THE SEVERAL STEPS FROM ICON TO SYMBOL, USING STRUCTURED CONE/PYRAMIDS

by

Leonard Uhr
and
Larry Schmitt

Computer Sciences Technical Report #481

August 1982

The Several Steps From Icon to Symbol, Using Structured Cone/Pyramids

Leonard Uhr and Larry Schmitt

Dept. of Computer Sciences, University of Wisconsin

Abstract

Layered pyramids, cones and quad-trees have been developed almost always with emphasis on only one of their several possibilities. Either:

1) They are used to reduce information, usually by some simple function like averaging or differencing, so that successively less detailed representations of the original raw image are obtained.

2) They are used to converge, collect, organize and broadcast information, as when building successively more global histograms from which a threshold is computed and then broadcast back down the pyramid.

3) They are used to effect step-by-step transformations that interpret quantitatively encoded images into qualitative identifying "features," "compound characteristics" and other types of "labels," with sets of lower-level "meaningless" labels successively combining into higher-level "meaningful" "external labels" like 'vertical', 'Langle', 'enclosure', 'house', 'chair', 'Einstein'.

This paper explores how these different uses can be combined, with emphasis on the development of successively higher-level descriptive labellings of the scene. It indicates how "models" of objects to be recognized can be decomposed into overlapping trees of relatively simple local transforms that are embedded into the layers of a pyramid, and executed efficiently in a hardware pyramid. It examines how the successive transformation from iconic-quantitative to symbolic-qualitative can be effected (as the programmer desires) in several small steps, so that there is a gradual transition, one where the very efficiently represented iconic-structural-geometric information implicit in the 2-dimensional retinal image is kept so long as it is useful.

Index Terms: Pattern Recognition, Scene Description, Recognition Cone, Pyramid, Quad-Tree, Icon, Symbol, Multi-Resolution, Hierarchical Model-Tree

The Several Steps From Icon to Symbol, Using Structured Cone/Pyramids

Layered pyramids (e.g., Levine and Leemet, 1976, Levine, 1978; Tanimoto, 1976, 1978), cones (e.g., Hanson and Riseman, 1974, 1978a; Uhr, 1972, 1978) and quad-trees (e.g., Klinger and Dyer, 1976; Rosenfeld, 1980) have, as this paper shows, the ability to transform from iconic raw image to symbolic representations in a gradual small-step manner that throws light on the "iconic-symbolic" issue. [See also Hanson and Riseman, 1978b and Tanimoto and Klinger, 1980, for additional related systems.]

The "raw" image input to the computer (e.g., via a TV camera) is the classic example of an "iconic" representation. It consists of a 2-dimensional array of integer numbers that represent the intensity of the image at each picture element ("pixel") of the array.

This image is an "iconic" representation in several respects:

a) It has the structure, the texture, the color, and the other qualities of the objects it represents; it "looks like" them.

b) It is spread out in the two spatial dimensions of a picture, with this geometric information stored implicitly in the array structure used to store this image. Again, it "looks like" the 2-dimensional projection of the 3-dimensional world that impinges upon the sensing medium.

c) Each individual pixel cell contains numbers (usually integers) that quantitatively represent the intensity of the image at that point.

The names and descriptions of objects, which are generated at the relatively much higher levels of the perceptual system, are, in contrast, symbolic. Thus an output like 'house, tree,

auto' or '2-story red brick house with green shingled eaves; tall oak tree to the left of the front door; green convertible entering the garage' is usually considered to be "symbolic." [Note that today's vision programs are just beginning to reach the point where they can achieve the unrelated symbols 'house', 'tree', 'auto' when input a TV picture of a real-world street scene; they are far from achieving the symbolic representation needed for a full description.]

The following examines simple examples of data structures and operations (called "transforms") that process these data structures in a "recognition cone" system for perceptual recognition, starting with the iconic information in the raw sensed image as input by a TV camera and ending with the symbolic names and descriptions of objects in that image.

There are several closely related issues of interest:

A) Is the information output by (sets of) transforms iconic, symbolic, or one or another mixture of both?

B) Is the transformation from input to output iconic, symbolic, or one or another mixture of both?

C) What is the most useful combination of iconic and symbolic information, from the point of view of achieving fast and efficient perceptual recognition?

The Structure of a Multi-Layered Converging Perceptual System

"Recognition cones" (Uhr, 1972, 1974, 1976, 1978; Uhr and Douglass, 1979; Douglass, 1977, 1978; Schmitt, 1981a, 1981b) are parallel-serial structures of converging layers of transforms sandwiched between input memories into which they look and output memories into which they merge implications.

The Structure of an Individual Transform

A "transform" is a specification of:

a) a structured set of entities to search for;

b) along with a rule for determining whether this structure has been found with sufficient assurance to consider that the transform has succeeded on this image;

c) plus a set of actions to be effected contingent upon success.

d) Weights can be associated with the entities to be looked for and the actions to effect.

Figure 1. The General Structure of a "Transform"

A) Description of Entities to be Searched for:
Relation(Entity1,weight1;Entity2,weight2;...EntityN,weightN)
B) Rule for Determining Success or Failure:
Combining-Rule; Threshold
C) Actions to be Effected Contingent Upon Success:
Action(Entity, Location-Result-Stored)

For example, an edge feature-detecting transform might be coded to look for a continuing gradient of intensity values, in a 3 by 7 window, outputting 'SVE' (for "short vertical edge") when successful.

A much higher-level compound characterizer transform might look for 'LEAF's, 'BRANCH'es, 'BARK' texture, GREEN patches and one 'VTT' (for "vertical tree trunk"), outputting 'TREE' if 'VTT' and at least a few of the other features are found.

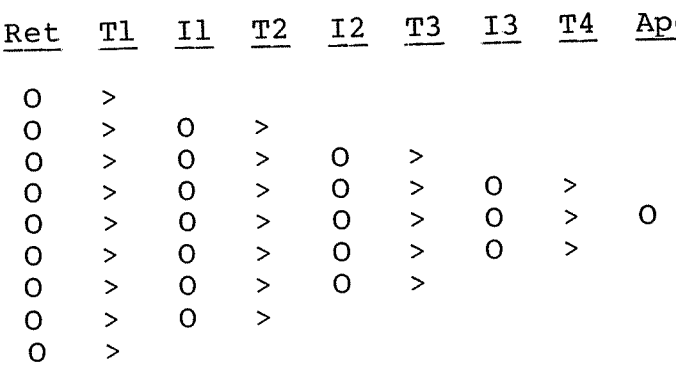
These are just two simple examples from the wide variety of transforms that can be specified for a "recognition cone" or pyramid system of the sort being examined. To make this examination completely precise and concrete, we will give examples of transforms built when using the interactive ICON system (Schmitt, 1981a), a system that prompts and helps a relatively naive user build recognition cone programs for object and scene description.

The Overall Structure of a Recognition Cone Perceptual System

A recognition cone starts with a "Retina" array, R, into which the "sensed image" is input (e.g., by a TV camera). This array forms the "base" of the cone (actually a pyramid when rectangular arrays are used).

An array of transforms, T1, is sandwiched between the Retina array and an internal buffer array, I1 (into which these transforms output their implications, that is, the transformed image). The transform array might contain 1,2, or up to several hundred transforms. Each transform is positioned at each and every cell in the array (in a parallel-serial multi-computer structure; alternately, such a system is simulated on the serial computer by iterating each transform over the entire array). Retinal transforms output their implications to array I1, which contains the first internal image.

Figure 2. The Overall Structure of a Recognition Cone



A second transform array, T2, looks at and interprets the internal image in I1, and outputs to I2. Each buffer is (usually, but not necessarily) smaller than the previous buffer (so that usually several outputs will be merged into the same cell). This gives the overall cone/pyramid shape to the structure. The converging process continues until an image buffer, IL, with only one single cell is reached. This forms the "apex" of the L-Layer

cone. Actions can also imply entities into a 'LOOKFOR' list, which results in the application of "dynamic" transforms [that gather, in a top-down manner, additional information.]

Model-Trees of Transforms Embedded into Pyramid Structures

Individual transforms can be combined within the cone/pyramid structure in any way that a user chooses. But a simple and efficient approach is to build trees of transforms that contain the "knowledge" about the different object-classes that the system must recognize dispersed among the trees' nodes.

Consider a more traditional vision program, one that tries to effect a serial match of a graph "model" of 'house' or 'auto' with a sub-graph of the image-graph into which "lower-level" processes have transformed the raw input image. Rather than try to effect this potentially explosive match (sub-graph isomorphism is NP-complete) one can decompose the single complex model-graph into a tree whose root node represents the object-class (e.g., 'house', 'chair', 'auto', 'Einstein'). Now each successively lower ply of the tree contains nodes that are transforms appropriate to that level of decomposition and detail. Thus, for example, 'house' might have 'roof', 'wall', 'porch', ... ; then (moving down the leftmost branches of the tree) 'eave', 'chimney' ... ; 'shingle' ... 'roofing', ... ; 'diagonal edge', 'pebbled', ... ; 'gradient', ... (continuing in this way to form the complete tree).

Many different trees will have nodes in common, especially at the levels nearer the raw input - the so-called "lower" levels. For example, almost all nameable objects have edges; many have horizontal or vertical edges, simple curves and identifying angles, colors and textures. Higher-level parts, like legs, en-

closures, assemblages of legs and supported horizontal planes, are also held in common by many still-higher-level objects.

All the different model-trees can be superimposed into a single tree of trees, and these embedded in a pyramid of arrays. The object-model nodes will reside at higher layers in the pyramid, their successively more detailed and local parts, sub-parts, characteristics and qualities at successively lower layers. Now rather than make a potentially explosive serial search for a match of each complex object-model stored in memory the system need only apply the transforms at each layer/ply of the tree-of-model-trees in turn.

Thus if each layer has T transforms, each with P parts (in the sense that a "part" is something that takes one instruction to examine, so that a P-part transform takes P instructions), and the pyramid has L (= log(N)) layers, with an N by N array at its base, then a recognition cone/pyramid will (when executed on a parallel hardware pyramid) take PTL (PTlog(N)) instructions (that is, time) to apply all models, plus a few more instructions to choose the best match. This can be further speeded up by placing T processors at each layer, rather than just one, to take only Plog(N) time. And if each processor is given a parallel window capability, so that it can examine all P parts in parallel, only log(N) time.

In contrast, a serial model-matching program will, at least in the worst cases, need extremely large amounts of time. Each model can be assumed to have O(PTlog(N)) nodes. That is, the number of nodes in the entire tree-of-model-trees is only a small constant times the number of nodes in each single model-tree (this is increasingly true to the extent that each node is shared

among many model-trees). But even in the extremely bizarre case where each node was used in only one model-tree each object-model has only PTlog(N)/M nodes, where M is the number of different object-models. Now M such PTlog(N) graphs must be matched as sub-graphs of the larger image graph. A variety of heuristics can be used that take advantage of facts about real-world objects to try to cut down this explosively large search; but since sub-graph isomorphism is NP-complete this is potentially an extremely long process. The pyramid serves as a global heuristic that makes use of the geometric structure of its arrays and the converging tree structure that links and organizes them together.

Hardware Multi-Computer Embodiments of Cones and Pyramids

This suggests a multi-computer pyramid structure embodied in hardware into which such a system can be embedded rather directly, one that will execute recognition cone programs with important increases in speed. In a hardware pyramid each transform node is executed in parallel by the entire array of processors in the layer at which that transform is embedded. This imposes the additional restriction that for efficiency each transform "look at," that is, input as its arguments, only locally available information. But that fits closely with the spirit of the pyramid, which decomposes very complex functions like 'Einstein' or 'house' into a tree of cascading functions where the decomposition continues until each function can be executed in minimal time with minimal hardware.

A pyramid is, essentially, a 2-dimensional N by N array of processors (like Duff's 1978 CLIP4, Reddaway's 1978 DAP and Batcher's 1980 MPP) each directly linked to its 4, 6 or 8 nearest neighbors, and also to the N^2 buds of a tree of processors. An

alternate construction would build successively smaller arrays (N/2, N/4, ... 1; or N/3, N/9, ... 1) linking each cell in the smaller array with the corresponding 4 or 9 cells in the larger array. There are a number of interesting variants in terms of details of linkages, convergence and overlap - e.g., processors might be linked only to their offspring and parents, or to siblings as well; linkage might be to only one parent, or to several parents (giving overlap). (See Dyer, 1982, Tanimoto and Pfeiffer, 1981, Tanimoto, 1982 and Uhr, 1981, 1982 for proposals and examinations of hardware pyramids.)

Such a system has the very good local interconnection topology of a grid, one that reflects well the physical forces of nature, where the closer together are several entities the more likely it is that they will interact. In addition, the tree/pyramid structure gives surprisingly good global interconnections. Essentially, the diameter of the array, which is O(N), is reduced to O(logN) in the corresponding pyramid. This is crucially important for making a global assessment and interpretation of a large object, as when combining the several parts of a transform into a larger whole. For example, when using an array alone to recognize a house one must at some point shift the various parts together in order to compute a transform like 'step', 'pane', 'window', 'wall' or 'house'. A large house might cover a region several hundred pixels in diameter; a small window might cover a region at least 8 or 16 pixels in diameter. The array would need the same number of instructions as a pyramid of arrays to actually compute T transforms. But whereas in a pyramid this need be multiplied only by log(N) (for the pyramid's processes of converging the parts together for the next-level computation) in

an array it must be multiplied by N to shift the parts together. With a 1,024 by 1,024 array the pyramid is 100 times faster.

The ICON System for Constructing Transforms and Recognition Cones

We will now look at a few transforms from a total set built with ICON (Schmitt, 1981a). The ICON program development system builds a cone/pyramid perceptual system as follows:

It asks a user to specify the number of layers in the system and the size (number of rows, number of columns) of each layer.

It also asks the user:

1) to name transforms to be placed at each layer (using names of transforms that have previously been stored in files that can be accessed by descriptors), and

2) to construct new transforms (when desired), file them away under appropriate user-designated descriptor names, and assign them to layers of the system presently being constructed.

The system helps the user construct a transform by specifying the types of information needed, displaying a 2-dimensional grid into which the transform can (to some extent iconically) be drawn, and displaying the finished transform, for verification, correction, and filing.

Thus a user can build a system of (potentially) any size, in terms of image buffers, number of layers and number of transforms at each layer.

The system can then be executed (in either interpreted or compiled form) on images input to it (either stored on disk or directly via a TV camera). The results of each layer of transforms, and of each transform at each layer, can be displayed for examination, so that transforms can be modified, discarded or added, as appropriate.

Examples of "Low Level" "Iconic" Transforms using the ICON System

One of the simplest and most iconic of operations is a straightforward averaging of the intensity values of several neighboring pixels, e.g., within a 2 by 2 window. This kind of operation is frequently used to eliminate noise, smooth the image, and/or reduce unnecessarily large images.

Figure 3 shows a 2 by 2 averaging transform, as displayed by ICON after construction has been completed.

Figure 3. A Transform that Averages a 2 by 2 Sub-Array

```
****************** TRANSFORM average     **************
THRESHOLD =           0
TYPE = numeric  FUNCTION = average

+-----------------------+-----------------------+
|     REF CELL          |          2            |
| 0          :10        | 0          :10        |
|                       |                       |
+-----------------------+-----------------------+
|          3            |          4            |
| 0          :10        | 0          :10        |
|                       |                       |
+-----------------------+-----------------------+

TRANSFORM HAS NO IMPLIEDS
```

NOTE: The transform is displayed in a 2-dimensional grid. Each cell of the grid contains (to the left of the colon) the object being looked for (in this case an integer) at that relative location, and (to the right of the colon) the weight of that object. In the case of an averaging transform, the object being looked for in each cell is a 0 (which means "don't care"). The weights, being all equal, will cause each cell to contribute equally to the transform function. This is an example of what is called, in the ICON system, an "absolute, numeric transform."

A second transformation that is often effected on the raw image is one that detects gradients. Figure 4 shows a simple (vertical) gradient detector that was constructed using ICON, as displayed by ICON.

Figure 4. A Simple Gradient-Detecting Transform

```
***************** TRANSFORM vertgrade   **************
THRESHOLD =          25
TYPE = numeric  FUNCTION = ratio
```

1	REF CELL	3
10 :10	20 :10	30 :10

NAME	IMPLIEDS WEIGHT	TYPE
horgrad	25	attribute

NOTE: This is an example of a "ratio, numeric transform." The Lookfor pattern describes a set of relative intensities to match in this transform's input image (in this case, the Retina).

Note that whereas the averaging transform shown in Figure 3 outputs an array of (averaged) numbers reflecting average intensities, and that intensities are stored in the input array, the gradient transform of Figure 4 outputs numbers reflecting a new dimension, "gradient." Here is the first aspect in which we begin to move away from the iconic:

The TV camera "transduces" the sensed image; if the transducer is of high enough quality it makes little or no change. Its result is a "good picture" almost indistinguishable from the image as directly viewed by the eye, and as "iconic" (in a common sense of the word) as it can be.

Averaging reduces the image and summarizes hence loses information, but its output is still in the same intensity domain. Since an "icon" is commonly thought of as a reduced, albeit often stylized, representation of the original, the averaged result (at least until it destroys most vestiges of the image) is usually considered among the most iconic of icons.

But when we move from the intensity domain to an array of gradient elements, things begin to change. To the extent that

one defines and thinks of an icon as a stylized representation in which the essential qualities of the object are heightened one might assert that this new representation is actually more iconic.

Examples of Simple Edge and Feature Detectors

Figures 5, 6, 7 and 8 give simple examples of the kinds of lowish-level edge and feature detectors most vision programs use. NOTE that these transforms use symbols to define their Lookfor patterns. Such "symbolic" transforms should not be interpreted as necessarily being non-iconic.

Figure 5. A Simple Vertical Edge Detector

```
****************** TRANSFORM vertedge   **************
THRESHOLD =          25
TYPE = symbolic

        ┌──────────────────┐
        │        1         │
        │vertgrade  :10    │
        │                  │
        └──────────────────┘
        ┌──────────────────┐
        │     REF CELL     │
        │vertgrade  :10    │
        │                  │
        └──────────────────┘
        ┌──────────────────┐
        │        3         │
        │vertgrade  :10    │
        │                  │
        └──────────────────┘

          IMPLIEDS
   NAME   WEIGHT    TYPE
vertedge  25     attribute
```

Figure 6. A Simple (But Slightly More General and Robust) Edge Detector

```
****************** TRANSFORM vertedge2  **************
THRESHOLD =          40
TYPE = symbolic
```

1
grad20deg :2 grad10deg :5 grad00deg :10

REF CELL
grad20deg :2 grad10deg :5 grad00deg :10

3
grad20deg :2 grad10deg :5 grad00deg :10

NAME	IMPLIEDS WEIGHT	TYPE
edge100deg	15	attribute
edge80deg	15	attribute
edge90deg	40	attribute

Figure 7. A Very Simple Detector for a Near-Perfect 'L-Angle'

```
****************** TRANSFORM Langle      **************
THRESHOLD =          40
TYPE = symbolic
```

1	2	3
edge90deg :10		

4	REF CELL	6
edge90deg :10		

7	8	9
	edge00deg :10	edge00deg :10

NAME	IMPLIEDS WEIGHT	TYPE
Langle	40	attribute

Figure 8. A Slightly More Robust Detector for an 'L-Angle'

```
***************** TRANSFORM Langle2      **************
THRESHOLD =          40
TYPE = symbolic
```

1	2	3
edge90deg :10		
4	**REF CELL**	**6**
edge90deg :10	edge00deg :5 edge90deg :5	
7	**8**	**9**
edge90deg :10 edge00deg :10	edge00deg :10	edge00deg :5

NAME	IMPLIEDS WEIGHT	TYPE
Langle	40	attribute

These are very simple examples of transforms. But they are rather interesting in the way they suggest, and can help us begin to tease out, some of the aspects of the icon-to-symbol sequence of transformations.

The edge and the gradient are closely related. The edge asserts a qualitative change at a certain point in the gradient. Both 'gradient' and 'edge' are new attributes. But somehow edge is farther away from the intensity domain. A gradient is a change in intensities; an edge is a statement that such a change makes something qualitatively different and new, the 'edge'.

It is important to emphasize that these distinctions hold only to the extent that they are subsequently made use of by the transforms that examine these implied entities as part of their input and interpret their significance. We might say that these distinctions lie in the eyes of their beholders, and that their beholders are just those transforms that interpret them.

The Langle is (as in Figure 7) simply a compound of edges or (as in Figure 8) a new label implied when enough of the expected sub-elements of an Langle are present. But the Langle label (especially in the second case) appears to be taking on more of the qualities of a symbol. There is an increasingly arbitrary relation between the label implied by the transform and merged into its output buffer and the information the transform examined and assessed in deciding whether to output that label.

One might conjecture that the iconic transformation is one where the output is computed as a function of the input, whereas the symbolic transformation is one where a table is used to store arbitrarily connected labels. But this does not seem nearly so simple as the commonly suggested analogy to a code book that arbitrarily associates and transforms between sets of symbols (e.g., between letters in English and Morse code, or between words in English and words in French). Whether the transform computes or uses table information is an implementation detail, of importance for reasons of efficiency in space for storage or time for processing, but scarcely related to the iconic/symbolic aspects of either transform or transformation.

Still, translation from English to Pig Latin suggests that some transformations can be effected by a very simple transform rule, whereas others need a code book or a dictionary. Translations from English to Pidgin English suggest that sometimes a simple transform rule must be augmented by a more or less simple mixture of code book plus additional rules. And of course from English to French we must add very deep and complex contextual syntactic and especially semantic rules to move beyond the very poor word-by-word translation.

Examples of "Higher-Level" More Symbolic Transforms

Figures 9 and 10 illustrate how transforms at deeper levels of the cone/pyramid can work with symbolic labels, sometimes in combination with numeric and more or less iconic information, to produce new (often more, sometimes less) symbolic labels.

Figure 9. A Simple 'Door' Built from Angles and Edges

```
***************** TRANSFORM door          **************
THRESHOLD =          60
TYPE = symbolic
```

1	2	3
gamma :10	edge00deg:7	revgamma :10
4	**5**	**6**
edge90deg:7 gamma :6	constarea:3	edge90deg:7 bgamma :6
7	**REF CELL**	**9**
Langle :2 edge90deg:7	constarea:3	edge90deg:7 bgamma :2
10	**11**	**12**
Langle :6 edge90deg:7	constarea:3	edge90deg:7 revLangle:6
13	**14**	**15**
Langle :10	edge00deg:7	revLangle:10

NAME	IMPLIEDS WEIGHT	TYPE
window	20	attribute
doorframe	40	attribute

Figure 10. The Implication of 'House' By Sub-Sets of Symbols & Icons

```
****************** TRANSFORM house       **************
THRESHOLD =          60
TYPE = symbolic
```

1	2	3	4	5
roof :4 chimney :8	roof :8	roof :8	roof :8	roof :4 chimney :8
6	**7**	**8**	**9**	**10**
	roof :4	roof :4	roof :4	
11	**12**	**REF CELL**	**14**	**15**
vertedge :4	window :6 vertedge :4 wall :5	window :6 wall :5	window :6 vertedge :4 wall :5	vertedge :4
16	**17**	**18**	**19**	**20**
vertedge :4	horedge :4 vertedge :4 wall :5	window :6 door :8 horedge :4 wall :5	horedge :4 vertedge :4 wall :5	vertedge :4
21	**22**	**23**	**24**	**25**
vertedge :4	Langle :8 horedge :4 vertedge :4 wall :5	door :8 horedge :4 wall :5	revLangle:8 horedge :4 vertedge :4 wall :5	vertedge :4

NAME	IMPLIEDS WEIGHT	TYPE
building	35	attribute
house	25	attribute

From Icon to Symbol; and Before, and Beyond

The transforms shown, plus all the additional transforms specified in their Lookfors, were input using ICON and built into a simple 4-layer pyramid (Figures 3 through 10 are the actual transform displays ICON generates). These transforms succeeded at finding edges, Langles, doors, roofs, houses, etc., in the very simple line drawings used to check them out. A much larger set of transforms, albeit of the same general types, must be used to make tests on a larger variety of real world images.

These examples suggest two important points:

I. There are several aspects or dimensions on which "icons" differ from "symbols."

II. It is fruitful to move gradually from the iconic raw sensed retinal image to the fully symbolic naming and description of the objects in a scene.

Indeed it appears to be preferable to consider five major stepping-stones in the transition from raw input image to completed perceptual recognition:

1) picture,

2) icon,

3) sign,

4) symbol,

5) semantic understanding.

The many-layered structure of a cone or pyramid appears to be a convenient system for controlling this gradual movement, so that the various kinds of information needed to achieve recognition are assessed, efficiently.

The original input is a (minimal) transduction; it is maximally iconic, or pictorial. It might best be called a "picture."

It reflects the 2-dimensional aspect of the array of energy being sensed, and also the quantitative amount of energy at each region in that array.

Some transformations, like simple averages, output similarly quantitative information in the same 2-dimensional intensity domain. These are usually thought of as the most iconic, or pictorial. But as averaging becomes increasingly global, so that it destroys the details needed to resolve and recognize the object, it gives less iconic results from the point of view of reflecting the objects rather than the raw image. [Note that the convergence to smaller output arrays is itself a type of averaging, as is the coarsening of the range of values over which intensity, or any other quantitative attribute, can vary.]

If a picture is input in color, even the raw sensed image is to some extent symbolic. For the primary red, green, blue colors must be labeled. This gives three arrays of intensity, instead of the single grey-scale intensity array for black-grey-white pictures. But note what this reveals: Even the raw grey-scale intensity array is to some extent symbolic, with the labelling-attribute 'grey-scale' implicit as the name of the single array. It is only because there is but one possibility, that is, only one attribute, that this need not be made explicit.

From the very beginning, images are transformed into new symbols/labels. Some of these symbols seem more iconic, e.g., 'gradient', 'purple' (the result of a transform that combines primary colors), 'hair-textured' (the result of a texture detecting transform), or even 'short-vertical-edge'. Some seem more symbolic. But can we draw a sharp line between e.g., 'S-curve' and 'Langle' and the symbols 'S', 'L', '{', '+'?

Between icons and symbols we should consider still another distinction, a "sign." And there appear to be several dimensions, each with several steps, along which entities can be placed, rather than a single path from icon to symbol.

To the extent that the symbol is more arbitrarily related to the region of information assessed by the transforms that implied it we tend to think of it as more symbolic. Thus we can consider the implied label from the point of view of the raw input image, or of the abstracted image in the layer at which it is implied, where it will be relatively more iconic.

To the extent that the symbol is "higher-level" it tends to take on a richer association of meanings, and therefore to appear more symbolic. For example, 'Langle', 'board', 'window', 'house' seem successively more symbolic. But this is simply a function of their symbolic/semantic import to us, that is, of the richness of associations and meanings they invoke in the "cognitive networks" of our minds. For the computer program they are all simply labels, or pointers. Those generated at a deeper level of the pyramid/cone will be the function of a deeper, probably larger, structure of transforms, and we might for this reason say they are indeed more "symbolic" and/or "semantic."

Toward Integrating Perceptual and Cognitive Sub-Systems

Alternately, we might consider that only after we have integrated the perceptual recognition program into a larger perceptual/cognitive system, where labels take on much richer meanings because they point to larger associated structures, can we begin to achieve the full "meaningful" "semantic" "symbol."

The typical current conception appears to be that of the "iconic" vision system and the "symbolic" semantic memory net.

This leads to the misconception that information stored in arrays, with their implicit 2-dimensional structure, and in quantitative attribute dimensions, like grey-scale-intensity or gradient-strength, is wholly iconic, whereas labels like 'dog', 'mammal', 'above' and 'is-a' are wholly symbolic. But these kinds of symbolic labels are necessary for perception programs, and are often found stored in arrays of symbols. And structural, quantitative, relational information, although rarely stored and used in today's semantic networks, would be very useful if not essential information in many situations.

It seems best, and simplest, to think of the array of spatial and other quantitative dimensions simply as a structure that can often be used in very powerful ways to access and handle quantitative information. Whereas graphs (list structures) are often convenient, e.g., when there are no simple orderings that can be computed, or when the non-zero entries in large arrays are rare (as they will be in "higher-levels" of perceptual systems that do not, as do the cones and pyramids, push them together toward the center).

Symbols like 'bright', 'very bright', 'above' and 'slightly above' might best be thought of as very coarse quantitative hence to some extent iconic intervals. A semantic memory graph might store, in addition to such symbols, little pieces of more precise quantitative information, including computational procedures, that define, explain and replace such coarse and vague concepts, as needed.

Thus it seems of interest to consider combining the more iconic aspects of the pictorial arrays used in perception programs with the more symbolic aspects of the networks of symbols

used in semantic memory systems. The perceptual array-oriented structure and the memorial graph-represented structure are each chosen (or evolved) because of their efficiencies in storage space and processing time (to transform and to access).

Symbols, and structures of symbols, can usefully be implied and stored in the perceptual system. Arrays and icons can be stored and used as nodes in the semantic memory network. The perceptual and the semantic/cognitive system must, of course, be connected. But they can both, potentially, gain greatly in power if they are intimately interconnected so that each can make use of the other, as needed.

Summary

Layered converging cones and pyramids allow a user to build pattern recognition and scene description systems whose "knowledge" of the object-classes they must identify is embedded into a hardware pyramid of arrays as follows: Each object is represented as a "model" in the form of a tree, starting with the root node that represents that object (e.g., 'house'), with the successive plies of offspring nodes representing successively lower-level parts, qualities and other characteristics (e.g., 'roof', 'wall' ... ; then 'window', 'door' ... ; then 'pane', 'panel' ... ; then 'vertical edge' ... ; 'Langle' ... ; then 'gradient' ..., and so on).

All such model-trees are combined into a tree-of-trees, where each node (e.g., 'window' or 'Langle') need be computed only once, even though it may be a part of many different model-trees (e.g., 'house', 'auto', 'store').

This tree-of-model-trees is then embedded into a pyramid of arrays, with the root model-nodes embedded into the higher layers

of the pyramid and the nodes that compute lowest-level features like gradients, short edges and local textures toward the base of the pyramid.

The pyramid architecture essentially superimposes the relatively good (diameter=O(logN)) interconnection topology of a tree over the near-neighbor topology of an array (which is especially useful for computing local interactions in images). An NxN array can speed up processing of an individual transform from O(NxN) to O(1), and a pyramid of such arrays, by eliminating the need to do O(N) shifts to move information together into the local window where successively higher-level transforms can be effected, can reduce the time needed to compute the transforms used by an entire model-tree from O(N) to O(logN).

Each transform node in this structure can look at and interpret any type of information, and output any type of information (which will then be input by the next-higher-level transform nodes). The raw input image is iconic, and therefore the lowest-level transforms must input and deal with iconic information. The identifying, descriptive information the system must output is symbolic. The movement from "icon" to "symbol" can usefully and naturally be effected (as the user chooses) in a gradual small-step manner. This paper gives examples of transforms taken from several different levels in one model-tree of a program with many model-trees that illustrate a variety of different steps in this process.

References

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

Douglass, R. J., Recognition and depth perception of objects in real world scenes, Proc. IJCAI-4, 1977, 657.

Douglass, R. J., A Computer Vision Model for Recognition, Description, and Depth Perception in Outdoor Scenes. Unpubl. Ph.D. Diss., Computer Sciences Dept., Univ. of Wisconsin, 1978.

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

Duff, M. J. B., Review of the CLIP image processing system, Proc. AFIPS NCC, 1978, 1055-1060.

Hanson, A. R. and Riseman, E. M., Pre-processing cones: a computational structure for scene analysis, COINS Tech. Rept. 74-7, Univ. of Mass., 1974.

Hanson, A. R. and Riseman, E. M., Visions: A Computer System for Interpreting Scenes, In Computer Vision Systems, A. R. Hanson and E. M. Riseman (Eds.), New York: Academic Press, 1978, 303-334.(a)

Hanson, A.R. and Riseman, E.M. (Eds.), Computer Vision Systems, New York: Academic Press, 1978.(b)

Klinger, A. and Dyer, C.R., Experiments on picture representations using regular decomposition, Comput. Graphics and Image Processing, 1976, 5, 68-105.

Levine, M. D., A knowledge-based computer vision system, In: Computer Vision Systems, A.R. Hanson and E.M. Riseman (Eds.), New York: Academic Press, 1978, 335-352.

Levine, M.D. and Leemet, J., A method for nonpurposive picture segmentation, Proc. 3d Int. Joint Conf. on Pattern Recog., 1976, 494-498.

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

Rosenfeld, A., Quadtrees and pyramids for pattern recognition and image processing, Proc. Fifth Int. Conf. on Pattern Recognition, 1980, 802-807.

Schmitt, L., The ICON perception laboratory user manual and reference guide, Computer Sciences Dept. Tech. Rept. 421, 1981.(a)

Schmitt, L., The use of a network representation of visual knowledge in a hierarchically structured vision system, Computer Sciences Dept. Tech. Rept., 1981.(b)

Tanimoto, S. L., Pictorial feature distortion in a pyramid, Computer Graphics Image Proc., 1976, 5, 333-352.

Tanimoto, S. L., Regular Hierarchical Image and Processing Structures in Machine Vision, In: Computer Vision Systems, A.R. Hanson and E.M. Riseman (Eds.), New York: Academic Press, 1978, 165-174.

Tanimoto, S.L., Programming techniques for hierarchical parallel image processors, In: Multi-Computers and Image Processing, K. Preston, Jr. and L. Uhr (Eds.) New York: Academic Press, 1982. pp. 421-429.

Tanimoto, S.L. and Klinger, A. (Eds.), Structured Computer Vision, New York: Academic Press, 1980.

Tanimoto, S.L. and Pfeiffer, J.J., Jr., An image processor based on an array of pipelines, Proc. Workshop on Computer Architec-

ture for Pattern Analysis and Image Data Base Management, IEEE Computer Society Press, 1981, 201-208.

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

Uhr, L., A model of form perception and scene description, Computer Sciences Dept. Tech. Rept. 176, Univ. of Wisconsin, 1974.

Uhr, L., "Recognition cones" that perceive and describe scenes that move and change over time, Proc. Int. Joint Conf. on Pattern Recog., 1976, 4, 287-293.

Uhr, L., "Recognition cones" and some test results. In: Computer Vision Systems A.R. Hanson and E.M. Riseman (Eds.), New York: Academic Press, 1978, 363-372.

Uhr, L., Converging pyramids of arrays, Proc. Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management, IEEE Computer Society Press, 1981, 31-34.

Uhr, L. Algorithm-Structured Computer Arrays and Networks: Parallel Architectures for Perception and Modelling, New York: Academic Press, in press.

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