v

THE BROWN UNIVERSITY GRAPHICS SYSTEM

by

George M. Stabler

The Brown University Graphics Project

CENTER FOR
COMPUTER & INFORMATION SCIENCES
BROWN UNIVERSITY
Providence, Rhode Island

February 15, 1973

TABLE OF CONTENTS

1. PROJECT OVERVIEW

1.1 Introduction

The aim of this document is to provide a unified overview of the current research activities of the Brown University Graphics Project.[1] The stated objectives of the project's activities are an investigation into the area of medium-cost, microprogrammable, intelligent graphics terminals and the "division of labor" trade-offs between a mainframe processor and the intelligent satellite. A high level system implementation language and a facility for online symbolic debugging of graphic data structures are to be provided for system implementers and users. In addition to these goals, we are also interested in examining the impact which microprogramming has on the design of other aspects of a graphics terminal, for example, system configuration and the local operating system design. The following sections give further detail on the motivation and aims of the project.

1.2 Motivation

For the past seven years, the Graphics Project at Brown has been engaged in the development of graphics system and application oriented software. This experience, most of which was gained on the IBM 2250 series of displays, has lead us to certain conclusions.

1.2.1 The facilities offered by the IBM 2250 display series are woefully inadequate for a large variety of applications, particularly those requiring sophisticated real-time interaction, arbitrary three-dimensional transformations, large or structured display files, or analog inputs for display control (e.g., smooth windowing and zooming on a large drawing with a joystick, tumbling a wire frame object in three space using a bowling ball).

---

[1]This research is being supported by the National Science Foundation, grant GJ-28401X, the Office of Naval Research, contract N00014-67-A-0191-0023, and the Brown University Division of Applied Mathematics; Principal Investigator: Andries van Dam. A list of publications resulting from this research is presented in Appendix F.

1.2.2 To date only one display has come close to alleviating all of these deficiencies: the Evans and Sutherland display system [6]. However, the E&S system has the disadvantages of being expensive and, because it uses hardwired logic, inflexible for fundamental investigations in architecture tradeoffs.

We believe in in an "intelligent terminal" approach, in which a small computer, capable of a nontrivial amount of local processing, is juxtaposed between the display and the mainframe computer.[2] We feel such local processing is necessary in our environment, as well as in most industrial and government installations, because the resources (core, CPU) of the mainframe computer are not available on a fulltime, high priority basis. Furthermore, we anticipate that the local processor can be made sufficiently powerful to handle the majority of interactions between a user and his graphics program. Consequently only occasional service (e.g., data base retrieval or large scale floating point calculations) should be demanded of the mainframe computer.

1.2.3 Finally, much work has yet to be done in the area of developing a powerful high level display instruction set. There are several areas (interrupt handling, arithmetic and logical capabilities, display file/user program synchronization, graphic device control) in which the current repertoire of display instructions should be expanded.

1.3 Experimental Aims

The design of the configuration described below was, of course, influenced by such factors as cost, available funds, interfacing capabilities, etc. However, a more important factor in our investigation into possible configurations was the requirement that the system be capable of fulfilling our research needs. These research requirements are as follows.

1.3.1 The capability for stand-alone operations. The satellite processor must be powerful enough to allow significant graphic operations to run without

---

[2]Note that E&S have now partially come around the "Wheel of Reincarnation" [11] with us by enhancing their display processor with a general purpose instruction set and allowing it to be interfaced to other computers [7]. However the E&S system still relies on mainframe memory for storage of instructions and graphic data.

interacting with the mainframe computer (an IBM S/360 Model 67). Even for programs which require the facilities of the mainframe, it should be possible to carry out a fair amount of local processing (e.g., data base editing) without recourse to the mainframe.

1.3.2 Research into graphic instruction sets. It must be possible to completely define (or redefine) the instruction set for the graphic processor or display system. We should have the widest possible latitude to implement and experiment with those features which should be included in a high-level graphics instruction set, for example, the direct display of graphic data from complex data structures.

1.3.3 The ability to implement a wide range of transformational capabilities. These transformations include full three-dimensional perspective display (homogeneous coordinates), dynamic (real time) transformations such as pan and zoom, and window and viewport operations.

1.3.4 Emulation of other display systems. It should be possible to use the system to emulate other display systems, such as the IBM 2250 Models 1, 3, and 4, to provide a smooth transition between existing applications programs and new ones taking proper advantage of the new features.

1.3.5 Interconnected Processing. The architecture of the satellite processor should be such as to facilitate the transfer of programs and data between it and the mainframe computer. This is in anticipation of "division of labor" studies to be carried out into the optimal assignment of the subtasks of a large graphics application between the two processors.

1.3.6 Symbolic Debugging. Creating programs involves two operations. First, the syntactic definition of the data items to be manipulated and the semantic relationships, if any, among these data items must be defined. Second, the method of transforming this data from its initial to its final form, as embodied for example in a program, must be enumerated. Debugging programs involves the availability of effective, low-overhead, user-oriented methods that provide both compile- and execution-time facilities for data and program examination and alteration. The system must facilitate the development of such tools.

1.3.7. High Level Language Support. The instruction set of the satellite processor should be such as to facilitate the devel opment of a compiler capable of generating code for both the satellite and the main frame.[3]

2. HARDWARE CONFIGURATION

2.1 System Overview

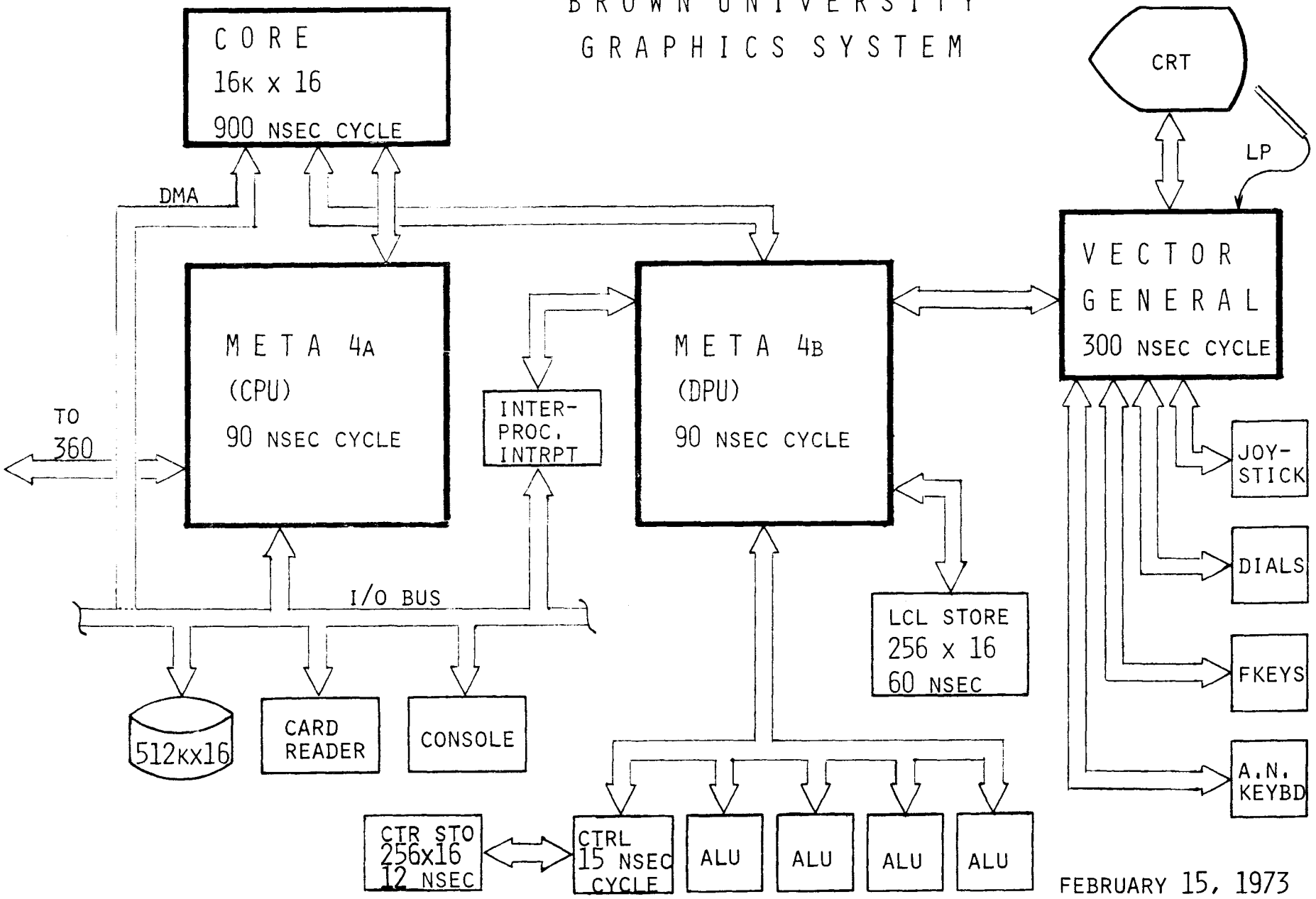
The configuration which has been assembled to meet the above requirements consists of two Digital Scientific Meta 4 microprogrammable computers, a Vector General Display vector/matrix arithmetic unit, and miscellaneous peripherals.[4] The two Meta 4's are designated Meta 4A and Meta 4B and are interconnected with the Vector General as shown in Figure 1. The Meta 4A functions as the satellite processor; the Meta 4B is a display processor, transforming high level graphics programs into a format accepted by the Vector General.

---

[3] A brief description of the language for Systems Development is presented in Appendix D.

[4] A brief technical description of the major hardware components of the system can be found in Appendix A.

BROWN UNIVERSITY GRAPHICS SYSTEM



FEBRUARY 15, 1973

SIMALE (TRANSFORMATION PROCESSOR)

The following sections describe the functions of each component of the system in detail.

2.2 The Local Processor

The first Meta 4 (the Meta 4A) is used as a small satellite processor for local processing. The instruction set which we have microprogrammed for the Meta 4A is very similar to that of a S/360 with additions and modifications to enhance the ability of the processor to meet the needs of the operating system and graphics applications.[5] The tasks of this processor will be:

2.2.1 I/O to the mainframe computer. Currently, the mainframe computer is an IBM S/360 Model 67.

2.2.2 I/O to local peripherals.

2.2.3 Running user written or system provided routines for interrupt processing, data base editing, etc.

2.2.4 Controlling the Display Controller (the Meta 4B).

2.2.5 Providing capabilities for stand-alone graphics applications (those not requiring the mainframe computer).

2.3 The Display Processor

The second Meta 4 (the Meta 4B) shares core with the Meta 4A and can be thought of as an interface between the Meta 4A and the Vector General. However, this interface is much more than a mere data transfer device since it has been given its own general purpose instruction set.[6] Much of this instruction set is identical to that of the Meta 4A, thus giving the Meta 4A/4B aspects of a multiprocessor configuration. The differences between the two instruction sets arise from two sources: instructions in the Meta 4A which have no obvious application to graphic data processing have been dropped, and a comprehensive set of powerful graphic data display instructions has been added. The tasks of the Meta 4B include the following.

---

[5]For a brief description of the Meta 4A target machine, see Appendix B .

[6]The Meta 4B target machine is described in Appendix C.

2.3.1 Fetching Display instructions and coordinate data from data structures or tables in core, processing them (including 3D transformations, clipping, and perspective calculations), and passing them to the display unit.

2.3.2 Providing capabilities not implemented in the display unit's hardware, e.g., light pen tracking and cursor manipulation.

2.3.3 Processing graphic device interrupts, either by status switching in the Meta 4E or by interrupting the Meta 4A.

2.4 The Vector Arithmetic Logic Unit

We have determined that even with the high speed of the Meta 4, it is not possible to provide full three-dimensional transformations in a reasonable amount of time. (We define "a reasonable amount of time" by the criteria of being able to display 1000-2000 vectors at a refresh rate of at least 30 frames per second.) Therefore, we have implemented an arithmetic logic unit of our own design which will provide the necessary vector/matrix operations.

The SIMALE (for Super Integral Microprogrammed Arithmetic Logic Expediter) is a small microprogrammable processor which operates at a basic cycle time of 15 nanoseconds and is closely integrated into the Meta 4B.[7] The functions of the SIMALE are as follows.

2.4.1 Vector-Matrix multiplication. As each component of a graphic coordinate is fetched from core, it is given to the SIMALE for multiplication by the corresponding row of a transformation matrix which is stored in the SIMALE's scratchpad memory. These partial sums are accumulated to form (after X, Y, Z, and possibly W have been fetched) the product of graphic coordinate and the transformation matrix. (The total time for this multiplication will be in the order of 5 microseconds.)

---

[7] A more detailed description of the SIMALE appears in Appendix A.

2.4.2 Windowing. Each point or line for display is then clipped to a user specified window, the coordinates of which are stored in scratchpad memory.[8]

2.4.3 Perspective Division. If the user has requested perspective 3D display, each coordinate is then divided by its Z-component to form a 3-dimensional cone of vision.

2.4.4 The Box Operation. In displaying a hierarchic display file employing nested windows, the SIMALE may be used to calculate the new clipping limits resulting from the composition of the window/viewport and master/instance transformations.[9]

2.4.5 Miscellaneous Matrix Operations. Since the SIMALE is driven from a writable control store, the user will be able to implement his own matrix algorithms. In addition, since the Meta 4B allows the Meta 4A to access the SIMALE directly, programs in the Meta 4A can utilize the capabilities of the SIMALE for non-graphics applications.

2.5 The Display Unit

The Vector General, which is driven by the Meta 4B, is itself a moderately powerful display processor. In addition to the normal complement of display instructions, there are instructions for loading, ANDing, ORing, or ADDing to internal registers, which can be up to 128 in number. These registers hold information such as the current beam coordinates, scale and displacement factors, and analog and digital input from the various graphic devices. The Vector General is currently equipped with a lightpen, 32 function keys, 10 control dials, a joystick, and an alphanumeric keyboard. The functions of the Vector General are as follows.

2.5.1 Driving the vector generator with coordinate data passed to it from the Meta 4B.

2.5.2 Performing scale and translation of coordinate data for viewport operations.

2.5.3 Passing graphic interrupts to the Meta 4B.

---

[8]The algorithm used for the clipping and windowing operations was developed by Sproull and Sutherland[12].
[9]The Box Operation is also completely described in [12].

2.5.4 Accepting analog and digital inputs from the various graphic devices and holding the information until requested by the Meta 4B.

2.6 Local Peripherals

The following lists the I/O devices which are currently available to a program in the Meta 4A. Possible future additions to the configuration include a printer, a small drum, and a real-time clock.

2.6.1 DSC Model 1444 Disk Storage Unit. The disk unit provides the Meta 4A with 1 megabyte of online storage. The disk cartridge is interchangeable with an IBM 2315 cartridge (used on an IBM 1130 Computing System). Average access time (seek time plus latency) is 180 milliseconds; data transfer rate to the Meta 4A is 72 KHz bytes.

2.6.2 DSC Model 3461 Card Reader. The Model 3461 is a 300 card/minute Hollerith Card Reader.

2.6.3 DSC Model 4153 Programmer's Control Panel. The control panel for the Meta 4A was designed to be used with DSC's IBM 1130 emulator, and thus is very similar to an 1130 control panel. However, the functions of the panel have been partially redefined by the Meta 4A firmware to allow, for example, display of all 16 general purpose registers and IPL device selection.

2.6.4 DSC Model 4132 Keyboard/Printer. The key-board/printer is an IBM Selectric typewriter which has been modified for use with DSC's 1130 emulator.

2.6.5 The Interprocessor Interrupt. The inter-processor interrupt (IPI) is a device accessible from both the Meta 4A and the Meta 4B. The function of the IPI is to allow either processor to interrupt the other with a halfword of system or user provided data. This makes possible communication between the processors without the use of core memory.

2.6.6 The S/360 Interface. The S/360 interface connects the Meta 4A to a S/360 multiplexor channel. The interface (together with the Meta 4A firmware) functions in accordance with IBM's Channel to Control Unit specifications[10] and is non-specific, that is, interpretation of channel commands is left to Meta 4A software. The interface allows data transfer between

the S/360 and the Meta 4A at rates up to 50-100KHz bytes (depending on other channel activity).

3. THE OPERATING SYSTEM FOR THE META 4A

3.1 Introduction

A fundamental decision in the design of the operating system for the Meta 4A was to discard the more common "supervisor and management" approach, typified by IBM's O.S.[2] and Univac's Exec 8, in favor of the "extended machine" or "structured" approach typified by Dijkstra's "THE" System[5], IBM's TSS[8] and CP/67[9], and Honeywell's Multics[3].

In the extended machine approach, the operating system provides a "more useful" target machine for the user by simulating it on a simpler machine, the host. By this definition, a micro-programmable machine could be said to provide an extended machine to the software, and it is clear that many levels of extension are possible, with each simulator running on the extended machine provided by the lower level. The first version of the operating system for the Meta 4A, BOGUS (Brown Operating Graphics University System), consists of an extended machine simulator running on the firmware-provided machine. This lowest level system will be referred to hereafter as Level 0. The "user" program, which may be an application program, an operating system or another level of simulation, runs on Level 1.

An immediate effect of this design is to blur the boundary between firmware and software. The user program on Level 1 sees only "hardware" below itself, with the characteristics of that hardware being simulated by Level 0 and by the firmware. It is therefore possible to test the usefulness of a function by implementing it in Level 0, and later moving it into the firmware if that is found to be appropriate. The Level 1 program will not notice the change except in terms of the speed of the function. Thus the extended machine concept and microprogramming together provide an ideal architecture for research into exactly what a "useful" machine for remote graphics (or many other things) should look like.

The major drawback to the extended machine approach is that the extended machine runs slower than would a real machine of the same type, especially in those areas which are difficult for the host machine to simulate. In fact, a simulated instruction may take fifty times as long to "execute" as a native instruction (one found on the host as well as the target machine and executed directly). It is our opinion (to

be tested by trial) that the architecture of the entire system makes this consideration less important, since actions requiring very fast, sub-millisecond response will be performed by the Meta 4B or by the firmware of the Meta 4A, or in special cases could be programmed so that they do not use extensive simulation.

3.2 Characteristics of the Level 0 Extended Machine

The following paragraphs describe the basic supervisory services provided by Level 0. Our criteria for what services are appropriate to Level 0 approximate the restrictions on IBM's O.S. Type 1 SVC's, that is, they must execute in disabled mode and cannot call any other Level 0 routines. (Of course, as Level 1 functions become well defined, they may be added to Level 0 even if they do not meet these preliminary restrictions.)

All Level 0 facilities are accessed via undefined operation codes. This feature has two advantages over the more conventional supervisor-call approach. First, the Meta 4A firmware parses the instruction and stores the effective addresses in low core before causing an illegal-operation interrupt. This saves code on Level 0 and permits more natural specification of parameters for the Level 0 routine. Second, when a Level 0 function is moved down into the firmware, it is possible for a previously assembled program to take immediate advantage of this fact, that is, there is no obsolete Level 0 overhead in accessing the "new" instruction.

3.2.1 Storage Management. Level 0 provides the equivalent of GETMAIN and FREEMAIN instructions for a simple dynamic storage allocation system. It assumes that a reserved location in low core contains the head of a free-area list, and allocates storage from blocks in that list in such a way as to avoid fragmentation. It also merges blocks into a single entry when a newly-freed block is adjacent to another free block. GETMAIN uses the SEARCH, ENQ, and DEQ target instructions to provide exceptionally fast operation.

It is possible for the Level 1 program to provide a more complex free-area management system by swapping different list heads in and out of the reserved location in low core used by Level 0. In this way it can provide subpool facilities, multiple user facilities, etc.

3.2.2 Input/Output. Level 0 extends the relatively primitive I/O repertoire of the Meta 4A to provide

S/360-like CCW (Channel Command Word) facilities to Level 1. Included are command and data chaining, programmed controlled interrupts, etc.

3.2.3 Interrupt Processing. The interrupt processing action of the Level 0 virtual machine generalizes the PSW-swap mechanism of the S/360 and provides the following facilities.

3.2.3.1 Allowance for an arbitrary number of interrupting conditions and exit routines, both hardware and program defined.

3.2.3.2 Support for dynamic allocation of control blocks and saving of "old machine state" (MSR, PC, general registers) so that the Level 1 Interrupt Handlers may be recursive and run enabled.

3.2.3.3 Queuing of interrupts which occur while they are disabled in a manner defined by the programmer.

3.3.3.4 Allowance for interrupt handlers to be specified for both explicit events and classes of events, such as one exit for Function Key 3 and another for all other Function Keys.

3.2.3.5 Allowance for user-defined events which may be "signalled" by software. Also allowance for software "signalling" of hardware-defined interrupts.

To meet these goals, we have generalized the notion of an interrupt into that of an event. The class of possible events includes hardware interrupts as a subset, but also includes software "interrupts" such as event notification. Every event in the system is given a 16-bit name which, along with control information, new machine status, and the amount of workspace the event routine requires, is kept in an Event Table maintained by Level 0 (the table is actually a tree). When an event occurs, a Level 0 event handler scans this table to find the corresponding event routine. If such a routine is found, all necessary linking and status switching is performed, and the routine is given control. If a routine for the specific event is not found, the last four bits of the event name are zeroed, and the search is repeated, etc. This effectively implements the capability noted in 3.3.3.4 above. Since the head of the Event Table is pointed to by a low-core location, it is possible for Level 1 to completely

redefine its event handling strategy merely by pointing this location at a new Event Table. Also, priority among events is a natural offshoot of the manner in which the table is searched.

3.2.4 Other facilities. Other facilities planned for incorporation into Level 0 include floating point subroutines, data conversion routines (e.g., EBCDIC to ASCII), and real-time clock support.

3.3 Program Development Software

Until such time as an assembler for the Meta 4A (and 4B) is implemented, we are using Waterloo's Assembler G (ASMG) for our assembler. ASMG (which runs in the S/360) is table driven and has been modified to recognize the mnemonics and generate machine instructions for the Meta 4A. This gives us a very powerful macro-generator and a rich assembly language. In a similar fashion, we plan to modify the CMS loader to link together Meta 4A object decks and produce a Meta 4A core image in S/360 core, which we can then ship across the interface and store on the Meta 4A disk. This will give us such features as automatic library lookup of unresolved names, V-cons, pseudo-registers, REP cards, etc.

We are also in the process of bootstrapping a version of PL/360[14] to run locally on the Meta 4A, thus providing a minimal local high-level language.

3.4 Debugging Tools

A major part of the initial work on the graphics system is concerned with the development of debugging tools, as users need to debug programs on the Meta 4A, the Meta 4B, the S/360, and even the SIMALE. Of immediate use is a primitive, machine-code-oriented package for debugging Level 0 code itself. This is similar in function to CMS's DEBUG program, but lacking some of the more advanced features.

3.5 High Level Language Support

For the development of local programs, but more importantly, to support the implementation of large applications which require the facilities of both the S/360 and the Meta 4A, a version of the Language for Systems Development [1] is being

implemented which will prcduce code for the 360 and the Meta 4A. When a user ccmpiles a program is ISD, he will be able to specify for which machine object code is to be compiled. In addition, facilities are being added to the language to allow, for example, specification that a procedure or variable refernced by a program in one machine is tc be found in the other processor. This will greatly facilitate the job of allocating (and re-allocating) the various modules of a large application between the two machines.

APPENDIX A: TECHNICAL INFORMATION

A.1 The Meta 4 Processor[10]

The Meta 4, manufactured by Digital Scientific Corporation, is a small logical processor controlled by a random-access Read-Only Memory (ROM). The following paragraphs describe the salient features of the processor.

A.1.1 Read-Only Memory. The control store which drives the Meta 4 is an inductive-type ROM which senses bit patterns which have been "peeled" onto special PC boards by either the user or DSC personnel. (An attractive feature of DSC's ROM is that it can be loaded and modified on-site.) The ROM has a 35 nanosecond access time and is available in up to 4096 16-bit words.

A.1.2 Hardware Registers. The Meta 4 CPU can hold up to 31 16-bit integrated-circuit registers. These registers, all of which are directly addressable by the microcode, can be either general purpose, memory access, scratchpad access, or input/output.

A.1.3 Data Processing Logic. A normal instruction involves three buses called the A, B, and D buses. An arithmetic/Boolean unit processes data received via the A-bus and the B-bus. The result of an operation is transmitted through a Skew unit to the D-bus and thence to a destination register. The arithmetic unit is a 16-bit, high-speed parallel adder. Carry-in controls, together with overflow and carry-out condition bits, allow multiple precision operations. The Boolean functions comprise the logical connectives AND, OR, and Exclusive OR. The skew unit manipulates the result of either an arithmetic or a Boolean operation, providing functions such as right and left shifts of one or eight bits and sign extend.

A.1.4 Sequence Control. Sequence Control for the processor is a program stored in ROM. Addresses in ROM instructions and in a special Link register are used by the branch-control unit to shift control between various sequences as the result of testing operations. Any single bit of any addressable register may be tested for zero or nonzero, 8-bit and 16-bit fields may be tested for zero or nonzero, and

---

[10]Parts of this description were taken with permission from[4].

a self-decrementing register may be tested for zero concurrently with operations of functional units. An indexed branch and multilevel execute facility are available.

A.1.5 Input/Output. I/O to system peripherals may be carried out via direct cable connections to special types of directly addressable I/O registers. The sequence control program may communicate with the system peripheral equipment through these registers. There is also a separate chassis which accepts standard controllers for various peripheral equipment on a plug-in basis. No wiring chages are required to add or delete peripheral equipment. Peripheral equipment controllers operate on a party-line I/O bus or directly to memory, as applicable.

A.1.6 Core Memory. Core memory is operated by the control program through special registers and controls. Four standard memory ports allow multiple processors or special equipment to share multiple banks of memory. Each bank of core memory is an independently operable unit. The processor can use additional memory registers or interleaving to overlap accesses to several banks. Core memory is available up to 65,536 18-bit (16 data bits plus parity and protect) words per memory I/O register and has a 450 nanosecond access time (900 nanoseconds full-cycle).

A.2 The Vector General[11]

The Vector General Graphics Display System is an interactive graphics cathode ray tube (CRT) display that may be connected to any computer system with standard input/output capability. The display interacts with an on-line user by displaying pictoral information on the surface of the CRT and by accepting inputs from external control devices. The inputs are requested and processed by computer programs that alter and maintain the output picture being presented to the user. The following paragraphs describe the capabilities of the Vector General System.

A.2.1 Interface and Controller. The interface between the Vector General transmits 16-bit words between the controller and the computer's memory. The interface also passes graphic interrupts to the computer, and gives the graphics program access to the 128 registers internal to the controller (64 of these are

[11] Parts of this description were taken with permission from [13].

writable). The controller allows register modification via Load, Add, OR, and AND instructions, arithmetic being carried out in parallel two's complement. Vector information is accepted by the controller in absolute, relative, short incremental, long incremental, and auto-increment formats. The controller includes a line locked frame clock. The internal cycle time of the controller is 300 nanoseconds.

A.2.2 Coordinate Transformation Generators. Vector General provides four coordinate transformation options. The first of these, the Dual DAC, merely provides digital-to-analog conversion. The second and third, two dimensional or two dimensional with rotation, provide image scale, translation, and rotation about a single axis. The fourth option allows image scale, translation, and rotation with 6 degrees of freedom.

A.2.3 The CRT and Vector Generator. The CRT is a rectangular 21-inch tube with a display area of 13X14 inches. Deflection is dual electromagnetic; spot size is 0.010-inch; and the dynamic range of the vector generator is 30X30 inches. There are 4096x4096 addressable locations and 16 intensity levels (plus optional continuous intensity modulation). At a 30 frame/second refresh rate, 12,000 linear inches or 6,600 short vectors (0.625 inches or less) can be drawn with end matching and closure of 0.020-inch or better. The vector generator accepts dot, dash, point, and solid vector modes and performs hardware scissoring.

A.2.4 The Character Generator. The character generator displays the 192 character extended ASCII set with 32 optional user defined special characters. Characters can be displayed in four sizes either horizontally or vertically. Characters are generated at an average rate of 10 microseconds/character.

A.2.5 Optional Control Devices. Control devices which can be connected to the Vector General include a 70 key alphanumeric keyboard (256 codes), 32 function keys, a solid state light pen with capacitive tip switch, 10 single-turn control dials, a 10x11 inch data tablet, a 3-axis joy stick, and an 8-channel A/D multiplexer. All of these devices feed analog or digital information into controller registers which may then be read by the graphics program. The keyboard, function keys, light pen, and tablet also generate interrupts which are passed to the computer.

A.3 The SIMALE

The SIMALE is a small microprogrammable processor which we have designed for the purpose of performing vector/matrix and matrix/matrix calculations which cannot be done in the Meta 4B at a sufficient rate of speed. The SIMALE consists of five logical units: the control unit and four parallel processors.

A.3.1 The Control Unit. Microprograms for the SIMALE are stored in a 256x16-bit control store which has an access time of 12 nanoseconds. Instructions stored in the control store contain information such as which data paths and processors are to be enabled, the arithmetic or logical operation to be performed, and the next instruction address. The control has a basic instruction time of 15 nanoseconds. A small push-down stack is also included to facilitate micro-subroutine linkage.

A.3.2 The Arithmetic Processors. There are four interconnected arithmetic processors, each containing two work registers, an arithmetic unit, and 16 18-bit words of scratchpad memory (access time 5 nanoseconds). The arithmetic unit accepts two inputs which can come from the work registers, scratchpad, another processor, or directly from a Meta 4 I/O register. The arithmetic (or logical) operation is performed in 24 nanoseconds, and the result is gated back to one of the possible destinations (which are the same as the possible sources). Typically, all four processors run independently and in parallel. Thus, for example, it is possible to multiply a complete row of a matrix (one element of the row is stored in each of the scratchpad memories) by a scalar in no more time than it takes to do a single multiplication. (A full 16x16-bit multiplication takes an average of 900 nanoseconds.) For certain windowing operations, the four processors can be cross connected by twos to form simultaneously the sum and difference of pairs of numbers.

APPENDIX B: THE META 4A TARGET MACHINE

B.1 Information Formats

B.1.1 Operands. Information on the Meta 4A is stored in main memory in 8-bit units, called "bytes", as in the S/360. Bytes may be handled separately or grouped together in fields. The most common field consists of 2 bytes, and is sometimes called a "halfword". These halfwords are the basic building blocks of CPU instructions and are also the size of the fixed-point 2's complement numbers operated upon by arithmetic instructions. These instructions require the halfwords to be located on an even byte boundary. Other instructions operate upon variable length fields of bytes, called "character strings". These character strings may be located anywhere in memory and may be of any length.

B.1.2 Addressing. Bytes in main storage are addressed consecutively from 0. The Meta 4A uses a 16-bit address, allowing for a maximum of 64K bytes. Currently we have only 32K, and addresses wrap from X'7FFF' to 0. A field of bytes (1 or more) is usually addressed by its leftmost byte. Effective address calculation is similar to that of the S/360 (e.g., Base-Displacement), with some instructions allowing one level of indirect addressing.

B.1.3 Arithmetic. All arithmetic on the Meta 4A is performed on 16-bit two's complement binary numbers using two's complement arithmetic. Any overflow that occurs is ignored in some operations (such as address computation), but may cause a program interrupt in certain others (such as the Add instructions).

B.2 Central Processing Unit (CPU)

B.2.1 Registers. Instructions can address information in sixteen registers, three of which serve special purposes. Registers have a capacity of one halfword and are addressed by a 4-bit number from 0 to 15. Register 0 is called the Machine Status Register (MSR), Register 1 is the Program Counter (PC), and Register 15 is the Stack Frame Pointer (SFP).

B.2.1.1 The Machine Status Register (MSR). The MSR, Register 0, contains the information

required for proper program control and execution. It contains a 3-bit condition code, various status flags, and a mask to enable and disable the various possible interrupts.

B.2.1.2 The Program Counter (PC). The PC, Register 1, is the Program Counter (or instruction address register) for the Meta 4A. It is incremented by 2 during instruction fetching for each halfword of the instruction fetched. It may be operated upon, in all respects, just as any of the other registers, although the effects upon program execution should be obvious.

B.2.1.3 The Stack Frame Pointer (SFP). The SFP, Register 15, is assumed by the ENTER and RETURN instructions (see B.2.2.4) to point at the savearea being used by the program currently in control. These instructions automatically update the SFP, thus providing an efficient means of program linkage.

B.2.2 Instruction Formats. The Meta 4A has 8 instruction formats corresponding closely to those of a S/360. Two notable differences are the Indexed-Branch format, which allows use of the PC as a base register (as on an IBM 1130), and the Varying-length SS format, which allows character operations on strings of arbitrary length. With the notable exception of floating point instructions, the capabilities of the Meta 4A instruction set also parallel those of the S/360. However, several special instructions have been added to the instruction set on the basis of our past experience in graphics programming.

B.2.2.1 Push/Pop. These instructions allow data from multiple core locations or registers to be placed on a LIFO-type stack in core. The firmware does all necessary pointer updating and can cause a stack overflow/underflow interrupt (or, optionally, set the condition code) when the stack limits are exceeded.

B.2.2.2 Search. This instruction is used to search a table or a linked list for a key which holds some relation to a search argument. The key can be from 1 to 255 bytes in length. The criterion for a successful search can be equal, greater, or less than, or ones, mixed, or zeros.

B.2.2.3 Enqueue/Dequeue. These instructions can be used tc add cr delete an element from a linked list, all necessary pointer manipulation being done by the firmware.

B.2.2.4 Enter/Return. These instructions implement save area chaining and automatic storage allocation (and de-allocation). The Enter instruction is generally executed as the first instruction of a subroutine and assumes that the SFP (Register 15) points into a "stack frame". A stack frame is a ccntiguous block of storage which is used by Enter for allocation of multiple saveareas, together with fcrward and backward pointers, and workareas of user defined length. If the Enter instruction finds that there is insufficient space in the current stack frame, a stack frame overflcw interrupt occurs, and the operating system can procure and link in a new stack frame. The Return instruction performs the reverse function, causing a stack frame underflow interrupt when the bcttom of the stack frame is reached.

B.2.3 Interrupt Handling. There are four types of interrupts on the Meta 4A: Supervisor Call (SVC), Program, I/O, and S/360. When an interrupt is detected and if it is not disabled by the MSR, the following actions occur.

B.2.3.1 The current MSR and PC are stored in two halfwords in low core, and an interrupt code is generated and stored. If the interrupt is an SVC or Program Check, the Instruction Length Code in the MSR indicates the length of the instruction causing the interrupt. In addition, for an illegal operation Program interrupt, certain information about the offending instruction (effective operand addresses, immediate data, etc.) is stcred. (I/O interrupts are further described below.)

B.2.3.2 A new MSR and PC are loaded from two other halfwords in low ccre, and execution continues with this new machine status.

B.3 Input/Output Handling

B.3.1 The Unit Control Block Table. The UCBT is a table of halfwords in low core, one for each I/O device attached to the Meta 4A. For each device, the corresponding entry in the table points to the Unit Control Block for that device. The UCB contains both firmware and operating system defined information. The firmware uses the UCB to store current device status and (for the S/360 interface) information about the current command and sense information. The operating system can use the rest of the UCB for such information as online/offline flags, I/O request queue pointers, etc.

B.3.2 I/O to Local Peripherals. The Start I/O (SIO) instruction is used to issue I/O commands to local peripherals. The operand of the SIO instruction is an IOCC (I/O Control Command) which is similar in format to an IBM 1130 ICCC. Following execution of the SIO instruction, the firmware sets the condition code to indicate the result of the I/O operation. Possible settings include Device Busy, Operation in Progress, and Operation Complete. After execution of the SIO (also after any I/O interrupt), the firmware also stores the current device status in the appropriate UCB.

B.3.3 I/O to the S/360. The Meta 4A communicates with the S/360 with three basic instructions: Execute Channel Command, Transfer Byte, and Send Status. The first of these is used to request block transfers of data in accordance with the Channel Command received from the S/360 channel and stored in the UCB for the S/360 interface. The Transfer Byte instruction can be used to request the transfer of a single byte to or from the interface, thus enabling operations such as "gather-read" and "scatter-write". Send Status is used to give asynchronous or ending status to the channel.

APPENDIX C: THE META 4B TARGET MACHINE

C.1 General Approach

The overall architecture of the Meta 4B is very similar to that of the Meta 4A as described in Appendix B. The Meta 4B shares the core memory of the Meta 4A and uses the same addressing schemes. Instruction formats, except as noted below, are identical, and a similar set of general purpose registers is available. Below are described those features of the Meta 4B which give it its graphic display processing capabilities.

C.2 Central Processing Unit (CPU)

C.2.1 Registers. There are sixteen registers available for display program use, however, unlike the Meta 4A, the majority of these are special purpose. Along with the Program Counter and Machine Status Register, registers will be dedicated to uses such as the current beam position, cursor address, a base register for paged data structures, an interrupt mask, etc. As in the Meta 4A, the programmer is free to modify any of these registers at his own risk.

C.2.2 Instruction Formats. Instruction formats are identical to those of the Meta 4A, however many instructions have been dropped from the repertoire, particularly the data conversion, list manipulation, and subroutine linkage instructions. A complete set of register manipulation, arithemetic, and logical instructions has been kept, and the following additions have been made.

C.2.2.1 Branch/Interrupt Instructions. The branch instructions of the Meta 4A have been modified so that a successful test may result in one of four actions: branch, interrupt the Meta 4A and stop, branch and interrupt, or interrupt the Meta 4A and continue.

C.2.2.2 SIMALE Instructions. Instructions have been added to load and store the contents of the SIMALE's control storage and scratchpad memory. A set of vector and matrix arithmetic instructions has also been added.

C.2.2.3 Vector General Instructions. Instructions have been added to access directly the registers in the Vector General. Examples of these instructions are Read Analog Inputs and Set Function Key Indicators.

C.2.3 Display Instructions. The display instructions form the heart of the Meta 4B instruction set, and a special attempt has been made to provide a complete and powerful set. A particular design aim has been to simplify the display of structured display files. While the graphics programmer is free to keep his coordinate data in conventional table format, display instructions are available to allow direct display from a variety of more complex data structures. Each display instruction contains the following types of information.

C.2.3.1 Data Type. The coordinate data being referenced by the display instruction can be absolute, relative or incremental vector, or character.

C.2.3.2 Vector Mode. Vector data can be displayed as solid lines, dashed lines, dotted lines, or end points.

C.2.3.3 Display Mode. The display mode refers to the manner in which the coordinate data is to be interpreted. Possible display modes include moving to the first point, then drawing to the second, alternating moves and draws, moves alternating with subroutine calls, and a crosshatching.

C.2.3.4 Addressing modes. The set of addressing modes allows display from a wide variety of different data formats, for example tables of immediate data, referenced data, linked lists or rings, etc. Pointers to data can be absolute 16-bit addresses or in base-displacement form. In addition, a paged mode is available in which a page base register is added to all pointers. This simplifies the processing of paged data structures.

C.2.4 Interrupt Handling. All graphic interrupts which the Meta 4B receives from the Vector General are kept in a status register which can be examined by the graphics program. If an interrupt has been "disabled" by the graphics program, the Meta 4B leaves the interrupt pending. The graphics program can then use the Branch/Interrupt instruction to act

on the interrupt at a later time. If an interrupt has not been disabled, the Meta 4B immediately stores the current status of the system in the appropriate UCB and interrupts the Meta 4A. An interrupt can be completely disabled by zeroing its enable bit in the Vector General.

APPENDIX D: LANGUAGE FOR SYSTEMS DEVELOPMENT

LSD is a general purpose procedure oriented language with many of the features and much of the syntax of PL/I. In contrast to PL/I, however, the language enables the programmer to "get at" the machine for which he is programming rather than hiding that machine from him. Thus the user can explicitly perform operations on main memory locations and registers; he can intersperse LSD code with assembly language or machine language (through the CODE/ENDCODE construct). LSD also provides a variety of extension mechanisms to permit the user to tailor the language to specific problems or programming styles. The language is oriented towards a knowledgeable and sophisticated class of programmers who wish to do systems programming as well as applications programming in a high level language. The compiler is optimized for very efficient code generation.

LSD is designed for a machine like the IBM System /360 and System /370. A number of constructs have been included in the language because they are present in the System /360: general and floating point registers, a Translate and Test function, decimal data type, etc.. Nevertheless the language is general enough so that programs can be written for any machine which is similar to the System /360 and compile to efficient code.

The programmer can exercise considerable control over the code generated by the compiler. Conversions are never performed unless the user specifically requests them by calling a built-in function. Instead code will be generated whenever possible regardless of the data types of the operands. Thus a substring can be taken of a floating point number (the number will be treated as a character string). The compiler will, however, generate a warning message whenever an operand type other then the expected one is used or whenever mixed mode operations are encountered to call the user's attention to these in case they are errors. If code cannot be generated for a construct, the compiler will automatically place one or more no-ops at that point in the generated code and provide a patch area at the end of the program. The user can also control which operands are retained in registers and reserve specific registers for his exclusive use (the compiler will not generate code using these registers). The programmer can control the generation of code to check for certain runtime errors (e. g. stringrange) and provide routines which are to be called when specific interrupts or errors occur.

The user can control the degree of optimization which the compiler will perform and the type of space-time tradeoff which is to be used in that optimization. Optimization on particular variables can be turned on and off. There is a construct in LSD for indicating which of a set of possible paths are most likely to be executed. This is used by the compiler in deciding what should be kept in registers and whether expressions should be moved out of loops. The user can request that the compiler generate assembly

language code which will be listed by source program statement. A programmer can then hand optimize critical portions of his programs (it is hoped that at the highest level of optimization the LSD compiler will produce code about as good as that produced by an experienced assembly language programmer).

LSD includes significant enhancements of a number of facilities available in PL/I. Multiple level pointer chasing in a single statement and the use of more then one pointer at each level are provided. For example, the address of a data element may be based on the address of a page, the relative displacement of a block from the top of the page, and the offset of the element from the top of the block. Pointers can be variables or constants of any data type. Other improvements are in the areas of: character facilities, debugging facilities, more flexible structure definitions, and the global scope type for variables.

A PL/I variable is given the data type CONTROLLED to provide a stack facility in PL/I. The only way the programmer can use this stack is to access the top element To access an element lower in the stack, the elements above it must be lost. The programmer cannot access the pointer which PL/I uses to access the variable. LSD has a data type of STACKED which extends the capabilities of PL/I. The language has a built-in function which returns a pointer to the previous element when given a pointer to an element of a stack.

LSD attempts to provide many of the character handling capabilities of SNOBOL4 in a PL/I syntax. There are infix operators and statements which insert and delete substrings, locate patterns, and trim trailing blanks. Special functions (ANY and NOTANY) permit the user to search for any of a group of characters or for any character except a specified group.

LSD is designed to be operating system independent and yet provide only a minimal runtime environment. The I/O facilities are very limited. Only character strings can be inputed or outputed and the user must do any data set initialization (OPEN) which is necessary. No facilities for multitasking or coroutining are currently available, although they may be added to the language in the future.

LSD provides a full set of structured programming primitives (the GOTO is also available for those who prefer or need it). The Select statement allows the user to execute one of several pieces of code depending on the run-time value (number or character-string) of an expression. DO WHILES and iterative DO's are available. The OUTOF and ENDOF constructs branch to either the statement following the end of a DO loop or the END statement of the loop. The If-THEN-ELSE is also available for binary choices.

APPENDIX E: LIST OF PUBLICATIONS AND ACTIVITIES

Publications and Activities Directly Resulting from National Science Foundation Grant GJ-28401X and Office of Naval Research Contract N00014-67-A-0191-0023:

Publications Accepted:

"Microprogramming for Computer Graphics", A. van Dam, SIGGRAPH, Vol. 5, No. 4, Winter 1971.

"Software Data Paging and Segmentation for Complex Systems", Frank W. Tompa and A. van Dam, Information Processing Letters, Volume 1, Number 3, North-Holland Publishing Company, February, 1972.

"Microprogramming for Computer Graphics", A. van Dam, SIGMICRO Newsletter, Vol. 3, No. 1, April, 1972.

"Some Implementation Issues Relating to Data Structures for Interactive Graphics", A. van Dam, International Journal of Computer and Information Sciences, Plenum Press, August 1972.

"Systems Programming Languages", D. Bergeron, J. Gannon, D. Shecter, F. Tompa and A. van Dam, Advances in Computers, Volume 12, Academic Press, October 1972.

"The Super Integral Microprogrammed Arithmetic Logic Expediter (SIMALE)", H. Webber, SIGMICRO, Vol. 6, No. 4, 1972.

"On the Unfolding of Singularities in Complex 2-space. Part 1: (Z**2,Z**3)" T. Banschoff and C.M. Strauss, American Mathematical Society, National Meeting, Dallas, Texas, January, 1973.

"A Machine and A Language for Interactive Graphics Satellite Processing", G. Stabler, I.Carlbom and M. Michel, ACM SIGPLAN/SIGMICRO Interface Meeting, May 1973.

"Computer Architecture and Instruction Set Design", Paul Anagnostopoulos, Gary Scckut, George Stabler, A. van Dam and M. Michel, National Computer Conference and Exposition, June 4, 1973.

"Operating System Design Considerations for Microprogrammed Minicomputer Satellite Systems", John Stockenberg, George Stabler, Roy Johnson, Paul Anagnostopoulos, Robert Munck and A. van Dam, National Computer Conference and Exposition, June 4, 1973..

Invited Papers:

"Microprogramming in Graphics", A. van Dam, Annual SEAS Conference, Pisa, Italy, September 1971.

"Software Support Packages for the Vector General on DSC's Meta 4 and DEC's PDP-11", M. Michel, Vector General User's Group Conference, FJCC, December 1972.

"Microprogrammed Intelligent Terminals and Satellites", A. van Dam, National Computer Conference and Exposition, June 4, 1973

Submitted Papers:

"Computer-Encouraged Serendipity in Pure Mathematics", C. Strauss and T. Banchoff, National Computer Conference and Exposition, June 4, 1973.

"Microprogrammed Device Controllers", M. Michel and A. van Dam, ACM Computer Surveys, January, 1973.

Theses:

"On Optimizing Compilers and Generation of Optimal Code in the LSD Compiler", J. Guttag, Masters Thesis, Brown University, August, 1972.

"A System for Systems Development", R. Bergeron, Phd Thesis, Brown University, in preparation.

Books in Progress:

Interactive Computer Graphics, A. van Dam and James Foley, (to appear in the Addison Wesley /IBM Systems Programming Series), 1974.

Readings in Computer Graphics, A. van Dam and Ira Cotton, to be published by Academic Press 1973.

Courses:

Applied Mathematics 29: Computers and Art (A public exhibition of the resulting computer art was favorably reviewed).

Applied Mathematics 276: Computer Graphics.

Major Publications Resulting from National Science Foundation Grant GJ-181:

Publications Accepted:

"An Introduction to Interactive Computer Graphics", A. van Dam, Proceedings of Delft Symposium on Interactive Computer Graphics, October 1970.

"A Microprogrammed Intelligent Graphics Terminal", L. Schiller, R. Abraham, R. Fox and A. van Dam, IEEE Transactions on Computers, July 1971.

"Programming Language Comparison Study", A. van Dam, R.D. Bergeron, J.D. Gannon, and J.V. Guttag, U.S. Army Safeguard Systems Command, September, 1971.

"Language for Systems Development", A. van Dam and R.D. Bergeron, ACM SIGPLAN Symposium on Languages for Systems Implementation, October 1971.

REFERENCES

1) Bergeron, R. Daniel, J. Gannon, D. Shecter, F. Tompa, A. van Dam, Systems Programming Languages, Advances in Computers, 12, Academic Press, (October, 1972).

2) Clark, W.A., The Functional Structure of OS/360, Part III: Data Management, IBM Systems Journal 5, 1(1966), 30-51.

3) Corbato,F.J. and Vyssotsky, V.A., Introduction and Overview of the Multics System, AFIPS Conf. Proc. 27(1965 FJCC), 185-196.

4) Digital Scientific Corporation, Meta 4(TM) Series Computer System Reference Manual, Publication Number 7032M0, (May 1971).

5) Dijkstra, E.W., The Structure of THE Multiprogramming System, CACM 11,5(May 1968), 341-346.

6) Evans & Sutherland Computer Corporation, Line Drawing System Model 1 System Reference Manual, (November 1970).

7) Evans and Sutherland Computer Corporation, Line Drawing System Model 2 System Reference Manual, (August, 1971).

8) IBM Corporation, System/360 Time Sharing System Resident Supervisor Program Logic Manual, Form Y28-2012.

9) IBM Corporation, Control Program-67/Cambridge Monitor SystemUser's Guide, Form GH20-0859.

10) IBM Corporation, IBM System/360 and System/370 Interface Channel to Control Unit Original Equipment Manufacturers' Information, Form GA22-6974.

11) Myer, T.H. and Sutherland, I.E., On the Design of Display Processors, CACM, (June 1968), 410-414.

12) Sproull, Robert F. and Sutherland, I.E., A Clipping Divider, Proceedings of AFIPS (1968).

13) Vector General Corporation, Graphics Display System Reference Manual, (July 1971).

14) Wirth, Niklaus, A Programming Language for the 360 Computers, JACM, 15,1, (January 1968), 37.