System Architecture for High-Performance Vector Graphics

by

Ingrid Birgitta Carlbom

Fil. Kand., University of Stockholm, 1968

M.S., Cornell University, 1971

Thesis

Submitted in partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in the Department of Computer Science at Brown University

June, 1980

This dissertation by Ingrid Birgitta Carlbom

is accepted in its present form by the Department of

Computer Science as satisfying the

dissertation requirement for the degree of Doctor of Philosophy

Date: November 28, 1979 [illegible]

Recommended to the Graduate Council

Date: November 15, 1979 [illegible]

Date: November 15, 1979 James D. Foley

Date: November 28, 1979 [illegible]

Approved by the Graduate Council

Date: June 1, 1980 [illegible]

ML

VITA

Personal Data

Full name: Ingrid Birgitta Carlbom
Date of Birth: May 25, 1945
Place of Birth: Hudiksval, Sweden
Citizenship: Swedish

Education

Sept. 1971	M.S. in Computer Science, Cornell University, Ithaca, New York
June 1968	Fil. Kand. in Applied Mathematics, University of Stockholm, Stockholm, Sweden

Professional Experiences and Activities

Co-Designer of the SIGGRAPH proposed standard for a Core Graphics System; Co-Author and Production Editor of the Status Report (Sept. 1976 - June 1977).

Vice-President and Co-Founder of Capital Advisors, Inc., Providence, Rhode Island, an investment advisory company. Developed and implemented a technical stock market model (Nov. 1975 - present). The exclusive rights to the product were purchased by Paine Webber Jackson and Curtis, Inc., a major New York Stock Exchange firm.

Research Assistant, Division of Applied Mathematics, Brown University, Providence, Rhode Island. Designed and implemented a microcode simulator for a Digital Scientific Corp. Meta 4; co-designed and developed an experimental compiler for the Language for Systems Development (Sept. 1971 - Dec. 1972; Sept. 1974 - May 1975).

Teaching Assistant, Division of Applied Mathematics, Brown University, Providence, Rhode Island. Designed and assisted in graduate courses in computer graphics (Jan. 1973 - May 1973; Jan. 1974 - May 1974).

Computer Programmer, Brown University, Providence, Rhode Island. Designed and implemented a Student Evaluation System, which was proposed as an alternative to the grading system (Sept. 1972 - Aug. 1973).

Teaching Assistant, Department of Computer Science, Cornell University, Ithaca, New York. Assisted in graduate courses in numerical analysis and picture processing (Sept. 1969 - May 1971).

Computer Programmer, Scholastic Magazines, Inc., Englewood Cliffs, New Jersey (Aug. 1970 - Sept. 1970).

Research Associate, Institute of Inorganic and Physical Chemistry, University of Stockholm, Stockholm, Sweden. Developed programs for crystallographic calculations and for illustration of crystallographic structures (Sept. 1968 - June 1969).

Research Assistant, Computation Group, Stanford Linear Accelerator Center, Stanford, California. Assisted in pattern recognition and graphical data processing research (July 1967 - Jan. 1968).

Computer Programmer, SAAB, Technical Department, Stockholm, Sweden. Classified work for the armed forces (June 1966 - Oct. 1966).

Vice-President of the Brown University Student Chapter of the ACM (Sept. 1971 - May 1973).

Theoretical Editor, Brown University Computing Review, Published by the Brown University Student Chapter of the ACM (Jan. 1972 - Dec. 1972).

Formal Publications

"Planar Geometric Projections and Viewing Transformations" (with J. Paciorek), to appear in Computing Surveys 10, 4 (Dec. 1978)

"Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH (Part II: General Methodology and the Proposed Standard)" (with D. Bergeron, J. Foley, P. Bono, T. Dreisbach, J. Michener, E. Sonderegger, A. van Dam), Computer Graphics 11, 3 (1977)

"A Microprogrammed Satellite Graphics System" (with G. Stabler, K. Magel), Proc. of SIGPLAN - SIGMICRO Interface Meeting, Harriman, New York (1973)

Internal Reports

"THESEUS, Preliminary Reference Manual", Brown University, Providence, Rhode Island (Nov. 1977)

"Survey Report on Vector General 3400, Adage GP/400, and Evans and Sutherland Picture System" (with J. Foley, A. van Dam, R. Burns, H. Webber), prepared for Vector General, Inc. (May 1976)

"A Comparison between LSD2, Algol W, HAL/S and SUE", Brown University, Providence, Rhode Island (Jan. 1976)

"LSD2 Language Reference Manual", Brown University, Providence, Rhode Island (June 1975)

"Algorithms for Transforming PDL-Expressions into Standard Form and into a Primitive Connection Matrix", Stanford Linear Accelerator Center Computation Group, Stanford, California, CGMT 38 (Febr. 1968)

Professional Associations

Association of Computing Machinery (ACM), ACM SIGGRAPH

Honors and Awards

Sigma Xi; Scholarship from Thanks to Scandinavia, Inc. (Sept. 1970 - May 1971)

TABLE OF CONTENTS

Acknowledgements

The research for this thesis could not have been accomplished without my advisor, Andries van Dam; his support is greatly appreciated.

A very special thanks goes to James Michener of Intermetrics, Inc., who, during Andries van Dam's sabbatical, acted as my thesis advisor. His guidance, encouragement, and criticism of this work and his many careful readings of this document were invaluable.

Many other people deserve mention. Robert Sedgewick of Brown University and James Foley of The George Washington University gave this thesis critical reviews and provided many constructive suggestions for its improvement. Dick Bulterman of Brown University collected most of the data concerning the VG3400 and, in addition, gave many, very helpful suggestions. Jeffrey Buzen of BGS Systems, Inc., provided invaluable assistance regarding the performance modelling techniques described in Chapters 4 and 5. Janet Michel, Russell Burns, and Harold Webber, all of Brown University, have, during this work, given many helpful suggestions, in particular regarding the design of Theseus.

It was a great pleasure to be able to use two existing, commercial graphics systems as case studies for this work. I want to thank Evans and Sutherland Computer Corporation and Vector

General, Inc., for making the necessary information available, and I want to thank individually Gary Watkins, Roy Keir, and Mickey Mantle of Evans and Sutherland Computer Corporation, and Allen Leinwand, Lou Schaeffer, and Neal Johnson of Vector General, Inc., for providing the material and for answering what must have seemed to be an endless number of questions.

Finally, I want to mention the staff of the Computing Laboratory at Brown University who provided a lot of special service to help me meet deadlines.

The research reported here was supported in part by the National Science Foundation Grant No. MCS-76-04002, by the Office of Naval Research Contract N14-75-C-0427, and by Brown University.

1 INTRODUCTION

1.1 The Subject Area of the Thesis

This research is concerned with high-performance vector graphics systems. The term "high-performance" is frequently used in the literature, although rarely is it precisely defined. Traditionally, graphics systems have been measured primarily by the number of vectors or inches of vectors that can be displayed without flicker. Although these measures are still relevant, other distinguishing parameters of current high-performance systems are their functional capabilities. In this thesis, therefore, a high-performance graphics system should have capabilities for:

- defining object primitives such as lines and text in 2D and 3D user coordinates,
- hierarchical object definition, which includes an object call stack for transformations, attributes, and status information,
- 2D and 3D modelling transformations: rotate, scale, translate,

- viewing transformations: perspective and orthographic projections, clipping, window to viewport mapping,
- interaction handling: sampling devices, event-causing devices, and access to the object call stack,
- dynamic updating of 2D and 3D objects in real-time,
- dynamic updating of the views of 2D and 3D objects in real-time.

The key features that distinguish a high-performance graphics system from a "medium-performance" system are facilities for dynamic updating of elements of a complex object and views of an object in real-time. If a system combines the functional capabilities listed above with capabilities for flicker-free refresh of a large number of lines and with the capabilities for smooth dynamic motion of a large number of lines, then we shall refer to it as "high-performance". In the following paragraphs the functional capabilities will be elaborated upon.

An object is defined in its own local coordinate system, independent of the coordinate system of the display screen. Objects can be combined into new objects. An object is constructed in a hierarchical fashion; each object consists of primitives (e.g., lines, text), and references to other objects. A reference to an object (an <u>instance</u> of the object) has an associated <u>modelling transformation</u> that positions the instance in the coordinate system of the higher-level object. In order to produce an image of an object, a _viewing transformation_ maps the portion of the object that is to be viewed to the display screen.

The view of an object as well as the object itself can be modified dynamically in real-time. In order to create a sequence of views of the object, the viewing transformation is modified repeatedly. In order to modify the object, either the primitives, the subobject references, or the modelling transformations are changed. Thus a high-performance graphics system allows access to the viewing transformation and all parts of the object definition.

Some simple types of user interactions are highlighting of "picked" primitives, keyboard echoing, and light-pen tracking. More sophisticated functions would be smooth, apparently continuous modification of coordinates or transformations according to the value of some analog input device, or conditional display of parts of an object depending on the state of some binary valued input device. A high-performance graphics system, as defined here, is capable of dynamic response to all common sampling and event-causing devices.

In the remainder of this thesis, the term "graphics system" will always refer to a high-performance vector graphics system, unless otherwise noted.

1.2 The Problem and Its Importance

In all system design activities there are trade-offs between functional capabilities and performance, and performance is often sacrificed for better functional capabilities. In graphics systems, as well as in other real-time systems, however, this is generally undesirable. If a graphics system cannot update the required number of lines in real-time or refresh the required number of lines without flicker, the system is inadequate regardless of its functional capabilities. For this reason, it is very important to be able to evaluate the cost of functional capabilities in terms of reduced performance.

Today's graphics systems are architecturally very complex, with processing power distributed over several processors. Because of the complexity of these systems, the trade-offs between performance and functional capabilities are generally not well understood. This section outlines briefly some of the problems faced by designers, programmers, buyers, and users of these systems.

The processing of an object from its representation in a user application data structure to the image on the display screen can be done in a pipelined fashion. Indeed, in the graphics systems that are the topic of this thesis, this is always the case. Each stage in the pipeline maps one representation of the object to another. By implementing each stage of the pipeline in a different processor, the processing of the different stages can be overlapped. This increases the rate of processing, which, in turn, increases the dynamic capabilities of the system. One side-effect of pipelining, however, is that the visual effect of a change to a representation is delayed because it must go through each subsequent mapping.

When an object is processed in several stages, the user interaction with the system can potentially directly affect the object's processing at any stage. The modification of the object in one representation must then be reflected into the other representations (at both higher and lower levels). Thus, one side-effect of processing objects in a pipelined fashion is that interaction handling becomes more complex.

The variety in approaches to the design of the processing pipeline and of the user interaction handling makes comparisons of graphics systems very difficult. There are no generally accepted comparison criteria for graphics systems to aid in the evaluation of the different types of systems. The alternatives are described only by timings for the individual processors in the pipeline, and these timings often do not correspond from one system to another. One vendor [VECT78b] provides measures such as "average time to process 3D line segments" and "average time to process 3D clipped line segments", and another [ESCC77] measures for "processing of non-visible lines", "processing of completely visible lines", "processing of lines with one endpoint clipped", and so on.

Even equivalent measures of processing power for individual processors would be of little help in the evaluation of the different types of graphics systems. The reason is that these measures take neither the concurrent processing in the pipeline nor the memory and bus contention into consideration. Any meaningful comparisons of graphics systems must be based not only on the processing power of the individual processors, but also on the processing capabilities of the system as a whole.

The evaluation of a graphics system is further complicated by the fact that most systems are micro-programmable. The evaluation cannot be limited to basic architecture, but must cover any new architectures that are obtained by altering the user instruction set in one of the processors. Furthermore, there is no existing method for determining quantitatively if the performance for a particular application can be increased significantly by changing the existing microcode.

Up to this time no methodology for the design of high-performance graphics systems has been published. The performance issues concerning dynamic capabilities and interaction handling capabilities have never been investigated. For example, there are no guidelines for what degree of processing ability each processor in the pipeline should have, or what is a good functional distribution between the processors in the pipeline.

The solutions to these problems clearly depend on the application program, so a certain amount of flexibility in dividing the processing among the processors should be left to the applications programmers. Indeed, the programmer is faced with "division of labor" problems similar to those of the system designer.

1.3 The Approach to the Problem

One of the key problems in both evaluation and design of graphics systems is the definition of performance. As was discussed in the previous section, hardware measures such as average line processing time, taken alone, are inadequate. The performance is of course influenced by the processing speed of the individual processors, but it is also influenced by the way these processors are interconnected.

Performance is also highly dependent on the task that the system is to perform. Performance, therefore, can be discussed only in the context of a particular application or, perhaps, a class of applications. To summarize, performance of a system is determined by:

- the performance of the individual components,
- the system structure, and
- the requirements of the application run on that system.

This thesis presents a functional model and a set of performance modelling techniques for graphics systems. The functional model defines the logical structure of such graphics systems. The model forms a basis for understanding, comparing, and contrasting graphics systems, and for illustrating the functional similarities and differences between systems. With this tool it is also easy to identify the system components and relationships that affect system performance.

The performance modelling techniques are used to define some quantitative performance measures for graphics systems. These measures are functions of the speed of individual hardware components, of the processing and I/O overlap in each component, of the processing overlap between components, and of the application requirements.

The performance modelling techniques are the basis for the solution to many of the problems discussed in the previous section. Performance models are used to develop some comparison criteria for graphics systems, and one particular model is used to illustrate the differences in performance between two existing systems that result from their different functional capabilities.

Another performance model is used to evaluate a proposed new set of user instructions and software constructs that combine the functional capabilities of the existing systems. Finally, the performance modelling techniques are used as an aid in developing a methodology for the design of hardware, firmware instruction set, and software for high-performance graphics systems.

Although the emphasis in this thesis is on high-performance devices, much of the methodology could be applied successfully to medium- and low-performance systems. Because of the many similarities between the processing of an object in a vector and a raster graphics system, it is also believed that the techniques discussed here, after modification to account for different internal object representations, could be applied to raster graphics systems.

1.4 Contributions

The major contribution of this thesis is a methodology for the evaluation and design of graphics systems. This methodology addresses issues of hardware, firmware, and software design. The methodology is used to evaluate existing graphics systems and to propose features as part of the design of a new graphics system. In particular, issues concerning dynamics and interaction handling are considered. The methodology is based on a functional model and a set of performance modelling techniques.

The functional model is used:

- to illustrate the basic characteristics of high-performance vector graphics systems,
- to illustrate the functional similarities and differences between two existing graphics systems, and
- to define the structure of the graphics systems so that the components of the systems and the relationship between the components that significantly affect the system performance can be identified.

The performance modelling techniques are used:

- to develop performance models that express the relationship between the performance measures and the system structure and application requirements,
- to define comparison criteria for existing graphics systems,
- to analyze trade-offs between functional capabilities and performance in two existing systems,
- to evaluate a new set of user instructions and software constructs for a graphics system.

1.5 Outline of the Thesis

Chapter 2 discusses research that relates to this thesis. This work includes performance models for graphics satellite systems, methodologies for optimal distribution of functions between a host and an intelligent display device, and a proposed new architecture for interactive graphics systems. This chapter also gives a brief overview of some past and current work in software design for graphics systems.

Chapter 3 presents the functional model. It discusses two existing high-performance graphics systems, the Vector General, Inc. 3400 (VG3400) and the Evans and Sutherland Computer Corporation Picture System 2 (E&S PS2), and illustrates how these systems relate to the functional model. Chapter 4 discusses some performance modelling techniques suitable for high-performance graphics systems, and Chapter 5 discusses how to derive a performance model using the techniques in Chapter 4.

In Chapter 6 trade-offs between functional capabilities and performance are discussed, and the performance model is used to develop some comparison criteria for graphics systems. The trade-offs are discussed using the E&S PS2 and the VG3400 as examples.

Chapter 7 treats the methodology for hardware and firmware design. In Chapter 8 an access method called Theseus is introduced for a hypothetical system that is suggested based on the results of Chapters 6 and 7. The final chapter summarizes the research presented in this thesis and suggests some areas of future work.

Appendix A gives a complete list of architectural features of the E&S PS2 and the VG3400, and Appendix B contains a set of PDP-11 assembly language code sequences that implement some functions on the E&S PS2. A complete description of Theseus can be found in Appendix C.

This thesis assumes that the reader is familiar with computer graphics at the level of [NEWM79] and uses some of the terminology of the Core Graphics System [GSPC77].

2 RELATED RESEARCH

This chapter discusses research that relates to the work of this thesis. Very little has been done in the area of design methodologies for graphics systems. Some notable exceptions discussed below include performance models for graphics satellite systems and methodologies for optimal distribution of functions between a host computer and an intelligent display device. In the area of hardware and firmware design, one attempt made in recent years to develop a radically different architecture for interactive graphics systems will be discussed briefly.

In contrast, there has recently been a lot of activity in the area of software design. Much of this activity has centered around the Core Graphics System [GSPC77, GSPC79]. The Core Graphics System and some graphics packages on which the design of "the Core" was based are reported on below.

The most extensive work in the area of graphics systems performance has been done by Foley [FOLE71], including work in collaboration with others [FOLE74a, FOLE75a]. The earlier work by Foley [FOLE71] concerns the optimum design of highly interactive graphics terminals for use with time-shared computers. The objective was to minimize a display system's response time subject to cost constraints. Foley developed a queueing model that used as input parameters the capabilities of the graphics hardware, such as disk file access time, data link transmission rates, core and disk storage space, and execution rate both of the host and the display controller. Additional input parameters were application characteristics such as the instruction occurrance frequencies, the user think time, the probability of the occurrance of each type of user interaction, the number of disk storage accesses which are made as a result of an interaction, and the length of messages sent over the data link. Foley used the model to establish an optimum configuration for several applications. Finally, these optimum configurations were used to define general display system design quidelines. Foley established that in order to lower response time, the components should be upgraded in the following order:

1) data link,

2) bulk storage,

3) display controller, and

4) core storage.

Foley's work was continued at the University of North Carolina by Foley et al. [FOLE74a, FOLE75a]. The queueing model of the earlier work was extended to treat issues of hardware/software trade-offs in a satellite graphics system. This extension resulted in algorithms for static assignment of tasks and data structures for a given application to either the host or the satellite, in order to obtain minimum response time to user requests.

Algorithms for dynamic assignment of tasks in an application program to either the host or the graphics satellite in order to minimize response time to user requests have been investigated by Stone [STON77] and by Michel and van Dam [MICH77]. Stone used algorithms based on network flow analysis to show that there always exists an optimal assignment of tasks between the host and the satellite for any load on the host, and developed algorithms to determine this distribution. Stone's work was experimentally verified using the Brown University Graphics System, BUGS, [STAB73], and the degree of performance improvement gained through load balancing was investigated by Michel and van Dam [MICH77].

Cislo [CICL72] developed a simulation model for the evaluation of interactive satellite graphics systems. The model was used to study the effects of the main CPU, the satellite CPU, and the data transmission rates between the two CPU's on response time. The model produced quantitative results demonstrating that the most influential hardware component was the satellite CPU, since the majority of time required to satisfy the user requests was spent in the satellite. Cislo also illustrated that (for the loads under consideration), the rate of the data transmission link had more influence on performance than did the speed of the host CPU.

Another approach to the optimal distribution of software between a host computer and an intelligent graphics terminal is demonstrated by Puk [PUK76]. Puk developed an integer programming model that determines a division of graphics software support between the host and the terminal that minimizes the communications over the data transmission link. The work assumes that the response in the graphics system is constrained only by the capacity of the data transmission link, not by any other aspects of hardware or software. As might be expected, the model shows that as the data transmission rate increases, more functions are assigned to the host computer. Puk uses the model to distribute a subset of the functions in GCS [MIDD74, PUK76] between a CDC 6500 and a display processor.

A new and different vector graphics system architecture is described by Stowell [STOW78]. Stowell proposes a microprocessor-based architecture with a shared memory system. The graphics picture processing pipeline was divided into nine stages, and each stage was allocated to one general-purpose microprocessor. Interaction handling has been ignored completely. The proposed architecture was never implemented, and Stowell unfortunately does not report any attempts at performance comparisons with existing systems.

In the area of graphics software development much attention has recently been focussed on a proposal for a graphics software standard (the Core Graphics System) by the ACM SIGGRAPH Graphics Standard Planning Committee [GSPC77, GSPC79]. The goal of this effort is a system that allows program portability, i.e., the ability to transport application programs from one system to another with only minimal program changes. The Core (as it is known) is a <u>viewing</u> system: an object is defined in an application dependent coordinate system (called the world coordinate system), and the object is mapped to the view surface by a viewing transformation. Functions for building a model of an object (i.e., a <u>modelling</u> system) are not included in the Core, but are the responsibility of the programmer using the Core.

The Core has the following features:[1]

1) Primitives - An application program describes objects to the Core as a sequence of lines, text strings, and markers. Geometrical aspects of primitives are specified in two or three dimensional world coordinates.

2) Attributes - Linestyle, linewidth, intensity, color, and character font specification determine the attributes (i.e., essentially non-geometrical aspects) of primitives.

---

[1]This is a description of the Core as described in [GSPC77]; some changes have been made that are reported in [GSPC79].

3) Viewing transformations - A viewing transformation selects the region of the world coordinate space that is to be displayed and specifies how objects in the selected region are mapped to the view surface. For 3D objects the Core allows specification of <u>all</u> planar geometric transformations [CARL78] as part of a viewing transformation.

4) Segments - Primitives to be displayed are placed in segments. The segments that are associated with a view surface determine the picture displayed on this view surface. A segment is the unit of picture modification: a segment can be added to and deleted from a picture, and the attributes of a segment may be modified. These attributes are visibility, highlighting, detectability and image transformation.

5) Logical input devices - The Core supports five classes of logical input devices: pick, button, keyboard, locator, and valuator. Devices in the first three classes are <u>event-causing</u>, i.e., the use of one of these devices causes an event (sometimes called an attention), that results in an event report containing input values being placed on an event queue. The application program can determine what user interaction takes place by asking for event reports from the event queue. Locator and valuator devices are <u>sampled</u> devices, which means

that at any time the application program can acquire the current values of these devices. Event-causing and sampled devices can be associated. When an event device generates an event, the values of its associated sampled devices are included in the event report.

Although most of the Core functions would be useful in a graphics support package for a high-performance device, as a whole, the Core would have some shortcomings if it were put to such a use. First, the Core does not include modelling transformations and does not allow easy access to powerful transformation hardware. Second, the generality that is necessary in a package designed for application program transportability and device independence has an associated overhead that will, in most cases, reduce its performance compared to software that is more closely tailored to a particular device or class of devices.

Device independence is generally accomplished by dividing the package into a device independent part and a device dependent part. The device independent part treats the display and the input device as logical devices; the device dependent part is usually implemented as separate device drivers, one for each type of device. Each device driver acts as an interface between the physical display and input devices and the logical devices. The idea of device independence originated with GINO [WOOD71a] and was later adopted by GINO-F [CADC75], GPGS [CARU75, VAND77], and GCS [MIDD74, PUK76].

GINO-F is a set of FORTRAN-callable subroutines. GINO-F provides primitives, attributes, and segmentation much like those in the Core. The input devices, however, are entirely event-driven. A set of low-level logical input devices are supported in GINO-F, on top of which most of the Core functions could be built. GINO-F provides both modelling and viewing transformations, although the viewing functions do not include all parallel projections.

GPGS [CARU75, VAND77] was developed from many of the ideas in GINO-F. It carries device independence a step further, however, in that it provides support for all the logical input devices of the Core. GPGS has a modelling package and supports the creation of a hierarchical representation of an object. The object hierarchy is traversed by software. GPGS allows a user to take advantage of transformation hardware in a limited way through <u>image transformations</u> (as does the Core).

GCS [MIDD74, PUK76] is a set of FORTRAN-callable sub-routines. The original package was two-dimensional, but a set of three-dimensional routies has been added. The package provides good support for storage-tube displays, but only limited support for refresh devices. GCS provides a large set of primitives, including curved line and surface primitives, and extensive primitive attributes. Input is handled through one subroutine, which can interface to most common input devices. GCS also provides modelling and viewing transformations, but, as GINO-F, does not include all of the planar geometric projections in its viewing capabilities.

Past research in performance modelling of graphics systems and in optimal distribution of tasks between processors in a graphics system have been focused on intelligent terminals used with time-shared computers and on satellite graphics systems, also with time-shared computers. There exist, up to this date, no methods for evaluation of design trade-offs in high-performance systems, e.g., the distribution of functions between the processors in such a graphics system. Furthermore, no comparison criteria for high-performance systems have ever been developed. As a result, it is difficult to make any realistic, meaningful comparisons of high-performance systems, and existing such systems are not optimal for a large class of applications. The work in this thesis is intended to advance the practice of design and evaluation of high-performance graphics systems.

3 FUNCTIONAL MODEL

The functional model in this thesis illustrates the operation of graphics systems. It illustrates how output data is mapped from one representation to another, starting with the application data structure and ending with an image on the display screen. The model also illustrates how input data flows from the input devices into the various parts of the processing pipeline and how the processing is affected by the input.

There are many models that illustrate the operation of graphics systems. One such model is a block diagram of individual hardware components and their connections. Such a structural model[2], however, does not meet the requirements of a functional model for several reasons. First, it is a model of only one graphics system, and second, a structural model need not illustrate the data flow in the system. In addition, it often contains unnecessary detail that obscures the essential functions of the system. A structural model does not show how a system operates, but is useful in explaining how a functional model applies to an actual system.

The functional model chosen for this thesis satisfies several requirements. First, it forms a basis for comparison of

[2]Structural models and different types of functional models are discussed in [SVOB76].

most existing vector graphics systems. In this sense, it is an abstraction of all these systems. Secondly, it represents the graphics systems at a suitable level of detail for our purposes. With too many details available, the systems would be harder to understand. With too few details, the performance evaluation could not be based on this model since some factors that have a significant influence on the performance of the systems would be omitted. Finally, the model is expandable to new architectures in order that these may be evaluated and compared to existing ones.

The functional model in this thesis has two roles: <u>descriptive</u> and <u>prescriptive</u>. It is <u>descriptive</u> in the sense that it illustrates the operation of a graphics system and <u>prescriptive</u> in the sense that it identifies the important parameters that affect the performance of a system. The functional model will be used both as a tool for describing the functional capabilities of graphics systems and as a basis for performance modelling of these systems.

The functional model in this thesis differs from other such models in, for example, [NEWM79], in three ways. First, <u>one</u> model is used to describe all graphics system; in [NEWM79] different models are used for different architectures which makes comparisons between the architectures more difficult. Second, the model in this thesis illustrates the flow of input data, which is generally not illustrated in other models. Finally, this model maps directly onto the hardware of most existing high-performance graphics systems, i.e., the logical processors in the model correspond to the actual physical processors in the graphics systems.

This chapter describes the functional model. The first section gives a general, high-level description of the model. The second section describes the output portion of the model in more detail and the third the input portion, also in more detail. The last section illustrates how two existing high-performance vector graphics systems, the Evans and Sutherland Picture System 2 and the Vector General 3400, are represented by the model. The following chapter discusses how the functional model is used to derive a performance model.

3.1 Representations, Processes, and Input in the Functional Model

The functional model has two aspects -- output and input. The output portion will be described first ignoring input, and then the input will be added to the model.

The output portion of the functional model is a pipeline consisting of various representations of an object[3] and logical processors that map one representation to another. This pipeline

[3]Conceptually, there is only one highest level object in the hierarchy.

starts with a description of an object in the application data structure and ends with the image of the object on the display screen.

It is immaterial if a logical processor in the functional model corresponds to one or to more than one physical processor, or if two logical processors in the model share a physical processor. Similarly, it is not important if the representations reside in one memory or in different memories.

The output pipeline contains of four representations of objects:

- ADS - the Applications Data Structure -- contains a description of an object in a format that is determined by the application.

- SDF - the Structured Display File -- contains a hierarchical description of an object.

- LDF - the Linear Display File -- contains graphical primitives and mode settings describing the object.

- Display Screen - shows the image of the object.

The pipeline also contains three logical processors, each mapping one representation to another:

- DFC - the Display File Compiler -- maps the ADS to the SDF.
- DPU - the Display Processing Unit -- maps the SDF to the LDF.
- DCU - the Display Control Unit -- maps the LDF to the image on the display screen.

Each representation and processor is described in more detail in the next section of this chapter.

The input portion of the functional model is described by the data flow from the input devices into the different processors in the pipeline. The mapping from one object representation to another is always subject to modification by input. The input/output model is illustrated in Figure 3-1.

The input data is categorized at each processor with respect to two aspects of its processing:

- how it enters the processor -- either directly from an input device or forwarded by another processor.
- how it is processed by the processor -- either the input is forwarded to another processor, it is used by the processor to modify its output, or both.

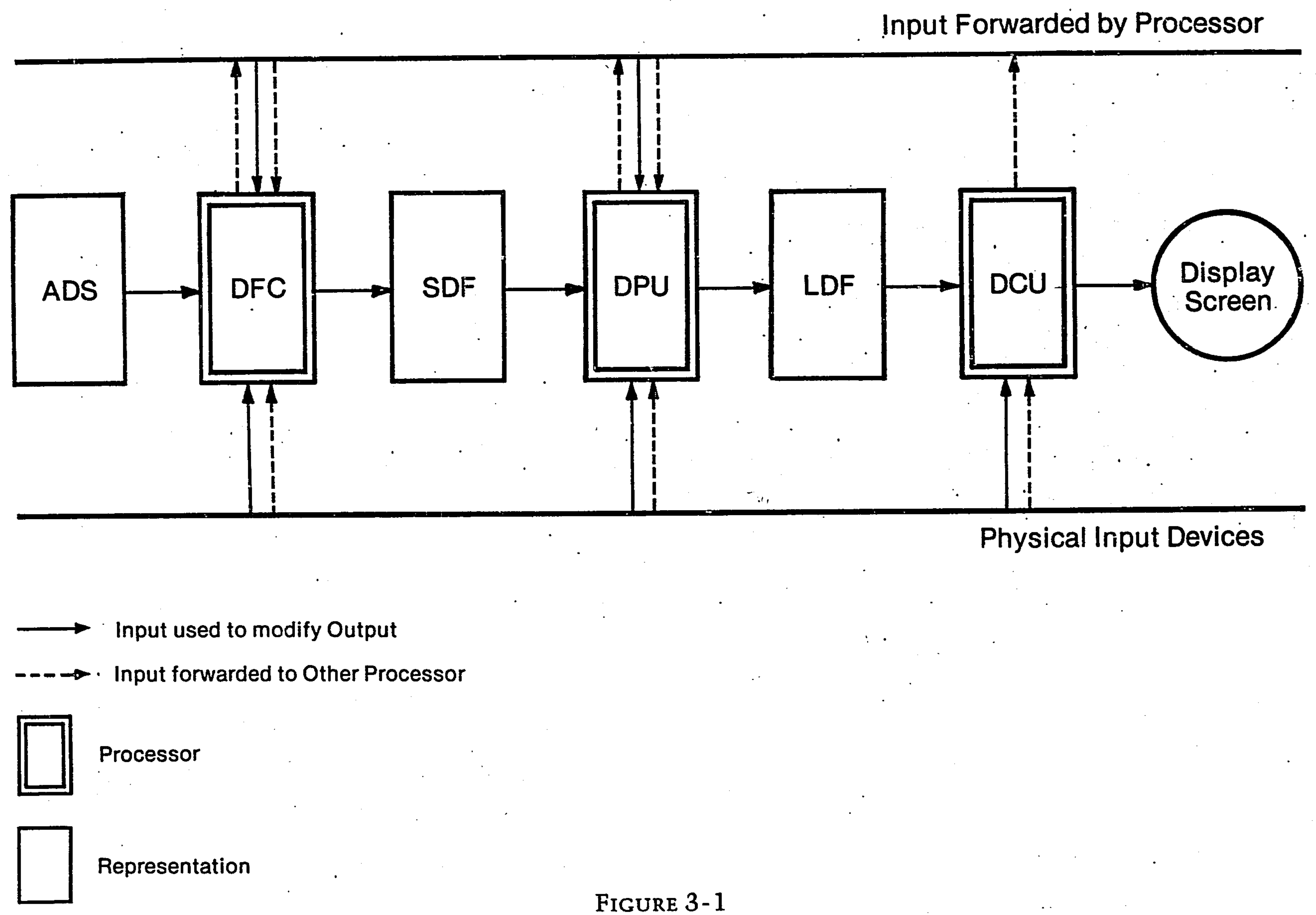

Input Forwarded by Processor
ADS
DFC
SDF
DPU
LDF
DCU
Display Screen
Physical Input Devices
Input used to modify Output
Input forwarded to Other Processor
Processor
Representation

FIGURE 3-1
Functional Model

Each type of input is described in more detail in a later section of this chapter.

As for the output portion of the functional model, the logical operation of the input portion is emphasized. Although an input device may be *physically* connected to several processors over a bus, on the level of abstraction of the functional model it is only *logically* connected to the processor that actually processes the input data. A logical input device can, of course, modify the processing in several processors in the pipeline. The change in one representation reflects in all following representations in the pipeline and thereby changes the processors' behavior.

The functional model represents a unified approach to describing graphics systems. Each representation contains a complete description of what is displayed. The representations differ only in the level of abstraction of object description. Each processor operates on a representation, and may also receive input from input devices either directly or through another processor. The input differs from one processor to another principally according to the level of abstraction of the representation upon which the processor operates. Input into the DFC can cause changes in terms of application objects; input into the DCU causes changes in terms of individual characters or lines. Each processor maps one representation of an object to another. The input may alter this mapping each time it takes place, that is, the binding of some variables only takes place at the time of mapping.

The functional model is general enough to describe most existing graphics systems. In each graphics system, an object is represented in a graphics-system-independent fashion in an ADS. Similarly, in each graphics system, an object is represented, at the end of the pipeline, as an image on the (conceptual) display screen. In order to map the ADS to the image on the screen, a set of successive reductions of the object representation in the ADS must take place. As a first step, the object is represented in a format such that it can be processed by the transformation hardware. This representation is the SDF[4]. Also at some point in the pipeline, the object must be represented in a format such that it can be processed by the digital-to-analog hardware. This representation is the LDF. The intermediate representations need not all exist in the form of data structures, but do always exist as a sequence of data in time, that is, the data is generated regardless of whether this generated data is actually stored in a data structure. It is easy to see that this modelling technique can be extended to include more internal representations should this be required to describe new architectures.

[4]Note that the hierarchy might have only one level.

The functional similarities and differences between existing systems are best understood by the number of object representations actually occuring in the form of data structures (as opposed to sequences of data in time), and by the complexity of each representation. One example of an object representation that does not always occur as a data structure in a high-performance system is the LDF. The VG3400 has a LDF as a data structure, whereas the Adage 4100 does not.

The differences between the "display procedure approach" and the "structured display file approach" to object description have been debated much during the past few years. These two approaches are easily understood in terms of the functional model. Display procedures [NEWM71] is a method of defining the structure of objects by nested procedures in the display file compiler rather than by a graphical data structure. Pure display procedures have no data structures at all. They are illustrated by the functional model in Figure 3-2. In the structured display file approach the complete picture is described in only one data structure, the SDF. A structured display file system is illustrated in Figure 3-3.

All eight possible combinations of data structures allowed by the model may be found in existing graphics systems.

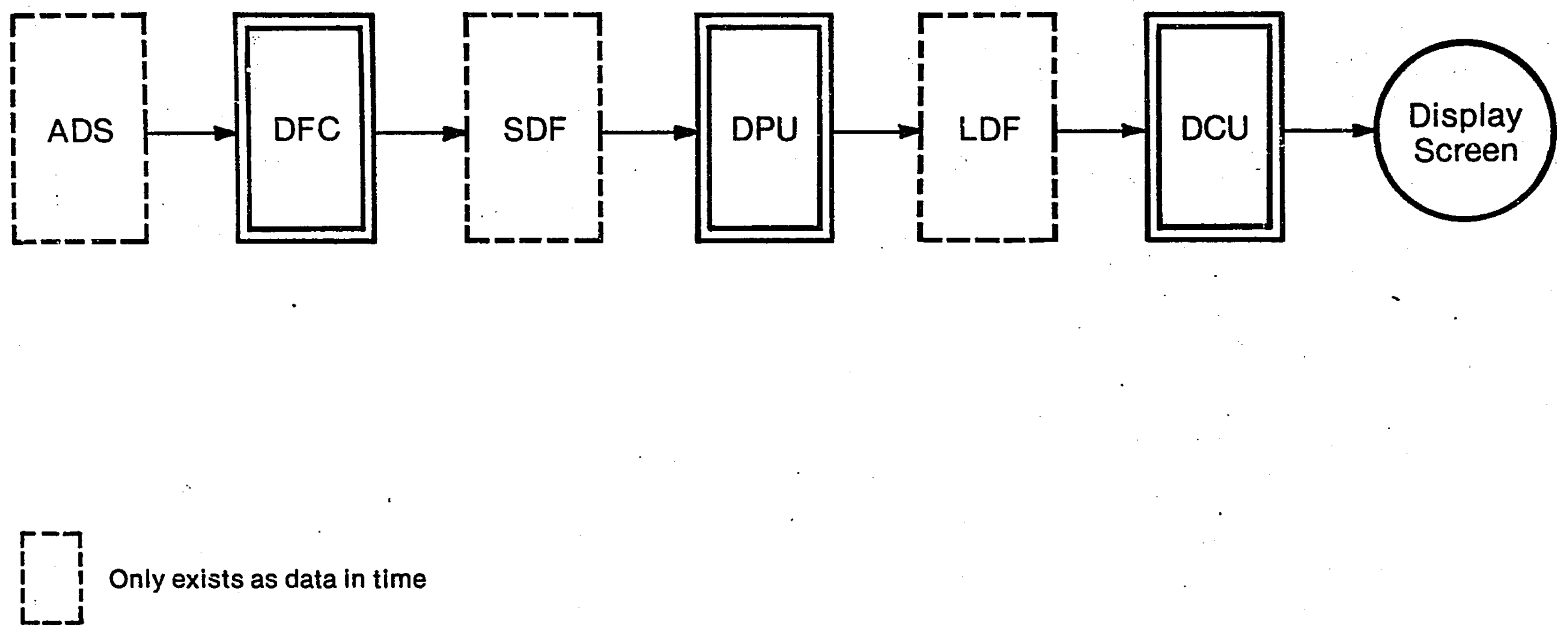


FIGURE 3-2
Display Procedure System

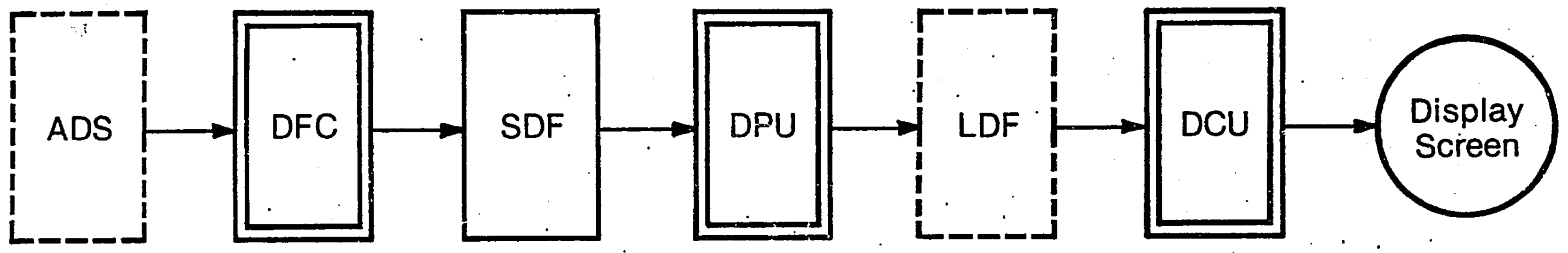


FIGURE 3-3
Structural Display File System

Data Structures	Graphics Systems
ADS, SDF, LDF	E&S PS2 [ESCC77, ESCC77a], VG3400 [VECT78a, VECT78b]
ADS, SDF	Brown University Graphics System (BUGS) [STAB73] Adage 4100 [ADAG75a, ADAG75c, ADAG78]
ADS, LDF	Evans and Sutherland Picture System [ESCC74] (the system was manufactured with an optional refresh buffer for the LDF)
SDF, LDF	Evans and Sutherland LDS-1 [ESCC70] (the system optionally allowed the user to create a LDF in the PDP-10 memory)
ADS	Evans and Sutherland Picture System [ESCC74] (without refresh buffer) Evans and Sutherland LDS-1 [ESCC70] (using its general purpose instruction set)
SDF	Evans and Sutherland LDS-1 [ESCC70] (using only its graphics instructions; a pure structured display file system)
LDF	Euler-G [NEWM71] (implemented a modified version of the "pure" display procedures with a refresh

	buffer)
none	"pure" display procedures.

3.2 Output in the Functional Model

This section discusses the output portion of the functional model. The properties of each data structure and processor in the pipeline are treated in detail. The data structures and the processors are illustrated in Figure 3-4.

3.2.1 APPLICATIONS DATA STRUCTURE

The *applications data structure* (ADS), is a *high-level*, *general purpose* data structure. It may mix graphical and non-graphical data. The part of the ADS used by the DFC is by definition the graphical data. It consists of output primitives, attributes, transformations, and control information. The rest of the ADS is the non-graphical data; it may, for example, be information associated with part of an object or pointers relating certain parts of an object. The format of the ADS is *independent* of the graphics system hardware. If a higher-level language like FORTRAN or PL/I is available in the CPU, an array or an array of structures may be used to express the ADS.

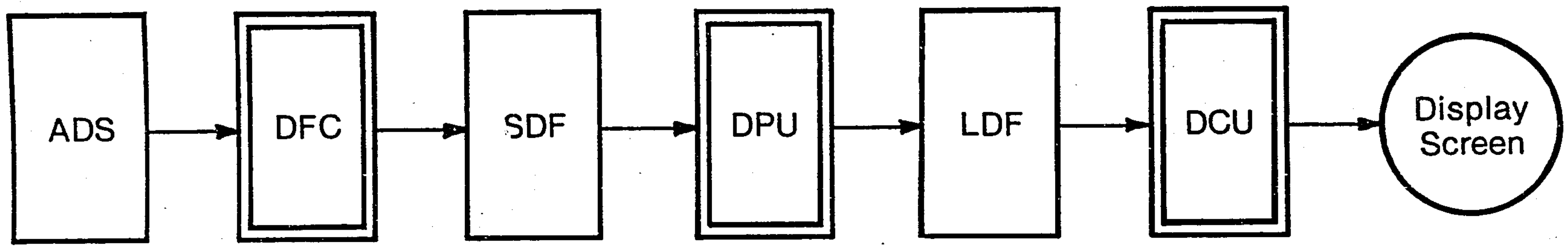


FIGURE 3-4
Output Portion of Functional Model

Applications range from cases in which the ADS contains a detailed description of an object to cases where no ADS is needed. A detailed description of the object may be necessary for several reasons. One reason would be that the object, for the sake of precision, must be represented by floating point data. This is not possible in the SDF, since processing of floating point data is very slow with today's technology. Another reason would be that the object must be described by a data structure not supported by a SDF, such as a ring structure. A third reason would be that the object is very large and only a small portion of it is visible at any one time. The SDF would then contain an ADS "distillate". (This last problem can sometimes be eliminated by <u>extents</u>, a construct that allows conditional execution of parts of a SDF. Extents are discussed in a later chapter.)

An ADS may not be needed. The object could be completely described by procedures in the DFC or by the SDF. The latter would be the case, for example, in some CAD applications. The object could be built from subobjects that are all defined in the SDF and there would be no need for a second representation of the object in the ADS.

Of course, there are many ADSs that are between the two extremes. An ADS may contain only non-graphical data or it may contain a highly compact description of the object. The contents and format of the ADS are entirely dependent on the type of application and the types of access that are neededd to the various parts of the object.

3.2.2 DISPLAY FILE COMPILER

The display file compiler (DFC) maps the ADS into the structured display file. The display file compiler is implemented by the user. It compiles the ADS using system-provided functions to produce the various parts of the SDF. These system functions usually include add, modify, and delete for primitives, attributes, objects, and object calls in the SDF.

3.2.3 STRUCTURED DISPLAY FILE

The structured display file (SDF) is a special purpose data structure containing a hierarchical definition of an object, which is display device independent. The SDF is graphics system dependent, although the general format of the SDF is often quite similar from one system to another. The SDF is analogous to a program structure consisting of procedures which may call other procedures. A SDF consists of objects which may reference, or call, other objects.

An object in the SDF is represented by output primitives, such as lines and text, object calls, attributes, such as line style and intensity, and transformations. Modelling transformations, also called object construction transformations, allow objects to be defined in a local coordinate systems such that these objects can be combined into new objects. Viewing transformations define how an object or portion of an object is mapped to the display surface. In some systems the SDF also contains general purpose instructions such as arithmetic, boolean, and flow of control operations. Systems vary a great deal in their support for text primitives, addressing modes of primitives and general purpose instructions.

The SDF, as was discussed in the last section, does not always exist except as a sequence of data in time. The object is in this case completely described either in the ADS or by nested procedures in the DFC.

3.2.4 DISPLAY PROCESSING UNIT

The display processing unit (DPU) maps the SDF into a linear display file. The DPU traverses the hierarchy in the SDF composing transformation matrices, applying these transformations to the graphical data, and clipping the data to the specified window. As an aid in interpreting or compiling the hierarchy, the DPU maintains a stack for transformations and attributes and a directory of the objects.

3.2.5 LINEAR DISPLAY FILE

The _linear display file_ (LDF) is a _low-level_, _special purpose_ data structure containing primitives and mode settings. The format of the LDF is highly dependent on the graphics system and the display device. Some systems support a segmented LDF and allow modification of portions (segments) of the LDF. If two segments are allowed, the recompilation of the SDF may be double buffered for better dynamics. If multiple segments are allowed, some portions of the object can remain unchanged while other parts are modified. In this case only the parts of the SDF that are altered need be recompiled.

The LDF, as was discussed in the previous section, does not always exist except as a sequence of data in time. The object is then described by a SDF, ADS, by nested procedures in the DFC, or by a combination of these.

3.2.6 DISPLAY CONTROL UNIT

The _display control unit_ (DCU) refreshes the picture to maintain a flicker-free image on the display screen. It fetches coordinate, text, and attribute data from the LDF. The DCU contains a line generator and a character generator that produce analog signals which are used to position the electron beam and control the intensity in the monitor.

3.3 Input in the Functional Model

The foregoing discussion on output totally ignored the input portion of the functional model. This section examines how input fits into the functional model. The physical input devices considered are:

- data tablet - flat surface on which user draws with a stylus and that produces (x,y) coordinates,
- joystick - lever that produces (x,y,z) coordinates,
- lightpen - pen-like device sensitive to light that produces an interrupt,
- alphanumeric keyboard - text entry device,
- control dial - potentiometer that produces coordinate data,
- function key - key that causes an interrupt when pressed,
- hardware pick - hardware x-y comparator that produces an interrupt.

It should be noted that one physical input device might correspond to two logical input devices, each with a different function. The two logical input devices can either be connected to the same processors or to two different ones. For example, a user may control the movement of a cursor with a tablet and also read the (x,y) coordinates of the tablet from the diplay file compiler when some specific event occurs. This is considered to be done with two logical devices:

- The DCU reads the tablet coordinates and updates the cursor position. The tablet input is to the DCU.

- The DFC reads the tablet coordinates. Logically, the tablet input is to the DFC. Depending on physical connections, the DFC may access the input device directly, or it may obtain the input data from a lower-level processor.

Some input details of the functional model are illustrated in Figure 3-5.

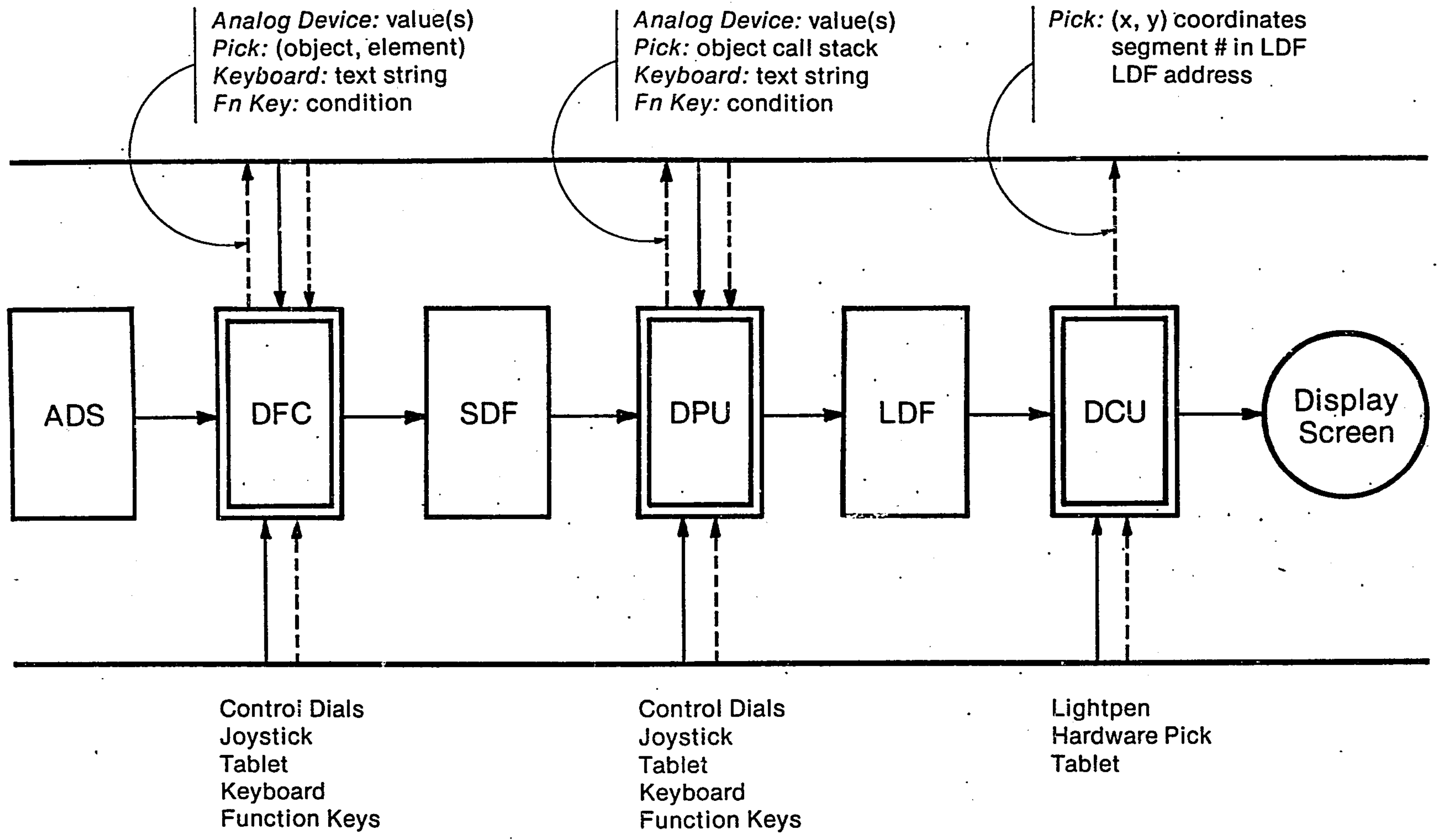


FIGURE 3-5
Functional Model — Input and Output

As was discussed in a previous section, input is categorized at each processor with respect to two aspects of its processing; by how it enters the processor, and by how it is processed by that processor. Some input data is only collected by the processor for transfer to a higher level processor possibly after some transformation of the data. Other input data is used by the processor to modify how it maps one representation into another. The first type of input is illustrated by the dashed lines in Figure 3-5, the second type by solid lines.

All three processors can receive input data. The data may be processed before it is passed on to a higher-level processor. For example, characters may be formatted into a text string, control dial values may be filtered, and lightpen data may be used to identify the item in the next higher-level data structure that corresponds to the item that was identified on the screen by the lightpen. The input data may also be used to modify the output of the processor. The input data only varies among processors in the types of changes it may cause to the output of the processor.

In principle, most physical devices could be connected to each processor. Two exceptions are the lightpen and the hardware pick that can only be connected to the DCU. The lightpen is a light-sensitive device that gives an interrupt upon a hit. Because of the overlapped processing in the pipeline, only the DCU can identify the item that was picked. Similarly, since the hardware pick uses the output of the DCU as the input for the comparator, only the DCU can identify what item was hit. In practice, the input devices that are connected to the DCU are quite restricted. Input into each processor is discussed below.

The input into the DFC can cause the most general type of changes, i.e., changes in terms of application objects. These changes range from modification of individual items in the SDF to modification of the structure of the SDF. The DFC can receive input directly from all physical devices, except the lightpen and the hardware pick, and can receive input forwarded from lower-level processors.

The input into the DPU is more restricted than that to the DFC in the type of changes it may cause. The processing of individual items, such as coordinate data, transformation data, and flags in the SDF can be modified, and, as a result of a change to a transformation, an entire instance of an object can be affected. Flags may be used to select for compilation one or another part of the SDF.

The input capabilities of the DPU vary a great deal from one system to another. In some cases all devices except the lightpen and the hardware pick may be processed. In other systems the DPU may only maintain a cursor and a text buffer, and, in still others, the DPU may not do any input handling.

The input into the DCU affects the mapping of the LDF to the image on the display screen. The input to the DCU can only cause very restricted types of changes to the image. Only individual coordinate positions and attributes can be modified. For example, primitives can be high-lighted upon a pick, a cursor position can be updated, or a lightpen tracking cross position can be updated. The only devices that generally are attached to the DCU are the lightpen and the tablet.

In the following section, both the output and the input processing in two existing graphics systems are described in terms of the functional model.

3.4 Case Studies

This section provides a summary discussion of the structural models of the E&S PS2 and the VG3400. The hardware components of the systems are discussed and the relationships between the structural models and the functional model are illustrated. The structural model of the E&S PS2 and its structure on the level of abstraction of the functional model are illustrated in Figures 3-6 and 3-7, and the two models for the VG3400 are illustrated in Figures 3-8 and 3-9, respectively. Only the features essential to this thesis are discussed in this section; a complete list of architectural features can be found in Appendix A.

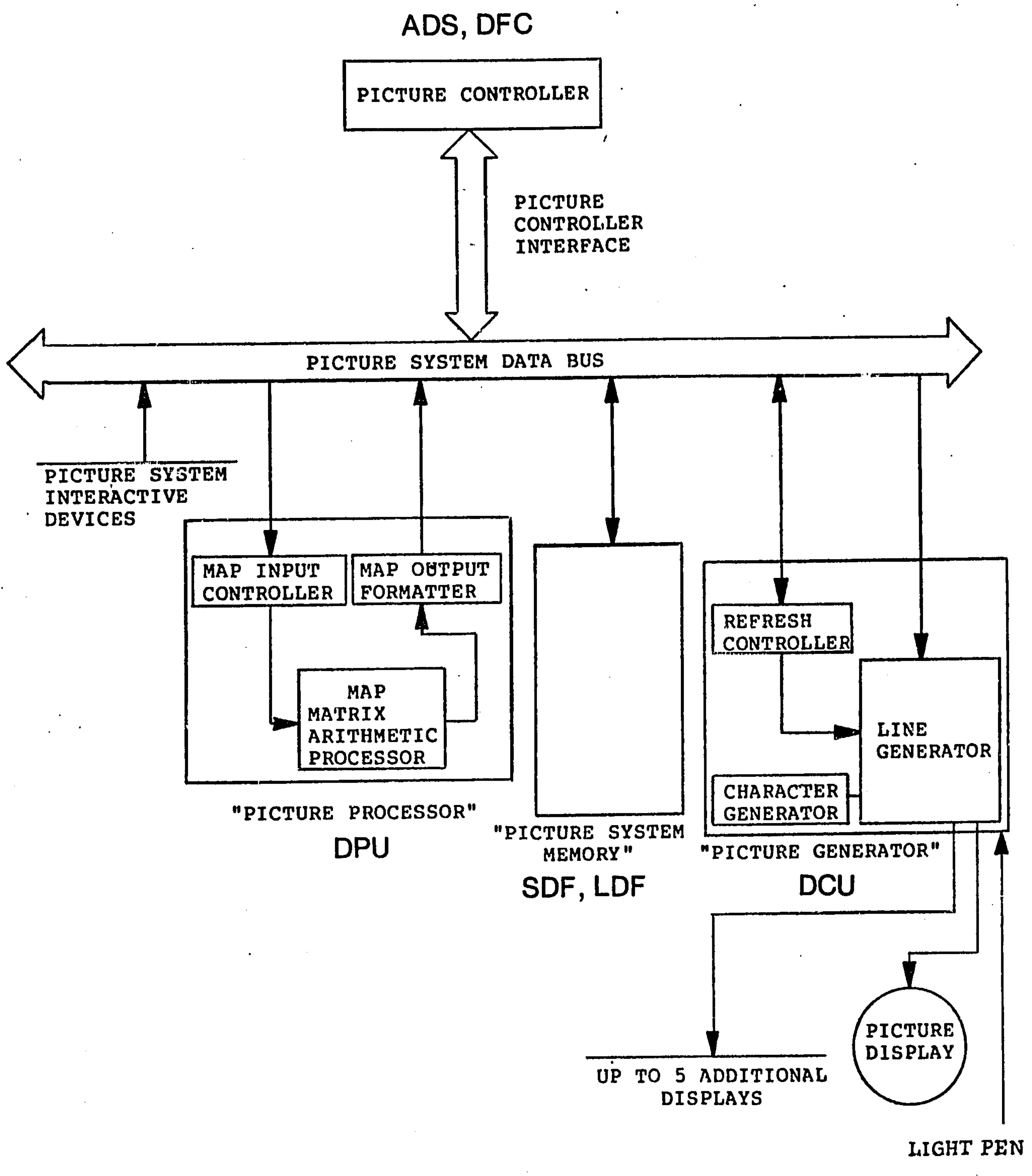


FIGURE 3-6
E&S PS2 Structural Model
(Evans and Sutherland Computer Corporation, PS2 Reference Manual)

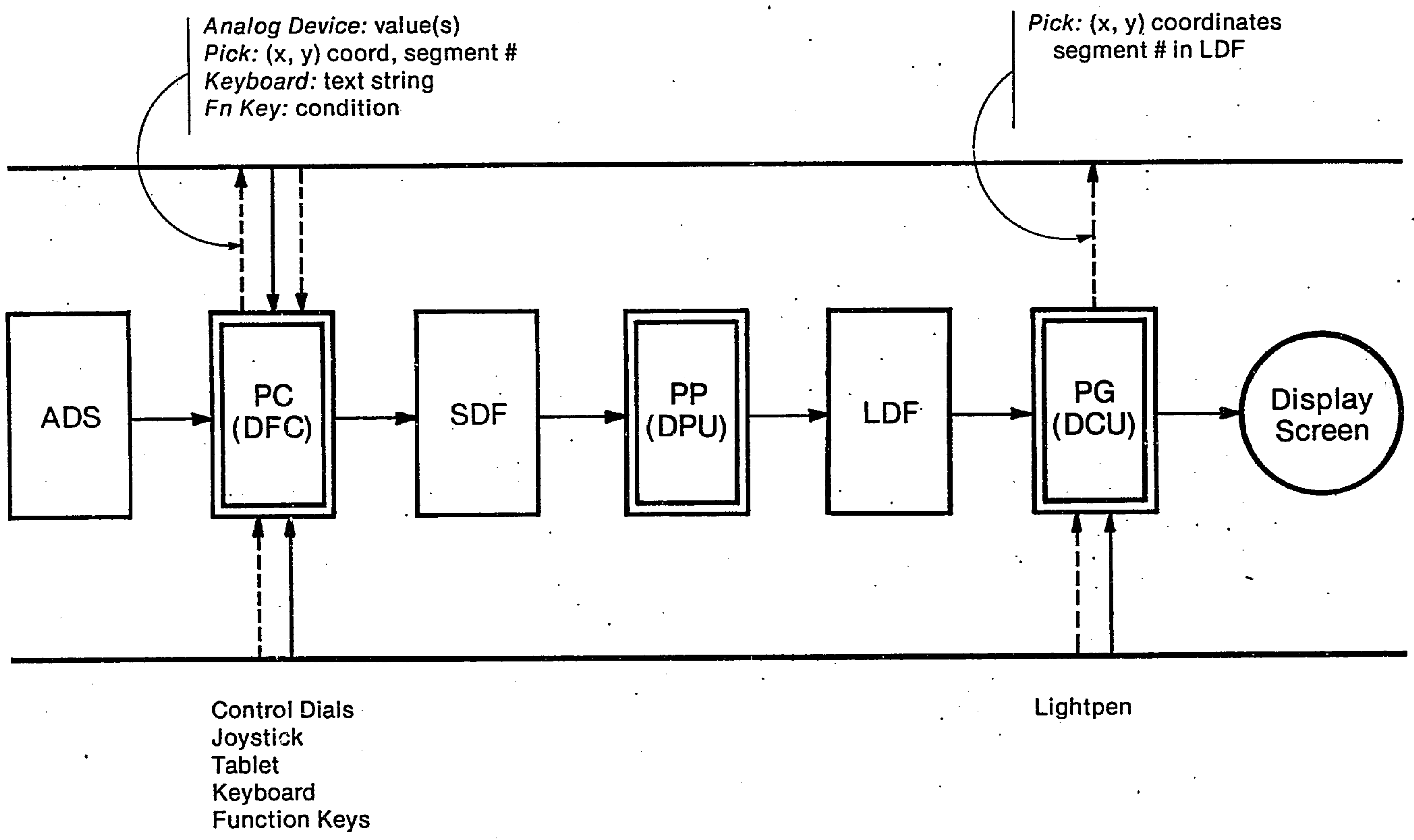


FIGURE 3-7
Functional Model: E&S PS2

Note that in general vendors' names for components refer to hardware units, while the names defined above for logical processors and representations are reserved for parts of the functional model.

3.4.1 EVANS AND SUTHERLAND PICTURE SYSTEM 2

The E&S PS2 [ESCC77, ESCC77a] consists of six components:

- the Picture Controller Interface
- the Picture Data Bus
- the Picture Processor
- the Picture Memory
- the Picture Generator
- the Interactive Devices.

These components are connected to the Picture Controller, which, as this thesis is concerned, is part of the total display system.

The Picture Controller is a general purpose computer, typically a mini-computer such as a DEC PDP-11. The Picture Controller contains the application program, the ADS, and the DFC. The E&S supplied software includes a set of FORTRAN-callable subroutines, that are used by the DFC to build the SDF from the object representation in the ADS and to modify the SDF according to input data.

The Picture Controller interfaces with the E&S PS2 through the Picture Controller Interface. The interface unit handles communication between the Picture Controller and the E&S PS2 over three I/O paths:

- the direct I/O path (DIO),
- the direct memory access path (DMA),
- the interrupt path.

The DIO path is used to transmit a single word of data to or from the E&S PS2 and the DMA path is used to tranfer blocks of data. The interrupt path transmits interrupts to the Picture Controller from the input devices and from other units such as the Picture Generator.

The list of components above includes a synchronous bus, the Picture Data Bus. The interface enables the Picture Controller to access all the components of the E&S PS2 over this bus.

The Picture Memory has a dual purpose in the E&S PS2:

- to store the SDF, and
- to store the LDF.

The SDF contains primitives, attributes, object calls, and viewing and modelling transformations. The reader should refer to Appendix A for a complete list of SDF features.

The Picture Processor is the DPU. It compiles the SDF in the Picture Memory to a LDF, also stored in the Picture Memory. The Picture Processor composes transformations, applies the transformations to the primitives, and clips the portions of the transformed object that are outside the window. The clipped object is projected onto the projection plane, and the resulting projection is mapped to the viewport.

The LDF resides in the Picture Memory and contains primitives and mode settings. The LDF may be single buffered, double buffered or segmented. When the LDF is in double buffer mode, the DCU continually reads one half of the buffer while the DPU updates the other. In segmented buffer mode, the LDF is divided into several segments, each of which can be individually updated.

The Picture Generator, Line Generator, and Character Generator function together as the DCU. The Picture Generator controls the processing of the LDF. It commands the Line Generator to convert the data in the LDF to analog signals that position the electron beam and control the intensity in the display monitor, and it commands the Character Generator to interpret the full ASCII character set. The user also has the option of defining new fonts. Individual characters in a text string are not subject to the viewing/modelling transformations, only the starting point of a text string is transformed.

The E&S PS2 supports seven interactive devices, all of which are interfaced directly to the Picture Data Bus. The input devices are:

- Data Tablet
- Control Dials
- Function Switches
- Alphanumeric Keyboard
- Lightpen
- Joystick
- Lighted Function Buttons.

With the exception of the lightpen, all devices can be accessed directly by the DFC in the Picture Controller. Some input data is passed on to the application program, and other input data is used as input to the DFC.

The Picture Processor can not accept any input data at all, and the Picture Generator has access only to the lightpen.

The E&S PS2 is one in a series of graphics systems built by Evans and Sutherland Computer Corporation. The Picture System, the first in the series, did not represent the SDF as an actual data structure. The first version of the Picture System 2 allowed the user to build a SDF in the PDP-11 memory that did not contain a hierarchy (the instances were expanded in the SDF). The second version, which is the version discussed in this thesis, supports a SDF with object calls. The latest version in the series, the Multi-Picture System supports more sophisticated interaction handling in the Picture Generator, a hierarchical description in the device dependent instance representation (called the LDF in the functional model), but does not support a device independent hierarchical representation of the object (i.e., SDF). Such a representation will, however, be added in later versions of the system [WATK79]. In the following sections, the VG3400 will be discussed and then the functional capabilities of the VG3400 will be contrasted with those of the E&S PS2.

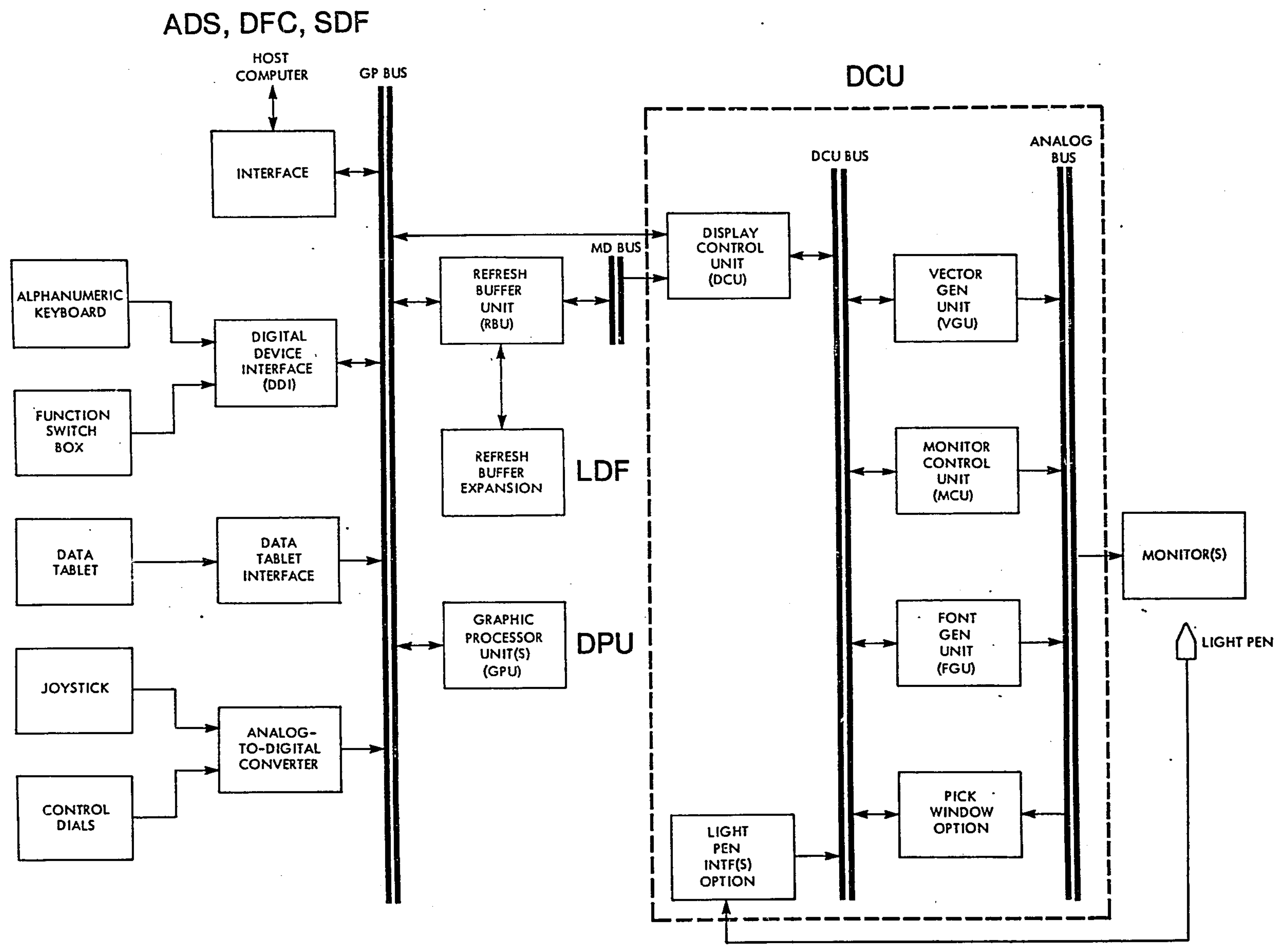


FIGURE 3-8
VG3400 Structural Model
(Vector General, Inc., VG3400 Programming Concepts Manual)

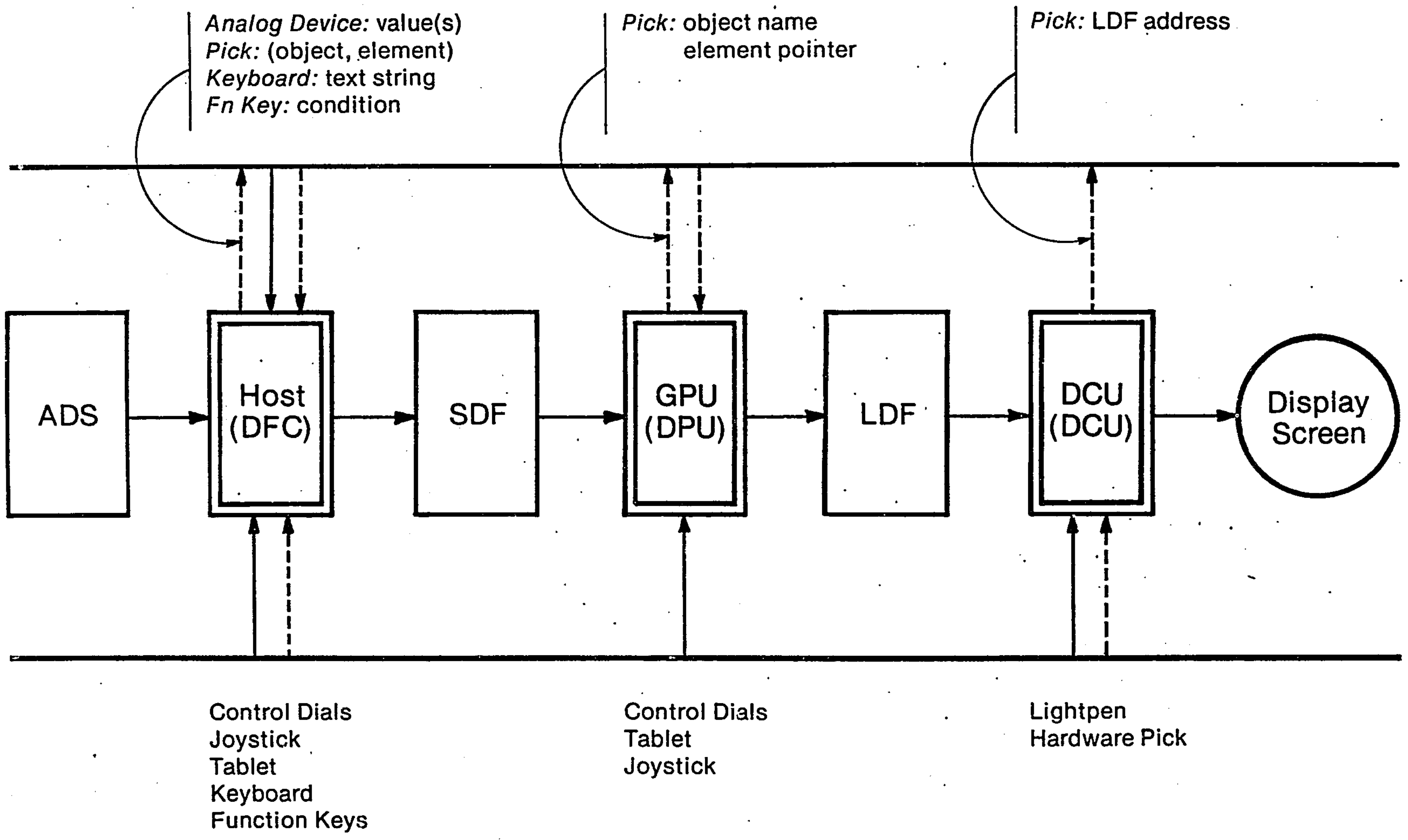


FIGURE 3-9
Functional Model: VG3400

3.4.2 VECTOR GENERAL 3400

The VG3400 [VECT78a, VECT78b] consists of seven components:

- the Host Computer Interface
- the GP Bus
- the Graphics Processor
- the Refresh Buffer
- the MD Bus
- the Display Control Unit
- the Peripheral Devices.

These components are connected to a host computer which, as far as this thesis is concerned, is part of the total display system.

The Host Computer[5] is a general purpose computer, and is, as for the E&S PS2, a mini-computer such as a DEC PDP-11. The Host Computer contains the application program, the ADS, the DFC, and, in contrast with the E&S PS2, also the SDF. The VG supplied software includes a set of FORTRAN-callable subroutines that are used by the DFC to build the SDF from the object

[5]This proper noun is the term [VECT78b] uses to refer to the host attached to the VG3400; it is used with the same meaning here.

representation in the ADS and to modify the SDF according to the input data.

The SDF contains primitives, attributes, object calls, and modelling and viewing transformations. The SDF may also contain general purpose instructions, such as arithmetic, boolean, and control instructions. The reader is referred to Appendix A for a complete list of SDF features.

The Host Computer interfaces with the VG3400 through the Host Computer Interface. The interface unit handles communication between the Host Computer and the VG3400 over three I/O paths:

- the programmed I/O path (PIO),
- the direct memory access path (DMA), and
- the interrupt path.

The PIO and DMA paths transmit single words of data between the Host and the VG3400. The interrupt path transmits interrupts from the input devices and from other devices such as the Display Control Unit.

The interface unit, the Graphics Processor, the Refresh Buffer, and the input devices (except the lightpen and the hardware pick) are connected to an asynchronous bus, the GP Bus.

The Graphics Processor is the DPU. It compiles a SDF in the Host Computer memory to a LDF that is stored in the Refresh Buffer. The Graphics Processor is a general purpose processor; in addition to traversing a hierarchy, transforming and clipping primitives, it also executes general purpose instructions.

The LDF resides in the Refresh Buffer and contains primitives and mode settings. The LDF may not be segmented, but the VG3400 provides an <u>edit-aid</u> feature. This feature allows a user to make tentative deletions, insertions, and modifications to the LDF. The edit-aid feature can, for example, be used as a buffer where a part of the object is modified while the remainder of the object is unchanged.

The Display Control Unit together with the VGU, MCU, and the FGU function as the DCU. The Display Control Unit uses the MD Bus to access the LDF. The VGU converts the data in the LDF to analog signals that position the electron beam and control the intensity in the display monitor. The FGU can interpret the full ASCII character set. The Font Generator contains a transformation matrix which is loaded with the current transformation matrix. The characters are thus subject to the viewing/modelling transformation with the exception that perspective foreshortening is not applied to individual characters in a text string.

The VG3400 supports seven input devices:

- Data Tablet
- Control Dials
- Function Keys
- Alphanumeric Keyboard
- Lightpen
- Joystick
- Hardware Pick

All the devices are interfaced directly to the GP Bus, except the lightpen and the hardware pick that are interfaced to the Display Control Unit.

The input devices connected to the GP Bus can be accessed by the DFC in the Host. Some input data is passed to the application program, and other input data is used as input to the DFC.

The sampling devices, i.e., the control dials, the joystick, and data tablet can be accessed directly by the Graphics Processor. Individual coordinates and transformation data can be updated according to the value of the analog device each time the SDF is compiled.

Only the lightpen and the hardware pick are attached directly to the Display Control Unit. Both devices can modify the output to the LDF in that both may highlight a "picked" item. The data of the lightpen and the hardware pick can also be forwarded to a higher level processor. The item in the ADS corresponding to the item picked on the display screen is found through a two-step process. First the word in the LDF is used to find the corresponding word in the SDF. The object call stack at the time of execution of this word is passed to the DFC, which in turn establishes the desired (object,element) pair.

The similarities and the differences between the two systems are discussed in the next section.

3.4.3 A FUNCTIONAL COMPARISON OF THE E&S PS2 AND THE VG3400

The E&S PS2 and the VG3400 are similar in that both systems implement the SDF and the LDF as actual data structures. However, the object representations and the processors vary a great deal in complexity. The E&S PS2 Picture Processor (which functions as the DPU) is a special purpose processor, whereas the VG3400 Graphics Processor Unit (which plays the same role) is completely general purpose. As a result of the differences in object representations, the input processing also differs a great deal between the two systems. Some examples of these differences between the two systems are discussed briefly.

I The SDF

a) Addressing Modes -- E&S PS2 allows only immediate data; the VG3400 allows both immediate data and referenced data. The referenced data may access any location in the Host, the general purpose registers in the Graphics Processor, and the analog input device registers. If the SDF in the VG3400 contains references to locations in the Host Computer, these locations are considered part of both the SDF and the ADS by the functional model. The application program can modify the data in the ADS and thereby the contents of the SDF get modified as well. If some of the immediate data in the E&S PS2 is to be modified, (part of) the ADS contains pointers to these data items. The actual coordinate data is located in the SDF and another copy may be located in the ADS.

b) Object Calls -- in the VG3400 the objects are referenced through a directory; in the E&S PS2 object references are direct pointers. The E&S PS2 stack cannot contain the names of the calling objects; as a result, the name of a picked object must be determined from a correlation table or by placing each instance of an object in a separate segment in the LDF.

c) General Purpose Instructions -- The VG3400 allows arithmetic, boolean, and control instructions; the

E&S PS2 does not allow any general purpose instructions.

d) Transformations -- In E&S PS2 all transformations are represented as general 4x4 homogeneous matrices; the VG3400 the rotation, scaling, and translation parameters are stored individually and a transformation matrix is composed each time the SDF is compiled. The VG3400 does not support shearing transformations and allows only uniform scaling.

e) Text -- In the VG3400 the viewing/modelling transformations apply to text, with the restriction that perspective transformations do not apply to individual characters. In the E&S PS2 only the starting point of a text string is subject to transformation.

f) Interaction -- On the VG3400 the compilation of the SDF may be modified by input from the analog devices by the use of referenced data. The E&S PS2 does not allow any input from analog devices into the DPU; all input is done to the DFC. Any modifications made to the SDF in the E&S PS2 on the basis of input from analog devices are the results of modifications by the DFC using pointers, stored in the ADS, to locations in the SDF.

II The LDF

Segmentation -- In the E&S PS2 the LDF can be segmented so that pieces of an object may be modified without the DPU regenerating the entire LDF. This is only possible on the VG3400 in a very limited fashion through the edit-aid feature.

In the next chapter, the functional model is used to develop a set of performance modelling techniques for graphics processors.

4 PERFORMANCE MODELLING TECHNIQUES

There are three common techniques [BEIZ78, SVOB76, FERR78] used in evaluation of computer system performance:

- measurement,
- simulation, and
- analysis.

Measurements are obtained by hardware or software monitors that measure the performance of a system running an application. Correct measurements are often difficult to obtain, in particular for software monitors, because the monitor tends to distort the measured performance. In addition, in order to measure the performance of a system, the actual system must be available. This makes performance comparisons of several systems very impractical and eliminates measurements as a method for making trade-off evaluations in the design stage.

Simulation is a technique by which the behavior of a system is reproduced. The performance of the simulated system is determined by establishing the resource requirements for the system under a particular load. Simulation does not require access to the actual system and could be used both to compare existing systems and to determine the impact of different design decisions. Simulation, however, can be quite costly in both computer time and programming effort. Changes to the simulated system may require changes to the simulator, changes which involve re-programming and debugging. Simulation methods are hence not as flexible as is desirable to investigate various trade-offs during the design of a system.

Analytical modelling uses only mathematical techniques. These techniques are, for several reasons, well-suited for graphics system modelling. First, graphics systems are simple enough that the computational effort in the analysis is manageable. Second, analytical techniques have the flexibility required for a performance model to evaluate trade-offs in the design process.

The objective of performance modelling in this thesis is to establish estimates for the time it takes to map an object from one representation in the pipeline to the next. At this level of abstraction, the behavior of a graphics system is predominantly deterministic. User input need not be considered as a source of non-determinism. Although the user may make input requests at irregular intervals, it can be assumed that he only makes zero or one request during the time intervals under consideration. These two cases can be handled separately. Because of the deterministic behavior, analytical modelling techniques are particularly attractive for graphics systems; performance models can be based only on elementary algebra, probability theory, and simple aspects of queueing theory.

The phrase "performance model" in this thesis refers to a mathematical model that treats the characteristics both of an application and of a graphics system as independent variables and the performance measures as dependent variables. The performance modelling technique consists of three steps:

1) The characterization of an application or class of applications, i.e., the characterization of the load.

2) The definition of a set of hardware parameters that characterize the operation of a graphics system.

3) The derivation of mathematical relationships in terms of the load and the hardware parameters; these relationships characterize the system behavior, and are the performance measures.

The application of these mathematical relationships to a particular system results in a performance model for that system.

A load in a graphics system is represented by an instruction mix. An instruction in a mix is referred to as a macro instruction and represents an abstract unit of work. It is preferrable to describe a load in terms of macro instructions rather than user instructions for two reasons. First, macro instructions are at the level of abstraction in which it is natural to express an application's processing. Second, macro instructions are graphics system independent, so they are more suitable for system comparisons than user instructions.

The processing time for a load is determined by the processing times for the macro instructions in that load. The processing time of a macro instruction is determined by mapping that macro instruction onto the system under consideration, i.e., by implementing it in terms of user instructions. The processing time for a macro instruction is then a function of the sequential processing in each processor of the pipeline, the I/O requests by each processor, and the contention at the I/O requests that is due to concurrent execution in the pipeline.

In this chapter, the techniques needed for deriving the processing time for a load are presented. The first section below introduces the concept of macro instructions. In the second section, the functional model that was presented in Chapter 3 is broken down into components, each of which is simple enough to be modelled analytically. The revised functional model is used to identify the hardware parameters that affect the performance measures. In the last section, methods for deriving the performance measures are discussed. These measures are functions that characterize the concurrent activities in a graphics system, as well as the contention for the resources which occur as a result of concurrent processing. Chapter 5 presents some methods for applying the performance modelling techniques.

4.1 Macro Instructions

A load on a graphics system is characterized in terms of a set of macro instructions. A macro instruction, in this thesis, is an abstract unit of work. This unit of work corresponds to some processing in one or more of the stages in the processing pipeline. The units of work are called macro instructions to distinguish them from user instructions.

> Definition 1: A <u>user instruction</u> is a unit of processing as defined by the graphics system hardware/firmware/ software.

In the first stage in the pipeline, a user instruction is an assembler instruction or a function that is part of a graphics system access method. In the second stage, a user instruction is one instruction in the SDF, and it is generally interpreted by firmware and hardware. In the third stage of the pipeline, a user instruction is one instruction in the LDF and it is also generally interpreted by firmware and hardware.

A macro instruction corresponds to operations in each stage of the processing pipeline. These operations are on different levels of abstraction than the user instructions. Some macro instructions correspond to an operation that is on a higher level of abstraction, i.e., a set of user instructions. Other macro instructions correspond directly to one user instruction in a data structure. It is also possible for a macro instruction to be at a lower level of abstraction than a user instruction, i.e., to correspond to only one of several processing paths of a user instruction. A macro instruction may also correspond to any combination of these three cases.

More formally, we can now state the following definition:

Definition 2: A macro instruction is a unit of work that is performed by one or more execution paths through a set of zero or more user instructions in each stage of the processing pipeline. These execution paths are called the implementation of the macro instruction.

Before we state some properties of the macro instructions, we have one more definition:

Definition 3: Given a set M of macro instructions describing a system, a load on that system is a mapping from M to the non-negative integers.

If L is a load on the graphics system described by M, and $m \epsilon M$, L(m) denotes the number of occurances of the macro instruction m in the work being performed by the graphics system in support of an application program over some given interval of time. An application program is thus described by a sequence of loads, but for practicality, we will deal only with a few loads representing average behavior or worst case behavior.

Macro instructions have the following properties:

1) Every occurance of a macro instruction in a load is independent of any other occurance of a macro instruction in the same load. That is, the processing of one macro instruction is independent of the processing of every other macro instruction.

2) The unit of work performed by a macro instruction does not change with load. The behavior of the system may change, the resource contention will probably change, but the processing of the individual macro instruction does not change.

3) The implementation of a macro instruction has only one sequence of I/O requests. This means that any parallel execution paths involve the same set of I/O requests.

The first two properties imply that the processing time of an entire load is the sum of the processing times of the occurances of the individual macro instructions. (The significance of the third property will become clear later.) The processing times depend on the hardware parameters of the resources in the graphics system. This dependency is discussed in the next section.

As was discussed above, a macro instruction describes one or more execution paths through a set of user instructions. It is easy to see that there is potentially a very large number of such paths and hence a very large number of macro instructions. In order to make it easier to specify the macro instructions, they are defined via macro instruction functions (for brevity, macro functions) that take a set of processing descriptors as arguments. The value of a macro function for specific values of its arguments is a macro instruction.

> Definition 4: A <u>processing descriptor</u> is an argument of a macro function whose values determine the possible alternatives of some aspect of processing at one or more stages in the pipeline.

Each processing descriptor can take a set of values. A value can select one in a set of alternate execution paths, or it can describe the processing in other ways, such as the number of iterations, or some other processing activity. A processing descriptor takes only discrete values.

More formally we now have the following definition:

Definition 5: A <u>macro function</u> is a function whose domain is the cross-product of the processing descriptor value sets and whose range is a set of macro instructions.

The full advantage of the macro functions as a "shorthand notation" for the macro instructions will become evident in later chapters.

<u>4.2 Hardware Parameters of a Performance Model</u>

In order to identify the hardware parameters of the performance model, the functional model is broken down into three stages:

- the SDF compilation component,
- the LDF compilation component, and
- the image generation component.

This is illustrated in Figure 4-1.

There is a slight notational change in this figure compared with the figures in Chapter 3. The interfaces between the data structures and the processors are illustrated as resources. This is to emphasize that the interface is a resource whose behavior characteristics are parameters of the performance model.

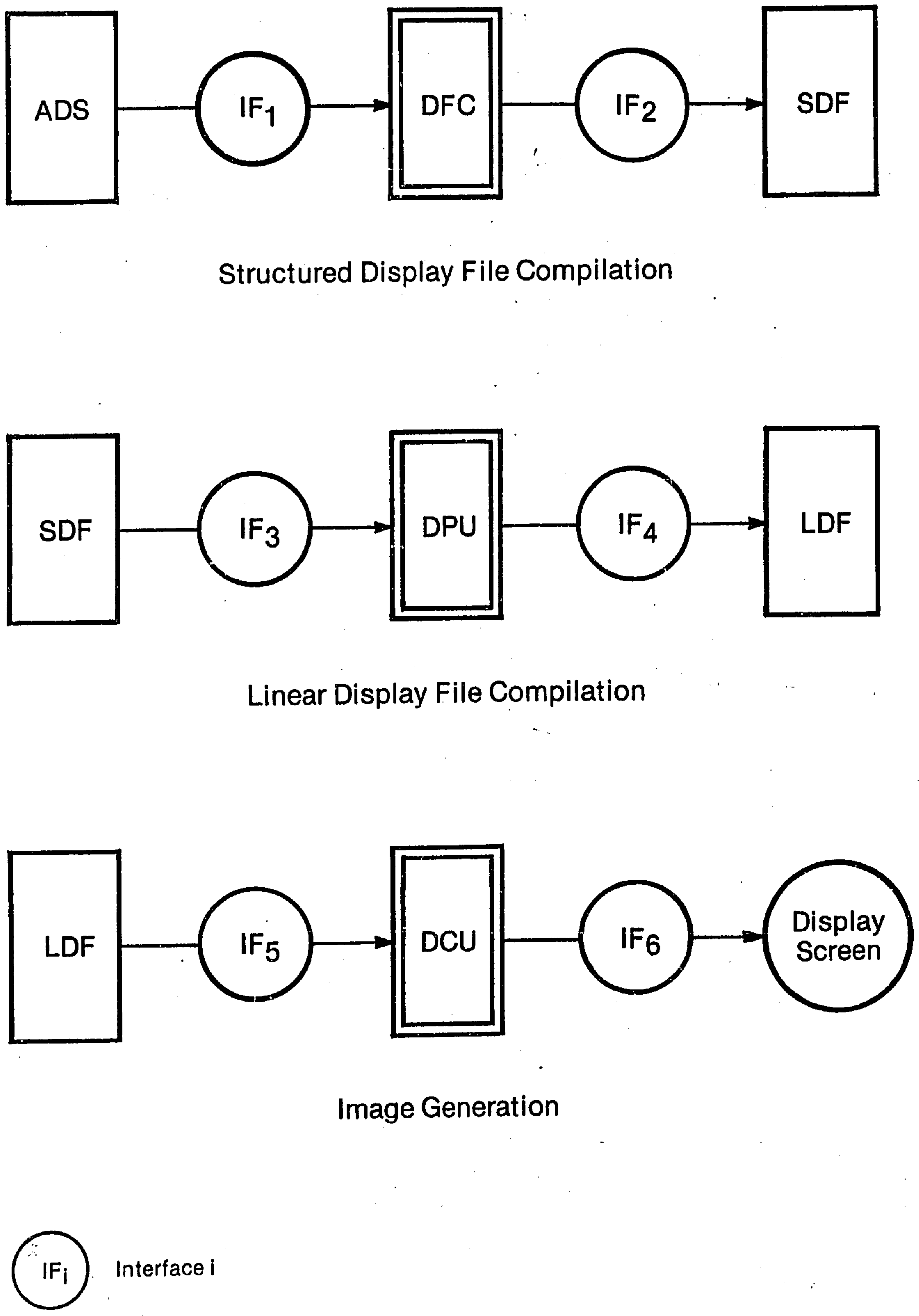


FIGURE 4-1
Three Stages of Picture Processing Pipeline

In this thesis, a performance model is a set of three formulas that express the processing times U_1, U_2, and U_3 for the three stages in the pipeline in terms of load and hardware parameters. U_1 is the total time required to map the ADS to the SDF, U_2 is the time to map the SDF into the LDF, and the time to generate the image on the display screen is U_3. Note that U_1 and U_2 are the times required to create a new or update an existing image and U_3 is the time required to refresh the image on the display screen.

The processing times for each stage in the pipeline can be expressed as sums of the processing times for each macro instruction in the processing stage, where the processing times for the macro instructions are functions of the hardware parameters. For each stage the processing time is an expression of the form:

$$U = \sum_j n_j Q_j(S_j, b_1, b_2, m_r, m_w).$$

In this equation,

n_j denotes the number of occurances of the j:th macro instruction,

Q_j denotes the total processing time in one stage of the pipeline generated by the j:th instruction in the macro instruction set (Q_j is zero if the j:th macro instruction involves no work in this stage of the pipeline),

S_j denotes a set of individual processing components in one stage in the pipeline generated by the j:th macro instruction. A processing component is a code sequence that is not divided by any requests for concurrent activity. The time for an individual processing component is denoted s_{jk} ,

b_1, b_2 denote the bus transfer rate of the bus over which data is read or written, respectively,

m_r , m_w denote memory read time and memory write time, respectively. (Note that the display monitor will be considered a "memory" into which data is "written," and the write time refers to line or text drawing rates for the monitor.)

Several things should be noted regarding the functions Q_j :

- All resource parameters, e.g., the bus transfer rates and memory transfer rates, are the rates in absence of any contention for the resource.

- If any of the intermediate representations only exist as data in time, the corresponding I/O parameters can sometimes be ignored. If, however, there is a buffer between the two processing stages, the buffer is treated in a fashion similar to a data structure.

- The only part of interaction handling that is taken into consideration is the input which directly affects output processing. The collecting of input data which is to be passed on to another processor may be ignored. This omission will not significantly affect the model, since, in general, at most one input device need be serviced in each processing stage during the time intervals of interest, and the time required to service one device is negligible. To be more precise, a term could be added to the processing time in each stage to include the interaction handling:

  $$U' = U + O$$

  where O is the overhead generated by any interaction devices in one update or refresh cycle for that stage of the pipeline.

The Q_j's are functions of the hardware parameters and characterize the concurrent processing in a graphics system as well as the contention for the resources that occurs as a result of the concurrent processing. In the next section, algebraic expressions for the Q_j's are derived.

4.3 Performance Measures

This section derives analytical expressions for the performance measures, which are the dependent variables of the performance model. In the first subsection, activity flows are introduced. These simplified flowcharts are used to derive expressions for the macro instruction processing times as functions of the hardware parameters and of the processing overlap in each stage of the pipeline. Due to the simultaneous execution of the three stages, there is contention for some of the resources. This contention results in a delay in each processing stage. In the second subsection, the different types of delay are incorporated in the performance measures.

4.3.1 ACTIVITY FLOWS

An *activity flow* [BEIZ78] is a flowchart representing one or more execution paths through a program. It generally does not correspond directly to a processing flowchart for the program, but corresponds to a trace through a processing flowchart for a particular (set of) parameters of the program. An activity flow shows computation and any relevant concurrent activities which can either be I/O or processing by a parallel processor. Thus, an activity flow contains the essential information for calculating performance data related to a particular macro instruction. An activity flow has a format as illustrated in Figure 4-2. (An algorithm for deriving an activity flow from a processing flowchart is described in Chapter 5.)

In order to derive the processing time in each stage for a macro instruction, the implementation in each stage of that macro instruction is represented by an activity flow. If the execution time for the k:th processing component of the j:th macro instruction is denoted s_{jk} and if the transfer time or execution time for each concurrent activity (I/O or processing) is r_{jk}, then the processing time, Q_j, for the j:th macro instruction is the sum of all processing components that have no concurrent activity, plus the sum of the maximum times for the alternate paths through each region of the activity flow where there is some concurrent activity. Q_j can be expressed as:

$$Q_j = \sum_{k \in K} s_{jk} + \sum_{M \in \Pi} \max_{L_{kM}} \Big(\sum_{l \in L_{1M}} s_{jl} + \sum_{l \in L'_{1M}} r_{jl} , \dots , \sum_{l \in L_{nM}} s_{jl} + \sum_{l \in L'_{nM}} r_{jl} \Big).$$

where K is the set of all integers k for which the k:th processing component has no concurrent activity, Π is the set of sequences of code where there is parallelism, M is any such code sequence, $L_{kM} \cup L'_{kM}$ is a unique sequence of integers denoting a path through one code sequence where there is concurrent activity, and $L_{1M} \cup L'_{1M} \dots \cup L_{nM} \cup L'_{nM}$ denotes all such paths through the M:th code sequence.

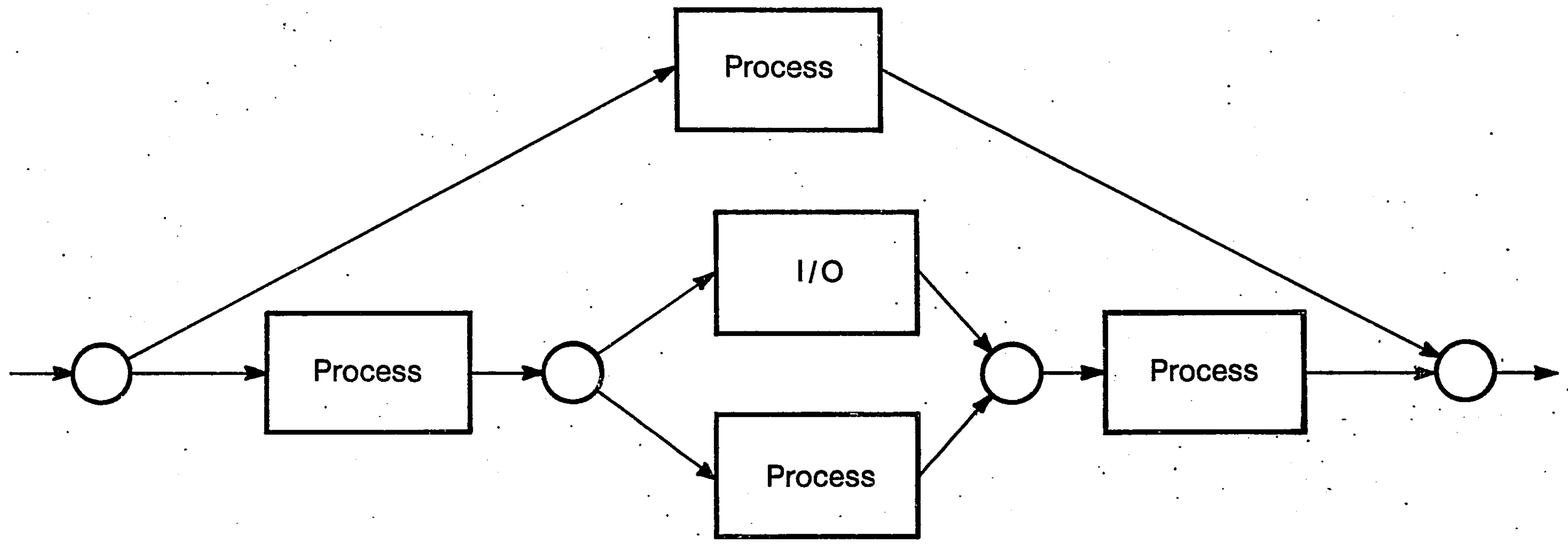


FIGURE 4-2
Activity Flow

For each request of a resource there is an associated delay due to simultaneous requests by the other stages in the pipeline. Thus, the processing time for a macro instruction includes a delay at each I/O request. Methods for deriving these delays are discussed in the next section.

4.3.2 RESOURCE CONTENTION

Contention for resources in a graphics system is the result of the simultaneous execution of the three stages in the pipeline; this causes simultaneous requests for the resources shared by the three stages. In this thesis two different measures of delays caused by this contention will be used -- stretch and lag[6]. Stretch is a measure of the increase in execution time over some time interval for a process demanding access to a resource that it shares with another process. Lag is a measure of delay for a single resource access.

If two processes share a resource, one of the processes must have higher priority than the other, so that concurrent requests can be resolved. The priority may manifest itself in three ways:

[6]Stretch and lag are terms that have been used to denote a measure of the increase in effective execution time due to multiprogramming and a measure of operating system dispatch delay, respectively [MICH77]. The worst case stretch which is discussed below has also been used by [BEIZ78] as a measure of delay due to resource contention.

the high priority process may preempt a resource request by the lower priority process instantaneously, it may preempt the lower priority process after some portion of a resource request is completed (e.g., the current memory access in the middle of a multi-access request), or it may not preempt the lower priority process at all, in which case priority only matters when resource requests by both processors occur at the same instant. Instantaneously preemptive priority, the first kind, does not occur in any situations studied in this thesis; the second and third kinds of priority, <u>preemptive</u> and <u>non-preemptive</u>, are both discussed, however.

Stretch and lag are used as measures of delay for both high and low priority processes. For a low priority process, the measures of delay are independent of whether priority is preemptive or non-preemptive. For a high priority process, however, the measures differ in the preemptive and non-preemptive cases. In general, the delay for the high priority process is a function of the service time of the low priority process's requests. With non-preemptive priority, this service time is defined to be the time to complete an entire request. On the other hand, with preemptive priority, service time for the low priority process is regarded as the maximum time it takes that process to relinquish control of the resource.

Stretch and lag are discussed in the first two sections below, respectively. As will be seen, the delays caused by large resource requests are not always modelled adequately by stretch or lag. This is discussed in the third section. Stretch and lag also cannot be applied to processes that are capable of buffering resource requests. The effect of buffering on delay is discussed in the last section.

The following variables will be used throughout the remainder of this chapter:

T is the length of the observation interval. The time units for T are also used to express all other time dependent variables.

a_i is the average time between requests to resource R by process P_i in absence of any contention for R.

c_i is the average service time for a request to resource R by process P_i in absence of any contention for R. A <u>cycle</u> is a processing interval followed by a resource request interval; the total time for a cycle in absence of contention is $a_i + c_i$.

d_i is the average delay for one access to R by process P_i.

n_i is the number of requests by processor P_i to resource R during time T.

r is the rate of resource R. The variable c_i is the average service time for a request by P_i and corresponds to some number of unit requests, each of length 1/r. Preemption of a low priority process's request to R cannot take place until R has serviced the low priority process an integer multiple of intervals 1/r.

An additional set of variables can now be defined in terms of T, a_i, c_i, d_i, n_i, and r.

b_i is the fraction of time T resource R is busy due to requests from process P_i in absence of any contention for R. The quantity b_i can be expressed as $b_i = c_i /(a_i + c_i)$.

α_i is the rate at which process P_i completes processing between resource requests. The quantity α_i can be expressed as $\alpha_i = 1/a_i$.

β_i is the completion rate for requests by process P_i to resource R in absence of any contention for R. The quantity β_i can be expressed as $\beta_i = 1/c_i$ or $\beta_i = n_i /T \cdot b_i$.

γ_i is the access rate by process P_i to resource R in absence of any contention for R. The quantity γ_i can be expressed as $\gamma_i = 1/(a_i + c_i)$.

R_i is the relative demand for resource R by P_i. The quantity R_i can be expressed as $R_i = \gamma_i /r$.

ρ_i is the stretch factor for processor P_i, i.e., the ratio between the effective time for one cycle and the time for one cycle in absence of contention. The quantity ρ_i can be expressed as $\rho_i = (a_i + c_i + d_i)/(a_i + c_i)$.

ϕ_i is the lag for one request by processor P_i to resource R, i.e., the difference between the effective time for one cycle and the time for one cycle in absence of contention. The quantity ϕ_i is simply equal to d_i.

4.3.2.1 STRETCH

This section derives analytical expressions for maximum and minimum stretch for a process and also an expected value for stretch. In this section, stretch is derived assuming that the contending process does not suffer any delay, i.e., that the stretch for the contending process is one[7]; in Chapter 5 an algorithm is presented for deriving stretch when the contending process does experience delay.

In the following theorem it is assumed either that P_1 is the lower priority process or that P_1 is higher priority but does not preempt P_2.

[7]This is equivalent to instantaneous preemption by the contending process.

<u>Theorem 4.1</u> If P_1 and P_2 are two processes that make demands on a resource R and during an observation interval T, P_2 keeps the resource busy during $b_2 \cdot T$ of the interval ($b_2<1$), then the maximum stretch of the execution time for P_1 is given by $\rho_1 = 1/(1-b_2)$.

Proof: If P_1 demands R at rate γ_1, then the mean time between accesses is $1/\gamma_1$ in absence of any contention. If x is the number of completed accesses by P_1 during T when P_2 is contending for R, and if D is the total delay during the observation interval, then the mean time between those accesses is

$$\frac{T}{x} = \frac{1}{\gamma_1} + d_1, \text{ and}$$

$$T = \frac{x}{\gamma_1} + x \cdot d_1.$$

Hence,

$$T = \frac{x}{\gamma_1} + D. \qquad (4.1\text{-}1)$$

Assuming worst case conditions, i.e., there is a conflict at each access, the total delay is $b_2 \cdot T$ which implies

$$T = \frac{x}{\gamma_1} + b_2 \cdot T \text{ and}$$

$$T = \frac{x}{\gamma_1(1-b_2)} .$$

The mean time between accesses in the presence of contention is

$$\frac{T}{x} = \frac{1}{\gamma_1(1-b_2)} .$$

Thus the stretch is

$$\rho_1 = \frac{\frac{T}{x}}{\frac{1}{\gamma_1}} = \frac{1}{1-b_2}$$ which proves the theorem.

In the following theorem, as in the previous one, it is assumed either that P_1 is the lower priority process or that P_1 has higher priority but does not preempt P_2.

<u>Theorem 4.2</u> If P_1 and P_2 are two processes that make demands on a resource and during an observation interval T, P_2 keeps the resource busy during $b_2 \cdot T$ of the interval ($b_2<1$), then the minimum stretch of the execution time for P_1 is given by $\rho_1 = \max(1, b_1/(1-b_2))$.

Proof: If β_1 is the completion rate for requests by processor P_1 to R in absence of contention, then the number of requests n_1 by P_1 during T is

$$n_1 = b_1 \cdot T \cdot \beta_1 \; .$$

Assuming best case conditions, the number of completed accesses x by P_1 during T when P_2 is contending for R is the smaller of the number of accesses in absence of contention and the number of accesses that can be made in the time remaining after P_2 has been serviced by R:

$$x = \min(b_1 \cdot T \cdot \beta_1, \; (1-b_2) \cdot T \cdot \beta_1) \; .$$

Thus the stretch is

$$\rho_1 = \frac{\frac{T}{x}}{\frac{T}{n_1}} = \frac{n_1}{x} = \frac{b_1}{\min(b_1, 1-b_2)} \text{, and}$$

$$\rho_1 = \max(1, \frac{b_1}{1-b_2}) \quad \text{which proves the theorem.}$$

These theorems easily generalize to multiple processes. Assume there are n processes and that P_1 is the process in question. If each process uses the resource for time $b_i \cdot T$ during T, then the maximum stretch for P_1 is $\rho_1 = 1/(1-\sum_{i>1} b_i)$ and the minimum stretch is $\rho_1 = \max(1, b_1/(1-\sum_{i>1} b_i))$ provided that the total utilization by the other processes is less than one, that is, $\sum_{i>1} b_i < 1$. If this utilization exceeds one, the stretch for the lower priority process is infinite.

<u>Theorem 4.3</u> Assume that P_1 and P_2 are two processes that make demands on a resource, and during an observation interval T, P_1, the process with the lower priority, keeps the resource busy during $b_1 \cdot T$ of the interval ($b_1 < 1$), and assume that the relative demand for the resource during T by the higher priority process, P_2, is R_2. If P_2 can preempt P_1 after an access to the resource that is no longer than $1/r$, then the maximum stretch for P_2 is given by $\rho_2 = \min(1/(1-b_1), 1+R_2)$.

Proof: Using equation (4.1-1) above, we get the following expression for T:

$$T = \frac{x}{\gamma_2} + D.$$

Assuming worst case conditions, we see that D is the smaller of P_1's utilization of the resource during T and a delay of 1/r for each request. Hence,

$$T = \frac{x}{\gamma_1} + \min\left(b_1 \cdot T, \frac{T}{\frac{1}{R_2}+1}\right).$$

The mean time between accesses is

$$\frac{T}{x} = \frac{1}{\gamma_1 \cdot (1-\min(b_1, \frac{R_2}{R_2+1}))} .$$

Thus the stretch is

$$\rho_2 = \frac{\frac{T}{x}}{\frac{1}{\gamma_1}} = \min(\frac{1}{1-b_1}, 1+R_2)$$ which proves the theorem.

The minimum stretch for a high priority process is clearly one.

It should be noted that in order to calculate the minimum and maximium stretch, no assumptions need be made regarding the distributions of either arrival or service rates for the requests to the resource. In order to derive an expected value of the stretch, however, such assumptions are obviously necessary.

Two processes P_1 and P_2 sharing a resource R with P_2 always gaining access to R without delay can be modelled as a simple Markov process. The model has four states as is illustrated in the state transition diagram in Figure 4-3:

$$\begin{tikzcd}
(p_2, p_1) \arrow[r, bend left, "\alpha_2"] \arrow[d, bend left, "\alpha_1"] & (r_2, p_1) \arrow[l, bend left, "\beta_2"] \arrow[d, "\alpha_1"] \\
(p_2, r_1) \arrow[u, bend left, "\beta_1"] \arrow[r, bend left, "\alpha_2"] & (r_2, r_1) \arrow[l, bend left, "\beta_2"]
\end{tikzcd}$$

FIGURE 4-3
State Transition Diagram

(p_2,p_1) both processes are processing,

(p_2,r_1) P_2 processing, P_1 accessing R,

(r_2,p_1) P_1 processing, P_2 accessing R,

(r_2,r_1) P_2 accessing R, P_1 waiting to access R.

In Figure 4-3 the state transitions are illustrated with the associated rates of transition.

For each state the rate at which the system leaves the state equals the rate at which the system enters the state. This gives the following equations (P(*,*) is the probability of being in state (*,*)):

$$P(p_2,p_1) \cdot (\alpha_2+\alpha_1) = P(p_2,r_1) \cdot \beta_1 + P(r_2,p_1) \cdot \beta_2$$

$$P(r_2,p_1) \cdot (\alpha_1+\beta_2) = P(p_2,p_1) \cdot \alpha_2$$

$$P(r_2,r_1) \cdot \beta_2 = P(p_2,r_1) \cdot \alpha_2 + P(r_2,p_1) \cdot \alpha_1$$

The sum of the probabilities is one:

$$P(p_2,p_1)+P(r_2,p_1)+P(p_2,r_1)+P(r_2,r_1) = 1$$

The expected completion rate of resource accesses by P_1 with P_2 contending for R is $\beta_1 \cdot P(p_2,r_1)$. Hence, the mean time between completion of resource accesses is $1/(\beta_1 \cdot P(p_2,r_1))$. The mean time between resource accesses by P_1 in absence of any contention is

$1/\alpha_1+1/\beta_1$ and the stretch is

$$\rho_2 = \frac{\dfrac{1}{\beta_1 \cdot P(p_2,r_1)}}{\dfrac{1}{\alpha_1}+\dfrac{1}{\beta_1}} .$$

Solving the simultaneous linear equations above for $P(p_2,r_1)$ gives the following theorem:

<u>Theorem 4.4</u> If P_1 and P_2 are two processes that make demands on a resource and P_2 can always gain access to the resource without delay, and if the processing times between resource requests for P_1 and P_2 and the service times of requests to R by P_1 and P_2 are independently exponentially distributed[8], then the expected stretch of process P_1's execution time in the presence of concurrent demands for R by process P_2 is

$$\rho_1 = \frac{1}{1-b_2} - \frac{\alpha_2}{(\alpha_1+\alpha_2+\beta_2)} \cdot \frac{(1-b_1)}{(1-b_2)} .$$

[8]This can be assumed valid if the ratio of the standard deviation to the mean is in the range (1/10, 2). If the ratio is less than one, the assumption leads to a somewhat pessimistic result; if the ratio is greater than one, it is somewhat optimistic [BUZE79a].

In practice, as will be seen later, there is often no need to calculate the expected value for stretch since the difference between the maximum and minimum stretch tends to be small.

4.3.2.2 LAG

This section derives analytical expressions for lag in one process for two cases, namely, where the requests by the contending process have constant or exponential service times. The lag is derived assuming that the contending process does not suffer any delay; in Chapter 5 an algorithm is presented for deriving lag when the contending process does experience delay.

Lag is used as a measure of the delay for a single resource request by a process caused by the fact that the contending process cannot relinquish the resource until the current request is serviced. Lag is used for both high and low priority processes, but, in the case that the higher priority process can preempt the other process, the service time of the lower priority process is the <u>unit</u> service time, i.e., the minimum time that the lower priority process is serviced before it is preempted.

<u>Theorem 4.5</u> If P_1 and P_2 are two processes that make demands on a resource R and during an observation interval T, P_1 keeps the resource busy during $b_1 \cdot T$ of the interval ($b_1<1$), and if the service time for the requests to R

by P_1 are a constant c_1, and the times between requests to R by P_2 are exponentially distributed and independent from the requests by P_1, then the expected lag for one request by P_2 is $\phi_2 = c_1/2 \cdot \min(1, b_1/(1-b_2))$.

Proof: The probability of P_2 being delayed is the ratio of the time that R is busy serving P_1 to the time that R is _not_ busy serving P_2. (P_2 may _request_ R only when not already _being served_ by R.) This ratio is $b_1/(1-b_2)$. The expected duration of the delay is $c_1/2$, which proves the theorem.

Theorem 4.6 If P_1 and P_2 are two processes that make demands on a resource R and during an observation interval T, P_1 keeps the resource busy during $b_1 \cdot T$ of the interval ($b_1<1$), and if the service times for the requests to R by P_1 are exponentially distributed with an average service time of c_1, and the times between requests to R by P_2 are exponentially distributed and independent from the requests by P_1, then the expected lag for one request by P_2 is $\phi_2 = c_1 \cdot \min(1, b_1/(1-b_2))$.

Proof: The expected duration of the delay is in this case c_1, which proves the theorem.

The next chapter provides some simple algorithms for how to derive the delay for cases where the contending process experiences delay.

4.3.2.3 LARGE RESOURCE REQUESTS

As was noted in section 4.3.2.1, the minimum and maximum stretch can be computed without regard for service time distributions. The expected value for stretch, however, is based on the assumption that the service times are exponentially distributed. If this assumption is inappropriate, a more careful examination of the effect of a resource request may be necessary. This can be done by considering the resource request as a separate process running concurrently with the processes in the three stages locking out some other process(es). The calculations of delay can be done when the combined delays for all three stages are calculated, as is described in the next chapter.

4.3.2.4 BUFFERED DEVICES

Two processes that share a resource are often capable of buffering their requests to the resource. If the total resource demand is less than the maximum resource rate, and if the buffers are sufficiently large, there is no noticable delay for either process. In order to handle startup conditions pessimistically, a delay of the maximum service time for the high priority process can be added to the total processing time for the lower priority process.

5 APPLICATION OF THE PERFORMANCE MODELLING TECHNIQUES

The previous chapter discussed performance modelling techniques suitable for graphics systems, and this chapter discusses how to derive a performance model using these techniques. This chapter also introduces a new construct, a composite macro instruction, which is a weighted average of a set of macro instructions. Composite macro instructions will be used in the remainder of the thesis to discuss implementation trade-offs.

The first section of the chapter discusses some criteria for how to derive a macro instruction set. In the second section a simple, straight-forward algorithm for deriving an activity flowchart from a program is presented, and in the third section timing sequence diagrams are used to derive processing times from the activity flows. The fourth section shows how to apply lag and stretch analysis to the macro instruction processing times and the last section discusses the composite macro instructions.

The performance modelling techniques are applied in the remainder of the thesis

- to compare the performance of certain types of applications on two existing systems and

- to develop a design methodology for determining the effect of various design tradeoffs on system performance.

5.1 Macro Instructions

Macro instructions were formally defined in Chapter 4. In this section, how to actually select macro instructions is discussed.

The choice of a macro instruction set is an important one. Of course, the macro instructions must have the properties discussed in Chapter 4. The choice also must meet the requirements of the user. Complete rules for making the choice are impossible to devise, but in what follows there are some guidelines for the design of a macro instruction set. Some examples of macro instructions can be found in the next chapter, which contains a part of a macro instruction set in a performance model for comparing two existing graphics systems.

From the point of view of a user who wants to estimate the performance of an application for a particular system, the macro instruction set should contain instructions in which it is convenient to express the application. The macro instructions should be at a level of abstraction that permits all capabilities of the graphics system to be taken advantage of. For example, if a user wants his application to continually update a coordinate according to a control dial, the instructions "read a dial" and "update coordinate" are at too low a level if the graphics system allows a coordinate data type in the SDF that reads a dial value directly. The appropriate level would be "update coordinate according to dial value." The implementation of the macro instruction may still, under certain circumstances, be two user instructions that correspond to the two macro instructions above. However, in order to use the performance model to determine which implementation is appropriate, the higher level instruction should be included in the instruction set.

If the intent of a particular performance analysis is to compare the performance of two or more systems for a particular application, care must be taken that the macro instruction set does not favor one of the systems significantly. The macro instruction set must be such that the application expressed in it can be implemented in the most efficient way for each system. This may mean that, for one of the systems being compared, the macro instruction set is at a higher level of abstraction than is dictated by that particular system alone.

Some further guidelines for the design of macro instructions are derived from the properties the macro instructions must possess. First, recall that every macro instruction is independent from every other macro instruction. This means that the processing of every macro instruction is independent of the processing of every other macro instruction. User instructions are, however, often interdependent since some user instructions modally determine aspects of the processing of other user instructions. For example, a viewing transformation user instruction determines the processing of subsequent line user instructions. In order to create macro instructions that are independent of each other, each type of processing of a user instruction corresponds to one macro instruction. For example, each type of line clipping (inside, outside, and crossing the window) gives rise to a different line macro instruction.

Secondly, the processing of each macro instruction is independent of load. Such a dependence could rarely occur since the user instructions are generally independent of load. However, there are instances when dependency among user instructions can occur. For example, the contents of a buffer may increase as the number of user instructions increases. If the processing of a "candidate" macro instruction changes as the number of macro instructions increases, this candidate instruction is treated as two or more different macro instructions with different processing requirements. The load factor is added as another argument to the relevant macro function. For any one load, the appropriate macro instruction is obtained by using the appropriate value for the argument.

Third, there is only one possible sequence of I/O requests in the implementation of a macro instruction. If there are multiple I/O sequences possible for a "candidate" macro instruction, this instruction should be treated as several different macro instructions, each with one type of I/O request sequence. This is accomplished by adding another argument to the macro function that distinguishes the I/O paths.

Finally, since a processing descriptor can only take a discrete set of values, there are only a finite set of macro instructions. Thus, it is necessary to categorize analog system behavior by discrete processing descriptors. For example, line drawing times are defined in a set of ranges of values, where the line drawing time for each range is the average line drawing time for a line in that range.

5.2 Activity Flows

The compute time for a macro instruction is derived from the routine(s) in each stage of the pipeline for the associated user instructions. For the first stage in the pipeline, the user instructions would generally be programmed in assembler language; for the second and third stages the user instructions would generally be expressed in firmware code.

An activity flow is derived from a processing flowchart. Therefore, first, a processing flowchart for the routines are drawn. The flowchart uses some common flowcharting symbols for processes, decisions, and junctions as illustrated in Figure 5-1. Any initiation of a concurrent activity, either for I/O or for a parallel process, is denoted by the first symbol in Figure 5-2. Similarly, the point where the result of a read operation is needed or where the parallel process must be completed is denoted by the second symbol in Figure 5-2. For simplicity, the third symbol in this figure is used when a join of some concurrent processes is immediately followed by the initiation of some concurrent processes.

Secondly, the processing flowchart is reduced in the following manner:

1) Replace a sequence of processing boxes by a single processing box denoting the number of instructions and the total processing time.

2) At a decision point remove any path that does not pertain to the current macro instruction.

3) At each decision point estimate the probability that each branch will be executed and annotate the branch with this probability.

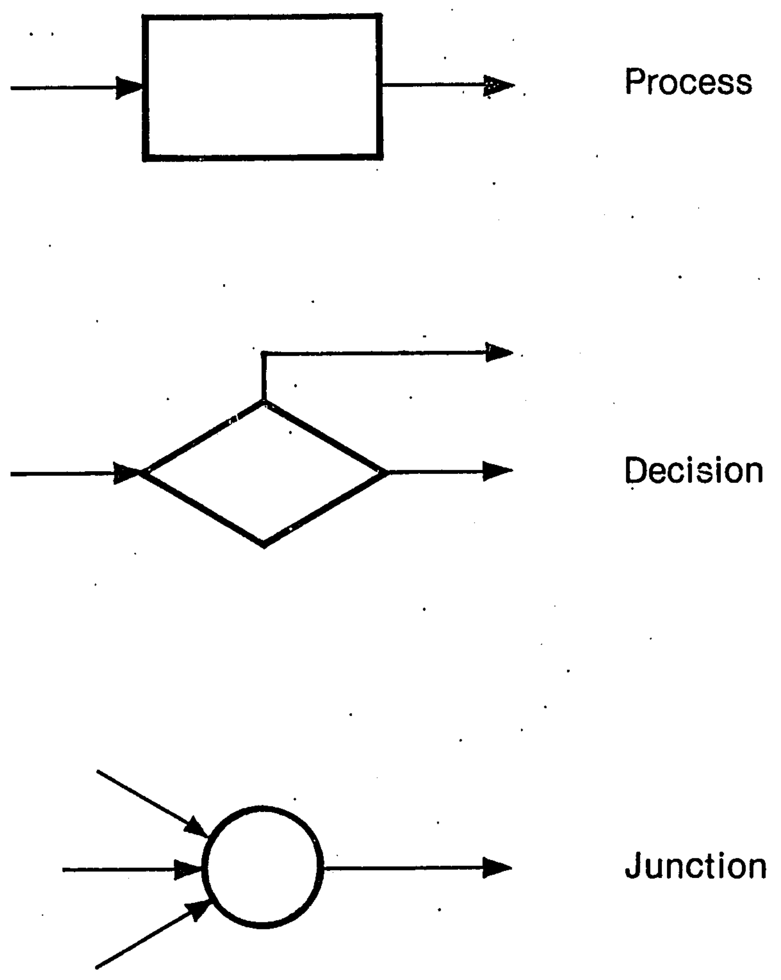


FIGURE 5-1
Processing Flow Symbols

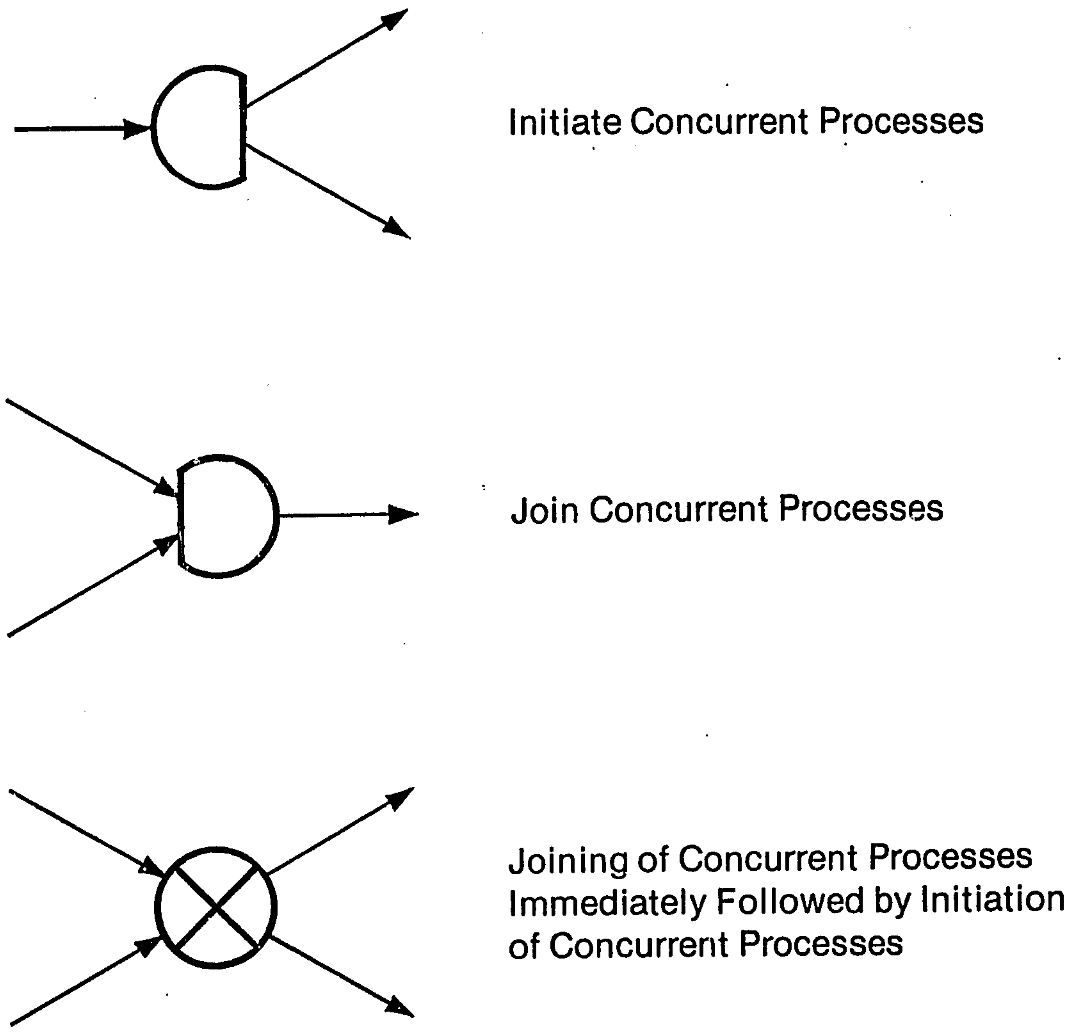

Initiate Concurrent Processes
Join Concurrent Processes
Joining of Concurrent Processes Immediately Followed by Initiation of Concurrent Processes

FIGURE 5-2
Activity Flow Symbols

4) At each decision point remove any branches that deal with error conditions and remove any branch that has a low probability of being processed, i.e., the branches that will not contribute significantly to the performance measures.

5) If all branches at a decision point only contain processing boxes followed by a junction, replace all branches with one processing box denoting the average processing time and average number of instructions, weighted by the probabilities at the decision point. If the processing times in the different branches differ significantly, it may be necessary to split the macro instruction into two or more macro instructions, with dissimilar branches corresponding to different macro instructions. This is to avoid introducing significant errors in forming the estimates.

6) In case of a loop, estimate the number of times it will be executed. If the loop does not contain any concurrent activities, replace the loop with a processing box denoting the total processing time and number of instructions executed in the loop. If the loop contains any concurrent activities, expand the loop the number of times it would be executed. If the number of times the loop is executed may differ significantly, it

may be necessary to split the macro instruction into two or more macro instructions, with each resulting macro instruction corresponding to a different number of iteration or to a different range of numbers of iterations. Again, this is to avoid introducing significant errors in the estimates.

7) At each subroutine call expand the called routine inline.

Repeat steps 1) through 7) until no further reductions are possible. This results in an activity flow. Because of property 3) in Section 4.1 an activity flow has only one path. It represents the processing of a macro instruction delineated by I/O requests and concurrent processes. A typical activity flow is illustrated in Figure 5-3.

In order to adequately model system performance, an activity flow for every statistically significant macro instruction must be derived. The following sections show how activity flows are used to to derive analytical estimates for macro instruction processing times.

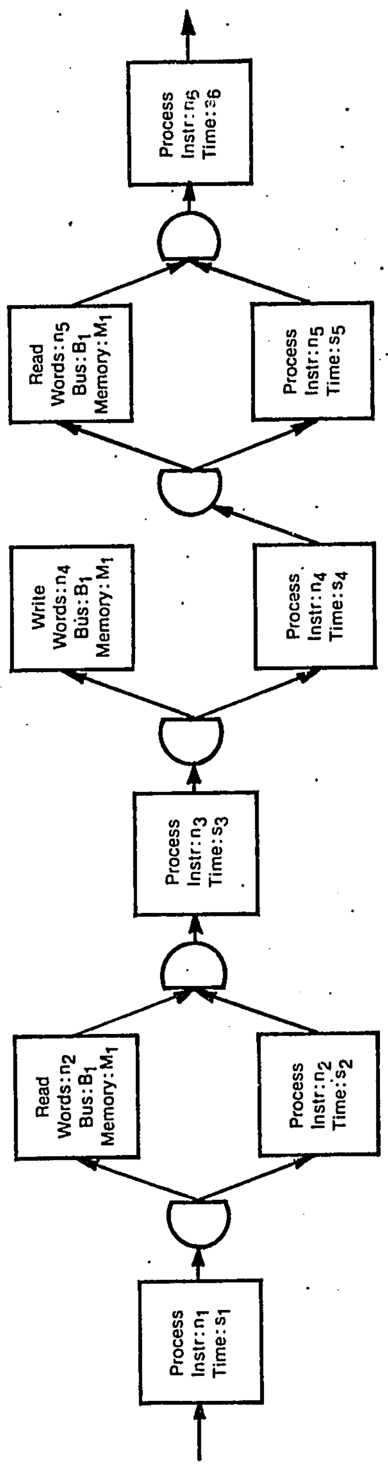
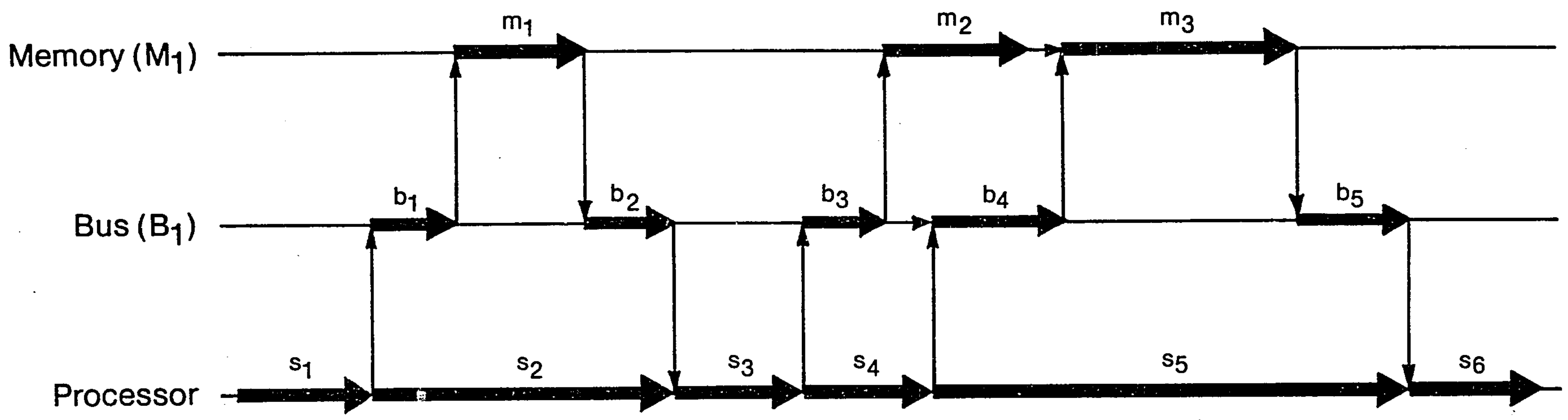


FIGURE 5-3
Activity Flow

5.3 Timing Sequence Diagrams

A timing sequence diagram [BEIZ78] is a summary of the implementation of one macro instruction in one stage of the pipeline. Such diagrams are used as an aid to derive algebraic expressions for the processing times for the macro instructions, i.e., the Q_j's.

A timing sequence diagram is derived from an activity flow. (Figure 5-4 shows a timing sequence diagram for the activity flow in Figure 5-3.) The bottom line in the diagram represents the compute time. The other lines represent other resources (including any parallel process) accessed by the processor represented by the bottom line. All physical resources such as memories, buses, and parallel processors should have their own lines in the diagram. (In Figure 5-4 the processor is reading and writing one memory over one bus.) The heavy arrows represent the time during which the resource is busy; in this example they represent the compute time, the bus transfer time, and the memory access time. The length of the heavy arrows are proportional to the time the resource is used. The light arrows represent points of synchronization. In general, both kinds of arrows indicate that the process initiated at any one point cannot begin until all the processes represented by incoming arrows have been completed.



b_i = Time Bus is Busy
m_i = (# of words) * Transfer Rate for Memory

FIGURE 5-4
Timing Sequence Diagram

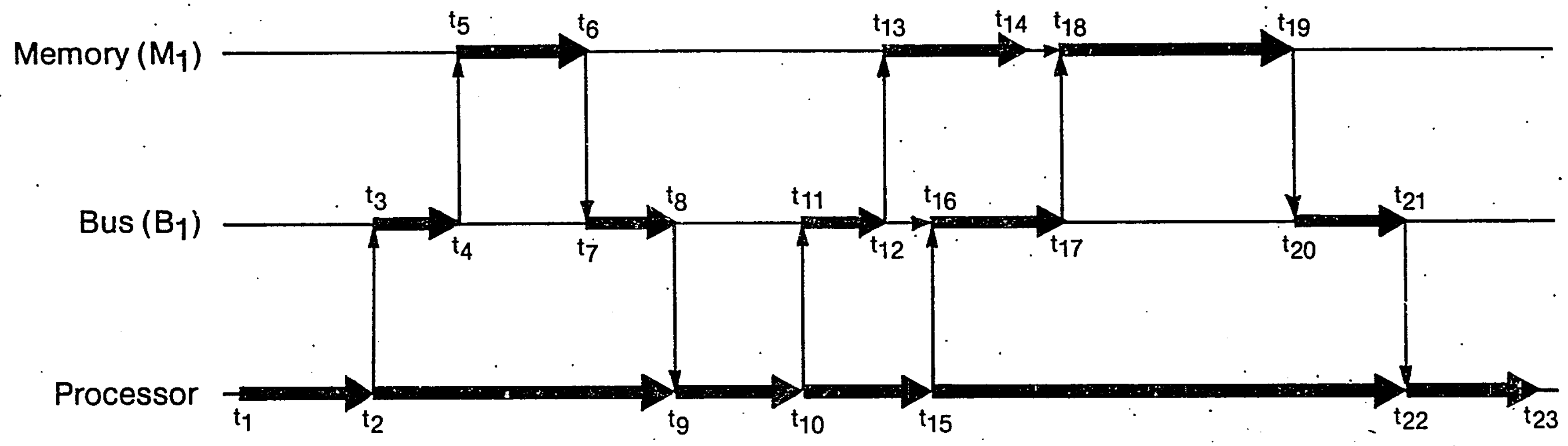


FIGURE 5-5
Timing Sequence Diagram
(with time variables, t_k, assigned)

Assuming that there are no concurrent demands on any of the resources by the two other processing stages in the pipeline, it is quite simple to compute Q_j. First a time variable t_k is assigned to each arrow junction in the diagram (see Figure 5-5). The time variable at an arrow junction is expressed as a function of the processes completing at this junction and the time variables just preceding each of these processes. Using the example in Figures 5-4 and 5-5 the following set of equations results:

$Q_j = t_{23}$

$t_{23} = t_{22} + s_6$

$t_{22} = \max(t_{21}, t_{15} + s_5)$

$t_{21} = t_{20} + b_5$

$t_{20} = t_{19}$

$t_{19} = t_{18} + m_3$

$t_{18} = \max(t_{14}, t_{17})$

$t_{17} = t_{16} + b_4$

$t_{16} = \max(t_{12}, t_{15})$

$t_{15} = t_{10} + s_4$

$t_{10} = t_9 + s_3$

$t_{14} = t_{13} + m_2$

$t_{13} = t_{12}$

$t_{12} = t_{11} + b_3$

$t_{11} = t_{10}$

$t_9 = \max(t_2 + s_2, t_8)$

$t_8 = t_7 + b_2$

$t_7 = t_6$

$t_6 = t_5 + m_1$

$t_5 = t_4$

$t_4 = t_3 + b_1$

$t_3 = t_2$

$t_2 = t_1 + s_1$

$t_1 = 0$

By successive substitutions an expression for Q_j can be derived:

$$Q_j = s_1 + s_3 + s_6 + \max(s_2, b_1 + m_1 + b_2) + \max(s_4 + s_5, m_3 + b_5 + \max(m_2 + b_3, b_4 + \max(b_3, s_4))),$$ or

$$Q_j = s_1 + s_2 + s_6 + \max(s_1, b_1 + m_1 + b_2) + \max(s_4 + s_5, m_2 + b_3 + m_3 + b_5, m_3 + b_3 + b_4 + b_5, m_3 + b_4 + b_5 + s_4)$$

The equation for Q_j expresses the processing time for a macro instruction as a function of the processor execute time, the memory and bus transfer times, the I/O-processor overlap, and the overlap by parallel processing. The equation does not, however, take into consideration the simultaneous demands on the resources by the other processing stages. In the next section, the formulas for the Q_j's will be modified to include the delay induced by the concurrent demand on the resources.

5.4 Application of Delays

In the previous chapter the formulas for stretch and lag were derived assuming that the processes contending for the resource both were running the entire time interval of interest. However, this is not generally the case for the processes in the three stages of the pipeline. A typical situation is illustrated in Figure 5-6.

In this figure, the time interval is 1/10th of a second, i.e., the length of one update interval in a typical application program. The process in the first stage of the pipeline (the

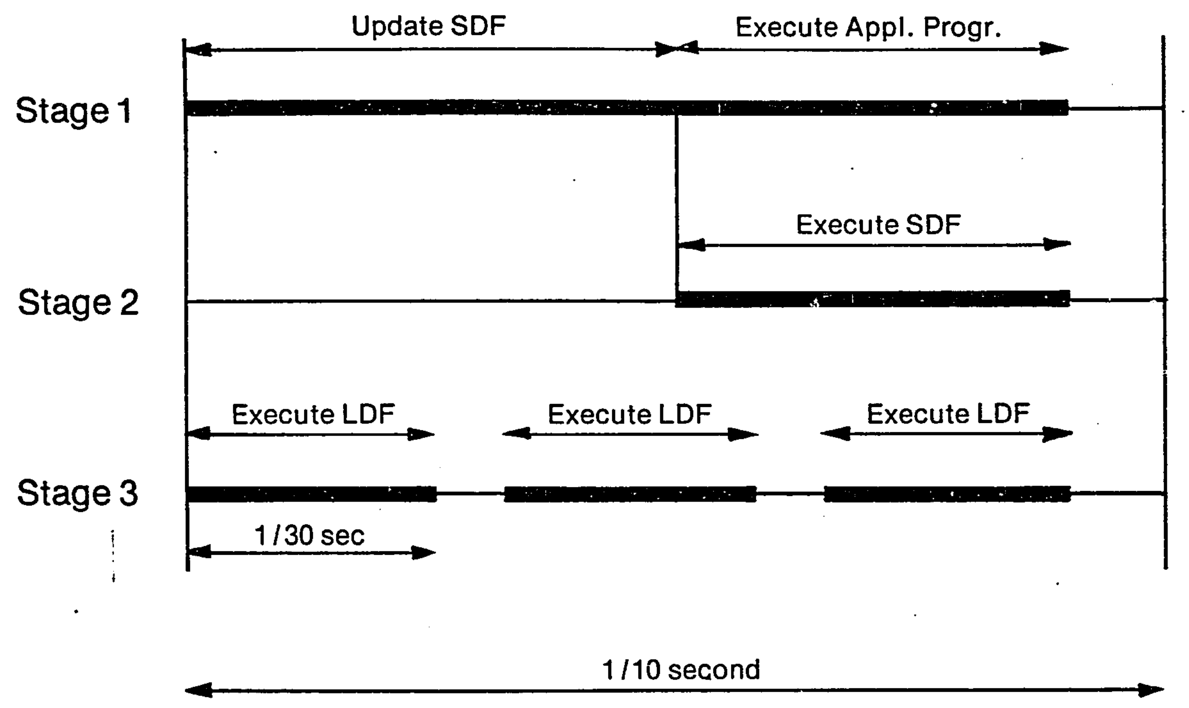


FIGURE 5-6
Concurrent Processing of Three Stages

DFC) is executed first, i.e., the SDF is updated, and, at the end of the update, the DPU starts executing the SDF. In this example, the processor running the DFC next continues processing the application program. The DCU executes the LDF three times during the update interval, i.e., it executes the LDF every 1/30th of a second. (Note that other types of synchronization of the three stages are possible.)

If the delay is applied globally over the entire time interval, some very pessimistic performance measures may result. In order to derive some better approximations of the delays, formulas for local delay must be devised.

An additional consideration is that the expressions for stretch and lag in the previous chapter were derived assuming that the contending resource was not delayed. This may also result in performance measures that are too pessimistic. The first section below develops expressions for local delay, and the second section presents an algorithm that combines delays for two processes.

5.4.1 LOCAL DELAY

In order to derive some analytical expressions for local delay, we return to the simple model of two processes and one resource. The two processes are denoted P_1 and P_2, and the resource is denoted R. The following variables will be used throughout the section:

T_1 is the elapsed time for P_1 in absence of delay.

T_2 is the elapsed time for P_2 in absence of delay.

D_2 is the delay for P_2 assuming that P_1 delays P_2's processing during all its (P_1's) processing, and can be evaluated using either lag or stretch.

E_2 is the elapsed time for P_2 including delay.

b_1 is the fraction of time T_1 that R is busy serving P_1.

b_2 is the fraction of time T_2 that R is busy serving P_2.

It is assumed that both processes begin at the same time, i.e., that in the absence of contention, P_1 is processing during the time interval $(0,T_1)$ and P_2 during $(0,T_2)$. In the case that both processes do not begin at the same time, the delay should only be applied from the first point where both are processing.

Assume there are only two processes executing during (part of) a time interval T, and that both processes access a resource R. Assume also that P_2 gets delayed by P_1. This delay only applies to the portion of P_2's processing that is overlapped by P_1's processing. In order to find the delay in this situation, we compare the time of P_1's processing to the time of P_2's processing including delay.

There are two cases to consider. Either the elapsed time for P_1 is greater that the elapsed time for P_2 including delay or it is less[9].

Case 1: $T_1 \geq E_2$.

In this case, P_1 clearly delays P_2's processing during its entire processing interval. Hence,

$$E_2 = T_2 + D_2$$

where D_2 is calculated assuming that P_1 accesses R for time $b_1 \cdot T_2$ during T_2.

Case 2: $T_1 < E_2$.

In this case, P_1 delays P_2's processing during an amount x of processing by P_2 (in absence of contention) and $x < T_2$. Delay affects P_2 during all of P_1's processing, i.e.,

$$T_1 = x + \frac{x \cdot D_2}{T_2}$$

where D_2 is calculated assuming that P_1 accesses R for time $b_1 \cdot T_2$

[9]The derivations for local delay are adapted from [BEIZ78].

during T_2. The portion of T_2 that is not delayed is T_2-x, and

$$E_2 = x + \frac{x \cdot D_2}{T_2} + T_2 - x \,.$$

Using the equation above,

$$E_2 = T_2 + \frac{D_2 \cdot T_1}{T_2+D_2} \,.$$

According to our assumption, T_1 is less than E_2, i.e.,

$$T_1 < T_2 + \frac{D_2 \cdot T_1}{T_2+D_2}$$

which gives

$$T_1 < T_2+D_2$$

which was indeed the assumption in this case and is the complement of the assumption in Case 1.

In summary, if $T_1 \geq T_2+D_2$ then $E_2 = T_2+D_2$ and if $T_1 < T_2+D_2$ then $E_2 = T_2+(D_2 \cdot T_1)/(T_2+D_2)$.

5.4.2 COMBINING DELAYS FOR PROCESSES

The discussion above showed how to compute delay for P_1 assuming P_2 was not delayed, and conversely, how to compute delay for P_2 assuming P_1 was not delayed. However, this may result in performance measures that are too pessimistic. By applying the following algorithm, a more accurate measure of delay for both P_1 and P_2 is obtained. (In what follows the notation of the previous section is used.)

1) Set $T_2' = T_2$.

2) Calculate the delay for P_1 using the formulas of Section 5.4.1. The delay is calculated assuming that P_2 accesses the resource R for the fraction $b_2' = (b_2 \cdot T_2)/T_2'$ of the interval T_1. Set $T_1' = E_1$.

3) Calculate the delay for P_2. The delay is calculated assuming that P_1 accesses the resource R for the fraction $b_1' = (b_1 \cdot T_1)/T_1'$ of the interval T_2. Set $T_2' = E_2$.

Iterate steps 2) and 3) until the values for E_1 and E_2 stabilize. It can be shown E_1 and E_2 always do stabilize. E_1 decreases monotonically with decreasing b_2' and is bounded below by T_1. Similarly, E_2 increases monotonically with increasing b_1' and is bounded above by T_2+D_2', where D_2' is calculated assuming P_1 is not delayed.

In graphics systems, as we have modelled them, there are three processes running in parallel, which makes the derivation of the processing times for each stage of the pipeline, the U_1, U_2, and U_3, more complicated. The idea is the same; first apply the delay to the processing times pairwise and then iterate until the values for the elapsed times stabilize.

Recall that each U is a weighted average of some Q_j's that are expressions of the form

$$\sum_k s_{jk} + \sum \max(\sum_k s_{jk} + \sum_k r_{jk} , \ldots , \sum_k s_{jk} + \sum_k r_{jk}),$$

where s_{jk} denotes the processing time for a processing component, and r_{jk} denotes the time for a concurrent activity. Some of the r_{jk}'s represent resource requests that have an associated delay due to contention for the resource. In Theorems 4.5 and 4.6 it was shown that the expected value for the delay d can be expressed as $\bar{d} = p \cdot \bar{x}$, where p is the probability that the resource request is delayed and $\bar{x}$ is the expected duration of the delay. The objective in what follows is to derive an expected value for each of the max terms in U.

For simplicity let us assume that each max term in U has the form

$$Y = \max(d+s_1,\ s),$$

where d represents the delay, s_1 represents the time a resource R is servicing a request plus other processing time that is independent of R, and s represents processing that is independent of R.

If $s_1 \geq s$, then the expected value of Y is $\bar{Y} = s_1 + p\cdot\bar{x}$. If $s > d_{max} + s_1$, where d_{max} is the maximum delay, then $\bar{Y} = s$. If, however, $s_1 < s < d_{max} + s_1$, then the expected value of Y can be expressed as

$$\bar{Y} = (1-p)\cdot s + p\cdot\bar{Y}_1,$$

where $\bar{Y}_1$ is the expected value of $\max(s_1+x,\ s)$, assuming that there is some delay, x, at each request.

Assume that x has a distribution with a probability density function

$$g(x) = \begin{cases} f(x) & \text{for } 0 \leq x \leq b, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Then the expected value of Y_1 is

$$\bar{Y}_1 = \int_0^{s-s_1} s \cdot f(x) \cdot dx + \int_{s-s_1}^{b} (x+s_1) \cdot f(x) \cdot dx \qquad (5.4.2-0)$$

Case 1: The service time for one request is a constant c. This means that x is uniformly distributed with a probability density function of 1/c for x between 0 and c. Hence, the expected value of Y_1 is

$$\bar{Y}_1 = \int_0^{s-s_1} (s/c) \cdot dx + \int_{s-s_1}^{c} (x+s_1)/c \cdot dx$$

which, simplified, yields

$$\bar{Y}_1 = c/2 + s_1 + (s-s_1)^2/(2 \cdot c) \qquad (5.4.2-1)$$

Case 2: The service time for one request is exponentially distributed. This means that x is also exponentially distributed, with a probability density function of $(1/c) \cdot \exp(-x/c)$ for non-negative x (0 otherwise). The expected value of Y_1 is again found by substituting into (5.4.2-0). The resulting value is given by

$$\bar{Y}_1 = c \cdot \exp(-(s-s_1)/c) + s. \qquad (5.4.2-2)$$

In summary, the expected value of Y is

$$\bar{Y} = \begin{cases} s_1 + p \cdot \bar{x} & \text{if } s_1 \geq s \\ (1-p) \cdot s + p \cdot \bar{Y}_1 & \text{if } s_1 \leq s \leq d_{max} + s_1 \\ s & \text{if } s \geq d_{max} + s_1 \end{cases} \qquad (5.4.2\text{-}3)$$

and $\bar{Y}_1$ is an expression as derived above that depends on the distribution of x. The delay, if the contending processor is not delayed, is $p \cdot \bar{x}$ in the first case, $p \cdot (\bar{Y}_1 - s)$ in the second case, and zero in the third case.

The algorithm to calculate U for each stage can be expressed as:

1) Assign values to the I/O hardware parameters (either for an existing system or for a system under design) and use these values to substitute values for the r_{jk}'s.

2) Using equation (5.4.2-3), calculate the processing time for each stage and the associated delay. The delay calculation for each stage assumes that the contending processors are not delayed, and that all processors run during the entire observation interval.

3) Using the iterative procedure discussed above, combine the delays for the processes pairwise, and calculate the effective times for U in each stage of the pipeline.

4) Repeat step 3) until the values for the elapsed processing times stabilize.

5.5 Composite Macro Instructions

As was mentioned in Section 4.1, a very large number of different macro instructions are needed to describe all possible execution paths through the user instructions at each stage of the pipeline. In order to make it easier to define a macro instruction set, macro functions were introduced as a shorthand notation for a (sub)set of the macro instructions; the value of a macro function for a particular set of arguments is a macro instruction. Although the macro functions aid in the definition of macro instructions, it is not always convenient to specify an application in terms of the number of occurances of a macro function for each set of arguments.

It is often easier to think of an application consisting of some number of primitives of which a certain percentage are visible, and the remainder is not, of which some percentage are static and the rest dynamic, and so on. In other words, it is convenient to attach a percentage, or fraction, to the occurance of each of the values of some or all processing descriptors. By forming the products of all possible combinations of occurance percentages for the values of the processing descriptors, weights representing the occurance percentage of the macro instructions are created. A weighted average of several macro instructions will be called a <u>composite macro instructions</u>.

Note that there is not necessarily a one to one correspondence between composite macro instructions and macro functions. A composite macro instruction can encompass several macro functions or, alternatively, correspond to a subset of all of the processing descriptor values for one macro function.

More formally, we create a composite macro instruction the following way. Assume that there is a macro function whose arguments are a_k , $k=1, \ldots, M$. For some of the a_k's, there are weights w_{kl} , $l=1, \ldots, N_k$ associated with each of the N_k possible values of a_k. (Note that $\sum_{l=1}^{N_k} w_{kl} = 1$.) In general some of the arguments will be assigned a constant value, and the remaining a_k's, with weights, will vary over all possible values. The processing time for the composite macro instruction is an expression of the form

$$\sum_{l} w_{1l} \sum_{m} w_{2m} \cdots \sum_{n} w_{Mn} \cdot Q_{I(l,m,\ldots\ldots,n)} \quad (\ldots)$$

where $I(l,m, \ldots, n)$ is an integer function that maps the indices of the arguments to an integer denoting the ordinal number of the macro instruction. This composite macro instruction only encompasses (part of) one macro function. By attaching weights to two such composite macro instructions and adding the weighted processing times, the processing times for a composite macro instruction that encompasses more that one macro function is obtained.

At first glance it may appear as if an expression for the processing time of a composite macro instruction could become very long. However, the expression can generally be simplified a great deal. Q_j is zero for a large number of macro instructions, in particular in the third stage of the pipeline. Futhermore, many Q_j's will be identical. This will occur whenever the implementations of two macro instructions in one processor are the same even though their implementation differ in a different processor.

In the next chapter, composite macro instructions will be used to compare features of two existing graphics systems.

6 COMPARISONS OF GRAPHICS SYSTEMS

6.1 General Methodology of Comparison

The performance modelling techniques discussed in Chapter 4 can be used as an aid in determining which of several graphics systems is the most suitable for a particular application from the point of view of performance, as defined in this thesis. (Issues such as ease of use, reliability, cost, etc., are not considered.) The use of the performance modelling techniques for this purpose is quite straightforward.

1) A set of macro instructions that is suitable for all the graphics systems in question is derived.

2) For each graphics system the expression for the processing times, the Q_j's, for every macro instruction is derived in absence of any contention.

3) All relevant loads for the application program are expressed in terms of the macro instructions.

4) For each of these loads, for each system, the delay is computed and applied to the expressions for the Q_j's.

5) For each load, for each system, the total processing times for each stage in the pipeline, the U's, are derived.

By comparing the resulting U's for the different systems it is easy to determine which of the graphics system(s) meet the application requirements.

By repeating this procedure for several applications that are "typical" for an environment, one can establish a firm basis for an opinion regarding which graphics system is the most suitable for this environment. This approach is suitable if the needs of a particular environment are limited to a few types of applications, but when the scope of applications is larger and not well defined, this approach is not practical. Even in this case, however, the performance modelling techniques can be used for quantitative comparisons of graphics systems.

In order to make comparisons when the scope of the application is not well-defined, one studies the trade-offs for certain key features. For example, one could study static lines versus lines that change with a dynamically changing viewing transformation, or the effects of different addressing modes for lines, such as immediate data versus addressed data. By identifying the key features of interest for an environment and by comparing their effects on performance for different graphics systems, one can establish some quantitative measures of the relative performance of the systems under consiration.

One way to study trade-offs for a key feature is via a composite macro instruction. The weights of processing descriptor values that are not of interest are held constant in the composite macro instruction, while the remaining weights are varied. This technique will illustrate the effects the features of interest have on performance. Composite macro instructions will be used in this chapter to illustrate some differences between the Evans and Sutherland Picture System 2 and the Vector General 3400.

6.2 Introduction to the Case Study

6.2.1 SELECTION OF DEVICES FOR THE CASE STUDY

There were four devices that potentially could have been used for a case study in this thesis. All four are "high-performance", as defined in this thesis, and data regarding the internal operations of the systems was available. The four candidates were: three commercial systems, the Adage 4100[10], the

[10]The Adage 4100 does not support perspective projection in the

Vector General 3400, the Evans and Sutherland Picture System 2, and one non-commercial system, the Brown University Graphics System. The Vector General 3400 and the Evans and Sutherland Picture System 2 were chosen for the following reasons:

- both were developed in the same time frame,
- both were developed for the commercial market. As such, neither had the financial constraints of a university environment, nor were they guided by the special interests of a university environment,
- both interface to the same set of host computers, the DEC PDP-11's, (and more recently also to the VAX 780), and
- these two systems employ radically different approaches to graphics system design.

The E&S PS2 and the VG3400 are, as was just mentioned, the result of two different design approaches. The E&S PS2 DPU has a limited instruction set operating only on immediate data. The VG3400 DPU, on the other hand, has a versatile, general purpose instruction set and a large number of addressing modes. While the E&S PS2 DPU is very fast, the VG3400 has paid a price in DPU processing speed for its versatility. These differences make the comparisons of the two systems particularly interesting; not only

firmware, but, because of its speed, can nevertheless be considered a high-performance system.

will they serve as an illustration of the performance modelling techniques in Chapter 4, but will illustrate some of the advantages and disadvantages of the two approaches from a performance point of view.

In chosing the host, any of the PDP-11's from the 11/34 to the 11/70 [DEC74, DEC76, DEC78] could have been selected. The analysis in this thesis uses the 11/45. Both graphics systems were designed for the 11/34 and the 11/45, and, for this reason, the vendors could only provide timing estimates for these hosts. The analysis will, however, be briefly extended to the 11/70, in order to illustrate the effects the choice of a CPU has on the performance of these systems.

6.2.2 SELECTION OF FEATURES FOR THE CASE STUDY

A conclusive comparison between the two systems would require that the performance for a very large set of loads covering all aspects of graphics programming be compared. It is neither possible to illustrate all differences between the two systems in this thesis, nor is it the purpose of this thesis to do so. The goal is to illustrate quantitatively the difference in performance between the two systems of some features that are common in highly interactive and highly dynamic applications. In order to do so, two composite macro instructions were chosen as representative cases of these types of applications:

1) Polyline -- a composite macro instruction that represents a 3D polyline with lines inside, outside, and crossing the clip window boundaries. The coordinate values of the polyline either remain unchanged and only the image of the polyline changes dynamically, or the coordinate values are altered continually from the application program.

2) Translation function -- a composite macro instruction that represents a modelling transformation function that translates an object relative to its local origin. The translation values either remain unchanged, or are continually updated from control dial values.

6.2.3 COLLECTION OF EXPERIMENTAL DATA

The data upon which the analysis in this chapter was based came from a variety of sources. Although some data came from reference manuals and hardware manuals, these sources were inadequate. Much of the detailed material was derived from firmware code for the DPU user instruction sets, and from hardware timing diagrams documenting I/O sequencing. In addition, there were numerous contacts with engineerers of Evans and

Sutherland Computer Corporation, Vector General, Inc., and Digital Equipment Corporation in order to obtain the remaining material. Finally, some of the comparisons required that implementations of macro instructions be programmed, and timing derived from these programs.

In the first of the following sections, a subset of a macro instruction set suitable for comparing some aspects of the two systems is derived. In the second section, a composite macro instruction set is developed from the macro instructions, and the performance figures for several features are determined applying the variable-weight technique described in the first section of this chapter. The final section summarizes the findings in this chapter and discusses validation and calibration of the performance model.

6.3 A Macro Instruction Subset

The macro instruction subset to be studied is represented by three macro functions; two for line drawing, "move" and "draw", and one for modelling transformations, "model". First, line macro functions for each individual system are proposed, each of which represents a set of macro instructions at a suitable level of abstraction so as to take advantage of the capabilities of the individual graphics systems. Second, the "move" macro functions for the two systems are combined into one function, and similarly, the "draw" macro functions are combined into one function; the new "move" and "draw" macro functions represent a set of line macro instructions for both systems. The same procedure is then repeated to derive a modelling transformation macro function.

The derivation of a macro function for a particular feature(s) is accomplished as follows. The user instructions relevant to the feature(s) of interest are examined at each stage of the picture processing pipeline to establish which parameters determine the different processing paths and processing characteristics. These parameters determine the processing descriptors of the macro function.

6.3.1 LINE MACRO FUNCTIONS

We first consider the move and draw macro functions for each system individually, and then we combine the two sets of line drawing functions into one set of functions.

Starting with the E&S PS2 [ESCC77, ESCC77a] we readily identify the following parameters pertaining to line processing in each stage of the pipeline and their associated values. (The parameters that correspond exactly to processing descriptors are underlined.)

parameter	values
STAGE 3	
draw length	ranges of values
move length	ranges of values
STAGE 2	
clipping	disabled
	nonvisible
	visible -- line completely visible
	exit -- line exiting clip window
	enter -- line entering clip window
	enter-exit -- line both entering and exiting window
dimensionality	2D
	3D
	4D
mode of data	absolute -- value is an absolute coordinate
	relative -- value is defined relative to the previous coordinate

	offset -- value is relative to a particular value
<u>first</u>	line endpoint is first in a sequence of lines
	not first
<u>beam</u>	on -- i.e., a draw
	off -- i.e., a move

STAGE 1

<u>alter</u>	no -- no addition or deletion
	add -- line endpoint is added to the SDF
	delete -- line endpoint is deleted from the SDF
<u>modify</u>	no -- no modification
	transform -- change associated modelling or viewing transformation
	CPU -- change coordinate value from application program

Some of the parameters require a brief explanation. When "alter" has the value "add", the remaining parameters define the type of line that is added to the SDF, as well as the type of processing of the line in the DPU and DCU. When "alter" has the value "delete", "modify" has no meaning. When "alter" has the value "add", "modify" can specify a line whose image is not altered at all ("no"), or a line whose image is changed by a modification in a viewing or modelling transformation ("transform"), or a line changed by the application program ("CPU"). In the first case the line could be stored in one segment in the linear buffer without a copy in the SDF; in the other cases a copy of the line must be stored in the SDF.

The parameter "first" refers to the first line in a sequence of lines. This processing descriptor distinguishes a line macro instruction that represents the first line endpoint in a polyline and hence involves extra overhead in form of instruction decoding. "move length" must be specified both for an explicit move when clipping is disabled[11], and for a line that reenters the clip window when clipping is enabled.

We now propose two macro functions "ESmove" and "ESdraw", that together define line processing for the E&S PS2. One function could have been proposed that would have a "beam" processing descriptor in addition to all the others. But, because fewer parameters are relevant to "beam off" processing, two functions are proposed, one for "beam off" and the other for "beam on". "dim" and "mode" need not be specified for a move since their values do not affect the processing time due to the horizontal micro instruction set in this device. These two macro

[11]When clipping is enabled, a move instruction does not generally result in a move of the electron beam, but only in a load of some registers in the DPU.

functions have processing descriptors that correspond one for one to the parameters above.

ESmove(alter, modify, clip, first, movelength)

ESdraw(alter, modify, clip, dim, mode, first, movelength, drawlength).

For the VG3400 [VECT78b] we can identify the following parameters and associated values:

parameter	values
STAGE 3	
draw length	ranges of values
move length	ranges of values
STAGE 2	
clipping	same values as for the E&S PS2
dimensionality	2D
	3D
mode(x,y,z)[12]	absolute

[12]This is equivalent to three separate parameters mode(x), mode(y), and mode(z), each of which can take on the values "absolute" and "relative".

	relative
first	line endpoint is first in sequence
	line endpoint is not first in sequence
beam	on, i.e. a draw
	off, i.e., a move
packed	word
	bytes
	byte/4
addressing modes	immediate
	reference to list
	reference in each coordinate value:
	immediate
	analog device
	stack
	register
	external
	local

STAGE 1

alter	no -- no addition or deletion
	add -- line endpoint is added to the SDF
	delete -- line endpoint is deleted from the SDF

Most of the parameters for the VG3400 resemble those for the E&S PS2. However, there are significant differences. For example, while E&S PS2 only allows immediate data in the SDF, the VG3400 allows a large set of different addressing modes. The data can be organized in three different ways: as immediate data, as a list of data referenced by a pointer in the SDF, or as so-called referenced data, i.e., data where each coordinate can have one of several addressing modes.

In the VG3400 each referenced coordinate can have one of eleven addressing modes, some of which are listed above. (Omitted are modes which are not relevant to any of the applications to be discussed.) Some of the addressing modes listed above require explanation: "analog device" refers to data obtained from an analog input device; "stack" refers to data in the object call stack; "register" refers to data in a general purpose GPU register; "external" refers to data addressed through the directory; and "local" refers to data referenced through a data area called "local own" that is associated with each object. Each value addressed in one of the modes above may either be coordinate data, or a pointer to some data, which in turn can be a pointer, and so on. This means there is potentially an infinite number of addressing modes. Modification of a line endpoint in the VG3400 is made not through explicit changes to the SDF, but through the modification of data referenced by pointers in the SDF.

Other differences between the two systems should be noted. First, whereas the E&S PS2 allows line sequences that start with a move or a draw, the line sequences in the VG3400 always begin with a move. Therefore, the VG3400 draw function does not need a processing descriptor "first". Secondly, whereas dimensionality does not affect the processing time for a move in the E&S PS2 it does affect the move processing time in the VG3400. The same is true for the mode of the coordinate data for a move. These differences are reflected in the processing descriptors for the macro functions below.

For the VG3400, separate move and draw macro functions are also desirable. The processing descriptors of these functions are the parameters listed above, except for the new modify(x,y,z) descriptor explained below.

VGmove(alter, modify(x,y,z), clip, dim, mode(x,y,z), first,
    packed, movelength)

VGdraw(alter, modify(x,y,z), clip, dim, mode(x,y,z), packed,
    drawlength, movelength).

The processing descriptor "modify" takes a set of values which represent the different types of modification that are possible through the use of different addressing modes.

descriptor	value
modify(x,y,z)[13]	no --no modification CPU -- value is modified from the application program and is addressed by a pointer in the SDF device -- value is modified by an analog device register -- value is modified in a general purpose register stack -- value is modified in the object call stack

There are, of course, several more possible values for this processing descriptor if all possible addressing modes are to be included. The values listed above are, however, sufficient for the purpose of illustrating the design and use of a macro instruction set for comparison of the two systems.

In order to create move and draw macro functions for both the systems, the processing descriptors for the two systems are combined, that is, where a processing descriptor occurs in only one system, it is included in the result, and where the same processing descriptors appear in both systems, the value spaces

[13]This is equivalent to three separate parameters modify(x), modify(y), and modify(z), each of which can take on the values listed.

are unioned. This insures that the functional capabilities pertaining to lines in each system are included and that the instruction set is not biased towards one of the systems, but that all line features can be implemented as efficiently as each system permits. The format of the macro functions for the two systems are:

```
move(alter, modify(x,y,z), clip, dim, mode(x,y,z), first,
    packed, movelength)

draw(alter, modify(x,y,z), clip, dim, mode(x,y,z), first,
    packed, movelength, drawlength).
```

These two macro functions will be used in a later section to compare some line processing features in the two systems.

6.3.2 A MODELLING TRANSFORMATION MACRO FUNCTION

The procedure for deriving a macro function for a modelling transformation for the two systems parallels that for the line functions. We first consider the "model" macro function for each system individually, and then we combine the two functions into one function for both systems.

In the E&S PS2 a modelling transformation does not generate any code for the DCU. In the E&S PS2 there is only one instruction pertaining to modelling transformations: "concatenate matrix", and all modifications of a modelling transformation take place in the CPU.

In order that a modelling transformation be implemented in the most efficient manner in the E&S PS2, matrices that are part of the same modelling transformation and that are not to be modified should be concatenated in the CPU before they are added to the SDF. On the other hand, matrices that are to be modified continually should be concatenated by the DPU, where matrix concatenation is much faster than in the CPU. Therefore, a transformation matrix that is added to the SDF must be identified as being either modifiable or non-modifiable. If a matrix is non-modifiable, it is also necessary to specify if it is the first matrix, the last matrix, or a matrix in the middle of a sequence of non-modifiable matrices. The first matrix is simply saved in the CPU, a matrix in the middle of a sequence is concatenated with the previous matrix in the sequence, and the last matrix in a sequence is concatenated with the previous matrices in the sequence and the result is written into the SDF.

From these considerations, one determines that a modelling macro function for the E&S PS2 can have the format:

ESmodel(alter, modify, sequence, type),

where the processing descriptors have the following values:

descriptor	value
alter	no -- no addition or deletion
	delete -- delete matrix
	add -- add matrix
modify	yes -- it is (to be) modified
	no -- it is not (to be) modified
sequence[14]	first
	middle
	last
type	scale(x,y,z,w)
	translate(x,y,z)
	rotate(x,y,z)
	all -- an entire matrix is to be added or modified

The processing descriptor "modify" has different meanings depending on the value of "alter". If "alter" has the value "add", "modify" and "sequence" determine if the matrix is added to the SDF or concatenated with the previous matrix, for reasons described above. If "alter" has either the value "no" or "add", "modify" determines if a matrix already in the SDF or just added to the SDF should be modified. The processing descriptor "type"

[14]Only relevant if "alter" has the value "add" and "modify" has the value "no".

specifies which scale, translate, or rotate coordinate(s) are added or modified. One or more values can be specified at each "type" definition. For example, "scale(x,,z)" means that the x and z scale are added to or modified in the SDF and that the y and w scale equal one in case of an add, and are unchanged in case of a modify.

In the VG3400, as for the E&S PS2, a modelling transformation does not generate any code for the DCU. In the VG3400, modification of a modelling transformation is accomplished just as for line endpoints, i.e., through the use of different addressing modes. A modelling macro function for the VG3400 can have the format:

VGmodel(alter, modify(x,y,z,w), type),

where the processing descriptors have the following values:

descriptor	value
alter	no -- no addition or deletion
	add -- add matrix
	delete -- delete matrix
modify(x,y,z,w)[15]	no -- no modification

[15]Each of the x,y,z, and w coordinates can have the values listed. The modes of modification selected affect the corresponding parameters of the type processing descriptor.

	CPU -- modification from the application program
	device -- modification by an analog device
	register -- modification in a general purpose register
	stack -- modification in the object call stack
<u>type</u>	scale(w)/translate(x,y,z)[16]
	translate(x,y,z)
	rotate(x,y,z)
	scale[17]

By combining the processing descriptors[18] for the two systems we get the following macro function:

model(alter, modify(x,y,z,w), sequence, type),

where the range of values for the processing descriptors are the union of the possible values for the processing descriptors for the two systems.

[16]Uniform scale and translate can be combined for more efficient execution times.

[17]VG3400 allows only uniform scaling.

[18]Since only uniform scaling is allowed in the VG3400 DPU, non-uniform scaling is excluded from the combined macro function.

In the next section, composite macro instructions based on this macro function will be used to compare the behavior of certain transformation features in the two systems.

6.4 Composite Macro Instructions for E&S PS2 and VG3400

In this section implementations of some operations, common in graphics applications, are compared for the E&S PS2 and the VG3400. The first set of operations concern line processing; the features of interest are line drawing data in the SDF whose coordinate values are modified from the application program and line drawing data whose coordinate values are not modified but whose image is modified through a change in a transformation. The second set of operations concern modelling transformations; the features of interest are modelling transformations that are not modified and modelling transformations that are modified from some analog device. The differences in processing times are illustrated using composite macro instructions for lines and for modelling transformations.

6.4.1 A COMPOSITE LINE MACRO INSTRUCTION

The move and draw macro functions from Section 6.3.1 will be used to create a polyline composite macro instruction, where the first instruction is a move and the remaining are draws. The relevant macro functions from the previous section have the format:

move(alter, modify(x,y,z), clip, dim, mode(x,y,z), first,
    packed, movelength), and

draw(alter, modify(x,y,z), clip, dim, mode(x,y,z), first,
    packed, movelength, drawlength).

A "move" composite macro instruction is obtained by assigning the following values and weights to the processing descriptors for the "move" macro function.

descriptor	value	weight
alter	no	1
modify(x,y,z)	x=y=z=no	1-t, $(0 \le t \le 1)$
	x=y=z=CPU	t
clip	enabled[19]	1

---

[19]This is any value but disabled.

dim	3D	1
mode(x,y,z)	x=y=z=absolute	1
first	yes	1
packed	fullword	1

For the purpose of this comparison, we are interested in the trade-offs in the first two stages. Hence, only the CPU and the DPU processing will be investigated, and the last two processing descriptors of "draw" will be ignored. By assigning the following values and weights to the remaining processing descriptors a "draw" composite macro instruction results.

descriptor	value	weight
alter	no	1
modify(x,y,z)	x=y=z=no	1-t, ($0 \le t \le 1$)
	x=y=z=CPU	t
clip	line nonvisible	w_1=.5
	line completely visible	w_2=.25
	line exiting clip window	w_3=.10
	line entering clip window	w_4=.10
	line both entering and exiting window	w_5=.05
dim	3D	1

mode(x,y,z)	x=y=z=absolute	1
first	no	1
packed	word	1

By assigning a weight .9 to the "draw" composite macro instruction and a weight .1 to the "move" composite macro instruction and combining the two weighted instructions into one, the resulting composite macro instruction represents a set of 3D polylines, with one move and nine draws in each sequence, and whose coordinate data is absolute, packed one coordinate per word, and of which 50% of the lines are outside the clip window and the remaining lines are either inside the window or crossing the window boundaries. The weights that are to be varied are associated with the processing descriptor "modify".

Eliminating macro instructions that correspond to processing descriptor values that have been assigned zero weights, we see that twelve macro instructions remain for consideration. They will be numbered 1 trouhgh 12 as follows:

1 move(no, (no, no, no), enabled, 3D, (abs, abs, abs),
yes, word)

2 move(no, (CPU, CPU, CPU), enabled, 3D, (abs, abs, abs),
yes, word)

3 draw(no, (no, no, no), nonvisible, 3D, (abs, abs, abs),

no, word)

4 draw(no, (no, no, no), visible, 3D, (abs, abs, abs), no, word)

5 draw(no, (no, no, no), exit, 3D, (abs, abs, abs), no, word)

6 draw(no, (no, no, no), enter, 3D, (abs, abs, abs), no, word)

7 draw(no, (no, no, no), enter-exit, 3D, (abs, abs, abs), no, word)

8 draw(no, (CPU, CPU, CPU), nonvisible, 3D, (abs, abs, abs), no, word)

9 draw(no, (CPU, CPU, CPU), visible, 3D, (abs, abs, abs), no, word)

10 draw(no, (CPU, CPU, CPU), exit, 3D, (abs, abs, abs), no, word)

11 draw(no, (CPU, CPU, CPU), enter, 3D, (abs, abs, abs), no, word)

12 draw(no, (CPU, CPU, CPU), enter-exit, 3D, (abs, abs, abs), no, word)

Recall that macro instruction timings are denoted Q_j for each stage. In what follows, it is important to distinguish the macro instruction timings in each stage; thus, these timings will be denoted Q_{ij}, where i denotes the stage in the pipeline, and j the number of the macro instruction. Composite macro instruction timings will be denoted V_i, in general, but when a specific system is under discussion the letter E or the letter V will be prefixed, in order to distinguish between the E&S PS2 and the VG3400. The processing time for the composite macro instruction is then an expression

$$V_i = .1\bullet((1-t)\bullet Q_{i1}+t\bullet Q_{i2}) + .9\bullet((1-t)\bullet \sum_{j=3}^{7} w_{j-2}\bullet Q_{ij} + t\bullet \sum_{j=8}^{12} w_{j-7}\bullet Q_{ij}) . \qquad (6.4.1-1)$$

Since all updating of line endpoints in the VG3400 is done through the use of addressed data, V_1 for the VG3400 (i.e., VV_1) is always zero. In the E&S PS2 it will be assumed that there is no overlap between the updating of the SDF by the CPU and the processing of the SDF by the DPU. Therefore, in order to compare the processing of the composite macro instructions in the two systems, VV_2 plus any delay in the CPU due to memory contention for VG3400 should be compared with EV_1+EV_2 for the E&S PS2. In order to make this comparison, the values of the Q_{ij}'s, first for the VG3400 and then for the E&S PS2, will be calculated.

In order to derive the Q_{ij}'s for the VG3400, we first derive the activity flows for each of the macro instructions. The activity flows for the first and fourth macro instruction are illustrated in Figure 6-1 and the activity flows for the second and ninth macro instruction in Figure 6-2. The activity flows for the VG3400 microcode are not very complex and the expressions for the Q_{ij}'s can be derived directly from the activity flows without the use of timing sequence diagrams.

It turns out that, for the VG3400, each term in the Q_{ij}'s for which there is concurrent I/O activity is similarly structured. In general, the expected value, $\bar{Y}$, for such terms can be expressed as (see Section 5.4.2)

$$\bar{Y} = \begin{cases} s_1 + p \cdot \bar{d} & \text{if } s_1 \geq s \\ (1-p) \cdot s + p \cdot \bar{Y}_1 & \text{if } s_1 < s < s + d_{max} \\ s & \text{if } s \geq s_1 + d_{max} \end{cases} \qquad (6.4.1\text{-}2)$$

where, in this case, s_1 is the data transfer time, s is the processing time, p is the probability of simultaneous requests by the PDP-11 and the GPU (see Figure 3-8) for memory, d is the delay at each request, $\bar{d}$ is the expected value of d, and d_{max} is the maximum delay. $\bar{Y}_1$ is the expected value of $Y_1 = \max(s_1+d, s)$ assuming that there is some delay at each request. As was discussed in the derivation of Theorem 4.5, p can be expressed as $b_1/(1-b_2)$, where b_1 is the fraction of time that the PDP-11 memory services its CPU, and b_2 is the portion of time the PDP-11 memory services the GPU.

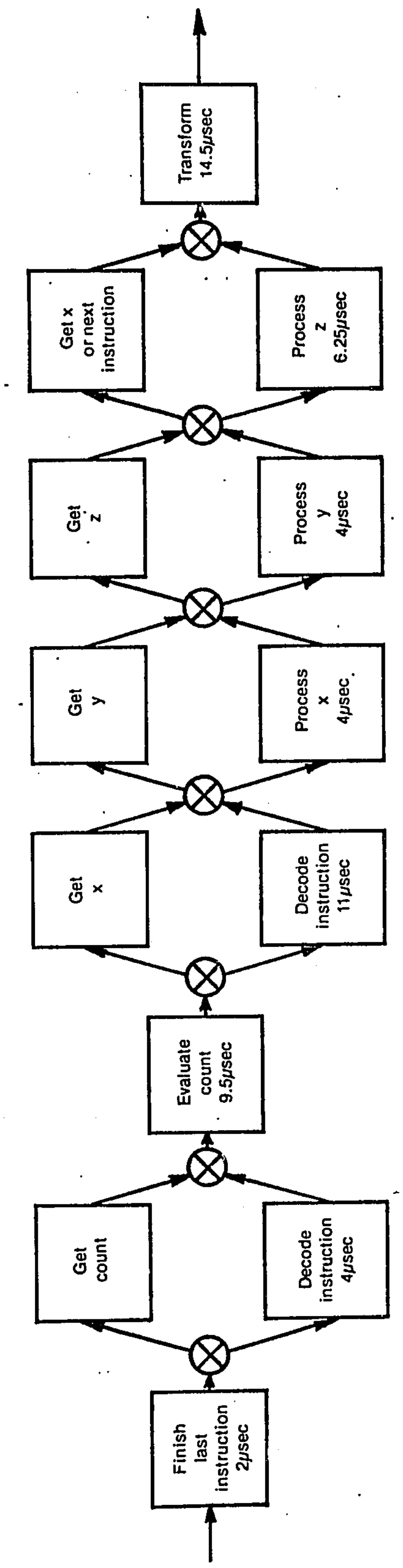


FIGURE 6-1a
Activity Flow for VGMOVE (immediate data)

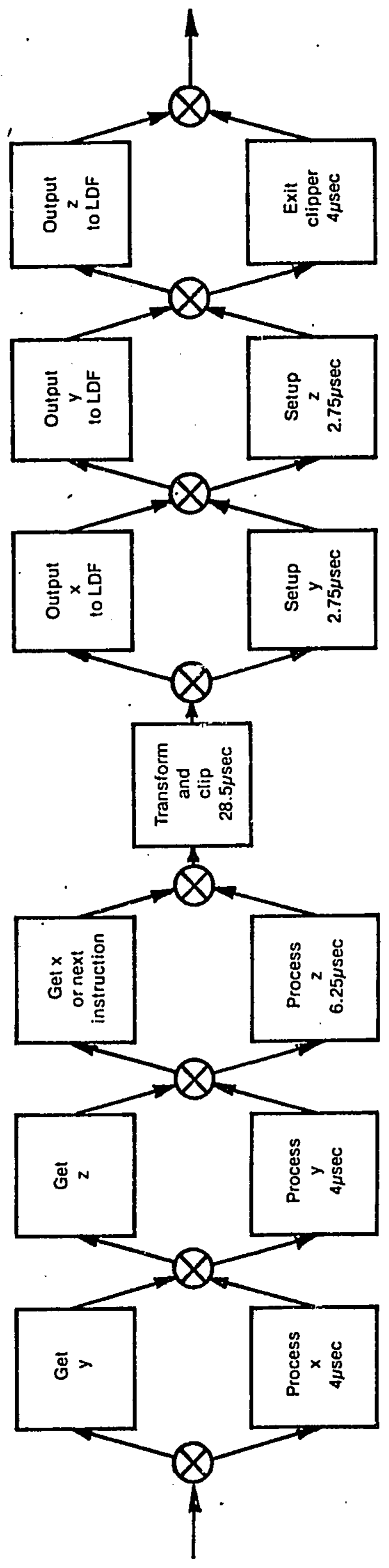


FIGURE 6-1b
Activity Flow for VGDRAW (immediate data)

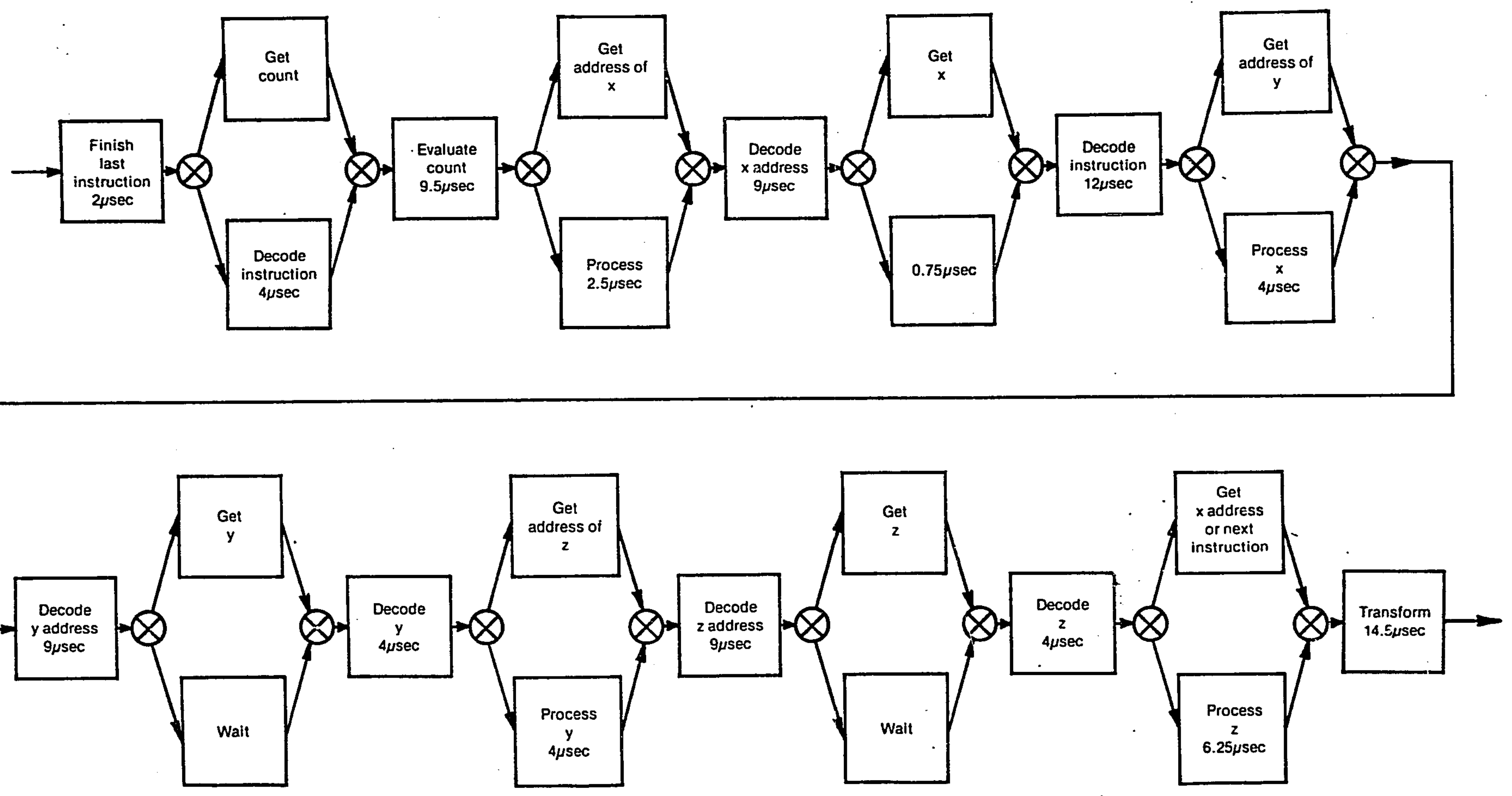


FIGURE 6-2a
Activity Flow for VGMOVE (addressed data)

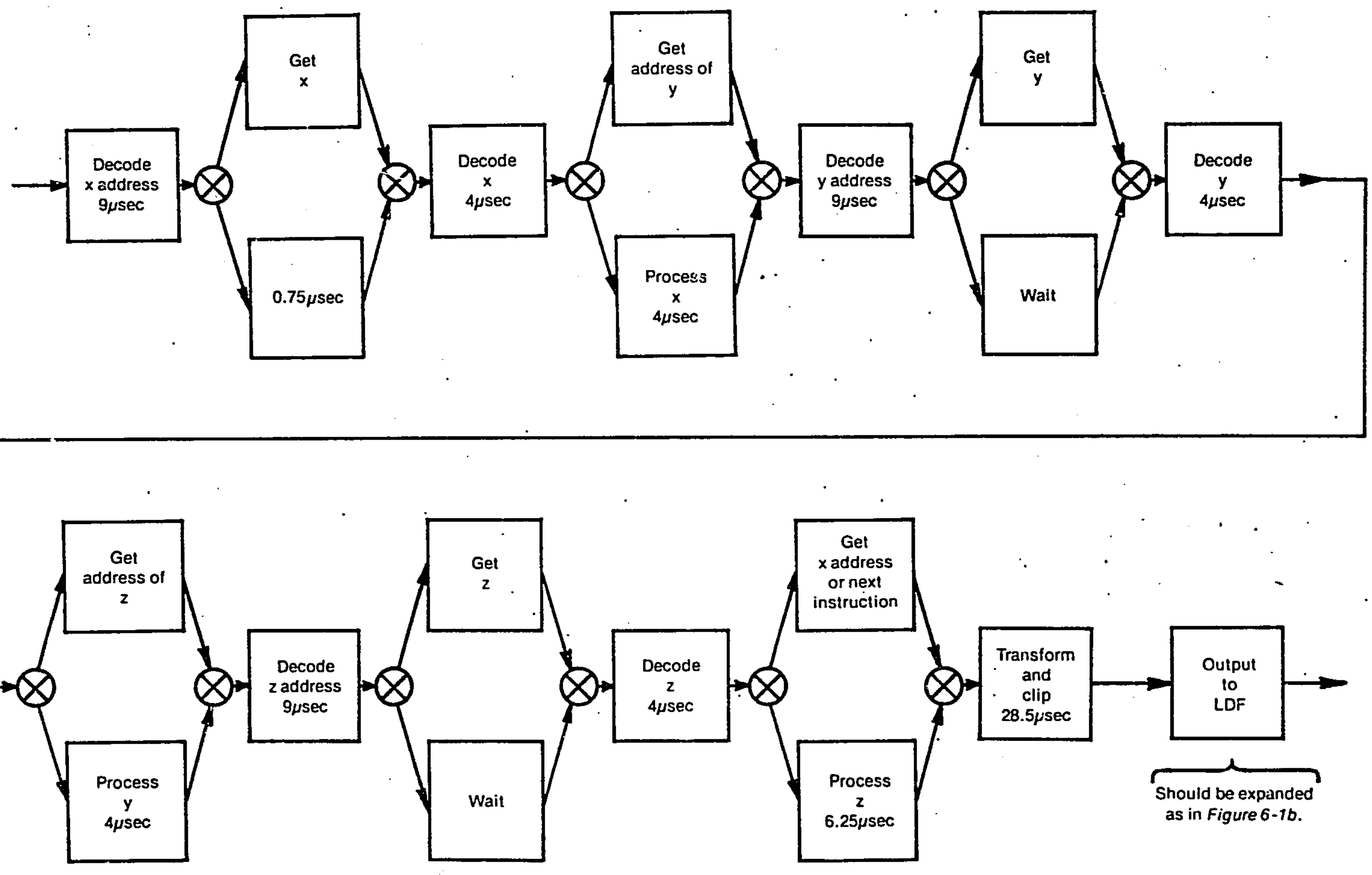


FIGURE 6-2b
Activity Flow for VGDRAW (addressed data)

The following assumptions are made regarding the behavior of a PDP-11/45 and the VG3400:

STAGES 1 and 2

1) The service time for one request by the GPU to the PDP-11 memory is $s_1=2.0$ microseconds [DEC74, DEC79, LEIN79].

2) The average instruction time on the PDP-11/45 is 3.0 microseconds [DEC74], the average number of memory cycles per instruction is 2.5 [BELL78a], and the core memory cycle time is approximately 1.0 microseconds [DEC74]. Hence, $b_1 = 2.5/3. = .83$.

3) The maximum delay at each PDP-11 memory request by the GPU is $d_{max} = 6.0$ microseconds [ARMS79, MOOR79, DEC79].

4) The delay for a request by the GPU to PDP-11 memory is exponentially distributed with a mean of $\bar{d}=2.0$ microseconds [ARMS79, MOOR79, DEC79].

STAGES 2 and 3

5) The minimum transfer time from the GPU to the RBU is $s_1 = 2.5$ microseconds [VECT78c, LEIN79].

6) The maximum delay before a transfer of one word from the GPU to the RBU is d_{max} = .9 microseconds [VECT78c, LEIN79].

Using equation (6.4.1-2), the values above, and the microcode compute times [VECT78d], the following equations for the Q_{ij}'s (expressed in microseconds) can be derived from the activity flows.[20] (The last column contains the number of PDP-11 memory accesses.)

$Q_{2,1}$	$43.3 + 3 \cdot \max(2.0+d, 4.0)$	5
$Q_{2,2}$	$85.3 + 3 \cdot \bar{d} \cdot p + \max(2.0+d, 2.5) + 3 \cdot \max(2.0+d, 4.0)$	8
$Q_{2,3}$	$20.8 + 2 \cdot \max(2.0+d, 4.0)$	3
$Q_{2,4}$	$44.3 + 2 \cdot \max(2.0+d, 4.0)$	3
$Q_{2,5}$	$62.3 + 2 \cdot \max(2.0+d, 4.0)$	3
$Q_{2,6}$	$84.8 + 2 \cdot \max(2.0+d, 4.0)$	3
$Q_{2,7}$	$91.8 + 2 \cdot \max(2.0+d, 4.0)$	3
$Q_{2,8}$	$65.8 + 3 \cdot \bar{d} \cdot p + 2 \cdot \max(2.0+d, 4.0)$	6
$Q_{2,9}$	$89.3 + 3 \cdot \bar{d} \cdot p + 2 \cdot \max(2.0+d, 4.0)$	6

[20]Note there may be a slight delay for transfers from the GPU to the RBU, but these are ignored since the concurrent compute time in the GPU almost always dominates.

$Q_{2,10}$	$107.3 + 3\cdot\bar{d}\cdot p + 2\cdot\max(2.0+d, 4.0)$	6
$Q_{2,11}$	$129.8 + 3\cdot\bar{d}\cdot p + 2\cdot\max(2.0+d, 4.0)$	6
$Q_{2,12}$	$136.8 + 3\cdot\bar{d}\cdot p + 2\cdot\max(2.0+d, 4.0)$	6

Substituting into equation (6.4.1-1) gives the following value for VV_2:

$$VV_2 = 41.0 + 2.1\cdot\max(2.0+d, 4.0) + t\cdot(44.7 + 3\cdot\bar{d}\cdot p + .1\cdot\max(2.0+d, 2.5)).$$

Also, the number of memory references is $3.2 + 3\cdot t$.

To derive the expected value for the polyline composite macro instruction processing time we need the expected value of Y when $s_1 < s < d_{max}$. Using equations (6.4.1-2) and (5.4.2-2) we get

$$\bar{Y} = (1-p)\cdot s + p\cdot(2.0\cdot\exp(-(s-2.0)/2.0) + s).$$

Substituting into the expression for VV_2 we get

$$VV_2 = 49.4 + t\cdot 44.9 + p\cdot(1.6+6.0\cdot t),$$

where $p = b_1/(1-b_2)$. The last term $D = p\cdot(1.6+6.0\cdot t)$ represents the delay. In order to calculate b_2, we recall that the number of memory references for a polyline composite macro instruction is $3.2 + 3\cdot t$ and that the average service time is 2.0 microseconds, which gives

$$b_2 = 2.0\cdot(3.2+3\cdot t)/(49.4+t\cdot 44.3)$$

in absence of any delay in the GPU.

In order to calculate the stretch for the application program in absence of any delay in the GPU, we will assume that the GPU program executes for the entire duration of the application program. Using Theorem 4.1, the stretch is $\rho = 1/(1-b_2)$. When t=0, the stretch factor is 1.15, and t=1 also gives a stretch factor of 1.15.

Using the algorithm in Section 5.4.2 to compute the stretch when there is a delay in the GPU, the stretch factor for the application program becomes $\rho' = 1.14$. The delay for the GPU when there is a delay in the application program is $D = 1.3 + t \cdot 5.0$.[21]

This concludes the derivations of the processing time for a polyline macro instruction on the VG3400. Next we will derive the corresponding expressions for the E&S PS2. In Section 6.5 these equations will be used to contrast the two systems.

The derivations of the macro instruction processing times for the E&S PS2 are much simpler than for the VG3400. This is due both to the fact that the E&S PS2 processors buffer their requests to shared resources and that the resource requests and the processing both at stage II and at stage III of the pipeline are overlapped (see Section 4.3.2.4). The Picture Processor, (stage II, see Figure 3-6) has an input buffer, and the Picture

[21]D can be approximated with a straight line because the stretch does not vary much with t.

Generator (stage III) has both an input and an output buffer. The Data Bus transfer rate and the Picture Memory write rate are such that even under worst case conditions no delays are experienced due to contention for that memory. Futhermore, the I/O is always completely overlapped by processing, except in the CPU at a DMA transfer from the CPU to the Picture Memory, so Q_{2j} and Q_{3j} represent pure processing times.

The Q_{2j}'s and the Q_{3j}'s can therefore be derived from the processing times in [ESCC77, WATK79]. However, the timings for the implementations of the macro instructions in stage I, the Q_{1j}'s, must be derived from PDP-11 assembler code sequences. The PDP-11 assembler code for macro instructions 2, and 8 through 12 can be found in Appendix B together with the instruction timings for a PDP-11/45 [DEC74].

The Q_{ij}'s have the following values (in microseconds).

$Q_{1,1}$	0	$Q_{2,1}$	9.8
$Q_{1,2}$	153.3	$Q_{2,2}$	9.8
$Q_{1,3}$	0	$Q_{2,3}$	9.5
$Q_{1,4}$	0	$Q_{2,4}$	11.9
$O_{1,5}$	0	$Q_{2,5}$	16.4
$Q_{1,6}$	0	$Q_{2,6}$	20.5

$Q_{1,7}$	0	$Q_{2,7}$	24.7
$Q_{1,8}$	153.3	$Q_{2,8}$	9.5
$Q_{1,9}$	153.3	$Q_{2,9}$	11.9
$Q_{1,10}$	153.3	$Q_{2,10}$	16.4
$Q_{1,11}$	153.3	$Q_{2,11}$	20.5
$Q_{1,12}$	153.3	$Q_{2,12}$	24.7

Substituting into equation (6.4.1-1) gives the following values for EV_1 and EV_2:

$$EV_1 = t \cdot 153.3$$

$$EV_2 = 12.4.$$

This concludes the derivations of the processing times for a polyline macro instruction on the E&S PS2. In Section 6.5 we will use these equations to contrast the two systems. In the following section, processing times for the "model" composite macro instruction are derived.

6.4.2 A MODELLING TRANSFORMATION COMPOSITE MACRO INSTRUCTION

A composite macro instruction for a modelling transformation consisting of a translation of an object can be derived from the modelling macro function in Section 6.3.2. Recall that this function has the following format:

model(alter, modify(x,y,z,w), sequence, type).

The processing descriptors for "model" are assigned the following values and weights:

descriptor	value	weight
alter	no	1
modify(x,y,z,w)	(no,no,no)	1-t, $(0 \le t \le 1)$
	(analog, analog, analog)	t
sequence	-	-
type	translate(x,y,z)	1

The resulting translation composite macro instruction represents a set of translation modelling transformations. The weights that are to be varied are associated with "modify"; some portion of the values remain unchanged, and the remainder are modified from an analog input device. Eliminating macro instructions that correspond to processing descriptor values that have been assigned zero weights, we see that only two macro instructions remain for consideration. They will be numbered 1 and 2.

1 model(no, (no, no, no),, translate(x,y,z))

2 model(no, (analog, analog, analog),, translate(x,y,z))

The processing time for the composite macro instruction is the expression

$V_i = (1-t)\cdot Q_{i1} + t\cdot Q_{i2}$. (6.4.2-1)

On the VG3400, the analog values are read by the GPU (stage II) and, as for polylines, VV_1 is zero. We will again assume that in the E&S PS2 there is no overlap between the updating of the SDF from the CPU and the processing of the SDF by the DPU. Therefore, the two functions that will be compared are VV_2 plus any delay in the application program for the VG3400, and EV_1+EV_2 for the E&S PS2. In order to make this comparison, the values for the Q_{ij}'s are calculated first for the VG3400 and then for the E&S PS2.

The Q_{ij}'s for the translate composite modelling instruction are derived in a manner similar to that for the polyline composite instruction; only the results will be presented here.

For the VG3400:

$Q_{2,1}$ $50.3 + \max(2.0+d, 3.0) + 2.0+\bar{d}\cdot p$ 4

$Q_{2,2}$	$96.3 + 3\cdot(2.0+\bar{d}\cdot p)$	4

Substituting into equation (6.4.2-1) gives the following value for VV_2:

$$VV_2 = 55.3 + 47.0\cdot t + D,$$

where D is the delay and is given by

$$D = p\cdot(3.2 + 2.8\cdot t) \text{ microseconds},$$

in absence of any delay in the host. The delay, D, for the GPU when there is a delay in the host is derived as the delay for the line instruction.

$$D = 2.6 + 2.3\cdot t$$

and the stretch in the application program is

$$\rho = 1.16, \text{ if } t=0 \text{ and } \rho=1.08 \text{ if } t=1,$$

assuming both processors are delayed.

For the E&S PS2 (see Appendix B for the PDP-11 assembler code):

$Q_{1,1}$	0	$Q_{2,1}$	31.0
$Q_{1,2}$	262.4	$Q_{2,2}$	31.0

Substituting into equation (6.4.2-1) gives the following values for EV_1 and EV_2:

$EV_1 = 262.4 \cdot t$

$EV_2 = 31.0$

In the next section these equations, together with the corresponding equations for the VG3400, will be used to compare the two systems.

6.5 Results of the Comparison

6.5.1 POLYLINE COMPARISON

In this section the formulas for the polyline composite macro instruction processing times will be used to illustrate some of the differences between the E&S PS2 and the VG3400. Recall from the previous section that the values to be compared are, for the E&S PS2, the sum of the processing times in the two stages, i.e., $EV_1 + EV_2$, and for the VG3400 the sum of the processing time in stage II (VV_2) and the delay in the application program.

As a first step, let us compare the processing times stage-by-stage. In the first stage, the processing time for the E&S PS2 (EV_1) is the time required to modify, as needed, line endpoints in the polyline composite macro instruction. (Note that in the analysis it is important to distinguish between s, which is the fraction of line endpoints that get modified, and t, which is the fraction of line endpoints that are modifiable.) EV_1, which is a function of s, is

$$EV_1(s) = s \cdot 153.3 \text{ microseconds.}$$

The VG3400 does not process any part of polyline in stage I, but there is a cost in the form of a delay in stage I. This delay, VV_1, is calculated in terms of the stretch, ρ, in stage I and the processing time, VV_2, in stage II, according to the formula

$$VV_1 = (\rho - 1) \cdot VV_2.$$

In this case, VV_1, as a function of t, is

$$VV_1(t) = 7.1 + t \cdot 7.0 \text{ microseconds.}$$

In the VG3400, the delay in stage I caused by one modifiable line endpoint is $VV_1(1) = 14.1$ microseconds. On the other hand, the cost of modifying one line endpoint (for the E&S PS2) is $EV_1(1) = 153.3$ microseconds. This means that the cost of modifying one line endpoint in the ES PS2 corresponds to the delay caused by having 10.9 modifiable endpoints in the VG3400.

A similar analysis shows that modifying one endpoint in the E&S PS2 corresponds to the delay caused by 21.6 non-modifiable endpoints in the VG3400.

In the second stage, the processing time for the E&S PS2 is

EV_2 = 12.4 microseconds.

In the VG3400 the corresponding formula

$VV_2(t) = 50.7 + t \cdot 49.9$

shows that, as is expected, the E&S PS2 is much faster regardless of whether a line is modifiable in the VG3400 or not.

To compare the total processing times, P_1, we combine the processing in stages I and II, yielding

$EP_1(s) = EV_1(s) + EV_2 = s \cdot 153.3 + 12.4$, and

$VP_1(t) = VV_1(t) + VV_2(t) = t \cdot 58.9 + 57.8$,

where s and t are as described above, and the subscript 1 on EP and VP are to distinguish the analysis of the polyline composite macro instruction from the analysis of the model composite macro instruction, discussed below. There are two interesting cases to consider, one in which all potentially modifiable line endpoints are indeed modified (s=t) and the other in which all line endpoints are potentially modifiable (t=1).

The first case analyzes the behavior of the two systems in terms of the fraction t of endpoints that are modifiable, assuming all modifiable line endpoints are modified. To determine which system is faster for what values of t, consider

$$EP_1(t) - VP_1(t) = t \cdot 94.4 - 45.4.$$

This formula is positive for t greater than 0.48, indicating that the estimated processing time for the VG3400 is less than that for the E&S PS2 if more than 48% of the line endpoints are modifiable (under the assumption that all modifiable line endpoints are modified), otherwise it is less for the E&S PS2. This is illustrated in Figure 6-3.

The second case analyzes the behavior of the two systems in terms of the fraction s of endpoints that are actually modified, assuming all line endpoints are potentially modifiable (t=1). All line endpoints must be implemented as addressed data points in the VG3400, so that values are fetched from their locations in the application program whether these values are modified or not. In the E&S PS2, <u>only</u> the points that are modified would be updated in the SDF. To determine which system is faster for what values of s, consider

$$EP_1(s) - VP_1(1) = s \cdot 153.3 - 104.3.$$

This formula is positive for s greater than 0.68, indicating that the estimated processing time for the VG3400 is less than that for the E&S PS2 if more than 68% of the modifiable line endpoints are actually modified, otherwise it is less for the E&S PS2. This is also illustrated in Figure 6-3.

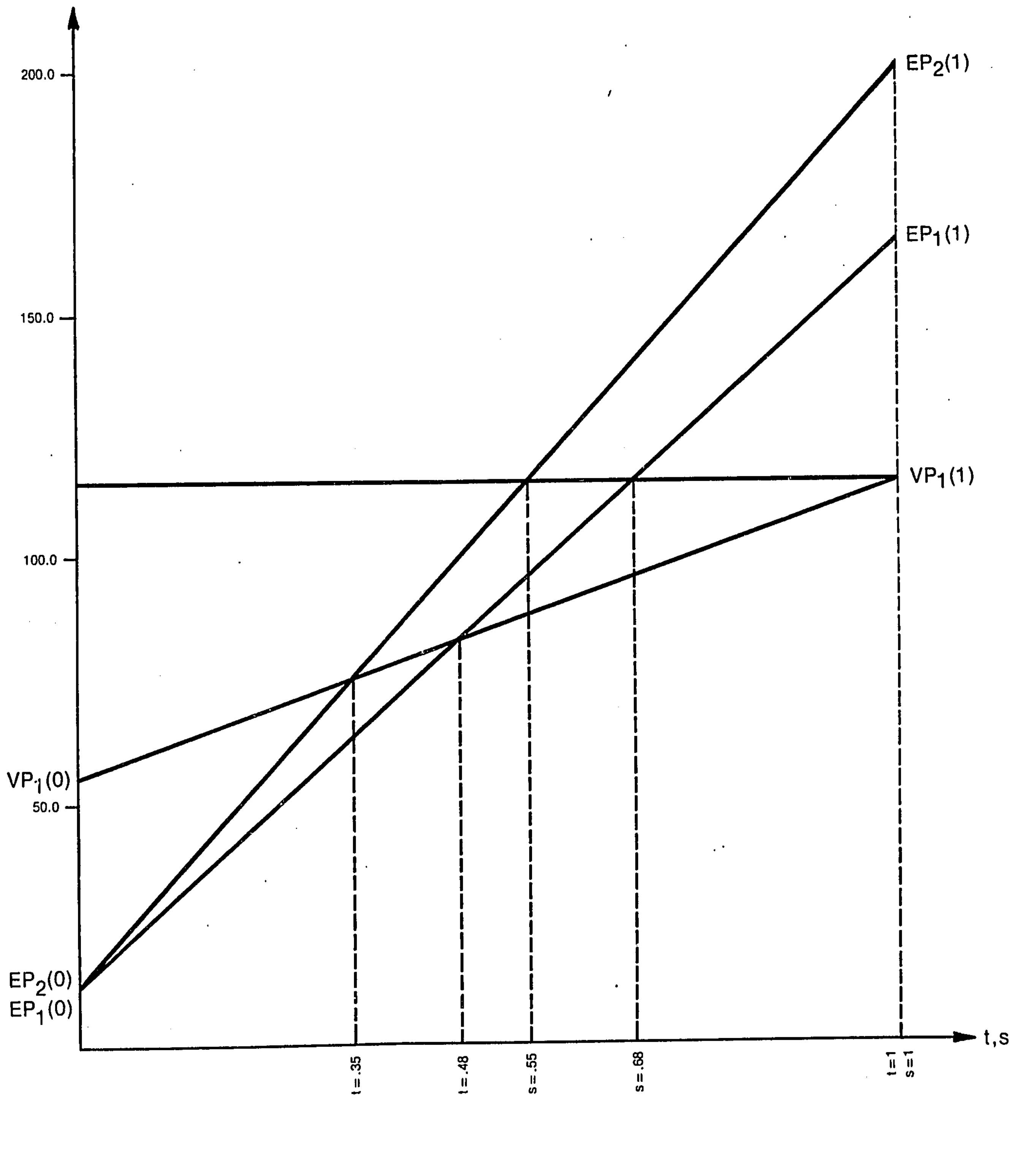


FIGURE 6-3
Comparison of Polyline Composite Macro Instruction on E&S PS2 and on VG3400

The processing times for the polyline composite macro instruction in the two systems do, of course, vary with the parameters of the composite macro instruction. The time to update a line endpoint on the E&S PS2, EV_1, was calculated assuming that the name of the object to be modified was the fifth entry in the directory. If, for example, this is changed to the eighth entry, the corresponding processing time for the composite line macro instruction becomes

$$EP_2(s) = s \cdot 189.5 + 12.4 .$$

Making the same comparisons as above between $EP_2(t)$ and $VP_1(t)$ gives a crossover point of the two curves at t=.35. This is illustrated in Figure 6-3. A comparison of $EP_2(s)$ and $VP_1(1)$ gives a crossover point at s=.55, which is also illustrated in Figure 6-3.

The values of t and s are not as sensitive to variations in the weights, w_i, which determine the portion of the polyline endpoints that are inside, outside, and crossing the clip window boundaries. Assuming that all line endpoints are inside the clip window, i.e., $w_2=1$, we get the following expressions for the processing times in the two systems:

$$EP_3(s) = s \cdot 153.3 + 11.7$$

$$VP_3(t) = t \cdot 56.9 + 61.3 .$$

Again, we make the same comparisons and find that the estimated processing time for the VG3400 is less than that for the E&S PS2 if more than 51% (t=.51) of the line endpoints are modifiable, (and all modifiable line endpoints are indeed modified,) otherwise it is less for the E&S PS2. Similarly, the estimated processing time for the VG3400 is less than that for the E&S PS2 if more than 69% (s=.69) are modified and all line endpoints are potentially modifiable.

The E&S PS2 processing times depend heavily, of course, on the host CPU processing speed. The execution time for the PDP-11 code on an 11/70 is approximately 50% - 60% of that for an 11/45. Thus for an 11/70,

$$EP_1(s) = s \cdot 92.0 + 12.4, \text{ and}$$

$$VP_1(t) = t \cdot 58.9 + 57.8$$

assuming an execution time ratio of 60% and the delay formulas are unchanged). In this case, the estimated processing times indicate that E&S PS2 out-performs the VG3400 for a 10 point polyline even when all line endpoints in a polyline are updated.

6.5.2 MODELLING TRANSFORMATION COMPARISON

In this section the processing time formulas for the translation matrix composite macro instruction will be used to illustrate some of the differences between the E&S PS2 and VG3400 that were not discussed in the preceding section. The analysis will proceed in much the same manner as for the polyline macro instruction.

The comparisons of the individual stages give results similar to those for polyline. In stage II, the E&S PS2 is much faster than the VG3400. In stage I, having 29 translation matrices potentially modifiable in the VG3400 delays the CPU an amount equal to the processing time for updating one translation matrix in the E&S PS2.

The combined processing times for stages I and II for the two systems are

$$EP_4(s) = EV_1(s) + EV_2 = s \cdot 262.4 + 31.0$$

$$VP_4(t) = \rho \cdot VV_2(t) = t \cdot 49.3 + 66.9,$$

where s and t have the same meaning as in the preceding section, and ρ is the stretch in the host.

A comparison of the expressions for the two systems in the case in which all modifiable translation matrices are changed on the basis of values read from some analog input devices (s=t) indicates that if t is greater than 17% then the VG3400 is faster. Recall that the VG3400 approach is to update the translation matrices locally in the DPU, which avoids using the CPU for processing.

A second interesting case for comparing the two systems' behavior with respect to the translation matrix composite macro instruction is when all matrices are potentially modifiable (t=1), and only some fraction (s) of them are actually modified in any given update cycle. In this case, if s is greater than 32%, then the expressions indicate that the VG3400 is faster, otherwise the E&S PS2 is faster.

In general, it is more efficient on the E&S PS2 to read the analog device registers, compare the new values with those read in the previous update cycle, and, only if any of the values changed, update the pertinent transformation(s). A code sequence implementing this for a translation matrix can be found in Appendix B. The processing time for this program can be expressed as

$$EP_5(s,r) = s \cdot 150.3 + r \cdot 128.2 + 31.0$$

where s is the fraction of lines that are actually modified and r is the fraction of lines that are potentially modifiable.

By comparing the estimated processing times for potentially modifiable translation matrices in the two systems, $EP_5(0,t)$ and $VP_4(t)$, we see that the estimated processing time for the VG3400 is less if t is greater than .46, otherwise it is less for the E&S PS2.

6.5.3 SUMMARY

Some of the results reported in the preceding sections are not particularly surprising. In the two extremes of the polyline macro instruction comparison, it was expected that the E&S PS2 would out-perform the VG3400 if no modification takes place and that the VG3400 would out-perform the E&S PS2 if all the line endpoints are being modified. Quantitative cut-off points have, however, been established. Further, it has been illustrated that the speed of the host can have a very significant effect on the outcome of performance comparisons.

One aspect of system performance that has not previously been discussed is the effect of delay in the host from memory requests by the DPU. These delays are a penalty paid for off-loading the update processing from the host. Being able to off-load the host is a great advantage. However, the analysis shows that the delay incurred by memory contention may, for some applications, be comparable to the time taken by update processing by the host in the absence of memory contention.

In another area, input device handling, there has been much debate regarding the advantages and disadvantages of doing the interaction handling in the DPU. Although it seems as if there could be potential time savings by allowing the DPU to access the input devices, the programmer often does not want to use the analog device values directly, but rather use those values as arguments to some function. The value of the function, then is in turn used to update some value(s) in the SDF. Only if the DPU has a general purpose instruction set can such transformation functions be executed in that processor.

At first glance, the analysis of the translation matrix composite macro instruction seems to indicate that there are greater advantages gained from reading dials in the DPU than from using addressed data. However, these figures are somewhat misleading.

An application program must read only the analog devices (dials, etc.) that are enabled, and at an average work station this corresponds to no more than approximately 15 device registers (10 dials, 1 joystick, and 1 tablet). Furthermore, in most application programs, not all devices are enabled at once. Although all enabled devices must be read in each update cycle, in general, no more than two input devices with three device registers each will change in any one cycle (the user has only two hands!). This means that in each cycle only a couple of updates are made. Hence, for most applications the savings in processing time resulting from accessing the input devices from the DPU rather than from the host probably fail to justify the added software and firmware complexity that stems from having to control and support the related features.

There are several other aspects of the two systems that must be investigated before more conclusive comparison can be made. Some of these are:

- Changing individual lines, instead of 10-point polylines.

- The effects of clipping--a further study of how the performance varies with the number of lines that are inside, outside, and crossing the clip window boundaries.

- The effects of the different text processing facilities--the VG3400 DPU supports "medium quality" text (as defined in the Core [GSPC77], whereas the E&S PS2 directly supports only text that is comparable to that referred to as "low quality". In the E&S PS2 medium quality text would have to be largely implemented in the host.

- The effects of off-loading the CPU using the general purpose instructions of the VG3400 DPU.

- The effects of matrix calculations in the DPU rather than the CPU (including sine and cosine functions).

For the results of this performance comparison to become credible, the accuracy of this model should be experimentally verified, and, if this verification shows that the experimental measures differ significantly from the analytical ones, the model should be calibrated until a sufficient degree of accuracy is obtained.

If the systems are available, the model could be verified by simple measurements for a set of loads. Since graphics systems are special purpose processors, it is comparatively simple to derive some representative loads (in contrast with, for example, loads for an interactive operating system which may be very difficult to characterize). Measurements could be obtained by reading the clock in the processors before and after the update and refresh cycles. These measurements are then compared to the U_i's for the corresponding loads. If, on the other hand, the systems are not available, the model can only be verified through simulation (there exist no generally applicable procedures for performance model verification [FERR78]), but, there is of course no guarantee that verification of the analytical model through simulation implies that the analytical model is valid.

If the empirical and the analytical measurements differ significantly, this discrepancy must be found and eliminated, or at least reduced, to obtain a desired degree of accuracy. Inaccuracies in a performance model may be the result of three types of errors [FERR78]:

1) formulation inaccuracies,

2) solution inaccuracies, and

3) parameter inaccuracies.

Formulation inaccuracies are generally the result of modelling at too high a level of abstraction. The model in this chapter is the result of very detailed analysis of the systems' behavior, and the only part of the model where the approximation of the system behavior may lead to non-negligible inaccuracies is the modelling of the PDP-11/45. If the simple stretch analysis were replaced by more detailed modelling, the the analytical measures might become more accurate. Note, however, that the model gives a range of the stretch, and that this range is quite narrow. In the next chapter we will see an example of a graphics system for which this range is much wider, and, for which more accurate modelling of the PDP-11's behavior might be necessary.

Solution inaccuracies may be caused by mistakes in the mathematical derivations or by round-off errors. These inaccuracies are quite simple to check for in a model as simple as the one discussed in this chapter.

Parameter inaccuracies are the result of approximations of the input parameters to the model and of estimates of distributions. Very few approximations have been made for these graphics systems. Rarely have alternate execution paths been averaged, and, in the few cases where this was done, the difference in execution time in the two paths was less than one microsecond.

For the PDP-11/45, on the other hand, the average instruction execution time and the average number of memory references per instruction were used to determine the PDP-11 memory utilization. These estimates were the result of extensive simulations by the Digital Equipment Corporation [DEC74, BELL78a], but these simulations did not model graphics applications in particular; similar simulations of graphics applications may lead to, for our purposes, better approximations of the PDP-11's behavior. (Note that the delay in the model discussed in this chapter is not very significant compared to the processing times.)

Finally, the delay at a memory request to the PDP-11 by the VG3400 is assumed to be exponentially distributed. As was stated above, this can be assumed valid if the ratio of the standard deviation to the mean is in a certain range [BUZE79a], and, for all situations examined in the course of this work, this assumption has indeed been valid. This range has, however, been experimentally derived, and may not hold for graphics applications; experimental studies with graphics applications may give different results. (Note again, the delay does not contribute heavily to the total processing time.)

In the next chapter we will look at a hypothetical graphics system; for this system the performance model could only be verified through simulation.

7 METHODOLOGY FOR HARDWARE AND FIRMWARE DESIGN

7.1 General Methodology of Design

The performance modelling techniques discussed in Chapter 4 can be used to help control the process of designing a graphics system. The techniques can be used to study the effects of particular design trade-offs and to determine what factors have the greatest effect on performance. The performance techniques apply both to the choice of technology and to the design of user instruction sets for each of the processors.

The goal in graphics system design is to define a set of functional capabilities that are appropriate for some market area and to implement these functional capabilities in a well-balanced[22] system that maximizes performance, subject to the cost constraints in the market area. The performance modelling techniques can be used as an aid in choosing the hardware components such that performance is maximized and such that the components form a well-balanced system. The performance modelling techniques can also be used as an aid in deciding what processing ability each processor should have in order to maximize performance, and in deciding what functional distribution is

---

[22]A system is well-balanced if no one component is the principal bottleneck and no one component is significantly underutilized.

desirable among the processors in order to achive a well-balanced system.

The design of a graphics system is an iterative process: propose a design, study the performance for some "typical" loads, modify the design, study the performance of the modified system, and so on. One approach to the design process has the following steps:

1) Choose a market area, i.e., define the application area(s) and the cost constraints.

2) Choose a set of functional capabilities that seem suitable for the application area(s) and express these as macro instructions.

3) Design a set of user instructions that are capable of implementing the macro instructions and select specific implementations of the macro instructions in terms of those user instructions.

4) Choose a set of hardware components that are within limits of the cost constraints in the market area(s).

5) Apply the performance model and study the effects of the hardware parameters and of the user instructions on performance.

6) Refine the user instructions and modify the choice of hardware in such a way that

   a) if there is a bottleneck in one of the processors, either upgrade the hardware, simplify the user instructions, or migrate (part of) the implementation of one or more macro instructions to another processor;

   b) if any processor is used much below its capacity, either add features to that component, decrease its hardware capabilities, or migrate (part of) one or more macro instruction implementations to it from another processor.

7) Judge the cost effectiveness[23], and, if the system does not meet the cost requirements, reduce the hardware components and return to point 5.

In the two subsections below we shall see how the performance model can be used to examine the effects of some design parameters. In the first subsection the effects of the data structure access parameters (i.e., combined bus and memory access rates) are discussed. In the second subsection some user instruction design trade-offs are studied; the effects of

[23]Cost and marketability are two important parameters that enter into many of the design decisions. The effects of these parameters on design decisions are, however, beyond the scope of this thesis.

combining versatile addressing modes with the speed of parallel transformation hardware are studied.

In Chapter 8, recommendations for some user instructions that serve to maximize performance for applications that have highly dynamic pictures are made, and software constructs that support these user instructions are proposed.

7.2 Effects of I/O Hardware Parameters on Performance

As part of the design process, it is necessary to be able to determine the effect of the hardware I/O parameters on performance. This need arises both in the choice of a CPU and in the design of the remaining components of the graphics system. If one stage in the processing pipeline is the bottleneck, this problem may be reduced by increasing the hardware I/O rates. Similarly, if one of the processors is used much below its capacity, the associated I/O hardware rates may be reduced without a decrease in the overall performance.

The effects of the I/O hardware parameter(s) can only be studied in the context of a particular load. In this section it will be assumed that the expressions for the processing times of a load, i.e., the U's for each stage of the pipeline, have been derived. Given these U's, the effects of modifications of the I/O hardware parameters on performance will be illustrated.

The processing time at each stage of the pipeline is a function of the time to read one word from a data structure, and the time to write one word of the result of the processing to another data structure. The read time and write time are the bus transfer time, combined with the memory read and write time, respectively. The processing time can be expressed as a function of these read and write times: U(r,w).

Assuming that w is constant, U(r,w) can be expressed as a constant plus a sum of a set of terms having the form

$$V_k(r) = \max(n_k \cdot r + d_k + t_k\ ,\ s_k),$$

where n_k is the number of words read, t_k and s_k represent processing times independent of r (but not necessarily independent of w), and d_k is the delay associated with the resource request(s) in V_k.

For a first approximation, d_k can be assumed to be a linear function of r (either a constant or a constant plus a factor times r). This assumption is reasonable because the delay is a function of the reads and writes to the memory by the contending processor. Under this assumption, one observes that V_k is a piecewise linear function of r. U, being the sum of various such terms, is also a piecewise linear function. Similar arguments hold for w, if r is held fixed. Thus, $U(r,w)$ is, in a sense, "piecewise planar".

By choosing, in a pessimistic fashion, the linear functions of r and w that express the d_k's (so that the maximum possible delay is estimated), an upper bound for $U(r,w)$ is obtained. To avoid the assumption that d_k is linear in r and w, one can perform a more detailed analysis, using equation (5.4.2-3) to express the value of each term $V_k(r)$. The function describing U derived thereby is not piecewise linear, but piecewise differentiable.

If, after the initial choice of hardware parameters, it is established that the i:th stage of the pipeline is the bottleneck, one wants to reduce U for this stage by reducing either the read time or the write time, but not both unless necessary. Let r_1 and w_1 denote the current choices for the read and write times. The variable that is the prime candidate for reduction, read time or write time, is the variable whose partial derivative of $U(r,w)$ evaluated at (r_1,w_1) is the larger. Using superscripts to denote partial derivatives, we write the two values to be compared as

$$U^r(r_1,w_1) \text{ and } U^w(r_1,w_1)$$

If the derivative with respect to r is greater than the derivative with respect to w at (r_1,w_1), then the read time should be the prime candidate for improvement, since a reduction in r will result in a larger performance improvement than a comparable reduction in w.

If the cost of reducing U(r,w) to an acceptable value by reducing the value of r is prohibitive, the possibility of a reduction of w should be explored. If the processing time for the i:th stage cannot be reduced sufficiently by reducing the data structure read and write times, either basic processor speed improvements should be considered, or some functions from this stage should be migrated to another stage or eliminated alltogether.

If, after the choice of hardware parameters, the i:th processor is used much below its capacity, then an increase in the read or write times could possibly be made without loss in overall performance. However, since the read time and write time for a data structure are related[24], care must be taken that the increase in read and write times do not affect the adjacent stages to such an extent that the relevant processing times violate design goals. If it is not possible to increase the read or write times for the i:th stage without creating a bottleneck in another processor, the possibility of migrating some functions to the i:th stage should be examined.

[24]The bus transfer times are equal and the memory read and write times are either equal or, for core memory, read time is some multiple of write time.

7.3 User Instruction Set Design

7.3.1 LINE DATA ADDRESSING MODES

In Chapter 6, some of the differences between two graphics systems were illustrated. The comparison demonstrated that a versatile, general purpose addressing scheme in the DPU can give better performance for some applications, but that the cost of a complex addressing scheme is very high for many other applications. Addressed data is, however, very convenient; if only immediate data is provided by the system, the burden of continuous updating of coordinate data in the SDF is placed entirely on the programmer.

In this section, the effects of combining the versatility of addressed data with the speed of parallel transformation hardware is illustrated. It will be shown that by restricting the types of addressing modes, the performance can be increased significantly. This performance increase comes about for three reasons: the simpler addressing modes permit simpler address decoding, the simpler addressing modes permit coordinate data to be fetched as triplets instead of one by one, and finally, and most importantly, the proposed addressing mode makes it possible to take full advantage of the speed of parallel transformation hardware. In what follows, it will first be shown that the parallel transformation hardware can be used with immediate data to improve performance; next the arguments for immediate data will be extended to cover addressed data.

Let us postulate a graphics system, which we will call X, attached to a PDP-11/45 with the SDF located in the PDP-11 memory. Assume further, that the X graphics system has parallel transformation hardware. One user instruction in the DPU is a line instruction with immediate 3D coordinate data, and with the number of line endpoints specified as immediate data. A simplified activity flow[25] for a move macro instruction implemented using this user instruction could have the format shown in Figure 7-1a, and the activity flow for a draw macro instruction implemented using this user instruction could have the format shown in Figure 7-1b.

By comparing these activity flows with the activity flows for the VG3400 move and draw macro instructions in Chapter 6, one sees that relativly little time has been gained. The instruction decoding is shorter, and the parallel transformation hardware will give some performance improvement. However, in the VG3400, part of the coordinate transformation is done in parallel with

[25]The output to the linear buffer is not treated in detail.

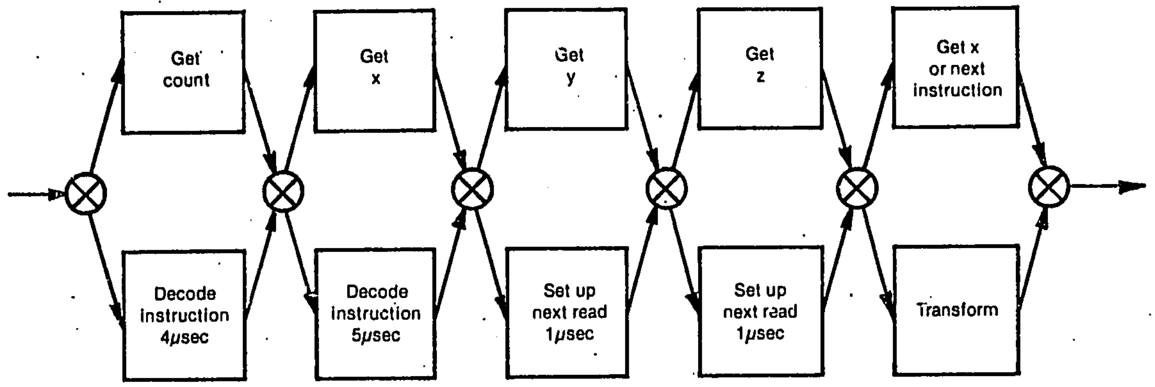


FIGURE 7-1a
Activity Flow for MOVE

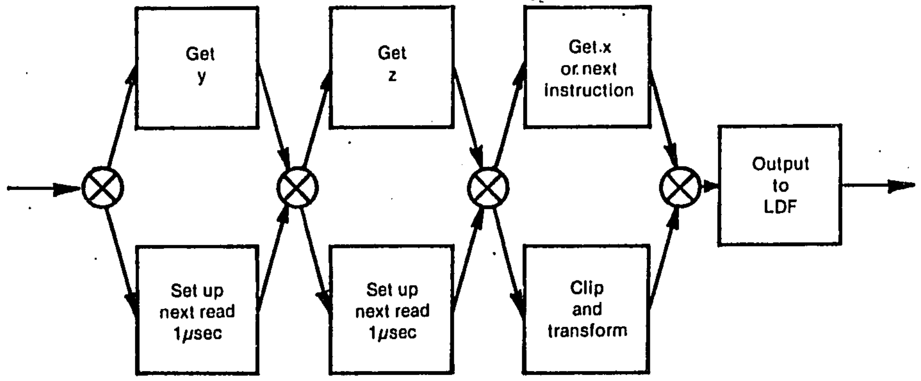


FIGURE 7-1b
Activity Flow for DRAW

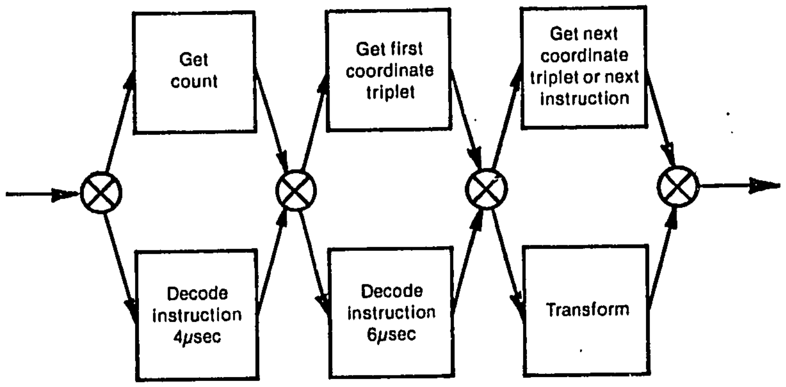

Get count
Decode instruction 4μsec
Get first coordinate triplet
Decode instruction 6μsec
Get next coordinate triplet or next instruction
Transform

FIGURE 7-2a
Activity Flow for MOVE (immediate data)

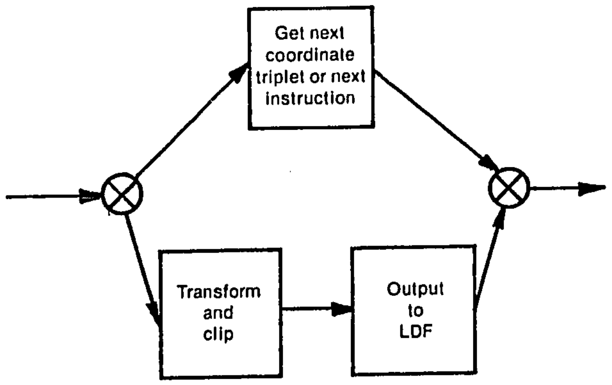


FIGURE 7-2b
Activity Flow for DRAW (immediate data)

the coordinate fetches, so the performance of the proposed system is not significantly better than that of the VG3400.

By fetching coordinate data triplet by triplet, and by overlapping the memory fetches and the processing, the activity flow for a move and a draw macro instruction with immediate data could have the formats illustrated in Figure 7-2a and Figure 7-2b, respectively[26]. As is seen in these activity flows, there is now complete overlap between the processing of a line endpoint and the coordinate fetches, with the exception of the first line endpoint. These activity flows will be used later in this chapter to develop timings for the polyline composite macro instruction that was discussed in Chapter 6, implemented on system X. Before we can study the performance of system X as the performance was studied for the VG3400 and the E&S PS2 in the previous chapter, we must develop a line user instruction using addressed data.

Assume that another user instruction in the DPU of the X graphics system is a 3D line instruction with addressed data; the number of line endpoints is specified as immediate data. For reasons similar to those for the line user instruction with immediate data, simply performing transformations in parallel with the fetch of the next address (or instruction) in the list of coordinate pointers is going to result in an instruction

[26]A processing descriptor "last" should be added to the move and the draw functions to distinguish the last endpoint in the polyline; no coordinate data, only the next instruction, is fetched during the processing of the last endpoint.

execution time comparable to that for the VG3400. However, if we assume that the components of each (x,y,z) triplet are located in adjacent locations, we need use only one pointer to the triplet, which cuts down the total number of words fetched. Futhermore, this allows the coordinate data for one triplet to be fetched from memory by one DMA block transfer rather than by three individual transfers, thereby reducing the total delay. (For most applications, keeping the (x,y,z) triplet in adjacent memory locations is not a severe restriction.) Simplified activity flows for move and draw macro functions implemented this way are illustrated in Figure 7-3a and Figure 7-3b, respectively.

These activity flows illustrate what is taking place conceptually; between the fetch of the next address and the next coordinate triplet a few instructions must be executed to start the second read. This can be done in one of two ways. Either each of the parallel paths are implemented in two different processors and the input processor issues the next read, or the main processor is interrupted at the conclusion of the first memory fetch and initiates a new read before continuing processing.

One concern regarding these user instructions is that the stretch in the host will be increased by fetching coordinate triplets instead of fetching each coordinate value separately. The increase due to the longer memory fetches is, however, not

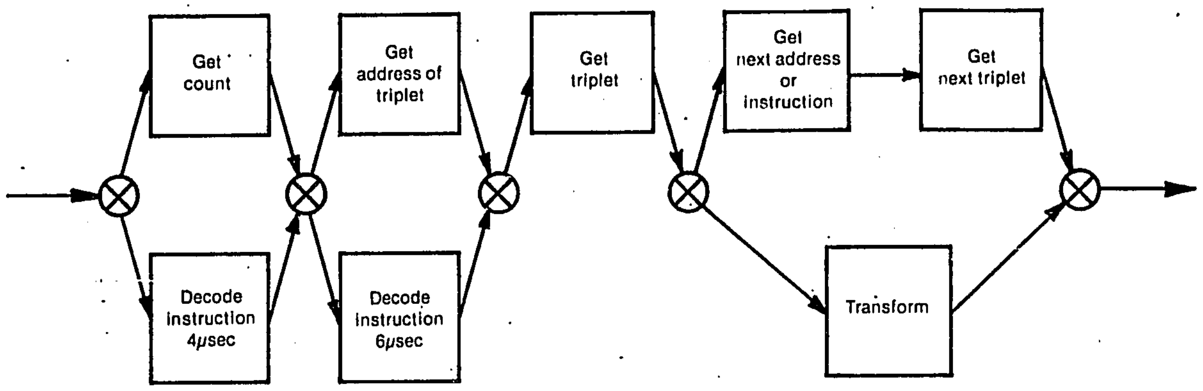


FIGURE 7-3a
Activity Flow for MOVE (addressed data)

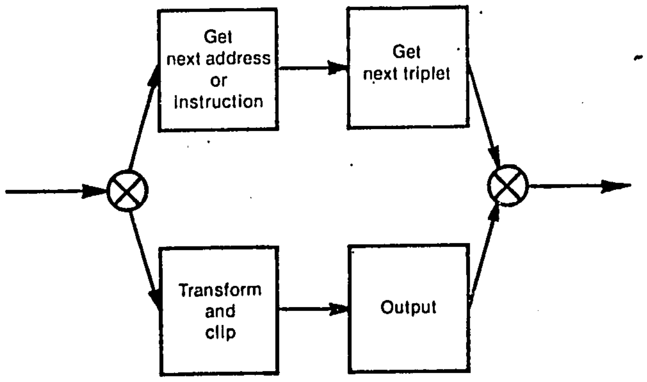


FIGURE 7-3b
Activity Flow for DRAW (addressed data)

significant, as will now be shown using the formula for expected stretch that was derived in Theorem 4.4.

Assuming that α_1 and β_1 are the completion rate for the processing in the host and the completion rate for memory fetches by the host, respectively, and that α_2 and β_2 denote the corresponding rates for the DPU for single word memory requests, and further making the somewhat pessimistic assumption that all memory requests are in form of triplets, the stretch can be expressed as follows:

$$\rho_1 = \frac{1}{1-b_2} - \frac{\alpha_2}{(3\alpha_1+\alpha_2+\beta_2)} \cdot \frac{(1-b_1)}{(1-b_2)} .$$

If b_1 is large, as is the case for the 11/45, the stretch is close to the maximum stretch and fetching triplets is not going to have any significant effect on stretch in the host. If, on the other hand, b_1 is small, as is the case for the 11/70, then α_1 is also small and the increase in stretch will not be very significant. As will be seen below, the user instructions in the DPU of system X will cause a large delay in the host for other reasons.

Now we are ready to develop timings for the polyline composite macro instruction discussed in the previous chapter. This is done in the same fashion as in Chapter 6; only the results will be reported here. It is assumed that system x's transformation hardware has approximately the speed of the E&S PS2 transformation hardware, and that the speed of the remainder of the system is comparable to the speed of the VG3400.

Using the same notation as in Chapter 6, XV_1 and XV_2 will denote processing delay in stage I and processing time in stage II in X, our hypothetical graphics system, and

$$XV_1(t) = 9.5 + t \cdot 4.6$$

$$XV_2(t) = 19.4 + t \cdot 1.4$$

where t is the fraction of endpoints that are potentially modifiable[27].

The first thing to notice is that the delay in the host is quite significant; in the case all endpoints are specified as addressed data, (t=1), the stretch factor is in the range 1.39 to 1.68.

As a first step, let us compare the processing times stage-by-stage to those for the E&S PS2. After that, the total processing times are compared. In the first stage, the processing time required to modify one line endpoint in the E&S PS2 is

$$EV_1(s) = s \cdot 153.3 \text{ microseconds,}$$

[27]Note that these functions are approximated by straight lines.

where s is the fraction of lines that are actually modified. The cost of modifying one line endpoint in the E&S PS2 corresponds to a delay in the host caused by our graphics system having 10.9 modifiable endpoints.

In the second stage, the processing time in the X graphics system of the polyline composite macro instruction is, as expected, larger than the processing time EV_2 = 12.4 microseconds in the E&S PS2.

To compare the total processing times P_1 we add the equations for the two stages which gives the equation

$$XP_1(t) = 28.9 + t \cdot 6.0$$

for the X system. This equation represents the processing time of a polyline with some portion (t) of the line endpoints potentially modifiable. The corresponding function for the E&S PS2 is

$$EP_1(s) = 12.4 + s \cdot 153.3.$$

Let us consider the two cases that were discussed in the previous chapter. In the first case, all potentially modifiable line endpoints are indeed modified (s=t), and, in the second case, all line endpoints are potentially modifiable (t=1).

In the first case we consider

$EP_1(t) - XP_1(t) = t \cdot 147.3 - 16.5.$

This formula is positive for t greater than 0.11, indicating that the X graphics system is faster if more than 11% of the line endpoints are modifiable (under the assumption that all modifiable line endpoints are modified), otherwise the E&S PS2 is faster. This is illustrated in Figure 7-4.

In the second case we consider

$EP_1(s) - XP_1(1) = s \cdot 153.3 - 22.5.$

This formula is positive for s greater than 0.15, indicating that the X system is faster if more than 15% of the line endpoints are actually modified; otherwise the E&S PS2 is faster. This is also illustrated in Figure 7-4.

Since system X is assumed to transform and clip lines at the approximate speed of the E&S PS2, the values for t and s do not vary much with the weights, w_i, which determine the fraction of lines that are inside, outside, and crossing the clip window boundaries. The values of t and s do, however, vary with the number of items in the E&S PS2 directory that have to be searched. The processing time for modification of the object whose name is in the eighth entry in the directory, $EP_2(s)$, is compared to $XP_1(t)$ and $XP_1(1)$ in Figure 7-4.

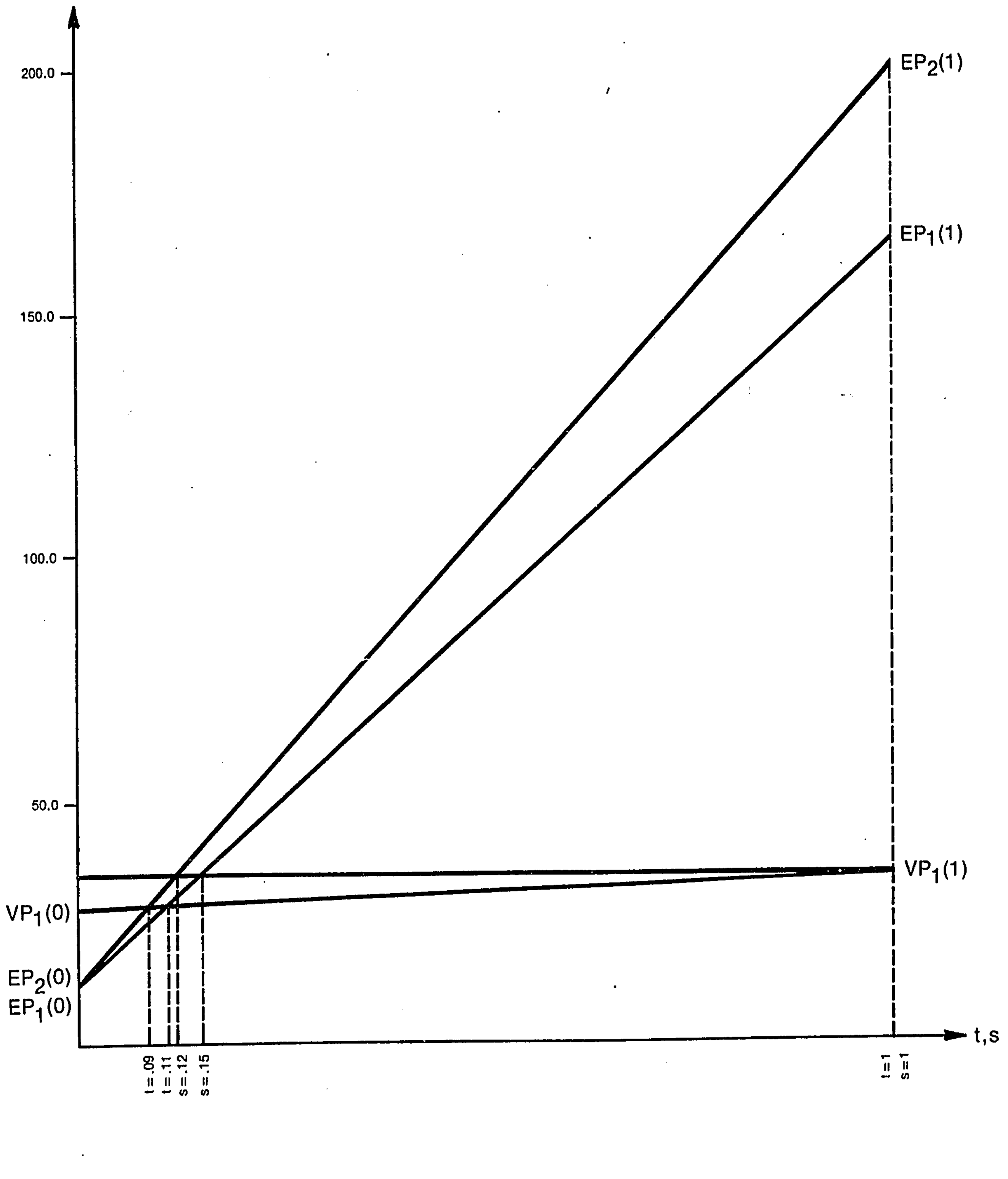


FIGURE 7-4
Comparison of Polyline Composite Macro Instruction
on E&S PS2 and on System X

7.3.2 SUMMARY

Our analysis has shown that the performance of the polyline macro instruction in the hypothetical graphics system is significantly better than the performance of the same instruction on the VG3400. The hypothetical system also seems to perform better than the E&S PS2 for a certain, large class of applications. The performance of the proposed graphics system could be even further improved by pipelining the processing in the DPU, such that the input, the transformations, and the output are all completely overlapped, as is done in the E&S PS2.

The significantly increased throughput in the graphics system does, however, result in a large delay in the host. This problem could be reduced in two ways: by decreasing the host memory utilization by increasing the memory rate or adding a cache, and by storing the SDF in a separate memory. In the latter case, only the actual coordinate data for addressed data would be fetched from the host memory; all other data would be fetched from the added memory, which would reduce the delay quite significantly.

8 GRAPHICS SOFTWARE

8.1 Introduction

Much emphasis on software design today is focused on language features and on qualities of the language such as readability, ease of learning, and the like.

"...running computer programs and measuring execution time and storage consumption would tell less than half the story. For most programming projects, qualitative aspects of the language are far more important than quantitative ones." [BLOO75]

Although qualitative aspects are important in real-time graphics software, execution speed plays a relatively much larger role than in other language design. Graphics software must give a programmer access to all hardware/firmware features, and he must be able to access all these features from the software without significant loss of efficiency due to software overhead.

Recall from Chapter 3 that there are two levels of software used for graphics applications. First, there is the applications program that builds and interacts with the applications data structure (ADS). Second, there is the display file compiler (DFC) that scans the ADS and builds and interacts with the structured display file (SDF). The DFC has two parts: the application dependent part which scans the ADS and the SDF access method, that is the routines that are used to actually build and interact with the SDF. In this chapter an access method for a high-performance graphics system will be discussed.

The result of the design process that was discussed in the beginning of Chapter 7 is a set of user instructions in each of the three stages of the pipeline that implement the macro instructions. By definition, an access method gives control over all user instructions in stages II and III. The access method provides <u>direct</u> control over all user instructions in stage II and <u>indirect</u> control over the user instructions in stage III through stage II. (Stage III will be ignored in what follows.) In addition, the access method should support some commonly occuring user instructions in stage I. One example of this is routines to set up transformation matrices.

In this chapter an access method, Theseus, for a high-performance graphics system is described. The features in the DPU of this graphics system are based on the findings in Chapter 6 and 7; various important features of the graphics system are listed below. The supported features are:

- Object hierarchy - support for objects, object directory, and object call stack; transformations, attributes, and names can be stacked.
- Segments - one level of segmentation within objects.
- Transformations - a 4x3 transformation matrix; clipping to a pyramid of vision (for perspective) centered around the z-axis and with the center of projection at the origin and a 90 degree apex angle; clipping to a parallelepiped centered around the z-axis. (The matrix transformations allow specification of an arbitrary viewing transformation.)
- Primitives - line primitives both as immediate and addressed data; text primitives.
- Attributes - similar to segment and primitive attributes in the Core Graphics System.
- Interaction handling - support for reading and writing of all standard interaction devices from the host; access to the object call stack from the host.
- Extents - a feature that can reduce processing time in a hierarchical system.

8.2 Theseus - A Graphics System Access Method

Much of the emphasis in this discussion of Theseus is placed on those graphics features that are the focus of this thesis: features for dynamics and interaction handling. Theseus allows access to applications data from the DPU by addressed data types in the SDF, and it allows access from the DFC to the data in the SDF through dynamos. Theseus allows access to the interaction devices only from the CPU; all updating of the display data from interactive devices takes place from the CPU either actively, through dynamos, or passively, through addressed data.

In the first section below an overview of Theseus is given together with a brief discussion of its general capabilities. In the following sections, features that are new to graphics software design are discussed in more detail; the reader is referred to Appendix C for a complete list of all functions in Theseus, together with a discussion of each function.

8.2.1 AN OVERVIEW OF THESEUS

Theseus is an access method for a high-performance vector graphics system implemented as a set of procedures callable from Algol W [SORG77]. Theseus allows a programmer to build and manipulate a hierarchical graphical data structure, i.e., a SDF. This data structure is analogous to a program structure consisting of procedures which may call other procedures. A graphical data structure consists of objects which may reference, i.e., call, other objects.

An object consists of one or more segments. A segment may be added to or deleted from an object. A segment has a set of associated attributes. A segment may contain output primitives, such as lines and text, object calls, and attributes and transformations that apply to these object calls. The modelling transformations (object construction transformations), are specified as part of object calls and allow objects to be defined in local coordinate systems and to be combined into new objects.

An object is displayed by creating a view of that object. This view has associated attributes and viewing transformations that determine how the object or a portion of the object is mapped to the display surface. When the graphical data structure is processed by the DPU, the viewing and modelling transformations are composed and applied to the local coordinate data of the objects. Similarly, the DPU composes the view, the object call, and the segment attributes before applying them to the output primitives.

The elements of the graphical data structure may be manipulated interactively. Theseus provides support for all common interaction devices.

Algol W was chosen as a source language because it is a general purpose programming language which allows procedures to be passed as parameters. Transformations, attributes, and extents (see Section 8.2.3) are specified as procedure parameters of routines that define objects, segments, views, and object calls. This has the advantage that the scope rules are self-evident: attributes and transformations apply only to the features whose definition they are part of. This syntax has the further advantage that it allows better error checking; any inconsistencies in the transformations or attributes will be identified at the time of definition, not at the time when the transformations and attributes are applied. Software systems that specify viewing parameters and attributes modally cannot perform some of the requisite checking when the values are specified, only later, when the values are applied.

8.2.1.1 OBJECTS AND SEGMENTS

An object is defined hierarchically; each object may reference, or call, other objects. At an object call, the environment, e.g., transformations, attributes, and names for identification (see Section 8.2.2) of the current object, is saved, the transformation matrix of the call is composed with the current transformation matrix, and the attributes of the call are composed with the current attributes. Upon the return from an object the environment of the calling object is restored. Instances of an object are easily added to and deleted from the data structure, but induce significant overhead in DPU processing time.

Theseus allows the programmer to create and delete objects, to modify objects by adding and deleting segments, and to create object calls. A dynamo (see Section 8.2.2) may be assigned to the object call as a whole, which allows replacement of the associated attributes and modelling transformations. Individual attributes may also be updated through the use of this dynamo. By assigning dynamos to individual modelling functions, each individual transformation may be modified.

An object may have an associated extent (see Section 8.2.3), and can be conditionally executed depending on the size of this extent. The body of the object consists of the extent size, and all segments in the object.

The creation of an object can be double buffered i.e., a new version of the object can be created in the CPU, while the previous version is executed by the DPU. The double buffering is implemented by a routine that swaps the bodies of the new object for the old. This method of updating an object is preferable from a performance point of view both to using dynamos and to using addressed data when a large portion of the object is to be modified. This is because double buffering allows the creation of the new object to be overlapped with the DPU processing of the old version of the object.

An object consists of one or more segments that are made up of primitives and object calls. A segment may be added to or deleted from the open object, may have associated attributes, and may be conditionally executed depending on the extent size of the object. Primitives cannot be added to a segment once it has been closed, but its attributes, its primitives and object calls may be modified through the use of dynamos, and primitives defined as addressed data can be modified by modifying their values in the application program.

Segments are cheaper in DPU processing time than are objects in that they are not referenced through a call and hence do not require as much of a change in environment that is required for an object call. Segments, however, do not offer the same flexibility: they may not be referenced by other objects, and may not have associated extents.

8.2.1.2 MODELLING TRANSFORMATIONS

Theseus provides both modelling and viewing transformations. Theseus maintains a logical separation between the two types of transformations, but, in order to utilize the transformation hardware/firmware, both modelling and viewing are part of Theseus.

The modelling transformations include routines for scaling, translation and rotation of an object in its local coordinate system, and can also be specified as any arbitrary 4x3 or 3x2 matrix. Each modelling transformation function invocation may optionally have an associated dynamo. If dynamos are not specified with the transformation functions, the corresponding matrices will be composed in the CPU before being added to the SDF. If, however, a modelling transformation function invocation has an associated dynamo, the corresponding matrix is added to the SDF and the composition takes place each time the SDF is executed. This allows continuous update of the transformation matrix, and the composition of the matrices takes place in the DPU, where this operation is significantly faster than in the CPU.

8.2.1.3 PRIMITIVES

Theseus provides line, point and text primitives. The primitives are added to the open segment, and are specified in absolute coordinates of a coordinate system local to the object. The coordinate data added to the segment consists either of immediate data, i.e., the actual coordinate values, or of addressed data, i.e., a pointer to the coordinate data. Each primitive may have an associated element name for identification purposes, and a primitive specified as immediate data may have an associated dynamo name for modification purposes.

The text primitives are part of two- or three-dimensional objects. Characters in the text strings are individually positioned, conforming to the programmer's specification as closely as the hardware character generator permits.

Individual primitives may not be deleted from a segment. They may, however, be altered either through an associated dynamo, or by a change in the coordinate location pointed to by an address in the SDF. The type of the primitive may not be changed, but a point and an endpoint of a line may be replaced by another point, and a text string with another text string. Note that the number of items in a primitive, i.e., the number of lines or the number of characters, must remain unchanged.

8.2.1.4 INTERACTION HANDLING

Theseus supports five classes of logical interaction devices:

PICK - identify object, segment, or primitive

BUTTON - select function

KEYBOARD - provide alphanumeric information

LOCATOR - provide coordinate information

VALUATOR - provide values.

These logical devices may be implemented by one or more physical devices, and the programmer may choose which physical interaction device most conveniently implements a logical device for his application.

The logical devices are divided into two classes: event causing devices and sampling devices. Each event causing device has an associated one-element event queue. Upon an event, an event report with data related to the event is placed on this queue. The event report must be removed from the queue by the application program before a new event report may be placed on the queue. The application program may poll any queue, or wait for an event from one or more devices. Sampling devices have values that may be sampled by the application program. The pick, button, and keyboard are event causing devices, the valuator and the locator are sampling devices.

The input handling in Theseus is similar to that of the Core [GSPC77, GSPC79]. The differences are that the logical to physical device mapping is explicit in Theseus, that each input device has only a one element queue, and that associations are not supported.

8.2.2 NAMING

As was seen in Chapter 7, it is important that modifyable primitives can be specified either as immediate or as addressed data. Line primitives, to be modified, should be specified as immediate data when it is not convenient to keep the coordinates for line endpoints in three adjacent locations in the ADS, or when a large number of line endpoints are potentially modifyable, but only very few are indeed modified in each update cycle.

Theseus provides a pointer mechanism that allows the host program to modify immediate data. These pointers in Theseus are called dynamos. Dynamos may be assigned to items in the SDF at the creation of these items, or may be assigned by the operator as he identifies an item on the display screen by a pick. This means that in addition to dynamos that are used for modification purposes, a set of names for identification purposes must be provided. The different kinds of names in Theseus are discussed below.

Names are used to identify parts of the data structure, i.e., to establish which item in the data structure was pointed at by a pick device. Names are also used to modify parts of the graphical data structure, i.e., to add, to delete, and to alter parts of the data structure. Names may be associated with objects, segments, views, object calls, primitives, attributes, and transformation matrices.

An _object name_ is used for the purpose of identification and modification of the object. It is defined when the object is defined or declared. An object name must be unique within the display data structure.

A _view name_ is used for identification and deletion of a view of an object. A view name is defined when a view is added to the display data structure, and must be unique within this data structure.

A _segment name_ is used for the purpose of identification and deletion of a segment within an object. It is defined at segment definition time. A segment name must be unique within an object, but need not be unique within the entire graphical data structure. A segment is, however, always uniquely defined by the name pair: (object, segment).

Theseus provides one level of naming within segments: individual primitives and calls may be named for identification purposes. Such _element names_ are specified as optional parameters of procedures that generate calls and graphical primitives. An element name need not be unique, even within a segment.

Dynamo names (or _dynamos_) are unique pointers to items such as segments, primitives, views, object calls, or to individual transformations in a modelling transformation in the graphical data structure, and are used to modify these items dynamically. A dynamo can be a shorthand for the pair (object, segment), or for the triplets (object, segment, element) or (object, segment, modelling matrix). A dynamo is either created when segments, object calls, views, and primitives are created, or when such items have been identified by a picking device.

8.2.3 EXTENTS

An extent [FOLE76a] of an object is a rectangular window which surrounds the object and whose edges are parallel to the axes of the local coordinate system. The extent is part of the object body. When the object call is executed by the DPU, the extent of the called object is transformed by the current transformation matrix. A new extent which surrounds the transformed extent, but whose edges are parallel to the transformed coordinate axes, is calculated.

The transformed extent is compared with the clip window, and an object whose transformed extent lies entirely inside the clip window can be trivially accepted, i.e., the entire object can be processed without clipping. Similarly, if the transformed extent lies entirely outside the clip window, the object can be trivially rejected, i.e., no further processing of the object is needed. The processing of one extent requires less execution time that the processing of one line, when both endpoints need clipping. Hence extents can, in most cases, result in substantial savings in DPU processing time.

Extents not only decrease execution time by eliminating lines from processing, but also provide facilities for conditional execution of an object or a segment of an object depending on the size of the associated extent relative to the size of the display screen. This allows an object to be defined with "levels of detail", where the amount of detail of an object that is displayed depends of the size of the object extent relative to the size of the display screen. The test to determine if an object or a segment is to be displayed requires less time than clipping of one line segment and can be used to reduce execution time by eliminating details from processing.

Extents are specified at object definition time; it is the responsibility of the programmer to determine the size of the extent surrounding the object. The size limits for conditional execution are specified at object and segment definition time.

8.2.4 VIEWING TRANSFORMATIONS

The Core Graphics System defines a set of viewing functions that allows specifications of all planar geometric transformations [CARL78]. The Core viewing functions were designed to provide a unified approach to the specification of perspective and parallel projections. Futhermore, they were designed to allow individual modification of the parameters that define a perspective projection [CARL78]. The Core viewing functions have, however, some disadvantages. One is that the window in the view plane is not fixed relative to the eyepoint, but moves along with the view plane as that plane is reoriented, which may cause unpredictable results. The other disadvantge is that they are very inefficient from a performance point of view.

As can be seen in [MICH79], the Core viewing transformations can be implemented as a sequence of 4x4 matrix transformations, followed by clipping to a pyramid of vision with its apex at the origin of the world coordinate system and with a 90 degree apex angle, or to an upright parallelepiped centered around the z-axis. The Core viewing functions are all defined relative to the world coordinate system. As a result, the viewing matrices cannot be modified independently. In order to modify one Core viewing function the entire viewing transformation must be modified! This is very undesirable in a highly dynamic application.

In what follows, a different set of viewing functions, functionally equivalent to those of the Core, are proposed. These functions allow individual modifications of each of the viewing functions, and have the further advantage that the window is fixed relative to the center of projection. These new viewing functions have the disadvantage, however, that the variables determining a perspective projection cannot all be modified independently, and that the specification of parallel and perspective projections are quite different.

In the Core, all viewing parameters are specified relative to a single point in the world coordinate system, the view reference point. In Theseus, only the view plane normal and the viewup vector are defined relative to the view reference point. Furthermore, the normal and the viewup direction are specified in one call; they can only be specified and modified jointly. For perspective, the center of projection is defined relative to the UVN-coordinate system. The window center is specified relative to the center of projection, and defines both the view plane distance and the position of the window in the view plane.

For parallel projections, the window center is defined relative to the origin in the UVN-coordinate system. Orthographic projections are specified by a separate function; an orthographic projection is, of course, entirely defined by the view plane normal and the viewup vector. Oblique projections are specified in terms of the properties desired for the projected object, that is in terms of the foreshortening ratio and the angle between the receding lines and the U-axis. For details of these functions, the reader is referred to Appendix C. The remaining viewing functions are similar to those in the Core.

Finally, Theseus allows the specification of a general 4x3 transformation matrix, which, combined with facilities for perspective transformations and clipping to the restricted view volumes, allows any arbitrary viewing transformation to be defined.

Some issues regarding the implementation of these viewing functions still remain for consideraton. These are:

- Should the viewing matrices be constructed in the CPU or the DPU?
- Should each viewing function have a separate dynamo, so that those viewing matrices that are not going to be modified can be composed before they are added to the SDF?

The performance modelling techniques in this thesis could be used to answer these questions.

8.2.5 SUMMARY

Theseus differs from the VG3400 and the E&S PS2 software [VECT78, ESCC77] in many respects. One difference is that Theseus provides a more structured approach to graphics system software. Transformations are not specified modally, but are specified at object calls. Similarly, attributes are specified at object calls and at segment definition time. A user of Theseus is forced to define his objects in an organized fashion, and, in addition, this structured approach to object definition permits better system error checking facilities.

Theseus allows modification of immediate data in the SDF, in addition to support for generation of addressed data. The VG3400 software does not provide support for modification of immediate data, which, under certain circumstances, may result in considerable performance improvements compared to the use of addressed data for the modification of the SDF.

Theseus has support for extents, which never have been used in any other graphics software package.

Finally, Theseus provides viewing parameters that allow specification of all planar geometric projections, which are not directly supported by either VG3400 or E&S PS2 software. Programmers of these systems must use modelling transformations in combination with the limited viewing facilities provided by the two systems in order to achive the effects of completely general viewing transformations.

9 CONCLUSION

9.1 Summary of Research

The main objective of this thesis, to develop an evaluation and design methodology for hardware, firmware, and software for high-performance graphics systems, has been met through a set of performance modelling techniques. The issues of system evaluation have been addressed by the development and use of a set of performance measures for system comparison. The hardware design issues have been addressed by the illustration of how performance measures that are functions of hardware parameters can be developed, and of how the quantitative effects of modifications of the hardware parameters can be determined. The firmware and software design issues have been addressed by the illustration of how to develop performance measures that can be used as an aid in deriving a user instruction set for a graphics system, and in deriving software supporting the graphics system user instructions.

The design methodology developed in this thesis aids in the design of a well-balanced system that maximizes performance for a certain class of applications. In particular, the main focus of this thesis has been on graphics systems for application programs requiring continuous dynamic updating of the picture, either because of the nature of the program itself, or because of operator interaction.

A functional model has been developed that illustrates the basic characteristics of high-performance graphics systems. This model has been used to show the functional similarities and differences between two existing high-performance graphics systems, the Vector General 3400 and the Evans and Sutherland Picture System 2. The primary role of the functional model is as a basis for the performance modelling techniques.

Similarly, the performance modelling techniques have been used to compare certain aspects of the E&S PS2 and the VG3400. Although some results were not unexpected, other results were more surprising. For example, some qualitative trade-offs between the two systems had previously been identified; for some of these trade-offs, the performance modelling techniques were used to determine definite quantitative cut-off points. Futhermore, new light has been shed on some issues regarding the effects of certain features over which there has been considerable debate but no general agreement; these issues have been greatly clarified. Finally, the comparison of the two systems illustrated that an organized comparison process draws attention to certain aspects of the systems that previously had been entirely overlooked.

The performance modelling techniques have also been used to illustrate that by combining some of the functional capabilities of the VG3400 with the speed of parallel transformation/clipping hardware like that of the E&S PS2, an improved graphics system can be derived for applications that require continuous updating of graphical objects.

As a result, some graphics system user instructions for the SDF that are well-suited for dynamics and interaction handling have been proposed, accompanied by a graphics system access method that allows a programmer to build and manipulate a SDF having the proposed user instructions.

The access method provides support for features that are necessary for dynamic updating of all parts of the SDF. It also contains a new set of viewing transformation functions that are functionally equivalent to the viewing functions of the Core Graphics System, but can be implemented at a great reduction in execution speed, in particular when the viewing transformation is updated dynamically. The graphics system access method also provides support for extents, a construct that can result in significant execution speed improvements.

In summary, it has been shown that a unified approach to the design of graphics systems based on a set of tools for quantitative analysis of design trade-offs leads to graphics systems with significantly better performance.

9.2 Future Extensions

The research presented in this thesis can be extended in several areas.

1) As a first step, the performance models used in this thesis should be experimentally verified; if the experimental measures differ significantly from the corresponding analytical estimates, the performance modelling techniques should be refined so that a closer correspondence between experimental and analytical measures is obtained.

2) One obvious extension of this work is the study of other features of high-performance vector graphics systems in order to derive firmware and software support to maximize performance. Some features that should be studied are:

   a) viewing transformations,

   b) text processing, both the type of text (sometimes called "text quality") that should be supported, and the implementation of these types of text,

c) higher level primitives such as conics, rectangles, and spline functions.

3) One aspect of high-performance graphics systems that was not addressed is the use of microprogrammable control store to increase performance for a given application. The techniques in this thesis could be used to suggest which features should be implemented in firmware and what performance improvements can be expected.

4) This thesis has focused only on high-performance vector graphics. It is natural to extend this work to medium- and low-performance devices. In all types of graphics, there is a pipeline where an object is mapped from one representation in an application program to another representation on the display screen. This pipeline can be mapped onto the functional model. For medium- and low-performance graphics, however, the intermediate representations and the processors may be different than for high-performance systems. Despite this difference, the same performance modelling techniques can be used.

5) High-performance raster systems (e.g., RAMTEK 9400) have processing pipelines similar to that presented in the functional model. The contents of the intermediate representations are different, since they must represent solid objects. Futhermore, the processing of the

intermediate representation differs in that a scan conversion is necessary. However, it is believed that the performance modelling techniques could be used as an aid in the design of raster systems.

In addition to the areas for further work mentioned above, the methodology in this thesis may be extended to examine other architectures for graphics systems. For example:

1) A processing pipeline with more intermediate representations.

2) A processing pipeline with more processing in parallel, e.g., concurrent processing of parallel paths in the object hierarchy.

BIBLIOGRAPHY

[ADAG75] Adage, Inc., ADAGE GP/400 Graphics Peripheral System User's Reference Manual, Revision E, Boston, MA (February, 1975).

[ADAG75a] Adage, Inc., GP/400 Graphics Programming Language and Compiler, GPL, User Reference Manual, Revision B, Boston, MA (December, 1975).

[ADAG75b] Adage, Inc., GP/400 Product Description Manual, Boston, MA (October, 1975).

[ADAG75c] Adage, Inc., GPOTS Graphics Peripheral Object Time System User's Reference Manual, Revision A, Boston, MA (May, 1975).

[ADAG78] Adage, Inc., ADAGE 4100 Product Description Manual, Boston, MA (August, 1978).

[ALLE78] Allen, A. O., Probability, Statistics, and Queueing Theory, with Computer Science Applications, Academic Press, New York (1978).

[ARMS79] Armstrong, R., Digital Equipment Corporation, Personal Communication (1979).

[BARB77] Barbacci, M. R., W. E. Burr, S. H. Fuller, and D. P. Siewiorek, (Eds.), Evaluation of Alternative Computer Architectures, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA (February, 1977).

[BEIZ78] Beizer, B., Micro Analysis of Computer System Performance, van Nostrand Reinhold Co, New York (1978).

[BELL78] Bell, C. G., R. Cady, H. McFarland, B. A. Delagi, J. F. O'Loughlin, R. Noonan, and W. A. Wulf, "A New

Architecture for Minicomputers--The DEC PDP-11", in Computer Engineering, A DEC View of Hardware Systems Design, edited by C. G. Bell, J. C. Mudge, and J. E. McNamara, Digital Press, Bedford, MA (1978).

[BELL78a] Bell, C. G. and J. C. Mudge, "The Evolution of the PDP-11" in Computer Engineering, A DEC View of Hardware Systems Design, edited by C. G. Bell, J. C. Mudge, and J. E. McNamara, Digital Press, Bedford, MA (1978).

[BELL78b] Bell, C. G., J. C. Mudge, and J. E. McNamara, (eds.), Computer Engineering, A DEC View of Hardware Systems Design, Digital Press, Bedford, MA (1978).

[BELL78c] Bell, C. G., J. C. Mudge, and J. E. McNamara, "Seven Views of Computer Systems", in Computer Engineering, A DEC View of Hardware Systems Design, edited by C. G. Bell, J. C. Mudge, and J. E. McNamara, Digital Press, Bedford, MA (1978).

[BERG76] Bergeron, R. D., "Picture Primitives in Device Independent Graphics Systems", Computer Graphics 10, 1 (1976) 57-60.

[BERG78] Bergeron, R. D., P. R. Bono, and J. D. Foley, "Graphics Programming Using the Core System", Computing Surveys 10, 4 (December, 1978) 389-443.

[BERG76a] Bergman, S. and A. Kaufman, "BGRAF2: A Real-time Graphics Language with Modular Objects and Implicit Dynamics", Computer Graphics 10, 2 (Summer, 1976) 133-138.

[BERK72] Berk, T. S., The Design and Implementation of TUNA, A High Level Graphical Programming Language, PhD. Thesis, Purdue University, Lafayette, IN (August, 1972).

[BLOO76] Bloom, H. J. and E. DeJong, A Critical Comparison of Several Implementations of Programming Languages, Department of Computer Science, Mathematical Centre, Amsterdam (December, 1976).

[BOEH69] Boehm, B.W., V.R. Lamb, R.L. Mobley, and J.E. Rieber, "POGO: Programmer-Oriented Graphics Operation", Proc. SJCC 34 (May, 1969) 321-330.

[BOUL72] Boullier, P., J. Gros, P. Jancene, A. Lemaire, F. Prusker, and E. Saltel, "METAVISU, A General Purpose Graphic System", Graphic Languages, edited by F. Nake and A. Rosenfeld, North-Holland Publishing Co., Amsterdam (1972) 244-267.

[BOYS75] Boyse, J. W. and D. R. Warn, "A Straightforward Model for Computer Performance Prediction", Computing Surveys 7, 2 (June, 1975) 73-93.

[BREN78] Brender, R. F., "Turning Cousins into Sisters: An Example of Software Smoothing of Hardware Differences", in Computer Engineering: A DEC View of Hardware Systems Design, edited by C. G. Bell, J. C. Mudge, and J. E. McNamara, Digital Press, Bedford, MA (1978).

[BUZE76] Buzen, J. P., "Fundamental Operational Laws of Computer System Performance", Acta Informatica 7 (1976) 167-182.

[BUZE79] Buzen, J. P., "Operational Anlysis: An Alternative to Stochastic Modeling, in Performance of Computer Installations, edited by D. Ferrari, North Holland Publishing Co., Amsterdam (June, 1979) 175-194.

[BUZE79a] Buzen, J. P., BGS Systems, Inc., Personal Communication (1979).

[CADC75] CAD Centre, GINO-F, The General Purpose Graphics Package Reference Manual, Cambridge, England (1975).

[CARL78] Carlbom, I. and J. Paciorek, "Planar Geometric Projections and Viewing Transformations", Computing Surveys 10, 4 (December, 1978) 465-502.

[CARU75] Caruthers, L. C. and A. van Dam, GPGS User's Tutorial, Informatica Group, Faculty of Science, University of Nijmegen, Nijmegen, The Netherlands (October, 1975).

[CHRI67] Christensen, C. and E. N. Pinson, Multi-function Graphics for a Large Computer System, Proc. FJCC 31 (November, 1967) 697-712.

[CISL72] Cislo, R. A., "Graphics Systems Performance Evaluation", Proc. ACM National Conference, (August, 1972) 432-442.

[COFF73] Coffman, E. G., Jr. and P. J. Denning, Operating Systems Theory, Prentice-Hall, Inc., NJ (1973).

[COOP72] Cooper, R. B., Introduction to Queueing Theory, The MacMillan Company, New York, (1972).

[COTT70] Cotton, I. W., "Languages for Graphic Attention-Handling", Proc. International Symposium on Computer Graphics, Brunel (April, 1970).

[COTT74] Cotton, I. W., "Network Graphic Attention Handling", in Readings in Computer Graphics, to be published by Academic Press.

[COTT68] Cotton, I. W. and F. S. Greatorex, "Data Structures and Techniques for Remote Computer Graphics", Proc. FJCC 32-2 (December, 1968) 533-544.

[DENE75] Denert, E., G. Ernst, and H. Wetzel, "GRAPHEX68--Graphical Language Features in Algol68", Computers and Graphics 1, 2/3 (1975), 195-202.

[DENN78] Denning, P.J. and J.P. Buzen, "The Operational Analysis of Queueing Network Models", Computing Surveys 10, 3 (September, 1978) 225-261.

[DEC71] Digital Equipment Corporation, Digital PDP11 Peripherals and Interfacing Handbook, Maynard, MA (1971).

[DEC74] Digital Equipment Corporation, Digital PDP11/45 Processor Handbook, Maynard, MA (1974).

[DEC76] Digital Equipment Corporation, Digital PDP11/70 Processor Handbook, Maynard, MA (1976).

[DEC78] Digital Equipment Corporation, Digital PDP11 04/34/45/55/60 Processor Handbook, Maynard, MA (1978).

[DEC78a] Digital Equipment Corporation, Digital PDP11 Peripherals Handbook, Maynard, MA (1978).

[DEC79] Digital Equipment Corporation, PDP-11 Unibus, Design Description, Preliminary, Maynard MA (May, 1979).

[ESCC70] Evans & Sutherland Computer Corp., Line Drawing System Model 1 System Reference Manual U0800-1-1, Salt Lake City, UT (November, 1970).

[ESCC74] Evans & Sutherland Computer Corp., The Picture System User's Manual, ES-PS-S001-003, Salt Lake City, UT (December, 1974).

[ESCC77a] Evans & Sutherland Computer Corp., Picture System 2/PDP-11 Reference Manual, E&S 901130-001-A1. Salt Lake City, UT (November, 1972).

[ESCC77] Evans & Sutherland Computer Corp., Picture System 2 User's Manual, E&S #901129-001 NC, Salt Lake City, UT (May, 1977).

[EWAL78] Ewald, R. H. and R. Fryer, (eds.), "Final Report of the GSPC State-of-the-Art Subcommittee", Computer Graphics 12, 1-2 (June, 1978) 14-169.

[FELD69] Feldman, J. A. and P. D. Rovner, "An ALGOL-Based Associative Language", CACM 12, 8 (August, 1969) 439-449.

[FERR78] Ferrari, D., Computer Systems Performance Evaluation, Prentice-Hall, Inc., Englewood Cliffs, NJ (1978).

[FOLE71] Foley, J. D., "An Approach to the Optimum Design of Computer Graphics Systems", CACM 14, 6 (June, 1971) 380-390.

[FOLE73] Foley, J.D., "Software for Satellite Graphics Systems", Proc. ACM National Conference 26, (August, 1973) 76-80.

[FOLE76] Foley, J. D., "Picture Naming and Modification: An Overview", Computer Graphics 10, 1 (1976) 49-53.

[FOLE76a] Foley, J. D., "Extents, Windows, and Instance Rectangles", Internal Report, Department of Computer Science, University of North Carolina, NC (1976).

[FOLE76b] Foley, J. D., "A Tutorial on Satellite Graphics Systems", Computer (August, 1976) 14-21.

[FOLE75] Foley, J. D., R. Hogan, and C. Dunham, Display-Independent Graphics System--DIGS, Department of Computer Science, University of North Carolina, Chapel Hill, NC (1975).

[FOLE76c] Foley, J. D., A. van Dam, R. Burns, I. Carlbom, and H. Webber, "Survey Report on Vector General 3400, Adage GP/400, Evans and Sutherland Picture System", prepared for Vector General, Inc., Woodland Hills, CA (May, 1976).

[FOLE74] Foley, J. D. and V. L. Wallace, "The Art of Natural Graphic Man-Machine Conversation", Proc. of the IEEE 62, 4 (April, 1974) 462-471.

[FOLE74a] Foley, J. D., V. L. Wallace, E. Britton, E. Brownlee, D. Mitchell, R. Zarling, and J. McInroy, Graphic System Modeling, First Annual Report, University of North Carolina at Chapel Hill, NC (June, 1974).

[FOLE75a] Foley, J. D., et al, Graphics System Modeling: Verification and Applications, Second Annual Report, University of North Carolina at Chapel Hill, NC (November, 1975).

[GILO75] Giloi, W. K., "On High-Level Programming Systems for Structured Display Programming", Computer Graphics 9, 1 (Spring, 1975) 61-69.

[GPGS75] GPGS-F User's Guide, RUNIT Computer Centre, University of Trondheim, Norway (September, 1975).

[GROO77] Groot, D., E. Hermans, L.C. Caruthers, and J. Schwartz, GPGS Reference Manual, Rekencentrum, Tech. Ho. Delft, and Informatica Group, Faculty of Science, University of Nijmegen, The Netherlands (May, 1977).

[GRAY67] Gray, J. C., "Compound Data Structure for Computer Aided Design: A Survey", Proc. ACM Nat. Meeting (1967) 355-365.

[GSPC77] Graphics Standards Planning Committee, "General Methodology and Proposed Standard", Computer Graphics 11, 3 (Fall, 1977) II-1 to II-117.

[GSPC79] Graphics Standards Planning Committee, "General Methodology and the Proposed Core System (Revised)", Computer Graphics 13, 3 (August, 1979) II-1 to II-179.

[HELL75] Hellerman, H. and T. F. Conroy, Computer System Performance, McGraw-Hill, Inc., New York (1975).

[HELL70] Hellerman, H. and H. J. Smith, Jr., "Throughput Analysis of Some Idealized Input, Output, and Computer Overlap Configurations", Computing Surveys 2, 2 (June, 1970) 111-118.

[HURW67] Hurwitz, A., J. P. Citron, and J. B. Yeaton, "GRAF: Graphic Additions to FORTRAN", Proc. SJCC 1967, (1967) 553-557.

[LEIN79] Leinwand, A., Vector General, Inc., Personal Communication (1979).

[LEVY78] Levy, J. V., "Buses, The Skeleton of Computer Structures", in Computer Engineering, A DEC View of Hardware

Systems Design, edited by C. G. Bell, J. C. Mudge, and J. E. McNamara, Digital Press, Bedford, MA (1978).

[MALL78] Mallgren, W. R. and A. C. Shaw, Graphical Transformations and Hierarchic Picture Structures, Department of Computer Science, University of Washington, Seattle, WA (June, 1978).

[MEGA79] MEGATEK Corporation, MEGATEK 7000 Display Format Manual, 0250-0005-01, San Diego, CA (May, 1979).

[MICH77] Michel, J. and A. van Dam, "Evaluation of Performance Improvement in Distributed Processing", Second Workshop on Distributed Processing, Brown University, Providence, RI (August, 1977).

[MICH76] Michener, J. C., The /GS Graphics Programming System, Volume I: Concepts and Facilities, IR-189, Intermetrics, Inc., Cambridge, MA (September, 1976).

[MICH79] Michener, J. C., Some Viewing Implementation Considerations for the 1979 GSPC Core System, Notes for Tutorial on Graphics Standards at SIGGRAPH 1979, Intermetrics, Inc., Cambridge, MA, (1979).

[MICH76a] Michener, J. C. and D. D. Struble, The /GS Graphics Programming System, Volume II: SPL/I/GS Reference Manual, IR-199, Intermetrics, Inc., Cambridge, MA (September, 1976).

[MICH78] Michener, J. C. and A. van Dam, "A Functional Overview of the Core System with Glossary", Computing Surveys 10, 4 (December, 1978) 381-387.

[MICH78a] Michener, J. C. and J. D. Foley, "Some Major Issues in the Design of the Core Graphics System", Computing Surveys 10, 4 (December, 1978) 445-463.

[MIDD74] Middleton, N. C., et al., Graphics Compatibility System (GCS) Programmer's Reference Manual, United States Military Academy, West Point, NY, NTIS #AD779211 (April, 1974).

[MOOR79] Moore, R., Digital Equipment Corporation, Personal Communication (1979).

[MYER68] Myer, T. H. and I. E. Sutherland, "On the Design of Display Processors", CACM 11, 6 (June, 1968) 410-414.

[MYER78] Myers, G. J., Composite/Structural Design, van Nostrand-Reinhold Co., (1978).

[NEWM68] Newman, W. M., "A System for Interactive Graphics Programming", AFIPS SJCC, 38, (1968) 47-54.

[NEWM71] Newman, W. M., "Display Procedures", CACM 14, 10 (October, 1971) 651-660.

[NEWM73] Newman, W. M., "An Informal Graphics System Based on the Logo Language", AFIPS Conference Proc., 42, 1973 National Computer Conference and Exposition, AFIPS Press (1973).

[NEWM75] Newman, W. M., "Instance Rectangles and Picture Structures", Proc. Conference on Computer Graphics, Pattern Recognition and Data Structures, University of California, Los Angeles (May, 1975) 297-301.

[NEWM73a] Newman, W. M. and R. F. Sproull, Principles of Interactive Computer Graphics, McGraw-Hill Book Company, New York (1973).

[NEWM74] Newman, W. M. and R. F. Sproull, "An Approach to Graphics System Design", Proc. of the IEEE 62, 4 (April, 1974) 471-483.

[NEWM79] Newman, W. M. and R. F. Sproull, Principles of Interactive Computer Graphics, Second Edition, McGraw-Hill Book Company, New York (1979).

[NEWM78] Newman, W. M. and A. van Dam, "Recent Efforts Towards Graphics Standardization", Computing Surveys 10, 4 (December, 1978) 365-380.

[NG73] Ng, Nam, An Environment-Independent Graphics Facility, PhD. Thesis, Tech. Report TR73-11, Department of Computer Science, University of Alberta, Canada (September, 1973).

[OBRI75] O'Brien, C.D. and H.H. Bown, "IMAGE: A Language for the Interactive Manipulation of a Graphics Environment", Computer Graphics 9, 1 (Spring, 1975) 53-60.

[PARE77] Parent, R. E., A System for Generating Three-Dimensional Data for Computer Graphics, PhD. Thesis, Ohio State University, Ohio (December, 1977).

[PFIS76] Pfister, G. F., "A High Level Language Extension for Creating and Controlling Dynamic Pictures", Computer Graphics 10, 1 (1976) 1-9.

[PUK76] Puk, R. F., The Optimal Distribution of Device-Dependent Graphics Functions, PhD. Thesis, Purdue University, Lafayette, IN, (May, 1976).

[PUK76] Puk, R. F., The 3D Graphics Compatibility System, U. S. Army Corps of Engineers, Waterways Experiment Station, Vicksburg, MS (October, 1976).

[ROSS67] Ross, D. T., "The AED Approach to Generalized Computer-Aided Design", Proc. ACM National Meeting, (1967) 367-385.

[ROVN69] Rovner, P. D. and J. A. Feldman, "The LEAP Language and Data Structure", Information Processing 68, North-Holland Publishing Company, Amsterdam (1969) 579-585.

[SMIT71] Smith, D. N., "GPL/I--A PL/I Extension for Computer Graphics", SJCC 1971 (1971) 511-528.

[SNOW78] Snow, E. A. and D. P. Siewiorek, "Impact of Implementation Design Tradeoffs on Performance: The PDP-11, A Case Study", Computer Engineering, A DEC View of Hardware Systems Design, edited by C. G. Bell, J. C.

Mudge, and J. E. McNamara, Digital Press, Bedford, MA (1978).

[SORG77] Sorgie, C.D., ALGOL W Reference Manual, BCL017-001-00, Brown University, Providence, RI (January, 1977).

[SPRO74] Sproull, R. F. and E. L. Thomas, "A Network Graphics Protocol", Computer Graphics 8, 3 (Fall, 1974) 27-51.

[STAB73] Stabler, G. M., The Brown University Graphics System, Brown University, Providence, RI (1973).

[STAC71] Stack T. R. and S. T. Walker, "AIDS--Advanced Interactive Display System", Proc. SJCC 1971, (1971) 113-121.

[STER74] Stern, R. A., Draft Report of GLYPH: A Graphical Extension to CS-4, Intermetrics, Inc., Cambridge, MA (May, 1974).

[STON77] Stone, H. S., "Multiprocessor Scheduling with the Aid of Network Flow Algorithms", IEEE Trans. of Software Engineering SE-3, 1 (January, 1977) 85-93.

[STOW78] Stowell, G. W., Microprocessor Based Architecture for Interactive Graphic Computers, Research Report, Department of Mechanical Engineering and Computer Science, Purdue University, IN (1978).

[STRE78] Strecker, W. D., "Cache Memories for PDP-11 Family Computers", Computer Engineering. A DEC View of Hardware Systems Design, edited by C. G. Bell, J. C. Mudge, and J. E. McNamara, Digital Press, Bedford, MA (1978).

[SULO75] Sulonen, R., A Study in Concept for an Interactive Graphic Programming Language, PhD. Thesis, Helsinki University, Finland (October, 1975).

[SUTH63] Sutherland, I. E., "SKETCHPAD: A Man-Machine Graphical Communication System", _Proc. AFIPS 1963 SJCC, Vol 22_, Spartan Books, New York (1963) 329-346.

[SUTH69] Sutherland, W. R., J. W. Forgie, and M. V. Morello, "Graphics in Time-sharing: A Summary of the TX-2 Experience" _Proc. SJCC 1969_ (1969) 629-636.

[SVOB76] Svobodova, L., _Computer Performance Measurement and Evaluation Methods: Analysis and Applications_, American Elsevier Publishing Co., Inc., New York (1976).

[THOM76] Thomas, E. L., "Methods for Specifying Display Parameters in Graphics Programming Languages", _Computer Graphics 10_, 1 (1976) 54-56.

[TURR75] Turrill, C. N. and W. R. Mallgren, "XPLG--Experiences in Implementing an Experimental Interactive Graphics Programming System", _Computers and Graphics 1_, 1 (1975) 55-63.

[VAND72] van Dam, A., _Some Implementation Issues Relating to Data Structures for Interactive Graphics_, TR No. 72-1, Center for Computer and Information Sciences, Brown University, Providence, RI (1972).

[VAND73] van Dam, A. and G. M. Stabler, "Intelligent Satellites for Interactive Graphics", _Proc 1973 NCC_, Spartan Books, Baltimore, MD (1973) 229-238.

[VAND74] van Dam, A., G. M. Stabler, and R. J. Harrington, "Intelligent Satellites for Interactive Graphics", _Proc. of the IEEE 62_, 4 (April, 1974) 483-492.

[VAND78] van den Bos, J., _Definition and Use of Higher-Level Graphic Input Tools_, Report No. 5, Nijmegen University, Nijmegen, Holland (February, 1978).

[VAND77] van den Bos, J., L. C. Caruthers, and A. van Dam, "GPGS - A Device-independent General Purpose Graphic System for Stand-alone and Satellite Graphics", _Computer Graphics 11_, 2 (Summer, 1977), 112-119.

[VECT77] Vector General, Inc., Series 3400 Technical Manual, Volume II, Graphic Processor Unit, Pub. No. M110380, Woodland Hills, CA (September, 1977).

[VECT78] Vector General, Inc., FGP34 FORTRAN Graphics Package, Woodland Hills, CA (1978).

[VECT78a] Vector General, Inc., Graphics Display System, Model 3404, Programming Concepts Manual, Pub. No. 113489, Woodland Hills, CA (July, 1978).

[VECT78b] Vector General, Inc., Graphics Display System, Model 3404, System Reference Manual, Pub. No. M110700REF, Woodland Hills, CA (August, 1978).

[VECT78c] Vector General, Inc., Series 3400 Technical Manual, Volume I, Graphics Display System, Pub. No. M110700, Woodland Hills, CA (March, 1978).

[VECT78d] Vector General, Inc., VG3400 Firmware Code (June, 1978).

[WALL75] Wallace, V. L., GRASP, A PL/I-Oriented Machine Independent Graphics Structure Handler, An Introduction, Department of Computer Science, University of North Carolina at Chapel Hill, NC (February, 1975).

[WALL76] Wallace, V. L., "The Semantics of Graphic Input Devices", Computer Graphics 10, 1 (Spring, 1976) 61-65.

[WATK79] Watkins, G., Evans & Sutherland Computer Corp., Personal Communication (1979).

[WEBB77] Webber, H. and R. W. Burns, The SIMALE Standard Graphics Package, Preliminary Description, Brown University, Providence, RI (December, 1977).

[WEGN79] Wegner, P., Editor, Research Directions in Software Technology, The MIT Press, Cambridge, MA (1979).

[WHIT64] White, P., "Relative Effects of Central Processor and Input-Output Speeds Upon Throughput on the Large Computer", CACM 7, 12 (December, 1964) 711-714.

[WILL75] Williams, R. and G. M. Giddings, "A Picture-Building System", Proc. Conference on Computer Graphics, Pattern Recognition and Data Structures, University of California, Los Angeles (May, 1975) 304-307.

[WOOD71] Wood, D. C. and E. H. Forman, "Throughput Measurement Using a Synthetic Job Stream", FJCC, (1971) 51-56.

[WOOD71a] Woodsford, P., "The Design and Implementation of the GINO 3D Graphics Software Package", Software Practice and Experience 1, 4 (1971) 335-365.

A APPENDIX: SUMMARY OF GRAPHICS SYSTEM CAPABILITIES

This appendix presents an item-by-item comparison of the DPU for the VG3400 [VECT78b] and the E&S PS2 [ESCC77, ESCC77a].

VG3400	E&S PS2
Line Primitives	
connected/disconnected	connected/disconnected
points (as attributes)	points
abs/rel different for x,y,z	abs/rel/abs-rel-rel...
2D - in xy, xz, or yz plane	2D in - xy plane
3D	3D
packed/unpacked	based
auto incr x, y, and/or z	
Text Primitives	
packed/unpacked	packed
rotation	limited rotation
start point transformed	start point transformed and clipped
all chars transformed	
italics	italics
clipping	clipping at right boundary of screen
margin control	

4 basic sizes, but these can be transformed	8 standard sizes
	user-defined fonts
	subscript, superscript
	controllable intercharacter and interline spacing
	microprogrammed - two fonts

Other Primitives

circle/arc	circle/arc
entire circle transformed and clipped	
cubic	
rectangle	

Transformations

rotation - angle	rotation - angle
uniform scale	x, y, z scale
translate	translate
perspective	perspective
2D/3D window	2D/3D window
3D clipping	3D clipping
2D, square viewport	arbitrary 2D/3D viewport (3d provides depth cueing)
	general 4x4 homogeneous matrix

Attributes

8 line textures	5 line textures
color	5 colors
blink	blink
intensity	intensity
depth cueing	64 for depth cueing
constant	4 for constant settings

Stacking and Compounding of Data

hardware stack - size specified at initialization time	hardware stack - 14 levels
anything can be stacked	matrices are stacked
program read/write stack access	program read/write stack access

Naming

for picking: object id and element within object	for picking: one level in buffer; hit testing in window gives content of name register
object and anything else can be stacked.	
for modification:	for modification:
name in correlation table	name in correlation table
pick: select/edit option	

Addressing Modes

instruction	instruction
immediate	immediate
referenced: 14 modes	
object call: 2 levels of indirection; base+displac.	object call: direct
addressing for:	addressing for:
main memory	main memory of PDP-11
registers: 256	refresh buffer
	register file

Flow of Control

unconditional call, return, break, interrupt	unconditonal call, return
conditional call, return	

General Purpose Instructions

instrs take 3 operands	none
plus, minus, times, divide	
and, or, xor	
shift, transfer	

Input Tools Sold By Manufacturer

light pen	light pen
tablet	tablet
keyboard	keyboard
joystick	joystick
dials	dials
function keys	function keys
track ball	lighted function buttons

B APPENDIX: PDP-11 ASSEMBLER CODE

This appendix contains some PDP-11 code fragments and subroutines. This code was developed in the course of establishing the results for the E&S PS2 in Chapter 6. The code fragments, once written, were analyzed to determine processing times. It is assumed that this code is part of a FORTRAN program, so the subroutines described here conform to FORTRAN calling conventions. (The timings assume that a PDP-11/45 is in use, with both program and data in 16K of core memory.)

This PDP-11 code fragment updates a line endpoint in the Picture Memory of the E&S PS2. It is assumed that the object name is in OBJECT, the offset to the three locations is in OFFSET, and that PTR points to the new coordinates.

```
MOV   #PARA,R5          ; ADDR OF PARAMETERS
MOV   #3,(R5)           ; NUMBER OF PARAMETERS
MOV   #OBJECT,2(R5)
MOV   #OFFSET,4(R5)
MOV   PTR,6(R5)
JSR   PC,UPVAL          ; UPVAL DOES THE UPDATING
```

The total time is 153.3 microseconds.

This subroutine updates three consecutive locations in Picture Memory. A typical call looks like:

```
        UPVAL (OBJECT, OFFSET, PTR)

UPVAL:  CMPB  (R5)+,#3         ; CHECK # OF PARAMETERS
        BNE   $ERROR
        MOV   @(R5)+,R0        ; GET OBJECT NAME
        JSR   PC,MAKDEX        ; FIND OBJECT IN DIRECTORY
        TST   R1               ; IS NAME IN DIRECTORY?
        BEQ   #ERROR           ; NO. ERROR.
        TSTB  P$OBJ(R1)        ; IS OBJECT REALLY IN DIRECTORY?
        BEQ   #ERROR           ; NO. ERROR.
        MOV   P$BASE(R1),R0    ; GET OBJECT PTR
        ADD   @(R5)+,R0        ; ADD OFFSET
        MOV   #3,R1            ; SET UP DMA COUNT
        MOV   @(R5),R2         ; SET UP DMA ADDR
        JSR   PC,DMAOUT
        RTS   PC
```

The total time for this subroutine is 131.3 microseconds.

This routine transfers data to the Picture Memory via a DMA. It is an internal routine, to be used by the graphics access method, not the application program.

Parameters:

R0 - PS2 address

R1 - DMA word count

R2 - DMA base address

```
DMAOUT: BIT    #200,IOST         ; DMA READY?
        BEQ    DMAOUT
        WTPS   R0,#DMAPSA
        MOV    R1,DMAWC
        MOV    R2,DMABA
        BIS    #1,IOST           ; START TRANSFER
        RTS    PC
```

The time for this routine is 41.2 microseconds.

This code sequence defines a macro, used on the previous page, that writes from PDP-11 "SRC" to E&S PS2 "DST"

```
.MACRO    WTPS   SRC,DST
          TST    IOST             ; TEST IF DIO READY
          BPL    .-4              ; DIO NOT READY. LOOP BACK
          MOV    DST,DIOPSA       ; SET UP ADDRESS
          TST    IOST
          BPL    .-4              ; NOT YET DONE WITH ADDRESS
          MOV    SRC,PSDATA       ; SEND DATA
.END      WTPS
```

This routine looks through the directory (BUFBAS) for a given object. It is an internal routine, to be used by the graphics access method, not the application program.

Parameters:

R0 - Object name

R1 - Return value: directory pointer if object found; 0 if object not found.

```
MAKDEX:  MOV   BUFBAS,R1          ; BEGINNING OF DIRECTORY
1$:      CMP   P$NAME(R1),R0      ; IS THIS THE OBJECT?
         BEQ   4$                 ; YES. EXIT.
         CMP   P$NAME(R1),#-1     ; END OF DIRECTORY?
         BEQ   5$                 ; YES. EXIT.
         ADD   #DIRLEN,R1         ; ADVANCE TO NEXT ENTRY
         BR    1$
5$:      CLR   R1                 ; PREPARE 0 (FAILURE) RETURN CODE
4$:      RTS   PC
```

The total time to find the fifth object in the directory is 57.6 microseconds.

This program fragment reads three dials (via the subroutine DIAL) and updates a translation matrix according to the values obtained. It is assumed that the translate portion of the matrix is located at a given offset (OFFSET) within a given object (OBJNAM).

```
        MOV     #PARA,R5            ; ADDR OF PARAMETERS
        MOV     #ONE,(R5)           ; CALL DIAL(1,3,DVAL)
        MOV     #3,2(R5)
        MOV     #DVAL,4(R5)
        JSR     PC,DIAL
        MOV     #3,(R5)             ; CALL UPVAL (OBJNAM,OFFSET,DVAL)
        MOV     #OBJNAM,2(R5)
        MOV     #OFFSET,4(R5)
        MOV     #DVAL,6(R5)
        JSR     PC,UPVAL
```

The total time for this code fragment is 262.4 microseconds.

This program fragment reads three dials (via the subroutine DIAL) and compares them with previous values. If any value is changed, it updates a translation matrix according to the values obtained. It is assumed that the translate portion of the matrix is located at a given offset (OFFSET) within a given object (OBJNAM).

```
        MOV     #PARA,R5          ; ADDR OF PARAMETERS
        MOV     #ONE,(R5)         ; CALL DIAL(1,3,DVAL)
        MOV     #3,2(R5)
        MOV     #DVAL,4(R5)
        JSR     PC,DIAL
        MOV     #DVAL,R0          ; COMPARE OLD AND NEW VALUES
        MOV     #DVALOLD,R1
        CMP     (R0)+,(R1)+
        BNE     1$
        CMP     (R0)+,(R1)+
        BNE     1$
        CMP     (R0),(R1)
        BE      NOCHNG
1$:     MOV     #3,(R5)           ; CALL UPVAL (OBJNAM,OFFSET,DVAL)
        MOV     #OBJNAM,2(R5)
        MOV     #OFFSET,4(R5)
        MOV     #DVAL,6(R5)
        JSR     PC,UPVAL
```

The total time for this code fragment is 128.2 microseconds if the matrix is not changed and 278.5 microseconds if it is changed.

This subroutine reads N dial values, starting with dial number DNUM, into the variable RESULT. A typical call looks like:

SUBROUTINE DIAL (DNUM, N, RESULT)

```
DIAL:   CMPB    (R5)+,#2        ; CHECK # OF PARMS
        BNE     $ERROR
        MOV     @(R5)+,R0       ; GET PARAMETERS
        MOV     @(R5)+,R1
        MOV     (R5),R2
        ADD     #DIALADDR,R0    ; ADDR OF ZEROTH DIAL
1$:     TST     R1              ; TEST IF DONE
        BLE     2$              ; YES. EXIT.
        RDPS    R0,(R2)         ; READ NEXT DIAL
        ADD     #2,R0           ; INCREMENT AND LOOP BACK
        ADD     #2,R2
        SUB     #1,R1
        BR      1$
2$:     RTS     PC
```

The time to read three dials is 95.3 microseconds.

This code sequence defines a macro, used on the previous page, that reads from E&S PS2 "SRC" into PDP-11 "DST"

```
.MACRO    RDPS   SRC,DST
          TST    IOST             ; TEST IF DIO READY
          BPL    .-4              ; DIO NOT READY. LOOP BACK
          MOV    SRC,DIOPSA       ; SET UP ADDRESS
          TST    IOST
          BPL    .-4              ; NOT YET DONE WITH ADDRESS
          MOV    PSDATA,DST       ; GET DATA
.END      RDPS
```

C APPENDIX: THESEUS

C.1 Introduction

In the description of Theseus that follows the routines are grouped by topic: first the routines that build and modify the hierarchy are defined, next the transformations, followed by the primitives, the attributes, and the interaction handling. The last section describes some miscelaneous features regarding the operation of Theseus.

The description of each routine is preceded by the syntax of the Algol W declaration for the routine. Upper case symbols denote Algol W and Theseus keywords; lower case symbols denote parameters to be specified by the user. Brackets ([ ]) indicate optional parameters. In the declaration, the different data types are separated by semicolons; note that in actual calls, all arguments are separated by commas. Optional procedure parameters can be omitted at a call, but the preceding comma must be specified. Optional integer parameters must be specified as non-positive integers, to indicate that they are to be ignored by Theseus.

Transformations, attributes, and extents are specified as procedure parameters of routines that define objects, segments, views, and object calls. An actual parameter corresponding to a procedure parameter can be any Algol W statement: a simple statement, a begin block, or a procedure statement. However, the only Theseus calls that can be used as part of any type of procedure parameters are Theseus routines of the same type. For example, a modelling transformation parameter (usually written in a declaration as "PROCEDURE mtrans") may only contain Theseus modelling transformation function invocations. Similarly, an attribute parameter may only contain Theseus attribute function invocations.

C.2 Hierarchy

The first three subsections below discuss those routines that are used to build and modify the components of the hierarchical data structure: objects and object calls, views, and segments. Each of these constructs can have associated names for identification and modification purposes. Naming is discussed in the following subsection. Extents and conditional execution of objects are discussed at the end of this section.

C.2.1 OBJECTS

Theseus allows the user to create and delete objects, to modify objects by adding and deleting segments, and to create object calls. All attributes and all modelling transformations of a call may be altered dynamically. In addition, an individual attribute or an individual modelling transformation may be altered dynamically, as will be discussed in Sections C.5 and C.3. An object may have an associated extent and can be conditionally executed depending on the size of this extent. Objects may be saved in picture files, and retrieved from these files for later use. Object creation, deletion and calling are discussed in this section; the discussion of picture files is deferred to Section C.7.

```
OPOBJ2D( INTEGER VALUE name; [PROCEDURE extent]; [PROCEDURE size])
OPOBJ3D( INTEGER VALUE name; [PROCEDURE extent]; [PROCEDURE size])
```

> make the object "name" the open object for addition and deletion of segments. An object name must be unique within the data structure. If the specified object name exists, the existing object is modified; otherwise a new object is created. The object remains the open object until a call to CLOSOBJ or DLTOBJ.

An extent, i.e., a surrounding window of an object, can optionally be specified as part of the object definition. An object's extent can be modified when that object is reopened. The extent is used to aid the clipping process, and is further discussed in Section C.2.5.

By specifying a minimum and maximum extent size relative to the size of the display screen, the object will be executed conditionally, depending upon its extent size. The procedure "size" defines the minimum and maximum extents (see Section C.2.5), and can be altered as the object is reopened. Note that if the object was created without an extent and without a size specification, an extent and the associated size cannot be added when an object is reopened.

Error Conditions:

1) No more core available for graphical data structure.

2) Dimensionality mismatch.

3) An object is already open.

4) Invalid name; must be a positive integer.

5) Invalid extent routine.

6) Invalid size routine.

CLOSOBJ

The open object is closed, and cannot be modified until it is again made the open object by a call to OPOBJ2D or OPOBJ3D.

Error Conditions:

1) No open object.

2) A segment was open; it has been closed.

DCLOBJ2D(INTEGER VALUE name)

DCLOBJ3D(INTEGER VALUE name)

declare a 2D and a 3D object, respectively, i.e., enter an object name in the directory. These procedures enable the user to generate an object call before the object is created, thereby allowing him to create a picture "top-down" instead of "bottom-up". The actual object may subsequently be defined by calling OPOBJ2D or OPOBJ3D.

Although these procedures may be invoked while there is an open object, this should generally be avoided since it causes inefficiency in DPU processing.

Error Conditions:

1) Object name already exists.

2) No more core available for graphical data structure.

3) Invalid name; must be a positive integer.

4) Dimensionality mismatch.

NULLOBJ(INTEGER VALUE name)

deletes the body of the named object, but leaves the object name in the directory. A new body for this object may subsequently be defined by reopening the object with OPOBJ2D or OPOBJ3D.

Error Conditions:

1) Object name does not exist.

2) Invalid name; must be a positive integer.

DLTOBJ(INTEGER VALUE name)

deletes the body of the object and removes its name from the directory. If there are any remaining calls to this object in the data structure they will be ignored until the directory entry is used again. Since the programmer cannot directly control allocation of directory entries, this can cause unpredictable results.

Error Conditions:

1) Object name does not exist.

2) Invalid name; must be a positive integer.

RNMOBJ(INTEGER VALUE oldname, newname)

renames the object "oldname" to "newname". All existing calls to object "oldname" will refer to "newname", and the object must subsequently be referenced by "newname".

Error Conditions:

1) Object newname already exists.

2) Object oldname does not exist.

3) Invalid name; must be a positive integer.

SWAPOBJ(INTEGER VALUE name1, name2)

exchanges the bodies of the objects "name1" and "name2". This routine can be used for double buffering. The object can be built as object "name2" while displayed as object "name1". When object "name2" is completed, the two objects are swapped and a new version of the object is again built as object "name2". No renaming is required!

Error Conditions:

1) Object name1 and/or name2 does not exist.

2) Dimensionality mismatch.

3) Invalid name; must be a positive integer.

CALLOBJ(INTEGER VALUE name, [elemname], [dynamo]; [PROCEDURE attr]; [PROCEDURE mtrans])

adds a call of the object "name" to the open segment in the display data structure. The element name may be specified for identification of the object call, and a dynamo name may be specified for modification of the object call. The attributes and transformations (see Sections C.5 and C.3) of the object call are specified by "attr" and "mtrans", respectively.

If no attributes are specified, null attributes will be applied. If the transformations are omitted, a call without a transformation is generated.

In order to delete a call to an object, the surrounding segment must be deleted. Any component of the call may be altered dynamically by CHGCALL, which is described below.

An object may call itself recursively.

Error Conditions:

1) No more core available for graphical data structure.

2) Object name does not exist.

3) Dynamo name already exists.

4) Dimensionality mismatch.

5) No open segment.

6) Invalid attribute routine.

7) Invalid modelling transformation routine.

8) Invalid name; must be a positive integer.

9) A menu object cannot be referenced by an object call.

CHGCALL(INTEGER VALUE dynamo, [name]; [PROCEDURE attr]; [PROCEDURE mtrans])

replaces the called object, and can modify all the attributes and the entire transformation of an existing object call pointed to by the "dynamo" (see Section C.2.4). "name", "attr" and "mtrans" are defined as for CALLOBJ. If an optional parameter of the call is a non-positive integer or is omitted, the corresponding component of the object call is left unchanged. Note that if the object call was generated without a modelling transformation, such a transformation may not be added.

Error Conditions:

1) Dynamo does not exist.

2) Object name does not exist.

3) Dimensionality mismatch.

4) Object call does not have an associated modelling transformation.

5) Dynamo does not refer to an object call.

6) Invalid attribute routine.

7) Invalid modelling transformation routine.

8) A menu object cannot be referenced by an object call.

C.2.2 VIEWS

An object is displayed by creating a view of the object. Each view has a unique name, and has associated attributes and viewing transformations. Views may be added to and deleted from the display data structure, and the image of the viewed object may be altered by changing the associated attributes and viewing transformations.

```
ADDVIEW(INTEGER VALUE objname, viewname, [dynamo]; [PROCEDURE
      attr]; [PROCEDURE vtrans])
```

adds a view with the name "viewname" of the specified object to the display data structure. A dynamo may be specified for modification of the view. The attributes and the viewing transformations are specified by "attr" and "vtrans", respectively, and are discussed in Sections C.5 and C.3 below.

If no attributes or viewing parameters are specified the null attributes and the default viewing transformation will be applied.

Error Conditions:

1) Object objname does not exist.
2) Invalid name; must be a positive integer.
3) Viewname already exists.
4) Dynamo name already exists.
5) Invalid attribute routine.
6) Invalid viewing transformation routine. (A default transformation has been used.)

CHGVIEW(INTEGER VALUE dynamo, [objname]; [PROCEDURE attr]; [PROCEDURE vtrans])

changes the view pointed at by "dynamo". This routine can change the view to refer to the object objname, and can modify all the attributes and the entire viewing transformation. "objname", "attr", and "vtrans" are defined as for ADDVIEW. If an optional parameter of the call is a non-positive integer or is omitted, the corresponding component of the view is left unchanged.

Error Conditions:

1) Dynamo does not exist.

2) Dynamo does not refer to a view.

3) Object "objname" does not exist.

4) Dimensionality mismatch.

5) Invalid attribute routine.

6) Invalid viewing transformation routine. No change was made.

DLTVIEW(INTEGER VALUE viewname)

deletes the specified view from the display data structure.

Error Conditions:

1) Viewname does not exist.

C.2.3 SEGMENTS

An object consists of one or more segments that are made up of primitives and object calls. A segment may be added to or deleted from the open object, have associated attributes, and can be conditionally executed depending on the extent size of the object. A segment can not be reopened once it has been closed, but its attributes, its primitives and object calls may be modified through the use of dynamos.

BEGSEG(INTEGER VALUE name, [dynamo]; [PROCEDURE attr]; [PROCEDURE size])

makes the segment "name" the open segment for adding primitives and object calls. The segment name must be unique within the object, but need not be unique within the data structure. The segment remains the open segment until a call to ENDSEG.

If one or more attributes are omitted, null attributes will be generated.

A dynamo name may be specified for modification of all the segment attributes (see CHGSEG below). The attributes for the primitives are defined by "attr", and are further discussed in Section C.5. "size", as for objects, defines the conditions under which the segment is displayed.

Error Conditions:

1) No more core available for graphical data structure.

2) No open object.

3) Segment name already exists in this object.

4) Invalid name; must be a positive integer.

5) Dynamo name already exists.

6) A segment is already open.

7) Invalid attribute routine.

8) Invalid size routine.

ENDSEG

The open segment is closed, i.e., no more primitives and calls may be added to the segment.

Error Conditions:

1) No open segment.

DLTSEG(INTEGER VALUE name)

deletes the specified segment from the open object.

Error Conditions:

1) No open object.

2) Segment does not exist in the open object.

CHGSEG(INTEGER VALUE dynamo; PROCEDURE attr)

changes all the attributes of the segment with the specified dynamo name. "attr" is defined as for BEGSEG.

Error Conditions:

1) Dynamo does not exist.

2) Dynamo does not refer to a segment.

C.2.4 NAMING

Names are used to identify or modify parts of the graphical data structure. Names may be associated with objects, segments, views, object calls, primitives, attributes, and transformation matrices. A name may be any positive integer.

An <u>object name</u> is used for the purpose of identification and modification of the object. It is defined when the object is defined or declared (see Section C.2.1). An object name must be unique within the display data structure.

A view name is used for identification and deletion of a view of an object. A view name is defined when a view is added to the display data structure, and must be unique within this data structure.

A segment name is used for the purpose of identification and deletion of a segment within an object. It is defined at segment definition time. A segment name must be unique within an object, but need not be unique within the entire graphical data structure. A segment is, however, always uniquely defined by the name pair: (object, segment).

Theseus provides one level of naming within segments: individual primitives and calls may be named for identification purposes (see Section C.6). Such element names are specified as optional parameters of procedures that generate calls (see Section C.2.1) and graphical primitives (see Section C.4). An element name need not be unique within a segment.

Dynamo names (or dynamos) are unique pointers to items such as segments, primitives, views, object calls, or to individual transformations in a modelling transformation in the graphical data structure, and are used to modify these items dynamically. A dynamo can be a shorthand for the pair (object, segment), or for the triplets (object, segment, element) or (object, segment, matrix). A dynamo is created either when segments, object calls, views, and primitives are created, or when such items have been identified by a picking device. (See Sections C.2, C.4, and C.6.) Dynamos should be deleted by the user when the referenced item no longer is going to be modified.

DLTDYN(INTEGER VALUE dynamo)

deletes the specified dynamo.

Error Conditions:

1) Dynamo does not exist.

C.2.5 EXTENTS

An extent of an object is a rectangular window which surrounds the object and whose edges are parallel to the axes of the local coordinate system. An extent is optionally specified as part of an object definition. When the object call is executed by the DPU, the extent of the called object is transformed by the current transformation matrix. A new extent which surrounds the transformed extent, but whose edges are parallel to the transformed coordinate axes, is calculated. The extent aids in the clipping process in that an object whose transformed extent lies entirely inside or outside the clip window can be trivially accepted or rejected, respectively.

An object can be conditionally executed depending on the size of its transformed extent. A segment can be conditionally executed depending on the size of the transformed extent of the surrounding object. This makes it possible to define "levels of detail" of an object with each level being one segment.

EXTENT2D(REAL VALUE xcenter, ycenter, xsize, ysize)

EXTENT3D(REAL VALUE xcenter, ycenter, zcenter, xsize, ysize, zsize)

specify the center and size of the extent window in local coordinates. The dimensions of the extent window are xsize*2 by ysize*2 (by zsize*2) and should be defined such that they surround the entire object. This routine is optionally specified at object creation time. An existing extent of an object may be modified, but an extent cannot be added to an object once the object is created.

Error Conditions:

1) Object reopened with an extent but no extent was specified when the object was first created.
2) This procedure can only be used as part of OPOBJ2D and OPOBJ3D.
3) Dimensionality mismatch.
4) Numeric data out of range.

5) No more core available for graphical data structure.

SIZE(REAL VALUE min, max)

is optionally specified at object and segment creation. The object or segment is displayed only if the ratio of the size of the extent of the object (for a segment: the object containing the segment) to the length of the display screen diagonal is in the range "min" to "max". The size of the extent of the object is calculated as the length of the diagonal of an extent surrounding the transformed extent after that surrounding extent has been mapped onto the display screen. Note that "min" and "max" are always in the range 0.0 to 1.0.

Error Conditions:

1) Numeric data out of range.
2) Min must be smaller than max.
3) This procedure can only be used as part of OPOBJ2D, OPOBJ3D, and BEGSEG.
4) Object has no extent. (An attempt was made to add a size to the object or one of its segments.)
5) An existing object without an associated SIZE was reopened with a non-null size routine.
6) No more core available for graphical data structure.

C.3 Transformations

Theseus provides both modelling and viewing transformations. Modelling transformations allow objects to be defined in local coordinate systems and to be combined into new objects. These transformations include rotation, scaling, and translation. In addition, the user may specify an arbitrary 4x3 or 3x2 transformation matrix to obtain any nonstandard modelling transformation.

The viewing transformations determine how objects are mapped onto the display surface. In 2D, a window determines which portion of the world coordinate space is to be viewed, and a viewport controls the placement of the image of the object on the display surface. In 3D, a view volume determines which part of the world coordinate space is to be viewed, a method of projection defines how the object is projected onto a plane, and a viewport controls the placement of the image of the projected object on the display surface.

Theseus uses a left-handed coordinate system. The world coordinate system has the origin at (0.0,0.0,0.0) and a range specified by the RANGE function.

C.3.1 RANGE OF WORLD COORDINATES

The world coordinate data and the transformations are specified as real fullwords. The DPU, however, only accepts coordinate data in the range -1.0 to 1.0. The mapping from world coordinate space to the DPU coordinate space must be specified by the user.

RANGE(REAL VALUE xyzscale)

specifies the range of the x, y, and z coordinates: -xyzscale <= x,y,z <= xyzscale. The default is xyzscale = 1. This procedure should generally only be invoked once in each program, before the first invocation of a procedure that uses coordinate data. Note, if the range is changed after some coordinate data has been created, a locate (see Section C.6) in world coordinate space might return erroneous data.

Error Conditions:

1) Parameter must not be 0.0.

C.3.2 MODELLING TRANSFORMATIONS

Modelling transformations are specified at object call generation, and their scope is thus the called object. A modelling transformation is specified as a sequence of modelling transformation function invocations. The matrices corresponding to the modelling transformation function invocations are composed in the order in which they are specified by premultiplying the new transformation matrix with the existing matrix. If no dynamos are specified in the function invocations, the composition takes place in the CPU; if, however, a dynamo is specified, the composition takes place in the DPU each time the SDF is executed. Not only can a modelling transformation be modified by modifying one of its component matrices, but the entire transformation can be altered dynamically as was discussed in Section C.2.1.

SCALE2D(REAL VALUE xscale, yscale; INTEGER VALUE [dynamo])

SCALE3D(REAL VALUE xscale, yscale, zscale; INTEGER VALUE [dynamo])

specifies a modelling transformation matrix with x, y, and z scale factors. A dynamo may be specified if the matrix is to be modified. The scale factors must be in the range 0.0 to 1.0. The default scale is xscale = yscale = zscale = 1.0.

Error Conditions:

1) Dimensionality mismatch.
2) This procedure can only be specified as part of an object call.
3) Numeric data out of range.
4) Dynamo name already exists.

TRANS2D(REAL VALUE xtrans, ytrans; INTEGER VALUE [dynamo])

TRANS3D(REAL VALUE xtrans, ytrans, ztrans; INTEGER VALUE [dynamo])

specifies a modelling transformation matrix with x, y, and z translation amounts. A dynamo may be specified if the matrix is to be modified. The default translation is xtrans = ytrans = ztrans = 0.0.

Error Conditions:

1) Dimensionality mismatch.
2) This procedure can only be specified as part of an object call.
3) Numeric data out of range.
4) Dynamo name already exists.

ROT2D(REAL VALUE angle; INTEGER VALUE [dynamo])

specifies a modelling transformation matrix for a 2D rotation of "angle" radians. A dynamo may be specified if the matrix is to be modified. The default rotation angle is 0.0.

Error Conditions:

1) Dimensionality mismatch.
2) This procedure can only be specified as part of an object call.
3) Dynamo name already exists.

ROT3D(REAL VALUE xcos, ycos, zcos, angle; INTEGER VALUE [dynamo])

specifies a modelling transformation matrix for a 3D rotation of "angle" radians about the vector defined by the direction cosines (xcos, ycos, zcos). A dynamo may be specified if the matrix is to be modified. The default rotation angle is 0.0.

Error Conditions:

1) Dimensionality mismatch.
2) This procedure can only be specified as part of an object call.

3) Numeric data out of range.

4) Dynamo name already exists.

TRANMAT(REAL ARRAY mat(*,*); INTEGER VALUE [dynamo])

allows the user to specify an arbitrary 3x2 (2D) or 4x3 (3D) modelling transformation matrix to be composed with the matrix specifying the modelling transformation. A dynamo may be specified if the matrix is to be modified.

Error Conditions:

1) Dimensionality mismatch.

2) This procedure can only be specified as part of an object call.

3) Illegal array bound.

4) Dynamo name already exists.

CHGTRAN(INTEGER VALUE dynamo; PROCEDURE mtrans)

changes a matrix which is a component of a modelling transformation that has the specified dynamo name. "mtrans" is one of the modelling transformation functions.

Error Conditions:

1) Dynamo name already exists.

2) Dynamo does not refer to the correct type of modelling transformation matrix.

C.3.3 VIEWING TRANSFORMATIONS

The viewing transformations are specified when a view is added to or changed in the display data structure, and define how the object is mapped to the display surface. A viewing transformation may be changed by modifying individual viewing parameters.

The 2D viewing transformations of Theseus are very similar to those of the Core Graphics System [GSPC77]. In 2D the viewing transformation is defined by a window and a viewport. The 3D viewing parameters provide the same functional capabilities as the Core Graphics System, but are specified in a different manner. For perspective projections the viewing transformation is defined by a view volume, a view plane, and a viewport, where the view volume is a pyramid. For parallel projections the viewing transformations are specified by a window, a viewport, a projection plane, and by the desired properties of the projected object.

VRP(REAL VALUE x,y,z)

defines the view reference point in world coordinates. Some 3D viewing parameters are defined relative to this point. The default point is (0.0,0.0,0.0).

Error Conditions:

1) This procedure can only be specified as part of ADDVIEW, CHGVIEW, or CHGVTRAN.

2) Dimensionality mismatch.

3) Numeric data out of range.

ORIENT(REAL VALUE nx,ny,nz,ux,uy,uz)

defines the view plane normal (nx,ny,nz) and the "viewup" direction (ux,uy,uz) as vectors relative to the view reference point.

Viewup is a vector whose orthographic projection onto the view plane defines the direction in the view plane that will be "up" on the display surface (except for plan oblique views; see the next paragraph). The viewup vector allows the definition of a coordinate system called the UVN-system. The orthographic projection of the viewup vector defines the V-axis. The U-axis is defined such that the U-axis, the

V-axis and the view plane normal, N, form a left-handed coordinate system.

In the case of plan oblique views, the viewup vector defines a "preferred direction" whose orthographic projection (on the view plane) will be at an angle (PI/2 + gamma) radians from the up direction of the screen. The angle gamma is the angle of the receding axis. The preferred direction allows the definition of a new coordinate system called the U'V'N-system. The orthographic projection of the preferred direction defines the V'-axis. The U'-axis is defined such that the U'V'N-system is left-handed. The UVN-system is obtained by a rotation of (PI/2 + gamma) of the U'V'N-system about the N-axis.

The window and the center of projection are defined in the UVN-system.

The default view plane normal is (0.0,0.0,1.0), and the default viewup direction is (0.0,1.0,0.0).

Error Conditions:

1) This procedure can only be specified as part of ADDVIEW, CHGVIEW, or CHGVTRAN.
2) Dimensionality mismatch.
3) All elements of a direction vector are zero. (The default has been used.)

4) Numeric data out of range.

5) Viewup direction is parallel to the view plane normal. (The defaults are assumed.)

ELEVOBL(REAL VALUE len,gamma)

PLANOBL(REAL VALUE len,gamma)

define the foreshortening ratio "len" of the receding lines and the angle "gamma" between the U-axis and the receding lines for an elevation oblique, and the foreshortening ratio "len" and the angle "gamma" between the U'-axis and the receding axes for a plan oblique. "receding lines" are the projection of lines parallel to the view plane normal. "len" and "gamma" define the direction of the projectors of a parallel projection. "len" should be positive, "gamma" is measured in radians.

Error Conditions:

1) This procedure can only be specified as part of ADDVIEW, CHGVIEW, or CHGVTRAN.

2) Dimensionality mismatch.

3) The foreshortening ratio is zero. (One has been used.)

ORTHO

defines an orthographic projection. The projectors are parallel to the view plane normal. The default projection is orthographic.

Error Conditions:

1) This procedure can only be specified as part of ADDVIEW, CHGVIEW, or CHGVTRAN.
2) Dimensionality mismatch.

PERSP(REAL VALUE x,y,z)

defines the center of projection of a perspective projection in the UVN-coordinate system.

Error Conditions:

1) This procedure can only be specified as part of ADDVIEW, CHGVIEW, or CHGVTRAN.
2) Dimensionality mismatch.
3) Numeric data out of range.

VIEWUP2D(REAL VALUE dx,dy)

VIEWUP2D defines the direction in the world coordinate system that will be "up" on the display surface. The vector (dx,dy) is defined relative to the origin of the world coordinate system. VIEWUP2D defines a rotation of the window about the origin.

The default viewup vector is (0.0,1.0).

Error Conditions:

1) This procedure can only be specified as part of ADDVIEW, CHGVIEW, or CHGVTRAN.
2) Dimensionality mismatch.
3) All elements of the direction vector are zero. (The default has been used.)
4) Numeric data out of range.

WINDOW2D(REAL VALUE xcenter, ycenter, xsize, ysize)

defines the center and size of the clip window for a 2D viewing transformation. The width and height of the window are xsize*2 by ysize*2.

The center and size are defined in the world coordinate system. The rectangle whose center and size are the parameters to WINDOW is rotated about the origin of the world coordinate system so that the vertical sides of the window become parallel to the viewup vector.

The window and the viewport define the mapping of the 2D world coordinate space to the display surface. If clipping is enabled, only those parts of the object inside the window will be mapped to the viewport. If clipping is disabled, the entire world coordinate space is mapped to the display surface in such a way that the window is mapped to the viewport. The default window is (0.0, 0.0, xyzscale, xyzscale), where xyzscale defines the range of the world coordinate system.

WINDOW3D(REAL VALUE xcenter, ycenter, zcenter, xsize, ysize)

defines the size of the clip window in the UVN-system and the center of the clip window in the UVN-coordinate system for perspective, orthographic, and elevation oblique projections, and in the U'V'N-system for plan oblique projections. The window center is defined relative to the center of projection for perspective and relative to the origin of the UVN-system or U'V'N-system for parallel projections. The center defines both the distance to the view plane ("zcenter") and the position of the window within the view plane ("xcenter" and "ycenter"). The sides of the window are always parallel to the U and V directions (not to the U' and V' directions); the lengths of the sides of the window are 2*xsize and 2*ysize. xsize is defined along the U-axis, ysize along the V-axis.

If a perspective projection is selected, the center of projection and the window define a semi-infinite pyramid. If a parallel projection is selected, the window and the direction of the projectors define a semi-infinite paralellepiped. If clipping is enabled, only the contents of the view volume, the pyramid or the parallelepiped, are projected onto the window in the view plane. If clipping is disabled, the entire world coordinate space is projected, by a parallel or perspective projection, onto the view plane. The view plane is then mapped to the display surface in such a way that the window is mapped to the viewport.

The default window is (0.0, 0.0, 0.0, xyzscale, xyzscale), where xyzscale defines the range of the world coordinate system.

Error Conditions:

1) This procedure can only be specified as part of ADDVIEW, CHGVIEW, or CHGVTRAN.
2) Numeric data out of range.

VIEWPORT(REAL VALUE xcenter, ycenter, xsize, ysize)

defines the center and size of the viewport in screen coordinates, i.e., fractional coordinates between -1.0 and 1.0. The dimensions of the viewport are xsize*2 by ysize*2. The default viewport is the entire screen, i.e., (0.0,0.0,1.0,1.0).

Error Conditions:

1) This procedure can only be specified as part of ADDVIEW, CHGVIEW, or CHGVTRAN.

2) The viewport extends outside the screen area. (The default has been used.)

VTRANMAT(REAL ARRAY mat(*,*))

allows the user to specify an arbitrary 3x2 (2D) or 4x3 (3D) viewing transformation matrix. Note that if this matrix is specified with other viewing functions, unpredictable effects may result.

Error Conditions:

1) Dimensionality mismatch.

2) This procedure can only be specified as part of a view.

3) Illegal array bound.

CHGVTRAN(INTEGER VALUE dynamo; PROCEDURE vtrans)

changes one or more viewing parameters in the view specified by "dynamo". "vtrans" specifies the viewing transformation functions that are to be modified.

Error Conditions:

1) Dynamo does not exist.

2) Dynamo does not refer to a view.

3) Invalid viewing transformation function requested. (No change was made.)

C.4 Primitives

Theseus provides line, point and text primitives. The primitives are added to the open segment, and are specified in absolute coordinates of a coordinate system local to the object. The line coordinate data added to the segment consists either of immediate data, i.e., the actual coordinate values, or of addressed data, i.e., a pointer to the coordinate data. Each primitive may have an associated element name for identification purposes, and a primitive specified as immediate data may have an associated dynamo name for modification purposes.

The text primitives are part of two- or three-dimensional objects. Characters in the text strings are individually positioned, conforming to the user's specification as closely as the hardware character generator permits.

Individual primitives may not be deleted from a segment. They may, however, be altered either through an associated dynamo, or by a change in the coordinate location pointed to by an address in the SDF. The type of the primitive may not be changed, but a point and an endpoint of a line may be replaced by another point, and a text string with another text string. Note that the length of a primitive, i.e. the number of lines or the number of characters, must remain unchanged.

Menu text is used for operator/machine interaction. Menus are objects that can be added to the display data structure by creating a view of the menu object. A menu is displayed in a user-defined viewport, but the user has no control over size and position of the menu text within the viewport.

C.4.1 LINE AND POINT PRIMITIVES

Theseus provides procedures for simple primitives such as points and lines, and for composite primitives such as polylines and polygons.

LINE2D(REAL VALUE x1, y1, x2, y2; INTEGER VALUE [elemname], [dynamo], linestyle)

LINE3D(REAL VALUE x1, y1, z1, x2, y2, z2; INTEGER VALUE [elemname], [dynamo], linestyle)

LINEADDR(REAL VALUE p1, p2; INTEGER VALUE [elemname], linestyle)[28]

specify a line from (x1, y1) to (x2, y2) of a two-dimensional object, a line from (x1, y1, z1) to (x2, y2, z2) of a three-dimensional object, and a line from the coordinates at p1 to the coordinates at p2, respectively. "linestyle" determines the type of line:

0 - solid

1 - dashed

2 - dotted

An element name may be specified for identification of the primitive. A dynamo name may be specified for modification of the primitive, if it is specified via immediate data.

[28]Algol W does not provide a pointer data type; p1 and p2 are the actual location of the X-coordinate values, and the address of p1 and p2 will be stored in the SDF.

Error Conditions:

1) No open segment.

2) No more core available for graphical data structure.

3) Dimensionality mismatch.

4) Numeric data out of range.

5) Invalid line style. (Solid has been used.)

6) Dynamo name already exists.

```
POINT2D(REAL VALUE x, y; INTEGER VALUE [elemname], [dynamo])
POINT3D(REAL VALUE x, y, z; INTEGER VALUE [elemname], [dynamo])
PTADDR(REAL VALUE p; INTEGER VALUE [elemname])
```

specify a point at (x, y), at (x, y, z), and at the coordinates at p, respectively. An element name may be specified for identification of the primitive. A dynamo name may be specified for modification of the primitive, if the primitive is specified via immediate data.

Error Conditions:

1) No open segment.

2) No more core available for graphical data structure.

3) Dimensionality mismatch.

4) Numeric data out of range.

5) Dynamo name already exists.

POLYLINE(REAL ARRAY xyz(*,*); INTEGER VALUE [elemname], [dynamo], linestyle)

PLINADDR(REAL ARRAY xyz(*); INTEGER VALUE [elemname], linestyle)

defines a sequence of connected lines starting at the first coordinate point in xyz and ending at the last. In POLYLINE, xyz contains immediate data; in PLINADDR, xyz contains addressed data. The dimensionality is determined by the first array index, which is either 2 or 3 depending on whether a 2D or 3D polyline is to be generated.

"linestyle" is defined as for line primitives.

An element name may be specified for identification of, and a dynamo name for modification of the primitive.

Error Conditions:

1) No open segment.

2) No more core available for graphical data structure.

3) Dimensionality mismatch.

4) Numeric data out of range.

5) Dynamo name already exists.

6) Invalid line style. (Solid has been used.)

POLYGON(REAL ARRAY xyz(*,*); INTEGER VALUE [elemname], [dynamo], linestyle)

PGONADDR(REAL ARRAY xyz(*); INTEGER VALUE [elemname], linestyle)

are identical to POLYLINE and PLINADDR except that the first and the last coordinate points in xyz are connected.

CHGPT2D(INTEGER VALUE dynamo, coordoffset; REAL VALUE x, y)

CHGPT3D(INTEGER VALUE dynamo, coordoffset; REAL VALUE x, y, z)

modify one coordinate in a two- and three-dimensional primitive, respectively. The primitive is referenced by the dynamo, and the actual point within the primitive is determined by "coordoffset", which is an offset into the set of coordinates that determine the primitive. "coordoffset" is always one for a point primitive or the position of a text primitive, one or two for a line primitive, and an offset into an array for polylines and polygons.

Error Conditions:

1) Dynamo does not refer to a segment.

2) Dimensionality mismatch.

3) Numeric data out of range.

4) Dynamo does not exist.

5) Coordinate set number out of range.

C.4.2 TEXT

```
TEXT2D(REAL VALUE x, y, dx, dy; STRING (120) VALUE chars; INTEGER
     VALUE length, [elemname], [dynamo])
TEXT3D(REAL VALUE x, y, z,  dx, dy, dz; STRING (120) VALUE chars;
     INTEGER VALUE length, [elemname], [dynamo])
```

defines a text string of length "length" with the center of the first character at (x, y) and (x, y, z), respectively. The inter-character distance is defined in world coordinate units by (dx, dy) or (dx, dy, dz) and defines the coordinate position for the center of each of the following characters. The coordinate positions are transformed by the composite transformation matrix, and the individual characters are displayed, in the screen plane, with the center of each character at the transformed character position. A character size appropriate for the transformed string length is chosen by Theseus.

An element name may be specified for identification of the primitive, and a dynamo name may be specified for modification.

Error Conditions:

1) No open segment.

2) No more core available for graphical data structure.

3) Dimensionality mismatch.

4) Numeric data out of range.

5) Text length not in the range 0 to 120; 120 has been used.

6) Dynamo name already exists.

CHGTEXT(INTEGER VALUE dynamo, length; STRING (120) VALUE chars)

replaces the text string referenced by the specified dynamo. Note that if the new string is longer than the original string it will be truncated; if it is shorter it will be padded with blanks.

Error Conditions:

1) Dynamo does not refer to a segment.

2) Dynamo does not exist.

3) Text length is either less that zero, or greater than length of string being modified. (It has been set to the length of the existing string.)

C.4.3 MENU TEXT

```
MENU( INTEGER  VALUE name;  STRING (120)  ARRAY (*) chars; INTEGER
     VALUE length, elemname1)
```

defines a menu object with the name "name". A menu is a special object containing text strings and can only be referenced from ADDVIEW or CHGVIEW. The text strings are displayed in the viewport defined by the viewing transformation. All other viewing parameters of the view are ignored. A menu is treated like all other objects. It can be added to the SDF, deleted from it, and its name can be replaced and swapped.

"length" is the length of the longest string in "chars", and "elemname1" defines the element name of the first menu text string in "chars". The following strings have the element names "elemname1"+1, "elemname1"+2, etc.

Each string will be displayed preceded by a bullet. A text string is identified by a pick. Upon a pick of a menu item, the attention queue will contain the triplet (name,0,elemname), and the picked text string will be highlighted.

Error Conditions:

1) Object name already exists.

2) No more core available for graphical data structure.

3) Invalid name; must be a positive integer.

4) Text length not in the range 0 to 120; 120 has been used.

5) A menu can only be referenced from ADDVIEW and CHGVIEW.

C.5 Attributes

Attributes allow the user to manipulate the images of objects and segments. These attributes are intensity, pickability, blink, vector mode, clip enable, extent enable, and size enable. The attributes are specified at generation of object calls, views, and segments. All the attributes of a call, view, or segment can be altered by the procedures CHGCALL, CHGVIEW, and CHGSEG, respectively, and one or more individual attributes can be changed by CHGATTR.

When the hierarchical data structure is processed by the DPU, attributes operate such that values at different levels combine. Rules for these combined effects are specified below. If an attribute specification is omitted when a view, object call, or segment is generated, a null attribute is applied. A null attribute has no effect on higher- or lower-level, non-null attributes. At the highest level all attributes have default values, specified below.

CHGATTR(INTEGER VALUE dynamo; PROCEDURE attr)

changes the specified attributes of the segment, object call, or view that has the specified dynamo name. "attr" is defined as for BEGSEG.

Error Conditions:

1) Dynamo does not exist.

2) Dynamo does not refer to an attribute.

INTENS(REAL VALUE intens)

defines the intensity at which the image of a object or a segment is displayed. "intens" can range from 0.0 to 1.0; 0.0 is the dimmest, 1.0 the brightest. When attributes are composed at a subobject call or at the beginning of a segment, the maximum of the two intensities is chosen as the new attribute. The default is intens = 0.5.

Error Conditions:

1) Attributes can only be part of an object call, a view, or a segment.

2) Numeric data out of range.

PICKABLE(LOGICAL VALUE pick)

This attribute, when <u>true</u>, enables the object or segment for picking (see Section 6). When attributes are composed, the <u>or</u> of the two attributes is chosen as the new attribute. This implies that when an object is pickable, so are all its segments and subobjects. The default is pick = <u>true</u>.

Error Conditions:

1) Attributes can only be part of an object call, a view, or a segment.

BLINK(LOGICAL VALUE blink)

This attribute, when <u>true</u>, causes the image of an object or segment to blink until the attribute is set to <u>false</u>. When the attributes are composed, the <u>or</u> of the two attributes is chosen as the new attribute. This implies that when an object is blinking, so are all its segments and subobjects. The default is blink = false.

Error Conditions:

1) Attributes can only be part of an object call, a view, or a segment.

VECMODE(LOGICAL VALUE vecmode)

This attribute, when true, changes the line style for the image of an object or a segment. Solid lines will change to dashed, dashed to dotted, and dotted lines to endpoints. When attributes are composed the or of the two attributes is chosen as the new attribute. The default is vecmode = false.

Error Conditions:

1) Attributes can only be part of an object call, a view, or a segment.

CLIP(LOGICAL VALUE clip)

allows the user to disable clipping of an object or segment against the window or view volume. When the attributes are composed the and of the two attributes is chosen as the new attribute. The default is clipping enabled, i.e., clip = true.

Error Conditions:

1) Attributes can only be part of an object call, a view, or a segment.

EXTCON(LOGICAL VALUE extcon)

allows the user to disable extent processing. If disabled, the DPU will process each primitive in an object for clipping. This attribute is mainly used for debugging. When the attributes are composed, the and of the two attributes is chosen as the new attribute. The default is extent processing enabled, i.e., extcon = true.

Error Conditions:

1) Attributes can only be part of an object call, a view, or a segment.

SIZECON(LOGICAL VALUE sizecon)

allows the user to disable the conditional test that determines if an object is too small or too large, relative to the object extent size, to be processed for display. If disabled, the segment or object will be processed regardless of its size. This attribute is used for debugging. When the attributes are composed, the <u>and</u> of the two attributes is chosen as the new attribute. The default is size processing enabled, i.e., sizecon = <u>true</u>.

Error Conditions:

1) Attributes can only be part of an object call, a view, or a segment.

<u>C.6 Interaction</u>

Theseus supports five classes of logical interaction devices:

PICK - identify object, segment, or primitive

BUTTON - select function

KEYBOARD - provide alphanumeric information

LOCATOR - provide coordinate information

VALUATOR - provide values

These logical devices may be implemented by one or more physical devices, and the programmer may choose which physical interaction device most conveniently implements a logical device for his application.

The logical devices are divided into two classes: event causing devices and sampling devices. Each event causing device has an associated one-element event queue. Upon an event, an event report with data related to the event is placed on this queue. The event report must be removed from the queue by the application program before a new event report may be placed on the queue. The application program may poll any queue, or wait for an event from one or more devices. Sampling devices have values that may be sampled by the application program. The pick, button, and keyboard are event causing devices, the valuator and locator sampling devices.

Each logical device and its physical implementation is described below. The event handling functions for polling and waiting, and for accessing event reports are described last.

C.6.1 PICK

A pick device is an event causing device that may identify a view, object, segment, or primitive in the data structure. The event report contains the DPU object call stack of (object, segment, elemname) triplets at the time of the pick. The programmer may obtain any triplet in the stack for identification purposes, or assign a dynamo to any item in the stack for modification purposes. If viewports overlap, a pick through the use of a cursor will occur in the first viewport whose data is executed by the DPU.

The position of the pick window may be explicitly controlled by the programmer or implicitly controlled by attaching it to the joystick or to the tablet.

PICKEXPL(REAL VALUE x, y, xscale, yscale)

enables the logical pick device for explicit picking. A pick-window of dimensions xscale*2 by yscale*2 is initially positioned at (x,y). x, y, xscale, and yscale are all specified in screen coordinates.

Error Conditions:

1) A pick device is already enabled.

2) Numeric data out of range.

3) A locator device is already enabled.

POSPICK(REAL VALUE x, y)

moves the center of the pick window to (x, y). x and y are specified in screen coordinates.

Error Conditions:

1) Explicit pick device has not been enabled.

2) Numeric data out of range.

PICKIT

performs an explicit pick at the current position of the pick window. If the pick is successful, the event report is placed on the pick event queue, unless there is an item in the queue already. This function and POSPICK allow the programmer to use any combination of sampling and event causing devices to perform a pick. Note, this function may be used to perform a pick when the joystick or the tablet is enabled as the pick device.

Error Conditions:

1) A pick device is not enabled.

PICKJOY(INTEGER VALUE fnkey, REAL VALUE xscale, yscale)

enables the joystick as a pick device. A pick-window of dimensions xscale*2 by yscale*2 is positioned at the current (x,y) joystick coordinates. The function key specified by "fnkey" is implicitly enabled and serves as a trigger. If the pick is successful, the event report is placed on the pick event queue, unless there is already an item in the queue. "xscale" and "yscale" are specified in screen coordinates.

Error Conditions:

1) A pick device is already enabled.

2) Function key is already enabled.

3) Numeric data out of range.

4) A locator device is already enabled.

5) Invalid function key number.

PICKTAB(REAL VALUE xscale, yscale)

enables the tablet as a pick device. A pick window of dimensions xscale*2 by yscale*2 is positioned at the current (x,y) tablet coordinates. The pick occurs when the tip-switch is pressed. If the pick is successful, the event report is placed on the event queue, unless there is an item in the queue already. "xscale" and "yscale" are specified in screen coordinates.

Error Conditions:

1) A pick device is already enabled.

2) Numeric data out of range.

3) A locator device is already enabled.

DISPICK

disables the current pick device, and flushes the queue for the pick device.

Error Conditions:

1) A pick device is not enabled.

C.6.2 BUTTON

A button is an event causing device that allows the programmer to select a function. The event report contains the number of the button. The buttons are implemented as function keys.

BUTKEY(INTEGER VALUE number)

enables the specified function key as a button, and lights the corresponding function key light.

Error Conditions:

1) Function key is already enabled.

2) Invalid function key number.

BUTKEYS

enables all function keys as buttons, except for any function key that is already enabled as a trigger. The corresponding function key lights are lit.

DISBUT(INTEGER VALUE number)

disables the specified button, and flushes the button queue.

Error Conditions:

1) Button is not enabled.

2) Invalid function key number.

DISBUTS

disables all buttons, and flushes each button queue.

C.6.3 KEYBOARD

The keyboard is an event device that allows the user to enter alphanumeric text. The keyboard event occurs when the carriage return is typed, and the event report contains the text string that was entered and the number of characters in that string.

Theseus maintains and displays a one-line character buffer for prompt text and for input from the alphanumeric keyboard. The prompt text and the input text are displayed in a programmer-defined viewport at a programmer-defined cursor position.

KEYVIEW(REAL VALUE xcenter, ycenter, xsize, ysize)

specifies the center and size of the text input viewport in screen coordinates. The dimensions of the viewport are xsize*2 by ysize*2. The default viewport is (0.0,0.0,1.0,1.0), i.e. the entire screen.

Error Conditions:

1) Numeric data out of range.
2) The viewport extends outside the screen area. (The default has been used.)

KEYBUF(INTEGER VALUE length)

specifies the total buffer size for the combined prompt message/input line. The maximum allowable buffer size is 120, and the default is 80 characters. The buffer size and the viewport size determine the size of the characters; the character size is chosen so that the entire buffer will fit on one line in the viewport. If the user tries to input beyond the length of the buffer the input is ignored.

Error Conditions:

1) Parameter out of range.

PROMPT(REAL VALUE x, y; STRING (120) VALUE message; INTEGER VALUE length)

enables the alphanumeric keyboard for input. A prompt message "message" of length "length" characters followed by a cursor is displayed at the coordinate position (x,y). (x,y) are specified in the range -1.0 to 1.0, and are coordinates in the input viewport defined by KEYVIEW. Upon carriage return the event report is placed on the event queue, and the keyboard is disabled for input.

Error Conditions:

1) Numeric data out of range.

2) Text length is either less than zero, or greater than the maximum buffer size. (It has been set to the maximum buffer size.)

C.6.4 LOCATOR

A locator is a sampling device that provides coordinate information in screen coordinates. The locator may either be controlled by the joystick or the tablet.

LOCJOY

enables the joystick as a sampling device. A cursor is displayed at the current (x,y) joystick position.

Error Conditions:

1) A locator device is already enabled.

2) A pick device is already enabled.

LOCTAB

enables the tablet as a sampling device. A cursor is displayed at the current (x,y) tablet position.

Error Conditions:

1) A locator device is already enabled.

2) A pick device is already enabled.

READLOC(REAL RESULT x, y)

reads the current locator. (x,y) is returned in screen coordinates.

Error Conditions:

1) A locator device is not enabled.

DISLOC

disables the locator.

Error Conditions:

1) A locator device is not enabled.

C.6.5 VALUATOR

A valuator is a sampling device that allows the user to read numeric values in the range 0.0 to 1.0. There can be several one-dimensional valuators--the dials--and 2 three-dimensional valuators--the joystick and the tablet.

READDIAL(INTEGER VALUE first, number; REAL ARRAY RESULT dial (*))

returns the value of "number" dials, starting with dial "first", in the array "dial".

Error Conditions:

1) Parameter out of range.

READJOY(REAL RESULT x, y, z)

returns the current (x,y,z) values of the joystick.

READTAB(REAL RESULT x, y, INTEGER RESULT z)

returns the current (x,y,z) values of the tablet. z indicates the position of the tablet pen:

0 - out of range of tablet

1 - in range of tablet

2 - touching tablet.

C.6.6 EVENT HANDLING

This section will discuss how the event queues can be polled, how an application program can wait for an event, and how to access the event reports.

POLL(LOGICAL expr)

is an integer procedure that polls the event queues for one or more devices. "expr" is a boolean expression of the form:

BUTTON OR PICK OR KEYBOARD OR LOCATOR.

Theseus will poll the event queues of the devices in the boolean expression in the order in which they are specified. If an event occurred, an integer equal to the index of the event device in the expression will be returned. If no event occurred, POLL will return the number of devices in the boolean expression plus one. For example, POLL(BUTTON OR LOCATOR OR PICK) would return:

1 for a button event

2 for a locator event

3 for a pick event

4 if no event had occured.

This function removes the corresponding event report from the queue. The dequeued report becomes the current event report for that device, and can be accessed by the functions described below.

Error Conditions:

1) Invalid routine in boolean expression.

WAIT(LOGICAL expr)

works much in the same fashion as POLL, except that WAIT will not return until an event has occurred. The queues will continuously be polled in the order in which the devices are specified in the boolean expression.

Error Conditions:

1) Invalid routine in boolean expression.

STACK

is an integer procedure that returns the depth of the object call stack in the current pick event report.

Error Conditions:

1) Current event report for the pick device does not exist.

GETPICK(INTEGER VALUE depth, INTEGER RESULT object, segment, elemname)

returns the triplet name (object, segment, elemname) at level "depth" from the top of the object call stack in the current pick event report. The top (level1) item of the stack contains (0,viewname,0). The bottom element contains the (object, segment, elemname) triplet for the primitive that was picked. Intermediate levels contain triplets for the intervening calls. If an element name was not specified for an object call or a primitive, an elemname of 0 is returned.

Error Conditions:

1) Current event report for the pick device does not exist.

2) Parameter out of range. (Depth exceeds number of items on the stack.)

DYNELEM(INTEGER VALUE depth, dynamo)

DYNSEG(INTEGER VALUE depth, dynamo)

create a dynamo with the name "dynamo", referring to the item corresponding to the triplet (object, segment, elemname) and the pair (object, segment) at level "depth" from the top of the object call stack in the current pick event report. DYNELEM creates a dynamo for a primitive, a view, or an object call, whereas DYNSEG creates a dynamo for a segment.

Error Conditions:

1) Current event report for the designated device does not exist.
2) Parameter out of range. (Depth exceeds number of items on the stack.)
3) Dynamo name already exists.
4) Invalid name; must be a positive integer.
5) No more core available for graphical data structure.
6) Object name does not exist. (It has been deleted since the pick.)
7) Segment does not exist. (It has been deleted since the pick.)

GETBUT

is an integer procedure that returns the number of the function key that caused the button event.

Error Conditions:

1) Current event report for the designated device does not exist.

GETTEXT(INTEGER RESULT number, STRING (120) RESULT text)

returns the length of input character string in the keyboard event report and the characters in this string.

Error Conditions:

1) Current event report for the designated device does not exist.

C.7 Control

This section discusses a set of unrelated issues regarding the operation of Theseus.

C.7.1 INITIALIZATION

A series of Theseus programs using the same graphical data structure can be executed. Theseus is invoked by executing the following command:

```
THESEUS <filename> .... <filename> [(<options>]
```

<filename> represents a file. The files are executed in the order in which they are specified, and operate on the same graphical data structure. The options can be specified in any order, separated by blanks. Theseus has the following options (in addition to the ALGOL W ones):

SD=x is the maximum depth of the object call stack. The stack depth must not be greater than 10; the default is SD=5.

D=xxxx specifies the dimensionality of the objects as either 2D, 3D or BOTH. The default is DIM=2D.

PCF=xx specifies the amount of core in K-bytes that Theseus uses for the largest object program. Theseus will utilize the remainder of core for the graphical data structure and the Algol W stack frame.

GCF=xx specifies the amount of core in K bytes that Theseus should use for its graphical data structure.

Note that only one of PCF and GCF should be specified. If both are specified then the latter will be used, and if both are omitted the default is GCF=4.

C.7.2 TERMINATION

Theseus is terminated and the space for the graphical data structure freed up at the end of the execution of the last program on a single command line.

C.7.3 CLOCK

The "frame" clock is not treated as an interaction device and is not accessible by Theseus. However, Algol W provides access to the CPU clock.

C.7.4 DEFAULTS

The attributes and the transformations have defaults defined by Theseus. These defaults are applied at view and object call generation if the attributes or transformations are not explicitly specified by the programmer. The defaults may not be altered by the programmer.

C.7.5 PICTURE FILES

Theseus allows a user to save an object on disk and to retrieve it for later use. Each object is saved in a separate disk file.

SAVEOBJ(INTEGER VALUE name, STRING(8) VALUE file)

saves the object "name" and all its subobjects in a disk file named "file". Note that the object is not deleted from the data structure.

Error Conditions:

1) Object name does not exist.

2) File with the specified name already exists.

LOADOBJ(STRING(8) VALUE file) loads the object and all its subobjects from the file with the name "file". Care must be taken that there are no duplicate object names in the disk file and the graphical data structure.

Error Conditions:

1) File with the specified name does not exist.

2) Object name already exists.

C.7.6 ERROR HANDLING

Runtime errors in Theseus are handled as runtime errors in Algol W. The error number and the coordinate of the Theseus statement that caused the error are printed on the operator's console.

There are two types of errors in Theseus: warnings and fatal errors. At a non-fatal error the procedure call will be ignored unless other action is indicated in the error message, and execution will continue. If an error occurs in a Theseus procedure that is passed as a parameter, only that particular procedure will be ignored. For example, if an error occurs in an attribute routine, only that attribute routine is ignored. The other attribute routines as well as the segment, view, or object call definition will be processed.

Note that procedures may return unpredictable values if the procedure generated an error.

There is only one fatal error: "No more core available for graphical data structure."