               <logic_id> ";"

<action_for>                  := FOR <identifier> "=" <arithmetic_expression> ","
                              <arithmetic_expression>["," <arithmetic_expression>]
                              <action_compound_statement>

<action_labeled>              ::= <label> ": :" <action_labeled_statement>

<action_labeled_statement>::=   <action_assignment_statement>
                              | <draw_diagram>
                              | <print_statement>
                              | <action_conditional>
                              | <action_while>
                              | <action_do>
                              | <action_for>
                              | <break>

<draw_diagram>                ::=  DRAW_DIAGRAM <identifier> ";"

<time_unit_statement>         ::= TIME_UNIT ":=" <arith_id> ";"
```

```
<component_modification> ::=  { <add_component> <add_component_attribute>
                          [ <modification_routine> ] }
                          | { <lib_add_attribute> [<modification_routine>] }

<lib_add_attribute>      ::= LIB_ADD_ATTRIBUTE "("<object_type> ","
                          <identifier> "," <identifier> "," <identifier> ")" ";"

<add_component>          ::= ADD_COMPONENT "(" <identifier> ")" ";"

<add_component_attribute>::= { <lib_add_attribute> }

<modification_routine>   ::= "{" <routine_declaration> "}" "{" <routine_body> "}"

<routine_declaration>    ::= { <declaration_statement> }
                          | { <type_statement> }

<routine_body>           ::=  <regular_assignment_statement>
                          | <comment_statement>
                          | <regular_compound_statement>
                          | <regular_control_statement>

<regular_assignment_statement::=  <left_hand_side> "="
                          <safe_non_action_expression> ";"
                          | <left_hand_side> ":=" <identifier> ";"

<regular_compound_statement> ::= "{" [ "{"<routine_declaration>"}" ]
                          <routine_body> "}"

<regular_control_statement>  ::=  <regular_conditional>
                              | <regular_while>
                              | <regular_do>
                              | <regular_for>
                              | <break>
                              | <goto>
                              | <regular_labeled>
```

```
<regular_conditional>      ::= IF <logic_id>
                           <regular_compound_statement>
                           | IF <logic_id> THEN <regular_compound_statement>
                           ELSE  <regular_compound_statement>

<regular_while>            ::= WHILE <logic_id>
                           <regular_compound_statement>

<regular_do>               ::= DO <regular_compound_statement> WHILE
                           <logic_id> ";"

<regular_for>              ::= FOR <identifier> "=" <arithmetic_expression> ","
                           <arithmetic_expression>["," <arithmetic_expression>]
                           <regular_compound_statement>

<regular_labeled>          ::= <label> ": :" <regular_labeled_statement>

<regular_labeled_statement>::=  <regular_assignment_statement>
                           | <regular_conditional>
                           | <regular_while>
                           | <regular_do>
                           | <regular_for>
                           | <break>
                           | <draw_diagram>
                           | <print_statement>
```

REFERENCES

[ANJA87] A. Ghosh, and P. Paparao, "Matrix preconditioning : a robust operation for optical linear algebra processors," ***Applied Optics, Vol. 26,*** Number 14, July 1987, 2734-2737 .

[BIAN90] S. Bian, K. Xu, and J. Hong, "Near Neighboring Neurons Interconnected Neural Network," ***Optics Communications, Vol. 76,*** Number 3,4, May 1990, 199-202.

[CAI 90] L. Z. Cai and T. H. Chao, "Optical Image Subtraction using a LCTV SLM and White-light Imaging Grating Interferometer," ***Journal of Modern Optics, Vol. 37,*** Number 6, 1990, 1127-1138.

[CASA77] David Casasent, "Spatial Light Modulators," ***Proceedings of the IEEE, Vol. 2,*** Number 1, 1977, 143-157.

[DATT89] A. K. Datta, A. Basuray, and S. Mukhopadhyay, "Arithmetic operations in optical computations using a modified trinary number system," ***Optics Letters, Vol. 14,*** Number 9, May 1989, 426-428.

[DEJA87] D. V. Pantelic, "Optical computation of determinants," ***Optics Communications, Vol. 64,*** Number 5, 1987, 421-424 .

[FATE84] M. T. Fatehi, K. C. Wasmundt, and S. A. Collins, "Optical Flip-Flops and Sequential Logic Circuits Using a Liquid Crystal Light Valve," ***Applied Optics, Vol. 23,*** Number 13, July 1984, 2163-2171.

[FEIT88] D. G. Feitelson, "Optical Computing," The MIT Press, 1988.

[FERR90] C. Ferreira and C. Vazquez, "Anamorphic Multiple Matched Filter for Character Recognition, Performance with Signals of Equal Size," ***Journal of Modern Optics, Vol. 37,*** Number 8, 1990, 1343-1354.

[FLAV90] M. A. Flavin and J. L. Horner, "Average Amplitude Matched Filter," ***Optical Engineering, Vol. 29,*** Number 1, January 1990, 31-37.

[FUKU90] Seiji Fukushima, Takashi Kurokawa, and Hideo Suzuki, "Optical Implementation of Parallel Digital Adder and Subtractor," ***Applied Optics, Vol. 29,*** Number 14, May 1990, 2099-2106.

[GHAF90] A. Ghafoor, M. Guizani, S. Sheikh, "Architecture of an All-Optical Circuit-Switched Multistage Interconnection Network," ***IEEE Journal on selected areas in Communications, Vol. 8,*** Number 8, October 1990, 1595-1607.

[GIND88] G. R. Gindi, A. F. Gmitro, and K. Parthasarathy, "Hopfield Model Associative Memory with Nonzero-Diagonal Terms in Memory Matrix," ***Applied Optics, Vol. 27,*** Number 1, January 1988, 129-134.

[GIAN84] Douglas C. Giancoli, "General Physics," Prentice-Hall, INC., 1984.

[GUIZ90] M. Guizani "An All-Optical Multistage Interconnection Network for Supercomputer Systems," **Phd desertation, in Electrical Engineering in the L.C. Smith College of Engineering, Syracuse University,** August 1990.

[HARA90] K. Hara, K. Kojima, K. Mitsunga, and K. Kyuma, "Optical Flip-Flop Based on Parallel-Connected AlGaAs/GaAs pnpn Structures," ***Optics Letters, Vol. 15,*** Number 13, July 1990, 749-751.

[HEAV91] O.S. Heavens and R.W. Ditchburn, "Insight into Optics," John Wiley & Sons, 1991.

[HECH87] Eugene Hecht, "Optics," Addison-Wesley Publishing Company, 1987.

[HONG90] H. Huang, L.Liu, and Z. Wang, "Parallel multiple matrix multiplication using an orthogonal shadow-casting and imaging system," ***Optics Letters, Vol. 15,*** Number 19, October 1990, 1085-1087.

[ISLA90] M. N. Islam, "All-Optical Cascadable NOR Gate with Gain," ***Optics Letters, Vol. 15,*** Number 8, April 1990, 417-419.

[JAHN88] J. Jahns and S. M. J. Murdocca, "Crossover Networks and their optical implementation," ***Applied Optics, Vol. 27,*** Number 15, August 1988, 3155-3160.

[JAHN90a] J. Jahns and S. J. Walker, "Imaging with Planar Optical Systems," ***Optics Communications, Vol. 76,*** Number 5,6, May 1990, 313-317.

[JAHN90b] J. Jahns and B. A. Brumback, "Integrated -Optical Split-and Shift Module Based on Planar Optics ," ***Optics Communications, Vol. 76,*** Number 5,6, May 1990, 318-320.

[JAHN90c] J. Jahns, "Optical implementation of the Banyan network," ***Optics Communications, Vol. 76,*** Number 5,6, May 1990, 321-324.

[JOHN90a] H. J. Caulfield "Improved relaxation processor for parallel solution of linear algebraic equations," ***Applied Optics, Vol. 29,*** Number 20, July 1990, 2978-2981.

[JOHN90b] H. J. Caulfield, J. Shamir, J. E. Ludman, and P. Greguss, "Reversibility and energetics in optical computing," ***Optics Letters, Vol. 15,*** Number16, August 1990,912-914.

[KARL88] K. Brenner, and A. Huang, "Optical implementations of the perfect shuffle interconnection," ***Applied Optics, Vol. 27,*** Number 1, January 1988, 135-137.

[LALA87] P. Lalanne, J. Taboury, and P. Chavel "A Proposed Generalization of Hopfield's Algorithm," ***Optics Communications, Vol. 63,*** Number 1, July 1987, 21-25.

[LIA 81] Laser Institute of America, "A Short Course on Laser Safety," Laser Institute of America and Engineering Technology, INC., July 1981.

[LIN 89] S. Lin, L. Liu, and Z. Wang, "Optical Implementation of the 2-D Hopfield Model for a 2-D Associative Memory," ***Optics Communications, Vol. 70,*** Number 2, February 1989, 87-91.

[MELL85] MELLES GRIOT, "Optics Guide 3," MELLES GRIOT, 1985.

[MIRS90] M. Mirsalehi, and T. K. Gaylord, "Analytic expressions for the sizes of logically minimized truth tables for binary addition and subtraction," ***Applied Optics, Vol. 29,*** Number 23, August 1990, 3339-3344.

[MUKH90] S. Mukhopadhyay, "AN OPTICAL CONVERSION SYSTEM : FROM BINARYTO DECIMAL AND DECIMAL TO BINARY," ***Optics Communications, Vol. 76,*** Number 5,6, May 1990, 309-312.

[MURT88] V. K. Murthy, "Exact parallel matrix inversion using para-Hensel codes with systolic processors," ***Applied Optics, Vol. 27,*** Number 10, May 1988, 2022-2024 .

[NEFF87] J. A. NEFF, "Major initiatives for optical computing," ***Optical Engineering, Vol. 26,*** Number 1, January 1987, 002-009.

[NEWP90] NEWPORT,"NEWPORT 1990 Catalogue," NEWPORT Coorporation, 1990.

[OITA90] M. Oita, J. Ohta, S. Tai, and K. Kyuma, "Optical Implementation of Large-scale Neural Networks Using a Time-Division-Multiplexing Technique," ***Optics Letters, Vol. 15,*** Number 4, February 1990, 227-229.

[ORIE85] ORIEL, "Light Sources Monochromators Detection Systems," ORIEL, Vol. 2, 1985.

[PERI87] D. Peri "Optical Implementation of a Phase Retrieval Algorithm," ***Applied Optics, Vol. 26,*** Number 9, May 1987, 1782-1785.

[SENI90] J. M. Senior and S. D. Cusworth, "Wavelength Division Multiplexing in Optical Fibre Sensor Systems and Networks: A Review," ***Optics & Laser Technology, Vol. 22,*** Number 2, 1990, 113-126.

[SIEG71] Siegman A. E., "An Introduction to Lasers and Masers," McGraw-Hill, 1971.

[WHER87] B. S. Wherret, S. Desmond Smith, F. A. P. Tooley, and A. C. Walker, "Optical Components for Digital Optical Circuits," ***Future Generation Computer Systems, Vol. 3,*** 1987, 253-259.

[WHIT85] D. R. J. White, K. Atkinson, and J. D. M. Osburn, "Taming EMI in microprocessor systems," ***IEEE Spectrum, Vol 22,*** Number 12, December1990,30-37.

[YEE 90] E. Yee and J. Ho, "Neural Network Recognition and Classification of Aerosol Particle Distributions Measured with a Two-Spot Laser Velocimeter," ***Applied Optics, Vol. 29,*** Number 19, July 1990, 2929-2938.

[ZHAN90] L. Zhang and L. Liu "Incoherent Optical Implementation of 2-D Complex Discrete Fourier Transform and Equivalent 4-f System," ***Optics Communications, Vol. 74,*** Number 5, January 1990, 295-300.