OUKA
Osaka University Knowledge Archive

Title	Programming of optical array logic. 1 : Image data processing
Author(s)	Tanida, Jun; Ichioka, Yoshiki
Citation	Applied Optics. 1988, 27(14), p. 2926-2930
Version Type	VoR
URL	https://hdl.handle.net/11094/3108
rights	
Note	

Osaka University Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

Osaka University

Programming of optical array logic. 1: Image data processing

Jun Tanida and Yoshiki Ichioka

Procedures for planning and executing arbitrary parallel processing with optical array logic are generalized as a systematic programming technique of optical parallel processing. Optical array logic is a technique for achieving any parallel neighborhood operation with simple coding and optical correlation. An original symbolic notation facilitates programming of parallel processing with optical array logic, so that many problems can be optically solved using optical array logic. Two examples of image data processing are presented to illustrate the programming procedure of parallel processing with optical array logic.

I. Introduction

Parallel nature is one of the attractive characteristics of optical information processing, as well as its communication capabilities. Using the parallelism of optics, massive data such as images can be processed efficiently. However, conventional optical information processing lacks controllability and flexibility, so that it is important to develop efficient methods for controlling optical information processing.

One of the promising methods for this purpose is the use of digital schemes. It is not too much to say that today's prosperity of electronic computers is based on the flexibility of digital processing, so that, using a digital scheme in optical information processing, controllable and flexible optical parallel processing can be expected. Under this concept a wide range of optical digital processing has been studied.[1-5]

We have developed an optical implementation of parallel logic gates using image coding and shadow-casting,[6] and have generalized the method as optical array logic (OAL).[7] OAL is a technique to achieve any parallel neighborhood operation for two binary images. Using OAL, controllability and programmability of processing can be obtained and an effective optical parallel computing system can be constructed. The optical computing system is called the optical parallel array logic system (OPALS).[8]

The authors are with Osaka University, Department of Applied Physics, Suita, Osaka 565, Japan.

Received 6 May 1987.
0003-6935/88/142926-05$02.00/0.

© 1988 Optical Society of America.

In this paper we describe the procedure for programming OAL for space-invariant processing. In Sec. II OAL is explained with logical expressions to clarify the correspondence between OAL and logical neighborhood operation. In Sec. III a generalized procedure for programming parallel processing with OAL is discussed. In Sec. IV two kinds of image data processing are presented as programming examples of OAL.

II. Concept of OAL

OAL is a way of achieving any parallel neighborhood operation for two 2-D binary data or binary images.

To help the following illustration, the logical meaning of OAL will be clarified in this section. A more intuitive explanation of OAL was presented in Refs. 7 and 8.

Let us consider two input binary images A and B consisting of $N \times N$ pixels. Any parallel neighborhood operation can be expressed by

$$c_{i,j} = f(\mathbf{a}_{i,j}, \mathbf{b}_{i,j}), \qquad (i,j = 1, \ldots, N), \tag{1}$$

where $f(\mathbf{a}_{i,j}, \mathbf{b}_{i,j})$ is the logical function for vectors of logical variables $\mathbf{a}_{i,j}$ and $\mathbf{b}_{i,j}$ defined as

$$\mathbf{a}_{i,j} = \{a_{i+m,j+n} | m,n = -L, \ldots, L\}, \tag{2}$$

$$\mathbf{b}_{i,j} = \{b_{i+m,j+n} | m,n = -L, \ldots, L\}. \tag{3}$$

In this paper the neighborhood area is defined as a square area of size L around a current pixel as shown in Fig. 1; a and b mean binary data in binary images A and B, respectively; the subscripts denote location of the data in the image.

Figure 2 shows the processing procedure of OAL. Two objects to be processed are encoded by the coding rule shown in Fig. 2 and converted into a coded image. The coded image is separately correlated with differ-

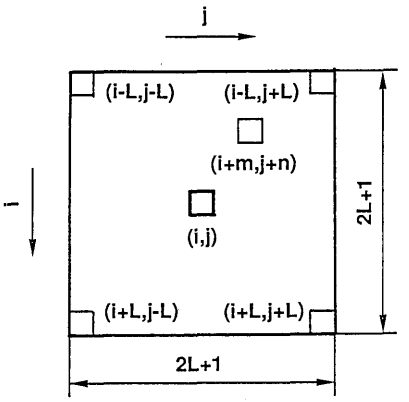


Fig. 1. Neighborhood area.

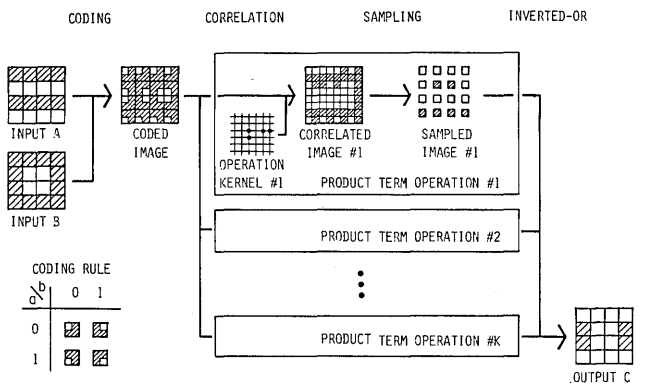


Fig. 2. Processing procedure of OAL.

ent pointwise function patterns called operation kernels. Individual correlated images are spatially sampled at one pixel intervals (double the size of a structural cell of the correlated images) along vertical and horizontal directions. After sampling, parallel NAND, or more exactly inverted-OR, operation for all the sampled images gives the result of parallel neighborhood operation. Throughout the operation correlated images are binarized as black-and-white images.

Parallel neighborhood operation executed in OAL is designated by a set of operation kernels used for the correlation. Using logical expressions, we can write the parallel neighborhood operation in Fig. 2 as follows:

$$c_{i,j} = \sum_{k=1}^{K} \prod_{m=-L}^{L} \prod_{n=-L}^{L} f_{m,n;k}(a_{i+m,j+n}, b_{i+m,j+n}), \quad (i,j = 1, \ldots, N), \quad (4)$$

where Σ and Π denote logical sum and logical product, respectively; $f_{m,n;k}(a,b)$ means a two-variable binary logic function for pixels a and b at the corresponding location in two input objects; subscripts m and n indicate relative address of the pixel in the neighborhood area centering on (i,j) pixels; and k is the identifier of product terms in parenthesis in Eq. (4). Note that any parallel neighborhood operation can be expressed by a combination of the $f_{m,n;k}$ terms, which corresponds to selecting a set of operation kernels in OAL. Thus it is obvious that programming in OAL is nothing but defining a set of operation kernels required for current processing.

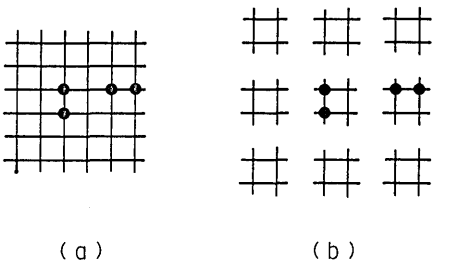


Fig. 3. Decomposition of an operation kernel into kernel units.

KERNEL UNIT	FUNCTION	SYMBOL	KERNEL UNIT	FUNCTION	SYMBOL
	1	..		a+b	PP
	$\bar{a}+\bar{b}$	NN		a⊕b	UU
	$\bar{a}+b$	NP		b	.1
	$\bar{a}$	0.		$\bar{a}b$	01
	$a+\bar{b}$	PN		a	1.
	$\bar{b}$	.0		$a\bar{b}$	10
	$\overline{a\oplus b}$	EE		ab	11
	$\bar{a}\bar{b}$	00		0	DD

Fig. 4. All patterns of a kernel unit corresponding to a two-variable binary logic function. Function symbols used for symbolic notation of OAL are also tabulated.

To clear the relationship between $f_{m,n;k}$ and the operation kernel, we introduce the concept of a kernel unit to describe operation kernels. As presented in Ref. 7, an array pattern of four-point sources configures an operation between the corresponding pixels in two input images, namely, the array pattern of the four-point sources is a primitive of operational configuration. A kernel unit is defined as an array pattern of the four points comprising operation kernels, so that any operation kernel can be decomposed into several kernel units as shown in Fig. 3. Using the kernel unit concept, a pattern of a kernel unit directly corresponds to a function of $f_{m,n;k}$ as shown in Fig. 4. Subscripts m and n of $f_{m,n;k}$ designate location of a kernel unit in the kth operation kernel. Figure 5 shows location of a kernel unit designated by (m,n).

Consequently, only if a desired parallel neighborhood operation is described by logical expressions in the form of Eq. (4) can operation kernels used for correlation be determined and the target operation be achieved according to the procedure shown in Fig. 2. All the procedures in OAL can be executed in parallel with optical techniques over an entire pixel data in

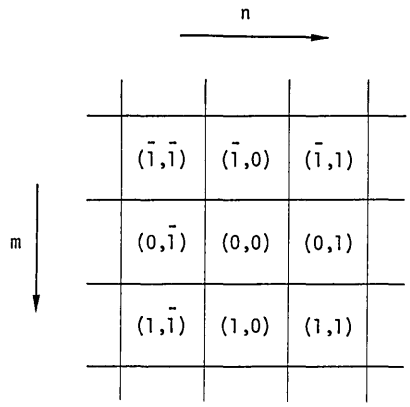


Fig. 5. Location map of kernel units referenced by (m,n).

objects to be processed, and hence N^2 neighborhood operations can be carried out simultaneously.

Attractive features of OAL are that (1) the processing principle of OAL is the same as that of array logic[9] in electronics except for parallelism[7] and (2) the OPALS based on OAL is an optical implementation system of cellular automata.[10] Hence, software resources in array logic and cellular logic in electronics, such as methods of planning and processing algorithms, can be fully utilized in OAL.

III. Programming in OAL

In this section we describe a generalized programming procedure of parallel processing with OAL and demonstrate the flexibility of OAL. To execute parallel processing with OAL, the following procedures are necessary:

(1) Modeling current problems as binary patterns.

(2) Formulating an algorithm with parallel neighborhood operations in the form of Eq. (4).

(3) Coding the parallel neighborhood operations with the programming language of OAL.

(4) Compiling the programming codes to generate operation kernels of OAL.

(5) Executing parallel processing according to the procedure of OAL.

Assuming that modeling problems and the formulating algorithm have already been done and the parallel neighborhood operation to be executed is expressed by Eq. (4), we consider how to code the operation and how to compile the coded program.

For convenience of programming in OAL, a symbolic notation, or in-a-broad-sense programming language, is used.[7] When an objective operation is expressed by Eq. (4), the $f_{m,n;k}$ terms can be uniquely determined. Thus the procedure for coding a product term (kth product term) is (1) to express the $f_{m,n;k}$ terms by the function symbols in Fig. 4, (2) to put them at locations corresponding to m and n referring to Fig. 5, and (3) to enclose by parenthesis each set of symbols concerned with the same k.

The function symbol at $(m = 0,n = 0)$ is underlined to mark the center of the neighborhood area. Applying the coding process to all product terms of the target

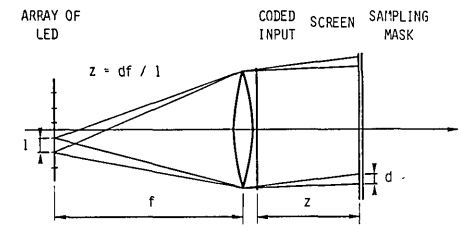


Fig. 6. Modified shadow-casting system for OAL.

operation and joining the parenthetical terms with + operators, we can code the given operation by OAL programming language. Note that our OAL symbolic notation does not necessarily restrict only in the manner of arrangement and size of neighborhood area. Using this notation, any parallel processing can be expressed.

Compiling the coded programs is to replace the function symbols in parenthetical terms with kernel unit patterns and to compose operation kernels. In this process, position adjustment is needed so that the underlined function symbol may be replaced with the kernel unit at $(m = 0,n = 0)$. All operation kernels for the desired operation can be obtained from all parenthetical terms in the coded program.

Using the operation kernels obtained by compilation, the target operation can be executed according to the procedure of OAL. Figure 6 shows an example of an optical setup for executing OAL, which is a modified shadow-casting system without magnification of projected images. Switching configurations of a light emitting diode (LED) array corresponds to an operation kernel for correlation. The result of a product term operation is obtained from the correlated image through a sampling mask in contact with a screen.

Since one product term operation is executed by a set of the optical system, many copies of the optical system should be prepared for simultaneously performing all product term operations in the current operation. When only one optical system in Fig. 6 is used, sequential control of the switching pattern of the LED array is required. In any case, several optical function devices must be used to execute parallel NAND operation in OAL.

IV. Image Data Processing with OAL

To demonstrate the usefulness of OAL, we attempted two examples of image data processing: edge detection and region extraction of connected area. Practical operational procedures in these examples can be applied to other processing.

A. Edge Detection

An object to be processed is assumed to be a binary image we call image A. An edge pixel to be detected is defined as a pixel satisfying the following conditions: (1) the value of itself, $a_{i,j}$, is 1, and (2) the value of any one of four connected neighbors, $a_{i-1,j}$, $a_{i+1,j}$, $a_{i,j-1}$, and $a_{i,j+1}$, is 0.

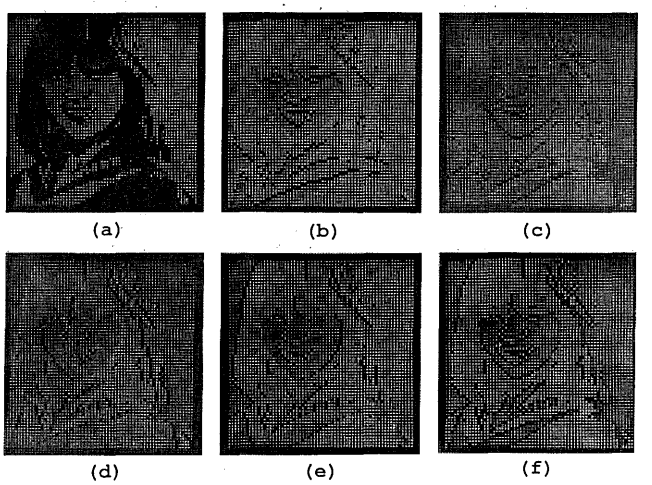


Fig. 7. Experimental results of edge detection with OAL: (a) object to be processed, (b)–(e) results of product term operation, (f) resultant image of edge detection.

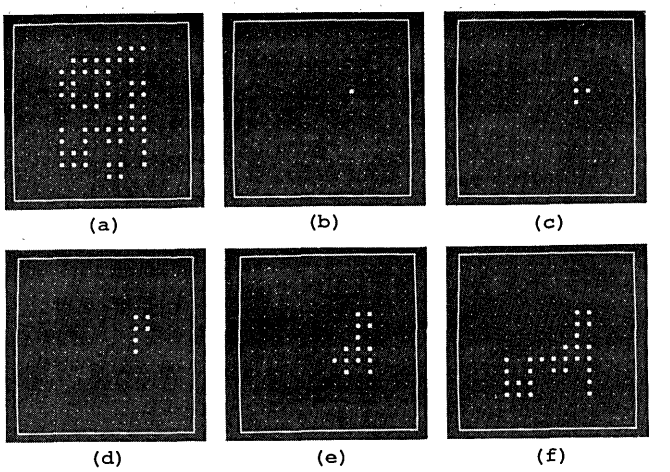


Fig. 8. Simulation results of extracting connected area: (a) object to be processed, (b) core pixel pointing to extracted area, (c)–(f) output of first, second, fifth, and thirteenth iteration, respectively, (f) final result of extraction of connected area including the sign pixel.

Then the neighborhood operation for detecting edge pixels is formulated as

$$c_{i,j} = a_{i,j}(\bar{a}_{i-1,j} + \bar{a}_{i+1,j} + \bar{a}_{i,j-1} + \bar{a}_{i,j+1}), \tag{5}$$

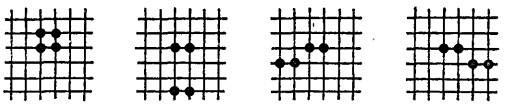
where $c_{i,j}$ is the detected pixel. Equation (5) is developed into

$$c_{i,j} = a_{i,j}\bar{a}_{i-1,j} + a_{i,j}\bar{a}_{i+1,j} + a_{i,j}\bar{a}_{i,j-1} + a_{i,j}\bar{a}_{i,j+1}. \tag{6}$$

Equation (6) is the same form as Eq. (4). Therefore, Eq. (6) is coded by the following symbolic notation:

$$\begin{bmatrix} \cdot\cdot & 0. & \cdot\cdot \\ \cdot\cdot & \underline{1.} & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & \cdot\cdot \end{bmatrix} + \begin{bmatrix} \cdot\cdot & \cdot\cdot & \cdot\cdot \\ \cdot\cdot & \underline{1.} & \cdot\cdot \\ \cdot\cdot & 0. & \cdot\cdot \end{bmatrix} + \begin{bmatrix} \cdot\cdot & \cdot\cdot & \cdot\cdot \\ 0. & \underline{1.} & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & \cdot\cdot \end{bmatrix} + \begin{bmatrix} \cdot\cdot & \cdot\cdot & \cdot\cdot \\ \cdot\cdot & \underline{1.} & 0. \\ \cdot\cdot & \cdot\cdot & \cdot\cdot \end{bmatrix}. \tag{7a}$$

Hence, (7a) can be compiled into a series of four operation kernels:



(7b)

Using these operation kernels for correlating operations in Fig. 2, we can execute edge detection in parallel.

Figure 7 shows an experimental result of edge detection using the optical system of Fig. 6. Figure 7(a) is the object to be processed which consists of 64 × 64 pixels. This object is encoded and recorded on a slide 32 mm square. Figures 7(b)–(e) are the results of product term operations with the above four operation kernels. Figure 7(f) is the resultant image obtained by NAND operation for images in Figs. 7(b)–(e). For the NAND operation a photographic technique using lithographic film is used.[7] In Fig. 7 the results are expressed by dark-true logic, so that dark signals indicate detected edge pixels.

B. Region Extraction of Connected Area

As shown in an earlier use of cellular logic,[10] iterative processing is useful for some kinds of image processing. We attempt to extract a region connected to a core pixel as an example.

An object to be processed in a binary image we call image A, and a core pixel, or an initial position of the extraction, is set in image B. They are depicted in Figs. 8(a) and (b). For the region extraction, the following augmentation is iteratively executed to image B. The output image at the kth processing step is used as input image B at the $k + 1$th step. After appropriate iteration, the connected area including the core pixel is extracted as the output.

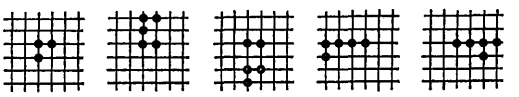
The augmentation is expressed by the following neighborhood operation:

$$c_{i,j} = a_{i,j}b_{i,j} + a_{i,j}a_{i-1,j}b_{i-1,j} + a_{i,j}a_{i+1,j}b_{i+1,j} + a_{i,j}a_{i,j-1}b_{i,j-1} + a_{i,j}a_{i,j+1}b_{i,j+1}. \tag{8}$$

This operation can be coded by symbolic notation, i.e.,

$$\begin{bmatrix} \cdot\cdot & \cdot\cdot & \cdot\cdot \\ \cdot\cdot & \underline{11} & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & \cdot\cdot \end{bmatrix} + \begin{bmatrix} \cdot\cdot & 11 & \cdot\cdot \\ \cdot\cdot & \underline{1.} & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & \cdot\cdot \end{bmatrix} + \begin{bmatrix} \cdot\cdot & \cdot\cdot & \cdot\cdot \\ \cdot\cdot & \underline{1.} & \cdot\cdot \\ \cdot\cdot & 11 & \cdot\cdot \end{bmatrix} + \begin{bmatrix} \cdot\cdot & \cdot\cdot & \cdot\cdot \\ 11 & \underline{1.} & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & \cdot\cdot \end{bmatrix} + \begin{bmatrix} \cdot\cdot & \cdot\cdot & \cdot\cdot \\ \cdot\cdot & \underline{1.} & 11 \\ \cdot\cdot & \cdot\cdot & \cdot\cdot \end{bmatrix} \tag{9a}$$

Compiling of Eq. (9a) produces the following five operation kernels:



(9b)

Using these five operation kernels, one step of augmentation is executed. The process is repeated until the output image becomes unchanged.

Figures 8(c)–(f) show results of a computer simulation for region extraction. After the first iteration four neighbors connected to the core pixel are obtained as shown in Fig. 8(c); Figs. 8(d) and (e) are the output images after the second and fifth iterations, respectively. The connected area grows step by step as the number of the iteration increases. Figure 8(f) shows the final result of region extraction of connected area including the core pixel after the thirteenth iteration. After that, the output of the process became unchangeable.

V. Conclusion

In this paper we have generalized the procedure for planning and executing arbitrary parallel processing with OAL as a systematic programming technique of optical parallel processing. The processing in OAL is formulated by logical expressions, so that practical processing can easily be implemented by OAL. The processing principle of OAL has a close relationship to those of array logic and cellular logic in electronics except for parallelism. Thus their software resources can be easily used in OAL.

Processing of OAL can be described by an original symbolic notation. Although this notation is incomplete as a programming language, it would serve to compose a powerful programming language capable of describing parallel processing linked to a conventional programming language.

Once target processing is described with symbolic notation, it can be executed by simple procedures for binary pattern data. We have applied the technique of OAL to two kinds of image data processing and verified the appropriateness of our discussion. Although the image processing is rudimentary, the important point is not the capability of such image data processing but the programmability of OAL. Use of OAL is not restricted to image data processing; by modeling problems to be solved as binary patterns, we can process those problems with OAL. Numerical data processing in OAL will be reported in part 2.[11]

The hardware to implement OAL has not been developed. However, procedures for executing OAL are simple, so that its implementation should not be difficult. We have been studying several kinds of hardware for OAL. Practical problems about the hardware were discussed in a previous paper.[12]

References

1. A. A. Sawchuk and T. C. Strand, "Digital Optical Computing," Proc. IEEE **72,** 758 (1984).
2. T. K. Gaylord, M. M. Mirsalehi, and C. C. Guest, "Optical Digital Truth Table Look-Up Processing," Opt. Eng. **24,** 48 (1985).
3. K-H. Brenner, A. Huang, and N. Steibl, "Digital Optical Computing with Symbolic Substitution," Appl. Opt. **25,** 3054 (1986).
4. Y. Fainman, C. C. Guest, and S. H. Lee, "Optical Digital Logic Operations by Two-Beam Coupling in Photorefractive Material," Appl. Opt. **25,** 1598 (1986).
5. A. W. Lohmann and J. Weigelt, "Spatial Filtering Logic Based on Polarization," Appl. Opt. **26,** 131 (1987).
6. J. Tanida and Y. Ichioka, "Optical Logic Array Processor Using Shadowgrams," J. Opt. Soc. Am. **73,** 800 (1983).
7. J. Tanida and Y. Ichioka, "Optical-Logic-Array Processor Using Shadowgrams. III. Parallel Neighborhood Operations and an Architecture of an Optical Digital-Computing System," J. Opt. Soc. Am. A **2,** 1245 (1985).
8. J. Tanida and Y. Ichioka, "OPALS: Optical Parallel Array Logic System," Appl. Opt. **25,** 1565 (1986).
9. H. Fleisher and L. I. Maissel, "An Introduction to Array Logic," IBM J. Res. Dev. **19,** 98 (1975).
10. K. Preston, Jr., and M. J. B. Duff, *Modern Cellular Automata Theory and Applications* (Plenum, New York, 1984).
11. J. Tanida, M. Fukui, and Y. Ichioka, "Programming of Optical Array Logic. 2: Numerical Data Processing Based on Pattern Logic," Appl. Opt. **27** (1988), same issue.
12. Y. Tanida and Y. Ichioka, "Modular Components for an Optical Array Logic System," Appl. Opt. **26,** 3954 (1987).

NASA *continued from page 2921*

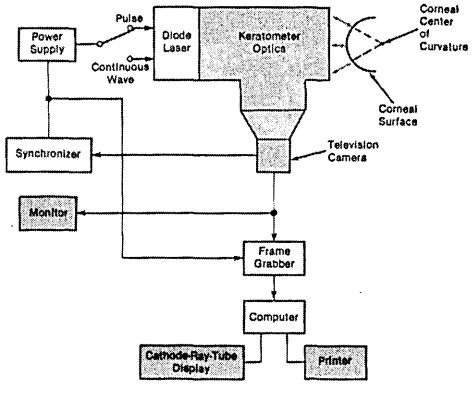


Fig. 5. In the data-analysis-and-display mode, the laser light is pulsed, and the moire pattern is converted to digital information in the frame grabber. The digital picture information is sent to the computer for processing and display.

imposed on the second grating, which is laid out on a fiber-optic plate that has a receiving-surface shape identical to that of the circular grating. Moire fringes are produced by the differences between the second grating and the reflected image of the first grating. Only these fringes, not the high-frequency grating structure, are transmitted by the plate to the moire fringe TV camera.

If the corneal surface is aspherical, the location and number of moire fringes differ from those produced by the spherical aberration of the optical system alone. The difference is measured as a change of transverse aberration resulting from asphericity of the corneal surface. The analysis is conducted along an azimuth, and a measurement of eight azimuths suffices to define the corneal shape.

The eye-fixation plane contains the ends of eight fiber-optic guides arranged uniformly around the axis. By focusing on each of these light spots as each is illuminated in turn, the patient cumulatively presents a large portion of the corneal surface for mapping.

The data-analysis-and-display system is shown in Fig. 5. The moire pattern is scanned at the standard TV scanning rate. This rate plus the pixel resolution necessary to define a full-field moire pattern require analog video signals to be converted to digital at a rate ~10–20 times as great as the maximum transfer speed possible using a typical desktop computer; consequently, a frame grabber is used between the TV camera and computer. The frame grabber

continued on page 2948