A Wavefront Array Processor for on the Fly Processing of Digital Video Streams

Georges QUÉNOT, Christophe COUTELLE, Jocelyn SEROT, Bertrand ZAVIDOVIQUE

Laboratoire Système de Perception
DGA/Établissement Technique Central de l'Armement,
16 bis Av Prieur de la Côte d'Or, 94114 ARCUEIL CEDEX (FRANCE)
Tel: (33 1) 42 31 92 80 Fax: (33 1) 42 31 99 64
Email: quenot@etca.fr, coutelle@etca.fr, jserot@etca.fr, zavido@etca.fr

Abstract

We present a wavefront array processor architecture developed at ETCA and dedicated to real-time processing of digital video streams. The core of the architecture is a mesh-connected three-dimensional network of 1024 custom processing elements. Each processing element can perform up to 50 millions 8- or 16-bit operations per second, working with a 25 Mhz clock frequency. Thus the theoretical peak power of the machine is 50 billions operations per second. Mapping of complex algorithms is facilitated by the routing capabilities of each processing element.

The machine is fully data-driven and is a "pure" data-flow one since there are no address flows. Algorithms and architecture are described using a data-flow graphs formalism. Image processing applications are decomposed into elementary operators that correspond to physical processors in a one to one fashion. Several algorithms can be simultaneously mapped and independently executed on the processor network.

Referring to the academic wavefront array paradigm, "exotic features" are exhibited. They are related to the wavefront propagation mode at run-time and to the heterogeneous nature of data-flows that are piped into or from the processor network. These features are shown to make the architecture well-suited for fast prototyping of low-level image processing automata.

1: Introduction

Real-time computer vision requires very high computing power. When a video sensor output is to be processed on the fly, computing powers in the range of billions operations per second are commonly encountered. For such calculations, Von Neumann style or other classical architectures fail to provide sufficient performance at a reasonable cost. Many image processing tasks appear to be based on the distinction between low-, intermediate- and high-level processing [7]. Examples in each of the three processing levels are:

- convolution, corners extraction or connected component labelling (low-level),
- right angles extraction in a line segment image or convex hull computation (intermediate level),
- model matching, asumption tree exploration and hypothesis validation (high-level).

The gap is so large between these processing levels (in term of inherent parallelism and data structures) that they cannot all be efficiently handled by a single computer architecture [8]. In particular, Von Neumann architectures are not suitable for low-level image processing because inherent parallelism of such processing cannot be exploited.

Thus many innovative architectures have been proposed as alternatives to Von Neumann ones, each being well-suited for a given computation level. For example, Multiple Instructions Multiple Data streams architectures are known to be well-adapted to high level processing where complex data structures are involved.

We focus on low-level processing and propose a wavefront array processor architecture able to process digital video flows on the fly (figure 1 and 2). Vision tasks are decomposed into elementary actions, each one being executed on a distinct physical processor. This functional parallelism is naturally expressed using the functional programming language concept [3][1] and the data-flow architecture concept [4] [2] which comprises wavefront array architectures. The semantic gap between algorithm description and architecture description is reduced by mean of the data-flow graph (DFG) formalism. DFGs can indeed be directly derived from functional expressions and are well-suited to algorithm mapping onto data-flow architectures [5]. Considering our "functional" approach, our machine will sometimes be referred to as the Functional Computer in the following pages.

Figure 1: Our wavefront array architecture

2: A wavefront array processor

The electrical signal delivered by a standard video camera is structured into frames and lines. When digitized, such a signal is thus naturally represented using data-flows. Data-flow stands here for a structured data set moving serially along a physical link. So in the general case, a data-flow can be viewed as a sequence of data symbols and control symbols.

In our particular case, three basic symbols were defined: "P" as pixel value (data symbol), "(" as left parenthesis (first control symbols) and ")" as right parenthesis (second control symbol). It is then possible to model digital video signals using these three symbols. A line is a pixel value list delimited with "(" and ")" parenthesis. A frame is a line list delimited with "(" and ")" parenthesis.

Pixel values and parenthesis are coded in a 9-bit field as follows:

b1xxxxxxxx	8-bit pixel value.
b01xxxxxx0	"(" control signal.
b01xxxxxx1	")" control signal.
b00xxxxxxx	no data to transfer.

A functional parallelism where all the operations corresponding to one iteration are executed in parallel for only one element at every time step (Multiple Instructions Single Data stream), is used instead of a data parallelism, where only one operation is performed for all the elements at every time step (Single Instruction Multiple Data stream, as this is the case for the Image Understanding Architecture [8] or the CM2 Connection Machine). There is a one-to-one correspondence between physical processors and operations in the algorithm instead of a one-to-one correspondence between physical processors and data elements.

But our architecture has rather to be considered as a wavefront array processor [5], essentially for two main reasons.

Firstly, the processor network is a very regular one (three-dimensional mesh-connected network) with only local interconnections where a high degree of pipelining is achieved. In fact, each implemented algorithm is executed through a gigantic pipeline typically involving more than one hundred simultaneous operations per time step.

Secondly, elementary processors are working in accordance with the data-flow principle [4][2] so that communications between two neighbor processors are asynchronous. Moreover the execution

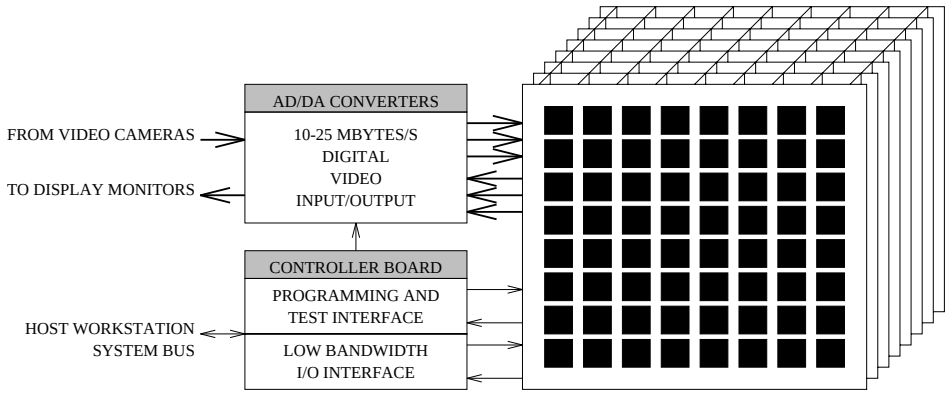

Figure 2: Wavefront array system architecture

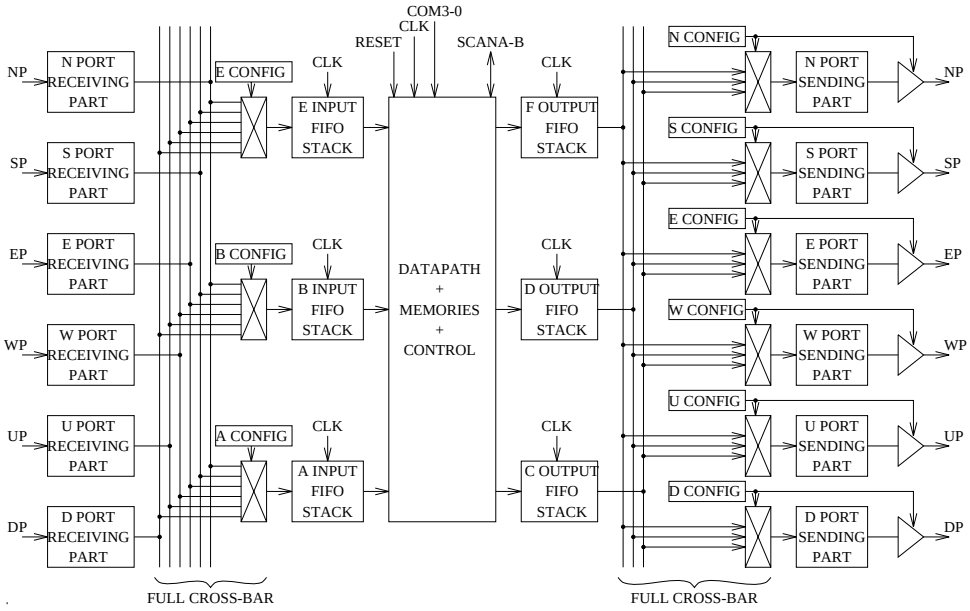

Figure 3: Internal structure of a processing element

model is a pure data-flow one since there is no address flow.

3: Hardware description

As appearing in figure 2, the core of our architecture is a mesh-connected three-dimensional network of processing elements. The network is organized in 16 interconnected 8×8 processor planes. It is physically supported by a set of 8 stacked circuit boards, each including 8×8 biprocessor chips (total amount of 1024 processors). A single processing element can perform up to 50 millions 8- or 16-bit operations per second. Thus the theoretical peak power of our machine is 50 billions 8- or 16-bit operations per second.

Due to the mesh-connected structure of the processor network, only local interconnections are involved. Communications links are 10-bit wide: 9 bits are reserved for data-flow passing, the last bit is a flag indicating whether the receiver is ready to accept data or not.

The processing element [6] has been designed using a 1μ CMOS technology and includes 80,000 transistors (total amount of 160,000 transistors per chip). The internal structure of one processor is shown in figure 3. Three main blocks are visible:

- The input/output ports, which are an interface between the processor heart and the outside world. Each port can be used either as an input port or as an output one.

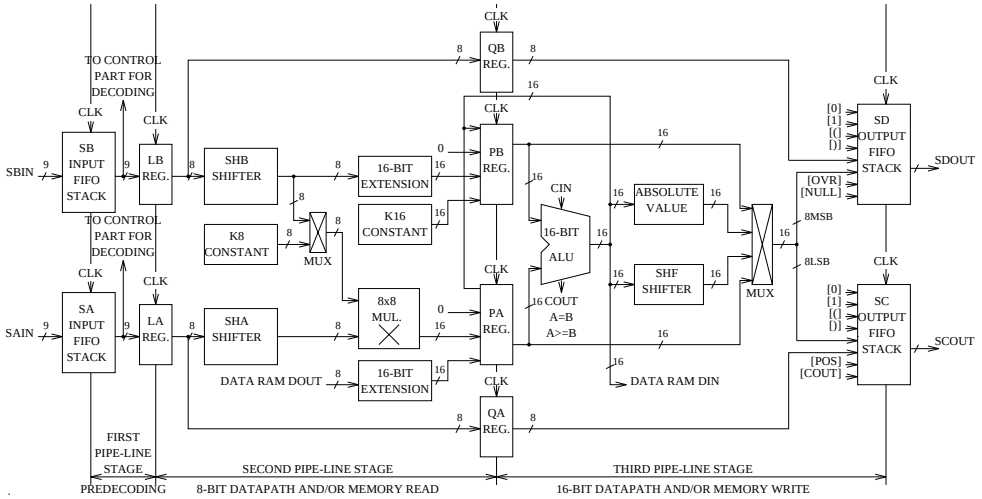

Figure 4: The processor datapath

- The input/output stacks are respectively entry and exit points in the datapath.
- Datapath and memories where all the computations are performed.

The processor datapath (figure 4) is organized as a three-stage pipeline. The first stage decodes input data and generates control signals for the following stages. The second stage is associated with a 8-bit datapath (8×8 multiplier and input shifters). 16-bit operations (ALU operations, absolute value, minimum or maximum and output shifter) are performed at the third stage. Moreover a 256-byte memory is provided and can be used either as a dual-port memory , as a FIFO or as histogram memory. The set of available operations has been constituted in order to satisfy most of the low-level image processing implementation requirements (presence of a hardwired histogrammer for example).

The computer also incorporates digital B/W and color video input/output subsystems, a global controller and low bandwidth interfaces coupled with a $SPARC^{tm}$ workstation.

4: Processing element programming

For each processing element, the model of execution is a data-flow model. A data-flow operator is viewed as a sequence of elementary actions. The execution of a given operator involves a programmable state machine (a state of which corresponds to an elementary action), a static programming register and a configuration register. The current state of the programmable state machine specifies the data needed on each input stack and the computation results to be placed on output stacks, for each clock cycle. According to a "dynamic firing rule", the processor controller validates operator execution if all needed data are available on input stacks and if output stacks are not full.

The programmable state machine is stored in a 64 32-bit word program memory. For a given state, the number of transitions is restricted to 1, 2, or 4. Transitions are enabled by a set of 0, 1 or 2 flags (for example indicating the received input data type). Depending on these flags, specific actions are validated. The first action type is related to stack control: data needed on stack A, shift out data from stack B, put ALU output most significant byte in stack D... The second action type is concerned with dynamic control of the datapath: ALU operand selection, counter control... So, operator execution is controlled by the parenthesis inserted in the incoming data-flow and is completely independent of what may occur in other processors. The operator is therefore fully "data-driven".

Execution stable commands are stored in a static programming register. This register is mainly concerned with ALU configuration (arithmetic or logic operation, input carry and output carry selection), on-board memories working mode (dual-port RAM, FIFO queue or histogram), multiplier and output multiplexers selection and operand interpretation (signed or unsigned).

Data-flows between stacks and input/output ports are routed through a full crossbar according to a configuration register. Input and output crossbar configurations are static and independent

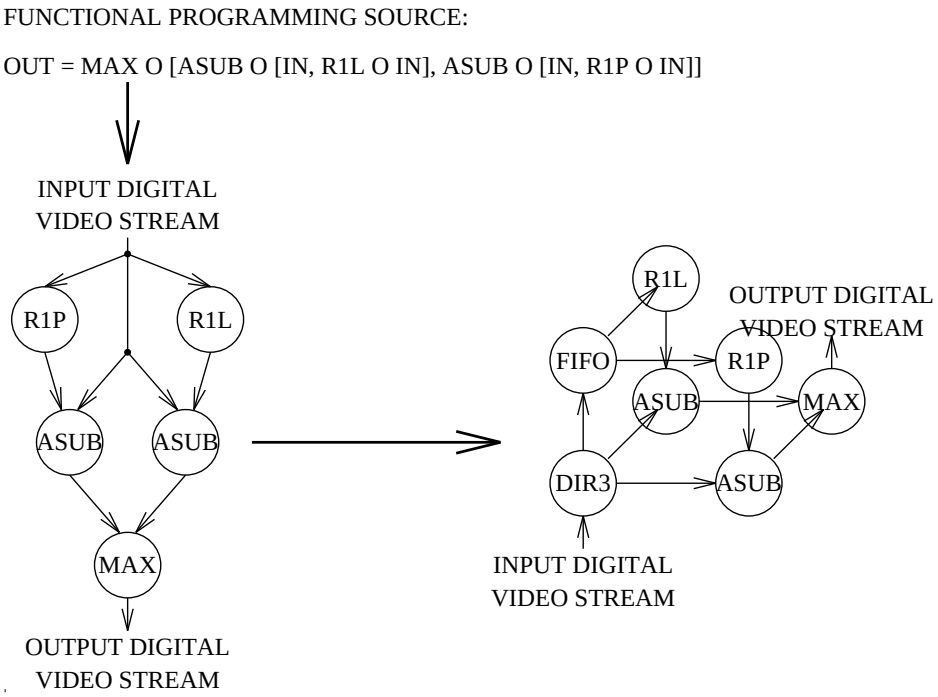

Figure 5: Elementary edge detector: functional source, data-flow graph and mapping

of the operator function. Routing crossbars are full crossbars so that any connection scheme is possible (obviously, as long as input ports and output ports are physically distinct).

5: System level programming

Programming our machine is done using the "data-flow graph" (DFG) formalism. An image processing algorithm is described using a DFG where nodes represent elementary operations involved in the algorithm. There is a one to one correspondence between an elementary operation and a physical processor. Graph arcs represent communication links between physical processors . The DFG formalism is naturally well-suited to data-flow structures processing and to low-level image processing where the two-dimensional image topology still prevails and where local computations are performed.

High-level representations of algorithms are obtained by using a functional programming language derived from the Backus's [3]. Figure 5 describes the three main steps involved in an elementary edge detector algorithm implementation: functional source writing, DFG translation and DFG physical mapping. The elementary operators used in the mapped DFG are described below:

name	function
DIR3	routing element
FIFO	FIFO queue
R1P	1 pixel delay
R1P	1 line delay
ASUB	difference absolute value
MAX	maximum

Besides it has been shown [5] that the DFG formalism is an adapted formalism to program wavefront array architectures.

Thanks to the data-flow programming model, there are no side effects, allowing a modular programming style. This is a very important feature when describing complex algorithms involving several hundred operations per pixel. It is thus possible to design "macro-functions" reusable in a number of different algorithms. Examples are convolution, windowing, large histogram computation, gradient direction computation, dynamic LUT... Data-flow principle meaning data-driven

processors, the same algorithm runs with different image size or format without modifying anything (except the depth of some FIFO stacks).

An elementary operator library has been designed so that image processing applications are built using predefined data-flow processor configurations (static and dynamic). Each configuration corresponds to an identified low-level image processing operator. The current elementary-operator library consists of about 150 operators. Another library including "macro-functions" is available.

Several data types are recognized inside a digital data-flow: k-vector (a list of k numerical values), pixel (a single numerical value), line (a list of numerical values delimited with two parantheses) and frame (a list of lines delimited with two parantheses).

In order to reduce the size of the operator library, polymorphic operators have been written, using the programmable state automata included in each processor. For example, the ADD operator is able to add:

1. a pixel P and a frame $F = ((P_{ij}))$, the result is the frame $G = ((G_{ij})) = ((P_{ij} + P))$
2. a pixel P and a line $L = (P_i)$, the result is the line $M = (M_i) = (P_i + P)$
3. two pixels...

To facilitate complex DFGs mapping, some elementary processors are used for routing purpose so that a physical node in the network can be considered as a processor with routing capabilities or as a routing element with processing capabilties. For instance, an adder operator could be configured as follows:

- Addition operand streams are received from east port and west port. The sum is send to the south port.
- A data-flow is routed from up port to down port.

With such routing capabilities, it is possible to reduce the number of used processing elements, allowing more complex graphs to be implemented.

Token balancing problems are solved using processing elements as fifos.

6: Functional computer specificity

Referring to academic wavefront array paradigm, our architecture can be viewed as an "exotic" wavefront array, essentially for two reasons.

The first one is that wavefronts propagating at execution are not simple analytical surfaces such as planes or spheres. Algorithms to be implemented are indeed described using data-flow graphs which can be very irregular. This reflects the ability of our architecture to support a large variety of low-level image processing algorithms. As an illustration, an example is given (figure 6) of a data-flow graph mapping for a colored object tracking algorithm. About 450 elementary processors are involved in the three-dimensional mapping - we show only one of the 16 8×8 processor plan, for sake of clarity. The proposed algorithm is composed of the following steps:

- A uniformly colored object is manually designated with the host worstation mouse. Object tracking is then entirely autonomous.
- Object average red, green, blue components are computed as soon as the object to be tracked has been designated.
- A distance is computed in the red, green, blue color space between object average components and pixel components of a tracking window.
- The above distance is thresholded. A vertical and an horizontal projection in the tracking window allow to compute a new position for the tracking cross-hair.

The second one is related to the nature of data-flows that are piped into or from the three-dimensional network. In fact two main different data-flow types are used in the machine: video flows involving high bandwidth interfaces (10-25 Mbytes/s) and asynchronous flows allowing a low bandwidth interface (there are also line rate, frame rate data-flows and others...). Therefore the pulsating rate of the wavefront is also irregular. Considering that our system is dedicated to video real-time processing, video signals are not controlled by the host computer (any frame generated by the camera is to be processed). So there is no buffering before piping data into the processor network. On the other side, asynchronous flows are totally under host machine control. This allows macro-function tests, intermediate results display (histograms, point of interest coordinates). More interesting is the possibility to interactively set control parameters for a given algorithm. For

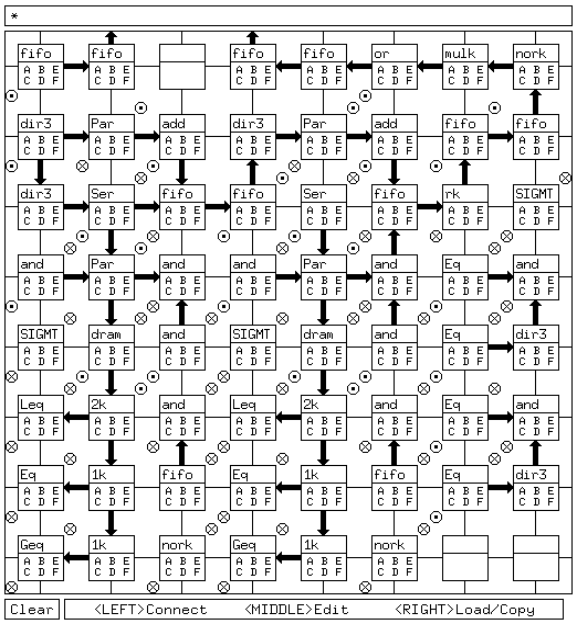

Figure 6: Part of a colored object tracking algorithm mapping (1 plane among 12)

instance, the above described tracking algorithm need manual designation of a target object. This is done using the workstation mouse. Mouse coordinates are piped into the network. The data-flow graph associated with the tracking algorithm then monitors a cross-hair display to visualize the chosen object and the tracking window is initially positionned according to received mouse coordinates.

Finally, it should be noted that several distinct algorithms can be simultaneously mapped and independently executed. The only constraint is volume: virtual functional processors are not implemented on this version of machine so all the data-flow graphs have to fit in the physical network.

Other image processing algorithms have been successfully implemented: Nagao like filter, motion detector, line extraction. Since the Functional Computer operates at fixed speed (from 16Mhz to 25Mhz), the achieved performance-level on a given algorithm implementation has rather to be expressed in terms of the number of processors involved in that task implementation: as soon as a graph fits in the physical network, 25 Hz video-frame sequences are processed in real-time. So main limitations to the Functional Computer performance is the finite size of the physical network and the lack of virtual functional processor implementation.

7: Future work

Industrial or military applications often require complex vision automata that are to be electronically integrated because of real-time constraints and/or volume or current supply limitations. This is typical of embedded systems. One of the bottlenecks when designing complex VLSI circuits is the functional validation which is time consuming and unsafe because of the huge number of configurations to be tested.

The Functional Computer will be used to remove this bottleneck. Our approach is a four step one:

- A low-level vision task is identified that is to be executed under video real-time constraints and that is supposed to be integrated because of additionnal constraints such as volume limitation.
- Several image processing algorithms are proposed to solve the problem.
- Each of these algorithms is implemented on the Functional Computer. It is then possible to select the most appropriate and to explore its dynamic behaviour by interactively modifying its associated control parameter set.

- The last step will be to derive an integrated automata from the implementation on the Functional Computer.

Our efforts are actually directed towards the derivation step. The first envisionned solution is to use Wafer Scaling technologies. The idea is to create physical interconnections (using additional metal layers) between processing elements directly on the wafer that has been used for their production.

8: Conclusion

We have described in this paper an original wavefront array processor well-suited for real-time low-level processing of digital video streams. The core of the actual machine is a three-dimensional network of 1024 VLSI processors allowing significant image processing algorithms to be implemented. Low-level vision tasks are easily programmed using the data-flow graph formalism to unify functional programming language and data-flow architecture descriptions.

Referring to the academic wavefront array paradigm, our architecture has been shown to be an "exotic" one. But its uncommon features are necessary to achieve a large variety of low-level image processing algorithms implementation and their interactive control. Thus the Functional Computer is a very efficient tool for fast prototyping of vision automata.

Finally the architecture simplicity and regularity (identical data-flow processors, mesh-connected network) will allow to derive integrated vision automata as WSI circuits.

References

[1] E. ALLART and B. ZAVIDOVIQUE. Functional image processing through implementation of regular data-flow graphs. In *Twenty First Annual Asilomar Conference on Signal, Systems and Computers*, November 1987. Pacific Grove, CA.

[2] K.P. ARVIND and D.E. CULLER. Dataflow architectures. Technical Report MIT/LCS/TM-294, MIT Laboratory for Computer Science, MIT, Cambridge, MA, 1986.

[3] J. BACKUS. Can programming be liberated from von neumann style ? a functional style and its algebra of programs. *Communications of the ACM*, 21(8), August 1978.

[4] J.B. DENNIS. Data flow supercomputers. *Computer*, 13, November 1980.

[5] S.Y. KUNG, S.C. LO, S.N. JEAN, and J.N. HWANG. Wavefront array processors-concept to implementation. *Computer*, 20(7), July 1987.

[6] G.M. QUÉNOT and B. ZAVIDOVIQUE. A data-flow processor for real-time low-level image processing. In *IEEE Custom Integrated Circuits Conference*, May 1991. San Diego, CA.

[7] C.C. WEEMS and al. Architectural requirements of image understanding with respect to parallel processing. *IEEE Expert*, 6(5), October 1991.

[8] C.C. WEEMS, S.P. LEVITAN, A.R. HANSON, E.M. RISEMAN, D.B. SHU, and J.G. NASH. The image understanding architecture. *International Journal of Computer Vision*, 2(1), January 1989.