JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 11, 1-24 (1991)

INVITED PAPER

The DARPA Image Understanding Benchmark for Parallel Computers*

CHARLES WEEMS, EDWARD RISEMAN, AND ALLEN HANSON

Computer and Information Science Department, University of Massachusetts at Amherst, Amherst, Massachusetts 01003

AND

AZRIEL ROSENFELD

Center for Automation Research, University of Maryland at College Park, College Park, Maryland 20742

This paper describes a new effort to evaluate parallel architectures applied to knowledge-based machine vision. Previous vision benchmarks have considered only execution times for isolated vision-related tasks, or a very simple image processing scenario. However, the performance of an image interpretation system depends upon a wide range of operations on different levels of representations, from processing arrays of pixels, through manipulation of extracted image events, to symbolic processing of stored models. Vision is also characterized by both bottom-up (image-based) and top-down (model-directed) processing. Thus, the costs of interactions between tasks, input and output, and system overhead must be taken into consideration. Therefore, this new benchmark addresses the issue of system performance on an integrated set of tasks. The Integrated Image Understanding Benchmark consists of a model-based object recognition problem, given two sources of sensory input, intensity and range data, and a database of candidate models. The models consist of configurations of rectangular surfaces, floating in space, viewed under orthographic projection, with the presence of both noise and spurious nonmodel surfaces. A partially ordered sequence of operations that solves the problem is specified along with a recommended algorithmic method for each step. In addition to reporting the total time and the final solution, timings are requested for each component operation, and intermediate results are output as a check on accuracy. Other factors such as programming time, language, code size, and machine configurations are reported. As a result, the benchmark can be used to gain insight into processor strengths and weaknesses and may thus help to guide the development of the next generation of parallel vision architectures. In addition to discussing the development and specification of the new benchmark, this paper presents the results from running the benchmark on the Connection Machine, Warp, Image Understanding Architecture, Associative String Processor, Alliant FX-80, and Sequent Symmetry. The results are discussed and compared through a measurement of relative effort, which factors out the effects of differing technologies.

INTRODUCTION

Knowledge-based image understanding presents an immense computational challenge that has yet to be satisfied by any parallel architecture. The challenge is not merely to provide a greater quantity of operations per second, but also to supply the necessary forms of computation, communication, and control. Consider that a sequence of images at medium resolution (512 × 512 pixels) and standard frame rate (30 frames per second) in color (24 bits per pixel) represents a data input rate of about 23.6 million bytes per second and, in a typical interpretation scenario, many thousands of operations may be applied to each input pixel in order to enhance and segment an image and to extract various features from it. But in addition, a vision system must organize extracted image features via perceptual grouping mechanisms, locate relevant models in a potentially vast store of knowledge and compare them to partial models derived from the input data, generate hypotheses concerning the environment of the sensor, resolve conflicting hypotheses to arrive at a consistent interpretation of the environment, manage and update stored knowledge, etc.

While traditional supercomputing benchmarks may be useful in estimating the performance of an architecture on some types of image processing tasks, as noted by Duff [3], they have little relevance to the majority of the processing that takes place in a vision system. Nor has there been much effort to define a vision benchmark for supercomputers, since those machines in their traditional form have usually been viewed as inappropriate vehicles for knowledge-based vision research. However, now that parallel processors are becoming readily available, and because they are viewed as being better suited to vision processing, researchers in both machine vi-

* This work was supported in part by the Defense Advanced Research Projects Agency under Contract DACA76-86-C-0015, monitored by the U.S. Army Engineer Topographic Laboratories.

0743-7315/91 $3.00

sion and parallel architecture are taking an interest in performance issues with respect to vision. The next section summarizes the other prominent work that has been done in the area of vision benchmarks to date.

REVIEW OF PREVIOUS VISION BENCHMARK EFFORTS

One of the first parallel processor benchmarks to address vision-related processing was the Abingdon Cross benchmark, defined at the 1982 Multicomputer Workshop in Abingdon, England. In that benchmark, an input image consists of a dark background with a pair of brighter rectangular bars, equal in size, that cross at their midpoints and are centered in the image. Gaussian noise is added to the entire image. The goal of the exercise is to determine and draw the medial axis of the cross formed by the two bars. The results obtained from solving the benchmark problem on various machines were presented by Preston [7, 8] at the 1984 Multicomputer Workshop in Tanque Verde, Arizona, and many of the participants spent a fairly lengthy session discussing problems with the benchmark and designing a new benchmark that, it was hoped, would solve those problems.

One concern of the Tanque Verde group was that the Abingdon Cross lacks breadth, requiring a reasonably small repertoire of image processing operations to construct a solution. The second concern was that the specification did not constrain the a priori information that could be used to solve the problem. In theory, a valid solution is to simply draw the medial lines since their true positions are known. Although this was never done, there was argument over whether it was acceptable for a solution to make use of the fact that the bars were oriented horizontally and vertically in the image. A final concern was that no method was prescribed for solving the problem, with the result that every solution was based on a different method. When a benchmark can be solved in different ways, the performance measurements are difficult to compare because they include an element of programmer cleverness. Also, the use of a consistent method would permit some comparison of the basic operations that make up a complete solution.

The Tanque Verde group specified a new benchmark, called the Tanque Verde Suite, that consisted of a large collection of individual vision-related problems (Table I). Each of the problems was to be further defined by a member of the group, who would also generate test data for his or her assigned problem. Unfortunately, only a few of the problems were ever developed, and none of them were widely tested on different architectures. Thus, while the simplicity of the Abingdon Cross may have been criticized, it was the respondent complexity of the Tanque Verde Suite that inhibited the latter's use.

In 1986, a new benchmark was developed at the request of the Defense Advanced Research Projects Agency (DARPA). Like the Tanque Verde Suite, it was a collection of vision-related problems, but was much smaller and easier to implement (Table II). A workshop was held in Washington, D.C., in November of 1986 to present the results of testing the benchmark on several machines, with those results summarized by Rosenfeld in [9]. The consensus of participants was that the results cannot be compared directly for several reasons. First, as with the Abingdon Cross, no method was specified for solving any of the problems. Thus, in many cases, the timings were more indicative of the knowledge or cleverness of the programmer than of a machine's true capabilities. Second, no input data were provided and the specifications allowed a wide range of possible inputs. Thus, some participants chose to test a worst-case input, while others chose "average" input values that varied considerably in difficulty.

The workshop participants pointed out other shortcomings of the benchmark. Chief among these was that it consisted of isolated tasks and therefore did not measure performance related to the interactions between the components of a vision system. For example, there might be a particularly fast solution to a problem on a given architecture if the input data is arranged in a special manner. However, this apparent advantage might be inconsequential if a vision system does not normally use the data in such an arrangement and the cost of rearranging the data is high. Another shortcoming was that the problems had not been solved before they were distributed. Thus, there was no canonical solution on which the participants could rely for a definition of correctness, and there was even one problem (graph isomorphism) for which it turned out that there was no practical solution.

Having a known correct solution is essential, since it is difficult to compare the performance of architectures that produce different results. For example, suppose architecture

TABLE I

Tanque Verge Benchmark Suite

Standard utilities	High-level tasks
3 × 3 Separable convolution	Edge finding
3 × 3 General convolution	Line finding
15 × 15 Separable convolution	Corner finding
15 × 15 General convolution	Noise removal
Affine transform	Generalized Abingdon Cross
Discrete Fourier transform	Segmentation
3 × 3 median filter	Line parameter extraction
256 Bin histogram	Deblurring
Subtract two images	Classification
Arctangent (image1/image2)	Printed circuit inspection
Hough transform	Stereo image matching
Euclidean distance transform	Camera motion estimation
	Shape identification

TABLE II

Tasks from the First DARPA Image Understanding Benchmark

11 × 11 Gaussian convolution of a 512 × 512 eight-bit image
Detection of zero crossings in a difference of Gaussians image
Construct and output border pixel list
Label connected components in a binary image
Hough transform of a binary image
Convex hull of 1000 points in 2D real space
Voronoi diagram of 1000 points in 2D real space
Minimal spanning tree across 1000 points in 2D real space
Visibility of vertices for 1000 triangles in 3D real space
Minimum cost path through a weighted graph of 1000 nodes of order 100
Find all isomorphisms of a 100-node graph in a 1000-node graph

A performs a task in half the time of B, but A uses integer arithmetic while B uses floating point, and they obtain different results. Is A really twice as powerful as B? Since problems in vision are often ill-defined, it is possible to argue for the correctness of many different solutions. In a benchmark, however, the goal is not to solve a vision problem but rather to test the performance of different machines doing comparable work.

The conclusion from this first DARPA exercise was that a new benchmark should be developed. Specifically, the new benchmark should test system performance on a task that approximates an integrated solution to a machine vision problem. A complete solution with test data sets should be constructed and distributed with the benchmark specification. And, the benchmark should be specified to minimize opportunities for taking shortcuts in solving the problem. The task of constructing the new benchmark was assigned to the vision research groups at the University of Massachusetts at Amherst and the University of Maryland.

A preliminary specification was drawn up and circulated among the DARPA image understanding community for comment. The specification was revised and a solution programmed on a standard sequential machine. In creating the solution, several problems that required corrections to the specification were discovered. The solution was programmed by the University of Massachusetts group, and the University of Maryland group then verified its validity, portability, and quality. Maryland also reviewed the solution for generality and neutrality with respect to underlying architectural assumptions. The Massachusetts group developed five test data sets and a sample parallel solution for a commercial multiprocessor (the Sequent Symmetry 81).

In March of 1988, the benchmark was made available from Maryland via network access, or on tape from Massachusetts. The benchmark release consisted of the sequential and parallel solutions, the five data sets, and software for generating additional test data. The benchmark specification was presented by Weems at the DARPA Image Understanding Workshop, the International Supercomputing Conference, and the Computer Vision and Pattern Recognition Conference [11–13]. Over 25 academic and industrial groups (Table III) obtained copies of the benchmark release. Nine of those groups developed either complete or partial versions of the solution for an architecture. A workshop was held in October of 1988, in Avon Old Farms, Connecticut, to present these results to members of the DARPA research community. As in the previous workshops, the participants spent a session developing a critique of the benchmark and making recommendations for the design of the next version.

The remainder of this paper begins with a brief review of the benchmark task and the rationale behind its design; then it summarizes results that were based on hardware execution or on instruction-level simulation of the benchmark. Myung Sunwoo [10] from the University of Texas at Austin and Alok Choudhary [2] from the University of Illinois also presented estimated results for proposed architectures, which are not included here. Also not included are timings from Active Memory Technology on its DAP array processor that are for a set of independent image processing tasks only somewhat related to the benchmark problem. Finally, the paper presents the criticisms raised at the workshop, along with recommendations for addressing them.

BENCHMARK TASK OVERVIEW

The overall task that is to be performed by this benchmark is the recognition of an approximately specified $2\frac{1}{2}$-dimensional "mobile" sculpture in a cluttered environment, given images from intensity and range sensors. The intention of the benchmark designers is that neither of the input images, by itself, should be sufficient to complete the task.

The sculpture to be recognized is a collection of 2-dimensional rectangles of various sizes, brightnesses, 2-dimensional

TABLE III

Distribution List for the Second DARPA Benchmark

International Parallel Machines	Hughes AI Center
Mercury Computer Systems	University of Wisconsin
Stellar Computer	George Washington University
Myrias Computer	University of Massachusetts[a]
Active Memory Technology	SAIC
Thinking Machines[a]	Eastman–Kodak
Aspex Ltd.[a]	University College London
Texas Instruments	Encore Computer
IBM	MIT
Carnegie–Mellon University[a]	University of Rochester
Intel Scientific Computers[a]	University of Illinois[a]
Cray Research	University of Texas at Austin[a]
Sequent Computer Systems[a]	Alliant Computer[a]

[a] Results presented at the Avon Workshop.

orientations, and depths. Each rectangle is oriented normal to the Z axis (the viewing axis), with constant depth across its surface, and the images are constructed under orthographic projection. Thus an individual rectangle has no intrinsic depth component, but depth is a factor in the spatial relationships between rectangles—hence the notion that the sculpture is $2\frac{1}{2}$-dimensional.

The clutter in the scene consists of additional rectangles, with sizes, brightnesses, 2-dimensional orientations, and depths that are similar to those of the sculpture. Rectangles may partially or completely occlude other rectangles. It is also possible for a rectangle to disappear when another of the same brightness or slightly greater depth (such that the difference in depth is less than the noise threshold) is located directly behind it.

A set of models representing a collection of similar sculptures is provided, and the task is to identify which model best matches the scene. The models are only approximate representations in that they permit variations in the sizes, orientations, depths, and spatial relationships between the component rectangles. A model is a tree structure, where the links represent the invisible links in the sculpture. Each node of the tree contains depth, size, orientation, and intensity information for a single rectangle. The child links of a node describe the spatial relationships between it and nodes below.

The scenario that was imagined in constructing the problem was a semirigid mobile, with invisible links, viewed from above, with portions of other mobiles blowing through the scene. The initial state is that previous processing has narrowed the range of potential matches to a few similar sculptures and has oriented them to match a previous image. However, the objects have since moved, and new images have been taken prior to this final step. The system must choose the best match and update the corresponding model with the positional information extracted.

The intensity and depth sensors are precisely registered with each other and both have a resolution of 512×512 pixels. There is no averaging or aliasing in either of the sensors. A pixel in the intensity image is an 8-bit integer grey value. In the depth image a pixel is a 32-bit floating-point range value. The intensity image is noise free, while the depth image has added Gaussian noise. The reason that only one of the images is noisy is that adding noise to the other image simply requires more of the same sorts of processing to be performed, and one goal of the benchmark designers was to maximize the variety of processing while minimizing programmer effort.

A pair of artificial test images is created by first selecting one model. The model is then rotated and translated as a whole, and its individual elements are perturbed slightly. Next, a collection of spurious rectangles with properties similar to those in the chosen model is created. All of the rectangles (both model and spurious) are then ordered by depth and drawn in the two image arrays. Lastly, an array of Gaussian-distributed noise is added to the depth image.

Figure 1 shows an intensity image of a mobile alone, and Figure 2 shows the mobile with added clutter. Depth images are not shown, because their floating-point representation makes them difficult to display accurately.

Processing begins with low-level operations on the intensity and depth images, followed by grouping operations on the intensity data to extract candidate rectangles. The candidates are used to form partial matches with the stored models. For each model, it is possible that multiple hypothetical poses will be established. For each model pose, stored information is used to probe the depth and intensity images in a top-down manner. Each probe tests a hypothesis for the

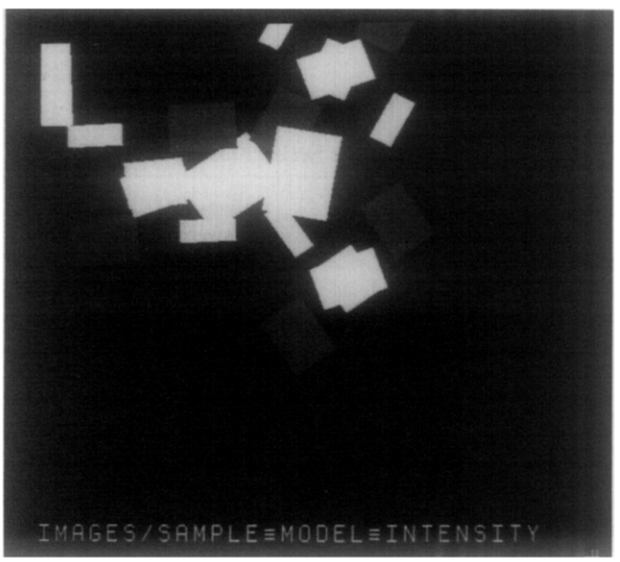

FIG. 1. Intensity image of Sample model alone.

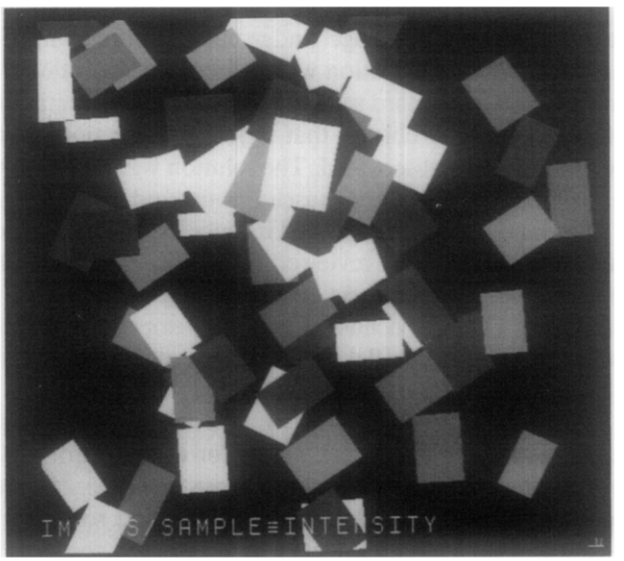

FIG. 2. Image of Sample model with clutter.

existence of a rectangle in a given location in the images. Rejection of a hypothesis, which only occurs when there is strong evidence that a rectangle is actually absent, results in the elimination of the corresponding model pose. Confirmation of the hypothesis results in the computation of a match strength for the rectangle and an update of its representation in the model pose with new size, orientation, and position information. The match strength is zero when there is no supporting evidence for the match and no evidence that the rectangle is absent, as in the case of a rectangle that is entirely occluded by another. After a probe has been performed for every unmatched rectangle in the list of model poses, an average match strength is computed for each pose that has not been eliminated. The model pose with the highest average is selected as the best match, and an image that highlights the model in the intensity image is generated. Table IV lists the steps that make up the complete benchmark task.

The benchmark specification requires these steps to be applied in implementing a solution. Furthermore, a recommended method for each step is described and should be followed if possible. However, in recognition that some methods do not work, or are extremely inefficient for a given parallel architecture, implementors are permitted to substitute other methods for individual steps. When it is necessary to differ from the specification, the implementor should supply a justification for the change. It is also urged that, if possible, a version of the implementation be written and tested with the recommended method so that the difference in performance can be determined.

Benchmark Philosophy and Rationale

In writing an integrated image understanding benchmark, the goal is to create a scenario that is an approximation of an actual image interpretation task. One must remember, however, that the benchmark problem is not an end in itself, but rather a framework for testing machine performance on a wide variety of common vision operations and algorithms, both individually and in an integrated form that requires communication and control across algorithms and representations. This benchmark is not intended to be a challenging vision research exercise, and the designers feel that it should not be. Instead, it should be a balance between simplicity for the sake of implementation by participants

TABLE IV

Steps that Compose the Integrated Image Understanding Benchmark

Low-level, bottom-up processing	
Intensity image	Depth image
Label connected components	3×3 Median filter[a]
Compute K-curvature	3×3 Sobel and gradient magnitude[a]
Extract corners	Threshold[a]

TABLE IV—*Continued*

Intermediate-level processing

Select components with three or more corners
Convex hull of corners for each component
Compute angles between successive corners on convex hulls
Select corners with K-curvature and computed angles indicating a right angle
Label components with three contiguous right angles as candidate rectangles
Compute size, orientation, position, and intensity of each candidate rectangle

Model-based, top-down processing

Determine all single-node isomorphisms of candidate rectangles in stored models
Create a list of all potential model poses
Perform a match-strength probe for all single-node isomorphisms (see below)[a]
Link together all single-node isomorphisms
Create a list of all probes required to extend each partial match
Order the probe list according to the match strength of the partial match being extended
Perform a probe of the depth data for each probe on the list (see below)
Perform a match-strength probe for each confirming depth probe (see below)[a]
Update rectangle parameters in the stored model for each confirming probe[a]
Propagate the veto from a rejecting depth probe throughout the corresponding partial match
When no probes remain, compute the average match strength for each remaining model pose
Select the model with the highest average match strength as the best match
Create the output intensity image, showing the matching model

Depth probe

Select an X–Y-oriented window in the depth data that will contain the rectangle
Perform a Hough transform within the window
Search the Hough array for strong edges with the approximate expected orientations
If fewer than three edges are found, return the original model data with a no-match flag
If three edges are found, infer the fourth from the model data
Compute new size, position, and orientation values for the rectangle

Match-strength probe

Select an oriented window in the depth data that is slightly larger than the rectangle
Classify depth pixels as too close, too far, or in range[a]
If the number of too far pixels exceeds a threshold, return a veto
Otherwise, select a corresponding window in the intensity image
Select intensity pixels with the correct value
Compute a match strength on the basis of the number of correct vs incorrect pixels in the images

[a] Subtasks involving floating-point operations.

and the complexity that is representative of actual vision processing. At the same time, it must test machine performance in as many ways as possible. A further constraint on the design was the requirement that it reuse tasks from the first DARPA benchmark where possible, in order to take advantage of the previous programming effort. The great variety of architectures to be tested is itself a complicating factor in the design of a benchmark. It was recognized that each architecture may have its own most efficient method for computing a given function.

The job of the designers was thus to balance these conflicting goals and constraints in developing the benchmark. One result is that the solution is neither the most direct nor the most efficient method. However, a direct solution would eliminate several algorithms that are important in testing certain aspects of machine performance. On the other hand, increasing the complexity of the problem to necessitate the use of those algorithms would require significant additional processing that is redundant in terms of performance evaluation. Thus, while the benchmark solution is not a good example of how to build an efficient vision system, it is an effective test of machine performance both on a wide variety of individual operations and on an integrated task. Participants were encouraged to develop timings for more optimal solutions, in addition to the standard solution, if they so desired.

The designers also recognize the tendency for any benchmark to turn into a horse race. However, that is not the goal of this exercise, which is to increase the scientific insight of architects and vision researchers into the architectural requirements for knowledge-based image interpretation. To this end, the benchmark requires an extensive set of instrumentation. Participants are required to report execution times for individual tasks, for the entire task, for system overhead, for input and output, for system initialization and loading any precomputed data, and for different processor configurations if possible. Implementation factors to be reported include an estimate of time spent implementing the benchmark, the number of lines of source code, the programming language, and the size of the object code. Machine configuration and technology factors that are requested include the number of processors, memory capacity, data path widths, integration technology, clock and instruction rates, power consumption, physical size and weight, cost, and any limits to scaling up the architecture. Lastly, participants are asked to comment on any changes to the architecture that they feel would contribute to an improvement in performance on the benchmark.

RESULTS AND ANALYSIS

Due to limitations of time and resources, only a few of the original participants were able to complete the entire benchmark exercise and test it on all five of the data sets. In almost every case, there was some disclaimer to the effect that a particular architecture could have shown better performance given more implementation time or resources. It was common for participants to underestimate the effort required to implement the benchmark, and several who had said they would provide timings were unable to complete even a portion of the task prior to the workshop.

Caution in Comparing Results

Care must be taken in comparing these results. For example, no direct comparison should be made between results obtained from actual execution and those derived from simulation, as noted by Carpenter [1]. No matter how carefully a simulation is carried out, it is never as accurate as direct execution. Likewise, no comparison should be made between results from partial and complete implementations. A complete implementation includes overhead for the interactions between subtasks and for the fact that the program is significantly larger than a partial implementation. Consider that individual subtasks might be faster than a complete implementation simply because less paging is required. It is also unwise to directly compare raw timings, even for similar architectures, without considering the differences in technology between systems. For example, a system that executes the benchmark faster than another is not necessarily architecturally superior if it also has a faster clock rate or more processors.

In addition to technical problems in making direct comparisons, there are other considerations to keep in mind. For example, what is impressive in many cases is not the raw speed obtained, but rather the speed with respect to the effort required to obtain it. While this has more to do with the software tools available for an architecture, it is still important in evaluating the overall usefulness of a system. Another consideration is the ratio of cost to performance. In addition, the size, weight, or power consumption may be of greater importance than all-out speed in some applications. Finally, each vision application has a different mix of bottom-up and top-down processing, which is unlikely to match the mix used in the benchmark. Thus, readers should not focus on the total time, but may find it more useful to combine timings for the subtasks to approximate the processing mix in some familiar application. One of the purposes of this exercise is merely to assemble as much data as possible so that the performance results can be evaluated with respect to the requirements of each potential application.

The Data Sets

Five data sets were distributed with the benchmark, having the unimaginative names of Sample, Test, Test2, Test3, and Test4. The Sample data set required the greatest processing time on the sequential processors and was in some ways the most complex. It had the greatest density of model elements (both large and numerous) and enough similarity between

models to require a significant amount of top-down processing to determine the best match. Sample also required that a 5×5 median filter be used, rather than the 3×3 that was specified for the other data sets (this was intended to necessitate a certain level of generality in the median filter routine and also to see if an architecture had special hardware for 3×3 window operators that might not extend to larger window sizes).

While Sample was intended to represent a processing balance between the bottom-up and top-down portions, the remaining data sets were somewhat biased toward one or the other of those portions. The Test and Test2 data sets deemphasized the top-down processing by having only one model that fit the images. It was possible to quickly reject all of the other models and simply use the top-down probes to determine the new positional information for the one remaining model. The Test3 and Test4 data sets emphasized the top-down portion by presenting several models that were nearly identical and which had considerable symmetry so that numerous poses would be hypothesized. Thus, there were several models that could not be eliminated, and a far greater number of top-down probes were required to determine the best match.

Figures 1 and 2 show the intensity images for the Sample model and input image data. Figures 3 and 4 show the model and intensity images for the Test data set (which is similar to Test2), and Figs. 5 and 6 show the Test3 data set (which is similar to Test4).

Reporting Conventions

To set the context for the results, we first describe each of the implementations. Results based on theoretical estimations are not included here. Because of the variation in the actual timings and the implementation information that was supplied, the data have been rearranged in a standard format, which specifies the timings only for the major subtasks. All timings have been scaled to seconds, even though for some of the processors they would be more readable if presented in milliseconds or minutes. A detailed presentation of all of the data for the minor subtasks can be found in [15]. The details of the architectures can be found in the appropriate references. Physical and cost data for the commercial systems are subject to change and should be obtained directly from the manufacturers.

In addition to the timings, the specification requested that a set of intermediate results be output to help verify that the subtasks were performing comparable operations. For example, the number of connected components in the intensity image and the number of probes performed were among the requested validation results. It was not possible for every system to generate all of these data, but whatever validation results were provided are included here.

From an architectural point of view, one useful measure is the percentage of time devoted to each subtask, which indicates the subtask's relative difficulty for a particular architecture. It also factors out all of the technological issues and provides one of the few measures that can be directly compared across architectures. For each implementation, we present a bar chart, showing the percentage of time spent on the major subtasks, for each data set. In several cases, where data were available for different machine configurations, the configurations are also compared. The different architectures are compared in a later section.

Sequential Solution

The sequential implementation was developed in C on a Sun-3/160 workstation and contains roughly 4600 lines of

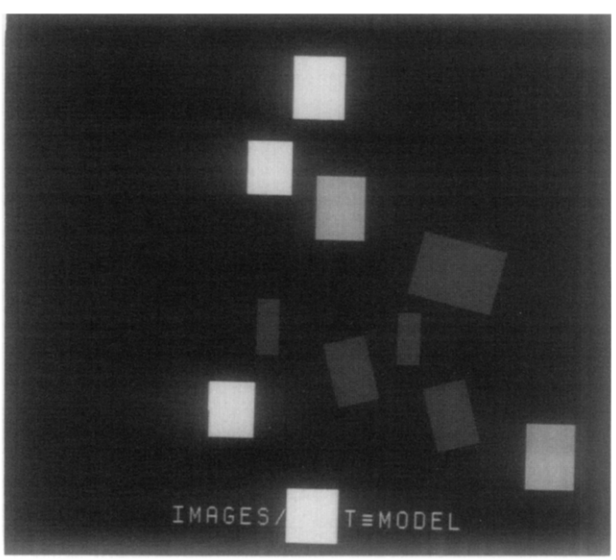

FIG. 3. Intensity image of model Test.

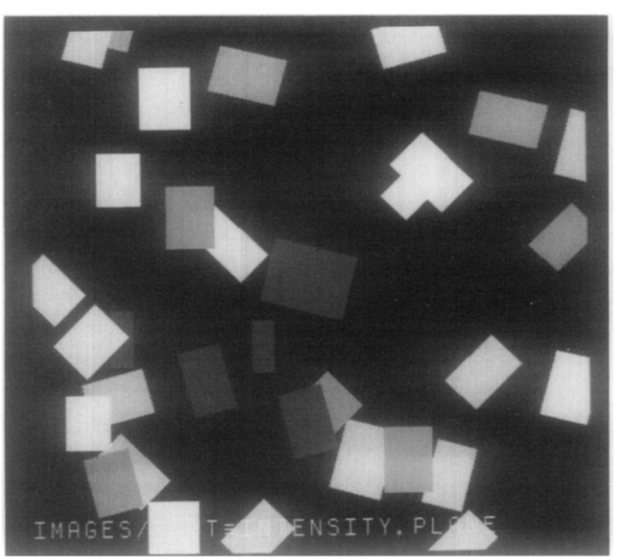

FIG. 4. Image of model Test with clutter.

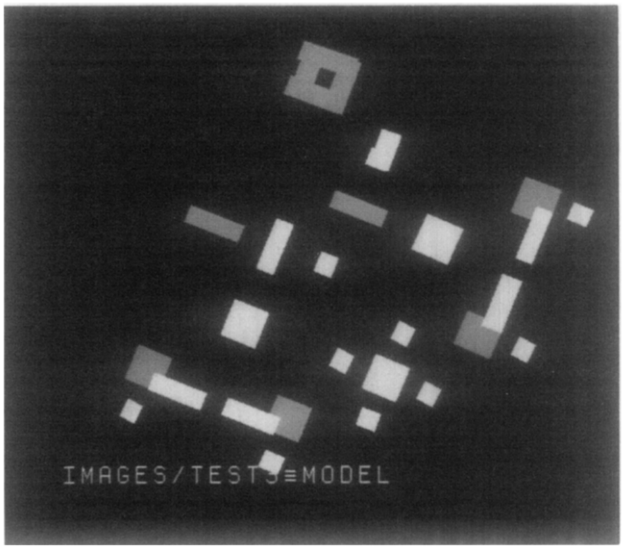

FIG. 5. Intensity image of model Test3.

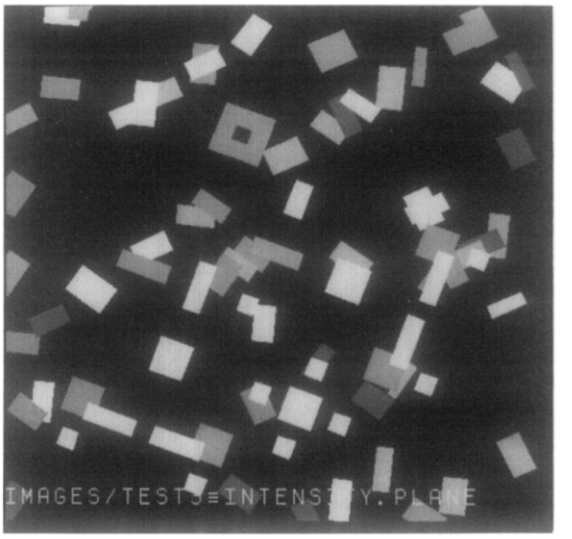

FIG. 6. Image of model Test3 with clutter.

code, including comments. It was designed for portability and has been recompiled on several different systems. The only system-dependent portion is the result presentation step, which uses the workstation's graphics display. The implementation differs from the recommended method on the Connected Component Labelling step in that it uses a standard sequential method for this well-defined function. The sequential method minimizes array accesses and the corresponding index calculations, which incur an avoidable time penalty on a sequential machine.

Timings have been produced for all five data sets and on three different machine configurations: a Sun-3/160 (a 16-MHz 68020 processor) with 8 Mbytes of RAM, a Sun-3/260 (a 25-MHz 68020) with 16 Mbytes of RAM, and a Sun-4/260 (a 16-MHz SPARC processor) with 16 Mbytes of RAM. The extra RAM on the latter two machines did not affect performance, since the benchmark runs in 8 Mbytes without paging. The 3/260 was equipped with a Weitek floating-point coprocessor, while the 3/160 and 4/260 used only their standard coprocessors. Tables V, VI, and VIII show the execution times for the Sun-3/160, Sun-3/260, and Sun-4/260, respectively. Tables VII and IX show the validation data that were output by the Sun-3 and the Sun-4 systems, respectively. Note the slight variation in the validation data, due to minor differences in the floating-point results. These variations are within the tolerances of the benchmark, and the final result is the same. The timings were obtained with the system clock utility, which has a resolution of 20 ms on the Sun-3 systems and 10 ms on the Sun-4.

Figure 7 compares the three configurations on each of the data sets with regard to the percentage of time spent on each major subtask. The key identifies the pattern associated with each major subtask. Note that the bottom-up portions are represented by shading, while the top-down portions are shown by cross-hatch patterns. The figure shows that Test and Test2 require far less top-down processing than the other three data sets. Closer examination reveals several interesting points. For example, despite the faster Weitek coprocessor, the Sun-3/260 spends proportionately more time than the Sun-3/160 in the median filter, which involves floating-point data. It is also interesting to note that the Sun-4 spends a larger percentage of time on the top-down tasks (especially the Hough probes) and overhead. Since the overhead depends mostly on disk access time, time spent on it should be expected to increase in percentage as the total time decreases.

Alliant FX-80 Solution

The Alliant FX-80 consists of up to 8 computational elements and 12 I/O processors that share a physical memory

TABLE V

Sun-3/160 Times for Major Subtasks

Data set	Sample	Test	Test2	Test3	Test4
Total	797.88	338.06	329.24	551.82	553.16
Overhead	5.08	4.94	5.64	5.64	5.52
Label connected components	27.78	27.82	28.40	28.22	28.24
Rectangles from intensity	6.50	4.14	4.38	5.44	5.34
Median filter	246.66	118.88	92.86	90.92	90.90
Sobel	135.48	133.30	136.10	135.28	135.42
Initial graph match	24.46	25.00	26.04	68.44	67.62
Match-strength probes	72.98	3.28	5.86	47.88	42.06
Hough probes	253.70	8.28	12.96	153.98	162.34
Result presentation	24.80	12.32	16.66	14.78	14.76

TABLE VI

Sun-3/260 Times for Major Subtasks

Data set	Sample	Test	Test2	Test3	Test4
Total	299.38	132.54	119.52	194.76	195.58
Overhead	2.92	3.04	3.44	3.44	3.44
Label connected components	14.52	14.46	14.46	14.58	14.66
Rectangles from intensity	3.74	2.38	2.48	3.16	2.98
Median filter	113.70	60.28	43.10	42.98	43.26
Sobel	41.00	38.50	38.34	38.76	38.56
Initial graph match	6.16	6.08	6.58	17.32	16.94
Match-strength probes	17.56	0.78	1.40	11.66	10.30
Hough probes	92.70	3.12	4.82	57.96	60.44
Result presentation	6.68	3.64	4.72	4.50	4.30

through a sophisticated combination of caches, buses, and an interconnection network. The computational elements communicate with the shared memory via the interconnection network, which links them to a pair of special-purpose caches that in turn access the memory over a bus that is shared with the I/O processor caches.

Alliant was able to implement the benchmark on the FX-80 in roughly 1 programmer week. The programmer had no experience in vision and, in many cases, did not bother to learn how the benchmark code works. The implementation was done by rewriting the system-dependent section to use the available graphics hardware, compiling the code with Alliant's vectorizing and globally optimizing C compiler, using a profiling tool to determine the portions of the code that used the greatest percentage of CPU time, inserting compiler directives in the form of comments to break implicit dependencies in four sections of the benchmark, and recompiling. Alliant provided results for five configurations of the FX-80, with one, two, four, six, and eight computational elements. To save space, only two of the configurations are presented here. Table X shows the results for a single FX-80 computational element, and Table XI shows an FX-80 with eight elements. Table XII shows the validation output from the Alliant, which was identical for all configurations. The validation output shows a small variation from that of the Sun-3, but this is due to differences in the floating-point calculations that are within acceptable limits and produce the same final result. Alliant pointed out that the C compiler was a new product at that time and did not yet provide as much optimization as the FORTRAN compiler (a difference of up to 50% in some cases).

Figure 8 compares relative times for the two FX-80 configurations. The parallel configuration achieves the greatest improvement on the floating-point operations (median filter and Sobel), the Hough probes, and the labeling of the connected components. These are the four subtasks that received special attention in optimizing the implementation. The proportional effort thus grew for overhead and the other tasks.

Image Understanding Architecture

The Image Understanding Architecture (IUA) is being built by the University of Massachusetts and Hughes Research Laboratories specifically to address the problem of supporting real-time, knowledge-based vision. The architecture consists of three different parallel processors, arranged in a hierarchy that is tightly coupled by layers of dual-ported memory between the processors. The low-level processor is a bit-serial, processor-per-pixel, SIMD, associative array. The intermediate-level processor is an SIMD/MIMD array of 4096 16-bit digital signal processors that communicate via an interconnection network. Each intermediate-level processor shares a dual-ported memory segment with 64 low-level processors. The high level is a mulitprocessor intended to support AI processing and a blackboard model of communication through a global shared memory, which is dual

TABLE VII

Sun-3/160 and 3/260 Validation Output (Identical Results for Both Configurations)

Data set	Sample	Test	Test2	Test3	Test4
Connected components	134	35	34	114	100
Right angles extracted	126	99	92	210	197
Rectangles detected	25	21	16	42	39
Depth pixels > threshold	21,256	14,542	12,898	18,584	18,825
Elements on initial probe list	381	19	27	400	249
Hough probes	55	3	5	97	93
Initial match-strength probes	28	20	15	142	142
Extension match-strength probes	60	3	5	110	97
Models remaining	2	1	1	2	1
Model selected	10	1	5	7	8
Average match strength	0.64	0.96	0.94	0.84	0.88
Translated to	151,240	256,256	257,255	257,255	257,255
Rotated by (degrees)	85	359	114	22	22

TABLE VIII

Sun-4/260 Times for Major Subtasks

Data set	Sample	Test	Test2	Test3	Test4
Total	121.01	42.64	40.94	81.05	82.84
Overhead	4.34	3.92	3.79	4.08	4.11
Label connected components	4.74	4.56	4.54	4.62	4.61
Rectangles from intensity	1.10	0.68	0.71	0.96	0.91
Median filter	30.53	14.64	11.30	11.30	11.34
Sobel	12.16	11.43	11.27	11.41	11.45
Initial graph match	3.42	3.46	3.54	10.10	9.94
Match-strength probes	9.91	0.45	0.79	6.65	6.08
Hough probes	51.20	1.73	2.64	29.52	31.99
Result presentation	3.38	1.67	2.24	2.07	2.02

ported with a segment of the intermediate-level processor's memory. A detailed description of the architecture can be found in [14].

Because the architecture is under construction, an instruction-level simulator was used to develop the benchmark implementation. The simulator is programmed in a combination of Forth and an assembly language which has a syntax similar to that of Ada assignment statements. The benchmark was developed over a period of about 6 months, but much of that time was spent in building basic library routines and additional tools that were generally required for any large programming task. A $\frac{1}{64}$th-scale version of the simulator (4096 low-level, 64 intermediate-level, and 1 high-level processor) runs on a Sun workstation and was used to develop the initial implementation. The implementation was then transported to a full-scale IUA simulator running on a Sequent Symmetry multiprocessor.

Table XIII presents the IUA results with a resolution of one instruction time (0.1 ms). There are several points to note. Because the processing of different steps can be overlapped in the different levels, the sum of the individual step timings does not equal the total time. Some of the individual timings are averages, since intermediate-level processing takes place asynchronously and individual processes vary in their execution times. For example, the time for all of the match-strength probes is difficult to estimate since probes are created asynchronously and their processing is overlapped with each other and with other steps. However, the time for match extension includes the time to complete all of the subsidiary match-strength probes. Thus, where the table would usually break the match extension step into separate times for match-strength and Hough probes, it shows the total time for match extension and the average time for an individual probe.

Table XIV shows the validation output for the IUA. The number of elements on the initial probe list is not given because parallel tasks were used, and thus there is no single initial probe list. The number of probes varies from the sequential version because a somewhat more robust variation of the probe algorithm, which vetoed poses at different points in the matching process was used. Also, the separate processes shared their probe results so that a few duplicate probes were eliminated. The added robustness in the probe algorithm also lead to a slightly lower average match strength.

Lastly, it should be mentioned that the intermediate-level processor was greatly underutilized by the benchmark (only 0.2% of its processors were activated), and the high-level processor was not used at all. The low-level processor was also idle roughly 50% of the time while awaiting requests for top-down probes from the intermediate level.

Figure 9 shows the relative time spent by the IUA on each major subtask, for each data set. The graph-matching and match extension processes are clearly a dominant factor. Because task parallelism was used to match each model separately, the maximum obtainable parallelism was a factor of 10, versus a factor of over 200,000 for the bottom-up subtasks, which were done with data parallelism. In practice,

TABLE IX

Sun-4/260 Validation Output

Data set	Sample	Test	Test2	Test3	Test4
Connected components	134	35	34	114	100
Right angles extracted	126	99	92	210	197
Rectangles detected	25	21	16	42	39
Depth pixels > threshold	21,254	14,531	12,892	18,579	18,822
Elements on initial probe list	381	19	27	389	248
Hough probes	55	3	5	93	92
Initial match-strength probes	28	20	15	142	142
Extension match-strength probes	60	3	5	105	97
Models remaining	2	1	1	2	1
Model selected	10	1	5	7	8
Average match strength	0.64	0.96	0.94	0.84	0.88
Translated to	151,240	256,256	257,255	257,255	257,255
Rotated by (degrees)	85	359	114	22	22

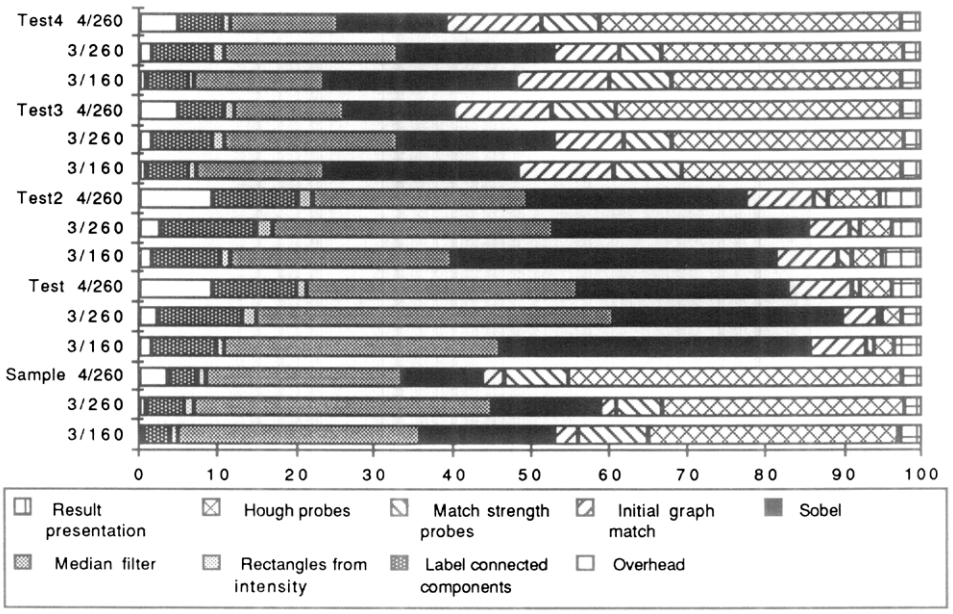

FIG. 7. Sun workstation percentage of effort for each major subtask.

the improvement due to task parallelism only equaled the number of models not vetoed during the matching process. However, the low utilization of the intermediate-level processors permits the task-parallel solution to operate on a set of several thousand models with about the same performance. The overhead for the IUA includes generating tables that are used in multiple places in the processing. Note that the time for labeling connected components is so small that it is invisible. For Test3 and Test4, the median filter is also too small to be visible.

Aspex ASP

The Associative String Processor (ASP) is being built by R. M. Lea at the University of Brunel and Aspex Ltd. in England [6]. It is designed as a general-purpose processing array for implementation in wafer-scale technology. The processor consists of 262,144 processors arranged as 512 strings of 512 processors each. Each processor contains a 96-bit data register and a 5-bit activity register. A string consists of 512 processors linked by a communication network that is also tied to a data exchanger and a vector data buffer. The vector data buffers of the strings are linked through another data exchanger and data buffer to another communication network. One of the advantages of this arrangement is a high degree of fault tolerance. The system can be built with 1024 VLSI devices, 128 ULSI devices, or 32 WSI devices. Estimated power consumption is 650 W. The processor clock and instruction rate is projected to be 20 MHz. Architectural changes that would improve the benchmark performance include increasing the number of processors (improves performance on *K*-curvature, median filter, and Sobel), increasing the speed of the processors and communication links (linear speedup on all tasks), and adding a separate controller to each ASP substring, resulting in an approximately 18% increase overall.

Because the system is under construction, a software simulator implementation was used. The benchmark was programmed in an extended version of Modula-2 over a period of 3 months by two programmers, following a 3-month period of initial study of the requirements and development of a solution strategy. A Jarvis' March algorithm was substituted for the recommended Graham Scan method on the convex hull. Table XV lists the major subtask times for the ASP. Timings were not provided for several of the minor steps in the model-matching portion of the benchmark, because a different method was used. The time under overhead accounts for the input and output of several intermediate images. The time under the section that extracts rectangles

TABLE X

Alliant FX-80 Single-Processor Times for Major Subtasks

Data set	Sample	Test	Test2	Test3	Test4
Total	207.890	104.561	95.139	139.808	142.162
Overhead	8.744	8.702	8.672	8.664	8.658
Label connected components	17.185	17.088	17.053	17.195	17.189
Rectangles from intensity	3.350	2.058	2.126	2.993	2.929
Median filter	77.464	43.812	32.049	32.073	32.046
Sobel	26.148	26.080	26.064	26.129	26.130
Initial graph match	2.546	2.460	2.624	7.485	7.384
Match-strength probes	7.235	0.316	0.576	4.768	4.371
Hough probes	60.956	1.901	3.312	37.631	40.632
Result presentation	3.271	1.862	2.390	2.179	2.176

TABLE XI

Alliant FX-80 Eight-Processor Times for Major Subtasks

Data set	Sample	Test	Test2	Test3	Test4
Total	60.112	33.138	32.915	53.952	53.620
Overhead	8.787	8.728	8.710	8.711	8.721
Label connected components	7.225	7.136	7.119	7.252	7.264
Rectangles from intensity	3.462	2.113	2.177	3.114	3.060
Median filter	10.113	5.857	4.323	4.324	4.318
Sobel	3.799	3.790	3.788	3.796	3.796
Initial graph match	2.578	2.493	2.655	7.615	7.473
Match-strength probes	7.232	0.317	0.576	4.804	4.404
Hough probes	13.090	0.554	0.898	11.374	11.716
Result presentation	3.267	1.861	2.384	2.180	2.175

from the intensity image accounts for the output and subsequent input of data records for corners and rectangles. The output and input of intermediate data were done to take advantage of the vector data buffers in the ASP, which allow strings to be quickly transferred out and then to be rebroadcast to the array. The implementors were thus able to cleverly recast the task-parallel orientation of the model-matching process into a data-parallel form by creating all of the different matching combinations, so that only a simple comparison was required to determine a match. However, for large sets of models, this technique is likely to result in an excessive number of combinations. The use of a data-parallel technique also makes it harder to compare the ASP with other systems which could have benefited from that method. Table XVI shows the validation output, which is similar to the sequential output except for some variation in the floating-point results and the absence of data for the number of elements on the initial probe list.

Figure 10 shows the percentage of time spent by the ASP on each major subtask. The time is dominated by labeling connected components and performing Hough probes, which require significant amounts of communication between strings, which must pass through the local data exchangers and then through the data exchanger that connects the ends of the strings together. Because of the data-parallel method that was used, the time for the initial graph match step is invisible in the figure.

Sequent Symmetry 81

The Sequent Computer Systems Symmetry 81 multiprocessor consists of Intel 80386 processors, running at 16.5 MHz, connected via a shared bus to a shared memory. The configuration used to obtain these results included 12 processors (one of which is reserved by the system), each with an 80387 math coprocessor and 96 Mbytes of shared memory. The system also contained the older A-model caches, which induce more traffic on the bus than the newer caches. The timings in Table XVII were obtained by the benchmark developers as part of the effort to ensure the portability of the benchmark.

About a month was spent developing the parallel implementation for the Sequent. The programmer was familiar with the benchmark, but had no previous experience with the Sequent system. Part of the development period was spent modifying the sequential version to enhance its portability. The low-level tasks were directly converted by dividing the data among the processors in a manner that avoided write contention. About half of the development time was spent adding data-locking mechanisms to the model-matching portion of the benchmark and resolving problems with timing and race conditions. It was only possible to obtain timings for the major steps in the benchmark, because the Sequent operating system does not provide facilities for accurately timing individual child processes. The benchmark was run

TABLE XII

Alliant FX-80 Validation Output

Statistics	Sample	Test	Test2	Test3	Test4
Connected components	134	35	34	114	100
Right angles extracted	126	99	92	210	197
Rectangles detected	25	21	16	42	39
Depth pixels > threshold	21,266	14,542	12,888	18,572	18,813
Elements on initial probe list	374	19	27	389	248
Hough probes	55	3	5	93	92
Initial match-strength probes	28	20	15	142	142
Extension match-strength probes	60	3	5	105	97
Models remaining	2	1	1	2	1
Model selected	10	1	5	7	8
Average match strength	0.65	0.96	0.94	0.84	0.88
Translated to	151,240	256,256	257,255	257,255	257,255
Rotated by (degrees)	85	359	114	22	22