Highly Parallel, Hierarchical, Recognition Cone Perceptual Structures

by

Leonard Uhr

Computer Sciences Technical Report #688

February 1987

Highly Parallel, Hierarchical, Recognition Cone Perceptual Structures

Leonard Uhr

Department of Computer Sciences
University of Wisconsin

To appear in: *Parallel Computer Vision*, L. Uhr (Ed.), Academic Press, 1987.

Highly Parallel, Hierarchical, Recognition Cone Perceptual Structures

Leonard Uhr

Department of Computer Sciences
University of Wisconsin
Madison Wisconsin 53706

1. Abstract

Massively parallel software/hardware structures are needed to perceive and understand the constantly changing real world. This chapter describes examples of such systems: recognition cone programs that are designed to be executed by appropriately augmented pyramid hardware. It shows how these structures can be used to recognize and describe static objects, to reason and make inferences, to model aspects of the living visual system, to describe scenes containing several objects, to handle binocular, N-ocular, and multi-modal images, and to recognize and track moving objects.

The following closely related topics are examined:

* The brain as an existence proof of achievable perceptual capabilities, as a source of potentially useful mechanisms, and as a system that is in need of and invites good theoretical models. The past few years have produced a wealth of suggestive and potentially helpful new anatomical, physiological, and behavioral findings as to how cats and primates perceive complex objects in 70-200 msec (which means within a few dozen or at most a few hundred serial steps).
* Appropriate multi-computer hardware, both general-purpose and algorithm-structured, along with structures of processes (expressed by algorithms and programs) that they can execute with speed and efficiency. Emphasis is placed on pyramids of arrays, truncated pyramids, systems that can simulate pyramids, internally augmented pyramids, structures containing multiple pyramids and arrays, and pyramids that are externally augmented, by being embedded in or/and linked to a small network of more powerful processors.
* Recognition cone perceptual systems that execute a shallow hierarchy of massively parallel, hence very fast, micro-modular processes to examine successively more global aspects of the scene, extracting, abstracting, and combining information as needed. When mapped into appropriate multi-computer architectures, these systems offer promise of perceiving complex objects as they move about in real time. They also (there is space only to mention this here) appear to be appropriate for learning, and for embedding into larger cognitive systems.

2. Introduction

To achieve the necessary power, efficiency, and speed, close compatibility is needed between the anatomy of the computer network and the physiology of the processes that this network applies to information flowing through it. This chapter focuses on the structures, both software programs and hardware multi-computer architectures, that I and my colleagues have been developing. The goal is to achieve systems capable of real-time perception and, ultimately, with the brain's ability to recognize, describe, and track complex objects within a few dozen or at most a few hundred serial steps.

The long-term goal of hardware/software structures for real-world perception is an extremely complex problem, and today we must simplify. One can concentrate entirely on some piece of the problem, for example, the detection of edges, textures, or patches; the linking of edges into contours, or of patches into regions; the computation of binocular disparity for depth, or of optical flow for motion. But it is at least as important to examine how to integrate the parts into a total perceptual system, to explore how well they work together. The parts can help one another (for example, contours and regions are two aspects of the same coin, the one bounding the other; depth must be computed using many diverse sources of information). Nor is it clear, until it is examined as part of the whole system, exactly what each particular part should do, or exactly how well it must work.

A perceptual system, whether realized with a computer program or a hardwired architecture of transistors, or neurons, is a structure of processes that can be represented as a data-flow graph. The image is input to the sub-graph of nodes (almost certainly this should be a 2-dimensional array) in that graph that represents the retina, processed by them, and flowed to the next processors. If the hardware had exactly the form of that graph, information could literally be input and flowed through it. To the extent that the hardware's topology is not isomorphic with the data-flow graph, the programmer must code additional processes to move and synchronize information as needed. Ideally, program and hardware should be designed together, as simply different aspects of the same system - roughly as the anatomy and the physiology evolved together as different aspects of a living system.

3. The Living Perceptual System

Humans and other higher animals are the only successful perceivers built do date, and they work beautifully. The enormously fast speed of human perception poses our problem. It also serves as an existence proof that the problem can be solved, and is a rich source of suggestive facts and possibilities.

3.1. The Great Speed of Human Perception, Despite Very Slow Basic Processors

We human beings can recognize complex objects (e.g., houses, trees, vehicles, furniture, tableware, people) in substantially less than a second. This is clear from behavioral experiments that examine recognition of complex objects presented for a millisecond or less, along with our ability to recognize rapidly flashed sequences of images (as in movies) and large fields of objects (as when reading). Even more striking are extracellular electrode studies in our close relatives, the primates. For example, Perrett et al., 1984, have found neurons in macacque monkeys that respond to complex objects, like faces, or a particular person's face, in from 70 to 200 milliseconds. This speed is achieved despite the fact that the brain's basic processing elements, the neurons that synapse and fire into one another, take at least a millisecond or two to fire (computing their basic thresholding functions) and send their outputs to the neurons into which they synapse.

The human brain is made up of 10^{10}-10^{11} neurons, each synapsing with hundreds or thousands of other neurons. A neuron fires (that is, sends a signal down its axon and into the synapses to other neurons) if enough neurons have fired into it during the recent past to satisfy the function that this neuron computes. When fired into, the neuron will start firing more frequently - for example, 100 or even 1000 times a second rather than its resting rate of 2 or 20 or so times a second. Thus information is encoded in terms of relative frequencies of firing. The exact function that a particular neuron computes is not known in precise detail. Indeed, it appears likely that this function changes from moment to moment as a result of a variety of different factors, including satiation, adaptation, feedback from other neurons, influences like alcohol, and variations in the neurotransmitters that actually serve to initiate firing across a synapse. But a very interesting and suggestive picture (to be described below) is emerging from a wealth of experimental evidence, of structures of each relatively simple and local functions that combine into the successively more global processes of perception.

A photograph of a scene that contains dozens of objects (for example, tables, chairs, people, animals) can be presented with a brief millisecond flash of light, and these objects will be perceived and recognized by humans within a few hundred milliseconds. A stream of such images, as presented on a movie or television screen, is also perceived without problems, even though neurons are millisecond devices and each image remains for only 20 or 30 milliseconds. The contrast with computer vision systems is striking. Today's programs take minutes or even hours to process a single image, although they handle only a small part of very simple examples of the total recognition problem - and even at that do not work very well. Yet the basic processor in today's computers is three to six or more orders of magnitude *faster* than a neuron. A small, cheap computer needs only a microsecond or two to compute one of the basic functions that have been hardwired into its central processing unit (e.g., 32-bit multiply or match). More expensive computers need only a few nanoseconds.

The only way to achieve adequately fast perception using basic processing elements that are so slow is to have very large numbers of processors all working together at the same time, in parallel. That is clearly the way the brain operates. There is a good deal of suggestive evidence as to how brains use millions of processors that are each far slower, smaller, and weaker than the processors in today's computers to accomplish perceptual tasks that are far beyond any that computer programs have achieved to date, at speeds far faster than any of these far weaker computer programs. This evidence (see Kuffler et al., 1984) is well worth examining - to give a (necessarily incomplete) blueprint for a computer program, and to focus on the brain's salient features, and on those of its mechanisms and techniques that can help in designing computer-based perceivers.

3.2. The Massively Parallel, Shallowly Hierarchical, Structure of the Brain

The brain is massively parallel; but crucial to its success is the fact that it is shallowly serial and hierarchical (otherwise it would be overwhelmed by the combinatorial explosion of possibilities). It applies very large numbers of each simple and local micro-modular neuronal processes. Each neuron, each micro-circuit (e.g., columns, hypercolumns, and other structures of neurons) processes its inputs, and outputs its results to a number of other neurons. This establishes a converging/diverging structure.

3.3. The Retina and the Lateral Geniculate

The eye's lens focuses images on the retina's 2-dimensional array of 6-10 million cones (that sense color) and roughly 100 million rods (that sense very slight changes in position and intensity). These fire into several layers of neurons, which do the following: Each of the millions of light-activated sensors sends excitatory signals to one or more neurons lying directly behind it, and inhibitory signals to neurons nearby but not directly behind - or the opposite. [The actual network is, as always in the brain, complex. The rods and cones fire bipolar neurons which in turn fire ganglion cells whose axons form the optic nerve that links, via the one layer of synapses in the lateral geniculate, to the primary visual cortex. In addition, rods and cones fire horizontal cells that link laterally to large numbers of bipolar cells, and bipolars fire amacrine cells that link laterally to, and in turn fire, large numbers of bipolar and ganglion cells.] This enhances differences and emphasizes gradients, computing the presence of a local spot, essentially as though using a Laplacian differencing operation. Thus retinal ganglion cells are sensitive to on center-off surround, or off center-on surround spots (Kuffler, 1953).

The simplest possible discrete Laplacian can be computed in a square array's 3-by-3 window, by assigning a high positive weight (8 is often used) to the center cell, and low negative weights (-1 is often used) to the 8 surrounding cells. The living retina computes a much more sensitive function, over a whole region of surrounding cells. It is also a less precise process. Although sensors are laid out in hexagonal arrays, especially in the central fovea, there are many local irregularities; each neuron does not synapse on every neuron in its surrounding field, and the weights of firings are not rigidly uniform.

The optic nerve - a cable of roughly 1,000,000 neurons - forms the rather large but typically unnoticed blind spot where it leaves the retina. Every human being also has a number of defective rods, cones, and larger regions of sensors. These become increasingly numerous with aging; but they are rarely noticed and, subjectively, have no effect on recognition or on the precision of vision. Here, as in many other aspects of vision, more global higher-level processes fill in, regularize, and in general make the image and its objects stable, complete, and constant.

3.4. The Cerebral Cortex

The cerebral cortex is a thin, crumpled, essentially 2-dimensional sheet containing a complex structure of six layers of densely linked neurons (the grey matter) under which is an approximately equal volume of long cortico-cortical axons that carry signals from one part of the cortex to another. The signals from the lateral geniculate flow into the primary visual cortex (also called the striate cortex or area V1) at the back of the brain, resulting in representations that continue to be topographically localized, much like the original image.

Sets of "columns" rise side-by-side through all six layers of this sheet (Hubel, 1982). Powell, 1981, summarizes counts of a column's neurons in many different species, including cat and several monkeys. These are strikingly similar. Almost all columns (and columns are pervasive throughout the cerebral cortex) have about 112 neurons +- 9, with the one exception of the striate cortex of primates, which have about 265 +-13. The columns may be the major functional units of the brain. It is they that appear to assess simple features (including oriented edges, colors, and textures), and (although here much less is known) successively more complex and more abstract features. Each column has rich connections to others (both laterally through the grey matter and via cortico-cortical links through the white matter), and also to different parts of the cortex and the rest of the brain. Information flows through and is processed in the layers of the columns, and from column to column - possibly once, probably cycling.

Roughly 2/3ds of a column's neurons are pyramidal cells whose processes rise vertically through all six layers of the cortex and whose axons link to other cortical or subcortical areas. Roughly 1/3d of the column's other neurons are inhibitory; many also appear to have direct links to other areas. The region surrounding each neuron is bushy and dense with its intertwined dendrites. In primates one type of neuron receiving inputs from the lateral geniculate in layer IV (where most of the extra primate striate neurons are found, in the form of small interneurons and pyramidal cells) is fired, much as is the typical lateral geniculate and retinal cell, by a spot. A second type of neuron with inputs from the lateral geniculate is a Hubel-Wiesel, 1962, "simple" cell, that is, a cell that fires most strongly to a moving edge of a particular orientation at a particular location. Other neurons in the same column are "complex" cells, firing to an oriented edge of a particular length, or anywhere in a much larger region.

Arrays of columns in V1 form "hypercolumns" that contain a progression of columns sensitive to differently angled moving edges, organized in side-by-side rows, one for each eye. Cortico-cortical links join a column that responds to a particular angled edge to other columns responding to that same edge, over distances of up to several millimeters (Ts'o et al., 1986). (The number of hypercolumns so linked, and the percent at different distances, are not yet known.)

The total process of visual perception was originally thought to involve a simple hierarchy of operations that arrive at successively more complex sets of information. This would seem logical and efficient, and there is some experimental substantiation. First, gradients are enhanced and spots extracted in the retina. Next, cells found in the primary visual cortex, and also in most other visual areas, are sensitive to simple oriented moving edges of several different types: located in a particular region, or anywhere in the visual field, with 2, 1, or 0 ends stopped. It was originally thought that the simple edge detectors are followed by more complex features (Hubel and Wiesel, 1962, 1977), but things are turning out to be far more complex (Stone et. al., 1979). What were originally taken to be angle-sensitive neurons now appear to be responding to 1-stopped edges. The actual links found between areas and the results of physiological experiments suggest a parallel-serial structure, rather than a simple strict hierarchy. For example, although it seems logical that the simple cells sensitive to a short moving edge with a particular orientation (e.g., vertical) at different small regions in the image should link and converge into the complex cells that fire to that same oriented edge no matter where it is located, networks that function this way have not yet been identified.

Krueger, 1985, using multi-electrode recordings from a 5-by-6 array of 30 electrodes to examine the firing pattern of a column, has found different but stable patterns for different stimuli. Many different stimuli appear to fire each neuron; but the pattern of firing over a whole set of neurons differs from stimulus to stimulus.

From the primary visual cortex information flows through a parallel-serial structure of at least 20 other visual areas, and also nonvisual areas that are involved in perception. Some of these are predominantly sensitive to and handle special types of features, for example, color, motion, or shape. Mishkin et al., 1983, conclude from a large amount of anatomical and physiological evidence that two major pathways through the subsequent visual areas diverge from the primary visual cortex. One, that processes color and shape, leading to the recognition of objects, moves into inferior temporal areas. The second, which moves into inferior parietal areas, is involved with the spatial interrelations among objects, and with how objects move about and change these spatial relations. Links that move into the temporal lobe and the frontal lobe make possible the association of visual information about objects with information from the other sense modalities, and also with emotions and motor behavior. Thus there seem to be at least two major parallel (and shallowly serial) types of processing - those involved with the recognition of objects using shape and color, and those involved with relations among entities, and with change (motion). (Obviously, since the system locates the objects it recognizes, these two kinds of information must later be combined, just as information from eyes, ears, and skin must at some point be combined.)

Extensive physiological and anatomical studies (Van Essen, 1985) show how the 20 visual areas found to date (in macaque monkey) are wired together by at least 40 major and 40 minor pathways. The overall structure appears to be a parallel-serial hierarchy with each area richly linked to other areas (two or three other areas on the average, and up to eight at most). [Note that this is far from the complete connectivity that is sometimes suggested to exist, which would link each of the 20 areas to all 19 others. Nor is it a simple hierarchical tree of the sort found in many computer vision systems.] When two areas are linked they are linked in both directions. There are also links from these visual areas into the association areas that combine vision, hearing, and touch. There is some evidence that, moving up through these areas, more and more abstract features are responded to by individual neurons. At the same time, one can conjecture that successive neurons are responsive to more and more complex features found in larger and larger regions of the retinal field. There is little firm evidence for this to date. But it appears that very few attempts have been made to find neurons sensitive to higher-level features - with the major exception of studies of the temporal lobe and amygdala.

High-level neurons of striking importance have been found, first by Gross and his associates (Gross et al., 1972, 1984; Bruce et al., 1981), and more recently in a number of other laboratories (e.g., Perrett et al., 1982, 1984; Rolls, 1984, 1985), to fire in a very specific manner to extremely complex stimuli. These neurons appear to exhibit the culminating step of the most complex structure of processes found to date. For example, a single neuron fires when the stimulus is a hand, no matter where it is presented in the visual field, but not to a face, or to a variety of other stimuli. Other neurons respond to a face, but not to a hand or to a picture that contains all the parts of a face, but jumbled around. One neuron responds to faces, another to a particular familiar face, another to a second face; one to faces in full view, another to profiles. Thus Bruce et al. found that 59 neurons of 199 examined responded most strongly to particular stimuli. Seven of these responded strongly to human and monkey faces, but not to simple stimuli like bars, spots, edges, or to complex stimuli like hands or brushes. Each of these cells would appear to be at a very high level in a complex network of information-transforming neurons, probably part of a larger structure that fires to that object.

There is also some evidence that the more complex the object the longer it takes to recognize it. For example, Bruce et al. found that neurons sensitive to simple stimuli, like bars, take roughly 45 milliseconds, but those sensitive to complex stimuli like faces take roughly 150 milliseconds.

3.5. A Summarizing Look at the Overall Picture, and a Few Comments

What are the brain's overall design principles? Its basic mechanism is the neuron. In roughly 1.5 milliseconds each neuron computes what appears to be a threshold function over the information transmitted to it by the large number of other neurons that synapse onto and therefore link to it. Reaction time experiments with human beings show that recognition of complex objects is completed within roughly 400 to 800 milliseconds.

Recordings from neurons in monkeys' temporal cortex that are sensitive to a face, or even a particular face, show that they respond in only 70 to 200 milliseconds. Therefore the serial depth of processing is extremely shallow, possibly only a few dozen, or a few hundred at most.

The cerebral cortex appears to be the locus of the higher mental processes, including perception. Roughly half of the cortex's many billions of neurons are involved with perception, predominantly with vision. Billions of neurons, each typically with thousands of synapse links, fire volleys of information to one another, apparently serving as discrete, but to some extent analog, probabilistic thresholding elements. The sheet-like cerebrum is organized into six layers, each containing large numbers of simultaneously firing neurons. Smaller structures of about 110 or 260 neurons are organized into columns that rise through all six layers. These are often arranged in hypercolumns that contain families of feature detectors. There are extremely large numbers of connections, including inhibitory connections, within local regions (columns and hypercolumns) and to more distant regions laterally within a layer. The overall structure is rich in both converging and diverging pathways, as well as cycles of links that appear to serve for rich feedback loops.

The visual system's structure is relatively clear: The sensory interface with the outside environment is, with its millions of input devices, massively parallel. Relatively (but not always) local near-neighbor operations successively transform sensed information, at the same time converging and diverging it, as it flows through the several layers of neurons in the retina and in the primary and 20 or so secondary visual areas. In addition the brain is packed with links to more distant areas, so that the potential for rich contextual interactions is great - although little is known about the actual details.

Anatomically, there is a serial depth of 2 or 3 transformations in the retina, followed by 1 transformation in the lateral geniculate and probably 3, or possibly 4 to 6, in the primary visual cortex. The perceptual process continues with transformations effected as information flows through a hierarchy of at least 20 visual areas. Since this is a parallel-serial hierarchy whose serial depth is around 4 to 6. Therefore, if 3 to 6 transformations are effected in each layer, the total depth of transformations would appear to be at least 18 to 46. In addition, the richness of interconnections and feedback loops from layer to layer within each area suggests it is likely that information continually cycles through an area. Thus processing proceeds in a massively parallel, and also a parallel distributed, but to some extent serial, manner moving through these perceptual areas. Within each area processes are massively parallel and also to a small but essential extent serial. Rich interconnectivity, with each neuron, column, or hypercolumn linked to large numbers of others, effectively gives both convergence and divergence of information.

Sometimes the visual system appears to examine different components of the total process in separate parts of the brain. For example, V4 is predominantly (but not entirely) involved with color, while MT appears to handle motion (Shipp and Zeki, 1985). At some stage this information must be combined, as is information from the two eyes, the two hemispheres, and the different sense modalities. The brain is especially adept at this crucial process of matching and identifying what pieces of information should be combined, and then merging them exactly. This also is an adaptive process, since the brain learns to make fine adjustments when the anatomy (or an experimental disruption) introduces less-than-perfect wired-in matchings.

Moving up through the visual system, it appears that more and more abstract aspects of the scene are recognized. This culminates in neurons that fire selectively to complex objects like faces and hands, and to complex multimodality stimuli, like the sound and the sight of a hammer striking a windowpane. Different neurons fire to different faces, or to face in general, or to face no matter how rotated, or only to full face, or only to profile. There appear to be enough of these neurons to give fault-tolerant redundancy. It is important to emphasize that each of these neurons is not a single "grandmother cell" recognizer of the complex object, but simply a member of a whole complex structure of neurons that have accumulated, transformed, and assessed the information in the image.

Although the brain is extremely parallel, it is crucial that it is not entirely parallel. The serial depth of the hardware and of the flow of information through it is absolutely essential, even if the serial flow is only through five or six (six-layered) areas, totaling a few hundred or even a few dozen synaptic firings that transform information.

4. Parallel-Serial Hardware Architectures for Perception Systems

The topology of a multi-computer network is conveniently represented as a graph whose nodes are computers and whose links are communications paths between them (Figure 1). A conventional single-CPU serial Von Neumann computer is the simplest example, since it is a 1-node graph. It is also the slowest; for (by definition) it does everything serially, only one thing at a time. Even the fastest possible serial computer will be far too slow to perceive complex real-world objects in real time. Nor could it even begin to process an image in the several dozen to several hundred serial steps taken by the living visual system. Perception is obviously a massively parallel process. The input sensor, whether the brain's retina with millions of rods and cones or a computer's TV camera input with hundreds of thousands or millions of individual pixel spots, is massively parallel. Only massively parallel networks are fast enough to process the large resulting image arrays, and the successive resulting structures of information. Parallel-hierarchical pyramid structures appear to be among the most attractive candidates.

A pyramid program, or a multi-computer designed in the form of a pyramid, is far simpler than the brain's perceptual system. But it keeps, in abstract and simplified form, many of the brain's essential features - in particular, the large 2-dimensional array (the base of the pyramid) into which the image is input, and the layered, converging structure of locally connected micro-modular elements. What exactly is a pyramid? Several hardware pyramids are now being built, and there is a much larger variety of possible pyramids and internally or externally augmented pyramids that might be built. Before describing them I will briefly examine a progression of simpler architectures that lead to pyramids, and suggest why they are useful.

4.1. Successively More Parallel Architectures for Computers

Consider programs as flowchart-like Petri net or data-flow graphs. Conventional serial computers are 1-node graphs into which any program graph can be decomposed. Or, to put it another way, the program graph can be flowed through the 1-node graph, but one node at a time. Many program graphs contain regular sub-structures that can be embedded in parallel hardware. Arrays are among the most common, especially for number-crunching matrix operations, and for image processing and perception. Array operations can often (but by no means always) be executed in parallel. For example, gradients, edges, textures, and many other features can be extracted everywhere in parallel using mask/window operations that look for local structures of information.

4.1.1. Pipelines of Processors and 1-Dimensional Arrays

Possibly the simplest way to speed things up is to pipeline the set of information in the array through a 1-dimensional line of processors. Each processor repeatedly executes the same operation, but on different data. This is the technique used by the vector processors (typically, 10 or so very powerful 64-bit floating point processors) in today's most powerful super-computers, like the Crays and Cybers (Riganati and Schneck, 1984). These data might be 1-, 2-, or N-dimensional; the only requirement is that there be a large number of sets of data all of which are to be processed in the same way, and none of which depend upon other sets of data.

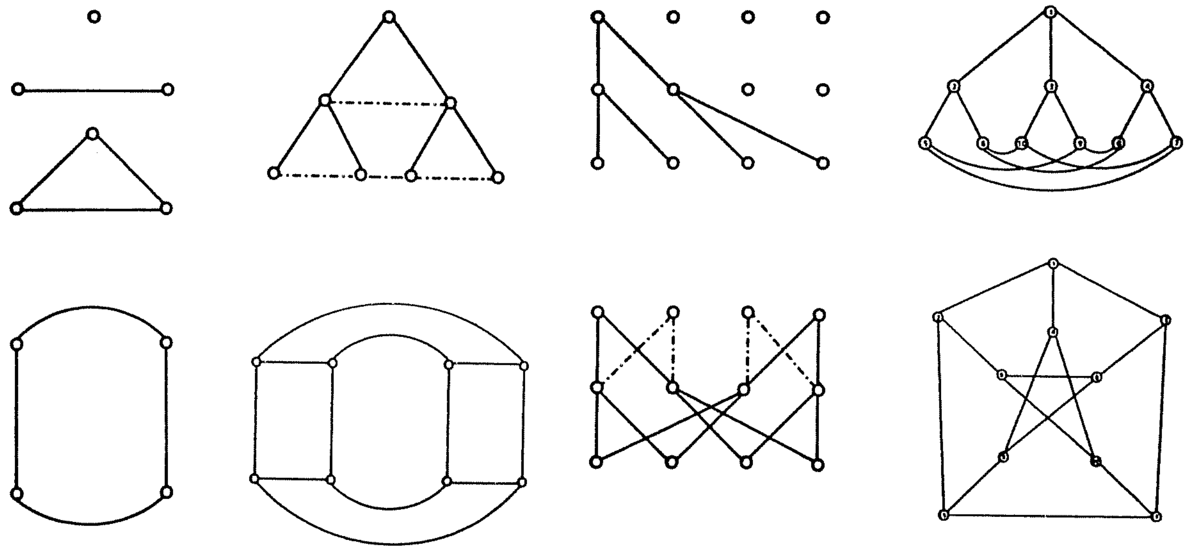

Figure 1. Basic Graphs: Serial computer; pipe (bus); polygon (ring); 2-D array (2-cube); tree (side of pyramid); 3-cube; NlogN net (1 tree, N trees); Petersen graph (2 drawings). All except the first and last are indefinitely expandable.

A number of image processing systems, for example the PIPE (Kent et al., 1985) and the Cytocomputers (Lougheed and McCubbrey, 1985), are of this sort. They typically flow a 256 by 256, 512 by 512, or 1,024 by 1,024 array of information through a short pipeline of from 1 to 10 processors. Each processor has built-in hardware to iterate over the entire array at TV scan rates, that is, in 30 milliseconds or less. This means that the TV image can be processed and transformed into some more abstract representation, then this abstraction can be processed, and so on, in any way the programmer desires. Often the processors are augmented with additional hardware that further speeds up important image processing operations, for example computing convolutions or logical mask operations over a 3 by 3 or even larger window in a single parallel operation.

Alternately, a 1-dimensional array of processors might all execute a sequence of processes on a 1-dimensional set of data. An N-dimensional set of data could similarly be processed, but one line (e.g., one row) at a time. The Pixie-5000 (Wilson, 1985), a 1-by-1,024 array of 1-bit computers designed to process images, is a good example of this kind of system.

4.1.2. 2-Dimensional Arrays

When the program must process 2-dimensional images (as is the case for visual perception), a 2-dimensional array of computers can substantially speed up performance. Now every set of data can be processed by a different processor - so long as there are enough. [If there are fewer processors than cells in the array of information (e.g., an N-by-N array of processors and an N-by-N image array), each can serially process an n/N-by-N/N sub-array stored in its memory, or the whole computer array can process each of the n/N-by-N/N N-by-N sub-arrays in turn.]

The largest 2-dimensional arrays built to date have up to 65,536 computers (each a processor with its own memory). In the 128-by-128 MPP (Batcher, 1980), 64-by-64 DAP (Reddaway, 1978), and 256-by-256 connection machine (Hillis, 1985), each processor fetches information from its own or any of its 4 directly-connected neighbor computers' memories, and executes whatever sequence of instructions the programmer has coded. Each processor in the 96-by-96 CLIP4 (Duff, 1978) fetches information from, and processes in parallel any or all of, its 8 directly-connected neighbors.

All the systems built to date are driven by a single controller, so that they all execute the same instruction but on a different set of data, in SIMD (single instruction multiple data-stream) mode. But any number of controllers could be used if more were found desirable. And more than one processor might be placed at each location. For example, 16 processors might be programmed to apply 16 different edge detectors, all in parallel. Additional links could be added to more distant processors, for example at powers of 2 distances, or using a shuffle network. The issue is not whether such augmentations are possible, but whether they are cost-effective.

4.1.3. Pipelines, 1-Dimensional Arrays, and 2-Dimensional Arrays Compared

A P-processor pipeline needs (NxN/P)+P instructions to execute one process on NxN pieces of data. A serial computer needs kNxN instructions (k is a small constant, the instructions needed to index and test for borders). An N-processor 1-dimensional array needs N instructions (indexing and border testing are handled by hardware). An N-by-N array needs only one instruction. Therefore a pipeline of 10 processors can execute 10 20 nanosecond instructions over a 1,024-by-1,024 array in 20 milliseconds. A 1-by-1,024 array needs only 10 instructions for each row of data, and 1,024 times that for the 1,024 rows - or 100 microseconds. A 1,024-by-1,024 2-dimensional array needs only 10 instructions - or 200 nanoseconds in toto.

Image processing often entails long sequences of local processes. When each object is small everything goes smoothly and with great speed. But complex objects are large, their very complexity dictating the size of the image array needed to resolve them. A 2-dimensional array is inefficient and slow when it must compute functions over several pieces of information that are distant from one another. Each computer can be linked to only a few other computers. This almost dictates that each is linked, as in all of today's arrays, to its 4, 6, or 8 nearest neighbors. More distant information must therefore be shifted one step at a time through these lateral links - which means that dozens or hundreds of shifts will be needed as objects grow bigger.

4.2. Asynchronous Networks of Completely Independent Computers

An important alternative is a much smaller network of independent and much more powerful computers, each with its own controller, working asynchronously in MIMD (multiple instruction multiple data-stream) mode. This kind of network can be built in a great variety of topologies - potentially, any possible graph (see Hwang and Briggs, 1984; Uhr, 1984). Many people have suggested the ideal would link all computers directly to one another, so that all could access one another's information, as though stored in a single common memory, and communicate with no overheads or delays. Unfortunately, no more than a very small number of computers can be linked directly using today's VLSI technologies, and processors would still have major problems determining where information is coming from and opening and closing communication channels. But this kind of system can be simulated by linking computers via a common bus (a high-bandwidth link), ring (a circular bus) or crossbar (an N-by-N array of switches). Buses and crossbars grow too expensive and/or create unacceptable bottlenecks when used for more than 32 or 64, or possibly a few more, processors. A shuffle-exchange based reconfiguring network (an NlogN bank of switches that can handle many, but by no means all, patterns of simultaneous messages between processors) can be used to link a few hundred, or possibly even a few thousand (Siegel, 1981, 1984). Among the most interesting systems of this sort is Siegel's PASM, a projected network of 1,024 computers, each with its own controller, plus 64 global controllers to which sets of computers can be assigned and reassigned, thus making it possible for the programmer to establish different levels of global control as needed.

For very large real-time problems - and perception is a prime example - it seems likely that many thousands of processors must work in parallel. Only point-to-point topologies, where each computer is linked to far fewer than all the computers in the network, are capable of achieving such large numbers. Today by far the two most popular such topologies are N-dimensional binary cubes and trees. However a large variety of other quite possibly more powerful topologies are known, and waiting to be explored (see Bermond et al., 1983; Uhr, 1984). For example, many researchers have suggested that computers should be packed close together, to minimize distances information must be sent. But it turns out that there are several known topologies (e.g., Moore graphs, De Bruijn graphs, hypertrees, good compounds of good basic graphs) that pack into a graph of a given diameter and degree orders of magnitude more processors than do N-cubes and trees. Many of these topologies have been discovered during the past few years, and there is good reason to expect continuing new discoveries.

These asynchronous systems are designed to have far fewer computers, each much more powerful, than do the massively parallel arrays and pyramids. Typically, 32-bit processors, each with its own controller, are used, rather than the 1-bit processors, all subject to a single controller, found in the large arrays. Because each computer has its own controller and works completely independently, major new problems arise in terms of coordination, contention, and message-passing. These problems are especially acute for computer vision, where massive amounts of data must be processed and information must be sent and received with virtually no delays. The "message-passing protocols" used by today's operating systems for such networks take many thousands of instructions to send information, even if only one or a few bits, from one computer to another. Additional hardware can be used to handle much of this, but at best it appears that message-passing takes hundreds of times longer than does processing.

This is in striking contrast to synchronized SIMD systems, where message-passing between directly linked neighbors takes no longer than fetching, and between computers that are N nodes apart takes only N times as long. It seems likely that substantial improvements can be made; but today MIMD systems, although they are conceptually much more flexible than rigid, synchronized SIMD systems, appear to be much slower and less cost-effective. However SIMD systems are efficient only to the extent that most of their processors do fruitful work most of the time. Possibly the most attractive alternative (to which we shall return) is to combine more or less specialized SIMD structures with an appropriate asynchronous MIMD network.

4.3. Pyramid Multi-Computers, Augmented Pyramids, and Specialized Networks

A (3-dimensional) pyramid in its most basic form is a stack of successively smaller (2-dimensional) arrays of computers, all linked via a tree. [Call the largest array, at the pyramid's base, the "retina input," and the smallest array the "apex output."] For example, the image might be transduced into the memory of a 1,024-by-1,024 array. This might link to a 512-by-512 "parent" array, with each parent linked to a 2-by-2 array of 4 "children" in the larger array. This converging structure might then continue, to 256-by-256, 128-by-128, 64-by-64, 32-by-32, 16-by-16, 8-by-8, 4-by-4, 2-by-2, and 1-by-1 arrays. Thus a 1,024-by-1,024-based pyramid with 2-by-2 convergence contains 11 arrays. Within each array each computer is linked to its 4 square, 6 hexagonal, or 8 square-plus-hexagonal "sibling" neighbors, exactly as in the 2-dimensional arrays. Schaefer, 1985, has built a 16-by-16-based pyramid of this sort, and Tanimoto, 1981, and Cantoni et al., 1985, are building larger systems.

A pyramid can be used in a great variety of ways (see, e.g, Burt, 1984, Dyer, 1982, Rosenfeld, 1983, Stout, 1983, Tanimoto, 1983, Uhr, 1972), many of which follow from its basic array+tree topology. Essentially, the array links give very efficient implementations of parallel window/mask local operations, while the logarithmic tree-based links reduce the order N distances between nodes in each of the pyramid's arrays to order logN. This is crucial since, for example, it reduces the thousand or so shifts that might be needed in a 1,024 by 1,024 array to at most 20. More important from the point of view of the recognition cone data-flow approach to be described below, as processes extract and abstract information the compressed results can be continually converged and flowed up through the pyramid. This means that operations can be made local, hence fast, by building hierarchies of successively more global processes, embedding those that examine several distant pieces of information high enough in the pyramid so this information will have converged into compact neighborhoods.

4.3.1. Simple Variations on the Basic Simple Pyramid Structure

There are a variety of potentially attractive variations and augmentations to pyramids (Ahuja and Swami, 1984; Tanimoto, 1985; Uhr, 1983, 1985a, 1985b, 1986, 1987). First, there are a large number of simple variations, as indicated by the following examples.

* The simplest basic pyramid will have one single controller for all computers, so that it broadcasts the same instruction to every array and works in strict SIMD mode.

* Probably preferable is a system with one controller for each array, or, simpler and better coordinated, one controller that can broadcast a different instruction to each array.

* Rather than converge 2 by 2, the system can converge N by N, or N by M. Or it can alternate in its convergence, N by M then M by N, or in a more complex alternation. Merigot et al., 1985, are building a pyramid that alternately converges 1 by 2 then 2 by 1.

* Rather than converge logarithmically, the pyramid might converge linearly, using a 3-dimensional array that has been carved out to the shape of a pyramid. It would converge too slowly and be a far more expensive system. For example, whereas a logarithmic pyramid with a 1,024-by-1,024 base has 11 layers, a linear pyramid would have 513. But logarithmic convergence may well be too fast.

* One way to slow down convergence is to sandwich slices of a linear pyramid as desired between the layers of a logarithmic pyramid. Another is to use linear conver-

gence to give a detailed zoom where needed, but to use logarithmic convergence to give the much faster and grosser combining of information that is usually sufficient.

* Different amounts of convergence can be put at different places. For example, the system might converge 3 by 3 for one or more arrays, then 2 by 2, then not at all, then diverge and expand for several arrays, then converge. This gives a 2-dimensional pipeline that, if it were built in a completely algorithm/process-structured manner would contract, expand, and remain steady in diameter as dictated by the reductions and expansions of the information it was processing.

* Layers can be linked with overlap, by having each child link to more than one parent. For example, a 3-by-3 converging pyramid might have each computer in a corner of the 3 by 3 link to 4 parents and the other computers on a border link to 2 parents. Uniform overlap can be got by having each child in a 2-by-2 converging pyramid link to 4 parents and each parent link to a 4-by-4 array of children, so that each 4-by-4 sub-array shares the 4-by-2 left sub-array of children with the parent to the left, the 2-by-4 bottom sub-array of children to the parent below, and so on. More generally, parents spaced m apart might each link to an N-by-N sub-array.

* Rather than converge to a 1-by-1 apex of 1 cell, the system can be built as a "truncated pyramid" that ends with an N-by-N array where N is greater than 1.

4.3.2. Internally Augmented Pyramids

All of these simple pyramids use the same computer, with the same processor and memory, everywhere. They also use simple SIMD controls. They differ chiefly in the way that processors are linked together. Several interesting variants can be achieved by augmenting the pyramid internally with additional types of more global links, more or more powerful processors, more memory, or/and more controllers:

* Stout, 1986, shows how NlogN networks or other better-than-array links within successively larger sub-arrays of nodes in layers above the base handle perimeter-bound problems like component labelling in logarithmic time.

* Each computer, or local sub-array of computers, can be linked to other computers at a distance, either via a switching network or directly. For example, Hillis' (1985) Connection Machine links local 8-by-8 subarrays of a 2-dimensional array via an N-cube of switches. A pyramid could be linked similarly or via an NlogN network, except that sub-arrays might come from different arrays, and even be of different sizes.

* Leiserson, 1985, "fat-trees" (with higher bandwidth communication links moving upward) can be generalized to pyramids, and to increasing the power and/or number of whatever resource (processor, memory, control, wire, switch) wherever needed.

* Rather than one processor for each computer, the system might have 2, 3, or N processors, or a reconfigurable B-bit processor that can be partitioned into several processors each handling smaller fields (Uhr et al., 1981; Sandon, 1985; Snyder, 1985). For example, each computer might be reconfigurable into 64 1-bit processors, 8 8-bit processors, 4 16-bit processors, 2 32-bit processors or 1 64-bit processor. These processors might be designed to efficiently fetch and process 2-dimensional arrays, e.g. 8x8, 4x4, 4x2, 2x4.

* Rather than have the same amount of memory for each processor, the system can have

successively more memory (or less memory) moving up from the base. Or it can have less memory moving toward the periphery within each array.

* Processors can be made more powerful, moving up the pyramid or/and moving toward the center of the array. This might be combined with more ability to reconfigure, and more memory. The University of Massachusetts CAAPP (Weems et al., 1984) is an interesting example, with its large SIMD array of 1-bit processors linked to a smaller array of 16-bit processors linked to a still smaller array of 32-bit processors.
* Different kinds of processors can be placed in different regions of the pyramid.
* Rather than have one controller for the entire pyramid, or for an entire array, the system can have different controllers for different sub-arrays or sub-pyramids (e.g., so that different types of objects could be looked for in different regions). If each cell of the pyramid has N processors, each can be assigned to a different controller (e.g., so that N different operations can conveniently be executed, everywhere, in parallel).
* A switching network can be used to give the system the ability to assign and re-assign computers to controllers under program control (Siegel, 1981; Ahuja, 1984).

4.3.3. Multi-Pyramids

Rather than use a single tree, one can expand to a multi-apex multi-pyramid, for example by linking 2, 3, . . . pyramids to a common base. Alternately, NxN entwined pyramids can be got, e.g. by building an NxNlogN network. This eliminates bottlenecks, but at a very heavy extra cost of logN more hardware (which will be largely wasted for a typical perception program that, inevitably, does indeed reduce information as it processes it). Because it applies different processes in different regions of the network it is less able to combine diverse types of information than is a single pyramid that has been commensurately augmented with more powerful processors. Such a structure gives more possibilities for divergence of information than does a single pyramid. But for a full-blown system that does more than process and converge information toward the goal of perceptual recognition, for example also moving into semantic memory searches, problem-solving, and perceptual-motor behavior, structures that as appropriate link 3-dimensional arrays with pyramids and other structures appear to be preferable. Alternately, only a partial network might be built, entwining a smaller number of pyramids (e.g., N, logNxN, or logN).

4.3.4. External Augmentations of the Pyramid (Embeddings Into Larger Networks)

Pyramids can be combined with a variety of different augmenting systems:

* The simplest augmentation would link a pyramid, at apex, base, or/and in-between, to a conventional serial computer. This could introduce a severe bottleneck between the massively parallel pyramid and the single serial computer.
* Link the pyramid to an MIMD network (Figure 2). A number of small networks are

attractive candidates (see Uhr, 1984, 1987). These include ones linked by buses, crossbars and reconfiguring networks, and also N-cubes, trees, MIMD pyramids, trees augmented with extra links at their leaves, and optimally dense graphs - that is, with the largest possible number of computers within a given diameter/distance, using a given degree (number of links to each computer). The computers in the MIMD network can, either individually or in groups, collect, process, and send back information - for example generating and distributing feedback, or concentrating processing on regions of special interest and important problems.

* For example, an 8-node 3-cube might have each of its nodes linked to a different sub-array of a pyramid's base, or to 4 sub-arrays of the base, and 4 sub-arrays from among the higher-level arrays (the links could be either dense or scattered).

* Or the 10-Node optimally dense Petersen (1891) graph could be similarly linked. (Note that this has 3 links to each node and a diameter of 2, whereas a 3-cube with only 8 nodes has 3 links to each node and a diameter of 3.)

* Or an MIMD pyramid (e.g., 4x4+2x2+1x1, or 3x3+1x1) could be used.

* Each computer in the MIMD network can have a sub-array, sub-pyramid, or other sub-structure of the pyramid linked directly to it, as a co-processor. This is much the same wedding of diverse systems as is found in the Cray-XMP and Cray-2, with 4 computers linked together, each linked to a specialized vector processor. The Intel and Floating Point System N-Cubes similarly link each traditional CPU in the N-cube network to a pipeline vector processor.

* The topology and processors of the system can be chosen to match as well as possible the structure of processes to be executed - for example, structures of probabilistic neuron-like threshold elements that compute brain-like processes for perceptual recognition. This suggests networks that appropriately link - for example, as does the brain - a number of each suitably specialized processors.

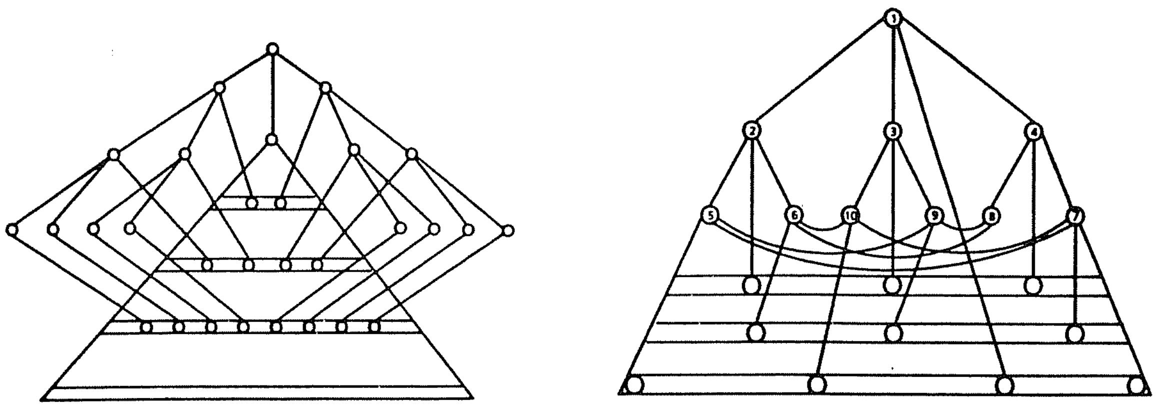

Figure 2. Several Examples of Pyramids Linked to Augmenting Networks.

4.4. Brain-Structured Augmented Multi-Pyramids and Arrays

Arrays of the proper size, shape, and convergence/divergence (both local and overall) to model the retina and each of the other vision-related areas could be linked together using the same graph that the brain uses. The processors might also be made more brain-like: rather than general-purpose ALUs, they might be simple integrating-thresholding elements. Making the processor as simple as possible, with only a very small amount of memory, would allow one to build a system with literally millions of computers. The large number of synapses to each neuron would be hard to handle using today's VLSI technology, but fan-in networks could be used to reduce links to buildable numbers. Much more elegant (not yet feasible, but a real possibility in 5-10 years) would be optical interconnects via holograms off the chip that replaced all on-chip links.

This would be a very exciting kind of system to build. It would also be very difficult to build it, and develop an operating system and programming languages for it. And it would be very expensive. By the time it was finished so much new would have been learned about the brain that it would no longer be the most appropriate. And a system of this sort would almost certainly be far harder to program to do something different from what it was specifically designed to do than would be a more general topology, whether array, pyramid, MIMD network, or augmented pyramid.

4.5. A Tentative Suggestion as to the Most Appropriate Hardware Substrate

An interesting structure, whether to model living visual systems or for artificial perceivers, is a pyramid with bottleneck-eliminating augmentations. This would be reasonably powerful and speedy for a much wider variety of problems than more specialized algorithm-structured architectures. Consider the following:

* Start with a basic pyramid of arrays. This gives the array's massively parallel ability to execute very large numbers of local operations, and the tree's capability to reduce communications distances and to handle the converging and compacting of information with great efficiency and speed.

* Augment the pyramid internally by using more powerful processors, each with its own controller, at the higher levels (e.g., in the 8x8, 4x4, 2x2, and 1x1 layers). Where indicated and cost-effective, also augment memories and links (both increasing the bandwidth of existing links and adding new links) and make processors reconfigurable (e.g., to 1 64-bit, 4 16-bit, 8 8-bit, or 64 1-bit).

* Link the internally augmented pyramid to a small network of independent powerful processors. This network might itself be a small pyramid, or one of the dense point-to-point topologies (e.g., the 10-node Petersen or the 50-node Singleton graph). Or it might more simply use a bus, crossbar, or NlogN reconfiguring network to link computers - whichever is the most cost-effective. The network should be linked to both higher and lower levels of the pyramid, so that it can be used as needed to distribute processed information, including feedback, from one region to any others, no matter how distant.

5. A Brief Look at some of the Ways Pyramids Can be Used for Perception

This section very briefly examines some basic pyramid algorithms and processes (see Burt, 1984; Dyer, 1982; Miller, 1984; Rosenfeld, 1984; Stout, 1983, 1986; Tanimoto, 1983; Tanimoto and Klinger, 1980). The next section will describe the ways that recognition cones have been developed to use micro-modular neuron-like processes to achieve massive parallel speed-ups and robust behavior over real-world variability.

5.1. Multi-Resolution Pyramids

Successively more global sums or averages of the raw image can be computed moving upward. A mask of weights can be convolved to compute a weighted average. Thus some researchers (e.g., Marr, 1982; Marr and Hildreth, 1979) feel the brain may compute a "DOG (difference of Gaussians) - a weighted average described by a bell-shaped distribution (where the weights are highest at the center) and also a 2-dimensional differencing of the surround from the center. Alternately, it might first apply a gradient or an edge detector, and average that. It might apply any other type of transform, to get a color, texture, or some other local characteristic, and average that. Higher-level, larger and more global, features can now be looked for at whatever level of resolution appears to be most appropriate, at several levels, to handle different sizes. This has the major advantage of turning a detailed examination of a large region into a simple assessment of its demagnified parts (possibly also focusing on one feature, blurring, and/or differencing).

5.2. Simple Algorithms for Pyramids

There are a number of basic image processing operations for which pyramids give substantial increases in speed. These include summing, choosing the maximum, median filtering, and other global functions (e.g., min, logical-or) over one or several variables, and also region growing and linking connected components. Consider the following Order(logN) processes: To compute the sum, starting with the lowest level of parents each parent simply sums its children's values and passes these up to its parent to sum. (If a node has more than one parent, as in an overlapped pyramid, only one receives its value.) To compute the max (or min, or parity) each parent examines its children's values and passes up the max (or min, or sum modula 2). For more complex algorithms like component labeling, mixtures of array and tree processes give the most efficient results, often with intermediate levels of the pyramid used to permute information to avoid the increasing tree bottleneck moving upward (Miller and Stout, 1987).

Where trees are appropriate, especially when local processes successively reduce information, pyramids can be very effective. For algorithms that do not reduce information, the pyramid's converging tree structure can present increasingly severe bottlenecks. Ideal are functions that can be computed locally in parallel everywhere, replacing a serial iteration through the array, then their results combined moving up the pyramid/tree. Whereas functions that reduce information - e.g., to give a single global maximum - can be computed and output extremely efficiently in logN steps, functions that order the set of information from maximum to minimum, or in some other way sort and preserve information, will need a minimum of N steps for N pieces of information. The output of the N pieces of information through the single node at the root of the pyramid forms the ultimate bottleneck, however clever might be the algorithm that does parallel local sorts. Similarly, a pyramid algorithm needs a minimum of B steps to output a B-bin histogram.

5.3. Multi-Level Convolutions and Compounds Over Large Regions

Especially important examples of potentially very powerful and efficient pyramid operations are those where a function must be computed over a potentially large region, but that function can be decomposed into cheap local functions. For example, consider convolving a 9x9, 18x18, 64x64, or even larger mask with a much larger array. A serial computer will be slow because it must successively position the mask over each cell in the larger array, get each element of the mask and the corresponding cell of the larger array, multiply them together, add all these results, and store them. A parallel array will be slow because it must shift all information to the processor that is computing the convolution (even though this shifting is done in parallel for all processors). A pyramid can decompose the large WxW convolution mask into masks (probably 3x3 or 2x2 will be the best) small enough to be computed with a minimal number of shifts. Then these results can be successively combined as they are passed up the pyramid, in logW time.

Not only convolutions, but any functions that can be decomposed into local functions and then successively combined, can be handled in this way. For example, indicants of small edges or regions can be successively combined to build bigger and bigger wholes.

5.4. Multi-Level Relaxation and Multi-Level Linking

Relaxation (the constraining of possibilities at each node as a function of what has been found in adjacent nodes; see Zucker, 1976) can take up to 2N iterations on an array. With a pyramid, information can be merged up to higher levels (e.g., or-ing for deterministic relaxation), where the constraint can be applied. Only logN iterations are needed. Relaxation can also be combined with other types of operations. For example, the results of several different edge detectors can be added together, relaxed to eliminate neighboring edges with incompatible slopes, and then those that remain used in compounding operations that attempt to find angles.

The system can attempt to link together regions, contours, clusters, and other conglomerations of parts of the scene that appear to go together by moving up, down, and around the pyramid. For example, when a local color or texture has been identified this result can be passed up (in an overlapped pyramid) to several parents that try to grow regions by looking for matches with other of their children, and in turn pass the results up to their parents and as needed down to children. When a fragment of a contour (at the lowest level this will usually be a short edge) is found, this information can similarly be passed up along with whatever more detailed information may be needed to specify exactly where that fragment started and ended, so that the parent can stitch it properly with other fragments, and pass that information up.

5.5. Computing Binocular and Temporal Changes, Shape-From, and Other Local/Global Operations

When two images are input via two sensors (e.g., TV cameras, eyes), or when successive images are input for a moving scene, the system must, basically, attempt to match sub-images in order to pinpoint those sub-regions where they differ. Pyramids can, potentially, do this very efficiently by making local matches at the lowest level and successively more global and/or more distant matches as successive convergings have moved originally distant pieces of information together. For example, binocular images are typically aligned with one another by looking for matches or correlations of spots, edges, or other local features at individual pixels, or small regions. Where correlations are not found, the system shifts one region until there are sufficient correlations and/or, failing that, interpolates. This is an expensive, potentially Order(area), operation. The pyramid, before each attempt to correlate, can shift one image through siblings, or converge both images up to parents, using whichever operation is more efficient. It appears that successive images of objects in motion can be handled in similar ways (see below).

Shape can often be computed from local information about changes in intensity, changes in texture, relative angles, and curvatures, and other of the many possible indicants of depth. When (as it usually is) the pertinent information is local, array operations are most appropriate. Moving up the pyramid the system can put this information together, and also, as needed, process it further, extracting more global information.

5.6. Embedding and Using Serial Procedures Short Enough to be Fast Enough

Any type of operation, whether parallel or serial, can be executed by a pyramid's general-purpose processors. A major reason for using pyramids is to achieve speed by parallelizing. But serial operations can be used whenever they are appropriate, and not too slow. For example, rather than a family of mask-matching operations to extract short edges of different slopes, a serial curve-follower might be used. After processing each new piece of information, this kind of procedure would decide what piece of information to get next, serially building up its picture of the curve. This is an Order(area) operation. However, if the area is small enough (e.g., 3x3 or 5x5) it will not be excessively slow (but note that separate controllers are needed for each such region, and the system must wait until all processes are completed). Such a procedure will output local edges and curves that can then be linked in any way desired.

5.7. Rotations and Other Transformations in Two and in Three Dimensions

Local information (e.g., about edges, angles, shape, motion) can be used to interact with and adjust information in other regions, whether nearby or distant. Successive combinings moving up the pyramid, using relaxation as appropriate, can handle near-by interactions and, in theory, can handle distant ones as well. Consider the example of rotating an object. First, some local edges, angles, and rectangles may suggest that some object (e.g., a house) might be there, tilted, with windows of a certain size. This information can be used to compute the relative expected sizes of other rectangles moving away, and the angles of other corners of the house. These computed results can then be passed up and down through the pyramid to the nodes that should look for these features.

Easier said than done, this can entail a complex set of criss-crossing messages, and the necessary computations may be quite cumbersome in an SIMD pyramid. Sometimes clever algorithms can produce very fast, efficient programs; but specialized systems like arrays, trees, and pyramids can be slow and inefficient where they are not appropriate. However it is important to point out that pyramids can compute any type of function, and with more flexibility and efficiency to the extent they are appropriately augmented (e.g., with more controllers and wider-bandwidth links) - albeit with more expense.

6. Recognition Cone Perception Programs, and Their Ultimate Goals

We are now in a position to examine recognition cone systems. Basically, they attempt to chop up, re-structure, and combine complex procedures like "find houses" or "recognize faces" into shallow tree-like hierarchies of very simple procedures that can be executed in a massively parallel, hence extremely fast, manner. They make use of hardware structures needed for powerful, flexible, robust, fast perception, especially arrays for massively parallel local operations and trees for combining or diverging information. Their underlying structure is therefore that of a pyramid of arrays, but with significant internal and external augmentations where these are needed to expedite a more complex flow of more independent processes.

The models of objects are encoded in structures of transforms (probabilistic neuron-like production-like IF <conditions> THEN <consequences> procedures) embedded in the pyramid, as indicated in the following very simple examples (numbers are weights):

```
IF CONVOLVE gradient-array WITH
          -1 -3 -5 -3 -1
           1  3  5  3  1        GREATERTHAN 0  THEN  hor1,5;

IF COMPOUND OF
     vert:2   vert:1
     hor1:1   hor1:3    hor1:1    GREATERTHAN 17  THEN  L-angle:27;
```

Transforms search for their structured sets of IF-conditions, combine the weights of those that succeed, and evaluate whether the transform as a whole has succeeded (e.g., because the combined weights exceed a specified threshold). When a transform succeeds, its THEN-consequences are merged into the indicated processor memory (for array operations), parent memory (for pyramid operations), or/and other structures (e.g., lists of things to lookfor and procedures to apply). This makes possible the combining of large numbers of no matter what type of information, whatever transforms specify.

6.1. Expandability of Recognition Cones to Handle Other Cognitive Processes

This structure was chosen because it can, potentially, be expanded to handle language, problem-solving, motor behavior and learning. There is no space to do more than mention these possibilities here.

The object-models of transforms used by the perceptual system have much the form of semantic memory networks. Indeed, among the information a perceiver must recognize are symbols, including letters, words, phrases, commands, sentences, and so on. Recognition cone programs have been given transforms that allow them to recognize letters and words (e.g., "touch fruit") as well as simple objects (e.g, banana, box), along with additional procedures that determine and carry out the import of semantic information. They are also thus able to use the object-models that effect recognition to help in semantic problems like describing objects and answering questions. This entails building, and making inferences using, semantic/cognitive networks. These systems make use of referential information, and determine what is a referential symbol and what it refers to. They can use perception to monitor and control motor behavior, referring feedback information to previous actions and percepts and modifying actions accordingly.

Several types of learning are also possible. Transforms can be modified by changing weights and other attributes of implieds and conditions, changing the threshold of success, and adding or modifying implieds, conditions, and their attributes. The whole structure of transforms can also be built, generating new transforms and evaluating how good they are, and discarding poor transforms. Since learning consists in changing the structure of transforms the program that applies them is protected from disruption.

6.2. The Basic Criteria and Goals for Recognition Cone Models of Perception

These programs are designed to address the following central issues:

1. *Recognition is effected by living systems in only 20 to at most 400 steps.* This is an existence proof that massively parallel, shallowly serial, artificial intelligence systems are possible, and can be built with all the perceptual capabilities of cats, monkeys, and human beings. Such speeds are needed for general real-world vision system.

2. *Information is input via a very large parallel 2-dimensional sensor* that transduces it into suitable form for further processing.

3. *A perceiver must be capable of recognizing very large numbers of different objects.*

4. *These objects must be recognized even when they are severely distorted or transformed* in any of a large number of different, unknown, ways (e.g., different chairs, running dogs and other animals, facial expressions).

5. *The system must be able to recognize and describe a whole scene of objects* - several or dozens but not hundreds or thousands. It must also be capable of describing individual objects - not by rote but emphasizing what is salient, pertinent, unusual, relevant.

6. *Moving, changing objects must be recognized,* their motions and changes described.

7. *The basic units that the system uses should be brain-like.* This suggests it should be built from a large number of very simple neuron-like probabilistic micro-modules.

8. *Basic units should combine into larger structures and processes* that are reasonably brain-like, or at least not implausible. These include layered pyramids, fan-out/fan-in nets, feedback loops, simple matchers of local windows of information, and processes that converge and combine information, and compare and choose.

9. *The overall structure should be reasonably brain-like* - information input to a retinal array, successively processed, locally converged and diverged, and sent to other areas.

10. *Models of Objects should be constructed from graphs of basic units.* The nodes of each object-graph should be combined and embedded in the larger structure.

11. *The system should routinely combine and decide using diverse, incomplete, information.* No matter what type of information is missing or defective, it should make necessary inferences and choose among alternatives.

12. *The system should be highly fault-tolerant and adaptable.* It should work even when parts are defective; degrade slowly and gracefully; make as good as possible a response rather than stop cold; use feedback and experience to improve its performance.

13. *The structures of micro-modular transforms should be learnable,* got by applying a variety of plausible learning mechanisms to information from the external environment.

14. *Processes should be probabilistic, approximate,* flexible, robust, human-like.

7. How Recognition Cones Can Be Used for Perception

The following are relatively simple examples of actual recognition cone systems that can be realized today. To be feasible, they make a number of small and big abstractions from what is already known about actual brains. For example, the first programs to be described take a largely bottom-up approach, starting with the given input image and successively transforming, extracting, and abstracting information moving upward. A full-blown system must merge this with more goal-directed top-down processes. Occasionally these programs have only the beginnings of some of the desirable properties sketched out above (e.g., little fault tolerance; only when given much larger sets of transforms will they be robust over a large number of real-world distortions).

7.1. Using Recognition Cones to Model More Details of Living Visual Systems

Programs are described as pyramids to make clear how they would actually be coded for a hardware (or simulated) pyramid. But less pyramid-like structures, for example the living visual system's network of areas, can be programmed. For example, if one area diverges its output to several areas that each work in parallel and then converge their results to another, this can be handled efficiently in a pyramid by having a single one of its arrays execute the processes for each of the parallel areas in turn (using a different bank of memories for each, then converging the results into the same memories). If these areas have different sizes, shapes, or linking structures, the system can simply have each processed by whichever array matches it best. If these most-suitable arrays are not adjacent, the system can pass the information through intermediate arrays unchanged.

In this way although the actual flow of information through processors in the hardware pyramid may be different, the desired flow is being modeled by the program. That is, the data-flow graph is as desired, although the actual execution with the less-than-isomorphic hardware available is to at least to some extent different, and slower and less efficient. [Note that the programmer can decide which of the many possible options, only some of which are mentioned below, to choose.]

7.2. The Flow of Processes in Recognition Cones

A. *Information is input into a large 2-dimensional retinal array.* Each processor in the 2-dimensional array of computers that forms the retina has in its memory one pixel of the image (or, if the processor array is too small, a sub-array of pixels).
B. *One or more processes are applied either in parallel or serially* (that is, each process is applied to the output of the previous process), or in a mixed parallel-serial mode (this is up to the programmer). The processes applied can be of any sort. However probabilistic IF. . . THEN. . . transforms that execute local array operations (e.g., where a small compact window of information around each cell is processed) are the most highly parallel and probably the fastest and most efficient.
C. *Whenever desired, the output of a process is merged into the parent array.* That is, rather than store the result in one of the processor's high-speed registers or in its main memory, the result is sent to the linked processor(s) in the next-higher (and smaller) array. Each parent must combine several pieces of information from its several children.

D. Whenever desired, parents send information to children. This might be a broadcast to all children, or specific information might be sent to a specific child or children. This information might be feedback, requested information, a request, or an order to execute some process or answer some question.
E. Whenever desired, processors send information to one another via siblings and/or ancestors, whichever is most efficient.
F. These kinds of processes continue to be executed, at every level of the pyramid.

For efficiency, the programmer, programming language, or/and operating system must load-balance processes across arrays, synchronize arrays reasonably well to pass information in unison in the same direction (that is, all passing upward, or all passing downward), and minimize unsynchronized passing of information that interrupts processing.

8. Simple Examples of Actual Image-Driven Recognition Cones

A general perception system must be able to notice and recognize anything it knows about, and not just a few things it has been coded to look for in top-down fashion. To cope with the unanticipated, it must include an image-driven bottom-up structure of processes like the following simple example:
1. The image is input to the retina, where it is averaged to smooth it a bit and eliminate specks of noise, and then differenced by a local gradient detector.
2. The next array looks for a family of local oriented edges, at 0, 45, 90, . . . degrees; and also for color and textural features. This is most quickly and economically done by applying small window masks (e.g., 3 by 3, 3 by 5).
3. The next array combines the local edges into longer edges (including longer straight line edges and also simple angles and curves). It also combines the local colors and textures into larger regions.
4. Up to this point the system has achieved a shallow hierarchy of a variety of different types of features. It can now use these to begin to imply possible objects, and it can also use them to continue to search for more and more global features, characteristics, subobjects, and objects.
5. The system chooses among alternative possible labels to store or output, and additional processes to apply. This must be done to generate the final output, and can be done (e.g., to prune) wherever desired.

Programs of this sort, ones that apply bottom-up sets of local probabilistic threshold operations (transforms), have - when given a small set (e.g., edges, angles and curves, enclosures and regions, simple objects, more complex objects) - recognized a variety of simple objects (Uhr, in preparation; Li and Uhr, 1986). When given a large enough set of carefully chosen transforms they have recognized complex objects like knives, forks plates, place-settings, windows, doors, houses (Uhr and Douglass, 1979). When given a still larger set of transforms, but one that is not unreasonably large, this kind of program should be capable of recognizing a larger variety of these objects, and a larger number of different objects. Comparative timings (Uhr et al., 1982; Li and Uhr, 1985) indicate that for large images at least million-fold speed-ups can be expected.

Depending upon the noise, distortions, and other types of variability in the images, the system might insist upon everything being perfect and exactly as specified; or it might allow for variations. As we move into real-world images of more complex objects, processes must accept more variations. This suggests that each local process, one that is plausibly executed by a small network of neurons, should be relatively imprecise, hence more general - and also therefore more fallible. For example, an operation might imply a long vertical edge when only half or fewer of the maximum number of smaller vertical components are found, and some of these are slightly misplaced. The system must now use and combine the results of a larger number of processes that examine different aspects of the complex feature, in order to build to adequate performance.

No system can recognize objects about which it has no information. But the information and procedures it is given must be general enough so that for each new variant of the same object, and for each new object, the program does not need commensurately more information. The recognition cone approach attempts to achieve the necessary generality, efficiency, and speed by combining several different capabilities:
* Each transform can be made more general - by using more general components, and by adjusting weights and thresholds so that it will succeed under as wide as possible a variety of (hopefully correct) circumstances.
* Each thing (feature, characteristic, sub-object, object, or whatever) can be implied by any desired number of different transforms, and each transform can imply any desired number of things. These systems' basic combining capabilities thus make it possible to construct families of transforms to handle complex variations.
* The model of each object is decomposed into a multi-level converging tree-like set of transforms (Figure 3), so that it can be applied in a parallel-hierarchical manner. All models are embedded into the overall pyramid structure.
* When several objects have features in common (this is usually the case for lower-level features like edges, curves, angles, textures), the transform need be applied only once.

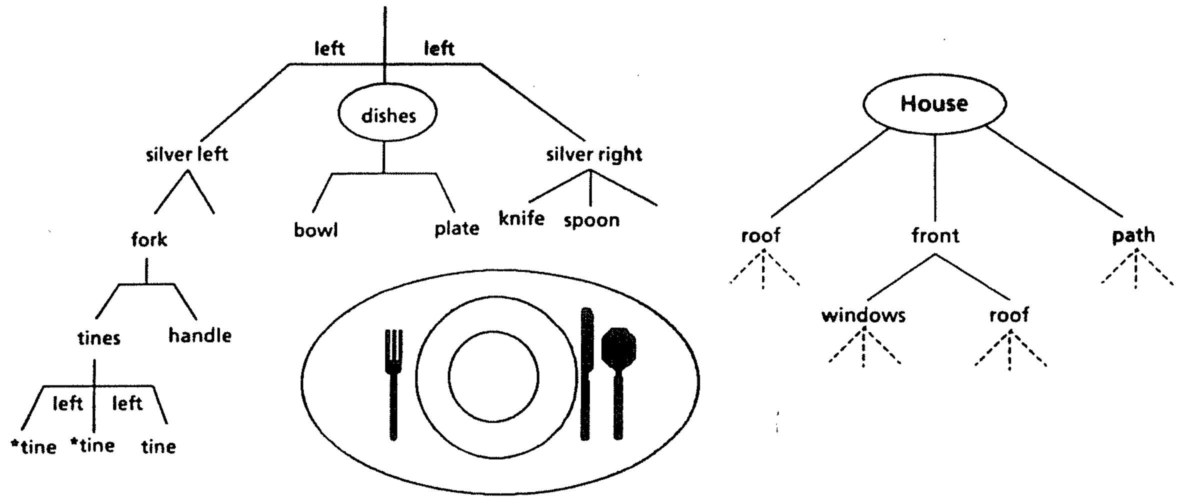

Figure 3. Simple Examples of Object-Models Represented as Graphs of Transforms.

9. Parallel Reasoning and Inference, to Increase Power and Flexibility

Potentially powerful parallel reasoning capabilities can be implemented in suitably augmented pyramids in any of several ways.

9.1. A Divide-and-Conquer Look at Separate Problems in Parallel

Rather than gather some information, then try to figure out what to do next, then do it, continuing in a strictly serial manner, the system can attempt to follow such a procedure in a divide-and-conquer manner at several different locations, and/or with respect to several different possible objects. This is straightforward if each sub-problem can be handled independently. A pyramid with a separate controller for each of the sub-pyramids in its 4-by-4 or 8-by-8 array might assign one or more of these sub-pyramids to each particular problem. Information can be shifted so that work is balanced over all the sub-pyramids, and each sub-pyramid contains all the information it needs. Alternately, the pyramid can link to a small MIMD network, a different computer assigned to each problem. Inputting the entire image to each computer will eliminate problems of communicating information or of accessing and sharing memory. However, such divide-and-conquer procedures are not likely to offer substantial speed-ups, and the same information may be examined redundantly by several different processors.

9.2. Using Key Features to Combine Image- and Model-Driven Processes

Recognition cones can be extended with relative ease to combine bottom-up, top-down, and lateral movement of information. The basic structure of the micro-modular transforming operation makes this possible, along with the subsequent steps of assessing, converging, and combining information, and choosing among alternatives. Information can just as easily be assessed (using an IF . . . THEN . . . neuron-like probabilistic threshold operation), converged, and combined (e.g. getting mean, min, or a compound) whether it is sent from below, from above, or from anywhere else, and the resulting alternatives chosen among.

Up to now it has been implicitly assumed that information on the right-hand-side of the IF . . . THEN . . . transform is merged into the parent cell. But some other location can be specified (e.g. a near neighbor or more distant sibling, grandparent, or child cell, or some cell that is external to the basic pyramid) and the information merged there. When actually implemented in pyramid hardware, the system may need to shift information through some up-down-around sequence of moves, along whatever is the most economical and convenient path; or alternately additional hardware links may be justified for frequently followed shift patterns. The links between IF and THEN must be made 2-

way, and it may be desirable to use flags that indicate a transform succeeded.

The following describes a system (Li and Uhr, 1987) that, by combining image-driven (bottom-up) with model-driven (top-down) and lateral operations succeeded in recognizing several different types of buildings and their component parts (Figure 4).

A. The program applies a sub-set of its transforms, those that assess "key features." It is up to the programmer what to choose as key features, but in general these are transforms that should give the most information - for example because they are more often present, tend to be invariant over distortions (e.g., size, shape, gaps, irregularities), and/or are most cost effective.

B. To break out of the rigid pyramid structure, where slight variations in translation can mean that something will match a quite different set of transforms, the program makes local lateral searches for features within a particular layer or adjacent layers.

C. When at the higher levels possible objects are tentatively implied, the program makes top-down searches using transforms that assess other aspects of these objects.

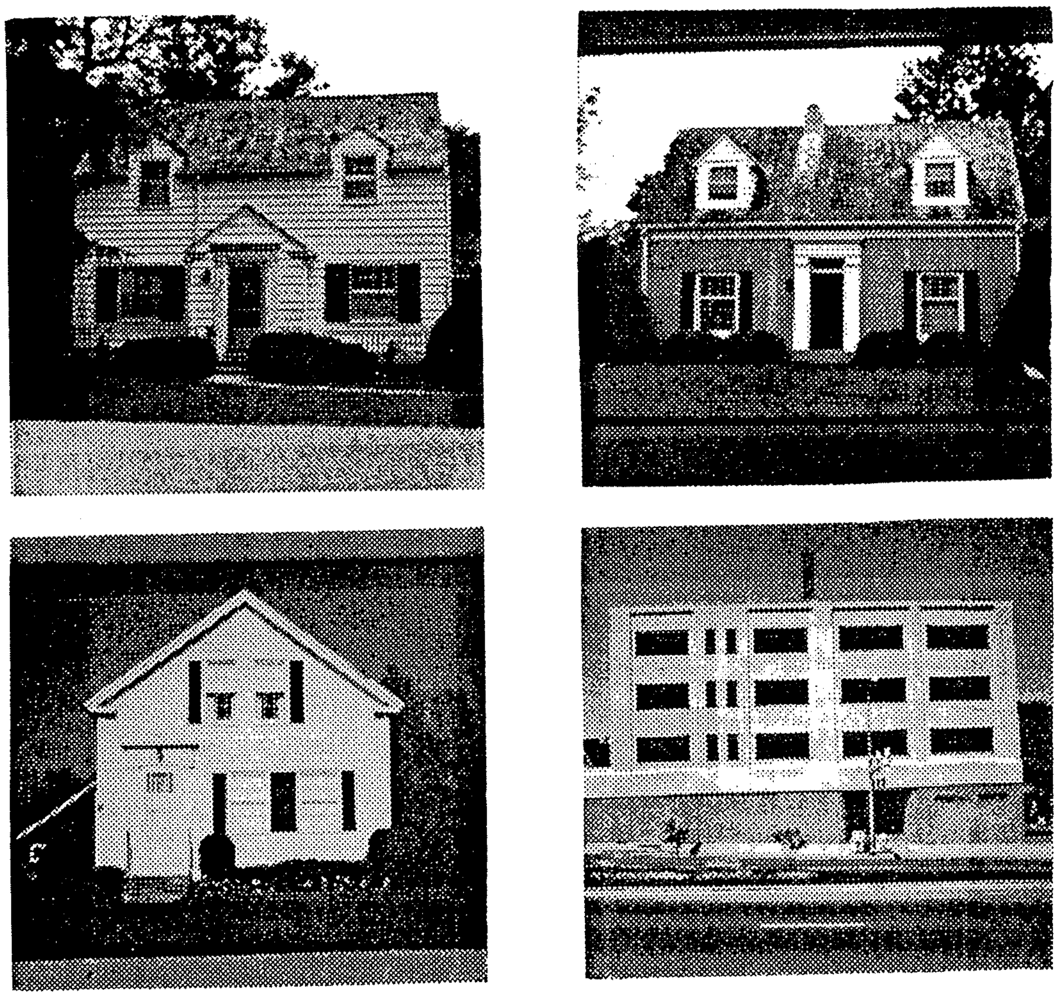

Figure 4. Digitized 512-by-512 Pictures of Buildings That Were Recognized by a Highly Parallel Image- and Model-Driven Program (Li and Uhr, 1987).

9.3. Cycling Through and Combining Image- and Model-Driven Processes

A more general program would more intimately integrate bottom-up and top-down processes, as follows: At each layer, apply whatever transforms are active. Start with a set of key features that are active, plus a set of objects to be looked for that are active. The objects to be looked for back-link to transforms, which in turn back-link to lower-level transforms, giving a top-down component. Follow these back-links, and combine what they point to with the bottom-up set of key transforms, and apply them both. In addition, apply transforms at adjacent locations (by shifting them laterally to siblings, and also up to parents and down to children). Here we see what are really reasoning and inference mechanisms embedded into a parallel system.

Greater speed-ups can be achieved by simultaneously executing bottom-up and top-down processes. Consider the following simple example: Each key feature (that is, a transform that is always applied) implies one or more higher-level things. Each of these things is implied by and back-links to one or more transforms that are not key features. A minimal fan-out to 2 implied things each of which back-links to 1 non-key feature already allows each key feature to imply up to 2 times as many new features to apply at each level moving upward. Larger fan-outs can be used as desired. Probably the best would have each feature imply whatever number of things it is a part of, context for, or in other ways implies, and each object be implied by whatever number of features is needed. That is, the more complex the object and the more difficult a problem it poses the more transforms should be involved in its recognition.

9.4. Combining Dynamically Implied Things to Lookfor with Key Features

An additional potentially very powerful top-down component can be got using transforms augmented to imply transforms-to-apply and objects-to-be-looked-for (that is, dynamic procedures) as well as features, objects, and other static things. (These are specially tagged elements in the transforms' right-hand-sides - usually at higher levels of the pyramid; but programmers can put them anywhere). These objects-to-be-looked-for then back-link down their object-graphs to the structure of successively lower-level transforms that imply them, and these are in turn applied, moving up the pyramid.

The program can be given an empty lookfor list, so that it begins with the bottom-up application of key features that, as they imply things that back-link to other features and also imply things to lookfor and transforms to apply, introduce a top-down component. Or it can be focused on one or a few things, by putting only their names into the lookfor list. A program that processes a sequence of related images (e.g., all house scenes or all pictures of fruit, or a movie) can put things into lookfor that will then be used for subsequent images. Thus parallel top-down processes use whatever objects the system now implies, or has inferred or been told, it should look for in each particular region.

9.5. Simultaneous Brain-Like Cycling of Bottom-Up and Top-Down Processes

Assume a new image is input to the retina at each time unit, every 30 msec or so. The system can process up, then all the way down, then up, as before. But it can in parallel move from each level where new things are implied back to apply additional transforms that imply those things at the next lower level - doing this at every level. When a thing is implied, it can back-link so that the lower level will next apply other transforms that imply it, appling both key features and implied features at each time unit. Dynamically implied things to lookfor and transforms to apply also introduce more global interactions. When back-linking from an implied thing the system moves down only one level in the hierarchy, and applies any additional transforms only in the several children cells. But things to be looked for and transforms to be applied can be implied from any level to any other level and then applied globally, wherever designated.

At Time 1 the key features at level 1 (the level that looks at the raw retinal input) will be applied, and their results merged into level 2. At time 2 the key features at level 2 will be applied and their results merged into level 3 - but at the same time all the features at level 1 that imply the things that were implied at level 2 are also fired (since they were back-linked to by these implied things). This back-linking process from things that were implied and hence are to be looked for is carried out at every level. In an N-level structure, after N time units all levels are applying not only their key features but also the additional transforms implieds back-link to. This means there is a constant bottom-up and simultaneous top-down flow of information to each level from its neighbor levels immediately below and immediately above. If all goes well, new tests are being called for to confirm each tentative identification, and therefore to discriminate among the alternative possibilities.

To keep within bounds, implieds can be faded by lowering their weights at each time unit (dividing or subtracting by a small number), erasing those whose weights go below a threshold. Weights can be kept from going above some upper bound, and their distribution re-normalized. And the system can prune, erasing all but a fixed number of the most highly weighted things. Such a system also needs mechanisms that tell it to stop. The simplest would stop after 1, or k, cycles from bottom to top and back, or after a fixed amount of time. The system can also decide whether to stop, using the mechanism that chooses among competing alternatives. The system can decide to stop when a winner is sufficiently strongly implied, is sufficiently above its chief competitor or competitors, and/or whether the cost of continuing is too great.

9.6. Serial and Parallel Reasoning Techniques Contrasted

High-level reasoning is typically incorporated into a perception system by first applying low-level bottom-up operations to segment the image (usually by getting contours and regions); then reasoning about what high-level object-models are most likely to be found, and how best to look for them. Thus the system might start to look for a house (because it has been programmed to do this first), by searching for a roof above a wall that contains several windows and a door. Upon finding a wall but no windows, it might reason that this might be a barn, and look for a very wide door, cows, and other signs. And so on.

Consider an analogous bottom-up example: Starting with the image, finding red patches and pebbled textures implies bricks, and suitably interrelated vertical and horizontal edges imply corners. These in turn imply windows, doors, and walls, which imply buildings. This is considered reasoning when each of these is done serially, with decisions interspersed to control what to look for next. But we have already seen how these kinds of processes can be executed much faster using parallel pyramid operations.

Similarly, upon finding a vertical ending in two right angles that extend to the right, the program might infer that it should look for a vertical with two right angles extending to the left. If it finds this, it might reason that these are the two sides of a building, and try to join them with a roof above and a foundation or grass below. If the angles at which the original vertical is joined to edges that extend to the left are not right angles, the program can pose and investigate several alternative hypotheses. For example: if the top edge slopes up but the bottom edge is horizontal, look for a peaked roof. If both slope up slightly, see if the building is being viewed from an angle so that it recedes from its left wall into the distance, and begin to adjust other features accordingly (e.g., expect a smaller right vertical for the more distant side of the building, and left-facing angles that slant appropriately downward to meet the right-facing angles of the left wall).

More generally, this kind of program will gather some information, reason about this information and infer its possible alternative consequences, decide what to do next (make some final choices and/or gather more information), and continue in this cycle until it has finished or time has run out. Reasoning gains power by augmenting top-down with bottom-up and lateral components. Recognition cones that use key features and objects to look for (both built-in and implied by what has been found so far) are reasoning in a parallel-serial manner that, potentially, can enormously speed up processing. Serial processes can use conditionals more freely, and it is much more difficult to develop parallel structures. But serial reasoning about something so complex as a scene of real-world objects is far too slow.

10. Recognizing and Describing Scenes of Several Objects

Only the very simplest of real-world images will contain but one single object. Much more common, more important, and much more difficult to handle, are images that contain a variable number of different objects. A complete perceptual system should also output the salient relations between the named objects (e.g., woman walking dog; spoons, knives, and forks properly positioned with respect to plates; man smiling at woman), higher-level groups (e.g., forest of trees, living room suite, fruit bowl, family), and important descriptive information about individual objects (e.g., awkward, faded, dusty, thick legged, smiling, wearing a red dress, wrinkled). For single object recognition, the perceiver can combine all the information it gathers into a single grand decision. To recognize and describe several objects, the perceiver must refer the information it gathers to the appropriate objects. Probably the most straightforward approach is to search top-down for each possible object's model everywhere in the scene, successively searching for each part, all properly interrelated. But this is much easier said than done for real-world images; programs that search for the parts of a high-level model have succeeded only in carefully constructed artificial images. And only a small handful of models can be searched for using this extremely serial, hence slow, approach.

Pyramids have several capabilities that offer real promise of achieving much more parallel, hence faster, systems. Let's assume that we have achieved a bottom-up/lateral/top-down pyramid program that can recognize a single object, whether it is large or small, no matter where it is in the image, and no matter how distorted. This program will recognize local and successively more global features, and sub-parts, simple objects, and more complex objects, at whatever level they become recognizable. This means that, when all (or most) of the object lies within the sub-array that forms the base of the sub-pyramid for which a node is the apex, the object can be chosen and recognized at that node. This information can then be passed up toward the grand apex (and also as needed in other directions), along with the location at which the object was found, the features that were found that implied this object, and any other important attributes.

If passing all this information up to the apex is too slow and costly, the system can:
* Search down for each needed piece of information.
* Re-process the raw image, moving upward, but this time pass up all the information, or a requested subset.
* Input and process a new (closely related) image.
* (Probably the best approach) use a judicious combination, for example one that passes up the most important attributes, searches down when needed for easily found attributes, and requests that any others needed be passed up when the image is re-processed.

This kind of system can compose a whole scene's description from the set of tree-like graphs of transforms that serve as its model of each chosen object, and the sub-graphs of nodes that actually succeeded for these instance of the objects. In addition, some of the chosen objects will be parts of higher-level objects and groups, and there will be similar graphs for other almost-chosen, hence in ways similar, objects. The simplest output would be a standard description - for example giving each chosen object's name, the N most highly implied nodes on the tree that led to its choice, and the one or two objects that it looked most like. A more sophisticated procedure might search for and output additional information - for example how each object differs from the usual. These pieces of information might be output for each object in turn, or in descending order of weight, or moving from the highest level downward, or in some combination of the above.

It seems unlikely that there can be very satisfactory standard descriptions of this sort. The program will have uncovered large amounts of information, most of it trivial but much of it of some interest to some people under some circumstances. Different people will want to know different things about the scene. Ideally, a description should state what is "salient," "interesting," "important" - all ill-formed concepts that are very hard to capture in code. But the program can be put into an interactive dialogue with the person who wants information about the scene, that person's intelligence guiding and shaping the description. So an attractive alternative is to write procedures that output only one or a few pieces of information at a time, then input questions, suggestions, and/or commands that tell the program what to do next. For example, "more" will elicit more features of the object presently being described; "next" will elicit the name of the next object; "what's to the left" will elicit the name(s) of object(s) to the left of the object presently being described; "how differs from X" will get features that implied this object but not object X. This kind of interactive dialog can continue until the human recipient is satisfied (or gives up).

11. Handling Binocular, N-Ocular, and Multi-Modal Images

There are several important computer vision problems where two or more images must be examined in combination. The perceiver must overlay this information, combining parts that go together.

* For binocular or N-ocular vision the several eyes' images must be combined, reinforcing each other and also leading to inferences about the depth and shape of objects.
* For images from different sense modalities the information about each object must be overlaid and combined.
* For successive images of objects in motion, each object's several images must similarly be combined, and used to help recognize objects and infer their trajectories.

11.1. Binocular Vision for Perfect, Distant Images

A system that inputs two images, as from two eyes or from two partially overlaid snapshots from a satellite, must, basically, find correlating matches. When there are perfect images, no distorting factors, and no differences in details, one image can be translated with respect to the second until a perfect match, or correlation, is found. For example: start with the two images superposed as any prior knowledge suggests would give the best match; move one image through the 8 cells in the border of the 3x3 window surrounding this point, matching after each move; move through the border cells of successively larger 5x5, 7x7, . . . windows; stop when a perfect match has been achieved. This needs up to NxN matches. But when (as is usually the case) the two images' expected relative locations are known only a small narrow rectangle surrounding the scan line joining them need be searched.

11.2. Less-Than-Perfect Images with Binocular Disparities

When different objects are at different distances their disparities will differ and nowhere will the two images match entirely. Therefore sub-parts must be matched. In addition, real-world images will not give perfect matches simply because there will be variations in lighting, shadows, and actual details as viewed by the two eyes. No longer solvable with a perfect rigid match, the situation becomes one with rubbersheet distortions, along with surface noise and different local details (from different views of objects that recede into the 3d dimension). With such less-than-perfect images, a potentially much more difficult stopping criterion must be used: stop when the match has peaked to a sufficiently high level, but then starts going down. And often different regions of the image will correlate to different degrees, so that an overall correlation is not sufficient.

A serial top-down approach would look for and attempt to recognize each expected object in turn in one or both of the images. When found the system would attempt to match an object with something similar in the other image. The objects found in one image might be used in a top-down manner to look for their parts in the same, and then in nearby, locations in the other image.

Pyramids offer a potentially elegant and very fast structure since they compute local matches at each and every node in every layer, combining them into successively more global matches moving up the pyramid. These local matches can be used to find highly matched regions (e.g., ones with flat facing surfaces and firm edges, or easily found landmarks), and also to shift sub-regions until they achieve local peaks. Each image is input to a retinal array, and transformed and converged. The pyramid first applies a structure of feature-detecting transforms. Any of the successively more abstract images in any or all of the pyramid's layers can now be matched. From the layer at which a good local match occurs (that is, a match that is got by a single node and therefore refers to the sub-array at the base of a sub-pyramid for which this node is the apex) the object's distance can be estimated, since this gives the relative displacement of that object in the two images. More exact estimates can be got by moving down the pyramid. Then these are matched and the results merged. A single hardware pyramid can handle this, storing the different images in different memory planes.

11.3. N-Ocular Vision

These methods for combining images will generalize to handle 3, 4, . . . or N different images. If all images are the same size, simply iterate any of the above processes. If they are not, first magnify the smaller ones to make them all the same size. For example, if the smallest is 128x128 and the largest is 1,024x1,024, each pixel in the smallest can be replaced by an 8x8 array where every cell contains that pixel's value, possibly with an additional step to smooth.

11.4. Combining Images from Two or More Different Sensory Modalities

Different sense modalities (e.g., vision, hearing, touch) often give important different kinds of information that, if they can be overlaid and used together, will appreciably improve perception. They can be fruitful sources of different types of features and characteristics. The cone/pyramid structure, which is designed to combine diverse types of information, is an attractive system for handling multi-modal perception. The local matches, successively combined moving upward, described for binocular and multi-ocular vision can also be used here.

For example, a small array of touch sensors (roughly like the palm of the hand) might feel edge, corner, and surface information that can very usefully be combined with information from a TV image. First the resolution of the two sensors must be made the same, by magnifying the coarser one, and they must be properly positioned with respect to one another. Then the information from the two of them can simply be converged together, and used exactly as though this was all visual information. This is possible because the cone's basic transform processes look at any type of information, no matter what the source. When source, or any other attribute, is known, that information can be passed up and stored, or back-linked to in a top-down manner.

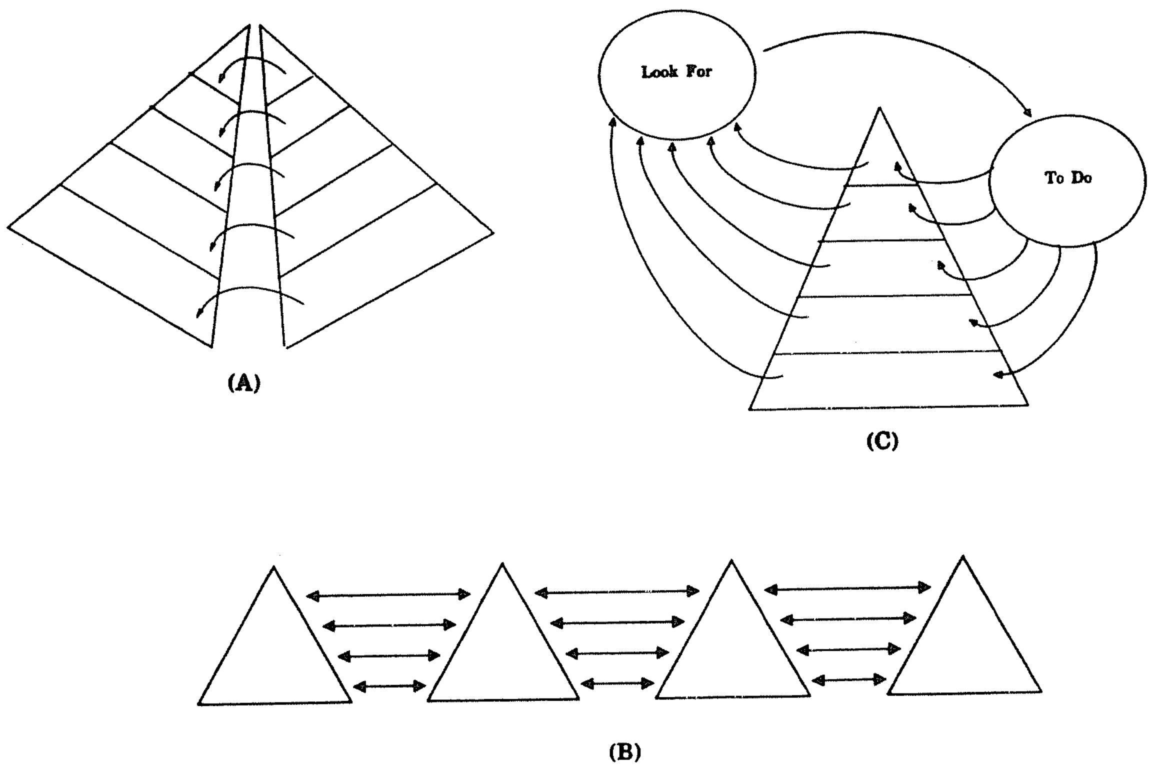

Figure 5. Examples of Structures that Combine and Augment Pyramids (Hardware or Software) to Handle Multi-Sensor or Motion Information.

12. The Perception of Objects in Motion

When a TV camera is used to input, every 10-50 milliseconds or so, successive static images, a perceiver will be overwhelmed with information - unless it is embedded in a suitably structured massively parallel multi-computer. Consider a very simple case: A sequence of two images containing unchanging non-occluding objects, all of which are stationary except for one single object that moves laterally (remaining the same distance from the observer and presenting the same view) in a known direction at a known speed. The program can immediately match everything that has not moved simply by subtracting one image from the other, leaving a template in each image with the precise shape of the moving object. Then it can shift the 1st image the proper direction and distance and subtract the 2d, to match and eliminate the moving object. Finally, it can apply whatever recognition procedures it uses to the remains. This is true only if absolutely nothing changes from image to image except for the position of the moving object, and also this object moves fast enough so that its two images do not overlap. If the moving object's images do overlap things are more complex: The overlapped part will be subtracted out of the image at the first step, so that the object's template will be broken and distorted in the middle. The leading and trailing edges should be sufficient to determine the motion; but recognition can be far more difficult, depending upon how much is missing and whether the perception program insists everything be there.

For the real-world problem, the perceiver will not know the direction and speed of the moving object. So it must be able to shift the 1st image to all possible places. This Order(NxN) process can be accomplished by spiraling through successively larger concentric squares, checking after each shift for zero difference - in which case the moving object has been found and matched, its direction and distance the direction and distance the first image has been moved. A more sophisticated program might when the match improves make a directed search in that neighborhood. Things become far more complex when images are noisy; or objects occlude, move in 3 dimensions, or change in shape; or/and several objects move in different directions at different speeds. It can no longer be assumed that everything remains unchanged except for motion; nor will simple subtraction eliminate the object completely. The system must use world knowledge to infer changes and judge what can be ignored. This inevitably takes too much time unless it is done in a highly parallel manner.

Parallel arrays can subtract or correlate to match two images using only a small sequence, k, of instructions (but they must in the worst case make NxN shifts and correlations). Serial computers need kNxN instructions, a far larger number. Serial computers can, at least in theory, reduce this drastically - but only by recognizing objects first, outputting their names and locations to short lists, and then matching names in the two lists. Alternately, the system can attempt to recognize objects in the first image; then make a top-down search for these objects in the second. In either case all the objects must be recognized - inevitably an extremely time-consuming process when done serially.

12.1. Pyramid Processing of Moving Images

Pyramids can, potentially, reduce the O(NxN) time needed by arrays and serial computers to O(logN). They can also handle images containing several different objects in motion. Rather than shift the image through the NxN array, the program can move, at the same time condensing and abstracting, the image up through the logN arrays of the pyramid. Since the pyramid embeds successively larger arrays of sub-pyramids at each level, each of its sub-pyramids can, potentially, handle a different object in motion.

The simplest procedure first correlates and subtracts the two raw images; then builds a pyramid for each image by sending averages of its children to each parent; then correlates the two pyramids. The sub-array under each cell that finds a correlation contains an object that has moved from one of its borders to another (if the movement had been smaller the correlation would have been got at a lower level). All processors at all levels can work in parallel, and several different objects can be captured in this way, in parallel. The information about the motion of each object can be combined with information collected from the previous images to compute an up-dated trajectory.

Rather than simply average images, the program can apply a structure of operations that attempt to recognize objects. The resulting tree of successively abstracted and compounded features (or, if desired, some sub-set) can be correlated across the two pyramids to assess motion. A program of this sort coded in C for the VAX to simulate a pyramid successfully recognized and tracked several different simple objects moving in different directions (Uhr, unpublished). This kind of program handles recognition and motion detection in combination, with the possibility of each helping the other. The greater the motion the higher in the pyramid (rather than the farther away in the array) will the correlation occur. Successively more exact direction and distance information can be recovered by searching down each subpyramid whose apex correlated.

Uhr, 1976, described and implemented in a demonstration program a potentially more efficient technique: Rather than use a different pyramid for each image, pipeline the successive frames of a moving image up through a single pyramid, applying object-detecting transformations at each level. Rather than initialize the pyramid by erasing everything after each image has been processed, simply reduce weights a small amount and erase only those things whose weights have fallen below some minimum threshold. (This can be adjusted to keep implied things around a suitable amount of time.) Now tag as "new" anything the first time it is implied into a particular cell. Whenever something that was tagged as new is then implied into a parent that already contains that thing, this means that it was previously recognized in the immediate past, but at a different location. Thus the single pyramid contains a time-weighted average over the recent past that it correlates with the new image. It continually processes the pipeline-like flow of successive images as they are transformed and moved upward, simultaneously recognizing objects and their motion.

13. Toward Software/Hardware Structures That Rival, Or/And Model, Living Visual Systems

An actual hardware pyramid-of-arrays, augmented internally with more powerful processors at higher levels and externally with a small network of more powerful processors, may well be the most efficient and cost-effective way to achieve a multi-computer that is sufficiently robust, flexible, powerful, and fast to recognize scenes of complex real-world objects in real time. It would be far faster than a single-CPU computer, or a pipeline or an array. It would execute appropriately structured programs with potentially great efficiency as well as speed. More powerful still (but possibly less cost-effective) would be a more extensively augmented pyramid that included diverging as well as converging structures; several different types of pyramid, array, and other suitably specialized processors; and much richer sets of links, to nearby and to distant processors.

Pyramids are general enough (at least when suitably augmented and linked to a network of serial computers in relatively simple, straightforward ways) so that a great variety of algorithms and programs can be implemented on them with reasonable efficiency and great speed. They are best viewed as convenient and efficient computational structures for executing highly parallel perceptual processes, whether brain-like or not. Ideally, the hardware and software structures should be developed together, as complementary parts of the single system. This dictates how the pyramid's hardware should be modified and augmented, to achieve actual, and not merely simulated, speed and power. The simultaneous development of good structures of perceptual processes is the crucial, and very hard, problem. But PARALLEL(-serial) converging(-diverging) augmented pyramids appear to be friendly structures within which to develop programs that extract, combine, and abstract, in order to recognize.

14. Acknowledgements

I am grateful to the National Science Foundation for support over the years from a number of research and equipment grants.

15. References

Ahuja, N. and Swami, S., Multiprocessor pyramid architectures for bottom-up image analysis. In: *Multiresolution Image Processing and Analysis* A. Rosenfeld (Ed.), Berlin: Springer-Verlag, 1984, 38-59.

Batcher, K. E., Design of a massively parallel processor, *IEEE Trans. Computers,* 29: 836-840, 1980.

Bermond, J.-C., Delorme, C. and Quisquater, J.-J., Strategies for interconnection networks: some methods from graph theory, *Manuscript M58, Phillips Research Lab.,* Brussels, 1983.

Bruce, C. J., Desimone, R., and Gross, C. G., Visual properties of neurons in a polysensory area in the superior temporal sulcus of the macaque, *J. Neurophys.,* 46: 369-384, 1981.

Burt, P. J., The pyramid as a structure of efficient computation. In: *Multiresolution Image Processing and Analysis,* A. Rosenfeld (Ed), New York: Springer-Verlag, 1984, 6-35.

Cantoni, V., Ferretti, S., Levialdi, S. and Maloberti, F., A pyramid project using integrated technology. In: *Integrated Technology for Image Processing,* S. Levialdi (Ed.), London: Academic Press, 1985, 121-133.

Duff, M. J. B., Review of the CLIP image processing system, *Proc. AFIPS Nat. Comput. Conf.,* 1978, 1055-1060.

Dyer, C. R., Pyramid algorithms and machines. In: *Multicomputers and Image Processing,* K. Preston and L. Uhr (Eds.), New York: Academic Press, 1982, 409-420.

Gross, C. G., Rocha-Miranda, C. E., and Bender, D. B., Visual receptive fields of neurons in inferotemporal cortex of the macaque, *J. Neurophys.,* 35: 96-111, 1972.

Gross, C. G., Desimone, R., Albright, T. D., and Schwartz, E. L., Inferior temporal cortex as a visual integration area. In: *Cortical Integration,* F. Reinoso-Suarez and C. Ajmone-Marsan (Eds.), New York: Raven, 1984.

Hillis, W. D., *The Connection Machine,* Cambridge: MIT Press, 1985.

Hubel, D. H., Exploration of the primary visual cortex, 1955-78, *Nature,* 299: 515-524, 1982.

Hubel, D. H. and Wiesel, T. N., Receptive fields, binocular interaction and functional architecture in the cat's visual cortex, *J. Physiol.,* 160: 106-154, 1962.

Hubel, D. H. and Wiesel, T. N., Ferrier Lecture: Functional architecture of macaque monkey visual cortex, *Philos. Trans. R. Soc. Lond. (Biol.),* 198: 1-59, 1977.

Hwang, K. and Briggs, F. A., *Computer Architecture and Parallel Processing,* New York: McGraw-Hill, 1984.

Kent, E. W., Shneier, M. and Lumia, R., PIPE - Pipelined image processing engine, *J. Parallel and Distributed Computing,* 2: 50-78, 1985.

Krueger, J., Investigation of a small volume of neocortex with multiple microelectrodes: Evidence for principles of self-organization. In: *Complex Systems - Operational Approaches,* H. Haken (Ed.), Berlin: Springer-Verlag, 1985, 71-80.

Kuffler, S. W., Discharge patterns and functional organization of the mammalian retina, *J. Neurophys.,* 16: 37-68, 1953.

Kuffler, S. W., Nicholls, J. G. and Martin, A. R., *From Neuron to Brain,* Sunderland, Mass.: Sinauer, 1984.

Leiserson, C. E., FAT-TREES: Universal networks for hardware-efficient supercomputing, *IEEE Trans. Computers,* 34: 892-901, 1985.

Li, Z. N. and Uhr, L., Comparative Timings for a Neuron Recognition Program on Serial and Pyramid Computers, *Proc. Comp. Arch. for Pat. Anal. and Image Data Base Management,* IEEE Computer Society Press, 1985, 99-106.

Li, Z. N. and Uhr, L., A pyramidal approach for the recognition of neurons using key features, *Pattern Recognition,* 19: 55-62, 1986.

Li, Z. N. and Uhr, L., Pyramidal Algorithms for Analysis of House Images, *Systems, Man and Cybernetics,* 1987, in press.

Lougheed, R. M. and McCubbrey, D. L., Multi-processor architectures for machine vision and image analysis, *Proc. Int. Conf. Parallel Proc.,* 1985, 493-497.

Marr, D., *Vision.* San Francisco: Freeman, 1982.

Marr, D. and Hildreth, E., Theory of edge detection, *Proc. Royal Soc. London B.,* 204: 187-217, 1979.

Merigot, A., Zavidovique, B. and Devos, F., SPHINX, a pyramidal approach to parallel image processing, *Proc. Comp. Arch. for Pat. Anal. and Image Database Management,* IEEE Computer Society Press, 1985, 107-111.

Miller, R. *Pyramid Computer Algorithms,* Unpubl. Ph.D. Diss., Dept. of Math., SUNY Binghamton, 1984.

Miller, R. and Stout, Q. F., Data movement techniques for the pyramid computer, *SIAM J. Comp.*, 1987, in press.

Mishkin, M., Ungerleider, L. G., and Macko, K. A., Object vision and spatial vision: two cortical pathways, *Trends in Neurosci.*, 6: 414-417, 1983.

Perrett, D. I., Rolls, E. T., and Caan, W., Visual neurones responsive to faces in the monkey temporal cortex, *Exp. Brain Res.*, 47: 329-342, 1982.

Perrett, D. I., Smith, P. A. J., Potter, D. D., Mistlin, A. J., Head, A. S., Milner, A. D. and Jeeves, M. A., Neurones responsive to faces in temporal cortex: studies of functional organization, sensitivity to identity and relation to perception, *Human Neurobiology*, 3: 197-208, 1984.

Petersen, J., Die Theorie der regularen Graphen, *Acta Math.*, 15: 193-220, 1891.

Powell, T. P. S., Certain aspects of the intrinsic organization of the cerebral cortex. In: *Brain Mechanisms and Perceptual Awareness*, O. Pompeiano and C. Ajmone Marsan (Eds.), New York: Raven, 1981, 1-19.

Reddaway, S. F., DAP - a flexible number cruncher, *Proc. 1978 LASL Workshop on Vector and Parallel Processors*, Los Alamos, 1978, 233-234.

Riganati, J. P. and Schneck, P. B., Supercomputing, *Computer*, 17: 97-113, 1984.

Rolls, E. T., Neurons in the cortex of the temporal lobe and in the amygdala of the monkey with responses selective for faces, *Human Neurobiology*, 3: 209-222, 1984.

Rolls, E. T., Neuronal activity in relation to the recognition of stimuli in the primate. In: *Pattern Recognition Mechanisms*, C. Chagas, R. Gattas, and C. G. Gross (Ed.), Vatican City: Pontif. Acad. Sci., 1985, 203-213.

Rosenfeld, A., Pyramids: multiresolution image analysis, *Proc. Third Scandinavian Conference on Image Analysis*, July, 1983, 23-28.

Sandon, P. A., A pyramid implementation using a reconfigurable array of processors, *Proc. Comp. Arch. for Pat. Anal. and Image Data Base Management*, IEEE Computer Society Press, 1985, 112-118.

Schaefer, D. H., A pyramid of MPP processing elements - experiences and plans, *Proc. 18th Int. Conf. on System Sciences*, Honolulu, 1985.

Shipp, S. and Zeki, S., Segregation of pathways leading from area V2 to areas V4 and V5 of macaque monkey visual cortex, *Nature*, 315: 322-325, 1985.

Siegel, H. J., PASM: a reconfigurable multimicrocomputer system for image processing. In: *Languages and Architectures for Image Processing*, M. J. B. Duff and S. Levialdi

(Eds.), London: Academic Press, 1981.

Siegel, H. J., *Interconnection Networks for Large Scale Parallel Processing*. Lexington, Mass: Lexington, 1984.

Snyder, L., An inquiry into the benefits of multigauge parallel computations, *Proc. Int. Conf. Parallel Proc.*, 1985, 488-492.

Stone, J., Dreher, B., and Leventhal, A., Hierarchical and parallel mechanisms in the organization of visual cortex, *Brain Res. Revs.*, 180: 345-394, 1979.

Stout, Q. F., Sorting, merging, selecting, and filtering on tree and pyramid machines, *Proc. Int. Conf. on Parallel Processing*, 1983, 214-221.

Stout, Q. F., Algorithm guided design considerations for meshes and pyramids. In: *Intermediate-Level Image Processing*, M.J.B. Duff (Ed.), London: Academic Press, 1986, 149-165.

Tanimoto, S. L., Towards hierarchical cellular logic: design considerations for pyramid machines, *Computer Science Dept. Tech. Rept. 81-02-01*, Univ. of Washington, 1981.

Tanimoto, S. L., Algorithms for median filtering of images on a pyramid machine, In: *Computing Structures for Image Processing*, M. J. B. Duff (Ed.), London: Academic Press, 1983, 123-141.

Tanimoto, S. L., An approach to the iconic/symbolic interface. In: *Integrated Technology for Image Processing*, S. Levialdi (Ed.), London: Academic Press, 1985, 31-38.

Tanimoto, S. L. and Klinger, A., (Eds.), *Structured Computer Vision: Machine Perception Through Hierarchical Computation Structures*, New York: Academic Press, 1980.

Ts'o, D. Y., Gilbert, C. D. and Wiesel, T. N., Relationships between horizontal interactions and functional architecture in cat striate cortex as revealed by cross-correlation analysis, *J. Neurosci.*, 6: 1160-1170, 1986.

Uhr, L., Layered "recognition cone" networks that preprocess, classify, and describe. *IEEE Trans. Computers*, 21: 758-768, 1972.

Uhr, L., "Recognition Cones" that perceive and describe scenes that move and change over time. *Proc. 3rd Int. Joint Conf. on Pattern Recognition*, 1976.

Uhr, L., Pyramid Multi-Computer Structures, and Augmented Pyramids, In: *Computing Structures for Image Processing*, M. Duff (Ed.), London: Academic Press, 1983, 95-112.

Uhr, L., *Algorithm-Structured Computer Arrays and Networks*, New York: Academic Press, 1984.

Uhr, L., Augmenting pyramids and arrays by embossing them into optimal graphs to build multicomputer networks. In: *Parallel Integrated Technology for Image Processing*, S. Levialdi (Ed.), London: Academic Press, 1985, 19-31. (a)

Uhr, L., Pyramid Multi-Computers, and Extensions and Augmentations, In: *Algorithmically Specialized Parallel Computers,* L. Snyder, L. H. Jamieson, D. B. Gannon, H. J. Siegel (Eds.), New York: Academic Press, 1985, 177-186. (b)

Uhr, L., Multiple image and multi-modal augmented pyramid networks. In: *Intermediate Level Image Processing,* M. J. B. Duff (Ed.), London: Academic Press, 1986, 127-145.

Uhr, L., *Multi-Computer Architectures for Artificial Intelligence,* New York: Wiley, 1987.

Uhr, L., *Perceptual Recognition and Parallel Computer Networks,* in preparation.

Uhr, L. and Douglass, R., A parallel-serial recognition cone system for perception *Pattern Recognition,* 11: 29-40, 1979.

Uhr, L., Lackey, J. and Thompson, M., A 2-layered SIMD/MIMD Parallel Pyramidal "Array/Net", *Proc. Comp. Arch. for Pat. Anal. and Image Data Base Management,* IEEE Computer Society Press, 1981, 209-216.

Uhr, L., Schmitt, L. and Hanrahan, P., Cone/pyramid perception programs for arrays and networks. In: *Multi-Computer Algorithms and Image Processing,* K. Preston, Jr. and L. Uhr (Eds.), New York: Academic Press, 1982. pp. 180-191.

Van Essen, D. C., Functional organization of primate visual cortex. In: *Cerebral Cortex: Vol. 3. Visual Cortex,* A. Peters and E. G. Jones (Eds.), New York: Plenum, 1985, 259-329.

Weems, C. C., Levitan, S. P., Foster, C. C., Riseman, E. M., Lawton, D. T. and Hanson, A. R., Development and construction of a content addressable array processor (CAAPP) for knowledge-based image interpretation, *Proc. Workshop on Algorithm-Guided Parallel Architectures for Automatic Target Recognition,* 1984, 329-359.

Wilson, S. S., The PIXIE-5000 - a systolic array processor, *Proc. Comp. Arch. for Pat. Anal. and Image Database Management,* IEEE Computer Society Press, 1985, 477-483.

Zucker, S. W., Relaxation labeling and the reduction of local ambiguities, *Proc. Third Int. Joint Conf. on Pattern Recognition,* 1976, 852-861.