A MINICOMPUTER GRAPHICS SYSTEM

by

VERNE ROY WALRAFEN

B.S. in Civil Engineering, University of Kansas, 1963

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1976

Approved by:

Major Professor

309

TABLE OF CONTENTS

Page

LIST OF FIGURES ... 5

CHAPTER ONE

INTRODUCTION ... 6
PAPER ORGANIZATION ... 6
MOTIVATION ... 7
INHERENT LIMITATIONS
Main Memory Size ... 8
Programming Effort ... 9
Display Speed ... 9
SPECIFIC AREAS OF CONVERSION EFFORT
Memory Size ... 10
Overlay Routine ... 10
Word Size ... 14
Fortran Language ... 14
Entry Point Definitions ... 15
DIFFICULTIES ENCOUNTERED
Operating System Protection ... 17
Operating System Modifications ... 17
Standard NOVA Software ... 19
Graphics Software ... 19
Hardware Support ... 19
EVALUATION OF PROJECT RESULTS ... 20

CHAPTER TWO

GRAPHICS SOFTWARE ORGANIZATION
Constructors ... 21
Compilers ... 22
Transformers ... 22
USAGE OF GRAPHICS PACKAGE
Overlay Routine Calls ... 23
Overlay Segment Files ... 24
NOVA's Text Editor ... 25
NOVA's Fortran Compiler ... 25
NOVA's Relocatable Loader ... 26
Problem Program Execution ... 26

CHAPTER THREE

CONSTRUCTION OR MODIFICATION OF GRAPHICS PACKAGE
Modification of Graphics Routines Source Code ... 30
Development of Additional Graphics Routines ... 30
Seperation of Graphics Routines Into Two or More Groups ... 31
Construction of Page Zero Linkage Routine
Phase 1 - "Labeled Common Definitions" ... 32
Determination of Overlay Area Size ... 33

Page

CONSTRUCTION OR MODIFICATION OF GRAPHICS PACKAGE (continued)
Determination of Fortran Library Subroutines
Required by Graphics Routines 34
Construction of Page Zero Linkage Routine
Phase 2 - "Fortran Library Subroutines" 35
Determination of Main Memory Address Boundaries
for the Overlay Area 35
Modification of Overlay Area Address Boundaries
in "OVERLAY" and "SMASH" 39
Creation of Core Image Files for All Overlay Groups 40
Execution of "SMASH" to Produce Overlay Segement Files 42
Construction of Page Zero Linkage Routine
Phase 3 - "Graphics Routines Entry Points" 43
You are now ready to use your new overlay segment files 44

APPENDIX "A"

SOURCE CODE FOR
OVERLAY.SR 46
SMASH.SR 50
USERLINK.SR 53
MYLINK.SR 54

APPENDIX "B"

SOURCE CODE FOR
CONSTRUCTORS 55

APPENDIX "C"

SOURCE CODE FOR
COMPILERS 66

APPENDIX "D"

SOURCE CODE FOR
TRANSFORMERS 80

APPENDIX "E"

SOURCE CODE FOR
EXAMPLE PROGRAMS 89
TEST PROGRAMS 91

APPENDIX "F"

SYSTEM HARDWARE
HARDWARE CONFIGURATION
Minicomputer 95
Disk Drive 95
Graphics Terminal 95

Page

APPENDIX "G"

MISCELLANEA

Explanation of Components of Final Step Command String 99
Definition of File Suffix by File Type 100
File Type Relationships 100
User Status Table Template 101
Wire List for CRT Plug on Synetics NOVA - Slot 9 102
Computek 300 / Synetics NOVA Connect Procedure 102
Computek 300 / Synetics NOVA Disconnect Procedure 102

LIST OF FIGURES

Page

ONE

INITIAL OVERLAY ROUTINE 13

TWO

FINAL OVERLAY ROUTINE 13

THREE

ILLUSTRATION OF REASON FOR ERRORS IN
ENTRY POINT DEFINITIONS IN "PAGE ZERO" 16

FOUR

GRAPHICS PACKAGE USAGE DIAGRAM 27

FIVE

PROCEDURE DIAGRAM for
CONSTRUCTION OR MODIFICATION OF GRAPHICS PACKAGE 29

SIX

OVERLAY ADDRESS BOUNDARY SELECTION 37

SEVEN

EXAMPLE MAIN MEMORY ADDRESS MAP 38

CHAPTER ONE

INTRODUCTION

During the spring semester of 1974 the graphics class (CS830) at Kansas State University wrote a set of interactive vector-graphics routines for a Computek 300 graphics terminal using timesharing Fortran IV on the GE635 system at the University of Kansas.

It was decided to convert a basic subset of the graphics package for use on the NOVA minicomputer system available in the KSU Computer Science Department.

The graphics package had about ten man months invested in it and it was estimated that to convert it for use on the NOVA minicomputer system would require approximately three man months.

This paper is a report on that effort and on how to use and/or modify the resulting graphics software package.

PAPER ORGANIZATION

Chapter One is intended for the reader that is interested only in the research description and the results and conclusions.

Chapter Two is intended for the reader that desires to use the resulting graphics package on the NOVA minicomputer.

Chapter Three is intended for the reader that might desire to either duplicate the conversion effort or modify the resulting graphics package in order to expand upon the basic subset of graphics routines selected for conversion in this research.

Appendices include the source code for all the selected graphics routines plus other material that supports or expands upon various items in the text of chapters one, two and three.

MOTIVATION

Dynamic graphic sequences were not feasible using the original graphics package on the GE635 system due to the low speed of data transmission via the communication lines. Only the two speeds of 110 baud and 300 baud were available and neither was sufficient to provide relatively smooth motion during dynamic graphic sequences. A higher data transmission rate of up to 1200 baud was available between the NOVA minicomputer and the Computek graphics terminal.

The line charges for using the original graphics package on the GE635 system between Lawrence and Manhattan were so high that the cost of running the package was excessive and as a result the availability of the package to Computer Science students at KSU was naturally very restricted. Since the NOVA minicomputer system is readily accessible to students at KSU, clearly the result of converting the original GE635 graphics package to run on the NOVA minicomputer would be good availability of the package to the student. The NOVA minicomputer system does not represent a large capital outlay when compared to larger systems, such as the GE635, and with the elimination of line charges an inexpensive standalone graphics system would result.

A definitive study in portability of software from a large machine environment to a minicomputer environment would provide both useful training and experience and, hopefully, some guidelines for anyone desiring to make a similar effort. This is particularly important to us in today's world of shrinking finances and expanding demands since minicomputers are rising to fill the gap.

INHERENT LIMITATIONS

As a result of the difference in size of main memory between the GE635 and the NOVA minicomputer, the amount of programming effort forced upon the graphics package user and the realizable display speed both exhibited induced limitations. The discussion that follows addresses itself to both the original inherent limitations and the resultant new induced "inherent" limitations.

Main Memory Size

It was quite obvious upon the most cursory examination that the small amount of main memory available in the NOVA minicomputer, only 16,384 words, simply would not be adequate to hold an operating system, all the graphics routines, the required fortran library subroutines, fortran's runtime stack and the user's problem program.

Since the DOS operating system on the NOVA did not provide for the capability to overlay segments of core with object code modules from secondary storage upon demand, a software overlay routine was the obvious requirement.

In addition, the size of the user's problem program would clearly be restricted considerably from those possible on larger machines and the precise limits on this could not readily be anticipated since the amount of main memory that would be remain could not be determined until the largest overlay segment, containing the graphics routines, was built.

However, since the resulting minicomputer graphics package was to be used as an instructional aid rather than in any sort of a production environment, this size limitation was construed to be tolerable.

Programming Effort

The existence of an overlay software routine created the burden upon the problem programmer to place calls to the overlay routine in his program whenever specific graphics routines he wanted to use were not resident in main memory.

This clearly would result not only in extra coding effort and larger problem programs, but also in the requirement that the user know at all times in his logic flow whether or not the routines he needs are in fact resident in main memory.

Display Speed

The time required to execute the overlay routine and to actually transfer the required overlay segment into main memory from secondary storage would clearly reduce the effective display speed considerably from the upper limit of 1200 baud.

The magnitude of this reduction could not be anticipated and would be critical if the hoped for results of smooth graphics sequences were to be realized.

The degree of optimization of the overlay routine's source code was of prime importance in the conversion effort because of this speed reduction.

The design of the user's problem program could degrade the display speed even further since the placement of the calls to the overlay routine would be completely under the user's control.

SPECIFIC AREAS OF CONVERSION EFFORT

Even though much of the conversion effort was actually done as a large number of interlinked actions and sometimes reactions, the presentation of this report requires that each specific area be treated seperately.

The following paragraphs attempt to give both the reasons for the various efforts and the results of said efforts.

Memory Size

The primary obstacle that had to be overcome was the small amount of main memory available in the NOVA minicomputer, particularly when one allowed for the fact that the operating system, DOS, took up some 34+% of the already limited main memory.

Whether or not all the graphics routines being transported could have resided concurrently in main memory or not was a moot question since, it was readily apparent that there would have been insufficient space remaining for the user's problem program.

The only viable solution to this was the development of software to overlay groups of graphics routines, overlay segments, in main memory under program control.

Overlay Routine

The initial overlay routine was designed to bring into main memory a complete new "Core Image" which included all the fortran library subroutines, the user's problem program and finally the new segment that contained the desired graphics routines to be executed.

This seemed a logical approach since the NOVA's relocatable loader could create only a complete "Core Image" file that contained the different groups of graphics routines.

Since one could not actually read into main memory those portions of code that had been assigned new values at execution time, the areas that contained variables such as the common area, the implementation of the first overlay routine simply read the "Core Image" file one buffer full at a time and did not actually transfer it into the area of main memory that corresponded to that area in the "Core Image" file.

Once the overlay routine read in enough buffers to, in effect, bypass all code up to the graphics routines, it then commenced actually transfering the new object code into the area of main memory being overlayed.

When the graphics package was actually executed using this first version of the overlay routine, it became readily apparent that the display speed reduction was excessive.

A further design fault was that the overlay routine always read the requested overlay segment into main memory even if the requested segment was already resident in main memory.

At this point the overlay routine was modified to do two things, one being to check first to see if the overlay segment being requested was already resident in main memory and if so to simply return at once to the user's problem program and the other was to commence loading of the overlay segment file immediately at the overlay point in main memory.

The immediate loading required that the "Core Image" file produced by the NOVA's relocatable loader be processed by a routine, which I named "SMASH", that would strip off the unwanted portion leaving only the object code for the overlay segment since the overlay routine no longer bypassed the unwanted portion.

The source code for both versions of the overlay routines and for the "smash" routine are presented in Appendix "A".

FIGURE ONE : INITIAL OVERLAY ROUTINE

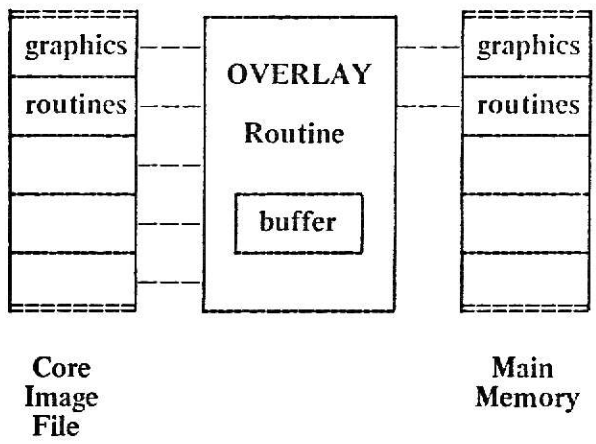
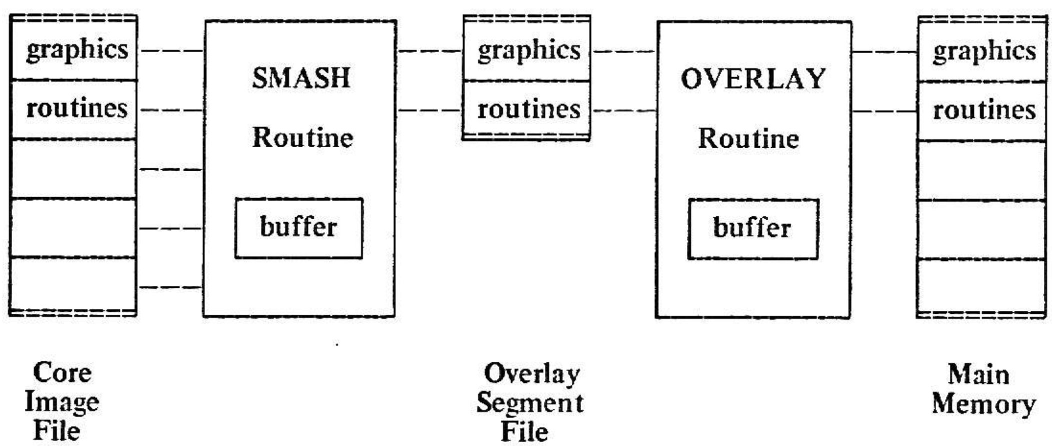


FIGURE TWO : FINAL OVERLAY ROUTINE



Word Size

The word size on the GE635 was four bytes while on the NOVA words are only two bytes in length.

Since the graphics routines do character manipulation any variable that the program logic handled was assumed to contain four characters and this was no longer the case once the graphics routines had been transported for execution on the NOVA minicomputer.

This resulted in having to modify the source code of some of the graphics routines.

Fortran Language

For compiler efficiency Data General's Fortran IV compiler requires a partial ordering of statements with all non-executable statements preceding any executable statements in the program unit.

Since this constraint was not present in the original Fortran language that the graphics routines were written in, all routines had to be re-ordered.

The breaks between groups of NOVA Fortran's statements are indicated to the compiler unambiguously by the control statements:

.SPEC
.EXEC
.BODY

The ordering of statements and control statements is:

1.) Specification Statements: "COMMON", "DIMENSION", Data-type Declarations, "EQUIVALENCE" and "EXTERNAL".
2.) .SPEC
3.) "DATA" Initialization Statements.
4.) .EXEC
5.) "FORMAT" Statements.
6.) .BODY
7.) Executable Statements including Statement Functions.

Entry Point Definitions

The NOVA's operating system, DOS, uses low core, commonly called "page zero", to store entry point definitions for all the routines in the user's problem program which includes the entry points into all of the Fortran library subroutines as well as the entry points into all of the user's own subroutines (the graphics routines in our case).

The relocatable loader created a different "page zero" for each core image file that it produced according to what subroutines were in the object modules being loaded.

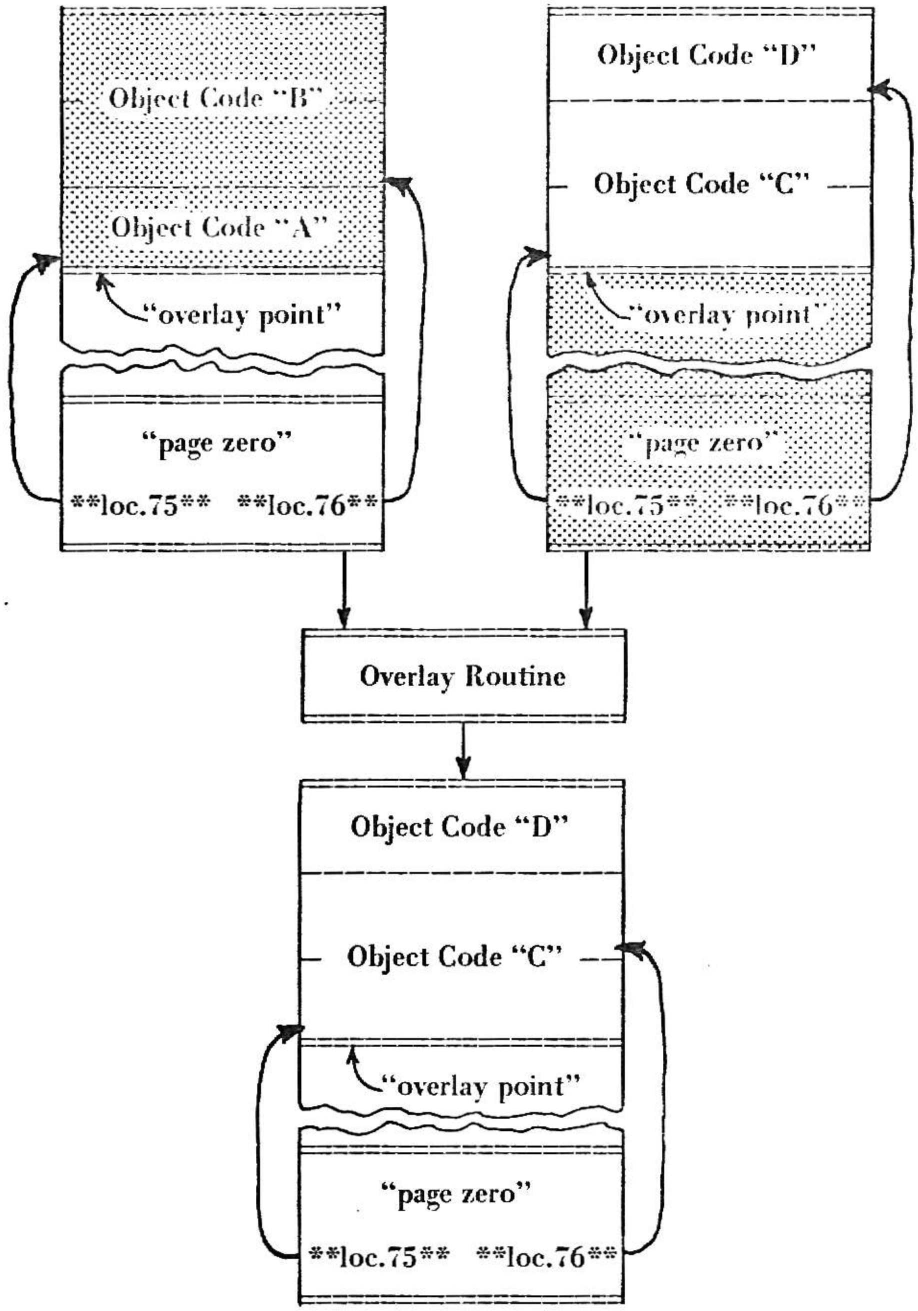
The result of this was that the entry point definitions in the user's core image file either did not have definitions for some of the subroutines that the graphics routines required or, in the cases where it did have them, they were incorrect.

It was necessary to force the relocatable loader to load entry point definitions in the same place in "page zero" for both the overlay core image files and the user's core image files.

The only way to do this was to create an assembler language routine that requested all the routines that the graphics routines would need, plus providing specific entry point definitions for the graphics routines themselves, plus providing definition and extents of all labeled common areas.

These assembler language "linkage" routines only force the relocatable loader to build the same "page zero" under different conditions and produce no actual words of executable code in the final core image files.

FIGURE THREE : ILLUSTRATION OF REASON FOR ERRORS IN ENTRY POINT DEFINITIONS IN "PAGE ZERO".



The results of overlaying the object code for "A" and "B" with the object code for "C" and "D" is, since the original page zero is retained, the addresses resolved in the routines calling "C" still, accidentally, point to "C" but the addresses resolved in the routines calling "D" point, not to the entry point of "D", but to the location where "B" used to reside and which is now someplace in the middle of the object code for "C".

DIFFICULTIES ENCOUNTERED

The problems presented for solution by the conversion of the graphics routines for use in the NOVA / Computek environment were not the only difficulties encountered.

The environment itself came with its own built-in set of problems both with the hardware and with the software.

Operating System Protection

The operating system provided no protection such that when one tested any unproven program he quite often found that an error had caused a branch into some portion of main memory that was not intended for execution and the contents of said area was immediately treated as valid instructions causing, as often as not, not only the overwriting of random portions of main memory but also the overwriting of random portions of secondary storage where all our source files previously existed.

This caused numerous restarts from paper tape versions of all routines plus re-entry of all modifications to the source code from the time of punching the paper tapes.

Operating System Modifications

The operating system changed twice during the project.

Once it expanded by an amount sufficient to cause the overlay segments previously created to be too large to fit into main memory and required the break down of the graphics routines into three groups rather than the original two groups.

This, of course, made it necessary to rebuild the linkage routines, to modify the overlay and smash routines, to regenerate all overlay segment files and to rebuild all test problem programs.

The next change was the replacement of the standard disk operating system, DOS, with a real-time disk operating system, RDOS, accompanied by the addition of another 16,384 words of main memory thus changing the NOVA's entire configuration.

This allowed the use of two overlay segments again rather than the three that had been developed due to the first change but this again required rebuilding, modifying and regenerating everything.

The new operating system, RDOS, had the overlaying capability built into it allowing the operating system to produce the overlays upon demand rather than making the user handle the effort.

As a result, the Overlay routine, the Smash routine and the Linkage routines were no longer needed to allow the production of overlays and the increase in main memory size made it no longer necessary to have any overlays at all since the operating system, the graphics routines and the user's problem program could now all reside in main memory at the same time.

The NOVA's new configuration, in effect, made the majority of the conversion effort expended up to that point in time redundant and the only justification for this report is that the conversion effort as made would be of use if it was to be recreated at some other installation or at least under another small machine environment that was very similar to the original NOVA configuration.

Standard NOVA Software

The standard software routines such as the text editor and the relocatable loader can best be described as primitive and as a result quite difficult to use particularly in view of the fact that it would quite regularly do something unexplainable and usually quite destructive.

Graphics Software

There were many logic flaws encountered, the majority of which were introduced during copying of the source code from hard copy (both the first time and subsequent times) while others insinuated themselves during the process of modifying the source for the difference in word size between the GE635 and the NOVA.

Hardware Support

The amount of time that one or more pieces of hardware was "down", inoperative, would have to have been experienced to be believed.

The unreliability of the high speed paper tape punch/reader coupled with the previously mentioned system software and standard NOVA software unreliability caused not a few total restarts from hard copy (teletype or selectric printed output of source files).

The portion of time that the entire system was operational was not of adequate duration to allow productive research.

This is probably a normal situation, particularly in a research environment, where only a very few users place demand on a relatively inexpensive system and there isn't enough pressure monetarily to rate immediate attention to breakdowns as normal in a large machine environment.

EVALUATION OF PROJECT RESULTS

The original graphics routines were designed with one objective being to make them easily portable and such would quite likely have been the case if the conversion effort had been from one large machine environment to another large machine environment.

The fact that they were portable from a large machine environment to a minicomputer environment, albeit with some difficulty, indicates that they were properly designed.

Had they been poorly designed the conversion effort made in this research would have been virtually impossible.

The problems addressed in this paper are common to many minicomputer environments and as such this effort has been both instructive and of potential future value.

It is proposed that the minicomputer environment, while remaining a useful tool in the execution of fully developed software packages, is excessively difficult to use in the actual developmental stages though obviously not impossible.

Depending upon the level of capabilities built into a minicomputer system, software development could be feasible but at present the large machine is the best software engineering tool for development of software for minicomputers.

The software and hardware support of a large machine environment provide the researcher a much firmer basis from which to pursue the actual problems postulated by his research.

CHAPTER TWO

GRAPHICS SOFTWARE ORGANIZATION

It was assumed that the user's problem programs would exhibit locality within certain basic types of graphics routines according to their functions and that by spliting the graphics routines into groups based on these function types the user could perform several, hopefully numerous, manipulations before having to call in a new overlay segment.

There appeared to be three basic function types that the graphics routines naturally segregated into and these were called Constructors, Compilers and Transformers.

Constructors

The constructors are the primitives by which all images are formed. They build arrays that store text information and an array called a "Pseudo Display File", abbreviated as "PDF", which is an "N" by "4" array, where:

PDF(I,*) = (Operation Code,X,Y,Z).

The constructor routines and their functions are as follows:

CLEAR	Clear the CRT screen.
START	Start a new picture, PDF, file.
WMODE	Enter write mode.
EMODE	Enter erase mode.
MOVES	Move the cursor with no trace.
VEC	Move the cursor with a straight line trace.
HTEXT	Display text horizontally from the current cursor position.
VTEXT	Display text vertically from the current cursor position.
SENDPDF	Marks the end of one picture, PDF, file.

See Appendix "B" for the source code for these routines and for the arguments that each routine expects when called.

Compilers

The compilers are routines which translate an "image" segment in the "Pseudo Display File" into an output buffer of code specifically formated for the Computek 300 graphics terminal and then initiates the actual I/O to get it there.

The compiler routines and their functions are as follows:

COMPIL	Translates an "image" segment.
GETC	Used by COMPIL to get a byte for translation from either the PDF or the Text arrays.
PUTC	Used by COMPIL to put the translated byte into the output buffer.
BSEND	Used by COMPIL to initiate the actual output to the Computek of the output buffer.

See Appendix "C" for the source code for these routines and for the arguments that each routine expects when called.

Transformers

The transformers are routines which build a "4" by "4" application matrix (array), called "T", which is applied to the "PDF" array thus causing a uniform transformation of the picture contained therein.

These routines all transform coordinate axes in the convention of Newman and Sproull.[1]

The transformation routines and their functions are as follows:

INIT	Initializes the transformation.
ROTATE	Performs axes rotation.
SCALE	Performs arbitrary scaling.
TRANS	Performs translation of the coordinate axes.
MATMUL	Used by ROTATE, SCALE and TRANS for matrix manipulation.
DAPPLY	Applies the transformation to the PDF.

See Appendix "D" for the source code for these routines and for the arguments that each routine expects when called.

1 - PRINCIPLES OF INTERACTIVE COMPUTER GRAPHICS, Newman and Sproull.

USAGE OF THE GRAPHICS PACKAGE

In order to use the minicomputer graphics package the user must know what functions are available (as explained under Graphics Software Organization), how to call the overlay routine including what graphics routines are in which overlay segments and how to use the NOVA's text editor, fortran compiler and relocatable loader.

Overlay Routine Calls

The overlay routine must be called by the user in his problem program every time that he needs a graphics routine that is not in the overlay segment that is currently resident in main memory.

It should be noted that the overlay routine checks to determine which overlay segment file is currently in main memory and will not waste time pulling in an overlay segment file that is requested by the user if it is already resident.

The following code shows the necessary call sequences for a hypothetical segment of a user's problem program assuming that overlay segment file "A" contains routines "a", "b" and "c" and that overlay segment file "B" contains routines "x", "y" and "z".

```
     :
     :
     :
CALL OVER("A")
CALL b
CALL OVER("B")
CALL x
CALL z
CALL OVER("A")
CALL c
CALL a
     :
     :
```

Overlay Segment Files

The constructor routines and the compiler routines were all placed in an overlay segment file named "CONSTRUCT" and the transformer routines were placed in an overlay segment file named "TRANSFORM".

The reason for grouping the individual routines by function type has already been discussed and the reason for grouping the constructor routines with the compiler routines was not because of any similarity of function but rather because the amount of object code in the transformer routines was considerably larger than either of the other two groups; thus the way to obtain the smallest maximum sized overlay segment was to place the two smaller groups together in one overlay segment file.

Contents of "CONSTRUCT".	Contents of "TRANSFORM".
CLEAR	INIT
START	ROTATE
WMODE	SCALE
EMODE	TRANS
MOVES	MATMUL
VEC	DAPPLY
HTEXT	
VTEXT	
COMPIL	
GETC	
PUTC	
BSEND	

See Appendix "E" for example user's problem programs first without calls to the overlay routine, as if all graphics were resident in main memory concurrently, and then with calls to the overlay routine inserted as required in order to run the user's problem program in the limited main memory available.

NOVA's Text Editor

Creation of a problem program source file on the NOVA is normally accomplished using the text editor routine.

The following gives a very simple example of how the text editor is initiated and used to create a hypothetical source file (lower case letters in example indicate where the user has an option of what to use in his application).

```
EDIT
*GWname.FR$$
*I    .
      .
      .
      source
      statements
      .
      .
      .
$PH$$
```

The above illustration would have created a source file named as whatever the user had inserted in place of "name" that contained the source statements the user had developed.

NOVA's Fortran Compiler

To create a relocatable binary file from the user's source file requires only the initiation of the fortran compiler which can be illustrated by the following very simple command string (lower case letters used as previously defined).

```
FORT name.FR
```

This creates a relocatable binary file with the same name as the user selected in creating the source file except that the suffix "FR" is replaced by the suffix "RB".

NOVA's Relocatable Loader

To create an executable core image "save" file for the user's problem program requires the use of the relocatable loader, the two routines developed for support of the minicomputer graphics package, USERLINK.RB and OVERLAY.RB, the relocatable binary file previously created from the user's problem program and the library file that contains the fortran subroutines.

The following command string illustrates the exact method needed to accomplish this final step.

RLDR 1000/N name.SV/S USERLINK.RB OVERLAY.RB name.RB SYS.LB 23240/N[1]

This creates a core image file, a "save" file, that has the same name as the user selected in creating the source file except that the suffix "FR" is replaced by the suffix "SV".

Problem Program Execution

The execution of the user's problem program is now accomplished by entry of the name that the user selected in creating the source file as a command.

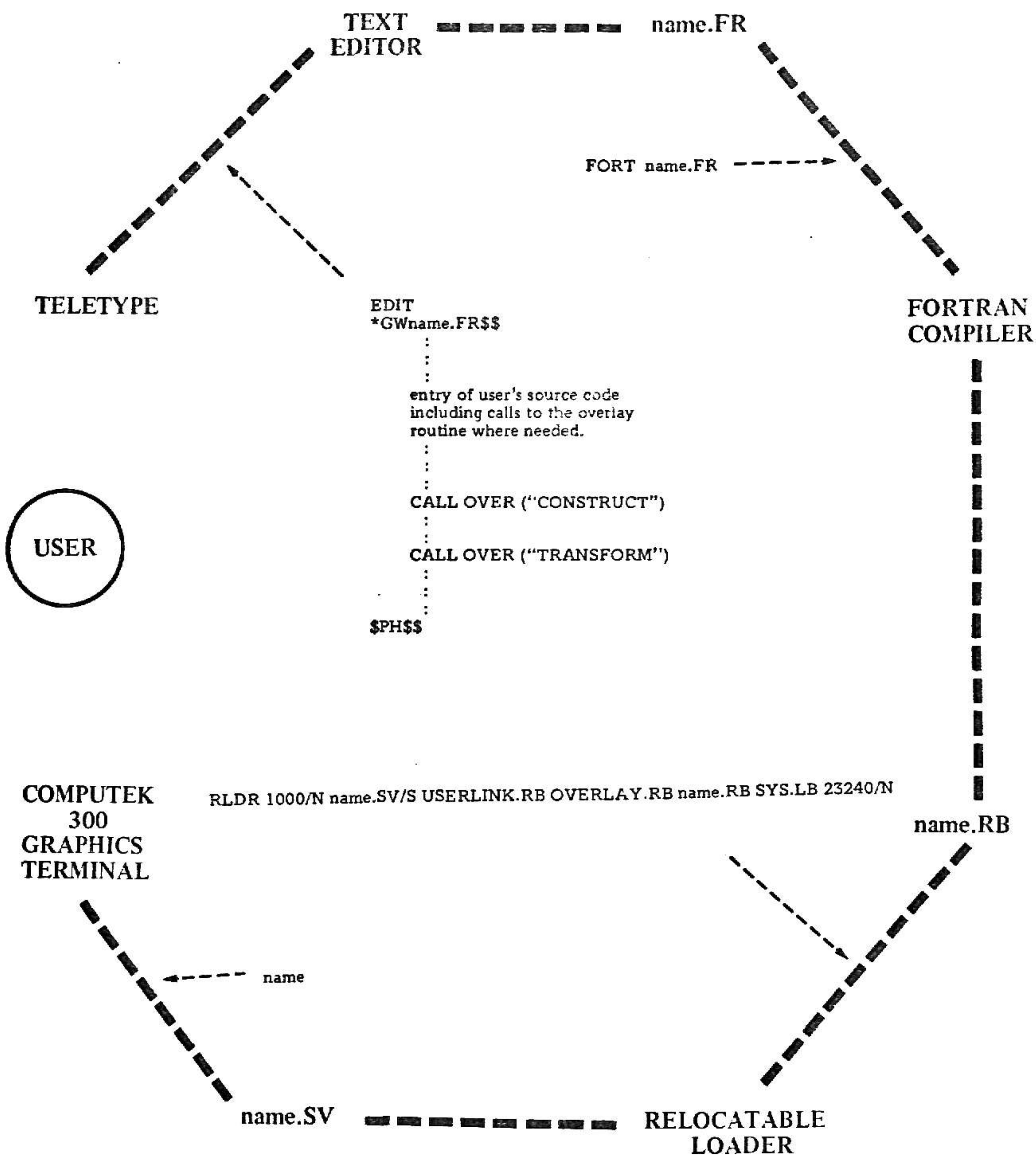
If the name chosen had been "testmain" for example then the following command would load the user's problem program into main memory and commence its execution.

TESTMAIN

Assuming that the user had not failed to insert the proper calls to the overlay program in his source code then the result of this command will be precisely whatever the user's source code indicates they should be.

1: See Appendix "G" for detailed explaination of components of this.

FIGURE FOUR : GRAPHICS PACKAGE USAGE DIAGRAM



CHAPTER THREE

CONSTRUCTION OR MODIFICATION OF GRAPHICS PACKAGE

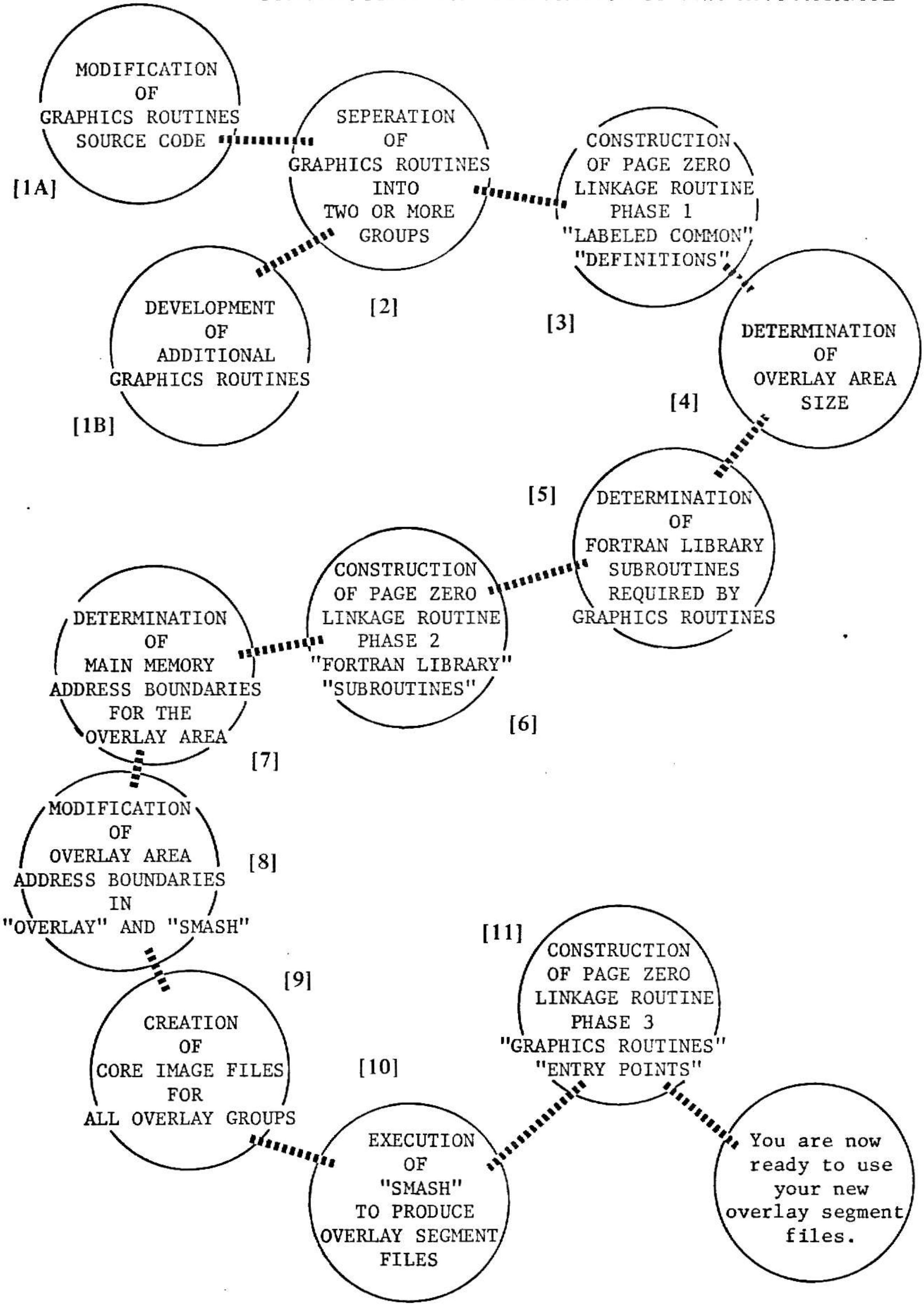
This chapter is provided not only as documentation of the exact procedure used in constructing the minicomputer graphics package but also as a guide on how to modify said package since in order to make any modification the same procedure would have to be followed.

The addition of new graphics routines, modification of the size of any of the graphics arrays (such as the TEXT and PDF arrays) in labeled common or even simply changing one word in an existing graphics routine would cause such things as; a change in the overlay area size causing the address constants in OVERLAY and SMASH to be invalid, the addition of new entry points that would be undefined in the USERLINK routine thus destroying page zero's accuracy, the shift of an entry point to an existing routine (even if only by one word) and the demand for some fortran library routine not previously required that would not be in main memory for use when the graphics routine called for it.

A procedure diagram, Figure Five, provides a good overview of what is required in setting up the easily used graphics package as indicated in Chapter Two and also provides a step by step guide for the layout of this chapter.

FIGURE FIVE : PROCEDURE DIAGRAM for

CONSTRUCTION OR MODIFICATION OF GRAPHICS PACKAGE



1A: Modification of Graphics Routines Source Code

Any existing graphics routine can be modified by following the few steps illustrated in this section.

First rename the original file with the command string:

```
RENAME  original-name.FR  temporary-name
```

Then by using the text editor to actually insert the modified code in the selected graphics routine as shown in the following short example.

```
EDIT
*GRtemporary-name$Y$$
*T$$
        :
        :
        INDEX = 1
        :
        :
*2L$1T$$
        INDEX = 1
*C1$2$$
*T$$
        :
        :
        INDEX = 2
        :
        :
*GWoriginal-name.FR$PH$$
```

Finally, to insure that the desired modification is actually reflected in the relocatable binary file for the selected graphics routine the modified source file must be recompiled using the fortran compiler, which would be initiated using the following command string.

```
FORT  original-name.FR
```

1B: Development of Additional Graphics Routines

The text editor can be used to develop new graphics routines in precisely the same manner as was exemplified in the development of the user's problem program in Figure Four.

2: Seperation of Graphics Routines Into Two or More Groups

The decision on how many groups to break the graphics routines into was reached by a bit of trial and error in which it was finally determined that approximately a 50-50 split of the object code for the graphics routines would allow room in main memory for all the other necessary space allocations, page zero, overlay routine, user's problem program, fortran library subroutines, common area, fortran's runtime stack and the operating system.

Seperate the graphics routines into two or more groups attempting to maintain some similarity in overall size but insuring above all, in so far as possible, the minimum interaction between groups when the user's problem program is in execution.

To prevent oversights and typographical errors later on during the entry of command strings the creation, using the text editor, of files that contain the names of the graphics routines, one file for each group, is most helpful.

The creation of such an "indirect command string" file, the way to actually use the file and the results of using the file are illustrated by the following example.

```
EDIT
*GWgroup-name1$$
*Iname1.RB name2.RB name3.RB$$
*PH$$

LIST @group-name1@
name1.RB          124
name2.RB          375
name3.RB          436

LIST name1.RB name2.RB name3.RB
name1.RB          124
name2.RB          375
name3.RB          436
```

3: Construction of Page Zero Linkage Routine

Phase 1 - "Labeled Common Definitions"

The length of the labeled common blocks must first be determined. The simplest method is to choose either a graphics routine, a user's problem program or a dummy fortran program that contains all the labeled common blocks and execute the fortran compiler on it with the assembler source file option switch on by entry of the following command string.

```
FORT/S program-name.FR
```

Then printing out the resulting assembler source file, it will have the same program name as "program-name.FR" except that the suffix will have changed to "SR", will provide the specific information needed.

```
TYPE program-name.SR
       :
       :
       :
      .COMM     common-name1     41
      .COMM     common-name2     4541
       :
       :
       :
```

Now the first phase of construction of the linkage routine can be carried out using the text editor again.

```
EDIT
*GWphase1-name.SR$$
*I     .TITL     LINK
       .NREL
       .COMM     common-name1     41
       .COMM     common-name2     4541
       .END
$PH$$
```

Finally, to produce a relocatable binary file for the linkage routine the use of NOVA's assembler is required and can be initiated using the following command string.

ASM phasel-name.SR

4: Determination of Overlay Area Size

Using the relocatable loader the exact size, extent, of each group of graphics routines can be determined thusly providing, by selection of the largest of the resulting values, the overall size necessary to be set aside for the overlay area.

The following example illustrates the required command strings and the results of each.

```
RLDR temporary-name/S phasel-name.RB 10000/N @group-name1@
         :
         :
         :
       NMAX     22474
         :
         :
         :

RLDR temporary-name/S phasel-name.RB 10000/N @group-name2@
         :
         :
         :
       NMAX     23237
         :
         :
         :
```

By selecting the largest resulting NMAX value and subtracting the starting address, 10,000, from it the overlay area extent results.

The 10,000 address was selected arbitrarily since it was known to be larger that the maximum address that would be allocated to labeled common, common's load point address of 1,000 plus the extent of all labeled commons.

5: Determination of Fortran Library Subroutines Required by Graphics Routines

In the previous step the execution of the relocatable loader produced a listing called a "load map" and the fortran library subroutines needed by the graphics routines are listed therein and may be recognized by the fact that their entry points are flagged, "U", as undefined.

An example of what to look for follows.

```
RLDR temporary-name/S phase1-name.RB 10000/N @group-name1@
          :
          :
          :
          common-name1                        005541
          common-name2                        001000
          graphics-routine-entry-name1        010000
          graphics-routine-entry-name2        010473
        U library-routine-entry-name1         010742
          graphics-routine-entry-name3        011144
        U library-routine-entry-name2         011273
        U library-routine-entry-name3         012215
        U library-routine-entry-name4         012277
          :
          :
          :
```

By taking the union of undefined entry points from both load maps the construction of phase 2 of the page zero linkage routine becomes possible.

The names of the fortran library subroutines themselves which one or more of the library routine entry names provide access to are of no particular interest to the researcher since the inclusion of the entry names in the page zero linkage routine will cause the desired library routines to be loaded by the relocatable loader without having to know the fortran library subroutine names specifically.

6: Construction of Page Zero Linkage Routine

Phase 2 - "Fortran Library Subroutines"

The expansion of the phase1-name.SR file to create a new linkage routine by the addition of external definition statements for all of the undefined entry points determined in step 5 is possible by using the text editor again in the manner illustrated by the following example.

```
EDIT
*GRphase1-name.SR$Y$$
*2L$$
*I      .EXTD      entry-name1, entry-name2
        .EXTD      entry-name3, entry-name4
$T$$
        .TITL      LINK
        .NREL
        .EXTD      entry-name1, entry-name2
        .EXTD      entry-name3, entry-name4
        .COMM      common-name1      41
        .COMM      common-name2      4541
        .END
*GWphase2-name.SR$$
*PH$$
```

Again, to produce a relocatable binary file for the linkage routine the use of NOVA's assembler is required and can be initiated using the following command string.

```
ASM phase2-name.SR
```

7: Determination of Main Memory Address Boundaries for the Overlay Area

This is a very critical step since if the placement of the overlay are in main memory is too low there won't be adequate memory available to execute the user's problem program and if too high the fortran runtime stack will not have room between NMAX and the operating system.

The relocatable loader is of use in the determination of the amount of main memory that will be used by labeled common, the overlay routine and the fortran library subroutines required by the graphics routines and such information is necessary in order to know exactly where the user's problem program work area will begin.

The following example gives the necessary command string to initiate the relocatable loader and the results that will be obtained.

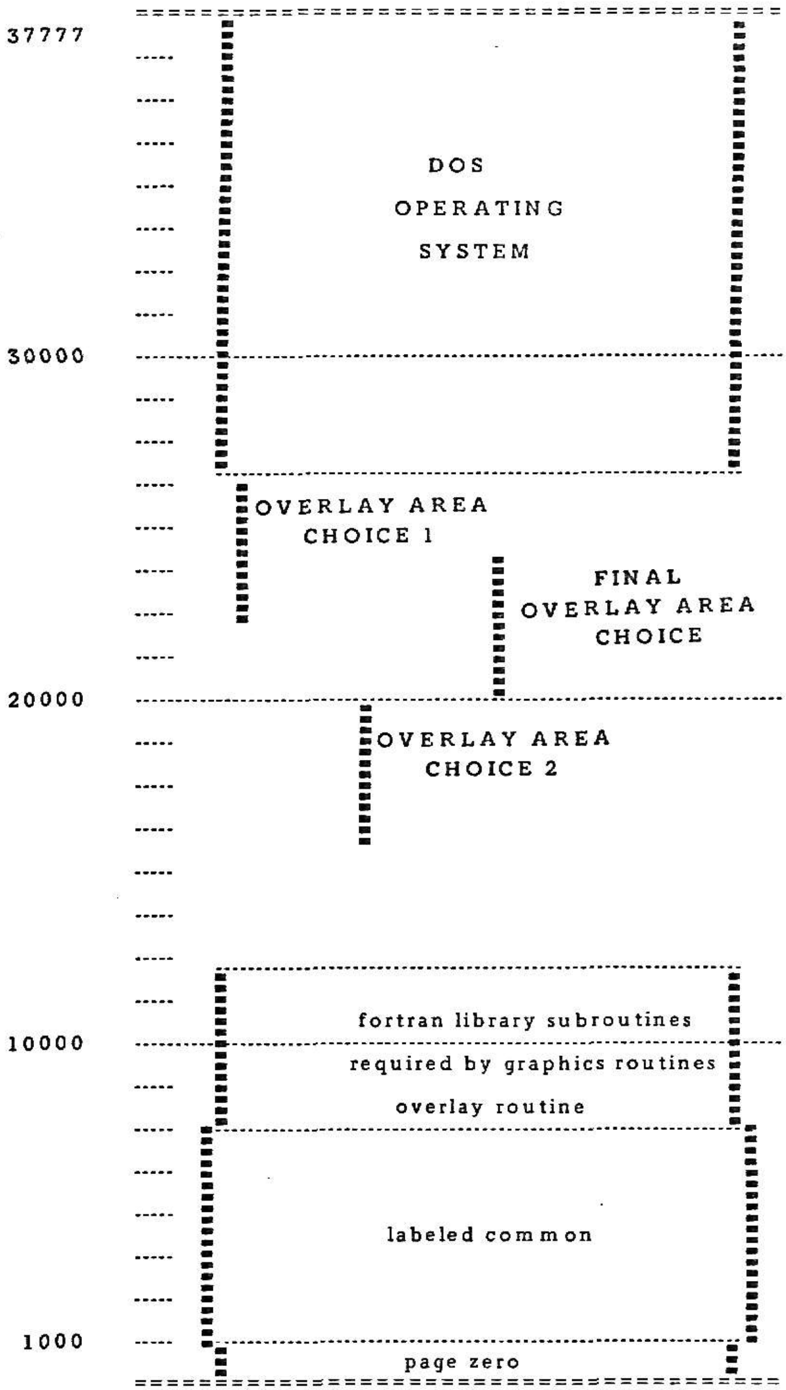
```
RLDR temporary-name/S 1000/N OVERLAY.RB phase2-name.RB SYS.LB
          :
          :
          :
        NMAX     11516
          :
          :
          :
```

The value of NMAX gives you the amount of main memory required by labeled common, the overlay routine and the library subroutines and therefore, the point where the user's problem program will be loaded.

The positioning of the overlay area must be such that some main memory is left between the top of the overlay area and the bottom of the operating system and such that as much space as possible is left between the bottom of the overlay area and the point where the user's problem program will be loaded so as to allow the execution of the largest feasible problem program.

Figure Six on the following page gives a graphical view of the considerations that were made in selecting the address of 20000 as the beginning of the overlay area.

FIGURE SIX : OVERLAY ADDRESS BOUNDARY SELECTION



CHOICE 1: Too high, not enough room for Fortran's runtime stack to fit inbetween the top of the overlay area, NMAX, and the bottom of the operating system.

CHOICE 2: Too low, not enough room for the user's problem program and the and the fortran library subroutines that it may require.

FIGURE SEVEN : EXAMPLE MAIN MEMORY ADDRESS MAP

It should be clearly understood that the following main memory map is from one specific test case and as such will not be the same as the map that the user might develop for a different problem program.

Many of the addresses would change but the relative order of the items described would not change.

00000-00377	Page "Zero".
00400-00427	User Status Table.**
01000-01040	COMMON/TRSF/.
01041-05601	COMMON/GRAF/.
05602-05667	Overlay Routine.
05670-11515	System Subroutines for Graphics Routines.
11516-14372	User's Problem Program.
14373-15725	System Subroutines for User's Problem Program (those not already loaded for graphics routines).
15726-17777	(unused).
20000-23237	Overlay Area.
23240-varying*	Fortran's Runtime Stack.
25116-37777	DOS Operating System (includes the bootstrap and binary loaders).

*note: If the runtime stack runs up into the DOS area during execution of the user's problem program a fatal fortran runtime error occurs.

**note: For the definition of the user status table refer to Appendix "G".

8: Modification of Overlay Area Address Boundaries in "OVERLAY" and "SMASH"

Since the low and high addresses of the overlay area exist as constants in both the overlay routine and the smash routine, any change of either value requires that the address constants, .OVLO and/or .OVHI, be changed correspondingly.

There are two methods to accomplish this.

There exists a routine, called the "octal editor", that provides for the direct modification of any type of file, not just text files, and the value that needs changed could be modified in a relocateable binary file, such as OVERLAY.RB, or in a core image file, such as SMASH.SV, but the location of the precise word within each file that contains the value to be modified, .OVLO and/or .OVHI, are not readily apparent and for this reason it is normally quicker and more accurate to alter the source code files using the text editor.

First rename the original source files with the following two command strings.

```
RENAME  OVERLAY.SR  temporary-name1
RENAME  SMASH.SR  temporary-name2
```

Then the use of the text editor to actually insert the modifications is illustrated in the following two short examples where it is assumed that only the value of .OVHI has changed.

```
EDIT
*GRtemporary-name1$Y$$
*S.OVHI:$L$1T$$
.OVHI:      old-value
*Cold-value$new-value$L$1T$$
.OVHI:      new-value
*GWOVERLAY.SR$PH$$
```

```
EDIT
*GRtemporary-name2$Y$$
*S.OVHI:$L$1T$$
.OVHI:     old-value
*Cold-value$new-value$L$1T$$
.OVHI:     new-value
*GWSMASH.SR$PH$$
```

The use of the NOVA's assembler at this point, by the following command strings, will modify the relocatable binary files, OVERLAY.RB and SMASH.RB.

```
ASM  OVERLAY.SR
ASM  SMASH.SR
```

Finally the use of the relocatable loader, by the following command string, will modify the core image file, SMASH.SV.

```
RLDR  SMASH.RB
```

In all of the following illustrations the values of .OVLO and .OVHI will be assumed to be 20000 and 23237 respectively and if the actual values being worked with vary from these then all occurences of 20000 and 23240 in the example command strings must be replaced by the actual values being worked with.

9: Creation of Core Image Files for All Overlay Groups

The production of the actual executable code with all addresses resolved correctly may now be carried out using the relocatable loader as illustrated by the following two command strings.

```
RLDR segment-name1/S 1000/N OVERLAY.RB
     phase2-name.RB SYS.LB 20000/N @group-name1@

RLDR segment-name2/S 1000/N OVERLAY.RB
     phase2-name.RB SYS.LB 20000/N @group-name2@
```

The output from the relocatable loader as a result of the two command strings previously illustrated is a load map that contains the entry point definitions for the graphics routines themselves as illustrated by the following example.

```
  :
  :
  :
graphics-routine-entry-name1    020000
graphics-routine-entry-name2    020473
  :
  :
  :
```

The names that presently exist as entry points into the graphics routines are as follows:

CLEAR
EMODE
MOVES
SENDP
START
VEC
WMODE
DTEXT
HTEXT
VTEXT
COMPI
BSEND
PUTC
DAPPL
INIT
MATMU
ROTAT
SCALE
TRANS

The reason that these entry point names differ from the subroutine names for which they are entry points is due to an internal restriction in NOVA assembler language that all identifiers, entry points included, be five characters or less.

The entry point definitions produced in this step will be needed in step 11 where the page zero linkage routine is finally completed so be sure and save the relocatable loader's output listing.

10: <u>Execution of "SMASH" to Produce Overlay Segment Files</u>

The core image files produced in the previous step, segment-name1.SV and segment-name2.SV (called CONSTRUCT.SV and TRANSFORM.SV in this conversion effort), must be reduced to only the overlay portion of the file and placed in a new file under the names that the user expects when he calls OVERLAY with said names as arguements, for example CALL OVER ("CONSTRUCT") and CALL OVER ("TRANSFORM").

The execution of SMASH.SV on the two core image files, as illustrated by the following examples, will produce the final reduced files that the overlay routine needs to function properly.

```
SMASH
segment-name1.SV
segment-name1

SMASH
segment-name2.SV
segment-name2
```

Care must be taken when executing the smash routine since there are no prompt characters indicating that SMASH is waiting for first the entry of the file to be reduced, the input file, and second the entry of the file to be created, the output file, and as a result the user of the smash routine may sit quietly waiting after the entry of the command SMASH and not realize that the routine is waiting for input from the teletype.

11: Construction of Page Zero Linkage Routine

Phase 3 - "Graphics Routines Entry Points"

The production of the actual linkage routine to be used by the user of the graphics package is now possible since all graphics entry point names and definitions are known.

The actual linkage routine (called USERLINK in this conversion effort) is produced using the text editor to modify the "phase2-name" linkage routine, as illustrated by the following example, by the addition of entry definition statements for all of the graphics routines entry points and by the addition of entry value statements.

```
EDIT
*GRphase2-name$Y$$
*S.COMM$L$$
*I      .ENT      graphics-routine-entry-name1
        .ENT      graphics-routine-entry-name2
          :
          :
$S.END$L$$
*I      graphics-routine-entry-name1=20000
        graphics-routine-entry-name2=20473
          :
          :
$T$$
        .TITL
        .NREL
        .EXTD     entry-name1, entry-name2
        .EXTD     entry-name3, entry-name4
        .ENT      graphics-routine-entry-name1
        .ENT      graphics-routine-entry-name2
        .COMM     common-name1      41
        .COMM     common-name2      4541
        graphics-routine-entry-name1=20000
        graphics-routine-entry-name2=20473
        .END
*GWactual-linkage-routine-name.SR$$
*PH$$
```

Again, to produce a relocatable binary file for the final linkage routine, the one to be used by the user of the graphics package, the use of NOVA's assembler is required and can be initiated using the following command string.

ASM actual-linkage-routine-name.SR

<u>You are now ready to use your new overlay segment files</u>

You have generated new overlay segment files (such as TRANSFORM and CONSTRUCT) and a new linkage routine (such as USERLINK.RB).

You have modified the overlay boundaries (.OVLO and/or .OVHI) in OVERLAY.RB and you are now ready to use your new graphics package precisely as explained in Chapter Two's "USAGE OF THE GRAPHICS PACKAGE".

APPENDIX "A"

SOURCE CODE

FOR

OVERLAY.SR

SMASH.SR

USERLINK.SR

MYLINK.SR

```
;
;        FILENAME:  OVERLAY.SR
;
;        OVERLAY ROUTINE ("FINAL VERSION").
;
;        PURPOSE:    TO PULL IN FROM DISK ANY OVERLAY SEGMENT FILE NOT
;                    ALREADY RESIDENT IN MAIN MEMORY AND OVERLAY THE
;                    PORTION OF MAIN MEMORY SPECIFIED BY THE VALUES OF
;                    .OVLO AND .OVHI AND PLACING SAID FILE IN MEMORY
;                    WITHOUT AFFECTING ANY OTHER PORTION OF THE CURRENT
;                    CORE IMAGE.
;
         .TITL    OVER
         .EXTD    .CPYL,  .FRET
         .ENT     .OVLO,  .OVHI,  OVER
         .NREL
         1
OVER:    JSR      @.CPYL
         LDA      2,C7
         STA      2,COUNT    ;COUNT WORDS TO COMPARE.
         STA      3,TEMP     ;SAVE AC3.
         LDA      3,T.,3     ;GET ADDRESS OF NAME.
         LDA      2,ALAST    ;GET ADDRESS OF CURRENT OVERLAY NAME.
ONE:     LDA      0,0,2      ;GET NEXT TWO CHARACTERS OF OLD NAME.
         LDA      1,0,3      ;GET NEXT TWO CHARACTERS OF NEW NAME.
         SUB#     0,1,SZR    ;SKIP IF THE SAME.
         JMP      TWO        ;DIFFERENT NAME.
         INC      2,2
         INC      3,3
         DSZ      COUNT      ;SKIP IF NAMES EQUAL.
         JMP      ONE
         JSR      @.FRET     ;NAMES WERE EQUAL - RETURN.
TWO:     LDA      3,TEMP
         LDA      3,T.,3     ;GET ADDRESS OF NEW NAME.
         LDA      2,ALAST
         LDA      0,C7
         STA      0,COUNT
THREE:   LDA      1,0,3      ;MOVE ONE WORD.
         STA      1,0,2
         INC      3,3        ;BUMP POINTERS.
         INC      2,2
         DSZ      COUNT      ;DONE?
         JMP      THREE      ;NO!
         LDA      0,ALAST    ;YES!
         MOVZL    0,0        ;GET BYTE POINTER TO NAME.
         SUB      1,1        ;CLEAR INHIBITS.
         .SYSTM
         .OPEN    7          ;OPEN FILE.
         JMP      ERROR
```

```
        LDA     0,.OVLO
        LDA     1,.OVHI
        SUB     0,1
        INCZL   1,1         ;GET BYTE COUNT.
        MOVZL   0,0         ;GET BYTE POINTER TO AREA.
        .SYSTM
        .RDS    7           ;READ OVERLAY SEGMENT FILE.
        JMP     ERROR
        .SYSTM
        .CLOSE  7           ;CLOSE OVERLAY SEGMENT FILE.
        JMP     ERROR
        JSR     @.FRET
ERROR:  .SYSTM
        .ERTN
        HALT
;
;  NOTE: .OVLO IS THE BEGINNING ADDRESS OF THE OVERLAY AREA AND
;        .OVHI IS THE MAXIMUM ADDRESS OF THE OVERLAY AREA ("NMAX"
;        MINUS ONE) AND MOST CERTAINLY WILL NEED TO BE CHANGED
;        WHENEVER ANY CHANGES ARE MADE TO THE GRAPHICS PACKAGE.
;
.OVLO:  20000
.OVHI:  23237
ALAST:  .+1
LAST:   .BLK    7           ;NAME OF OVERLAY FILE CURRENTLY IN MEMORY.
C7:     7
COUNT:  .BLK    1
TEMP:   .BLK    1
        T.=-167
        .END
```

```
;
;        FILENAME:  OVERLAY.SR
;
;        "ORIGINAL" OVERLAY ROUTINE.
;
;        PURPOSE:   TO PULL IN FROM DISK THAT PORTION OF THE SAVE FILE
;                   (CORE IMAGE FILE) SPECIFIED BY THE VALUES OF .OVLO
;                   AND .OVHI AND PLACING SAID SEGMENT IN MEMORY WITHOUT
;                   AFFECTING ANY OTHER PORTION OF CURRENT CORE IMAGE.
;
        .TITL   OVER
        .EXTD   .CPYL,  .FRET
        .ENT    OVER,   .OVLO,  .OVHI
        .NREL
        1
OVER:   JSR     @.CPYL
        LDA     0,T.,3
        LDA     1,.LAST
        SUB#    1,0,SNR
        JSR     @.FRET
        STA     0,.LAST
        MOVZL   0,0
        SUB     1,1
        .SYSTM
        .OPEN   7
        JMP     ERROR
        LDA     0,.OVLO
        LDA     1,.OVHI
        SUB     0,1
        INCZL   1,1
        LDA     2,.SUBS
        SUBL#   0,2,SNC
        JMP     THREE
        SUB     0,2
        NEGZL   2,2
        MOVZL   0,0
ONE:    ADCL#   1,2,SZC
        JMP     TWO
        .SYSTM
        .RDS    7
        JMP     ERROR
        SUB     1,2
        JMP     ONE
TWO:    STA     1,.TEMP
        MOV     2,1
        .SYSTM
        .RDS    7
        JMP     ERROR
        LDA     1,.TEMP
        MOV     0,0,SKP
```

```
THREE:   MOVZL    0,0
         .SYSTM
         .RDS     7
         JMP      ERROR
         .SYSTM
         .CLOSE   7
         JMP      ERROR
         JSR      @.FRET
;
;  NOTE: .OVLO IS THE BEGINNING ADDRESS OF THE OVERLAY AREA AND
;        .OVHI IS THE MAXIMUM ADDRESS OF THE OVERLAY AREA ("NMAX"
;        MINUS ONE) AND MOST CERTAINLY WILL NEED TO BE CHANGED
;        WHENEVER ANY CHANGES ARE MADE TO THE GRAPHICS PACKAGE.
;
.OVLO:   20000
.OVHI:   23237
.TEMP:   .BLK     1
.LAST:   .BLK     1
.SUBS:   16
         T.=-167
ERROR:   .SYSTM
         .ERTN
         HALT
         .END
```

```
;
;       FILENAME:   SMASH.SR
;
;       PURPOSE:    TO REDUCE AN OVERLAY SAVE FILE
;                   (CORE IMAGE FILE) TO ONLY THE
;                   OVERLAY SEGMENT ITSELF.
;
        .TITL   SMASH
        .ENT    .OVLO,.OVHI
        .NREL
SMASH:  .SYSTM
        .MEM
        JMP     ERROR
        SUB     1,0
        .SYSTM
        .MEMI
        JMP     ERROR
        LDA     0,BPTTI   ;='$TTI'.
        SUB     1,1       ;CLEAR INHIBITS.
        .SYSTM
        .OPEN   1         ;OPEN CHANNEL 1.
        JMP     ERROR
        LDA     0,BPIN    ;BYTE POINTER TO INPUT FILE NAME.
        .SYSTM
        .RDL    1
        JMP     ERROR
        .SYSTM
        .OPEN   2         ;OPEN CHANNEL 2.
        JMP     ERROR
        LDA     0,BPOUT   ;BYTE POINTER TO OUTPUT FILE NAME.
        .SYSTM
        .RDL    1
        JMP     ERROR
        .SYSTM
        .CREAT
        JMP     ERROR
        .SYSTM
        .OPEN   3         ;OPEN CHANNEL 3.
        JMP     ERROR
        .SYSTM
        .CLOSE  1         ;CLOSE $TTI.
        JMP     ERROR
        LDA     0,.OVLO
        LDA     1,.OVHI
        SUB     0,1
        INCZL   1,1       ;GET BYTE SIZE OF OVERLAY AREA.
        STA     1,TEMP    ;SAVE BUFFER SIZE.
        LDA     2,.SUBS   ;BASE OF SAVE FILE.
        SUBL#   0,2,SNC   ;SKIP IF NOT AT .OVLO.
        JMP     THREE
```

```
        SUB     0,2       ;GET NUMBER OF WORDS TO SKIP.
        NEGZL   2,2       ;BYTES TO SKIP.
        MOVZL   0,0       ;BYTE ADDRESS OF BUFFER.
ONE:    ADCL#   1,2,SZC   ;MORE BYTES THAN BUFFER?
        JMP     TWO       ;NO.
        .SYSTM
        .RDS    2         ;YES, READ A BUFFER FULL.
        JMP     ERROR
        SUB     1,2       ;DECREMENT BYTES TO SKIP COUNTER.
        JMP     ONE       ;AND REPEAT.
TWO:    MOV     2,1       ;GET REMAINING BYTE COUNT.
        .SYSTM
        .RDS    2         ;READ RIGHT UP TO .OVLO.
        JMP     ERROR
        LDA     1,TEMP    ;RESTORE BUFFER SIZE.
        MOV     0,0,SKP   ;SKIP BYTE POINTER CREATION.
THREE:  MOVZL   0,0       ;CREATE BYTE POINTER TO BUFFER.
        .SYSTM
        .RDS    2         ;READ OVERLAY AREA.
        JMP     .+2
        JMP     OK
        LDA     1,TEMP    ;RESTORE BUFFER SIZE.
        LDA     3,C6
        SUB#    3,2,SNR   ;CHECK FOR END OF FILE.
        JMP     OK
;------------------------------------------------------------------
ERROR:  .SYSTM
        .ERTN
        HALT
;------------------------------------------------------------------
OK:     .SYSTM
        .WRS    3         ;WRITE BUFFER TO NEW FILE.
        JMP     ERROR
        .SYSTM
        .RESET
        HALT
        .SYSTM
        .RTN
        HALT
;------------------------------------------------------------------
;  NOTE: .OVLO IS THE BEGINNING ADDRESS OF THE OVERLAY AREA AND
;        .OVHI IS THE MAXIMUM ADDRESS OF THE OVERLAY AREA ("NMAX"
;        MINUS ONE) AND MOST CERTAINLY WILL NEED TO BE CHANGED
;        WHENEVER ANY CHANGES ARE MADE TO THE GRAPHICS PACKAGE.
;------------------------------------------------------------------
.OVLO:  20000
.OVHI:  23237
BPTTI:  2*.+2
        .TXTM   1
        .TXT    "$TTI"
```

```
BPIN:     2*.+2
          .BLK     20          ;INPUT FILE NAME.
BPOUT:    2*.+2
          .BLK     20          ;OUTPUT FILE NAME.
.SUBS:    16                   ;SAVE FILE BEGINNING ADDRESS.
TEMP:     .BLK     1
C6:       6
          .END     SMASH
```

```
;
;       FILENAME:  USERLINK.SR
;
;       PURPOSE:   PHASE THREE PAGE ZERO LINKAGE ROUTINE USED
;                  TO ACTUALLY FORCE THE RELOCATEABLE LOADER TO
;                  CREATE THE SAME PAGE ZERO ADDRESSES FOR THE
;                  USER'S CORE IMAGE FILE AS WAS CREATED FOR THE
;                  TWO OVERLAY SEGMENT FILES, "CONSTRUCT" AND
;                  "TRANSFORM".
;
        .TITL   LINK
        .NREL
        .EXTD   .FRED,  .FALO,  .FSUB,  .FSBR,  .CGT,  MO.
        .EXTD   IA.S,   .SMPY,  .SDVD,  IF.X,   XI.X,  SI.N
        .EXTD   .FARG,  .FRGL,  .FRG0,  .FRG1,  DB.E,  SN.L
        .EXTD   FL.AT,  .LD0,   .LD1,   .LD2,   .ST0,  .ST1
        .EXTD   .ST2,   .STOP,  .PAUS,  .FCAL,  .FSAV, .FRET
        .EXTN   .RTER,  .RTE0,  .RTES,  .WRCH,  .COUT, .CIN
        .EXTD   .LDBT,  .STBT,  .MOVE,  .CPYA,  .CPYL, .MAD
        .EXTD   .MADO,  SUCOM,  .SOSW,  .NDSP,  AFSE,  .IOCA
        .EXTD   SP,     .OVFL,  .SV0,   QSP,    NSP,   FLSP
        .EXTD   SIN.,   COS.
        .ENT    CLEAR,  EMODE,  MOVES,  SENDP,  START, VEC
        .ENT    WMODE,  DTEXT,  HTEXT,  VTEXT,  COMPI, BSEND
        .ENT    PUTC,   DAPPL,  INIT,   MATMU,  ROTAT, SCALE
        .ENT    TRANS
        .COMM   TRSF    41
        .COMM   GRAF    4541
        CLEAR=20042
        EMODE=20137
        MOVES=20234
        SENDP=20371
        START=20467
          VEC=20547
        WMODE=20703
        DTEXT=21005
        HTEXT=21173
        VTEXT=21317
        COMPI=21440
        BSEND=23024
         PUTC=23152
        DAPPL=20057
         INIT=20334
        MATMU=20476
        ROTAT=20735
        SCALE=21417
        TRANS=21675
        .END
```

```
;
;       FILENAME:   MYLINK1.SR
;
;       PURPOSE:    PHASE ONE PAGE ZERO LINKAGE ROUTINE USED
;                   TO DETERMINE THE OVERLAY AREA SIZE.
;
        .TITL    LINK
        .NREL
        .COMM    TRSF     41
        .COMM    GRAF     4541
        .END
```

```
;
;       FILENAME:   MYLINK2.SR
;
;       PURPOSE:    PHASE TWO PAGE ZERO LINKAGE ROUTINE USED
;                   TO DETERMINE THE AMOUNT OF MAIN MEMORY USED
;                   BY THE FORTRAN LIBRARY SUBROUTINES REQUIRED
;                   BY THE GRAPHICS ROUTINES.
;
        .TITL    LINK
        .NREL
        .EXTD    .FRED,   .FALO,   .FSUB,   .FSBR,   .CGT,    MO.
        .EXTD    IA.S,    .SMPY,   .SDVD,   IF.X,    XI.X,    SI.N
        .EXTD    .FARG,   .FRGL,   .FRG0,   .FRG1,   DB.E,    SN.L
        .EXTD    FL.AT,   .LD0,    .LD1,    .LD2,    .ST0,    .ST1
        .EXTD    .ST2,    .STOP,   .PAUS,   .FCAL,   .FSAV,   .FRET
        .EXTD    .RTER,   .RTE0,   .RTES,   .WRCH,   .COUT,   .CIN
        .EXTD    .LDBT,   .STBT,   .MOVE,   .CPYA,   .CPYL,   .MAD
        .EXTD    .MADO,   SUCOM,   .SOSW,   .NDSP,   AFSE,    .IOCA
        .EXTD    SP,      .OVFL,   .SVO,    QSP,     NSP,     FLSP
        .EXTD    SIN.,    COS.
        .COMM    TRSF     41
        .COMM    GRAF     4541
        .END
```

APPENDIX "B"

SOURCE CODE

FOR

CONSTRUCTORS

```
C
C     FILENAME:  GRSTART.FR
C
      SUBROUTINE  START
      COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
      INTEGER  OUT, TXTIND, TEXT
      .SPEC
      .EXEC
      .BODY
      INDEX=1
      TXTIND(1,1)=1
      TXTIND(2,1)=1
      RETURN
      END
```

```
C
C       FILENAME:  GRCLEAR.FR
C
        SUBROUTINE  CLEAR
        COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
        INTEGER  OUT, TXTIND, TEXT
        .SPEC
        .EXEC
        .BODY
        PDF(1,INDEX)=2
        INDEX=INDEX+1
        RETURN
        END
```

```
C
C     FILENAME:  GRWMODE.FR
C
      SUBROUTINE  WMODE
      COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
      INTEGER  OUT, TXTIND, TEXT
      .SPEC
      .EXEC
      .BODY
      PDF(1,INDEX)=3
      INDEX=INDEX+1
      RETURN
      END
```

```
C
C     FILENAME:  GREMODE.FR
C
      SUBROUTINE  EMODE
      COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
      INTEGER  OUT, TXTIND, TEXT
      .SPEC
      .EXEC
      .BODY
      PDF(1,INDEX)=4
      INDEX=INDEX+1
      RETURN
      END
```

```
C
C     FILENAME:  GRSENDPDF.FR
C
      SUBROUTINE  SENDPDF
      COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
      INTEGER  OUT, TXTIND, TEXT
      .SPEC
      .EXEC
      .BODY
      PDF(1,INDEX)=6
      INDEX=INDEX+1
      RETURN
      END
```

```
C
C     FILENAME:   GRMOVES.FR
C
      SUBROUTINE  MOVES (X,Y,Z)
      COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
      INTEGER  OUT, TXTIND, TEXT
      .SPEC
      .EXEC
      .BODY
      PDF(1,INDEX)=0
      PDF(2,INDEX)=X
      PDF(3,INDEX)=Y
      PDF(4,INDEX)=Z
      INDEX=INDEX+1
      RETURN
      END
```

X : X Coordinate (floating point)
Y : Y Coordinate (floating point)
Z : Z Coordinate (floating point)

```
C
C     FILENAME:  GRVEC.FR
C
      SUBROUTINE  VEC (X,Y,Z)
      COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
      INTEGER  OUT, TXTIND, TEXT
      .SPEC
      .EXEC
      .BODY
      PDF(1,INDEX)=1
      PDF(2,INDEX)=X
      PDF(3,INDEX)=Y
      PDF(4,INDEX)=Z
      INDEX=INDEX+1
      RETURN
      END
```

X : X Coordinate (floating point)
Y : Y Coordinate (floating point)
Z : Z Coordinate (floating point)

```
C
C     FILENAME:  GRDTEXT.FR
C
      SUBROUTINE  DTEXT (N,ISTR,IND)
      COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
      INTEGER  OUT, TXTIND, TEXT
      DIMENSION  ISTR(30)
      EQUIVALENCE  (IP1,TXTIND),
     *             (IP2,TXTIND(2,1))
      .SPEC
      .EXEC
      .BODY
      IP1=IP1+1
      IND=IP1
      TXTIND(1,IP1)=IP2
      TXTIND(2,IP1)=N
      NN=(N-1)/2+1
      DO 1 I=1,NN
      TEXT(IP2)=ISTR(I)
    1 IP2=IP2+1
      RETURN
      END
```

N : Number of characters in text string.
ISTR : Text string (integer array).
IND : Location of table entry for text.

```
C
C       FILENAME:  GRHTEXT.FR
C
        SUBROUTINE  HTEXT (N,ISTR)
        COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
        INTEGER  OUT, TXTIND, TEXT
        DIMENSION  ISTR(30)
        .SPEC
        .EXEC
        .BODY
        CALL  DTEXT (N,ISTR,IND)
        XN=IND
        PDF(1,INDEX)=100.+XN
        INDEX=INDEX+1
        RETURN
        END
```

N : Number of characters in text string.
ISTR : Text string (integer array).

```
C
C      FILENAME:  GRVTEXT.FR
C
       SUBROUTINE  VTEXT (N,ISTR)
       COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
       INTEGER  OUT, TXTIND, TEXT
       DIMENSION  ISTR(30)
       .SPEC
       .EXEC
       .BODY
       CALL  DTEXT (N,ISTR,IND)
       PDF(1,INDEX)=-100-IND
       INDEX=INDEX+1
       RETURN
       END
```

N : Number of characters in text string.
ISTR : Text string (integer array).

APPENDIX "C"

SOURCE CODE

FOR

COMPILERS

```
C
C     FILENAME:  GRCOMPIL.FR
C
C     *********************************************
C     GRAPHICS COMPILER WITH TEXT HANDLING CAPABILITY
C     *********************************************
C
      SUBROUTINE  COMPIL  (I1,I2,L)
      COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
      INTEGER  OUT, TXTIND, TEXT, GETC, OPB, OPW, OPC,
     *         PARM1, PARM2, PARM3, PARM4, D, X, Y, B(6)
      .SPEC
      .EXEC
    1 FORMAT  ("0ERROR - UNTERMINATED PDF.")
    2 FORMAT  ("0ERROR - ILLEGAL TEXT INDEX.")
    3 FORMAT  ("0ERROR - ILLEGAL PDF OPERATION CODE.")
    4 FORMAT  ("0ERROR - UNIMPLEMENTED PDF OPERATION CODE.")
    5 FORMAT  ("0ERROR - OUTPUT TEXT BUFFER OVERFLOW.")
      .BODY
      PARM1=200
      PARM2=50
      PARM3=200
      PARM4=500
      MODE=0
      IP=I1-1
      OPW=I2
      OPB=0
```

I1 : The index into the PDF of the first entry to be compiled.

I2: The pointer into the "OUT" buffer where the compiled instructions are to start.

L : Amount of the buffer used.

```
   10 IP=IP+1
      IF  (IP.LE.PARM1)  GOTO 20
C
C     ERROR - UNIMPLEMENTED PDF.
C
      WRITE  (10,1)
      STOP 1
   20 OPC=PDF(1,IP)+SIGN(0.1,PDF(1,IP))
      IF  (IABS(OPC).LE.7)  GOTO 30
      IF  (IABS(OPC).LE.100)  GOTO 130
      ITXT=IABS(OPC)-100
      IF  (ITXT.LE.PARM2)  GOTO 40
C
C     ERROR - ILLEGAL TEXT INDEX.
C
      WRITE (10,2)
      STOP 2
   30 IF  (OPC.GE.0)  GOTO 60
C
C     ERROR - ILLEGAL PDF OPERATION CODE.
C
      WRITE  (10,3)
      STOP 3
   40 IF  (OPC.LT.0)  GOTO 50
      D=1
      GOTO 220
```

```
 50 D=3
    B(2)=8
    B(3)=10
    GOTO 220
 60 IF  (OPC.EQ.0.OR.OPC.EQ.1)  GOTO 90
    IF  (OPC.NE.2)  GOTO 100
    C=12
 70 N=0
    IF  (MODE.EQ.0)  GOTO 80
    B(1)=64
    N=1
    MODE=0
 80 N=N+1
    B(N)=C
    GOTO 180
 90 N=0
    IF  (MODE.NE.0)  GOTO 170
    B(1)=28
    N=1
    MODE=1
    GOTO 170
100 IF  (OPC.NE.3)  GOTO 110
    C=15
    GOTO 70
110 IF  (OPC.NE.4)  GOTO 120
    C=14
    GOTO 70
```

```
  120 IF  (OPC.LT.7)  GOTO 140
C
C     ERROR - UNIMPLEMENTED PDF OPERATION CODE.
C
  130 WRITE  (10,4)
      STOP 4
  140 IF  (MODE.NE.0)  GOTO 150
      N=0
      GOTO 260
  150 OPB=OPB+1
      IF  (OPB.LE.2)  GOTO 160
C         ========
      OPB=1
      OPW=OPW+1
      IF  (OPW.GE.PARM4)  GOTO 210
  160 CALL  PUTC (OPW,OPB,64)
      MODE=0
      GOTO 260
  170 Y=MOD(IFIX(PDF(3,IP)+0.5),256)
      X=MOD(IFIX(PDF(2,IP)+0.5),256)
      N=N+1
      B(N)=OPC+2+16*MOD(Y,4)
      IF  (B(N).LT.32)  B(N)=B(N)+64
      N=N+1
      B(N)=Y/4
      IF  (B(N).LT.32)  B(N)=B(N)+64
      N=N+1
```

```
      B(N)=16*MOD(X,4)
      IF  (B(N).LT.32)  B(N)=B(N)+64
      N=N+1
      B(N)=X/4
      IF  (B(N).LT.32)  B(N)=B(N)+64
  180 DO 200 I=1,N
      OPB=OPB+1
      IF  (OPB.LE.2)  GOTO 190
C          ========
      OPB=1
      OPW=OPW+1
      IF  (OPW.GT.PARM4)  GOTO 210
  190 CALL  PUTC (OPW,OPB,B(I))
  200 CONTINUE
      GOTO 10
C
C     ERROR - OUTPUT TEXT BUFFER OVERFLOW.
C
  210 WRITE  (10,5)
      STOP 5
  220 NC=TXTIND(2,ITXT)
      ITXTB=TXTIND(1,ITXT)
      I=0
      IF  (MODE.EQ.0)  GOTO 240
      OPB=OPB+1
      IF  (OPB.LE.2)  GOTO 230
C          ========
```

```
      OPW=OPW+1
      OPB=1
      IF  (OPW.GT.PARM4)  GOTO 210
  230 CALL  PUTC (OPW,OPB,64)
      MODE=0
  240 I=I+1
      IF  (I.GT.NC)  GOTO 10
      B(1)=GETC(TEXT(ITXTB),I)
      DO 250 J=1,D
      OPB=OPB+1
      IF  (OPB.LE.2)  GOTO 250
C         ========
      OPW=OPW+1
      OPB=1
      IF  (OPW.GT.PARM4)  GOTO 210
  250 CALL  PUTC (OPW,OPB,B(J))
      GOTO 240
  260 OPB=OPB+1
      IF  (OPB.GT.2)  GOTO 270
C         ========
      CALL  PUTC (OUT,OPW,OPB,0)
      GOTO 260
  270 L=OPW-I2+1
      IF  (OPC.EQ.5)  RETURN
      CALL  BSEND (I2,L)
      RETURN
      END
```

```
C
C     FILENAME:  GRGETC.FR
C
      INTEGER  FUNCTION  GETC (STRING,NUMBER)
      COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
      COMMON/TRSF/  IDM, T(4,4)
      INTEGER  OUT, TXTIND, TEXT
      INTEGER  NUMBER, WORD, BYTE, STRING(30)
      .SPEC
      .EXEC
      .BODY
      WORD=(NUMBER+1)/2
      BYTE=MOD(NUMBER-1,2)+1
      GOTO (1,2),BYTE
    1 GETC=MOD(STRING(WORD)/256,256)
      GOTO 3
    2 GETC=MOD(STRING(WORD),256)
    3 RETURN
      END
```

```
C
C     FILENAME:  GRPUTC.FR
C
      SUBROUTINE  PUTC (W,C,CH)
      COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
      COMMON/TRSF/  IDM, T(4,4)
      INTEGER  OUT, TXTIND, TEXT, W, C, CH
      .SPEC
      .EXEC
      .BODY
      GOTO (1,2),C
    1 OUT(W)=256*CH
      GOTO 3
    2 OUT(W)=OUT(W)+CH
    3 RETURN
      END
```

```
C
C     FILENAME:  GRBSEND.FR
C
      SUBROUTINE  BSEND (LOC,LEN)
      COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
      INTEGER  OUT, TXTIND, TEXT
      .SPEC
      .EXEC
      .BODY
      L = LEN + LOC - 1
      DO  3  I=LOC,L
      DO  3  J=1,2
      IF  (J.EQ.2)  GOTO 1
      IOUT = OUT(I)/256
      GOTO 2
    1 IOUT = OUT(I)
    2 ITTY = IOUT
    3 CONTINUE
      RETURN
      END
```

The Fortran compiler was used on this source file with the "output assembler source file" switch and the resulting ".SR" file was altered as shown in the following source listing.

```
; C
; C       FILENAME:  GRBSEND.SR
; C
;         COMMON/GRAF/ INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
;         INTEGER  OUT, TXTIND, TEXT
;         .SPEC
;         .EXEC
          .COMM    GRAF     4541
          .NREL
          .TITL    BSEND
          .ENT     BSEND
          .NREL
          .TXTM    1
          .EXTU
          .EXTN    .I
.F1:
.F2:      0
          .CSIZ    0
A11.:                       ;TEXT
          .+3
          .GADD    GRAF,4231
          310
          3
          401
          1
          310
A10.:                       ;TXTIND
          .+3
          .GADD    GRAF,4065
          144
          5
          401
          1
          2
          1
          144
A7.:                        ;OUT
          .+3
          .GADD    GRAF,3101
          764
          3
          401
          1
          764
A6.:                        ;PDF
          .+3
          .GADD    GRAF,1
          3100
          5
          1002
          1
          4
          1
          1440
```

```
        FS.
BSEND:
        JSR     @.CPYL
        JMP     @.+1
        L1.
;       .BODY
:       L = LEN + LOC - 1
L1.:
        LDA     0,@T.+1,3       ;LEN
        LDA     1,@T.+0,3       ;LOC
        ADD     1,0
        SUBZL   2,2
        SUB     2,0
        STA     0,T.+2,3        ;L
;       DO  3  I=LOC,L
        LDA     0,@T.+0,3       ;LOC
        STA     0,T.+3,3        ;I
        JMP     L5.
        L4.
L3.:    LDA     0,T.+3,3        ;I
        INC     0,0
        STA     0,T.+2,3        ;L
L5.:    LDA     1,T.+2,3        ;L
        SUBZR   2,2
        AND     1,2
        ADDL    0,2
        SUB     0,1,SNC
        JMP     @L3.-1
;       DO  3  J=1,2
        SUBZL   0,0
        STA     0,T.+4,3        ;J
        JMP     L10.
        L7.
L6.:    LDA     0,T.+4,3        ;J
        INC     0,0
        STA     0,T.+4,3        ;J
L10.:   JSR@    .LD1
        .C5
        SUBZ    0,1,SNC
        JMP     @L6.-1
;       IF  (J.EQ.2)  GOTO 1
        JSR@    .LD1
        .C5
        SUB     0,1,SZR
        SUB     1,1,SKP
        ADC     1,1
        MOV     1,1,SZR
        JMP     .+3
        JMP     @.+1
        L11.
        JMP     @.+1
        L12.
```

```
L11.:
;         IOUT = OUT(I)/256
          JSR       @.FSUB
          3
          A7.                           ;OUT
          VS.+1
          V.+3                          ;I
          JSR@      .LDO
          .C7
          LDA       1,@TS.+1,3
          JSR       @.sdvd
          STA       1,T.+5,3            ;IOUT
;         GOTO 2
          JMP       @.+1
          L13.
;       1 IOUT = OUT(I)
L12.:     JSR       @.FSUB
          3
          A7.                           ;OUT
          VS.+1
          V.+3                          ;I
          LDA       0,@TS.+1,3
          STA       0,T.+5,3            ;IOUT
;       2 ITTY = IOUT
L13.:     LDA       0,T.+5,3            ;IOUT
;------------------------------------
;  ORIGINAL CODE WAS:
;
;         STA       0,T.+6,3            ;ITTY
;
;------------------------------------
;  NEW CODE INSERTED IS:
;
          .SYSTM              ;TTYIO ROUTINE
          .PCHAR              ;SEND CHAR IN ACO
          JMP       .+1       ;ERROR RETURN
;------------------------------------
;       3 CONTINUE
L2.:      JMP       @.+1
          L6.
L7.:
          JMP       @.+1
          L3.
L4.:
;         RETURN
          JSR       @.FRET
;         END
          JSR       @.FRET
.C7:      000400
.C6:      000062
.C5:      000002
.C4:      000764
.C3:      000310
.C2:      000001
```

```
.C1:    000004
        FS.=11
        SFS.=0
        T.=-167
        V.=200+T.
        TS.=T.+6
        FTS.=T.+0
        VS.=V.+6
        FVS.=V.+0
        .END
```

APPENDIX "D"

SOURCE CODE

FOR

TRANSFORMERS

```
C
C     FILENAME:  GRINIT.FR
C
      SUBROUTINE  INIT (IDM1)
      COMMON/TRSF/  IDM, T(4,4)
      .SPEC
      .EXEC
      .BODY
      IDM=IDM1
      ID1=IDM+1
      DO 1 J=1,4
      DO 1 I=1,4
    1 T(I,J)=0.
      DO 2 I=1,ID1
    2 T(I,I)=1.
      RETURN
      END
```

IDM1 : Number of dimensions (2 or 3 only).

```
C
C     FILENAME:  GRDAPPLY.FR
C
      SUBROUTINE  DAPPLY(I1,I2)
      COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
      COMMON/TRSF/  IDM, T(4,4)
      INTEGER  OUT, TXTIND, TEXT
      DIMENSION  X(4)
      .SPEC
      .EXEC
      .BODY
      ID1=IDM+1
      DO 3 I=I1,I2
      DO 2 L=2,ID1
      X(L)=0.
      LL=L-1
      DO 1 J=2,ID1
      K=J-1
    1 X(L)=X(L)+PDF(J,I)*T(K,LL)
    2 X(L)=X(L)+T(ID1,LL)
      DO 3 J=2,ID1
    3 PDF(J,I)=X(J)
      RETURN
      END
```

I1 : Location of first entry in PDF to which the transformation is to be applied.

I2 : Location of last entry in PDF to which the transformation is to be applied.

```
C
C     FILENAME:  GRSCALE.FR
C
      SUBROUTINE  SCALE (X,Y,Z)
      COMMON/TRSF/  IDM, T(4,4)
      DIMENSION  F(4,4)
      .SPEC
      .EXEC
      .BODY
      DO 1 J=1,4
      DO 1 I=1,4
    1 F(I,J)=0.
      XTEM=X
      YTEM=Y
      IF  (IDM.eq.2)  GOTO 2
      ZTEM=Z
      F(1,1)=XTEM
      F(2,2)=YTEM
      F(3,3)=ZTEM
      F(4,4)=1.
      GOTO 3
    2 F(1,1)=XTEM
      F(2,2)=YTEM
      F(3,3)=1.
    3 CALL  MATMUL (F)
      RETURN
      END
```

X : Scale factor with respect to the X axis.
Y : Scale factor with respect to the Y axis.
Z : Scale factor with respect to the Z axis.

```
C
C     FILENAME:  GRTRANS.FR
C
      SUBROUTINE  TRANS (X,Y,Z)
      COMMON/TRSF/  IDM, T(4,4)
      DIMENSION  F(4,4)
      .SPEC
      .EXEC
      .BODY
      DO 2 I=1,4
      DO 1 J=1,4
    1 F(I,J)=0.
    2 F(I,I)=1.
      XTEM=-X
      YTEM=-Y
      IF  (IDM.EQ.2)  GOTO 3
      ZTEM=-Z
      F(4,1)=XTEM
      F(4,2)=YTEM
      F(4,3)=ZTEM
      GOTO 4
    3 F(4,4)=0.
      F(3,1)=XTEM
      F(3,2)=YTEM
    4 CALL  MATMUL (F)
      RETURN
      END
```

X : Distance in negative X direction that the object is to be moved.

Y : Distance in negative Y direction that the object is to be moved.

Z : Distance in negative Z direction that the object is to be moved.

```
C
C      FILENAME:   GRROTATE.FR
C
       SUBROUTINE  ROTATE (L,THET)
       COMMON/TRSF/  IDM, T(4,4)
       DIMENSION  F(4,4)
       .SPEC
       .EXEC
       .BODY
       DO 1 J=1,4
       DO 1 I=1,4
     1 F(I,J)=0.
       F(4,4)=1.
       TEM=3.14159265/180.
       TEMP=THET*TEM
       XCOS=COS(TEMP)
       XSIN=SIN(TEMP)
       XNSIN=-XSIN
       IF  (IDM.EQ.2)  GOTO 4
       GOTO (2,3,5),L
     2 F(1,1)=1.
       F(2,2)=XCOS
       F(3,2)=XSIN
       F(2,3)=XNSIN
       F(3,3)=XCOS
```

L : Axis of rotation (1:X 2:Y 3:Z).

THET : Degrees of rotation in clockwise (right hand rule) direction about specified axis.

```
      CALL  MATMUL (F)
      RETURN
    3 F(1,1)=XCOS
      F(3,1)=XNSIN
      F(2,2)=1.
      F(1,3)=XSIN
      F(3,3)=XCOS
      CALL  MATMUL (F)
      RETURN
    4 F(4,4)=0.
    5 F(1,1)=XCOS
      F(2,1)=XSIN
      F(1,2)=XNSIN
      F(2,2)=XCOS
      F(3,3)=1.
      CALL  MATMUL (F)
      RETURN
      END
```

```
C
C     FILENAME:  GRMATMUL.FR
C
      SUBROUTINE  MATMUL (F)
      COMMON/TRSF/  IDM, T(4,4)
      DIMENSION  F(4,4), X(4,4)
      .SPEC
      .EXEC
      .BODY
      N=IDM+1
      DO 1 I=1,N
      DO 1 J=1,N
      X(I,J)=0.
      DO 1 K=1,N
    1 X(I,J)=T(I,K)*F(K,J)+X(I,J)
      DO 2 J=1,N
      DO 2 I=1,N
    2 T(I,J)=X(I,J)
      RETURN
      END
```

F : The new transformation matrix (floating point).

APPENDIX "E"

SOURCE CODE

FOR

EXAMPLE PROGRAMS

AND

TEST PROGRAMS

EXAMPLE GRAPHICS PROGRAM (NOT USING GRAPHICS ROUTINES)

```
        .TITL   SHOW
        .NREL
;
;  OUTPUT THE PICTURE STORED IN "BUF" TO THE COMPUTEK 300.
;
SHOW:   IORST
        LDA     2,ABUF
        LDA     3,NUMB
        STA     3,TIMES
        LDA     0,000,3
        JSR     OUT
        INC     2,2
        DSZ     TIMES
        JMP     .-4
        JMP     SHOW
NUMB:   18.
TIMES:  0
;
;  OUTPUT DRIVER WITH DELAY ACCORDING TO CONSOLE SWITCHES.
;
DELAY:  0
OUT:    DOAS    0,TTO
        SKPDN   TTO
        JMP     .-1
        READS   0
        STA     0,DELAY
        ISZ     DELAY
        JMP     .-1
        JMP     000,3
;
;  BUFFER CONTAINING PICTURE TO BE OUTPUT.
;
ABUF:   .+1
BUF:    34      ;ENTER FOUR BYTE MODE.
        103     ;Y1, FIRST BYTE.
        104     ;Y2, SECOND BYTE.
        100     ;X1, THIRD BYTE.
        40      ;X2, FOURTH BYTE.
        103     ;Y1, FIRST BYTE.
        40      ;Y2, SECOND BYTE.
        60      ;X1, THIRD BYTE.
        77      ;X2, FOURTH BYTE.
        103     ;Y1, FIRST BYTE.
        40      ;Y2, SECOND BYTE.
        100     ;X1, THIRD BYTE.
        40      ;X2, FOURTH BYTE.
        63      ;Y1, FIRST BYTE.
        77      ;Y2, SECOND BYTE.
        100     ;X1, THIRD BYTE.
        100     ;X2, FOURTH BYTE.
        100     ;ENTER ALPHA MODE.
;
        .END    SHOW
```

By simply modifying the buffer's contents other pictures can be driven to the Computek by the preceding example program.

Once such modification that was used is as follows:

```
BUF:    34      ;ENTER FOUR-BYTE MODE.
        103     ;FIRST BYTE "X".
        104     ;SECOND BYTE "X".
        100     ;THIRD BYTE "Y".
        40      ;FOURTH BYTE "Y".
        100     ;ENTER ALPHA MODE.
        101     ;ASCII "A".
        102     ;ASCII "B".
        103     ;ASCII "C".
        40      ;BLANK.
        40      ;BLANK.
        40      ;BLANK.
        34      ;ENTER FOUR-BYTE MODE.
        63      ;FIRST BYTE "X".
        77      ;SECOND BYTE "X".
        100     ;THIRD BYTE "Y".
        100     ;FOURTH BYTE "Y".
        100     ;ENTER ALPHA MODE.
```

TEST GRAPHICS PROGRAM (USING GRAPHICS ROUTINES) WITHOUT OVERLAYS

```
      COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
      COMMON/TRSF/  IDM, T(4,4)
      INTEGER  OUT, TXTIND, TEXT
      .SPEC
      .EXEC
      .BODY
      CALL  START
      CALL  CLEAR
      CALL  MOVES (20.,20.,0.)
      CALL  VEC (200.,200.,0.)
      CALL  MOVES (200.,20.,0.)
      CALL  VEC (20.,200.,0.)
      CALL  MOVES (20.,100.,0.)
      CALL  VEC (200.,100.,0.)
      CALL  SENDPDF
      CALL  COMPIL (1,1,L)
      CALL  INIT (2)
      CALL  TRANS (10.,0.,0.)
      ID = INDEX - 1
      DO  2  I=1,5
      CALL  DAPPLY (1,ID)
      CALL  COMPIL (1,1,L)
    2 CONTINUE
      END
```

TEST GRAPHICS PROGRAM (USING GRAPHICS ROUTINES) WITH OVERLAYS

```
COMMON/GRAF/  INDEX, PDF(4,200), OUT(500), TXTIND(2,50), TEXT(200)
COMMON/TRSF/  IDM, T(4,4)
INTEGER  OUT, TXTIND, TEXT
.SPEC
.EXEC
.BODY
CALL  OVER ("CONSTRUCT")
CALL  START
CALL  CLEAR
CALL  MOVES (200.,100.,0.)
CALL  VTEXT (10,"DR.HANKLEY")
CALL  MOVES (230.,20.,0.)
CALL  VEC (250.,35.,0.)
CALL  VEC (230.,35.,0.)
CALL  VEC (250.,20.,0.)
CALL  VEC (240.,45.,0.)
CALL  VEC (230.,20.,0.)
CALL  MOVES (20.,200.,0.)
CALL  HTEXT (12,"V.R.WALRAFEN")
CALL  SENDPDF
CALL  COMPIL (1,1,L)
CALL  OVER ("TRANSFORM")
CALL  INIT (2)
```

```
      CALL  TRANS (10.,0.,0.)
      ID = INDEX - 1
      DO  5  I=1,5
      CALL  OVER ("TRANSFORM")
      CALL  DAPPLY (1,ID)
      CALL  OVER ("CONSTRUCT")
      CALL  COMPIL (1,1,L)
    5 CONTINUE
      END
```

APPENDIX "F"

SYSTEM HARDWARE

HARDWARE CONFIGURATION

The NOVA minicomputer system hardware configuration consists of a Data General NOVA minicomputer, a Caelus model 303 disk cartridge drive, a Computek 300 graphics terminal, an IBM Selectric typewriter, a highspeed paper tape punch and reader and a KSR-33 Teletype.

Minicomputer

The NOVA[1] minicomputer in the KSU Computer Science Department has 16,384 bytes (8,192 words) of main memory and two communication rates of 110 baud and 1200 baud.

All main memory locations are commonly expressed using octal numbers; thus the maximum addressable storage location in main memory is $37,777_8$ which is $16,383_{10}$.

Disk Drive

The Caelus[2] disk drive conected to the Department's NOVA has one removable IBM5440 type disk cartridge and one fixed disk. Each disk has 205 tracks per surface with a recording density of 2200 bytes per inch yielding a storage capacity of 7500 bytes (3750 words) per track and 24.6 megabits (1,537,500 words) per disk.

Graphics Terminal

The Computek[3] graphics terminal has a black and white display screen consisting of a matrix of 255 by 255 addressable points, which are either on or off, with the screen matrix origin being at the lower left hand corner of the screen.

There is no intensity variation and no color control (although intensity could be simulated by the dot density variation and color photographs could be generated using multiple exposures with color filters).

Internal hardware provides straight line vector capability only but curved lines can be approximated by a series of short straight line vectors.

The screen cursor (moveable crosshairs) can be positioned at any addressable point in the screen matrix using the "slew" keys on the keyboard, or using a pen and tablet which is electronically connected with the screen cursor, or by the receipt of a command string via the input channel from the minicomputer.

Similarly, the cursor position can be read from the terminal by the minicomputer under program control.

The result of moving the screen cursor depends upon the "mode" the terminal is in when the movement is initiated.

In the "move" mode simple repositioning of the screen cursor occurs, while in the "write" mode a straight line vector is approximated by illumination of addressable points between the original screen cursor position and the new position specified and in the "erase" mode erasure of all addressable points from the original screen cursor position to the new position specified occurs.

Character generation is by internal hardware using a seven (7) dot vertical by five (5) dot horizontal matrix where the character position is defined by the screen matrix address of the lower left hand dot in the character matrix.

Characters are erased by overwriting them on the display screen with a blank character.

1 - <u>How to use the Nova Computers</u>,
DG NM-5,
April 1971,
Data General Corporation,
Southboro, Massachusetts 01772

2 - <u>CAELUS CONTROLLER - Disk Drive / Nova Computer - Operation and Maintenance Manual</u>,
Revision A,
2/72,
Caelus Memories, Inc.,
967 Mabury Road,
San Jose, California

3 - <u>300 SERIES User's Manual</u>,
009-00021,
Sept. 1972,
Computek, Inc.,
143 Albany Street,
Cambridge, Massachusetts 02139

APPENDIX "G"

MISCELLANEA

Explanation of Components of Final Step Command String

The final step command string that creates an executable core image "save" file for the user's problem program was given as:

```
RLDR 1000/N name.SV/S USERLINK.RB OVERLAY.RB name.RB SYS.LB 23240/N
```

The explanation of each component and its function is as follows:

RLDR - The filename for NOVA's Relocatable Loader. This causes the relocatable loader to be placed in main memory and execution to commence using the remaining components of the command string as arguements.

1000/N - Indicates that the loading of the program is to start at location 1000_8. In our case this is where our first labeled common will start.

name.SV/S - Causes the final core image "save" file for the user's problem program to have the name that was chosen by the user for his program. Without this the relocatable loader would pick up the name of the first file, in our case "USERLINK", and name the final core image file by that name.

USERLINK.RB - Produces no executable code but causes all fortran library subroutines to be loaded in the same spots as was done in producing the overlay "segment" files. It also gives "page zero" definitions for all entry points into the graphics routines that will be called in later in the overlay area when the requested overlay "segment" file is loaded.

OVERLAY.RB - Brings the object code for the overlay routine in and places it in the final core image "save" file.

name.RB - Brings the object code for the user's problem program in and places it in the final core image "save" file.

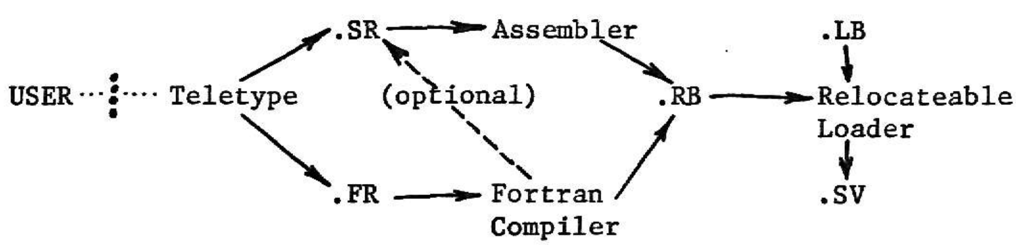
SYS.LB - Provides the object code for all the fortran library subroutines required by the graphics routines (as indicated by USERLINK), by OVERLAY.RB and by name.RB.

23240/N - Forces the value of "NMAX" in the user status table to have the same value as the maximum overlay segment so that the fortran runtime stack will start above the maximum address needed by the graphics routines. Otherwise the runtime stack would start just above the last library subroutine just loaded.

Definitions of File Suffix by File Type.

File Suffix	File Type
.SR	NOVA assembler language source file.
.FR	NOVA Fortran IV language source file.
.RB	Relocateable binary file.
.SV	Core Image file ("Save" file).
.LB	Subroutine library file.
none	Overlay segment file or text file.

File Type Relationships.



User Status Table Template

Address	Contents
400	Program Counter.
401	ZMAX.
402	Start of Symbol Table.
403	End of Symbol Table.
404	NMAX.
405	Starting Address.
406	Debugger Address.
407	Highest Address Used by Load Module.
423	Save Storage for AC0.
424	Save Storage for AC1.
425	Save Storage for AC2.
426	Save Storage for AC3.
427	Save Storage for Carry Bit.

Wire List for CRT Plug on Synetics NOVA - Slot 9.

PLUG PIN	MOTHER BOARD PIN	WIRE COLOR	NOTE
1	B97	BROWN	+5V
2	A89	RED	
3	B69	ORANGE	
4	A6	YELLOW	-5V
5	B100	GREEN	(GROUND)
6	A85	BLUE	
7	A83	VIOLET	
8	A87	GRAY	
9	B99	WHITE	GROUND

Computek 300 / Synetics NOVA Connect Procedure.

1.) Power off NOVA and teletype.
2.) Pull TTY board out 2" in slot 3.
3.) Push CRT board in solid in slot 9.
4.) Connect Computek at the teletype plug on the upper right front corner of the NOVA.
5.) Attach the two jumpers:
    Slot 3...A95-A96
    Slot 3...A93-A94
6.) Power on NOVA and Computek.
7.) Computek "ON", "LINE", "READY" and 1200 baud ("4" on rear).

Computek 300 / Synetics NOVA Disconnect Procedure.

1.) Power off NOVA and Computek.
2.) Pull CRT board out 2" in slot 9.
3.) Push TTY board in solid in slot 3.
4.) Disconnect Computek at the teletype plug on the upper right front corner of the NOVA.
5.) Remove the two jumpers:
    Slot 3...A95-A96
    Slot 3...A93-A94
6.) Power on NOVA and teletype.

A MINICOMPUTER GRAPHICS SYSTEM

by

VERNE ROY WALRAFEN

B.S. in Civil Engineering, University of Kansas, 1963

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1976

The implementation of the transportation of a basic subset of an interactive vector graphics software package from a large machine environment to that of a minicomputer is presented. The subset selected contains only the most primitive operations necessary to allow a user to construct non-interactive problem programs.

The necessary instructions to allow a user to build a problem program and thus actually use the graphics package are presented. In addition the reader is taken step by step through the necessary activities that would allow him to modify and/or build extensions to the subset selected initially.

It is proposed that the minicomputer environment, while remaining a useful tool in the execution of fully developed software packages, is excessively difficult to use in the actual developmental stages though obviously not impossible. The software and hardware support of a large machine environment provide the researcher a much firmer basis from which to pursue the actual problems postulated by his research.