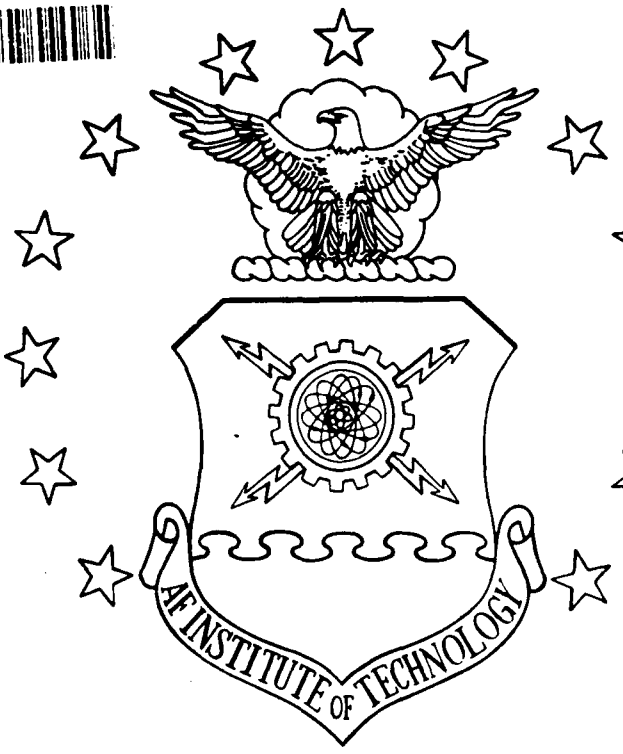


DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/91-D-11

Enhanced Animation Of Parallel Algorithms
Vol I

THESIS

Michael D. A. Lack
Flight Lieutenant, RAAF

AFIT/GCS/ENG/91-D-11

AFIT/GCS/ENG/91-D-11

Enhanced Animation Of Parallel Algorithms

Vol I

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science (Computer Systems)

Michael D. A. Lack, B.Eng., Grad.Dip.Bus.Comp.
Flight Lieutenant, RAAF

December, 1991

Acknowledgments

There are a number of people who deserve my thanks, for without them I would not have completed this task. Firstly, I would like to thank my advisor, Dr Lamont, for his guidance and advice, and for enduring my periods of lateral thinking. I would also like to thank a number of my fellow students. Don Blankenship, Tim Jacobs and Joe Simpson, without whose help I would never have been able to write "Hello World", let alone performance animations in the C language. Thanks also to the students in the graphics lab for advice on system details and Latex and the students in the "parallized" group for their insight and assistance, in class and on the golf course.

Certainly I must acknowledge the efforts of my sponsor, Lt J. P. Nauseef, who attemted to drag me away from my work at every possible moment for a couple of drinks at Harrigans. Never has there been such an inspiring environment for developing massively parallel visualizations and headaches.

Finally I'd like to thank the many friends I have made in the short time that I have been in this country. Your support has been invaluable.

Michael D. A. Lack

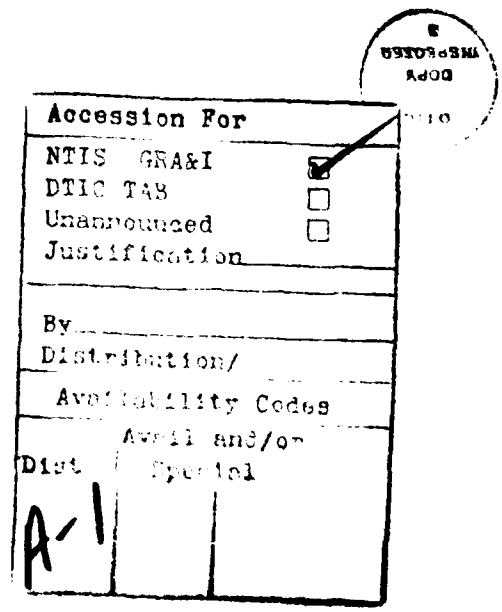


Table of Contents

List of Figures

List of Tables

Abstract

Algorithm animation is a visualization method used to enhance understanding of the functioning of an algorithm or program. Visualization is used for many purposes, including education, algorithm research, performance analysis, and program debugging. This research, which follows from previous work, examines algorithm animation requirements for the various visualization purposes and extends the capabilities of an existing facilities, the AFIT Algorithm Animation Research Facility (AAARF). The structure of AAARF is summarised and its visualization capabilities presented, analysed and compared with other similar packages.

This research focuses on the parallel data requirements of the AAARF users, and the meaningful display of large amounts of data. This paper discusses a series of refined animations suitable for a variety of purposes which are capable of conveying detailed information in a concise and timely manner and the development and implementation of a new performance animation type is presented. These animations are built upon pedagogical efforts in a classroom environment. To assist novice users in efficiently using AAARF, an "expert system" interface has been implemented to advise on optimizing environment configuration. Considerable effort has been devoted to porting AAARF to the X Windows envrionment and the lessons learned and progress made are discussed.

Enhanced Animation Of Parallel Algorithms

Vol I

I. Introduction

Multiprocessor computers offer great possibilities in terms of speedup over single processor machines. Exploitation of this potential is dependent on the effective use of machine resources by well constructed parallel algorithms. However, the software structure for parallel machines is complex, and the production of the software is correspondingly more difficult. The behavior of a complex parallel program and its associated data structures is more difficult to picture mentally than a sequential program and this can hinder the job of programmers and designers during development and debugging of the software.

The importance of visualization as a communication tool has long been acknowledged. Recently, a growing consensus has emerged regarding its' potential for promoting the understanding of complex behaviors exhibited by physical phenomena and parallel computations. The AFIT Algorithm Animation Research Facility (AAARF) has been developed as an aid to programmers, software engineers, and educators to visualize the dynamic behavior of parallel or sequential algorithms and their data structures during the execution of appropriately instrumented programs written in the C language. AAARF is a window-based system operating in the Sunview(54) environment. AAARF provides a range of graphical animation structures which detail communications between processing nodes, utilization of nodes and node balance. This is accomplished using graphical representations such as Kiviat (processor utilization) and Feynman (utilization and communication) diagrams, based on those developed for another parallel visualization package, Paragraph(23). Other estab-

lished graphical forms are also employed, such as Gantt charts, bar charts and the like. Additionally, AAARF may also be configured to animate data sortedness, search status, process states, and more, the variations limited only by the users imagination. The animations are designed for use on color terminals, but most can be used with monochrome monitors. The performance diagrams are all effective for processor configurations of up to 32 processing nodes(62).

1.1 AAARF Background

The current configuration of AAARF can be used for the analysis of sequential and parallel algorithms. It is the result of combining an earlier version of AAARF(20), used to animate sequential algorithms, with PRASE (Parallel Resource Analysis Environment)(31), a system developed at AFIT for the animation of processor utilization and communications on the Intel hypercubes. The current animations for parallel resource analysis are sufficient for limited applications but many are inadequate for the animation of large amounts of data, or real time operation.

The extremely high volume of information produced during the execution of a parallel algorithm greatly exceeds the human ability to assimilate it in textual form. This is in part due to the sequential processing of textual information. The human visual system is more suited to processing information in the form of images. Humans can process large quantities of image information in parallel, detecting and tracking complex visual patterns with incredible speed. Nevertheless, as the number of processes grows, the viewer's ability to understand the resulting image can be rapidly saturated unless the displayed information's level of abstraction is increased. Consider the problem of meaningfully representing the individual performance parameters of processors in a 128 node cube environment. This required the display of large amounts of data, clearly not a trivial task, but one that must be accomplished to satisfy the requirements of users working in an increasingly common environment. To simply produce a large scale version of the existing animations provided

by AAARF or other similar systems results in a screen image so crowded that little insight into the system performance can be gained. For this reason, abstraction plays an important role in visualization. By providing flexible abstractions, a visualization system can help the user select displays that are easily specified and understood.

The range of AAARF animation types already provided to the user by AAARF is large (twenty-seven). Each animation type illustrates different aspects of the algorithm execution and processor operation with varying effectiveness. A combination of up to four animation types can be produced by AAARF simultaneously. The correct combination of animations can provide a comprehensive picture of what is occurring during algorithm execution. With such a large number of combinations possible (> 100,000), selecting the most appropriate configuration is difficult, especially for the novice user. A system to assist the user in constructing an environment was needed.

Important to the future of AAARF, as a useful tool for programmers, is its' transportability and utility. Key to these concepts is AAARF's ability to be used on a range of machine architectures, with a range of operating systems.

1.2 Problem

Despite its usefulness, the current configuration of AAARF is restricted to animating sequential algorithms operating on Sun workstations and parallel programs operating on the Intel hypercubes, all of which must be written in C. Also, AAARF's current animation capabilities are not suited to the task of representing algorithms and processor architectures, which produce large amounts of display data. The operation of AAARF, though aided by a menu driven interface, can be less than friendly to the novice user, and incorrect operation can result in system hangups.

1.3 Objectives

The goal of this study is to build upon the solid foundations of the current AAARF, enabling it to be portable to multiple architectures and operating systems, improving its ability to display large amounts of data, increase user friendliness and improve the package robustness. To attain this goal the extended system must:

- retain the valuable assets of the current configuration of AAARF
- assist the user in animation selection
- use animations that can be quickly displayed and updated
- represent data in an efficient, comprehensible manner
- support multiple levels of abstraction
- fail in a safe condition
- operate in a readily transportable environment

The problem divides itself into the components described in the following paragraphs.

1.3.1 Improved Animation Most of the recent effort dedicated to the development of AAARF has been in the integration of PRASE and an earlier version of AAARF, and the development of a real-time display ability(62:6-5). The animations used by AAARF for parallel resource analysis are those originally designed for the PRASE system. These animations are currently restricted to the animation of 16 node architectures and were not designed for real time performance. The current set of animations is incapable of instantaneously displaying data related to the total processing time, and is instead limited to current values or windowed intervals of time. Many of the current animation types suffer from lengthy production processing delays, and much of the real time performance advantage is lost. Consequently, many improvements can be made in the methods of animation to improve efficiency and comprehension. Areas to be researched with regard to this objective include:

- facilities provided by similar algorithm animation packages
- provision of variable size animation windows
- examining the feasibility of "process" unique animations
- the use of an expert system to prioritise the data to be displayed and configure the display appropriately
- the representation of data using color, 3 dimensional displays, sound and user interactive displays.
- variation in data abstraction levels for better real time and play back effectiveness and comprehension

1.3.2 User Interface As the AAARF package expands, the number of configuration options available to the user for its configuration to suit his purposes increases. In general, the user knows how his algorithm operates and in what aspects he is interested. However, he may not know which AAARF *animations best* illustrate this behavior. Therefore, a method of transforming the users knowledge into AAARF configuration parameters is required. The feasibility of using an expert system to improve the performance of the user interface is examined in this investigation. An expert system is used to attain the user's level of expertise, and where necessary, aid the user's selection of animation formats, by analyzing the users interests, algorithm and data types. Additionally, the expert system may advise or implement the appropriate animations. The development of the user interface is discussed in detail in Chapter 4. Moreover, the robustness of the package is improved to avoid system crashes caused by user error.

1.3.3 Operating Environment There are two major reasons for considering a change in operating environment for AAARF. The existing AAARF system is restricted to use on UNIX workstations with Sunview operating environment compatibility. This field includes Sun3, Sun4 and Sparc workstations. While these are

common systems in many academic and research facilities, there are other operating environments that are commonly available and are more readily transportable. Another reason for considering alternative operating environments, is that the existing AAARF configuration requires a large amount of resources. While the Sunview environment does not greatly restrict AAARF from a physical output perspective, it does restrict its use to workstations, with at least 20 megabytes of swap space. An operating environment such as X Windows allows portability among different hardware environments(46) and will allow AAARF to run acceptably on minimally capable machines while allowing it to benefit from higher performance systems. This topic is discussed at length in Chapter 5.

1.4 Assumptions

The following assumptions are based on conclusions drawn by Williams(62:3), Fife(20:4) and others(33, 31):

- Algorithm animations cannot in general be generated automatically. Since the operations performed by each type of algorithm are different or use different combinations, automated detection and interpretation of these operations is not possible. A similar argument holds for developing the graphical view of the data. The same data can mean different things when presented in different ways. This does not rule out the use of an automated system for identifying obvious operations or important events, but a programmer familiar with the algorithm is needed to complete the task.
- Any algorithm can be animated. All algorithms manipulate data structures, so there is always some possibility for displaying that data in order to gain insight into the operation of the algorithm.

1.5 Scope

The object of this study is to extend the existing AAARF configuration and develop new techniques for visualizing the operation of sequential and parallel programs. Different algorithms are implemented to test these techniques, but algorithm research is not a part of this study. The result of this research is an improved prototype system for animating parallel algorithms and recommendations for future development.

1.6 Summary of Current Knowledge

The field of algorithm animation is still in its infancy, even more so where parallel programming is concerned, and AFIT's entry into this foray, AAARF, is comparable in performance to most other documented systems for algorithm visualization (see Chapter 2). Moreover, the field of graphical representation of data has, and continues to be, an area of conflicting views and experimentation. The Animation methods used by similar packages, such as Seeplex(23), ParaGraph and Balsa(10), are of a similar style to AAARF (as discussed in Chapter 2) and therefore other areas of dynamic animation must be looked to for insight into more effective graphical representation techniques. Such applications include such diverse areas as: the Pilots Associate program(49), statistical analysis packages, and nuclear power plant monitoring systems, all of these applications are required to impart large amounts of knowledge to the user in a short amount of time. Many of the lessons learned in these areas apply equally to algorithm visualization(30, 3, 56). These techniques and their application to algorithm animation are discussed in Chapter 3.

1.7 Approach

Recognizing the significant effort already invested in the AAARF package, the development of the improved package does not dramatically change the overall flavor of AAARF. Sound software engineering practices have been observed, resulting

in a well-structured and informatively documented package, and the principles of object-oriented software design are adhered to, allowing for future modification. The transfer ot AAARF from the Sunview environment to the X Window environment and the integration of an expert system is accomplished with as little disturbance to the virtual environment as possible. Several alternative approaches to performance animation are considered and the most efficient/feasible are chosen. The alternatives are evaluated using these criteria:

- Complexity - The usefulness of an animation is dependent of its complexity. More complex diagrams are harder to animate and consequently less useful in a real time environment.
- Flexibility - Can small variations be made to make the animation suitable for real time playback? Can the animation be used productively with monochrome displays?
- Understandability - Can the animation be looked at and understood in real time?
- Environment - Can existing and future animations be produced in the chosen environment?
- Data Requirement - Can the animation be produced with the data provided by the current instrumentation?

Environment alternatives were evaluated using the following criteria:

- Portability - What are the hardware requirements for the environment? What are the operating system requirements of the environment? How prolific are these?
- Flexibility - What capabilities does the environment require of the host machine e.g. graphics capability, screen size.

The choice of an expert system language to be used for the development of an expert user interface was based on the following criteria:

- Language Structure - Is the structure of the language appropriate to the problem?
- Portability - What type of machine, and operating system is required for the use of the language?
- Availability - Is a compiler for the language readily available to intended users? Ideally a compiler should be available at low cost or for free.

Once the foundation is established, techniques from other fields can be used to aid construction. The use of expert systems as user interfaces is becoming an increasingly popular method to extract and interpret the required information from the user in a diverse range of computing application fields(50, 22).

1.8 Summary

The goal of this thesis investigation is to further develop the ability of the user to observe what is occurring during the execution of a program on a parallel system and expand AAARF's usefulness as both a program debugger and educational tool. This goal is achieved through the development of user interface tools, extended animation facilities and multiple operating system compatibility. The following chapters discuss the approach taken in the development of AAARF and the research undertaken.

II. Graphical Parallel Performance Animation Techniques

This chapter discusses the basic principles of parallel algorithm visualization. The chapter begins with a brief introduction to the concept of algorithm animation, explaining the terms used, and alternative approaches and tools that may be applied. Following this is a review of the current performance of visualization by AAARF, which discusses its' strengths and shortfalls. The last section describes the range of animation techniques and styles developed for the animation of algorithms by other designers and compares them with those used by AAARF.

2.1 Background

The purpose of this section is to introduce terms and definitions for the subject areas that form the basis for this research and to familiarize the reader with the current capabilities of AAARF. If additional information is required, the reader is directed to the references which provide a more detailed discussion of the subjects.

Program Visualization Program visualization is the use of graphics to illustrate some aspect of a program's execution(43:2). This is a very broad definition, but within program visualization there are four major categories based on the type of information displayed and the method used for the display:(43:15)

1. Static display of program code
2. Static display of data
3. Dynamic display of program code
4. Dynamic display of data

The last category is also called *algorithm animation*(20:19).

There is much research being done in program visualization, and Fife(20) and Myers(43) review many of the systems available now. Additionally a brief review of the capabilities of some of these other systems is presented later in this chapter.

Instrumentation The term instrumentation is usually used in reference to electronic equipment that is used to monitor the conditions within a circuit or system. When applied to software, the term means very much the same. Instrumentation in software consists of modules that are inserted into the users code to monitor an activity and send data out to another system so that it can be displayed. An undesirable side effect of instrumentation is that it can disrupt the timing and performance of the program it is monitoring(62).

McLaren and Rogers(38) suggest that instrumentation for performance monitoring must perform three basic tasks:

1. Data Acquisition
2. Data Storage
3. Processing and data reduction

The ability of different systems to meet these goals is discussed later.

Data acquisition can be accomplished in two different ways. These are *event detection* and *sampling*. Event detection generates event data every time the machine changes state. The data is then stored by the monitor for subsequent processing. If sampling is used, data is collected and stored at periodic intervals. The trade-off between these two approaches is the accuracy of the data versus the impact of the monitor on the target system. A sampling monitor offers us decreased accuracy but has less impact on the target machine. Given the asynchronous nature of distributed computing, the generally preferred method is event driven data capture.

Having made the decision as to when to collect data, a decision must be made on how the data is to be collected. The three basic classes of instrumentation system monitors are:

1. hardware,
2. software, and
3. hybrid.

Hardware monitors use physical probes to collect information from the electrical signals of the system. Carefully implemented, these have little or no impact on the system performance. The type of data collected is at a very low level and is often difficult to interpret. Additionally the state of the machine within on-chip devices such as caches and processors is not visible, making this a less than optimal solution.

Software monitors on the other hand use software probes, i.e. lines of code embedded in the program with the intention of passing control to an instrumentation routine upon the occurrence of an event. The impact of this method upon the target machine may be extreme, but careful implementation can minimize the effect. The type of data obtained is at a very high level and is easily interpreted.

A hybrid monitor attempts to minimize performance impact while maximizing the usefulness of the data. Events are detected by adding trace statements to the code, or modifying the compiler to generate code to log specified events for specified operations. The processing of the data is performed by special purpose hardware. By compressing data such as event-counts, frequencies, and times, the amount of data to be stored can be drastically reduced.

Where software-correlated events are of interest, as is the case for multiprocessor systems, hybrid monitors offer the best compromise between performance impact and usefulness of data obtained.

2.2 *Current Capabilities of AAARF*

AFIT's contribution to the field of parallel algorithm visualization is AAARF(62), which was developed as an algorithm animation facility for parallel processes executing on different architectures, from multiprocessor systems to uniprocessor multitasking systems, with primary emphasis on the Intel hypercube. AAARF uses software probes to perform event sampling. Despite the inherent overheads associated with these techniques, test results have shown the overall system impact to be minimal(62:2-10). AAARF was written using an object oriented design approach, and the observance of proven software engineering practices resulted in an easily maintainable product.

AAARF animations are produced by a host machine, connected to the parallel machine where the algorithm of interest is being executed. This capability means that the parallel system need not have the graphics capability required to produce the detailed animations generated by AAARF. This flexibility is often lacking in similar systems. AAARF has a built-in capability to display performance data from the parallel system. AAARF is also able to animate existing programs that were developed independently from the animation system. This allows AAARF to also be used as a debugging tool. This last feature is one of AAARF's most powerful, and can assist in the optimization of parallel performance on a given architecture. Other similar systems (highly respected ones at that) such as BALSA(10), require code to be specifically developed for use with the animation system and are not suitable for use with parallel architectures. The heart of AAARF's ability to analyze parallel performance is another AFIT product, the Parallel Resource Analysis Software Environment (PRASE)(20). PRASE is a totally integrated instrumentation package that extracts system performance data from programs on the Intel Hypercubes. PRASE data is generated by system events (e.g. CSEND, Exit, etc.) and user defined interesting events (e.g. IE_data(type, pointer, length))(19). Though similar instrumentation packages exist (e.g. Paragraph(23)), the designers of AAARF elected to use

Elements Sorted	Original Sort (ms)	*Instrumented With*	
		PRASE (ms)	PICL (ms)
1024	50	58	58
2048	85	101	95
4096	167	179	176
8192	376	386	392
16384	726	735	732
32768	1565	1577	1573
65536	3321	3329	3327

Table 2.1. Results of the Probe Effect Comparison

PRASE due to its well structured and documented source code. Williams(62:2-10) studied the probe effect of AAARF using a simple parallel shell sort, instrumented appropriately. The results are presented in table 2.2.

AAARF presents the user with a real time view of the system performance as the desired application is executed. The AAARF system performance animations fall into two format categories:

1. Windowed Time Intervals (WTI) - These animations are scrolling windows of execution time. They provide a short history (the length of which is dependent on the time interval displayed in the window) of the parallel performance.
2. Instantaneous Value Animations (IVA) - These animations show the current state of the performance parameters. Some also include a *high tide mark* which indicates the peak value of the parameter recorded up to the current point in time.

A summary of the views is presented in table 2.2 and they are described in the following paragraphs:

1. *Utilization* This view is a histogram which displays the number of active and inactive processors on the vertical axis, and time on the horizontal axis. The

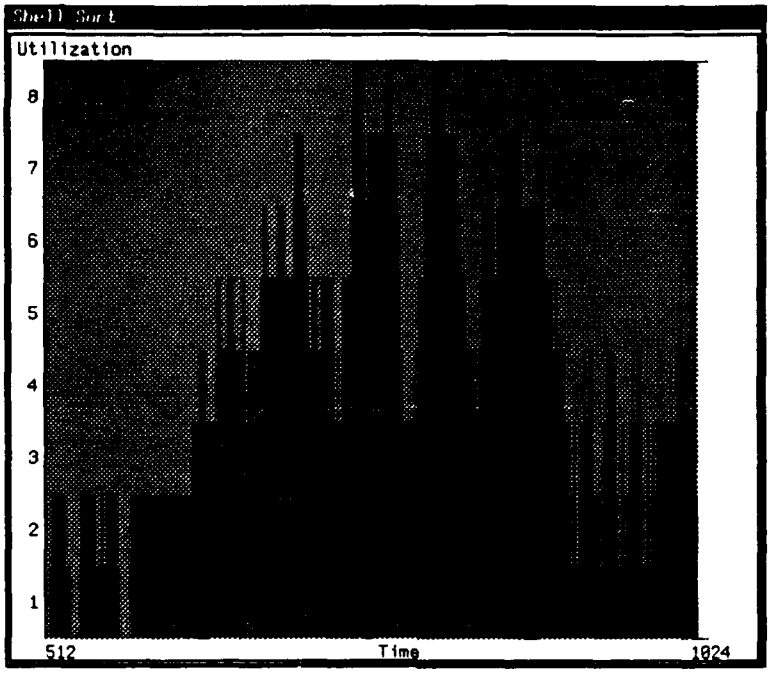


Figure 2.1. AAARF Processor Utilization Histogram

active vs idle state of the processor is derived from the message traffic: if a processor is blocked waiting for a message, then it is idle. Otherwise the processor is active. See figure 2.1.

2. *Gantt* This is a traditional view of processor activity. Each processor has a section of the vertical axis, and time is shown on the horizontal axis. The status of the processor is determined in the same way as for the utilization view. See figure 2.2.

3. *Feynman* This view shows two different types of data: processor status and message passing activity. The processor status is shown in a similar fashion to the Gantt view, except that the data is shown as a thin horizontal line that is broken when the processor is blocked. Once again, the horizontal axis represents time. The message activity is shown by drawing lines between the horizontal lines representing the sending and receiving processors: The ends of the lines are positioned according to the send and receive times. This clearly

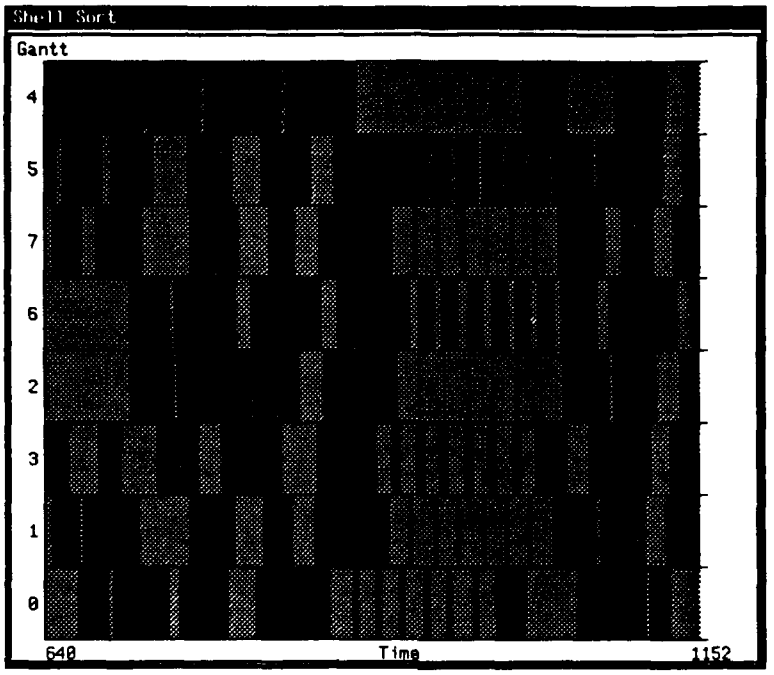


Figure 2.2. AAARF Gantt Chart

shows message passing patterns, delays, and bottlenecks. These displays are generated using trace records from the send and receive system calls. See figure 2.3.

4. *Communication Statistics* This view provides a detailed look at the communications activity of a single (selectable) node. The data displayed can be chosen from three types: processor source or destination for each message, length of the messages, or the type of the messages. See figure 2.4.

5. *Communications Load* This view shows statistics on pending messages for the entire system versus time. Either the number of pending messages or the total length of all pending messages can be displayed. See figure 2.5.

6. *Queue Size* Statistics for the input queue for one (selectable) node are displayed in this view. See figure 2.6.

7. *Message Lengths* This view uses a different method of presenting message passing information. The view contains a matrix, and a send operation causes an

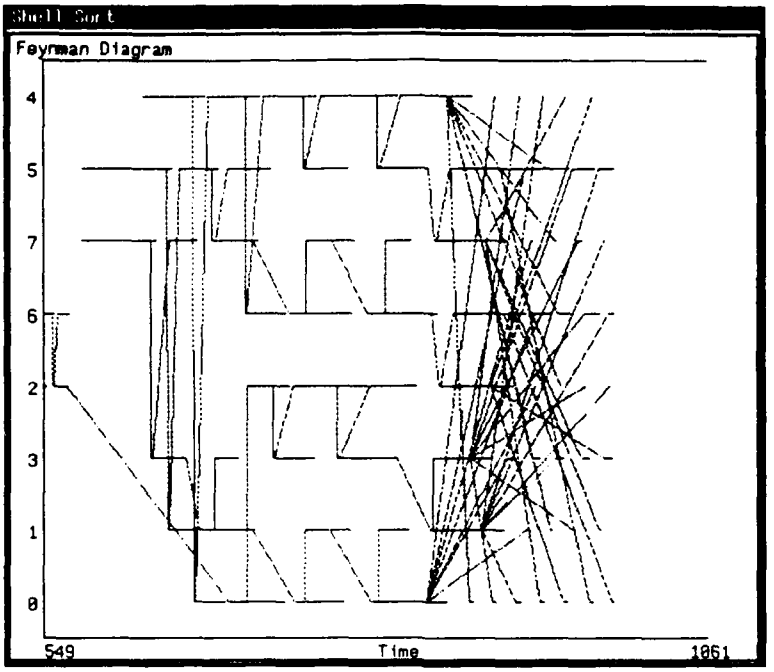


Figure 2.3. AAARF Feynman Diagram

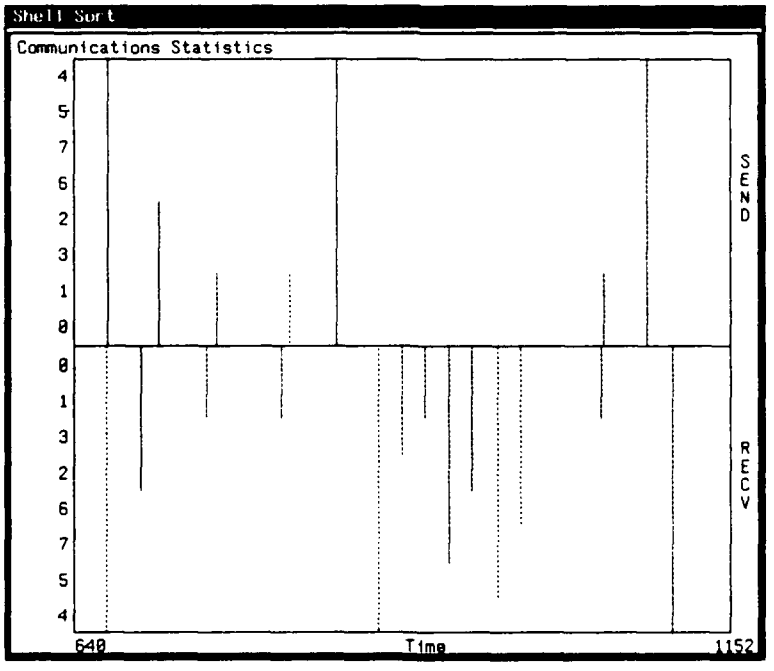


Figure 2.4. AAARF Communication Statistics

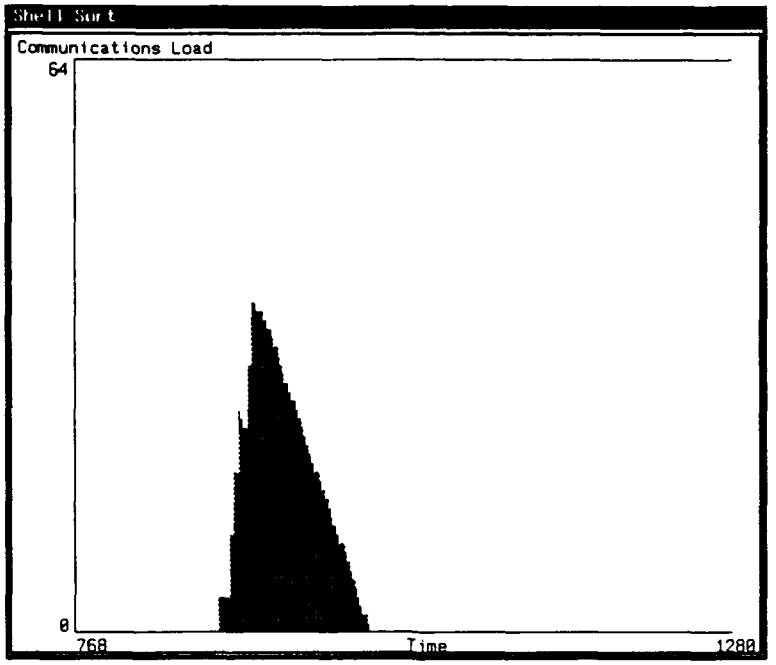


Figure 2.5. AAARF Communications Load Statistics

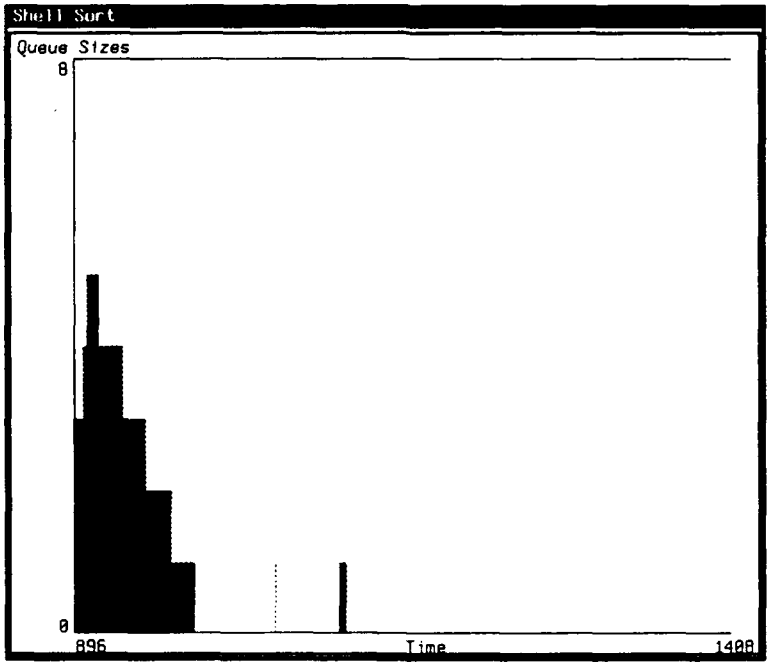


Figure 2.6. AAARF Input Queue Statistics for Single Node

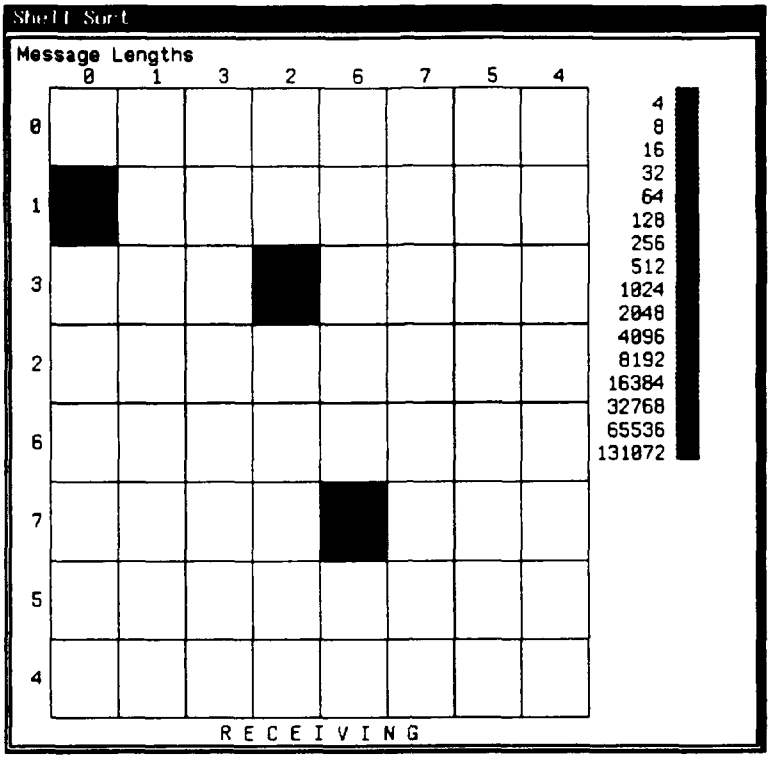


Figure 2.7. AAARF Message Length Chart

element of the matrix to be colored. The sending processor determines the row and the receiving processor determines the column. The length of the message determines the color. See figure 2.7.

8. *Message Queues* This view displays the current status of the input message queues for all the processors. The display is a histogram with the processors along the horizontal axis. The vertical axis can show either the number of messages in the queue or the total length of the messages in the queue. As the queue levels rise and fall, a "high tide mark" is left behind to mark the highest level that was attained for the run. See figure 2.8.

9. *Kiviat* Similar to the traditional kiviat chart - the processors are located at the perimeter of a circle and "spokes" are drawn from each processor to the center of the circle. The CPU utilization for each processor is calculated and plotted along the corresponding spoke of the wheel, with the center representing 0%.

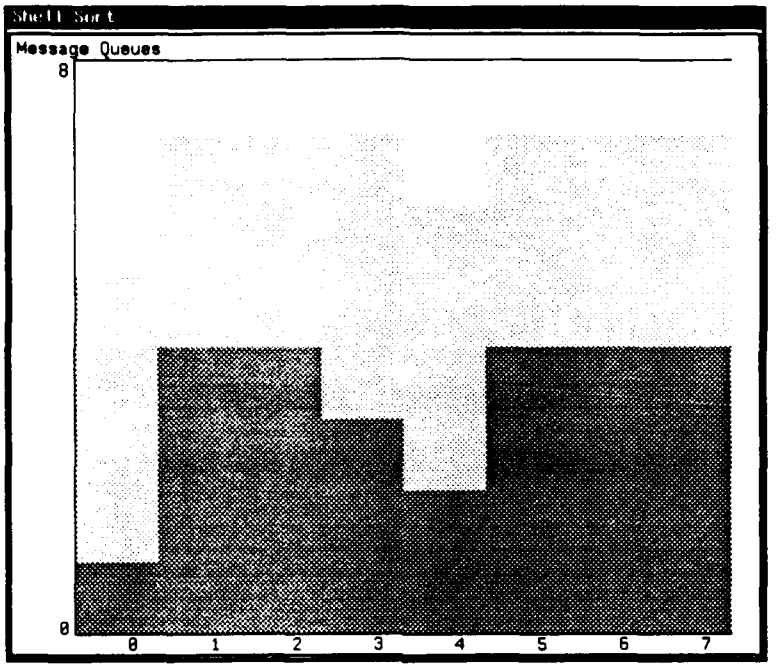


Figure 2.8. AAARF Input Message Queues for All Nodes

The polygon constructed by connecting points on adjacent spokes indicated the balance of the load. Once again a "high tide mark" is left. See figure 2.9.

10. *Animation* Processors are once again arranged in a circle. The color of the processor indicates its status - idle, busy, blocked, or sending. Lines are drawn between processors to indicate message activity. See figure 2.10.

In addition to these graphical representations AAARF is also capable of graphically representing algorithms as they proceed. Williams(62) describes in detail the representation of the Set Covering Problem through the use of a search tree structure, which grows as the search goes deeper. Other animation capabilities include a variety of representations of sorts using sticks, dots and rainbows. Examples of these can be seen in figures 2.11 and 2.12.

AAARF is a well structured and comprehensive assessment tool for the hypercube multiprocessor environment performance. It has been designed using an object oriented approach and has been well documented, with further development in mind.

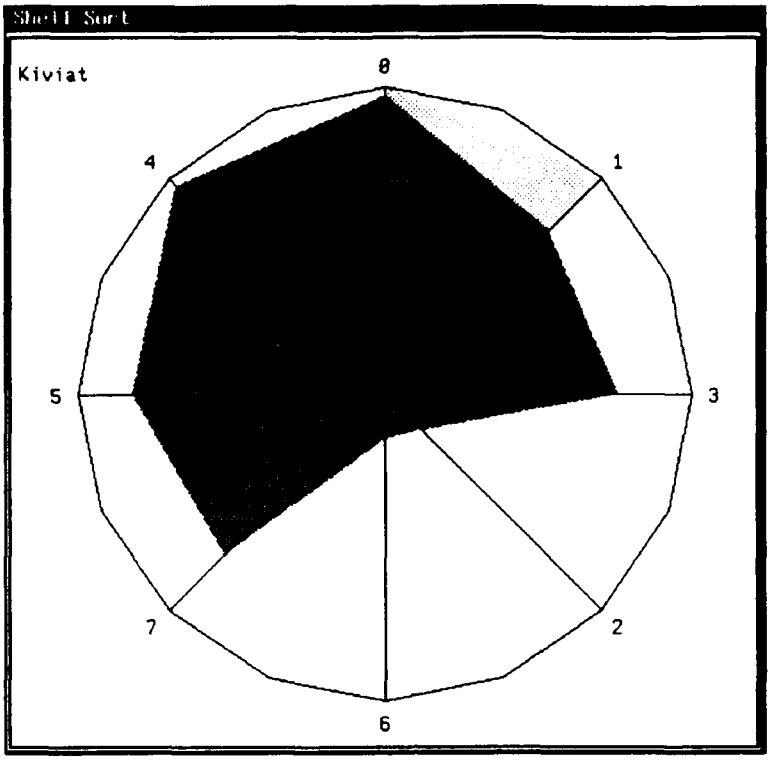


Figure 2.9. AAARF Kiviat Chart

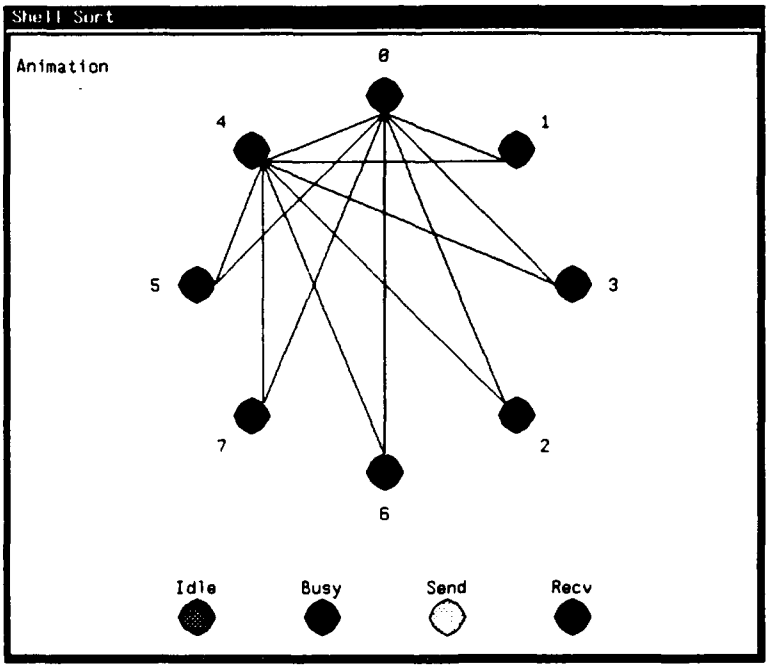


Figure 2.10. AAARF Processor Activity Animation

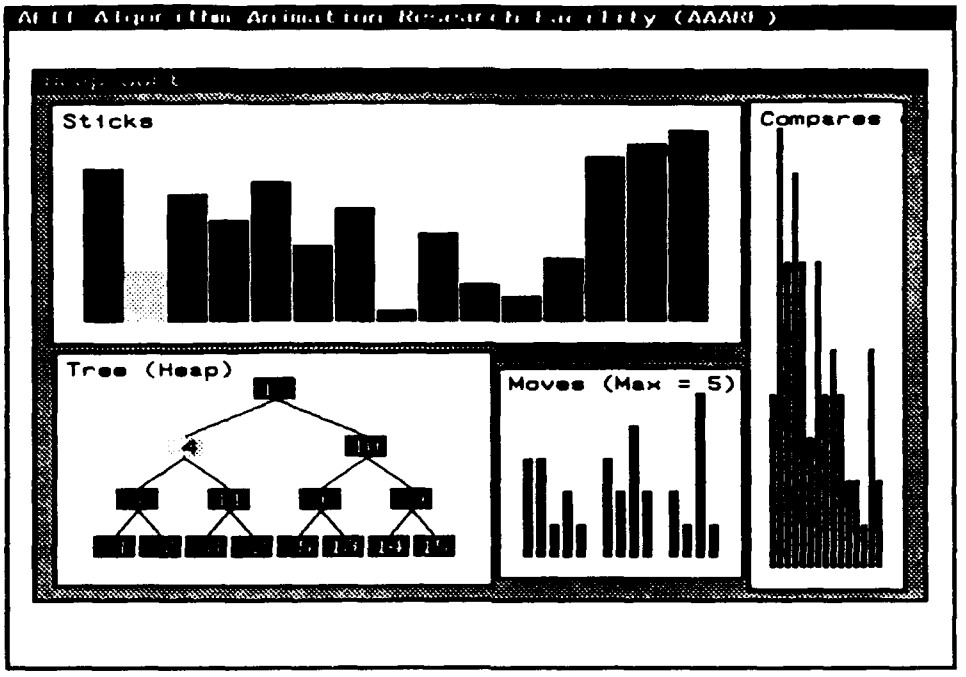


Figure 2.11. AAARF Sorting Animations

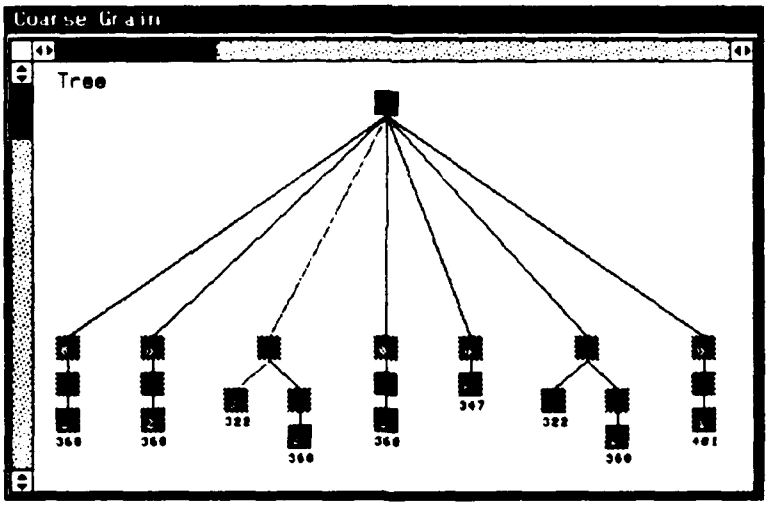


Figure 2.12. AAARF Set Covering Problem Animation

VIEW	TYPE	COLOR REQUIRED	HIGH TIDE	UNITS
Utilization	WTI	NO	NO	STATE
Gannt	WTI	NO	NO	STATE
Feynman	WTI	NO	NO	STATE
Comm. Stats.	WTI	NO	NO	STATE/TIME
Comms. Load	WTI	NO	NO	# OF MESSAGES
Queue Size	WTI	NO	NO	# OF MESSAGES
Mess. Lengths	IVA	YES	NO	BYTES
Mess. Queues	WTI	YES	YES	# OF MESSAGES
Kiviat	IVA	YES	YES	STATE
Animation	IVA	YES	NO	STATE

Table 2.2. AAARF System Performance Animations Characteristics

In its original form AAARF was restricted to use on workstations using the Sunview environment.

2.3 AAARF Requirements Analysis

Further to the requirements described by Fife(20) and Williams(62), the parallel animation system should meet the following requirements

1. The ability to animate programs in an acceptable time.
2. Be able to produce end-of-execution summary reports in a graphical format.
3. Be transportable between machines of different capabilities and operating systems.
4. Be able to animate large amounts of data in a readily comprehensible manner.

2.4 Parallel Algorithm and Performance Animation Packages

The following is a review of other packages developed to capture parallel algorithm performance information and present it to the user in some sort of meaningful format. The findings are summarized in table 2.4.

PACKAGE NAME	PROBE TYPE	INSTR PACKAGE	DISPLAY ENV.	ARCH.	MAIN PURPOSE	REAL TIME
Seeplex	Software	Simplex	Not reported	N Cube	Sys. Perf.	yes
MaTRIX	Software	PICL	X Windows	Various	Matrix Ops.	no
Paragraph	Software	PICL	X Windows	Various	Sys. Perf.	no
PIE	Software	Mach 1	X Windows	Various	Sys. Perf.	no
NIST	hybrid	not reported	not reported	MIMD	Sys. Perf.	yes
PAWS	N/A	N/A	X Windows	Simulated	Arch. Perf.	yes
Task Grapher	N/A	N/A	not reported	N/A	Scheduling	yes
Los Alomos	Software	Custom	Audio	Various	App. Perf.	yes
PF-View	Software	PF-Trace	Not reported	IBM 3090	App. Perf.	no
IVE	Software	Custom	Not reported	SIMD	App. Perf.	no
MTOOL	Software	Custom	X Windows Graphics	Silicon Bottlenecks	Identify	no
HyperView	Software	Simplex	X Windows	Hypercube	Sys. Perf.	no
AAARF	Software	PRASE	Sunview	Hypercube	Sys. Perf. Alg. Anim.	yes

Table 2.3. Parallel Visualization Packages

2.4.1 Seeplex The Seeplex performance monitor is a real time version of the Seecube post-mortem analyzer. In their article "Monitoring Parallel Executions in Real Time" Couch and Krumme(16) discuss the application of this package in the analysis of a game playing algorithm which incorporates load-balancing heuristics which may only be properly tuned by observation of the program behavior.

Seeplex uses high density color displays to depict program executions on a hypercube multicomputer. Seeplex obtains performance data from Simplex, an original operating system for the NCUBE processing nodes which contains extensive instrumentation for execution monitoring. The user has control over the program execution, the amount and nature of the monitoring and how it is displayed. The monitoring software is designed to cause minimal impact on the program being monitored.

Seeplex can be configured to monitor process state information, message traffic

rates, sizes of message queues, and the rate of issuance of system calls. The quantities displayed can be the sum, maximum, minimum, range, average, or standard deviation of the quantities on the constituent processors. The performance data is recorded in a file so that it may be reviewed in a step through backward or forward manner. Integration of the event history data into Seeplex is under development, as well as consequent fine-grain data analysis capabilities.

Discussion of the intricacies of the heuristics of the program developed for this analysis is beyond the scope of this review, however of particular interest is the way that the information regarding the operation of this program is conveyed to the user. The amount of control that the user has over this is startling.

Of key importance to the structure of the data presented is the "Trellis Editor". The "trellis editor" display contains "sources" (data sources) which provide raw data from the hypercube, "filters", which modify it in some way, and "displays" which accept filtered data and render it graphically. The user assembles the desired connections of sources, filters, and displays to form a display environment. Each source, filter, or display is represented as an icon on a large panel, where each icon may be opened into an associated display and/or control panel. The behavior of a filter is controlled by opening the filter's icon, modifying options, and closing it. The display associated with a display icon is the open form of that icon. At any time a random collection of the Trellis display icons will be open corresponding to the user's need to know particulars about parts of the data manipulation process. The "trellis editor" would appear to be a very powerful tool, however, it also appears to be a tool which requires the user to possess detailed knowledge effectively use it. The actual performance displays are similar to their counterparts produced by AAARF.

Seeplex is a powerful performance analysis and algorithm debugging tool which requires the user to have apriori extensive knowledge about likely areas of interest. Unfortunately, at this time its use is limited to the NCUBE hypercube due to its dependence on the Simplex operating system.

2.4.2 MaTRIX MaTRIX (Matrix TRace In X)(44) was designed for the performance evaluation of parallel algorithms for dense matrix operations. Colors and patterns are used to identify activity and differentiate between unique processors and various operations. The animation uses post processed trace data which may be played back at varying speed. This is a positive capability shared by AAARF since much dynamic data cannot be appreciated when displayed in real time due to the high frequency of events. MaTRIX was written to use X Windows and employ the tracing facilities in the Portable Instrumented Communication Library (PICL). This makes it portable to a wide range of parallel architectures and visual display bases.

MaTRIX uses software instrumentation of parallel matrix algorithms to collect trace data for visualization.

MaTRIX produces the following displays:

1. *Main Display* This display is a skeleton representation of the operand matrix. The area of the matrix being worked upon and the nature of that work is represented by different colorings. While AAARF produces no direct equivalent to this, the instrumentation code used by AAARF extracts enough information for such an animation to be produced. application.
2. *Utilization Graph* This graph bases activity on user information rather than processor state information. Hence, if a processor is performing "useless" work or overhead, it will not be shown as processor utilization. This form of utilization animation is ideal for identifying work conducted as a result of system overheads, if it is compared with the AAARF form of utilization graph.
3. *Gannt Chart* A variation of the Gantt chart produced by AAARF, this display gives the type and duration of operations at each processor, but only for those operations directly pertaining to the processing of the matrix.

MaTRIX appears to be a very powerful package for the analysis of matrix operations but as a generic tool for the monitoring and profiling of multiple algorithm type performance in multiprocessor environments it falls well short of the mark.

2.4.3 ParaGraph ParaGraph(23) is a graphical display system for visualizing the behavior of parallel algorithms on message passing multiprocessor architectures. It takes as an input, execution profile data provided by the PICL. PICL produces an execution trace during an actual run of a parallel program on a message passing machine, and the resulting trace data can then be replayed pictorially with ParaGraph to provide a dynamic depiction of the parallel program processor operations.

ParaGraph is based on the X Window system, and thus runs on a variety of workstations. Although most effective in color, ParaGraph also works on monochrome monitors. The user interface for ParaGraph is menu-oriented. The execution of ParaGraph is event driven, including both user-generated window events and trace events in the data file produced by PICL.

The displays produced are the same as those produced by AAARF, except that AAARF produces additional animated displays for specific algorithm representations. This is not surprising as ParaGraph was the inspiration for the designers of PRASE. However, ParaGraph does not share AAARF's ability to animate algorithm execution, nor does it have the capability to produce real time animations.

2.4.4 The PIE System PIE (Parallel Programming and Instrumentation Environment) supports a comprehensive software development methodology extended to the analysis, verification, and validation of a computation's performance(34, 35). The PIE system is a theoretical framework for developing techniques to predict, detect, and avoid performance degradation. It's basis is a programming paradigm in which functional and performance behaviour is made visible.

PIE is currently implemented on top of Mach(1), an operating system developed at Carnegie Mellon University; however, the system designers claim that PIE can be transported to other "Unix-like" operating systems. PIE is not architecture specific. Like AAARF it is a portable system whose platform is a workstation. PIE requires that the workstation be running X Windows, and have a minimum of 8 megabytes of RAM and 70 megabyte hard disk drive. The monitoring instrumentation is capable of running on a number of hardware platforms, including the VAX, the Multimax and the Encore.

PIE automatically generates a visualization of the principal constructs of an appropriately instrumented program, using a representation that looks much like a parallel version of a flow chart. After the constructs have been visualized, PIE gathers performance information during program execution which may be displayed using standard histogram and time plots and a parallel execution view.

The parallel execution view is similar in appearance to the AAARF Gantt chart except that the view may be structured so that the vertical intervals can represent program constructs or processing nodes depending on the users requirements. This view also has a zoom capability allowing the user produce an exploded view of a particular time interval. When the vertical intervals are used to indicate node performance, the plots are color coded to indicate which of the constructs are being executed at each node. These views are produced after execution of the program. The designers state that PIE will operate during runtime with limited capabilities, but do not state what these limitations are.

The parallel execution view generated by PIE appears to be a very useful tool. Its ability to display both processor performance and algorithm-specific performance is very good idea. This type of animation could be produced by AAARF with appropriately instrumented code. The most attractive aspect of this animation is its potential to animate parallel algorithm performance data for almost any type of application.

2.4.5 VISTOP VISTOP is a tool for the animation and visualization of the dynamic behaviour of concurrent programs running under the distributed operating system Multiprocessor Multitasking Kernel (MMK) on an INTEL iPSC/2 hypercube system(4). VISTOP is one of a series of tools in a package called TOPSYS, which are designed to simplify the usage and the programming of parallel systems. MMK offers a transparent multitasking process model. The necessary information for visualization is collected by the distributed monitoring system running on the iPSC/2 system. VISTOP operates on a UNIX workstation using the X Window System for its graphical user interface. The communication between the workstation and the hypercube system is performed by Intel's remote hosting software. In the current state of implementation only communication and synchronization events are visualized.

VISTOP works in much the same way as AAARF(5). VISTOP's basic data structure is a doubley-linked list, which contains snapshots of the program execution. Each snapshot consists of the status of all observed MMK objects at a certain time. VISTOP consists of two independent parts: the data acquisition part and the data animation part. The data acquisition part collects data from the application and stores it in the animation queue. The data animation part reads the contents of the animation queue and produces a sequence of pictures which animate the program execution.

In order to collect data of interesting communication events, VISTOP specifies a breakpoint predicate, which depends on the MMK objects selected by the user. The break predicate stops the application, whenever a communication event happens where one of the selected objects is involved. This has serious implications with regard to the performance of VISTOP. VISTOP is clearly not suitable for real time animation, and has a major impact on the execution of the application, as compared to the effect of the AAARF software probe.

2.4.6 Los Alamos National Laboratory Hotchkiss and Wampler(24) have implemented a system which auralizes, rather than visualizes, functions and data. Their work takes advantage of the recent developments in sound technology, including the Musical Instrument Digital Interface (MIDI), and the NeXT computer, which is equiped with a Digital Sound Processor (DSP).

They contend that within the next five years, multiprocessors will be generating teraflops. Research has shown that the human visual system is capable of processing raw data on the order of a few gigabits per second and therefore visualization packages will not be capable of conveying all the necessary information, say the authors.

There are a number of sound parameters that can be used to for the interpretation of scientific data. Hotchkiss and Wampler concentrate on the three basic parameters: frquency, amplitude and time. Additionally they also make use of the variety of musical instruments made availbale through the use of a Yamaha synthesizer. They cite come very interesting results:

> Mathematical functions can have a remarkable musical beauty. Exponential functions like y=x**n creates the sound of a buzz-saw when played as a set of drum rim shots. If one uses three different frequencies for the function f(t) = t + sin(wt), sounds are made that convey all three curves, but gives one an inherent feeling of an engine that is having difficulty warming up but finally does so. Chaotic functions such as the bifurcating function x(i+1)*x(i)r, when reiterated to convergence, clearly conveys the converged functional behaviour as well as the chaos where convergence does not occur....we have discovered that audibilized mathematical functions can create sounds that no man has ever heard before and truly excite even the non-musical mind.

A problem the authors freely admit to, is that the interpretation of sound is a subjective process, but they believe that this subjectivity leads to "a greater sense of being inside a function." Difficulties will occur when assesing what sound indicates correct, desired, improved, or erroneous function behaviour.

This approach to data representation is a very new one, and offers great potential, particularly if it were to be combined with a visualization package, resulting in process "audiovisualization".

2.4.7 PF-View PF-View(58) is a tool for visualizing the behaviour of parallel Fortran programs, designed specifically for use by scientific application programmers. It adds postprocessing of textual trace information to IBM's Parallel Fortran Trace Facility (PF-Trace). A particularly valuable aspect of this package, is the provision of a source code window that is cross-corelated to the application animation, allowing the user to easily identify application behaviour with the responsible source code.

PF-View works in much the same way as AAARF, in that PF-Trace reports the occurence of pre-defined events, in the form of time-stamped trace records written to a trace file. Additionally, a mechanism exists for user-defined events to also be recorded. Because PF programs are capable of producing incredibly large trace files, PF-Trace applies a filtering mechanism which controls which events are recorded and which are ignored.

Because PF-View is designed for use by a specific audience, the designers were able to take a unique approach to the problem of mananging the complexity of behavioural information. PF-View deals with this problem:

1. by identifying which events relate to the programmers work and applying a filter to remove all others;
2. by clustering together low-level events to correspond with the more abstract occurrences defined by the source code constructs; and
3. by employing a hierachical presentation schema to minimize the amount of display detail.

The animations shown in (58) indicate that PF-View provides only a high level illustration of processor and application behaviour. As a result, PF-View is a less

generalized package than others reviewed here, but is perhaps easier to use in its specialized domain.

2.4.8 The Integrated Visualization Environment (IVE) IVE(17) is a joint research project of Harvard University and the Lockheed Palo Alto Research Laboratories. The research breaks visualization into three components:

1. *program visualization*, which represents logical and structural software relationships,
2. *process visualization*, which depicts the execution of programs for the benefit of the programmer, and
3. *application visualization*, which presents the results of a computation in a perspicuous form for the end user.

Instrumentation of software is achieved in much the same way as other packages, but the trace data is processed in a manner which gives the user far more freedom over the design of animations. Interesting events are described as *phenomenon.* A *property* is a function that extracts an explicit and efficient description of one attribute of a phenomenon, and provides it to the *process monitor.* Sometimes a *property* may apply to more than one phenomenon. A visualization of one or more phenomena is created and updated by interpreting a *visualization template.* The time and effort required of the user to prepare visualization templates is the major cost of using the IVE system.

The IVE monitor allows the user to load, start, and stop a subject program, and to enable and disable the visualizations specified in the program. For each enabled visualization, a separate instantiator/renderer (IR) is established. To minimize the delay inherent in having many visualizations active simultaneously, the IVE system executes the IRs on several networked computers. the set of available hosts is specified byt ehuser when the system is invoked.

Currently under investigation by the IVE designers is a tool capable of assisting users in designing and implementing visualizations called Cooperative Computer-Aided Design (CCAD). CCAD allows the user to make the creative choices and guide the design process, while the computer infers detail that is consistant with the user's design decisions.

The prototype IVE system has been successfuly used to help debug a C program for comparing DNA sequences on a Connection Machine. The range of animations produced is very impressive, and they provide great insight into the applications behaviour. However, resource parameters, such as utilization and communication, are not animated.

The IVE system is an impressive package offering great freedom to the user, however it is obvious that this is a package that requires a trained user, and not a tool suitable for novices. There is a substantial overhead associated with the use of the IVE system in the design and creation of new visualizations, despite the provision of CCAD, and the IVE system may benefit from the provision of a library of animations for common applications.

2.4.9 MTOOL MTOOL(27) is an integrated tool for isolating perfromance bottlenecks in shared memory multiprocessor applications. MTOOL's window-based user interface displays information about bottlenecks hierachically at a whole program, process, loop, procedure,and synchronization object level. For portability and ease of modification, the user interface was built with an object-oriented toolkit, which runs on top of the X Windows system. MTOOL's users vary from numerical analysts to graduate student in parallel programming classes.

MTOOL isolates performance loss as follows:

1. MTOOL first instruments a program by adding basic block counters.

2. Using a knowledge of instruction latencies and the basic block counts obtained by runing the instrumented program, MTOOL builds an execution profile describing how much time is spent in each block.

3. MTOOL instruments a program to collect two kinds of information: actual time spent in selected loops, procedures, and synchronization calls and basic block counts.

4. Memory losses can be islolated by comparing actual time measuements to compute time estimates made using the basic block counts under the assumption of an ideal memory system.

5. Synchronization overheads can be identified directly from execution time mearurements of synchronization calls.

6. Extra work can be estimated using instruction count information whenever either the user tags instructions as "extra work" or MTOOL can identify the instructions as extra work because they occur in parallel control constructs.

The output provided by MTOOL is a set of statistics, bar charts and histograms produced at the completion of execution, which identify the allocation of processor activity during execution time.

2.4.10 HyperView HyperView(36) is the result of a collaboration between the Center for Supercomputing Research and Development and researchers at MIT. HyperView is a prototype perfromance visualization tool for distributed memory parallel processors configured as hypercubes. HyperView was inspired by Seecube and uses many of the same displays, however, HyperView is implemented in the X Windows environment.

Because X Windows supports a client-server paradigm, the data analysis and display portions of HyperView are decoupled, potentially executing on different systems. At present the HyperView visualization is driven by data obtained from the

simulation of communication hardware for different message passing paradigms, including store-and-forward message switching, circuit switching, staged circuit switching, and wormhole circuit switching.

The animations provided by HyperView are similar to those provided by AAARF. However, HyperView provides a more graphically accurate presentation of the cube structure. The AAARF *Animation* display, shown in figure 2.10, shows the processing nodes arranged in a circle, the HyperView equivalent presents agraphical representation of the cube as a cube.

HyperView was designed primarily to display hardware performance, and as such is not easily extensible to display of application performance. However, by manually instrumenting application code, and using extensive preprocessing applications, the designers were able to display application data using Mathematica, a symbolic manipulation system and mathematician's assistant. Current development is concentrating on integrating an application visualization capability.

As with many other systems reviewed here, HyperView does not offer near real-time performance. A posteriori examination of performance data mans that all dta of potential interest must be captured apriori.

2.4.11 NIST (National Institute of Standards and Technology) Project This project focuses on a hybrid measurement system implemented in MIMD multiprocessor systems, both loosely and tightly coupled(40). The designers propose a system using software (embedded code) triggers and hardware sampling, to introduce a minimal amount of perturbation to the executing program. The disturbance can be as small as a single memory write instruction per measurement sample.

The design uses two VLSI chips, and requires two small programmable logic devices to interface to a multiprocessor. The designers suggest that manufacturers could assist by making the desired signals readily available to allow the field addition of measurement facilities.

Still on the drawing board, this project promises much. The designers describe methods of producing output details similar to those produced by previously mentioned packages with some added benefits in terms of low level hardware performance.

2.4.12 PAWS The Parallel Assessment Window System (PAWS)(45) is an experimental system performing machine evaluations and comparisons. It provides a user friendly X Window based environment that allows analysis of existing, prototype, and conceptual machine architectures running a common application.

PAWS provides four basic tools:

1. *The Application Characterization Tool* This tool translates applications written in a high-level language (Ada) into a single data dependence graph. This allows users to view their application attributes. The data-flow graph is a machine-independent representation that may be mapped into any number of architectures.
2. *The Architecture Characterization Tool* PAWS assumes four basic classes of parallel architecture MIMD, SIMD, MISD and SISD. These are then broken down further according to the following guidelines:

   (a) the number and flexibility of different functional units;

   (b) the number of processors;

   (c) memory bandwidths and memory hierarchies; and

   (d) the types of interprocessor communication mechanisms.

   This tool allows the user to create, store, and retrieve descriptions of machines in a database. This approach permits users to evaluate conceptual machines before building any hardware. For meaningful results to be obtained, the data provided by the user must include such details as the geometry of physical and

virtual processors, the dimension of communication, and the data size. Files for the Thinking Machine's CM-2 and the Encore Multimax have already been developed. Heuristic algorithms for mapping into MIMD and SIMD machines are currently under development.

3. *Performance Assessment Tool* allows users to evaluate the performance of any application entered in the system by generating a set of performance metrics. These performance metrics include speed-up curves, parallelism profile curves, and execution profiles. These metrics are generated for both the ideal case and for the application after it has been mapped onto a specific machine architecture.

4. *Interactive Graphical Display Tool* provides the user interface for accessing all other PAWS tools. It has been implemented as a hierarchial menu-driven system, allowing multiple windows to be opened in a single session.

PAWS, as a package, is still in its infancy and the data it produces is at a very low level. The PAWS package was developed under a grant from Rome Air Development Center. This type of tool could be used interactively with a tool such as AAARF with successive approximations improving in accuracy.

2.4.13 Task Grapher Task grapher(18) may be considered as a simulation tool used for studying "optimal parallel program task scheduling on arbitrarily interconnected parallel processors". That is, task grapher allows the user to input a program represented as a precedence constrained task graph, and the desired interconnect topology of the target machine, and in return provides the following displays:

1. *Gantt Chart* A little different from AAARF's. The processors are represented as numbers on the horizontal scale and time is shown on the vertical scale. Light shaded regions represent tasks waiting for communication and dark shaded regions are those tasks being executed.

2. *Speed-up Line Graph* This compares six different heuristics for scheduling the task graph on a "fully connected machine". The speed up is displayed where the number of processors varies from 1 to n. Speed up is equal to the overall execution time running on p processors divided by the execution time of the program running on one processor.

   Obviously the performance of a parallel program represented as a task graph is going to be influenced by the connection topology. The fully connected topology will produce very different results to those for tree, mesh, toroid or other topologies.

3. *Critical Path* The critical path in a task graph represents a path from the beginning node to the terminating node which has the longest inherent delay. Therefore, it is along this path that the most significant improvements to performance may be made.

4. *Processor Utilization Chart* A bar chart is displayed showing the percentage of total processor time performed by each of the m processors. The information provided by this display is similar to that for AAARF's Kiviat chart, but it is not dynamic.

5. *Average Processor Efficiency Chart* a bar chart is displayed showing the parameter $E(i)$, the efficiency of processor i, where i varies from 1 to an upper limit specified by the user. This chart helps study the effect of using target machines with differing numbers of processors.

6. *Dynamic Activity Display* This display dynamically displays the assignment of task graph nodes to processors on an animated representation of the connection topology. This is a very interesting animation type, and one that warrants consideration for inclusion in the AAARF package.

Task grapher is not a solution in itself, and the figures produced by it are dependent on the accuracy of the input data. Therefore best results are obtained by using it in an iterative fashion, constantly updating the time estimates given to it.

2.4.14 Review Summary The packages described here are representative of the state of the art in parallel process animation. Some packages offer features that AAARF does not have, but these packages are process specific (e.g. MaTRIX) and do not offer the general applicability that AAARF does. Some packages offer similar general applicability capabilities as AAARF (e.g. ParaGraph) but do not offer AAARF's pseudo-real time capability.

2.5 Summary

This chapter has detailed the requirements for parallel animation visualization. The capabilities of AAARF and other packages have been reviewed and evaluated. No single package stands out as being clearly superior in all aspects of process visualization. AAARF's existing real time capability and versatility justify its continued development. Additionally, due to its object oriented structure, AAARF is ideally suited to further expansion. The direction of this expansion is discussed further in the following chapters.

III. Development of Improved Graphic Animations

This chapter presents the requirements analysis process for improved graphical animation of parallel algorithms. Based on this analysis, the functional requirements for the enhancement of AAARF animations are developed. Since the graphical animation of data is not restricted to the algorithm visualization field, the development of animation of data in other fields is also considered. The final section discusses how the information discussed in the previous sections may be applied to improve the graphical animation performance of AAARF, and the functional requirements are identified.

3.1 Current Animation Techniques

Before designing new graphical representation styles, it is important to understand the good and bad aspects of the existing AAARF animations.

AAARF produces animations in very close to real time where the monitored parameters of the algorithm change state at a relatively slow rate, thus allowing the animation to more accurately represent the current state of the process.

True real time animation of algorithms can rarely be achieved since the algorithmic time complexity of the processing required to produce the animation is often higher than that of the algorithm itself. Additionally, the algorithm is being performed on a fast parallel machine, while the animation is generally being produced by a more limited sequential processing workstation.

The "pseudo-real" time animations that are produced by AAARF give it distinct advantages over systems which only produce animations in a playback mode, particularly when being used as a debugging tool. Consider the case where the program of interest takes two hours to execute. Using an animation tool that provides only a playback facility e.g. Paragraph, the programmer is forced to wait two hours

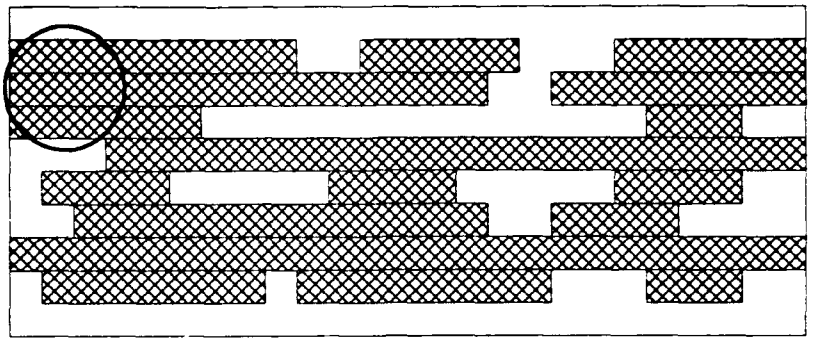
before he can view the operation of the program. Though not true "real time", AAARF allows simultaneous viewing and execution of a process, so that errors and inefficiencies can be identified much more quickly. As soon as the erros are found, the program can be halted and the errors corrected.

As mentioned in chapter 2, AAARF animations fall into two broad categories, windowed time intervals (WTIs) and instantaneous value animations (IVAs). The WTI are of order O(kN), since the animations must update the state of each node, while IVAs are only order O(k), since only one node state is updated, where k represents the algorithmic time complexity of the processing required per node. While this may not be a major problem for WTIs where the number of processing nodes is small, as the number of nodes increases the delays will become more and more intolerable. The solution to this problem is to reduce the number of nodes that need to be updated.

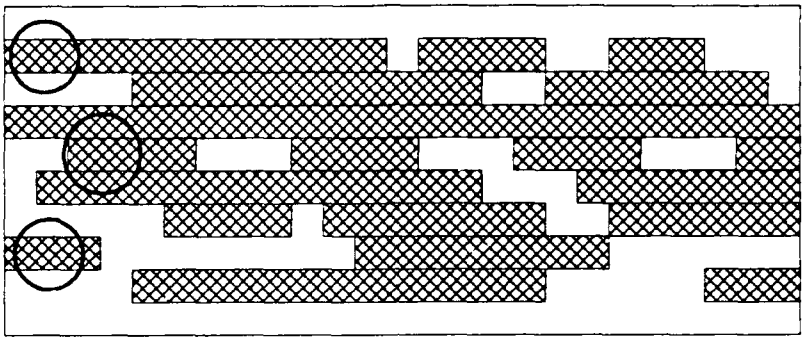
3.2 Ratio Value Animations

One approach to this problem would be to introduce a zooming feature i.e., some way of selecting a portion of the animation to observe in a WTI. However this approach is fraught with peril. How does one decide which nodes are of interest? If the display is animating the activity of a large number of nodes it may be impossible to distinguish those nodes which are of interest. Even if the activity of certain nodes is identified as being of interest, how does the user go about selecting only these nodes to be displayed. This would be easy if the nodes would were grouped together in the display as shown in figure 3.1a, where a simple zoom function could be employed. But they are more likely to be scattered throughout the display as shown in figure 3.1b. Additionally, because of the non- determinism of parallel applications, it would be impossible to predict the nodes of interest from one execution to the next.

A second approach would be to exclusively use IVAs, which are inherently quicker. This method too, is not without its disadvantages. By exclusively using



a. Interesting nodes grouped.



b. Interesting nodes scattered.

Figure 3.1. Location of nodes of interest

IVAs, the history aspect of the animations is all but lost.

This problem has an analogy in car racing. Imagine that AAARF is being used to animate a car race. How can the positions of the cars be identified? The lap data of each car for each lap may be examined, but using a WTI, we are restricted to observing only the most recent laps, and have no data regarding previous laps, as shown in figure 3.2. An IVA would merely tell us the cars' current speeds and the watermark would indicate the cars' top speeds to date, as shown in figure 3.3. Neither of these animations tells us enough. The solution is to have an animation which indicates the average speed of the car up to the current time. The higher the average, the higher the car's position. Figure 3.4 shows that car 23 is winning

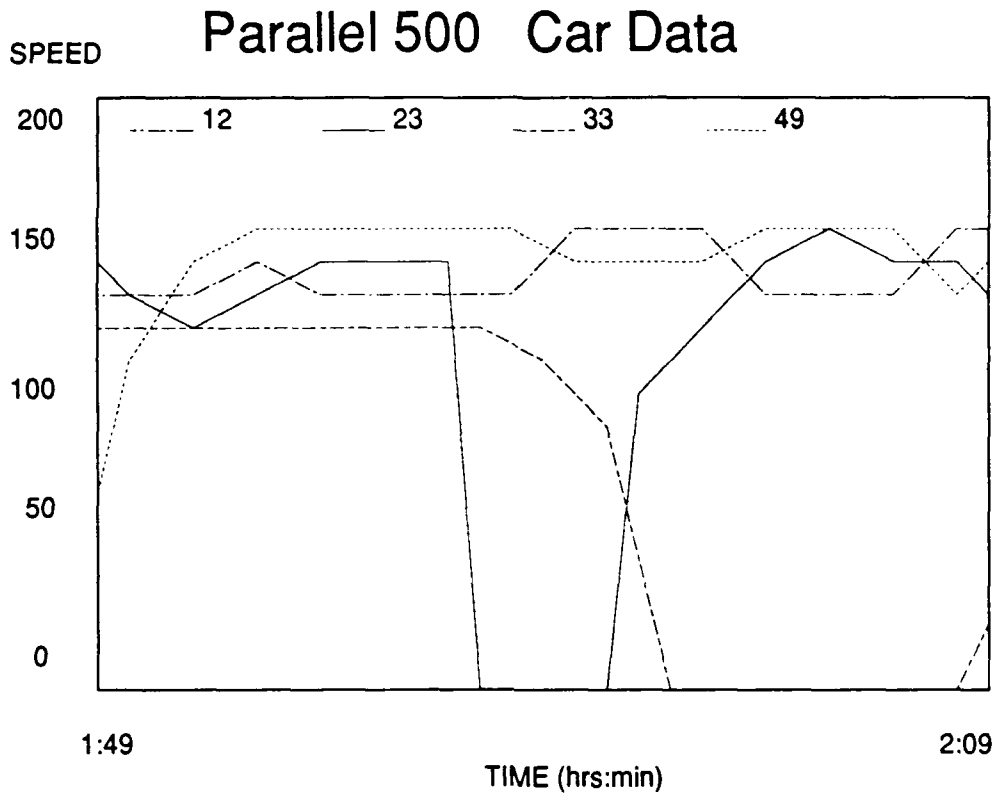


Figure 3.2. Windowed time interval of car performance

the Parallel 500. Coming back to the world of parallel performance observation, the application is clear. The use of some type of IVA to indicate the average performance of a processing node up to the current point in time is of great value. Noting of course, that they are not necessarily restricted to comparing the ratio of a parameter against total time. In fact almost any ratio can be analyzed, and the average value to date displayed. These animations are therefore called Ratio Value Animations (RVA). RVAs retain the simplicity of IVAs, but also capture an aspect of the performance history. Though they don't display exact values through time as WTIs do, they do represent a total history since the start of the execution, which WTIs don't.

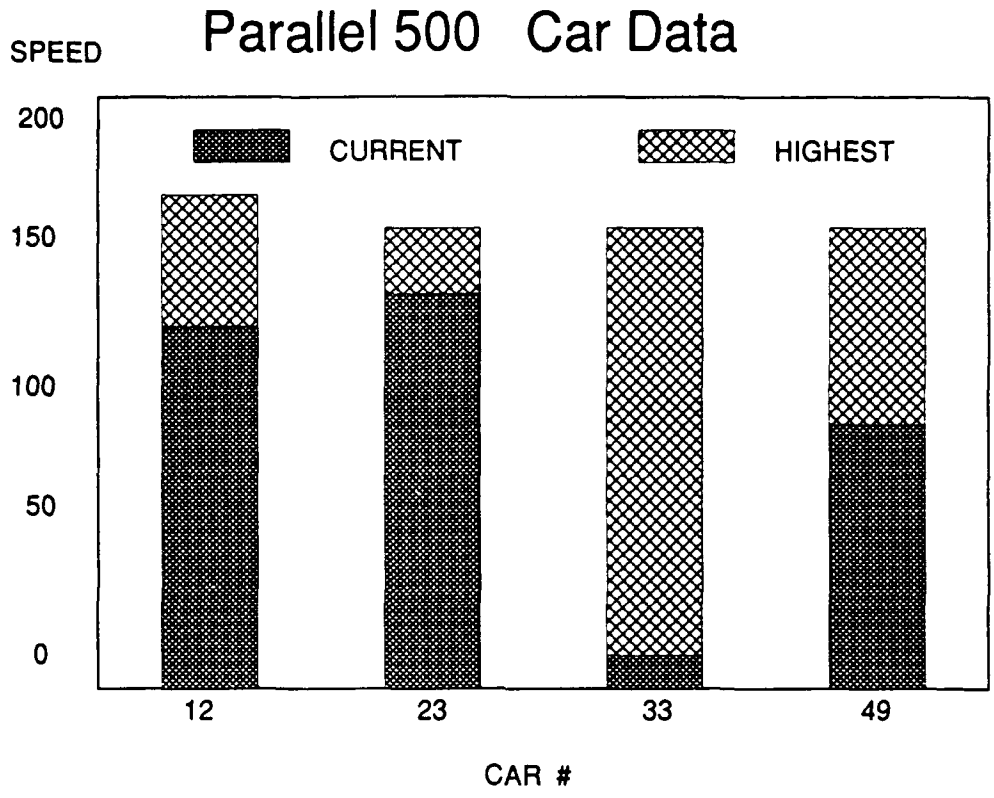


Figure 3.3. Instantaneous value animation of car performance

The range of ratios that can be calculated and displayed using the data produced by the current PRASE instrumentation process is vast. These parameters include:

1. utilization time/execution time
2. communications time/utilization time
3. communications time/execution time
4. receive blocking time/communication time
5. bytes sent/(send time + receive blocking time)
6. bytes sent/packets sent

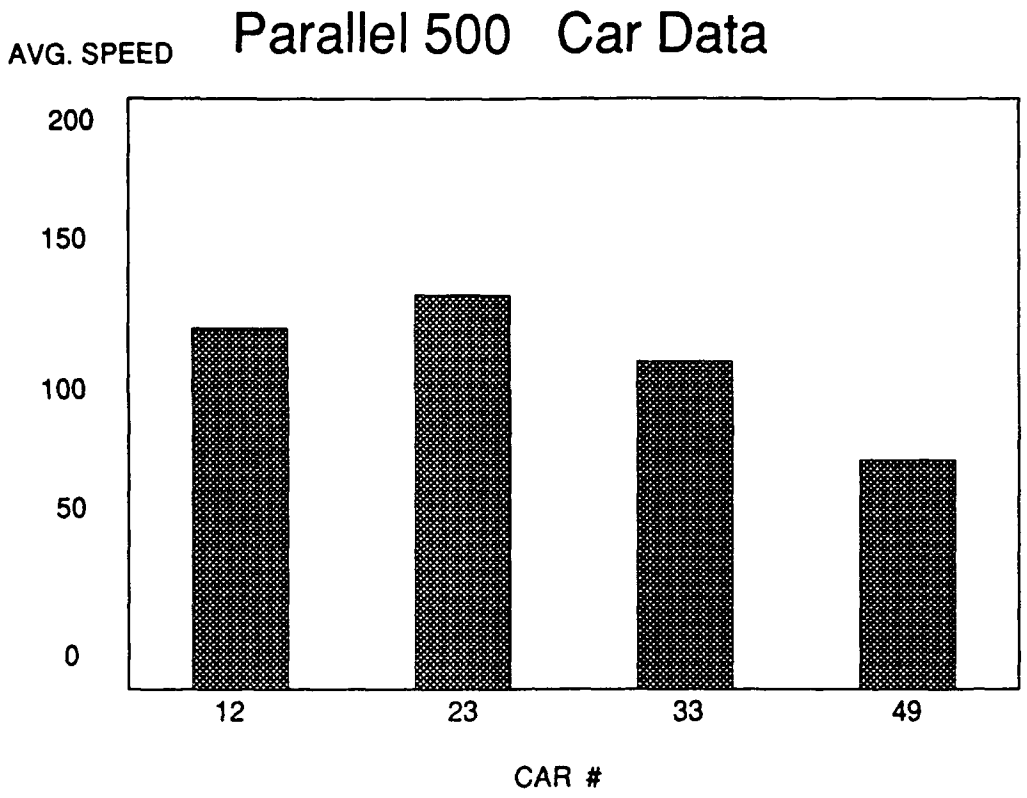


Figure 3.4. Average car speed animation

Thus RVAs are capable of displaying a history of parallel performance at a more suitable level of abstraction than either WTIs and IVAs, particularly where the amount of data to be displayed is large (i.e. systems composed of more than 32 nodes).

3.3 A Review of Graphical Representation Styles

The next problem is to decide the most suitable graphical representation format for physically implementing RVAs. To look only to other algorithm visualization packages for inspiration to improve the range and type of animations provided by AAARF would not only artificially limit the possibilities available, but may also transfer to AAARF limitations of the other systems. Therefore the search for possible

RVA formats is broadened to examine many forms of scientific visualization and human-machine interface design.

3.3.1 Introduction to Graphical Representation The representation of quantity seems to have developed at the same time as written language in human history, but only recently have graphic forms for representing quantitative ideas been developed. The concept of drawing upon the human visual system's capacity for perceiving and combining patterns, thus allowing the integration of large numbers of individual information items, seems to have flowered in the late 18th century(56). While interest in graphical representation has varied over the years since, in many respects, the field has not progressed beyond those early works(57). Furthermore, there has only recently evolved the concept of investigating the characterizations and forms of various graphical presentations by observing the performance of the user.

The last two decades have been a period of innovation, particularly in exploring methods of data analysis(25), due mainly to the "computer revolution". Modern advances in science and technology have had a major impact on the human-interfaces of complex systems such as those found in the aircraft cockpit or process control monitoring station(2). Ironically, though, these advances of technology often contribute to the increased complexity of the systems they are intended to support.(48, 60). Furthermore, there have been some efforts expended towards the establishment of standards for graphic presentation(52, 51). These efforts often appear to have had little impact and examples of poor graphics continue to appear regularly, ranging from those that are merely confusing to some that are deceptive (perhaps deliberately so).

Work on the graphical representation of quantitative information is found in the literature of a variety of fields, each of which promotes its own point of view. In reviewing this work, it is necessary to range over a broad spectrum of disciplines, from statistics, graphical arts and cartography to education and ergonomics. Within this

variety, though, or maybe because of it, there has developed no accepted theoretical basis for visual graphics. Rather, there exists only a number of generally agreed upon beliefs.

According to Andre and Wickens(2) the successful application of any display design principle relies on an accurate assessment of the frequency and nature of the task information processing requirements, both of which can be determined by appropriate cognitive task analysis procedures. Accordingly, multi-element display processing can be described by three sub-categories of task requirements. The requirements for the human operator are: (1) to correctly integrate the displayed information that must be combined, *information integration*, (2) to engage in a sufficient degree of parallel processing so that critical displayed channels are not neglected even as information in other channels is attended, *parallel independent processing*, and (3) to allow, when needed, information to be correctly extracted from a single variable without being distorted or biased, and without attention being distracted by information in other channels, *focused attention*.

Recent theoretical development in the display-cognitive interface has provided useful guidance for improving human information extraction and processing performance with multi-element display interfaces. In particular, the *proximity compatibility principle*(8, 12, 59) has been proposed as a means of guiding the development of display formats that support human information processing requirements. This principle asserts that the degree to which multiple channels of information are similarly displayed should directly correspond to the degree to which the appropriate task requires similar or integrated processing of these information channels(13).

3.3.2 Color and Shape The recent period of interest in graphics has seen a variety of new techniques and formats being proposed, particularly in the field of statistics. As computer technology has advanced, new graphical forms have been developed to take advantage of the computers capacity for data manipulation. Re-

search concerning the relation of graphic techniques and performance has also gained interest(57).

Principles and uses of color in applied environment have been described in a number of studies(7, 53). Many of these studies refer to issues such as the benefits of redundant color in target search or the advisable number of colors for coding. However it is uniformly agreed upon that designing with "aesthetic overindulgence" should be avoided(37), and that sound human factors principles should be followed when designing color displays(21).

In general, color has been shown to strongly affect the relationships between attributes (i.e., the perceptual organization) of a display. Gestalt principles of perception suggest that visual search is affected by perceptual organization(55). Hence visual search for items in a target color are organized according to Gestalt principles of proximity and similarity(11). Like-colored stimuli, for example, will more likely lead to a perception of proximity than will separately colored stimuli. In addition, same colored stimuli will more likely be remembered together than those of different colors(42). Accordingly, color can be used to group informational sources that need to be integrated, even when these sources are displayed in physically separate locations(32). Clearly, to the extent that color display variables can restrict the focus of attention to relevant information channels, performance will benefit(6).

Jubis and Turner(30) investigated the effectiveness of redundant color-codes and monochrome shape-codes for coding the operational states of scanpoints displayed on a CRT-displayed process-control diagram. User response-times were faster with color-coding than with shape-coding, and this relationship generally held true across all levels of display density and inspection load, with both search and identification tasks. Also, color-coding mitigated the detrimental effects of increased density and load. Coding did not affect response-accuracy on the search task, but on the identification task more errors were produced with shape coding.

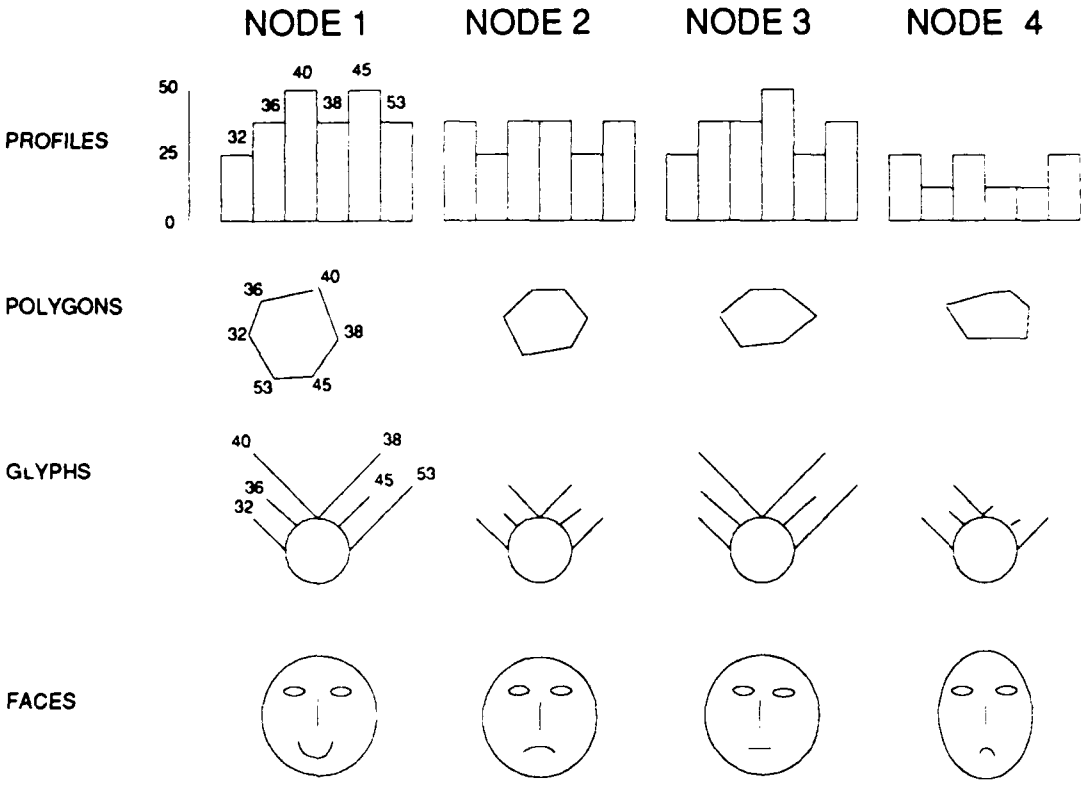


Figure 3.5. Iconic Representations

3.3.3 Iconic Representation One group of techniques employed for the representation of multivariate quantitative data, termed point representation, uses a particular symbol or icon to represent each point. The different techniques vary from each other primarily in the use of different symbols, ranging from profiles or bar charts to characterized representations of the human face. Some examples are shown in figure 3.5. The basic procedure is to represent each object of the set of objects to be compared as an individual graphical unit. The symbols appearance is determined by the values of the variables measured by the object. The objects of the set can then be compared by observing the set of symbols generated. Each symbol provides a single pattern of that data point. Such techniques are usually employed for preliminary cluster identification, detection of unusual data points, and, to a

lesser degree, trend identification. Further, most of these techniques are designed to make use of computer processing and graphic capabilities. This allows the creation of automated systems for producing graphics for initial or exploratory analysis of data. Jacob(28) reported on the use of cartoon faces, for the classification of personality profiles. An interesting use of faces in presenting basic statistical concepts to beginning students has also been reported(57).

The relationship between iconic representation techniques and performance has been explored in several studies. In a comparison of faces with polygons and digits for clustering data with nine variables by pattern recognition, faces were found to be significantly more accurate. Polygons were as fast, but were less accurate. Mezzich and Worthington(39) investigated pattern recognition of data by comparing seven forms of representation, including linear profiles, circular profiles(polygons), faces, linear and polar Fourier series, factor analysis in two dimensions, and an ordinal multidimensional-scaling. The factor analysis and multidimensional scaling representations were found to provide the best performance, with the polar Fourier icons next. These studies for the most part, have been comparative in nature, there has been little investigation of the relationship between the graphic characteristics of a particular type of representation and the performance with that graphic.

3.3.4 Graphical Presentation Effectiveness Although new methods and forms of the graphical presentation of data have been developed, there has been relatively little empirical evidence established for the preference of particular methods in a given circumstance, or for the use of particular features of a graphic type to best serve the function intended. It is interesting to note that Charles Babbage, the forefather of computing machinery, was one of the first to express concern for the presentation of data and its effect on the observer(57).

Bar and pie charts continue to be the most frequently used graphs in most fields. Many software packages use them, e.g., Quattro Pro and Lotus 123. And

yet, there is still not a solid basis for using or rejecting (completely) one form or the other, or for selecting the features that will convey the desired information.

Considering the widespread use of graphics for the representation of data and the availability of computer programs to produce displays of data, though, there has been relatively little work to indicate which types or characteristics of a given type of display are best for fulfilling a particular purpose. For example, Cleveland and McGill(14) demonstrate the advantage of showing the difference between five approximately equal portions of a whole, by using dots rather than circle charts. However, we should ask whether it is the difference we actually want to convey, or whether the approximate similarity more important.

3.4 Animation Requirements Specification

The animation types described in the previous sections all have individual merits, each suitable for displaying particular types of information. This section evaluates the range of animation types for use as a starting point for this research, and explains the reasoning behind the final selection of a format for implementing RVAs.

3.4.1 Discussion The first step in the evaluation is to consider the complexity of generating the display types. If the display type is complex to produce, then much of the advantage gained by raising the level of abstraction is lost. Therefore, the more complex display types are immediately rejected from consideration. Many of the graphical display types described in the previous section have not been used to display data related to parallel algorithm performance. A discussion of the best alternatives follows:

1. *Iconic Faces* Iconic faces represent a method of quick ball park assessment of any aspect of performance. Iconic faces may represent intervals of performance. e.g. 0 to 20% may be represented by a happy face, while 80 to 100% may be

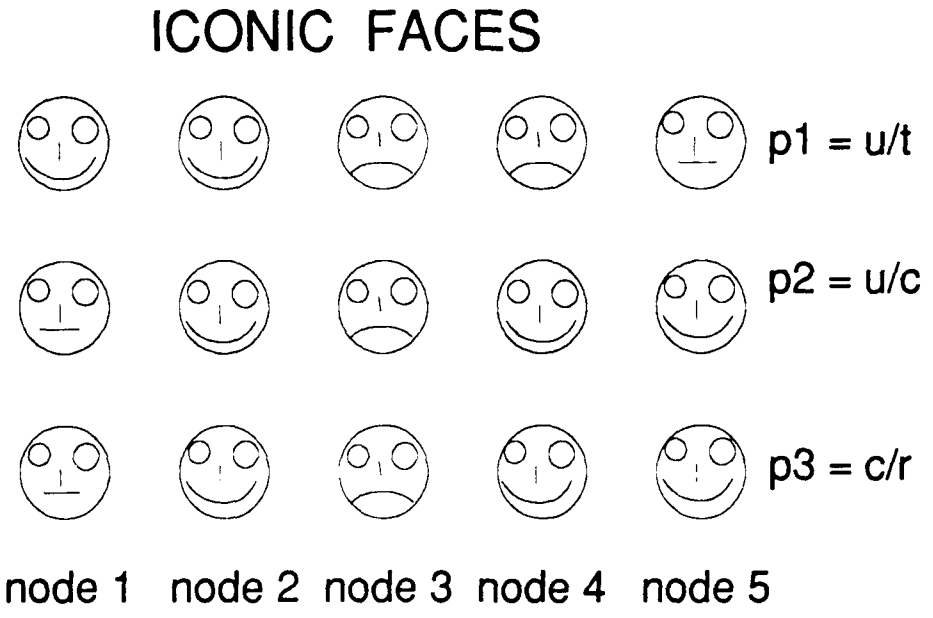


Figure 3.6. Iconic Faces

represented by a sad face, a simple example is shown in figure 3.6. Iconic faces are simple to produce and update, making them suitable for real time animation. One drawback is that the user must know in advance the appropriate intervals to assign to the different faces, since all nodes may produce figures within the 0 to 20% range and therefore it would be more desirable to set a happy face to represent 0 to 4% and a sad face to represent 16 to 20%, than to have all nodes represented by a happy face. A second problem with the use of iconic faces is that of recognition. The tests by Mezzich and Worthington(39) indicate that iconic faces have poor recognition performance. Consider the difficulty in recognizing a dominant trend when confronted with a display of 512 faces.

2. *Polygons* or Multiple Axis Plots. Compact, and able to display great amounts of data, these may be suitable to use one per processing node, as shown in figure 3.7. By making use of the proximity principle, the user would be able to quickly identify the relationship between different parameters. The use of color and known intervals makes them easy to use and interpret, but once again a problem of scale is encountered. A display containing a large number of these plots would be very difficult to interpret. Additionally there is a problem with scaling each of the axis to accurately indicate the relationship between parameters. One axis may be indicating the ratio of time spent processing versus time spent communicating in the range of 10 and 40%, while a second axis may be representing time spent processing versus time spent idle in the range of 80 to 100%.

3. *Bar Chart* Once again these could be used to show several parameters of a single node, with many bar charts representing multiple nodes, but bar charts are not efficient in doing this. As with multiple axis plots the problem of scaling for multiple parameters is an issue. In fact, even more so, since all bars are read against a single scale on the vertical axis, as shown in figure 3.8.

4. *Graduated Circle Charts* These could be used in much the same way as iconic faces. However the use of color, in addition to size allows the color proximity principle to come into play, resulting in better recognition performance than can be achieved with faces. An example is shown in figure 3.9. However, the difficulty of selecting scales is still a problem.

3.4.2 Analysis The simplicity of generation for the iconic faces and graduated circles make them very strong candidates for the implementation of RVAs. However each is not without its disadvantages:- recognition difficulties in the case of iconic faces and scaling problems for both. It is concluded that the recognition problem for the faces could be overcome if colors rather than facial features are used, e.g. red 0

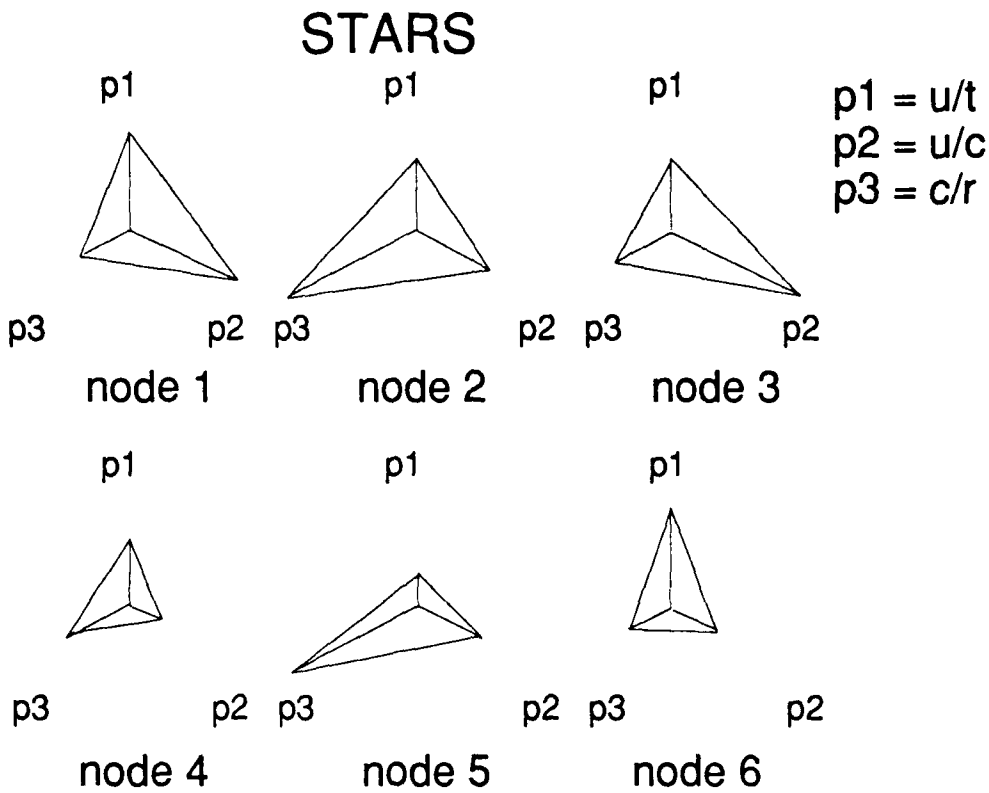


Figure 3.7. Polygons or Multiple Axis Plots

to 20%, blue 80 to 100%. The result is an animation that is, in essence, graduated circles that don't change size, but rather change color.

A simple test (detailed at Appendix B) is designed to assess the effect on recognition, that changing sizes had, and whether it justifies the more complex code. The test consists of displaying a series of images to a user, some containing circles of different size and color, the others containing circles of differing color only. The user is asked to assess the most prolific color in each display. The test indicates that changing sizes are actually a detriment to recognition rather than an enhancement. As a result of the test findings, this combination of iconic faces and graduated circles is used to implement RVA's.

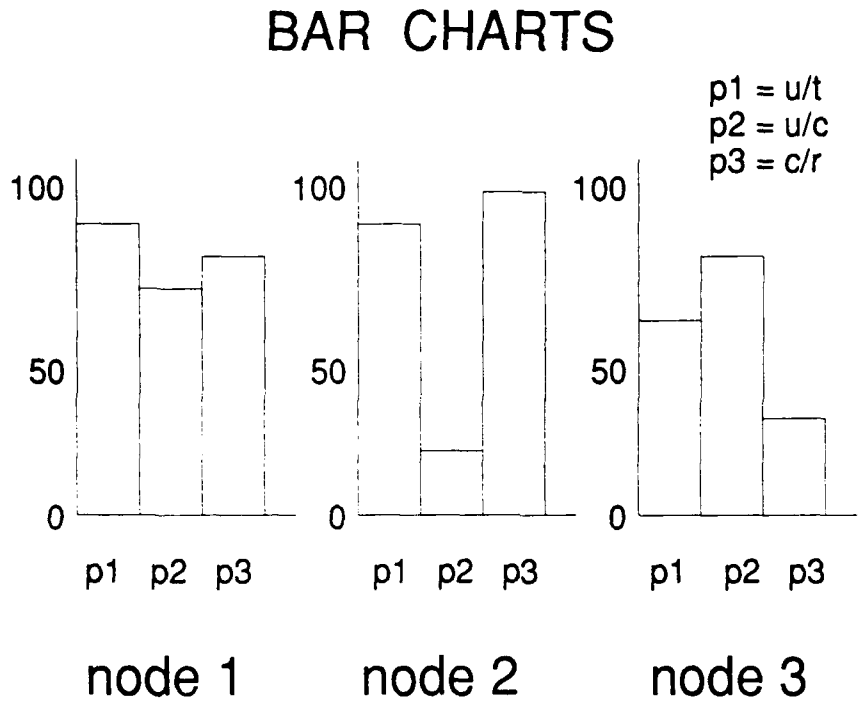


Figure 3.8. Bar Charts

3.5 Implementation of Ratio Value Animations

The final step in the development of this new animation is to decide on how it can be implemented. The two major considerations in this step are, the layout of the circles in the animation so that it is possible to effectively relay the information about a large number of nodes, and secondly, the provision a considerable amount of user control to overcome the scaling problems mentioned earlier.

3.5.1 RVA Layout The existing "Animation" display is a useful starting point for prototyping the RVA. This display has a circle representing each node in the system, which changes color depending on the activity occurring at the node. A rapid prototype of the RVA can be implemented by simply changing the code which

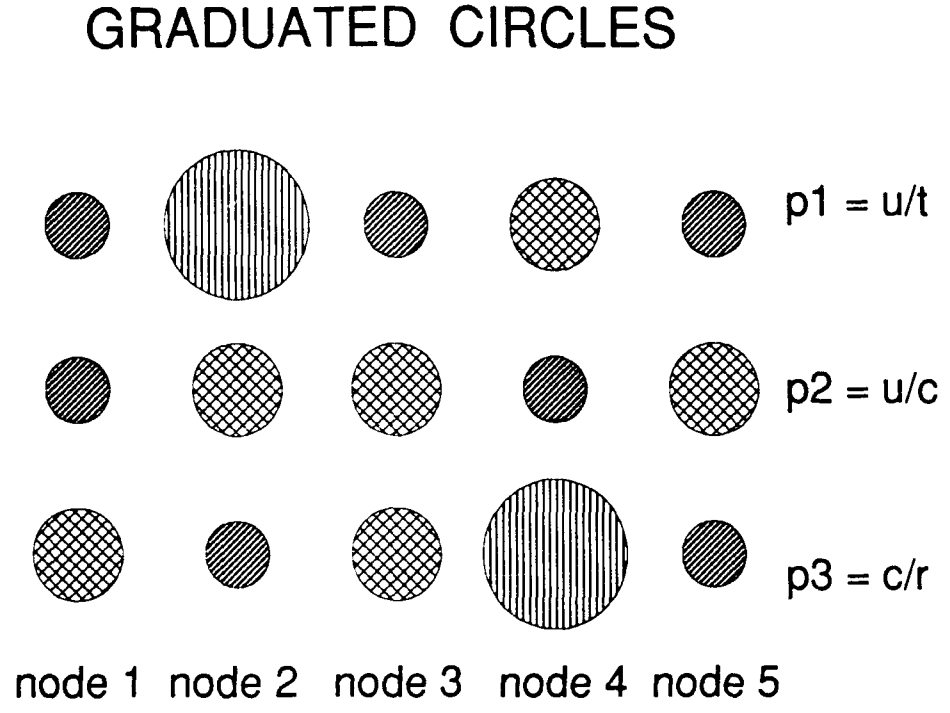


Figure 3.9. Graduated Circle Charts

controls the color of the node, and adjusting the legend appropriately.

Figure 3.10 shows the first RVA prototype. The ratio being measured is active time/total execution time. Five ratio intervals are provided, each represented by a different color. The colors are assigned as follows (based on the rainbow):

- red 0 - 20%
- brown 20 - 40%
- yellow 40 - 60%
- green 60 - 80%
- blue 80 - 100%

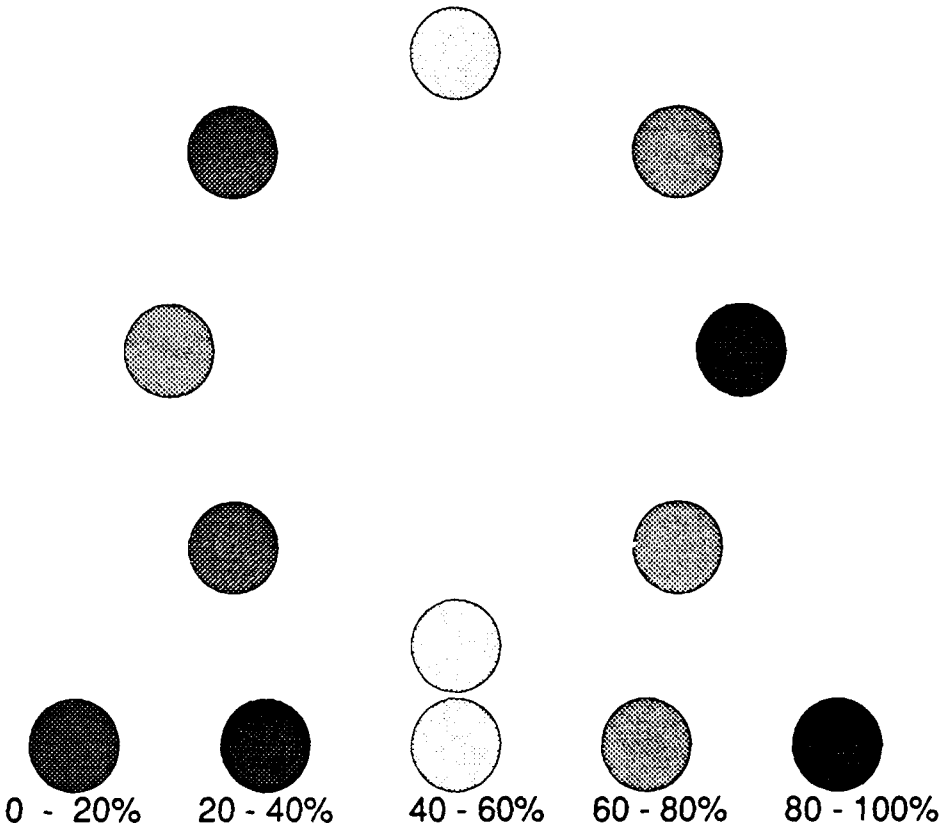


Figure 3.10. RVA First Prototype

This prototype is an effective visualization for up to 16 processing nodes, but it is less efficient for larger groupings due to great amount of wasted space in the center of the display. While this format is useful for the "Animation" display which shows communications between nodes, it is less than optimal for RVAs. A more efficient use of space is obtained by arranging the circles in a compressed polygon as shown in figures 3.11 and 3.12. This type of representation also provides a more suitable format for identifying color ratios.

3.5.2 RVA Control Having decided on an appropriate format there is still the issue of scaling difficulties. The animations shown in figures 3.10, 3.11 and 3.12 are ineffective if all processing nodes are operating in the same interval, e.g. all

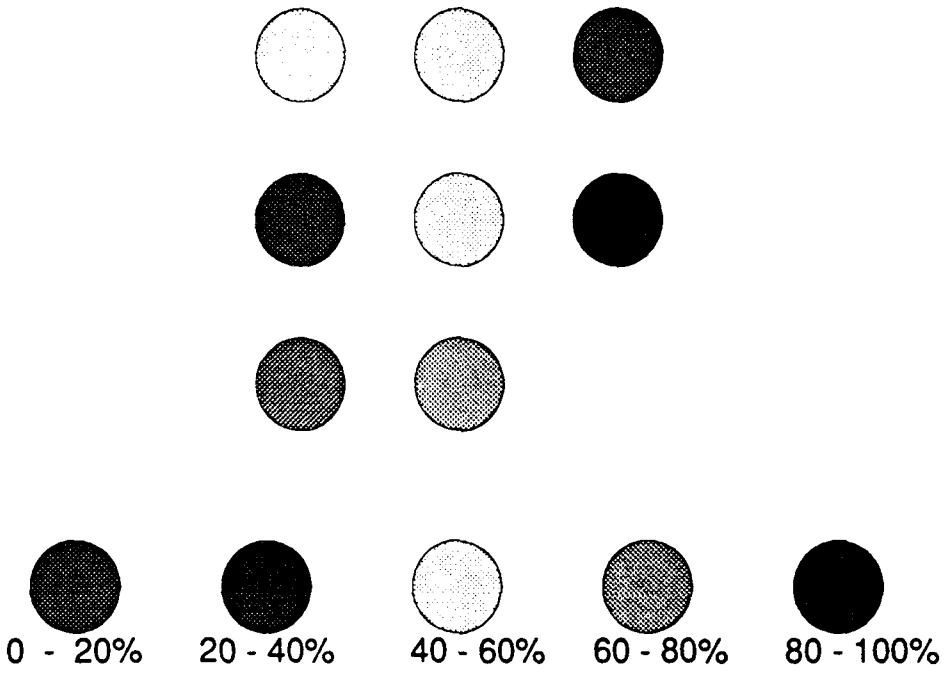


Figure 3.11. RVA Second Prototype - 8 nodes

have a ratio in the 60 - 80% interval and are therefore all shown as green circles. There is clearly a requirement for improved accuracy.

This improvement is gained through the provision of user control over the ratio intervals and the adoption of an iterative approach to RVA visualization. The user is able to adjust the ratio intervals that are displayed. So, if the user finds that the utilization ratios of all nodes falls between 20 and 60%, he can adjust the intervals displayed, e.g. red 20 - 28%, brown 29 - 37%, yellow 38 - 45%, green 46 - 53% and blue 53 - 60%. By taking this iterative approach, the user can obtain a high degree of accuracy.

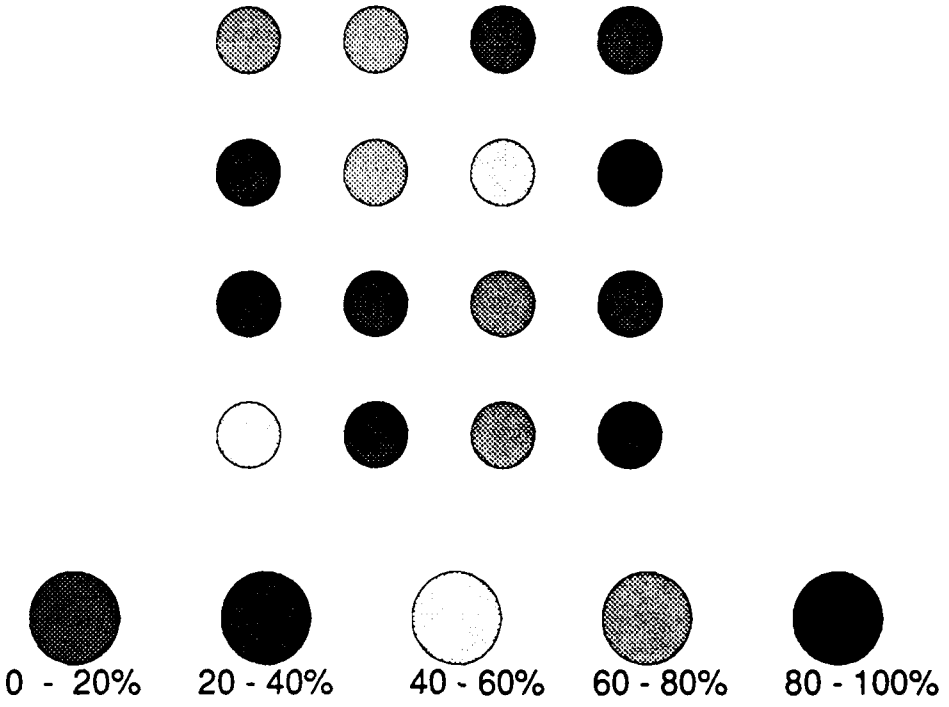


Figure 3.12. RVA Second Prototype - 16 nodes

3.6 Summary

The field of Graphical Representation is diverse, and generally without any formal prescription as to which type of representation best suits a given application. The challenge of formalizing the assessment of representation capabilities has attracted considerable attention over the last decade, as the use of computer based displays has grown, but most proposals remain subjective and domain specific.

A new form of display called Ratio Value Animations has been designed using the findings of a number of researchers and the inherent physical time constraints as guidelines. RVAs combine the high recognition performance of graduated circle representation with the simplicity of structure of iconic faces.

IV. Development of the AAARF Expert User Interface

This chapter presents the requirements analysis design and implementation process for the expert system to be used as a user interface tool for AAARF. First a brief introduction to the terms and expressions used to describe expert systems is presented. The following section reviews the basic principles of expert system structure and operation and the alternative methods available. A more detailed discussion of these topics may be found in Appendix A. The final section discusses the use of these techniques in relation to the AAARF user interface.

4.1 Introduction

The purpose of this section is to introduce the terms and definitions for the subject areas that form the basis for this research.

4.1.1 Definition Knowledge-Based Expert System Historically the term "Expert System" was liberally used to describe many artificial intelligence (AI) applications. The term expert system is being used to imply that the system has captured on computer the knowledge of a human expert. However, many subsequent AI applications have been used to assist in difficult decision making processes but could hardly be described as replacing a human expert. These systems are more accurately defined as "Knowledge Systems." Leading expert system researcher, Professor Edward Feigenbaum, has defined an expert system as:

> ...an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution. Knowledge necessary to perform at such a level, plus the inference procedures used, can be thought of as a model of the expertise of the best practitioners of the field. The knowledge of an expert system consists of facts and heuristics. The "facts"

constitute a body of information that is widely shared, publicly available, and generally agreed upon by experts in a field. The "heuristics" are mostly private, little-discussed rules of good judgement (rules of plausible reasoning, rules of good guessing) that characterize expert-level decision making in the field. The performance level of an expert system is primarily a function of the size and the quality of a knowledge base it possesses.

4.1.2 Principles of Expert Systems The purpose of this section is to briefly describe the basic structure and operation of an expert system, and to perhaps dispel some of the myths surrounding these structures. If additional information is required, the reader is directed to the references which provide a more detailed discussion of the subjects. There are two basic building blocks common to all expert systems: *knowledge representation* and *heuristics.*

Knowledge RepresentationKnowledge representation may be thought of as the method used to impart knowledge of the domain of concern to the computer. Then given an appropriate reasoning capacity, that knowledge can be used to allow the system to adapt to and exploit its environment(9, 26).

Heuristics Heuristics are rules used to focus on key aspects of particular problems and to manipulate symbolic descriptions in order to reason about the knowledge they are given. Conventional programming languages use algorithms oriented towards numerical processing and symbolic processing, while knowledge systems use heuristics for symbolic processing only. While the algorithms and heuristics are identical in implementation, they differ significantly in user specification due to the structure of the high level fourth generation languages. A knowledge system may contain thousands of heuristic rules.

4.1.3 Approaches to Knowledge Representation The last decade has seen a consolidation of the methodologies employed in the knowledge representation field, and a formalization of the theory associated with the differing methods. Formal

semantics are now an obligatory accompaniment of the description of a novel KR system. Unfortunately this increase in KR theory has come at the expense of experimental development work in the field(9).

Today there are four representational paradigms (data structures) which seem to predominate:

- *semantic networks*
- *frames*
- *predicate logic*
- *object-attribute-value (OAV) triplets*

4.2 Languages

AI languages are programming languages that exhibit one or more of the previously mentioned characteristics. These languages are used to construct Expert Systems. There are a large number of AI language packages, sometimes called expert system shells, *currently available. Many of these are simply variations of established AI languages with added features (eg. windowing systems, advanced debuggers and macros). Among the most well known languages are Lisp, Prolog, and Scheme which are backward chainers and Clips which is a forward chainer. Final versions of expert systems can also be implemented in standard languages (e.g. C++, Ada, Pascal, etc.) for improved efficiency, effectively using the AI language as a prototyping tool.*

4.3 AAARF Expert User Interface

The AAARF expert interface is intended to provide an advisory system to suggest possible animation configurations to the user based on his requirements.

From the users point of view, the selection of AAARF animations for the visualization of an algorithm is dependent on the following:

- *the host machines capabilities,*
- *the algorithm type,*
- *the areas of operation of most interest to the user, and*
- *the user's personal preferences*

The problem of selecting appropriate animations is well suited to solution by the use of a knowledge based system. By asking the user a series of simple questions, the system can establish the users needs. By using a forward chaining system, the interrogation of the user can be structured in a logical manner, and then the users responses can be exhaustively matched against the heuristics of the system.

4.3.1 CLIPS The forward chaining language chosen for the implementation of the AAARF expert user interface is CLIPS. CLIPS(15) was developed by the Artificial Intelligence section of the Mission and Analysis Division at NASA/Johnson Space Center. CLIPS is freely available to US Government Organizations and their contractors. CLIPS is available outside of the US Government and its contractors through COSMIC, the NASA software distribution center. The cost of CLIPS is about $350 for both source code and MS-DOS or Macintosh executable. Unlimited copies can be made after purchasing CLIPS through COSMIC. The same source code will run on nearly any machine which can support an ANSI or K&R C compiler. The ready accessibility and transportability made CLIPS the obvious choice for use as the basis for the AAARF expert system interface.

4.3.2 Knowledge Base The aim of the AAARF expert interface is to enable an inexperienced user to make informed decisions regarding the optimal configuration of AAARF for his application. It is therefore essential the expert interface contains the knowledge of an experienced (expert) AAARF user. This knowledge pertains to the suitability of the available AAARF animations to specific purposes. This

Table 4.1. Table/Knowledge Matrix

DISPLAY	INTEREST	O/P COLOR	VIEW	DISPLAY GROUP	ALGVIEW
GANNT	UTIL	COL	ALL	GROUP 1	
FEYNMAN	UTIL		ALL	GROUP 1	
COMM. STATS	INT/EFF		SINGLE		
COMM. LOAD	EFF		ALL		
MESS. LENGTH	EFF	COL	ALL	GROUP 1	
MESS. QUEUES	EFF		ALL	GROUP 1	
KIVIAT	UTIL		ALL	GROUP 1	
UTILIZATION	UTIL		ALL		
ANIMATION	INT	COL	ALL	GROUP 1	
QUEUE SIZE	EFF/INT		SINGLE		
RVA	ANY	COL	ALL		
TREE					TREE
BUBBLE					BUBBLE
STICK					STICK
RAINBOW		COL			RAINBOW

knowledge is represented by the matrix in table 4.1 below, and the inherent logic of the heuristics.

The matrix shows that the animations are assessed by a combination of five parameters:

1. Recommended Interest: *There are three possible categories here, processor utilization, processor interplay, and processor efficiency.*
2. Recommended Output Color: *This is a qualification by exception, most AAARF animations are suitable for production on monochrome or color monitors, however some exhibit very poor performance unless produced in color.*
3. Recommended View: *This parameter distinguishes between those animations which show some aspect of performance for all processing nodes and those which represent a single node only.*

4. Display Group: *Animations that fall into the category of display group 1 are best suited to the visualization of parallel implementations using up to sixteen nodes. Animations not designated a display group are equally suitable for configurations of all dimensions.*

5. Algview: *Not yet fully implemented, this parameter currently serves as a simple matching parameter for the algorithm animations and the expert system conclusions.*

4.3.3 Heuristics The heuristics are designed to extract from the user what he knows about his application. This information is then processed to transform it into information which pertains to the parameters for the selection of AAARF animations. The questions are of the multiple choice variety and are designed to be presented in a logical order, extracting the information from the user in a natural progression. The questions ask the user for basic information concerning his application. Of course the question answers must be general enough to apply to any application, therefore a "none of the above" option is available where the user has no specific response. In addition to asking about application information, the questions also ask the user for any personal preferences he may have with regard to the animation displays.

```
( defrule quest11 ""
?rem <- (ask-question)
(or (color-monitor yes)
(color-monitor y))
(not (preferred-color ?))
=>
(retract ?rem)
(printout t "Do you prefer your display to be : "t)
(printout t " a. full color "t)
```

(printout t " b. monochrome "t)

(printout t " c. unknown "t)

(assert (preferred-color =(read))))

The aim of the heuristics is to weight the user's responses appropriately and combine them wit.. the knowledge implicit in the system rules. Based on the objective responses of the user, the expert system makes decisions regarding the user's requirements. These decisions are then compared with the user's subjective responses. If a conflict exists, the user's preferences are weighted accordingly. A simple example of this would be where the user would prefer *to see a rainbow representation of data to be sorted, but has stated that he is using a monochrome monitor. The knowledge base shows the rainbow representation to be relatively ineffective on a monochrome screen, the users preference is not simply disregarded, but rather it is weighted appropriately, probably in the area of 60% depending on other inputs. Where no conflict occurs, the users preference is given a 100% weighting.*

The final result of this sifting of data is a set of 5 parameters each with an associated weighting. These are then compared with the animation characteristics and the worth of the animations is graded.

4.3.4 Code Structure The expert user interface code is comprised of six phases, shown in figure 4.1 and described below. A full listing of the expert user interface code may be found in Volume II.

1. Phase 1 - Declarations *This phase simply defines the parameters to be evaluated in selecting the appropriate set of displays and also defines the valid answers to the questions that the user may be asked to respond to.*

   (define valid-combinations ""

   (combine key-interest)

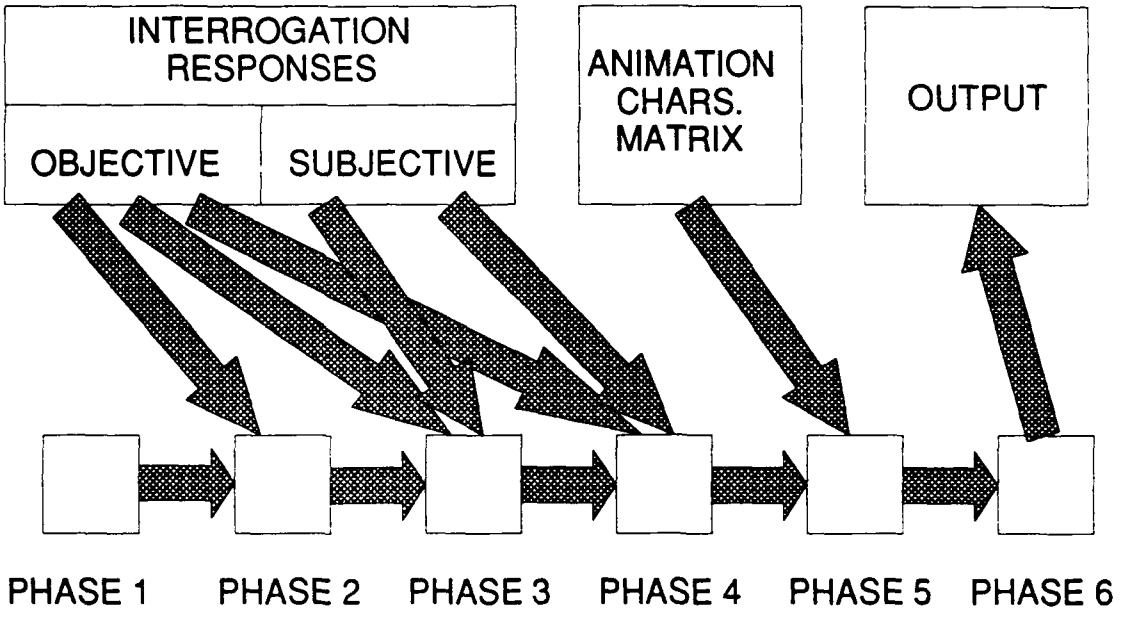


Figure 4.1. Interface System Phases

(combine the-op-color)

(combine display group)

.

.

.

2. Phase 2 - Choose Qualities *These are the rules which directly interpret the users responses to the questions. The users responses are used to apply weightings to the parameters used by the later phases.*

(defrule choose-interest-for-efficiency ""

(choose qualities)

(or (has-special-interest yes)

(has-special-interest y))

(special-interest c)

=>

(assert (key-interest a 80 = (gensym))))

3. Phase 3 - Recommend Qualities *The rules in this section cross correlate the results of Phase 2. This is necessary since many parameters are affected by the answers to more than one of the questions. An example of this is the* best view *parameter which is influenced by the users main interest and the type of interplay between processors he wishes to view.*

(defrule preferred-color-recommended ""

(recommend-qualities)

(preferred-color ?acolour)

(the-op-color ?acolour ?per ?)

=>

*(assert (recommended-op-color ?acolour = (/(*30 ?per) 100) = gensym))))*

4. Phase 4 - Default Qualities *If there is no logical solution possible based on the objective data and the user has shown no preference towards a particular parameter, a default value is assigned in this phase, based on those animations which provide a broad amount of data.*

(defrule use-comms-if-no-key-interst ""

(default-qualities)

(not (key-interest ? ? ?))

=>

(assert (recommended-interest ?interest b 100 =(gensym))))

5. Phase 5 - Select Display *The interpreted data and default values are compared to the attributes of the animation types provided by AAARF and the certainty for using each is calculated.*

(defrule recommend-gantt ""

(select-display)

(recommended-interest a ?per1 ?)

(recommended-op-color a ?per2 ?)

(recommended-view a ?per3 ?)

(display-group group 1 ?per4 ?)

=>

(assert (display Gannt = (min ?per1 ?per2 ?per3 ?per4) = (gensym))))

6. Phase 6 - Remove Marginal Responses *This section evaluates the strength of recommendation of each animation as evaluated in Phase 5 and discards those with less than 30% certainty. This is done for two reasons. First, to simply reduce the number of options presented since no more than four can be used simultaneously. And second, and perhaps more importantly, recommendations with less than 30% certainty are more accurately described as guesses.*

(defrule eliminate-marginal-responses ""

(remove-marginals)

?rem <- (display ? ?per ?)

(test (< ?per 30)

=>

(retract ?rem))

The code has been constructed with maintainability in mind. The addition of new animations is easily accomplished by modification of Phase 5, due to the object oriented approach to the design of the code. Should new parameters of interest be required to be introduced, this too can be easily accomplished with little effect on the existing code.

4.4 AAARF Expert User Interface Implementation

Having designed and tested the expert user tool as a stand alone package, it is necessary to integrate it with the AAARF system. In keeping with the overall flavor of the AAARF user interface, the expert tool is made available as a menu selection from the AAARF main menu.

When the expert tool is selected from the menu, a window is opened, in which all interaction between the user and the tool takes place. The questions are presented to the user, who makes his selection by pressing the key corresponding to the desired selection.

4.5 Summary

Expert (or knowledge-based) systems are designed to capture the knowledge of experts in the domain of their application. There use is intended to supplement the knowledge of a novice with that of an expert. These systems can be structured in a number of ways, using a variety of knowledge representation formats, and rule processing techniques.

A number of packages are commercially available which make use of the knowledge representation and rule processing techniques. These packages come in a variety of formats, ranging from bare bone languages to total automated development envi-

ronments. The AAARF expert user tool employs a forward chaining language called CLIPS. CLIPS is a low priced package that is readily available and is suitable for use in a number of operating environments.

The AAARF expert tool takes the users knowledge of his application and translates this to visualization display parameters, in a six phase process. The tool is made available to the user as a menu option from the AAARF main menu.

V. *Operating Environment Requirements Analysis*

This chapter discusses the restrictions placed upon AAARF by the current operating environment and suggests that these may be overcome through the use of an alternate windowing system. With this in mind, the capabilities of the X Windows system are briefly reviewed and the possible benefits of implementing AAARF in X Windows are discussed. The next section looks at the work involved to transport AAARF from Sunview to X Windows, and finally the prototype X Windows version of AAARF is evaluated.

5.1 Introduction

AAARF is currently configured to operate under the Sunview environment(54). The Sunview environment restricts AAARF's use to Sun3 and Sun4 workstations and also restricts AAARF from an output perspective. The number of resources required by AAARF under the Sunview environment is large. The SCP algorithm class for example, with its large tree view requires 24 MB of real memory, mainly for the large canvasses allocated for the views. This also limits the number of algorithm classes that can be simultaneously active to one! The number of window devices required when four algorithm classes are active exceeds 128(62). These resource requirements exceed the available resources on most available workstations since many Sun workstations are only configured with 8 MB of memory and 64 window devices. The additional window devices can be added, but this adds additional overhead in the UNIX kernel.

5.2 X Windows

An alternative environment, that is worth considering for the implementation of AAARF is X Windows, as suggested by Williams(62). The X Window System is a large and complex system. Much of that complexity comes from X Windows

requirement for compatability with almost all types of computer graphics displays available. X Windows also attempts to provide a complete graphics system - complete enough to create windowing interfaces and handle graphics intensive page design or computer-aided design (CAD) packages. The X Windows System shows great promise in solving a seemingly intractable problem: how to provide a common interface across many different computers that are running a number of operating systems with a number of different displays. Almost every major computer vendor is committed to the X Window System(29).

X Windows, by being operating-system independent, encourages the portability of software. The standard X Windows C library routines consist of the same set of calls on every machine running X Windows(29). Thus, the interface code ports directly from one machine to another. The X Window System has been adapted to several different operating systems, including UNIX, MS-DOS, A/UX, and VMS. X Windows allows portability among different hardware environments(46) and has the ability to allow AAARF to run acceptably on minimally capable machines while allowing it to benefit from higher performance systems.

X Windows is made up of several components:

1. *Xlib* contains about 300 routines that map to XProtocol requests or provide utility functions.
2. *X Toolkits* are program subroutine libraries that can make programming easier. Essentially, toolkits are preprogrammed graphics routines written in C. The X Toolkit provides a standard set of objects or "widgets",such as menus, window frames, scroll bars, panels and other graphical tools that can be assembled into a graphical user interface. The X Toolkit sits on top of the Xlib library, providing a higher-level access to graphic interface routines than do the Xlib functions(47)

3. *The X Network Protocol* defines data structures used to transmit requests between clients and servers. Technically speaking the X network protocol is an asynchronous stream-based interprocess communication instead of being based on procedure calls or a kernel-call interface(29).

5.3 Benefits

The structure of X Windows allows resources to be allocated when they are needed, instead of allocating all resources at initialization. Each item in a display under X Windows does not require a specific entry in the device tables. Each view can be contained in a separate window, allowing it to be allocated and deallocated on demand.

Another major benefit of using X Windows is that it has built-in support for distributing the processing and display activities over several systems connected by a network. This can be used to ease the load on the system driving the display. In this way, the different components can be placed on systems that are best suited to the task. The use of X Windows can also help ease the display speed problem by utilizing multiple displays on different systems: one X Windows client can connect to multiple X Windows display servers.

5.4 Adapting AAARF to X Windows

The adaption of AAARF to X Windows was begun in early 1991 by Williams(61). The code produced duplicated most of the Sunview based AAARF functions with X Window constructs. By using the available libraries and tools, the X Window code is kept much simpler than its Sunview counterpart. However, the code produced is neither executable or documented.

The efforts of this work are dedicated to interpreting the inherited code, documenting it, and bringing it to an executable state.

5.4.1 Compiling the X Windows Code The first lesson to be learned when using X Windows is that porting packages from one operating system or architecture to another is not as easy as it sounds. Yes, X Windows functions are coded the same for each system, but to enable these functions to operate correctly requires a considerable amount of ancillary work.

When compiling an X Windows application, the system configuration parameters must be included in all directory makefiles that are to be used. These configuration parameters are not insignificant, amounting to nearly 200 lines of code. To aid this process, X Windows provides a makefile template called Imake.tmp. Before the template can be used, platform specific parameters must be set in the appropriate .cf configuration files. Site wide parameters are set in the site.def file. If any parameter is changed then a full rebuild is required. X Windows applications cannot be compiled successfully until these parameters are set correctly.

Using the Imake template is a relatively simple task once the configuration parameters have been set. The first step is to write a standard makefile for the compilation of the relevant files, called Makefile. The next step is to initiate X Windows and use the command, "make Makefile", which invokes the Imake template and adds the necessary configuration parameters to Makefile.

After dealing with the X Windows configuration issues, the next task is to ensure compatibility of the rest of the code with the host machine. Though the X Windows functions are transportable after the configuration parameters have been set, other parts of the AAARF package still require rebuilding.

The X Windows version of AAARF makes use of two customized libraries, libAAARFutil.a and libAAARFcommon.a. The archived files contain functions common to many of the AAARF processes. It is essential that these files are rebuilt before the package is used on a new platform. This may trap the novice user. The error messages provided by the C compiler, if compilation is attempted before rebuilding the archive files, are less than informative in identifying the problem. The

compiler provides an error message which says that the library files are missing a table of contents (TOC), and that a TOC should be added to each using the "ranlib" command.

The ranlib command can successfully be applied to the libraries, and a table of contents will be added to each. The next time the X Windows AAARF package is compiled, the compiler produces a message that indicates that the host machine has run out of memory while trying to load the library files. This is true, but also misleading. The cause of this error is not the memory constraints of the host, but rather, since the library files were built on a different platform, the necessary EOF control characters are not present, causing the compiler to keep reading in data until the memory limits are reached. Successful compilation can be achieved only after the libraries have been remade in their home directories and then copied to the library directory.

Further compatibility problems are encountered when attempting to compile the X Windows version of AAARF on the Silicon Graphics workstation connected to Olympus. Not only are the aforementioned problems required to be dealt with, but the C shell of the silicon graphics machine is different enough to require the rewriting of the makefiles to adhere to its syntax rules.

5.4.2 Executing AAARF in X Windows After successfully compiling the X Windows version of AAARF, the next step was to attempt execution. The package cannot be successfully initiated from outside of the X Windows environment. The AAARF user interface implementation in X Windows is very similar to that in Sunview. Small changes have been made to take advantage of some of the beneficial features of X Windows, but these are of an aesthetic nature rather than functional.

The menu system has taken on a slightly different look by using the ready made pull down menu feature of X Windows. The help screen, environment panel and environment menu are almost exactly the same as their Sunview counterparts.

Coding is complete for the set of AAARF performance animations, including RVAs. These displays cannot be tested at this point, since coding for the "AAARF Central Control Panel" is not yet complete.

5.4.3 Preliminary Evaluation The compilation problems associated with platform configuration parameters, and library files are major drawbacks. The Sunview version of AAARF compiles equally well on the Sun3, Sun4 and Sparc stations with only minor changes to path names. Additionally, once compiled, and resident on a file server such as Olympus, any of the above workstations running Sunview can access the same code and operate correctly. This is not possible with X Windows applications, and causes a serious problem where users of different workstation types are accessing the same file server, conceivably requiring each user to rebuild the package before using it every time.

5.5 Summary

Many of the limitations of AAARF may be overcome by implementing it in an operating environment other than Sunview. One such operating system is X Windows, which offers improved transportability, and a library of tools that simplify the coding effort. The adaptation of AAARF from Sunview was begun by previous researchers, and has been continued to bring it to a semi-operable state. One of the objectives stated in Chapter 1, is to implement AAARF in a "readily transportable" environment, and unfortunately X Windows goes only part way in satisfying this objective. Certainly X Windows offers the ability to transport AAARF between machines of varying architecture and operating systems, but this is certainly not achieved "readily".

VI. Conclusions and Recommendations

6.1 Introduction

The closing chapter of this thesis reviews from a high level , the content of this thesis and how it supports the objectives put forward in the first chapter. Following the review of the thesis, the most significant aspects of this work are presented. Finally, the last section discusses possible directions for future work in the development of AAARF.

6.2 Conclusions

The overall objective of this work is to improve the performance of the AFIT Animation Research Facility (AAARF), in several key areas. The areas concentrated on are the performance animation views, resource utilization, and user interface.

Chapter 2 presents a review of the basic principles of parallel performance visualization. The methodologies and tools employed are described, and the different levels of data discussed. These terms are then used in the discussion of the state of the art of parallel performance visualization. The capabilities of the existing AAARF package are discussed at length, and its positive and negative characteristics identified. A number of other visualization packages are also reviewed and compared to AAARF. These packages have been developed for a variety of purposes. Some are dedicated to the visualization of parallel performance, while others are more application directed, such as the animation of matrix processing, and provide parallel performance animation as a secondary product. These packages though differing in specific detail, all present the parallel performance data in a similar format, using diagrams such as Gantt charts, speed-up curves and Kiviat charts. The graphical displays can be categorized into two broad groups, Windowed Time Intervals (WTIs) and Instantaneous Value Animations (IVAs). WTIs show performance as a function

of time in a scrolling interval of time as execution continues. IVAs show the state or value of a parameter at the current point of execution. many of the packages use the Portable Instrumented Communication Library (PICL), which prevents them from producing displays simultaneous to execution. AAARF allows simultaneous execution and display, but the display is not in true real time due to the complexity of producing the animations and the restrictions of the host processor. The situation worsens as the amount of data to be displayed increases as is the case in systems using a large number of processing nodes. As the amount of data being displayed increases, another problem is encountered: the ability of the user to comprehend and interpret the data. A new form of animation is required to communicate large amounts of data, in real time, in a form that is readily comprehended, but not so simple as to be of little worth.

Chapter 3 discusses the limitations of WTIs and IVAs and suggests an alternate form of graphical display, Ratio Value Animations (RVAs). RVAs represent parameters as a ratio of another parameter or time. Examples include utilization time vs execution time and message lengths vs communication time. In displaying these ratios a historical aspect is presented (overcoming the major deficiency of IVAs), as a single value thus reducing the complexity of animation production, compared with WTIs.

Since the parallel performance packages all use much the same type of display forms, a technique for the implementation of RVAs that allowed for the comprehension of large amounts of data in real time had to be found in another field.

A broad overview of the field of graphical data representation is presented. The field is found to be one of diverse opinions with few formalized guidelines. Most recommendations are application/domain specific, and based on largely subjective judgement. Several of these proposals may be applied to the design of a suitable format for RVAs. The "Proximity Compatibility Principle" asserts that the degree to which multiple channels of information are similarly grouped should correspond

to the similarity of processing required. Other studies have shown that like colored stimuli will lead to a perception of proximity. These two proposals suggest that the use of like colors to represent similar parameter states or values will allow the user to identify general trends among large amounts of data.

Iconic representations are one form of data display which apply these principles. The different types of iconic representation are discussed and analyzed with respect to their applicability to the representation of RVAs. The discussion, and the results of a simple test lead to the development of an RVA format which combines the properties of two of the forms of iconic representation, iconic faces, and graduated circles. The format represents each node as a circle which changes color according to the value of the parameter being measured. The test shows this from to be simple to produce and readily and accurately comprehended.

The new format has been implemented and is available as a menu selection as are all performance animations.

AAARF produces a variety of performance animations, which may be used be used singularly or in configurations of up to four different animations. The combinatoric possibilities for configurations is extremely large. Therefore the task of choosing the optimal configuration can be very difficult for the novice user. A possible solution to this problem is the provision of an expert system to aid the user in selecting the animations which best suit the application.

Chapter 4 discusses the basic principles of expert systems and presents the development of an expert system designed to aid users in constructing AAARF presentation configurations to suit their particular applications and purposes. The expert system is constructed using the CLIPS expert system language. CLIPS is a forward chaining language that is readily available at a low cost. CLIPS applications are easily transported from one operating environment to another. The expert system is available to the user as a selection from the AAARF main menu.

Chapter 5 examines the limitations placed on AAARF by the Sunview operating environment. These limitations include system memory capabilities, and a restriction in the number of host platforms. Williams(62) suggested that these limitations may be overcome by implementing AAARF in the X Windows operating environment. The basic features of X Windows are reviewed and the possible benefits discussed. The task of transporting AAARF to the X Windows environment and the lessons learned are described in Chapter 5. The X Windows operating environment certainly increases the range of platforms for AAARF, but this is not achieved without considerable effort and some trade-offs.

6.3 Recommendations

There are three recommendations for further work in the development of the AAARF package:

6.3.1 Expert Configuration Control To increase the user friendliness of the AAARF package, the user should be given the ability to directly implement the recommendations of the AAARF Expert Tool. This would greatly reduce the amount of work required to setup the required environment.

6.3.2 Completion of the X Window version of AAARF This package is close to completion; however, considerable work is still required to adequately document it. The documentation required includes new manuals for both AAARF users and AAARF programmers. Additionally, the code itself requires further documentation.

6.3.3 Animation Views The addition of RVAs to the AAARF performance animation arsenal has considerably enhanced the real time capabilities of AAARF. These animations could be made more user friendly by the provision of menu based control over the display variables. Such a menu would allow the user to select the numerator and denominator of the ratio to be animated and set the appropriate

color intervals.

Appendix A. *Introduction to Expert Systems*

This appendix presents a brief overview of the terms and expressions used to describe expert systems and the ways in which they may be structured. The aim of this appendix is to introduce the reader to basic concepts in order to better their understanding of the design and implementation of the AAARF Expert User Interface. Further information may be gained from the references(22, 9, 41).

A.1 Introduction

Artificial Intelligence (AI) is becoming ever increasingly popular in applications in banks, offices, weapons platforms and factories. People looking at their own applications frequently ask "Can we use this new technology? Are we missing out on something? How do we get started?"

AI concentrates on methods of building human intelligence and knowledge into machines. Essentially, Ai deals with various forms of knowledge processing. It includes the fields of robotics, image processing, natural language understanding and expert systems. The predominant and formalized of these is expert systems.

A.1.1 Definition An expert system is problem-solving software that embodies specialized knowledge in a narrow task domain to do work usually performed by a trained, skilled human (an expert). It is a software that can cope with incomplete and inexact data, can deal with complexity, can provide explanations of its conclusions, and can perhaps learn from experience.

No expert system lives up to every part of that definition, but we can say that an expert system has the ability, given some state of process, to figure out what to do next, based on the knowledge that it can apply to that situation.

A.1.2 Features of an Expert System Expert systems can be considered to have seven defining dimensions. These dimensions are:

1. *Expertise* The most important goal in expert system work is to attain the high level of performance that a human expert achieves in some task. Although successful performance of a task is part and parcel of expert behavior, it is not sufficient in itself to determine that a system is expert. The way in which a successful behavior is achieved is what counts.

2. *Symbol Manipulation* The first important factor that distinguishes work on expert systems from simply high-quality special-purpose programming is its relation to AI in general and to symbolic, representational reasoning in particular.

3. *General Problem-Solving Ability* The principal quality that general knowledge and inferential ability produces, over and above what expert rules do, is robustness. As new, unanticipated patterns crop up, inflexible, compiled solutions fail. general problem-solving abilities allow a more graceful degradation at the outer edges of domain knowledge as well as permit interpolation between high-level rules that are not complete within the domain.

4. *Complexity and Difficulty* Certain domains do not qualify a potential arenas for expertise because they are not complex enough. The reasoning required to solve problems in such a domain may not involve enough steps or enough alternatives at any branch point.

5. *Reformulation* One of the critical tasks that real experts do and that expert systems are just beginning to approach is to take a problem stated in some arbitrary initial form and convert it into the form appropriate for processing by expert rules.

6. *Abilities Requiring Reasoning About Self* Expert systems need the ability to rationalize their own decisions. Rationalization is the ability for the system to

CONVENTIONAL	EXPERT
operate primarily on numbers	operate primarily on symbols that represent real world objects or concepts
processes are coded as detailed step by step procedures	processes are coded as discrete rules or events
algorithmic	heuristic
processes must be fully understood before coding	supports iterative development - prototyping
difficult to modify programs	easy to modify programs
conclusions reached are correct or incorrect	conclusions reached are satisfactory but not always perfect - fuzzy logic

Table A.1. Characteristics of Conventional and Expert Systems

look at its own chain of reasoning and understand it as if it were another task domain requiring expertise.

7. *Task* The final defining dimension of expert systems is the generic task that the system is built to do. Different tasks may substantially change the picture of a system's architecture.

A.2 Conventional vs Expert Systems

A.2.1 Typical Characteristics Conventional and expert systems can be differentiated by a quick comparison of their general characteristics, organization, and goals. Table A.2.1, table A.2.1, and table A.2.1 show such a comparison.

The organization of an expert system is shown in figure A.1.

A.3 Expert System Knowledge Representation

Expert knowledge in the task domain must lend itself to a formalized representation to be implemented in a knowledge base. Knowledge representation involves

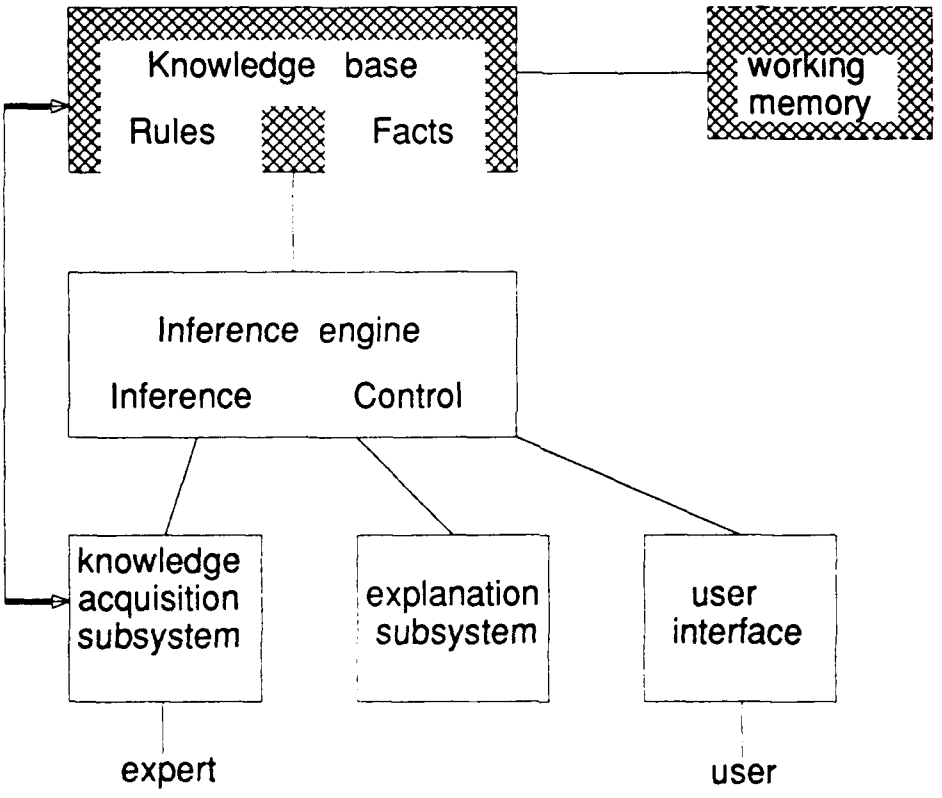


Figure A.1. Organization of an Expert System

CONVENTIONAL	EXPERT
creating, updating, deleting	data retrieval/analysis
validating	what-if simulation
reporting	suggestion
	automatic decision making

Table A.2. Goals of Conventional and Expert Systems

CONVENTIONAL	EXPERT
two levels: data and control	three levels: data, control and task knowledge
task knowledge and control information are both explicitly contained in program code	separation of control from task knowledge can reduce difficulty in constructing complex programs

Table A.3. Organization of Conventional and Expert Systems

decisions on how to structure explicit knowledge, how to manipulate it to infer additional data, and how to acquire the knowledge required by the system.

For many applications, developers may use a commercially available expert system shell - a program in which some of these decisions have already been made. If a standard shell is used for development, the format and structure for working memory, and the format for rules to use and manipulate knowledge are part of that shell.

The last decade has seen a consolidation of the methodologies employed in the knowledge representation field, and a formalization of the theory associated with differing methods. A formal semantics is now an obligatory accompaniment of the description of a novel KR system. Unfortunately this increase in KR theory has seemingly come at the expense of experimental development work in the field.

Today there are four representational paradigms which seem to predominate:

- semantic networks

- object-attribute-value (OAV) triplets
- predicate logic
- frames

A.3.1 Static Knowledge versus Instances There is an important distinction between a static unchanging object and a dynamic instance of that object that changes with time. Figure A.1 shows the knowledge base is connected to a box labeled "working memory". that box is drawn with dashed lines to indicate that it isn't always there. When an expert system is "between consultations" its static knowledge of facts and rules is stored in the knowledge base. When a consultation begins, the system seeks to determine the values of various attributes relative to the particular situation. As values are determined, the system stores this dynamic information in its working memory. the process of determining specific values for the attributes is often called instantiation.

A.3.2 Semantic Networks One of the most general representation schemes is the semantic network as shown in figure A.2. A semantic network is comprised of nodes and links. The nodes represent objects and descriptors. The links relate objects and descriptors. A link may represent any relationship. Semantic networks offer great flexibility since new nodes and links may be added almost at will. The other side to flexibility is that it increases the chances of inconsistency. Semantic networks also support the property of inheritance. This too has a downside - the difficulty of exceptions. This can be combatted by the use of exception arcs, but the implementation of these can be very difficult.

A.3.3 OAV Triplets Another common way to represent factual information is as object- attribute-value (O-A-V) triplets. In this scheme objects may be physical entities, or conceptual entities. Attributes are general characteristics or properties associated with objects. The final member of the triplet is the value of the attribute.

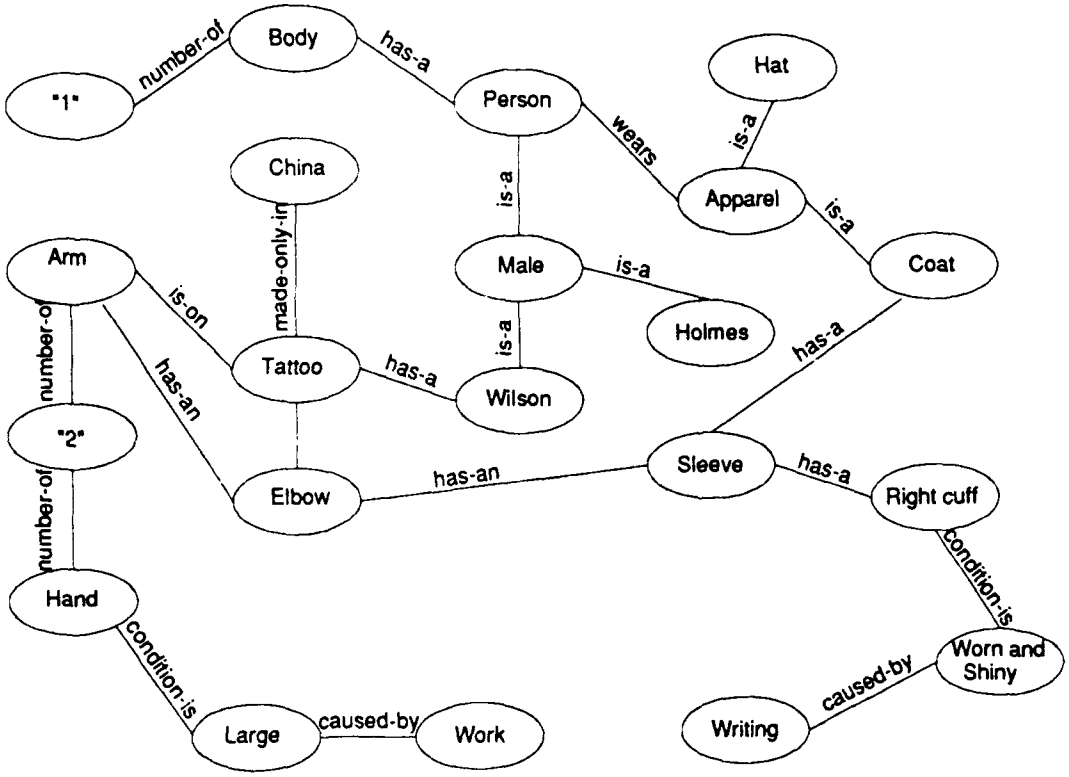


Figure A.2. Semantic Network

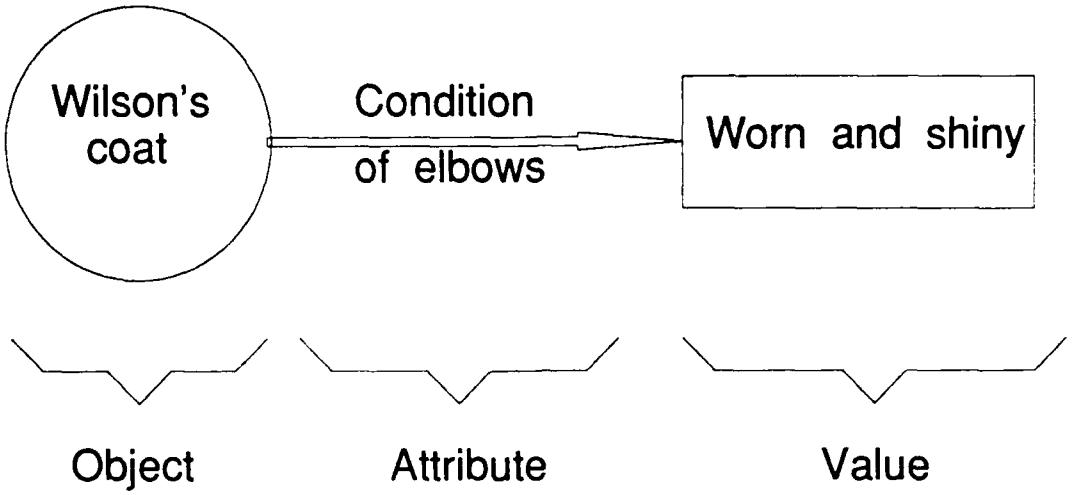


Figure A.3. Object-Attribute-Value Triplet

The value specifies the specific nature of an attribute in a particular situation. Figure A.3 shows facts represented as an O-A-V.

A.3.4 Predicate Logic Predicate calculus is an extension to propositional logic used by most programming languages. The elementary unit in predicate logic is an object. Statements about objects are called predicates. For example, "is-girl(Mary)" is an assertion that says that Mary is a girl. This assertion is either true or false. Predicates can address more than one object. The statement that "daughter- of(Mary, Rose)" is an example of a two-place predicate. This statement asserts that Mary is the daughter of Rose. Ordinary connectives can be used to link together predicates into larger expressions. Thus the statement "daughter-of(Mary,

Rose) AND daughter-of(Mary, John)" is an expression that is either true or false, depending on whether Rose and John are, in fact the parents of daughter named Mary(22).

A.3.5 Frames Frame representation as shown in figure A.4 was first proposed by Minsky(41) in 1975. Minsky's concern for more realistic commonsense reasoning, using prototypes and defaults, complex object descriptions, "differential diagnoses" of situations, etc., led to the development of numerous frame representation systems. In general a frame may be considered as a description of an object that contains slots for all of the information associated with the object. Slots may contain attribute values, default values, pointers to other frames, sets of rules, or procedures by which values may be obtained. Procedures for obtaining values may be constructed in two ways: *procedural* and *declarative* as shown in figure A.5. Frames represent a richer representation of knowledge, but are more complex and more difficult to develop than other simpler representations.

A.4 Backward and Forward Chaining

Once knowledge is represented in some form, a reasoning procedure to draw conclusions from the knowledge base is required. The more advanced search techniques developed for AND/OR graphs are in a way even more advanced than those usually used in expert systems. On the other hand, there is often and emphasis in expert systems on the style of search, from its semantic point of view, in relation to human reasoning; that is, it is desirable to sequence the reasoning in ways that humans find "natural" in the domain of application. This is important when interaction with the user occurs during the reasoning process and we want to make this process transparent to the user. This sequencing falls into one of two categories, *backward chaining* or *forward chaining*, as discussed below and represented in figure A.6.

Coat	
Slots	**Entries**
Owner	Wilson
Condition	Rumpled
Condition of cuffs	Worn, shiny
Condition of elbow	Worn, shiny
Number of arms	Default: 2
Fabric	*Default: wool*
Pockets?	Default: yes
Size	If needed, find owner's height and weight, see table X.
Style	If needed, find out collar:, pockets:, and see table Y.

Figure A.4. Frame Representation

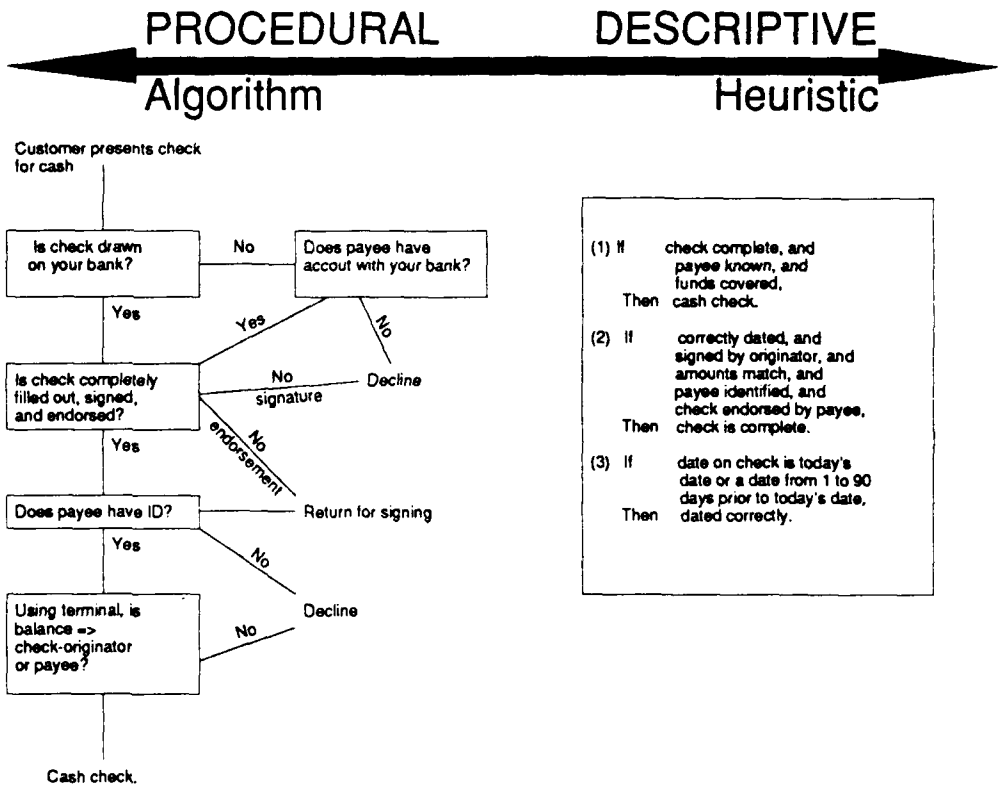


Figure A.5. Procedural and Declarative Representations

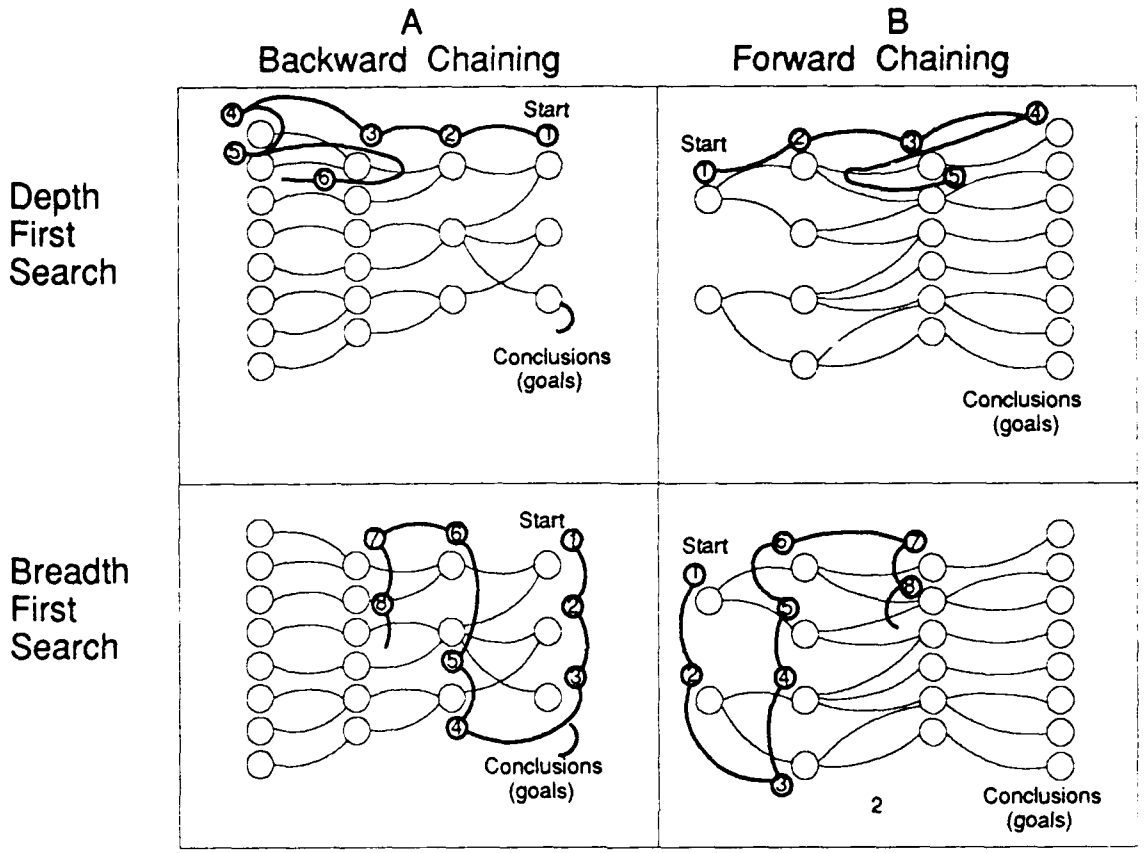


Figure A.6. Backward and Forward Chaining

A.4.1 Backward Chaining Backward chaining begins with a hypothesis or goal and works "backward" through subgoals in an effort to show that the hypothesis is true or false. If the possible outcomes are known, and if they are reasonably small in number, (i.e., small search/solution space), then backward chaining is very efficient.

A.4.2 Forward Chaining Forward chaining does not start with a hypothesis, but with some confirmed findings. The premises of the rules are examined to see whether or not they hold, given the information at hand, and transitions are made in the search graph. The outcome is added to the list of known facts and the rules are examined again, this process is repeated until the known facts reach a static state or goal state.

A.5 Summary

This appendix has presented the salient points of expert system structure to assist the user in appreciating the AAARF expert user interface.

Appendix B. *Icon Recognition Test*

B.1 Aim

The aim of this experiment is to compare the comprehensibility of three types of iconic representation: coloured circles, graduated circles, and coloured graduated circles. A secondary aim of the experiment is to analyse the best contrast arrangements where colours are used. A format for the implementation of Ratio Value Animations will be selected based on the results.

B.2 Method

A sample population of AFIT students and staff will be shown a series of displays on a computer terminal screen. Each display will be shown for only 5 seconds. The testees are to view each display and indicate which colour or shape is the most dominant and what percentage of the icons in the display are of that shape or colour. There are 8 displays of each type, 24 in total. The displays for each type represent the same range of shape/colour distributions. The coloured graduated circles and coloured circles displays are shown in figures B.1 thru B.16.

The responses of the participants will be compared to the actual percentages and the accuracy or ease of recognition of each icon form calculated. The comparison will be made using the following equation:

X - PY = Z

X = the sum of the percentage estimates by participants

P = the number of participants

Y = actual percentage of dominant icon

Z = error margin

		TESTEE 1			*TESTEE 2*			*TESTEE 3*			*TESTEE 4*		
display	actaul	cc	gc	cgc	cc	gc	cgc	cc	gc	cgc	cc	gc	cgc
1	38r/t	30r	30t	30r	40r	30t	30r	35r	30t	40r	40r	30t	30r
2	43r/t	30r	30t	40r	40r	40t	35r	45r	30t	40r	45r	35t	35r
3	33rg/tb	30r	40b	40g	35r	40b	40g	30g	40b	35g	35g	40b	40g
4	43g/b	40g	50b	50g	45g	50b	50g	40g	60b	60g	45g	40b	45g
5	53g/b	40g	60b	50g	50g	70b	60g	50g	50b	60g	55g	50b	50g
6	53g/b	50g	60b	50g	50g	60b	50g	60g	60b	50g	50g	55b	60g
7	53g/b	50bl	60b	60g	50g	60b	60g	60g	70b	60g	50g	60b	50g
8	60br/s	60br	50s	50br	50br	50s	50br	60br	70s	60br	70br	60s	70br
		TESTEE 5			*TESTEE 6*			*TESTEE 7*			*TESTEE 8*		
display	actaul	cc	gc	cgc	cc	gc	cgc	cc	gc	cgc	cc	gc	cgc
1	38r/t	35r	35t	40r	40r	30t	30r	40r	35t	30r	40r	30t	30r
2	43r/t	40r	30t	40r	35r	35t	35r	45r	40t	45r	45r	35t	40r
3	33rg/tb	30r	40b	40g	35r	40t	40r	30g	40b	35r	35r	40t	40g
4	43g/b	40g	50b	50g	45g	50b	50g	45g	50b	45g	45g	55b	50g
5	53g/b	40g	60b	60g	50g	70b	55g	50g	50b	60g	55g	60b	50g
6	53g/b	50g	60b	50g	50g	55b	50g	60g	60b	50g	45g	55b	60g
7	53g/b	50bl	60b	60g	50g	55b	60g	60g	70b	60g	50g	60b	50g
8	60br/s	60br	50s	60br	55br	70s	55br	60br	70s	60br	65br	60s	70br

Table B.1. Icon recognition test results

B.3 Results

The results of the test are shown in table B.3.

B.4 Analysis and Conclusions

Table B.4 presents the data analysis results.

The results clearly indicate that superior accuracy is produced with the coloured circles. The worst performance is achieved by the non-coloured graduated circles. A possible cause of this is that as circles become larger they use a correspondingly larger portion of the display, therefore though only 20% of the circles may be of the largest size, they cover 30% of the display. The use of coloured graduated circles has better results, but the same principle applies.

TOTALS AND DIFFERENCES							
display	actual	*Coloured Circle*		*Graduated Circle*		*Col. and Grad. Circle*	
	PY	X	Z	X	Z	X	Z
1	304	300	4	250	54	260	44
hline 2	344	325	19	275	69	300	44
hline 3	264	260	4	320	56	310	46
4	344	345	1	405	61	400	56
5	424	360	64	470	46	445	21
6	424	415	9	465	41	420	4
7	424	420	4	495	51	460	36
8	480	480	0	480	0	475	5
total Z			105		378		256

Table B.2. Icon recognition test analysis

These results are consistant with those obtained by other researchers indicating colour to be a better recognition tool than shape. Additionally it was shown that the addition of changing colours to changing shapes produces an improvement but that the addition of changing shapes to changing colours resulted in a deterioration in performance.

The contrast of colours proved to be only slightly different from their order in the spectrum. Testees found difficulty in discerning between the pair of red and brown, and the pair of green and yellow. There were no other significant trends identified in this area.

Therefore, the coloured circles are recommended as the best method for the representation of RVAs, using colours in the following order: red, brown, yellow, green, blue.

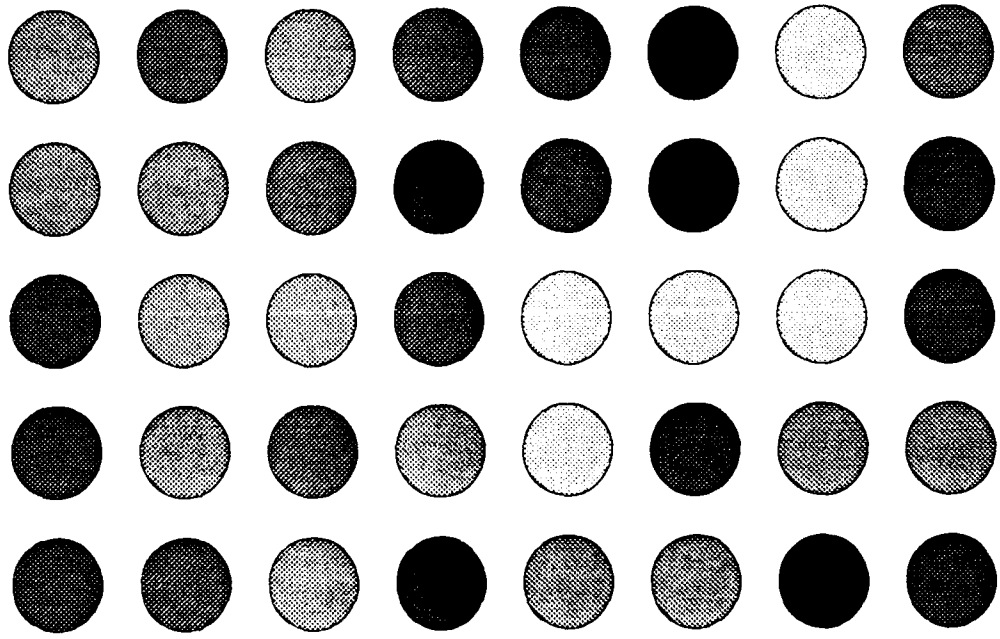


Figure B.1. Icon Recognition Test Display

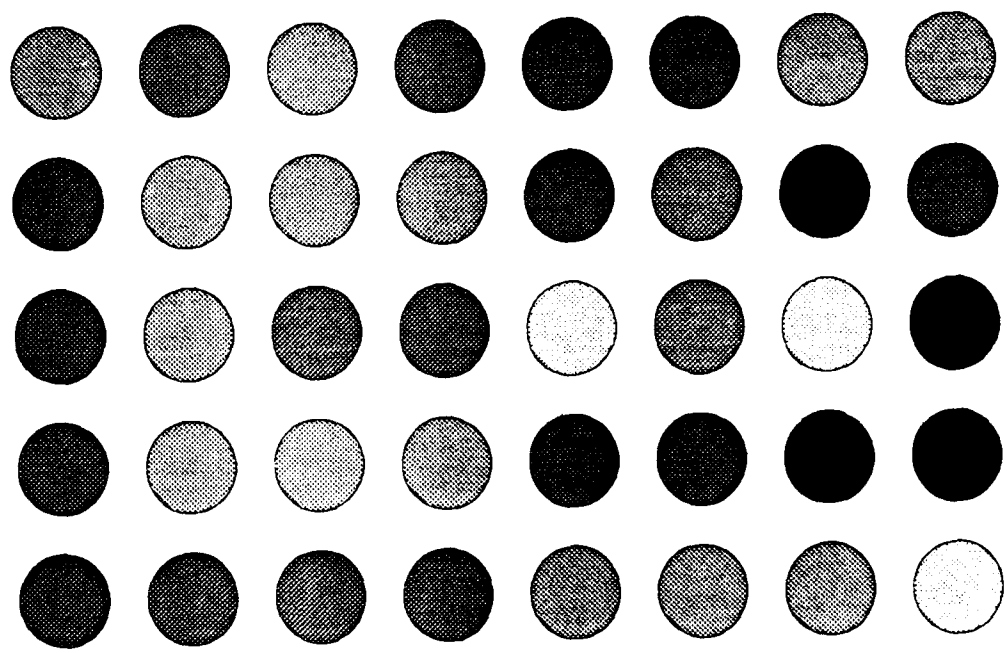


Figure B.2. Icon Recognition Test Display

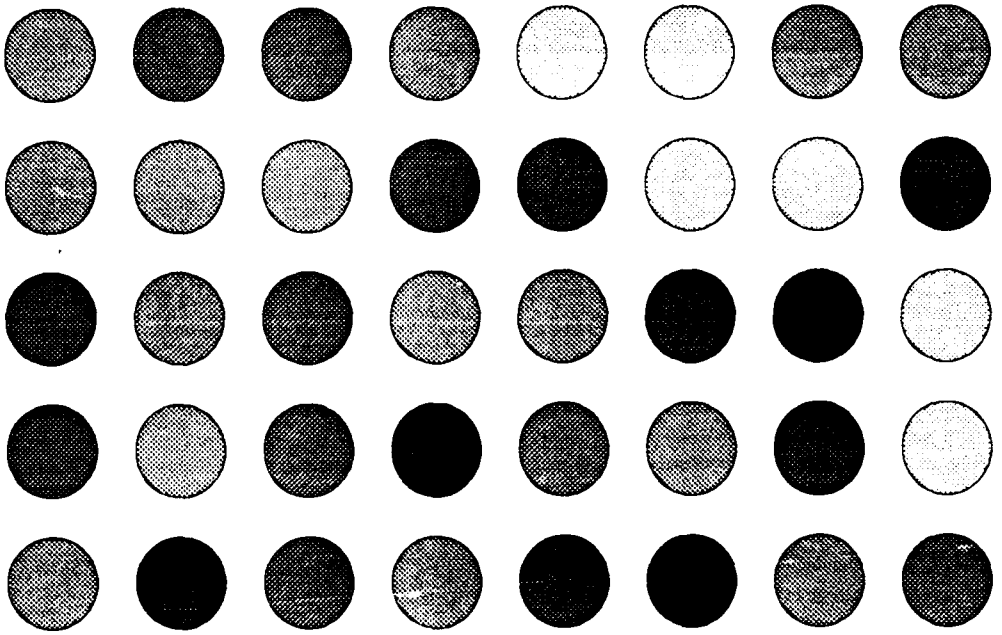


Figure B.3. Icon Recognition Test Display

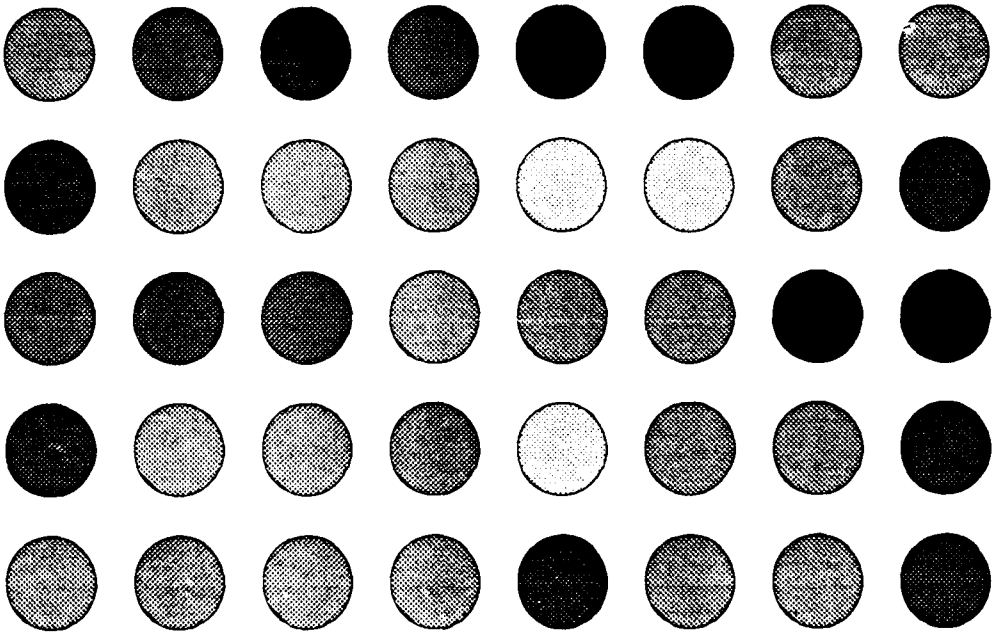


Figure B.4. Icon Recognition Test Display

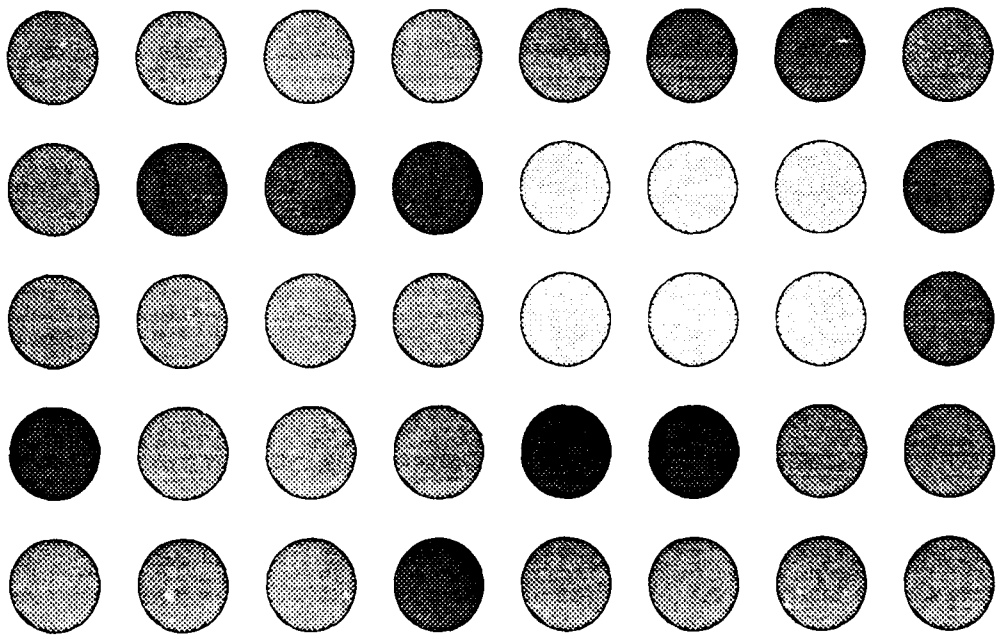


Figure B.5. Icon Recognition Test Display

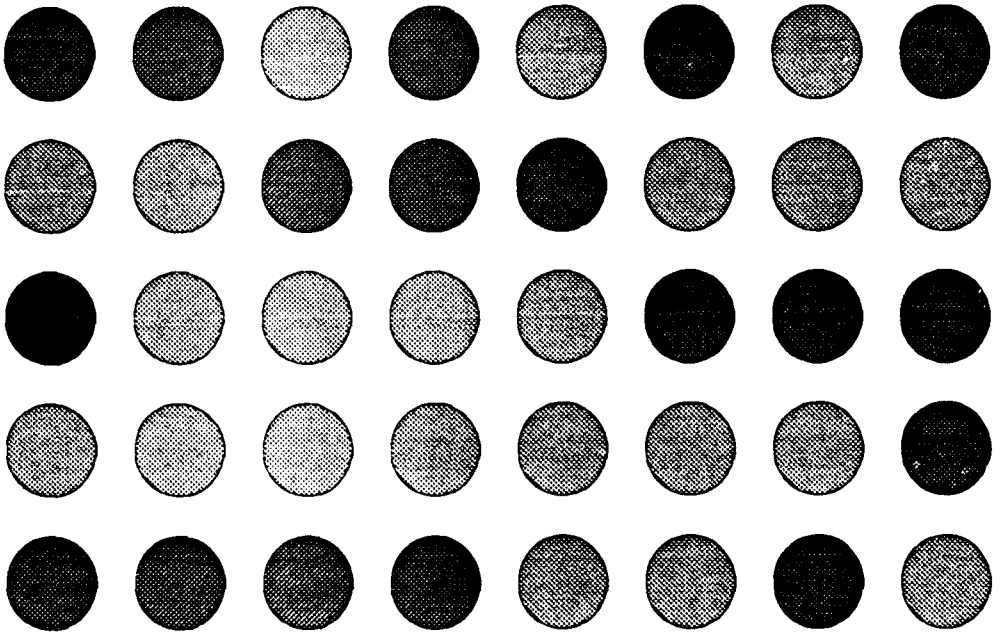


Figure B.6. Icon Recognition Test Display

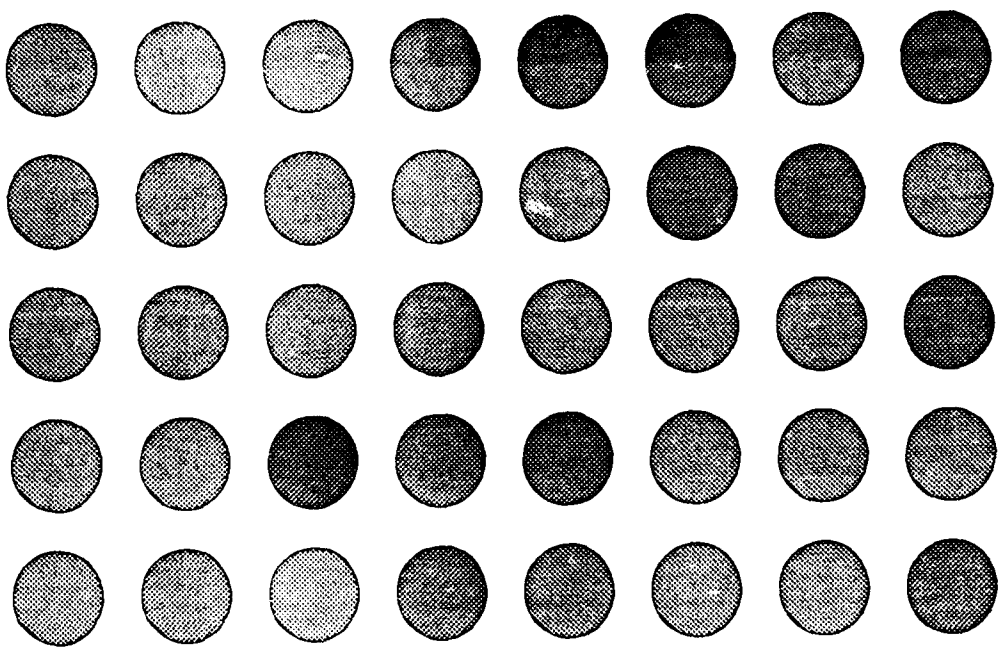


Figure B.7. Icon Recognition Test Display

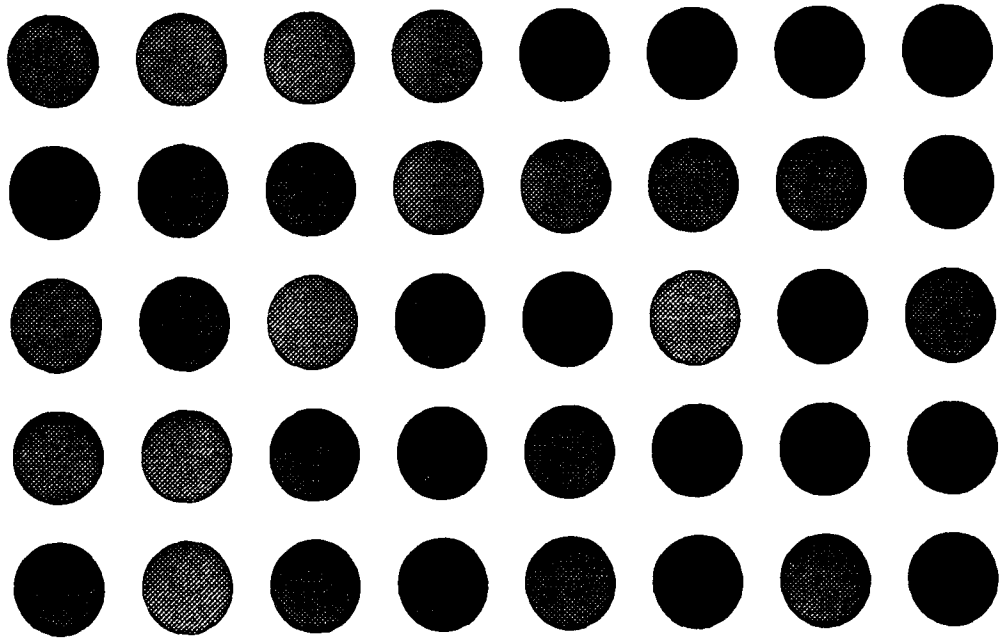


Figure B.8. Icon Recognition Test Display

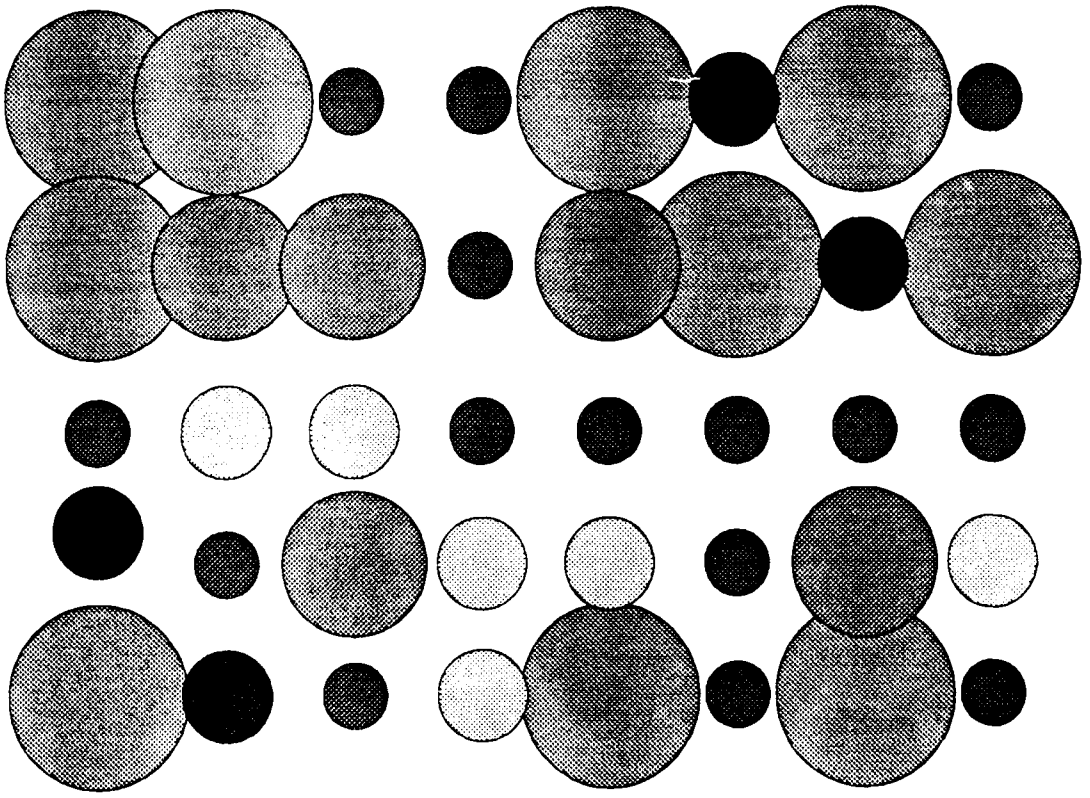


Figure B.9. Icon Recognition Test Display

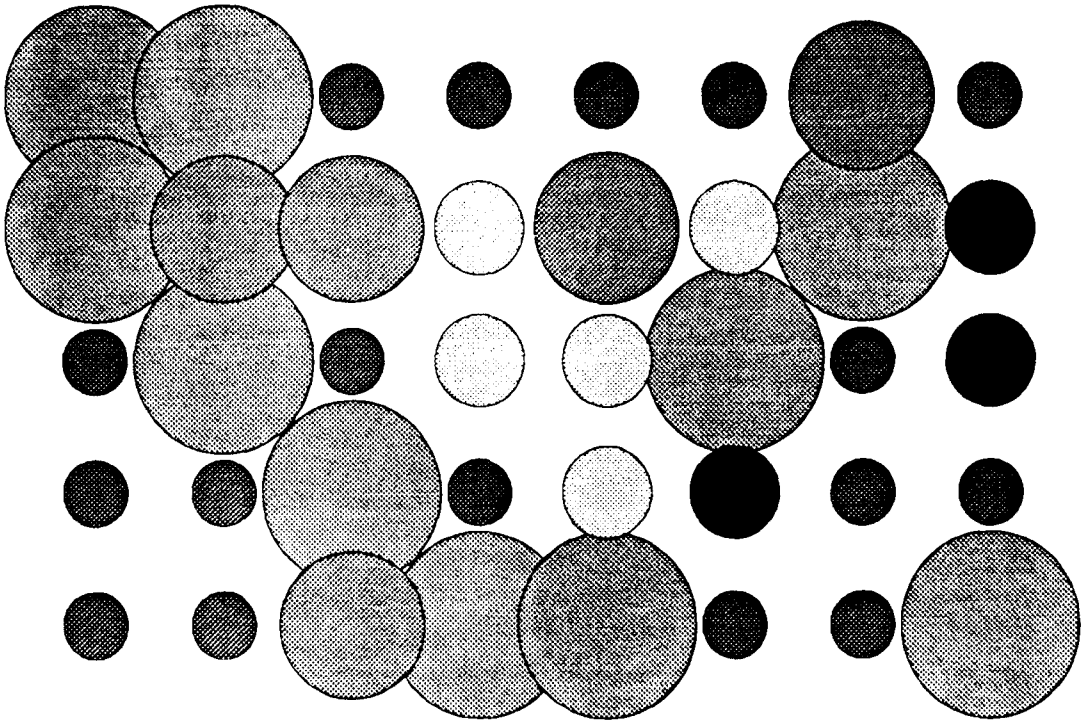


Figure B.10. Icon Recognition Test Display

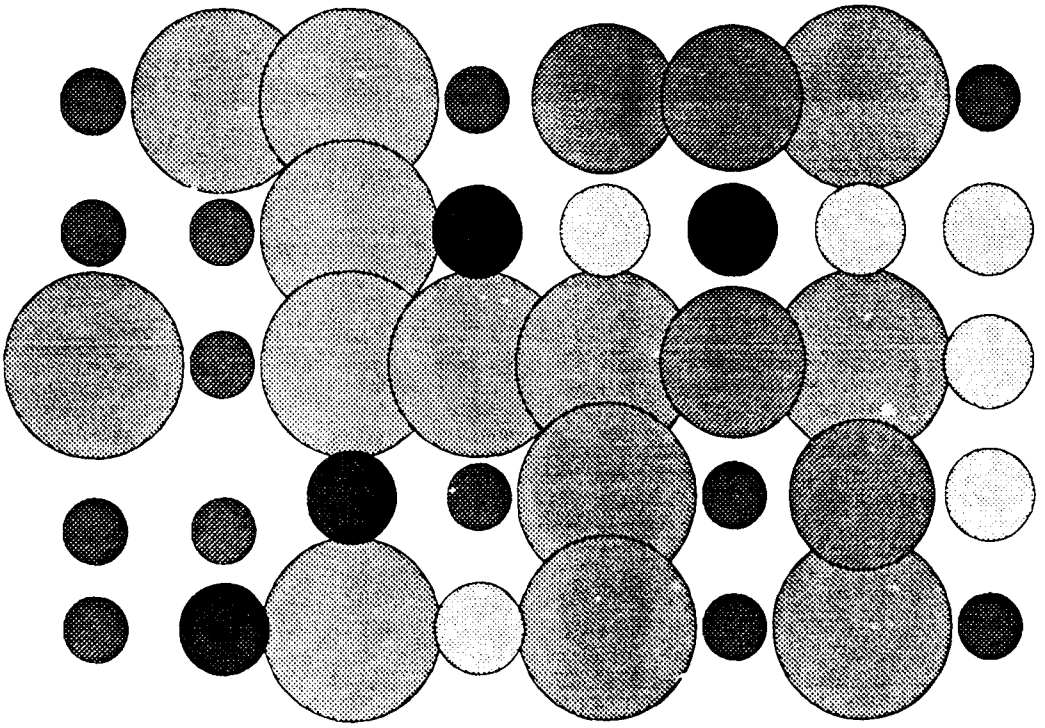


Figure B.11. Icon Recognition Test Display

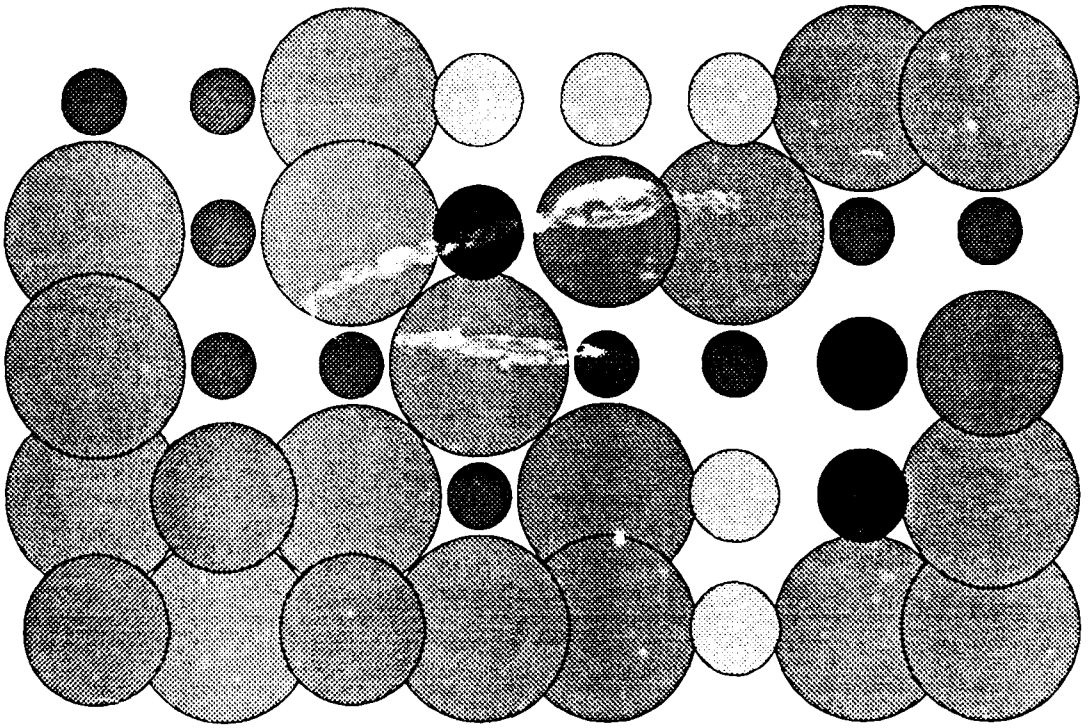


Figure B.12. Icon Recognition Test Display

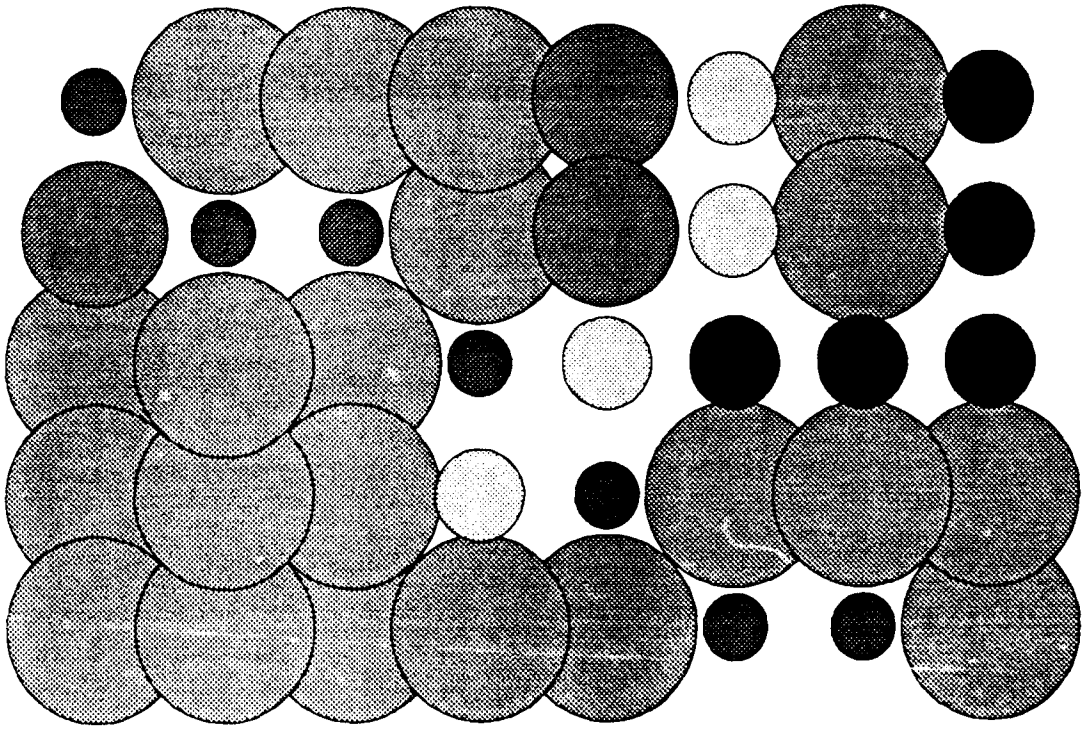


Figure B.13. Icon Recognition Test Display

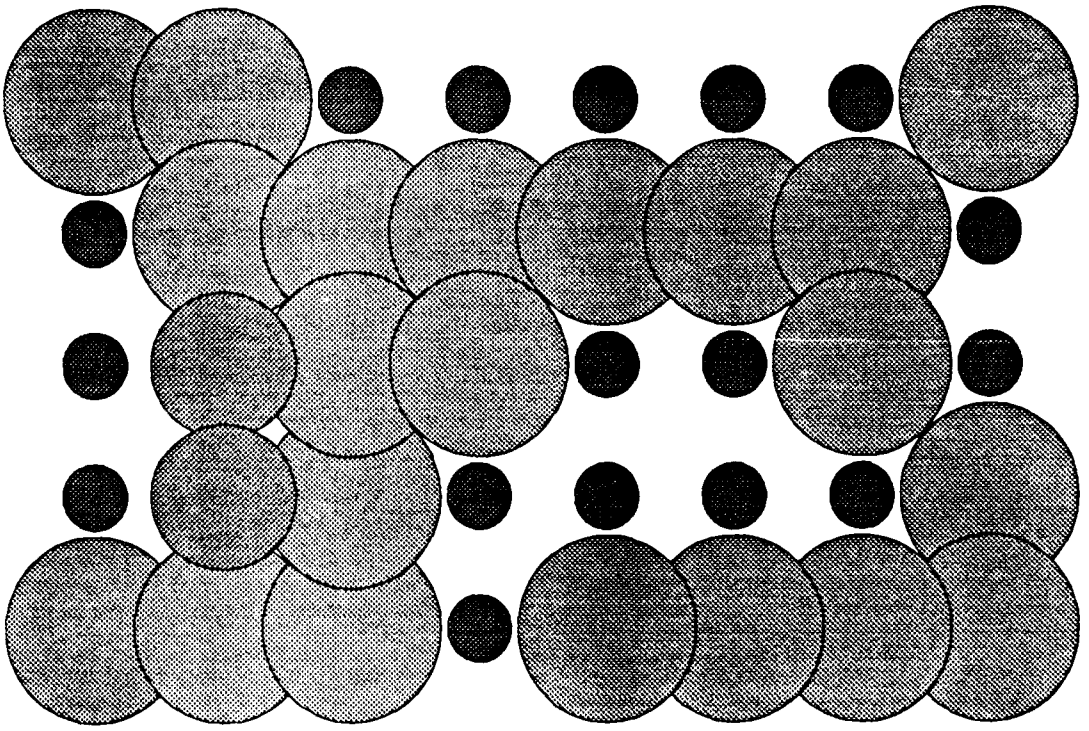


Figure B.14. Icon Recognition Test Display

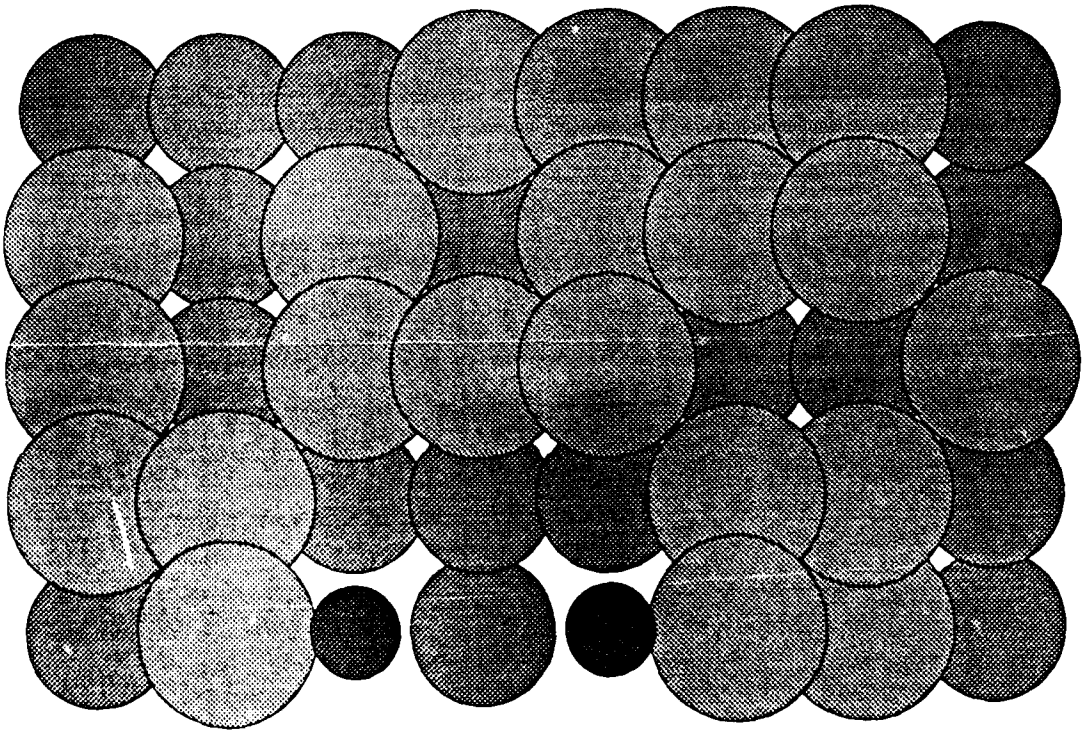


Figure B.15. Icon Recognition Test Display

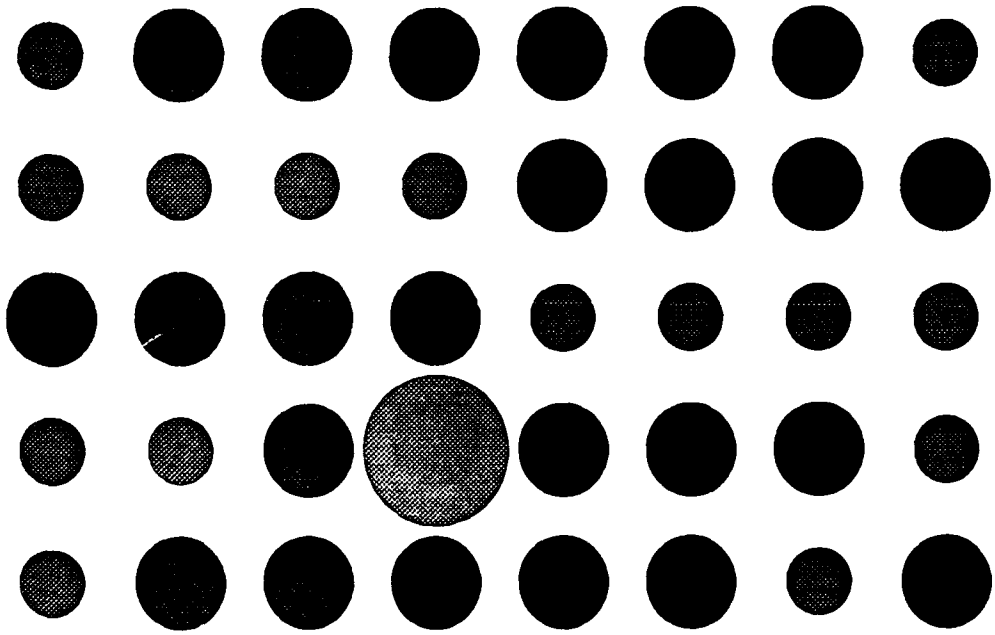


Figure B.16. Icon Recognition Test Display

Bibliography

1. Accetta, M et. al. "Mach: A New Kernel Foundation for Unix Development." In *Proceedings of USENIX 1986 Summer Conference*, pages 93–112, 1986.
2. Andre, Anthony D. and Christopher D. Wickens. *Proximity Compatability and Information Display: The Effects of Space and Color on the Analysis of Aircraft Stall Conditions*. Technical Report, Aviation Research Laboratory and Department of Psychology, University of Illinois, October 1989.
3. Bemis, S. V. et al. *The Efficacy of Color-Coded Symbols to Enhance Air-Traffic Control Displays*. Technical Report, Naval Ocean Systems Center, San Diego, CA, 1988.
4. Bemmerl, Thomas et al. *The Design and Implementation of TOPSYS*. Technical Report, Institut fur Informatik, Munich, Germany, July 1991.
5. Bemmerl, Thomas et al. *TOPSYS User Manual*. Technical Report, Institut fur Informatik, Munich Germany, July 1991.
6. Berggrund, U., et al. "Colour coded vs monochrome situational maps." In *Proceedings of the Workshop on Colour Coded vs Monochromatic Electronic Displays*, pages 11.1–11.10, 1984.
7. Boff, K. R. and J. Lincoln. *Engineering data compendium: Human perception and performance*. Technical Report, AAMRL, Wright-Patterson AFB, OH, 1988.
8. Boles, D. B. and C. D. Wickens. *A comparison of homogeneous and heterogeneous dispaly formats in information integration and nonintefration tasks*. Technical Report, Engineering-Psychology Research Laboratory, University of Illinois, Urbana, IL, 1987. Technical Report EPL-83-6/ONR-83-6.
9. Brachman, Ronald J. "The Future of Knowledge Representation." 1990.
10. Brown, Marc H. *Algorithm Animation*. Cambridge, Massachusetts: The MIT Press, 1987.
11. Bundesen, C. and L. F. Pedersen. "Color segregation and visual search," *Perception and Psychophysics*, *33*(5):487–493 (1983).
12. Carswell, C. M. and C. D. Wickens. "Information integration and the object display: An interaction of task demands and display superiority," *Ergonomics*, (30).511–528 (1987).
13. Carswell, C. M. and C. D. Wickens. *Comparative graphics: History and applications of perceptual integality theory and the proximity compatability hypothesis*. Technical Report, U.S. Army Human Engineering Laboratory, Aberdeen Proving Ground, MD, 1988. Technical Memorandum 8-88.

14. Cleveland, W. S. and R. McGill. "Graphical perception: theory, experimentation, and application to the development of graphical methods," *Journal of the American Statistical Association*, (79):531–554 (1984).

15. COSMIC: NASA Software Distribution Center. *CLIPS Release Notes*, 1989.

16. Couch, Alva L and David W. Krumme. "Monitoring Parallel Executions in Real Time," *Proceedings of the Fifth Distributed Memory Computing Conference* (1990).

17. et. al., Friedall Mark. "Visualizing the Behaviour of Massively Parallel Programs." In *Proceedings of Supercomputing '91*, pages 472–480, 1991.

18. et. al., T. G. Lewis. "Task Grapher : A Tool for Scheduling Parallel Program Tasks." In *Proceedings of the Fifth Distributed Memory Computing Conference*, 1990.

19. Fife, Keith C. *The AAARF Programmer's Guide*. Air Force Institute of Technology, November 1989.

20. Fife, Keith C. *Graphical Representation of Algorithmic Processes*. MS thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1989.

21. Godfrey, G. W. *Principles of display illumination techniques for aerospace vehicle crew stations (second Edition)*. Tampa, Florida: Aerospace Lighting Institute, 1982.

22. Harmon, Paul. *Expert Systems*. New York: John Wiley and Sons, Inc., 1985.

23. Heath, Michael T. "Visual Animation of Parallel Algorithms for Matrix Computations." In *Proceedings of the Fifth Distributed Memory Computing Conference*, 1990.

24. Hotchkiss, Robert S. and Cheryl L. Wampler. "The Auditorialization of Scientific Information." In *Proceedings of Supercomputing '91*, pages 453–461, 1991.

25. Izenman, A.J. "Development in statistical graphics, 1960-1980." In *Proceedings of the first general conference on social graphics*, pages 51–79, October 1978.

26. J., Amsterdam. "Building a Flexible Knowledge Representation Scheme," *AI Expert* (November 1986).

27. J., Goldberg Aaron and John L. Hennessy. "Performance Debugging Shared Memory Multiprocessor Programs with MTOOL." In *Proceedings of Supercomputing '91*, pages 481–490, 1991.

28. Jacob, R. J. K. *Facial representation of multi-variate data*, pages 143–168. New York: Academic Press, 1978.

29. Johnson, Eric F. and Kevin Reichard. *X Window : Applications Programming*. Portland, Oregon: Management Information Source Inc., 1989.

30. Jubis, Rebecca M. T. and Donald C. Turner. *The Effectiveness of Redundant Color-Coding on Search and Identification in a Process-Control Task*. Technical Report, Defense and Civil Institute of Environmental Medicine, Downsview, Ontario, Canada, 1988.

31. Kahl, Mark Albert. *PRASE: Instrumentation Software for the Intel iPSC Hypercube*. MS thesis, 1988.

32. Kramer, A. F., et al. "Extraction and comprehension of data relations: An application of the display proximity principle." In *Proceedings of the 10th Symposium on Psychology in the Department of Defense (DOD)*, 1986.

33. Lack, Michael D. Allott and Gary B. Lamont, "Visualization of Hypercube Pedagogical Parallel Programs." Unpublished, August 1991.

34. Lehr, Ted et. al. "Visualizing Performance Debugging," *IEEE Computer*, pages 38–51 (October 1989).

35. Lehr, Ted et. al. "Visualizing System Behaviour." In *Proceedings of the 1991 International Conference on Parallel Processing*, August 1991.

36. Maloney, Allen D. and Daniel A. Reed. *Visualizing Parallel Computer System Performance*, pages 59–90. New York: Addison-Wesley Publishing Company, 1989.

37. Martin, K. W. "Display visual performance prediction in a military cockpit environment." In *Proceedings of the Workshop on Colour Coded vs Monochromatic Electronic Displays*, pages 7.1–7.15, 1984.

38. Mclaren, Russell D. and William A. Rogers. "Instrumentation and Performance Monitoring of Distributed Systems." In *Proceedings of the Fifth Distributed Memory Computing Conference*, Volume II, April 1990.

39. Mezzich, J. E. and D. R. L. Worthington. *A comparison of graphical representations of multidimensional psychiatric data*, pages 123–139. New York: Academic Press Inc., 1978.

40. Mink, Alan, et al. "Multiprocessor Performance-Measurement Instrumentation," *IEEE Computer*, pages 63–75 (September 1990).

41. Minsky, M. *A framework for representing knowledge*, pages 211–277. New York: McGraw Hill, 1975.

42. Moar, I. "Spatial organization in the free recall of pictorial stimuli," *Quarterly Journal of Experimental Psychology*, *29*(4):699–708 (1977).

43. Myers, Brad A. *The State of the Art in Visual Programming and Program Visualization*. Technical Report, Carnegie Mellon University, February 1988. CMU-CS-88-114.

44. Paul, Richard F. and David A. Poplawski. "Visualizing the Performance of Parallel Matrix Algorithms." 1990.

45. Pease, Daniel, et. al. "PAWS: A Performance Evaluation Tool for Parallel Computing Systems," *IEEE Computer*, pages 18-29 (January 1991).

46. Pountain, Dick. "The X Window System," *Byte*, *9*(14):353-360 (January 1989).

47. Quercia, Valerie and Tim O'Reilly. *X Window System User's Guide*. Sebastapol, Cal,: O'Reilly & Associates, Inc., 1990.

48. Rasmussen, J. *Information processing and human-machine interaction: An approach to cognitive engineering*. New York: North-Holland, 1986.

49. Reising. Personal Interview, June 1991.

50. Rychener, Michael D. *Expert Systems for Engineering Design*.. San Diego CA: Academic Press, Inc.,, 1988.

51. Schmid, C. F. *Statistical Graphics: design principles and practices* (second Edition). New York: Wiley, 1983.

52. Schmid, C. F. and S. E. Schmid. *Handbook of graphic presentation* (second Edition). New York: Wiley, 1979.

53. Silverstein, L. D. *Human factors for color display systems: Concepts, methods, and research*, chapter 4. San Diego, CA: Academic Press, 1987.

54. Sun Microsystems. *Network Programming*, 1988.

55. Taylor, R. M. "Colour design in aviation cartography," *Displays*, *6*(4):187-201 (1985).

56. Tufte, E. R. *The visual display of quantitative information*. Cheshire, Conn.: Oxford, 1983.

57. Turek, R. O. and J. S. Greenstein. *Polygons, Stars, and Clusters : An Investigation of Polygon Displays*. Technical Report, Navy Personnel Research and Development Center, San Diego, CA, January 1988.

58. Utter-Honig, Sue and Cherri M. Pancake. "Graphical Animation of Parallel Fortran Programs." In *Proceedings of Supercomputing '91*, pages 491-500, 1991.

59. Wickens, C. D. "Ergonomic-driven displays: A set of principles of the perceptual-cognitive interface." In *Proceedings of the 4th Mid-Central Ergonomics/Human Factors Conference*, 1987.

60. Wiener, E. L. and R. E. Curry. "Flight Deck Automation: Promises and problems," *Ergonomics*, (23):995-1011 (1980).

61. Williams, E. X Windows code for AAARF, April 1991.

62. Williams, Edward M. *Graphical Representation of Parallel Algorithmic Processes*. MS thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1990.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1991	Master's Thesis

4. TITLE AND SUBTITLE
Enhanced Animation of Parallel Algorithms

5. FUNDING NUMBERS

6. AUTHOR(S)
Michael D. A. Lack, Flight Lieutenant, RAAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/91D-11

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
WL/AARA

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Algorithm animation is a visualization method used to enhance understanding of the functioning of an algorithm or program. Visualization is used for many purposes, including education, algorithm research, performance analysis, and program debugging. This research, which follows on from previous work, examines algorithm animation requirements for the various visualization purposes and extends the capabilities of an existing facility, the AFIT Algorithm Animation Research Facility (AAARF). The structure of AAARF is summarised and its visualization capabilities presented, analysed and compared with other similar packages.

This research focuses on the parallel data requirements of the AAARF users, and the meaningful display of large amounts of data. This paper discusses a series of refined animations suitable for a variety of purposes which are capable of conveying detailed information in a concise and timely manner and the development and implementation of a new performance animation type is presented. These animations are built upon pedagogical efforts in a classroom environment. To assist novice users in efficiently using AAARF, an "expert system" interface has been implemented to advise on optimizing environment configuration. Considerable effort has been devoted to porting AAARF from Sunview to the X Window environment, and the lessons learned and progress made are discussed.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Parallel Processing, Scientific Visualization	131
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to **stay within the lines to meet optical scanning requirements.**

Block 1. Agency Use Only (Leave Blank)

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency. Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ..., To be published in .... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availablity Statement. Denote public availability or limitation. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR)

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- DOD - Leave blank
DOE	- DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports
NASA	- NASA - Leave blank
NTIS	- NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.