IGWS

Integrated graphics workstation

by

Ketil Bø

SIU
P.B. 660
7001 Trondheim-NTH

August 1983
I. ACKNOWLEDGEMENT

I would like to thank the members of the Technical/mathematical group at RUNIT and the diploma students of mine which have implemented a considerable part of the ideas presented in this thesis and given me valuable feed-back.

Since the graphics package IDIGS is proposed to become an international graphics standard, the work has of course been influenced by GKS (Graphics Kernel System), GSPC (The "CORE") and a number of interesting discussions with colleagues in ISO TC 97/SC5/WG2 - Graphics.

The work on IDIGS has been supported by the Royal Norwegian Council of Research.

Let me also thank my supervisor professor Kjell Bratbergsengen, IDB.
II. PREFACE

II 1. The purpose of the thesis

The purpose of the thesis is to design an integrated and powerful workstation for technological applications. Emphasis has been put on the graphics subsystem, but also other subsystems utilizing human abilities for communication is included.

Only for the graphics subsystem is the detailed research work reported. For the other subsystems I have restricted myself to describe the properties and give references to work done by myself and members of my group at RUNIT.

II 2. The content of the thesis

The thesis consists of three papers built up in hierarchical order:

1. "Integrated Graphics Workstation (IGWS)" which purpose is

   i) Design of an Integrated Graphics Workstation for technical applications which offers an efficient tool for man/computer interaction.

   ii) Present the methodology, deliberations and research which lead to the proposals presented in the following two papers which address some of the subsystems of IGWS in more detail.

2. IDIGS (Interactive Device Independent Graphics System) Submitted to ISO as a proposal for an international 3D-graphics standard.
   IDIGS is the functional design of a basic 3D-graphics system.

3. An extension of "Compound logical input devices" which was published by NORTH HOLLAND 1981.
   This paper is presenting the scientific work in connection with the input part of the IDIGS proposal.
Fig. 1. The hierarchy of the work.
III. ABSTRACT

This paper describes some of the problems related to man/computer interaction in technical applications and a proposal how they can be solved by designing an Integrated Graphics Workstation (IGWS) (the details of the Graphics subsystem in IGWS are discussed in the other papers of this thesis).

The Integrated Graphics Workstation offers fast data capturing by use of softwaresupported digitizing and image processing, efficient man/computer conversation by utilizing the operators ability to communicate and a powerful drawing production subsystem.

Some of the research work in connection with the preparation of the more detailed papers of the thesis are also included.
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.  ACKNOWLEDGEMENT</td>
<td>II</td>
</tr>
<tr>
<td>II.  PREFACE</td>
<td>III</td>
</tr>
<tr>
<td>III.  ABSTRACT</td>
<td>V</td>
</tr>
<tr>
<td>CONTENTS</td>
<td>IV</td>
</tr>
<tr>
<td>1.  INTRODUCTION</td>
<td></td>
</tr>
<tr>
<td>1.1. Classification of the problems</td>
<td>1</td>
</tr>
<tr>
<td>2.  A PATTERN OF SYSTEM GROWING</td>
<td>4</td>
</tr>
<tr>
<td>2.1. Step one</td>
<td>5</td>
</tr>
<tr>
<td>2.2. Step two</td>
<td>5</td>
</tr>
<tr>
<td>2.3. Step three</td>
<td>6</td>
</tr>
<tr>
<td>2.4. Step four</td>
<td>7</td>
</tr>
<tr>
<td>2.5. Step five</td>
<td>9</td>
</tr>
<tr>
<td>2.6. Discussion of the &quot;growing strategy&quot;</td>
<td>10</td>
</tr>
<tr>
<td>3.  THE GRAPHICS HARDWARE</td>
<td>11</td>
</tr>
<tr>
<td>3.1. Classification of hardware</td>
<td>11</td>
</tr>
<tr>
<td>3.2. Distributed Intelligence System</td>
<td>14</td>
</tr>
<tr>
<td>3.2.1. The satellite protocol</td>
<td>17</td>
</tr>
<tr>
<td>4.  THE DATA RECORDING SUBSYSTEM</td>
<td>19</td>
</tr>
<tr>
<td>4.1. Initialization</td>
<td>20</td>
</tr>
<tr>
<td>4.1.1. Menues</td>
<td>20</td>
</tr>
<tr>
<td>4.1.2. Surface administration</td>
<td>20</td>
</tr>
<tr>
<td>4.1.3. Automatic corrections</td>
<td>21</td>
</tr>
<tr>
<td>4.1.4. True Scale</td>
<td>21</td>
</tr>
<tr>
<td>4.2. Digitizing</td>
<td>21</td>
</tr>
<tr>
<td>4.2.1. Commands</td>
<td>22</td>
</tr>
<tr>
<td>4.2.2. Point digitizing</td>
<td>22</td>
</tr>
<tr>
<td>4.2.3. 2 1/2D digitizing</td>
<td>22</td>
</tr>
<tr>
<td>4.2.4. Orthogonal projections</td>
<td>22</td>
</tr>
<tr>
<td>4.2.5. 3D reconstruction</td>
<td>23</td>
</tr>
<tr>
<td>4.2.6. Additional information</td>
<td>23</td>
</tr>
<tr>
<td>4.2.7. Digitizing of standard symbols</td>
<td>24</td>
</tr>
<tr>
<td>4.2.8. Interface to IDIGS</td>
<td>24</td>
</tr>
<tr>
<td>4.3. Editing</td>
<td>24</td>
</tr>
<tr>
<td>4.3.1. Point editing</td>
<td>24</td>
</tr>
<tr>
<td>4.3.2. Symbol editing</td>
<td>24</td>
</tr>
<tr>
<td>4.3.3. Information editing</td>
<td>24</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>5.</td>
<td>THE IMAGE PROCESSING SUBSYSTEM</td>
</tr>
<tr>
<td>5.1.</td>
<td>Data input from image processing</td>
</tr>
<tr>
<td>5.1.1.</td>
<td>3D reconstruction of scanned images</td>
</tr>
<tr>
<td>5.2.</td>
<td>Combined applications</td>
</tr>
<tr>
<td>5.3.</td>
<td>Capturing of data from the creative process</td>
</tr>
<tr>
<td>6.</td>
<td>BASIC GRAPHICS SOFTWARE</td>
</tr>
<tr>
<td>6.1.</td>
<td>Standardization of Graphics Software</td>
</tr>
<tr>
<td>6.1.1.</td>
<td>The impact of the IFIP WG 5.2</td>
</tr>
<tr>
<td>6.1.2.</td>
<td>Two-dimension versus three-dimension</td>
</tr>
<tr>
<td>6.1.3.</td>
<td>Output primitives</td>
</tr>
<tr>
<td>6.1.4.</td>
<td>Input</td>
</tr>
<tr>
<td>6.1.5.</td>
<td>Text handling</td>
</tr>
<tr>
<td>6.1.6.</td>
<td>Viewing</td>
</tr>
<tr>
<td>6.1.7.</td>
<td>Storage of graphical information</td>
</tr>
<tr>
<td>6.2.</td>
<td>System Design</td>
</tr>
<tr>
<td>6.2.1.</td>
<td>Collection of user requirements</td>
</tr>
<tr>
<td>6.2.2.</td>
<td>System requirement specification</td>
</tr>
<tr>
<td>6.2.3.</td>
<td>Research in order to create new methods and techniques</td>
</tr>
<tr>
<td>6.2.4.</td>
<td>Functional specification and testing</td>
</tr>
<tr>
<td>6.3.</td>
<td>Examples of some issues</td>
</tr>
<tr>
<td>7.</td>
<td>THE COMMAND LANGUAGE</td>
</tr>
<tr>
<td>7.1.</td>
<td>A user oriented command language</td>
</tr>
<tr>
<td>7.2.</td>
<td>Some important aspects of the command language</td>
</tr>
<tr>
<td>8.</td>
<td>VOICE INPUT/OUTPUT</td>
</tr>
<tr>
<td>8.1.</td>
<td>Voice input</td>
</tr>
<tr>
<td>8.2.</td>
<td>Voice output</td>
</tr>
<tr>
<td>8.3.</td>
<td>Voice recognition</td>
</tr>
<tr>
<td>8.4.</td>
<td>Experiences</td>
</tr>
<tr>
<td>9.</td>
<td>THE TEXT HANDLING SUBSYSTEM</td>
</tr>
<tr>
<td>9.1.</td>
<td>Representation for document production</td>
</tr>
<tr>
<td>10.</td>
<td>THE DRAWING PRODUCTION SUBSYSTEM (DPS)</td>
</tr>
<tr>
<td>10.1.</td>
<td>Functional specifications of DPS</td>
</tr>
<tr>
<td>10.1.1.</td>
<td>Drawing layout</td>
</tr>
<tr>
<td>10.1.2.</td>
<td>Drawing frame</td>
</tr>
<tr>
<td>10.1.3.</td>
<td>Viewport transformation</td>
</tr>
<tr>
<td>10.1.4.</td>
<td>Number of copies</td>
</tr>
<tr>
<td>10.1.5.</td>
<td>Logging capabilities</td>
</tr>
<tr>
<td>10.2.</td>
<td>Picture editor</td>
</tr>
<tr>
<td>10.3.</td>
<td>Operators command language</td>
</tr>
</tbody>
</table>
10.4. Technical specification
10.4.1. The application computers
10.4.1.1. Header information
10.4.1.2. Picture code
10.4.2. The graphics computer
10.4.2.1. Automatic drawing
10.4.2.2. Drawing started from the console
10.5. Experiences

11. CONCLUSION

REFERENCES
1. INTRODUCTION

The success of most computer systems is highly dependent on their ability to communicate with the user.

Despite of the internal quality of the programs, it is the users acceptance or rejection of the system which decide its position and life span in the working environment.

Unfortunately, the interaction tends to be one of the more neglected parts of many computer systems for several reasons:

1. System builders have been more concerned about making the system work, than the fact that it is supposed to be used by a human being as a tool for problem solving.

2. Man/Machine interaction is a fairly poor understood aspect by system builders, because it involves so many factors which he can't pin down by an algorithm [1].

3. The human factors involved in communication has not yet been understood sufficiently. Even in the "classical" literature like Newman and Sproull [2], it is hardly mentioned.

4. Necessary techniques & technology for utilizing all the human resources in communication has not been available [3].

In order to improve the working environment, more emphasis must therefore be put on the man/machine interaction part of the system development.

That means study of cognitive psychology, perception, the communication process and the design of powerful and flexible workstations where this knowledge can be implemented. This paper is concentrated on the last problem only.

A workstation is the operators work place and can take on many different forms, from simple storage tubes with a keyboard as input tool up to sophisticated dynamic displays with digitizing equipment and a variety of input devices.

There are, however, several problems with the workstations as we know them today:

1. Except for specialized systems such as Computer Vision [4] and Auto-Trol [5], todays systems consist of a number of freestanding
subsystems such as the digitizing- and the drawing production subsystem
which are frequently offline systems with considerable conversion problems
involved.

2. When the workstation capabilities are to be increased, traditionally
a lot of rewriting is involved and the result is usually unfamiliar
to the operator because the expansion is not a well defined part of
a growing strategy [2].

3. Most of today's workstations are utilizing only a fraction of the
operators natural abilities for communication. By offering alpha-
numeric input/output devices with limited graphics capabilities only,
such as the traditional static storage tube with keyboard and thumb-
wheels [6], the user's communication remedies are very limited.

This work propose solutions to solve some of the problems by designing a
framework of a workstation which allows a controlled evolution of capabili-
ties where new hardware, firmware and software features can be introduced
and integrated in such a way that the operator continuously feels familiar
with his working environment.

1.1. Classification of the problems

To identify future problems, we can learn a lot by reviewing the past
and current research on interaction.

On the other hand, we must not be trapped by traditions, which has been
another major obstacle to the development of truly interactive systems.

Many of the clever emergency solutions for the early systems, due to
lack of technology, are unnecessarily repeated for the present systems
and creates both problems and constraints which now could be avoided
because of advance in knowledge and technology [7].

To identify the problems, it may be convenient to classify man/machine
interaction into three main subgroups:

1. Input of initial data for processing and manipulation. (Data
capturing)

2. Manipulation of the data and results as "real time" interactions
between the user and the computer (Man/computer conversation).
3. Output of data and results (data presentation).

At the present time, data capturing is far behind the conversation and presentation when time is concerned. It may take days to prepare the input of the data to a program which takes only minutes or seconds to solve the actual problem.

In order to integrate and streamline the total man/computer process for technical applications, the Integrated Graphics Workstation (IGWS) is designed and partly implemented.

IGWS is designed with two subsystems for data capturing:

The Data Recording Subsystem, discussed in chap. 4., which is designed for efficient utilization of digitizers and the Image Processing Subsystem, discussed in chap. 5, which is designed for (semi-)automatic data capturing.

As far as conversation is concerned, the major problem is to utilize all the users abilities to communicate. The vision is the fastest human information channel [8], Computer graphics, discussed in chap. 6, is therefore the major tool for man/computer conversation in IGWS.

However, written and spoken languages have become the major vehicle for exact communication. IGWS is designed with a powerful command language, discussed in chap. 8, and text processing capabilities, discussed in chap. 9. The output is far the most developed and understood part of the interaction, seen from a technical point of view, but there may still be a lot to gain in studying human perception. Because it is not obvious that a detailed message (in some form) which demands broad bandwidth or a long time frame, communicates the ideas better than one with low data content presented in the right way. (A more detailed discussion of this problem is found in [9].)

However, even if we feel that parts of the output are fairly well understood, the integration of systems for efficient and economical production of hardcopies such as drawings, films etc. are still to be designed (see chap. 9).
2: A PATTERN FOR SYSTEM GROWING

Computer graphics is potentially the most powerful and convenient means of man/computer conversation for a very wide range of applications, but only recently has the necessary hardware become sufficiently cheap, and the software problems sufficiently mastered, for the breakthrough in graphics to become a reality. The eye is man's most efficient data-channel - over 75% of all information received by the brain comes from visual input [11]. Many complex problems become simple when translated into images, and the ability to manipulate images easily and accurately can enormously enhance man's problem-solving abilities. In many applications, computer graphics have for many years promised tantalising rewards in terms of the creative productivity of designers, engineers, and managers. Why has the pay-off so far been so meagre, except in very specialised areas?

High hardware costs, the exceptionally complex software, lack of competent personell and poor flexibility of the graphics workstations have been to blame for the slow penetration of graphics into mainstream data processing.

But now the technology is getting available because of the rapid decrease of hardware cost. The basic graphics software is about to make it possible to start the system development on a relatively high level and the users requirements for graphic tools are getting strong.

Computer graphics uses more diverse equipments than most branches of computer science. According to Frost and Sullivan [12], there are 163 different graphic suppliers in business today. And very few of the graphic products on the marked are in any way standardized.

This variety of equipment may make the utilization of computer graphics work consuming and expensive, unless one from the very beginning prepare for an integration which gives room for growing of the systems both in hardware, software and functionality.

By studying a number of graphics installations and serving as a consultant for companies going into graphics, we have found two dominating patterns:

Jumping into graphics or growing into it. In the first case expensive specialized turn-key systems are provided with the unavoidable problems that such an approach introduce. By careful planning the problems can of course be overcome and we have several successful examples (Selvågbygg etc.). But this approach is outside the scope of this paper (it is however, discussed by the author in [13]). In the second case an inexpensive "trial" configuration is provided and as needs increase, new equipments are usually added more or less accidentally. To make the last approach viable the design of a conceptual workstation framework is now presented into
which the configuration can grow in a controlled manner:

The purpose of this chapter is not to devise an optimal procedure, but to give one approach and analyse what kind of requirements are to be fulfilled.

2.1: Step one

Besides the personal computer, the easiest and least expensive way to get started in computer graphics is to get a simple graphics display, connect it to an existing computer with available graphics software either directly or via a telephone line and start working:

![Diagram of a simple graphics system](image_url)

**Fig. 2.1 Step one: Simple graphics system (Display and modem)**

Such a "test configuration" is cheap enough for a trial and error period after which the whole investment can be written off if one don't find the tool profitable enough.

2.2: Step two

If it turns out profitable, the next step will normally be to provide a computer if no suitable computer is available inhouse.

To make the transition smooth, it may be useful to install a switch for utilization of both the remote computer and the new inhouse computer.
To increase the power of the configuration the mini-computer should be connected to the Host computer as a simple network. By this step specialized programs can still remain in the host without being converted, while the time critical programs are moved to the mini/micro (distributed intelligence system). The question is now: Is the graphic software portable enough to be moved to the mini-computer in order to achieve an efficient resource sharing. (The solution is discussed in chapter 5.)

By portability we mean the ability to transport graphical programs from one installation to another with minimal program changes.

2.3: Step three

As the use of computer graphics increase, new equipments such as plotters, displays etc., may be added to the mini-computer.
Fig. 2.3. Step 3 Configuration: Additional graphics devices.

The question is now: Is the software device independent enough to support all the different devices? (The solution is discussed in chap. 5.)

By device independence we mean the capability of switching from one device to another without rewriting the programs, or to include new devices without a substantial rewriting of the basic graphics software.

2.4. Step four

As new devices and functions are added, the load on the computer will increase and degrade the system such that the throughput and response time is not acceptable any longer. One solution to this problem is to get another computer and dedicate the old one as a special purpose graphics computer for handling the most resource demanding devices.
The question is now: Is the software powerful enough to support graphics network? (The solution is discussed in chap. 10.)

In the simplest case "networking" may be achieved by generating the complete picture on the application computer in device independent compressed code which is sent to the graphics computer for drawing.

In more advanced solutions the connection will also be interactive, and constitute a powerful network. This solution is discussed and implemented by a diploma student of mine [14].
2.5. Step five

If the software and the interfaces are modular and powerful enough, a number of functions such as digitizing (discussed in chap. 4), image processing (discussed in chap. 5) voice input/output (discussed in chap. 8) etc. can be added to the system and several independent systems may be connected to a special purpose graphics computer for producing resource demanding drawings on specialized or expensive hardware (discussed in chap. 10).

Fig. 2.5. Step five configuration: Integrated Graphics Workstation.
Besides that the workstation is integrated and modular, it may also be fully distributed. One may for instance sit in Oslo on a display, generate a picture on a computer in Trondheim and get the result plotted on the colour jet plotter in Lund.

2.6. Discussion of the "growing strategy"

There are several strategies for introducing graphic aids into an institution:

1. Procure standard graphics hardware and software directly from the vendors (As most institutions do)

2. Procure graphics hardware and software according to a predesigned framework (like IGWS), such an approach may be built around a high speed network which serves as glue between different modules.

3. Procure turnkey systems

Data which we have collected shows that:

1. The institutions want to introduce graphics and computers for technical applications; slowly and gradually gain experience.

2. The institutions don't want to loose the investments in hardware, software and know how, that has been achieved during the introduction period.

3. Computer Graphics has a "narcotic" effect. When first started, the use of it will increase.

Taking this experience into consideration, the growing strategy is advantageous to most small and medium sized institutions. Unfortunately until recently, there has not been, to my knowledge, an overall design strategy of a growing pattern combined with basic software to support such a strategy until the design of IGWS.
3. THE GRAPHICS HARDWARE

3.1. Classification of hardware

The variety of graphics devices on the marked are growing wild creating a lot of problems for the system designer. These problems can be reduced by making a classification of the devices and investigate the characteristics of the classes on the system performance.

There are several ways of classifying graphics hardware, but for our purpose it is convenient to break it up in two main classes: Vector devices and raster graphics devices.

Fig. 3.1. Classification of Graphics hardware.
Important parameters to consider in this context are such as hardware cost, software complexity, load on host computer, response time, real time motion, remote operation, bandwidth, need for operators etc. Even if it is difficult to overlook the impact of different hardware on the system, there are some rules of thumb:

Vector graphics plotters are faster for internal generation of pictures, but slower for the actual drawing on the viewing surface than raster plotters.

High performance plotters such as large flatbed plotters and colour ink jet plotters need frequently operator intervention, which means that they should be utilized as centralized resources handled by a dedicated computer.

Unbuffered, single CPU refresh vector graphics displays put a lot of load on the host computer for refreshing the images which may degrade the computer considerably.

Distributed intelligence systems (as discussed in chap. 2.2) usually have a very good response time and possibilities for real time motion, even if the load on the host is very high, because the time critical processes are running in the workstation itself.

Storage tubes can operate reasonably well with small bandwidth.

Framebuffers without powerful local processors and instruction set demands extremely broad bandwidth.

To help evaluating graphics hardware a course table of properties for characterizing the hardware is shown in fig. 3.2.
<table>
<thead>
<tr>
<th>FUNCTIONS</th>
<th>VECTOR GRAPHICS</th>
<th>RASTER GRAPHICS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STORAGE</td>
<td>REFRESH</td>
</tr>
<tr>
<td>ADDRESSABILITY</td>
<td>HIGHER</td>
<td>HIGHER</td>
</tr>
<tr>
<td></td>
<td>LOWER TO</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>RESOLUTION</td>
<td>HIGHER</td>
<td>HIGHER</td>
</tr>
<tr>
<td>LINE SMOOTHNESS</td>
<td>HIGHER</td>
<td>HIGHER</td>
</tr>
<tr>
<td>BRIGHTNESS LEVEL</td>
<td>LOWER</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>COLOUR POTENTIAL</td>
<td>NONE</td>
<td>LIMITED</td>
</tr>
<tr>
<td>SHADED AREAS</td>
<td>LIMITED</td>
<td>NONE</td>
</tr>
<tr>
<td>MOTION DYNAMICS</td>
<td>LIMITED</td>
<td>HIGHER</td>
</tr>
<tr>
<td>FLICKER LIMITS</td>
<td>NONE</td>
<td>HIGH/FAST</td>
</tr>
<tr>
<td>UPDATE/DELETE</td>
<td>LIMITED</td>
<td></td>
</tr>
<tr>
<td>COST TREND</td>
<td>SLIGHTLY LOWER</td>
<td>LOWER</td>
</tr>
<tr>
<td></td>
<td>LOWER</td>
<td></td>
</tr>
<tr>
<td>DISPLAY DEFICIENCY:</td>
<td>-STATIC</td>
<td>-FLICKER</td>
</tr>
<tr>
<td></td>
<td>-NO SELECTIVE</td>
<td>-PRICE</td>
</tr>
<tr>
<td></td>
<td>ERASE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-COLOUR</td>
<td>-DYNAMIC</td>
</tr>
<tr>
<td></td>
<td>-BRIGHTNESS</td>
<td>-HANDLING</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISPLAY ADVANTAGES:</td>
<td>-LINE QUALITY</td>
<td>-MOTION- DYNAMICS</td>
</tr>
<tr>
<td></td>
<td>-FLICKER</td>
<td>-DEPTH</td>
</tr>
<tr>
<td></td>
<td>FREE QUALITY</td>
<td>-TV-COMPATIBILITY</td>
</tr>
<tr>
<td></td>
<td>-RESOLUTION</td>
<td>-SUPER- IMPOSE</td>
</tr>
<tr>
<td></td>
<td>-BRIGHTNESS</td>
<td>SLIDES GRAPHICS</td>
</tr>
</tbody>
</table>

Fig. 3.2. Characteristics of different graphics device classes.
3.2. Distributed Intelligence Systems

The distributed intelligence systems need further discussion because they may range from simple refresh terminals which impose quite a heavy load on the host; to very advanced workstations with capabilities for running stand alone.

The distributed intelligence system try to exploit the advantages without suffering the disadvantages of dedicated stand alone systems and of large time-shared computers that directly support one or more display consols.

Two basic software techniques are used today with these systems: "fixed-function" and "programmable-satellite".

In fixed-function systems a fixed set of general purpose graphical functions are chosen and implemented on the satellite by the graphics installation's software programmer. The application program is totally host resident and controls the satellite by "dummy" subroutine calls to this fixed set, these calls are converted to messages which are sent to the satellite where they are activating the corresponding actual subroutines.

This approach relieves the application programmer of the burden of programming two computers. But because the application program runs exclusively on the host computer, a lot of user interactions have to be reported to, and serviced by the host. These so-called "significant events" include many decisions at a very low level which have to be done by the application program and which cannot be predetermined or which are not supported by the satellite resident set. Each time a message has to be sent from the satellite to the host, the host computer's operating system has to activate the application program which takes a - perhaps trivial - decision and sends a message back to the satellite. Some delay will obviously occur using this procedure.

Nevertheless, the fixed-function approach is useful in a lot of cases, especially in those where only a low interaction rate between the graphics system and the user is required.

Programmable-satellite systems, on the other hand, allows the application programmer to directly program the satellite computer. Thus, the strong connection between satellite and host can be relaxed, and the application programmer can try to find a division of labor optimized for a particular application. On the other hand, this might constitute a danger if the satellite has enough computing power and sufficient fast background memory: it can favor and induce a software pendant of the "wheel of reincarnation" [15].
In order to optimize the division of labour one needs optimization criterions and both static and dynamic data related to the particular application:

- Static data is the data which is not changing during the run and can be extracted directly from the programs, such as:
  - length of object code per module
  - type and number of parameters
  - type and number of global data (Commons);
  - etc.

- Dynamic data is data which must be collected at run time, such as:
  - run time per module
  - number of modules called
  - type and number of I/O-activities
  - type and frequency of user interrupts

The static data may partly be provided by the operating system and their collection is not difficult in nature.

Dynamic data are utilization and data dependent because they depend on how many times the particular routines are activated and with the dimensionality of the data. Therefore they have to be collected during a real employment of the program to be distributed.

The optimization imposes at least two distribution steps. First of all, an initial distribution has to be found. Any realizable distribution is useful, one may therefore start with the "minimal distribution" of a fixed-function satellite, i.e. only the software absolutely necessary is satellite-resident (interrupt handlers etc.). The entire rest of the software to be distributed are then running on the host computer. Generally, this initial distribution will be far from an optimal one, but it permits the collection of dynamic data during a program run.

After the program has been analyzed, one can optimize according to the chosen criteria such as "average response time", "host usage", "link usage" etc. The parameters for this optimization are the requirements of the software and the user's characteristics contained in it on one hand, and the hardware configuration's characteristics on the other. The decision of which programs and data modules are to be assigned to which processor are an iterative process.
The optimization considerations discussed by Foley in [16] have proven to be useful for this task:

For that purpose the computer network is described by characteristics of the hardware and software:

The hardware characteristics are described by:

- number of network computers
- storage capacities available on each direct access device (including main store) at each network computer
- seek times plus latency times for each storage device at each network computer
- transfer times from these storage devices
- run-times of each subroutine on each computer
- data link transmission time from each computer to each other one
- times for transmitting a message of zero length from each computer to the others

The software characteristics are described by:

- a set of subroutines or processes
- a set of data elements which is to be used by the subroutines
- the size of each data element
- the initial activation probabilities of the subroutines
- the inter-subroutine transition probabilities
- the data element reads per subroutine activation
- the amount of information passed, whenever a subroutine invokes another one

There are several algorithms available for deciding the "optimal" distribution:

The "max flow-min cut" algorithm designed for operating systems to achieve a better paging behavior [17] has been used in the BUGS system [18].

While [19] suggest a related method based on "interconnectivity":
First the "interconnectivity" between the software modules are determined, then the algorithm starts with at least one software module assigned to each computer, whether by technical reasons or by a cyclic search. These modules do not necessarily have to be part of the application program to be distributed.
Out of this initial assignment and the measure "interconnectivity" the "adherence to computers" is computed as a function of all software modules. Now the order of software modules, given by this adherence, allows to obtain the optimal distribution in at most \(n-1\) checks, \(n\) being the number of software modules.

### 3.2.1. The satellite protocol

After the problem of distributing an application program has been treated, we need to discuss how the distributed parts of it can communicate with each other.

As a result of the optimization process, we get the destination for the software modules which are supposed to run on two (or more) different processors. We suppose that the problems of pure data communication between these processors are solved, i.e. that programs on these processors can exchange informations.

In order to handle the communication between the distributed programs, three software events have to be taken care of:

- intercomputer subroutine calls and returns (remote-call remote-return)
- references to global data (Commons)
- I/O to/from external files.

This may be achieved by a preprocessor that modify the subroutines that may be activated by remote calls in such a way that two "dummy routines" are generated for each subroutine in question. This couple of dummy routines takes care of the communication process supported by the run time system which does the necessary data conversion and handles information in Common.

![Diagram of subroutine distribution](image)

**Fig. 3.3. Distribution of subroutines.**
Fig. 3.3 shows a remote-call to subroutine B. The activated dummy B writes a "Call B"-messages which is read by dummy B' (or some other dummy subroutine activated before) on the other computer and ends in a real "Call B". Dummy B now tries to read the response which can be the return from B, but also another remote-call, a file I/O, or an access to a Common area can be required. Dummy B has to handle all these, aided by the runtime system [19].
4. THE DATA RECORDING SUBSYSTEM

As mentioned earlier, data capturing is generally more primitive than the man/computer conversation and data presentation when elapsed time is concerned. The Integrated Graphics Workstation (IGWS) is therefore offering a number of facilities for improving this important task.

The data capturing process can be divided into two main categories:

1. Information which is already available, but has to be transformed into computer readable form. Such as drawings, photographs, blueprints, maps, real world scenes, etc.

2. Information which is being created by the operator as he is working, such as sketches, associated alpha/numeric attributes (dimensions, connectedness, prices, etc.) or spoken words.

The standard tool for inputting the first category of data is the digitizer.

We find two main categories of digitizing equipments on the marked today:

1. The pure hardware systems, such as Tektronix-4954 [20], which picks up the position of the cursor and leaves it up to the user to take care of the generated information

2. The software supported digitizing systems, such as Summagraphics Datagrid II, Digitas [21] etc. which is off-line systems with capabilities for formatting and storing the digitized information. These systems also support simple editing of the data. The final result of the digitizing process is a static file of data which usually needs postprocessing in order to be utilized.

The IGWS subsystem for data recording presented here is an on-line digitizing system integrated with the graphics subsystem in such a way that the information can be immediately edited, used, manipulated and presented.

The use of the data recording subsystem in IGWS is divided into three different phases:

1. Initialization
2. Digitizing
3. Editing
4.1. Initialization

Before starting the digitizing, the user is guided through the initialization phase by a question/answering procedure which is utilizing default values.

The system is also offering functions for device selection, surface administration and menu handling.

Data and commands may be entered to the system from different sources (digitizer, screen, keyboard etc) controlled by the user. In the same way echoes and prompts can be routed to devices chosen by the operator.

4.1.1. Menues

The system keeps an arbitrary number of named menus which may be picked and positioned by the user. Each menu can have an arbitrary number of entries and levels. The menu can be automatically generated on the screen, drawn on a printer/ploter for "clip-and-place" on the digitizing surface, or manually drawn. The physical size of the menu is also specified by the user.

4.1.2. Surface administration

The screen and the digitizing area can be dynamically subdivided into different working areas like: Digitizing area, Menu area, Error message/promting area etc.

Dependent on the application, the digitizing area may again be divided into one, two or four subareas for digitizing in orthogonal projections.

![Diagram](image)

Fig. 4.1. Surface layout.
The subdivision is indicated by means of the digitizing device.

For digitizing in one projection the plane can be given by entering three points on the plane.

4.1.3. Automatic corrections

Usually the drawing to be digitized have to be placed very carefully on the digitizing surface in order to get correct coordinates. In IGWS it is not necessary for the operator to position the axis of the menu and the drawing which is to be digitized parallell to the axis of the digitizer. The operator place the drawing within the correct areas and indicate the directions of the axis. The system will then automatically perform the necessary corrections.

![Diagram](image)

Fig. 4.2. Automatic correction.

4.1.4. True Scale

The operator may want to digitize the drawing etc. in true scale. This will be achieved by entering two points along each axis on the drawing and the actual distance between them.

4.2. Digitizing

The basic element consist of a coordinate tuple for a point together with possible naming or descriptive information.

Information can be added to an element by invoking a special procedure each time additional data are to be entered. It is also possible to set INFO MODE where the system expects information added to every element. The length of the info field is dynamic.
4.2.1. Commands

The standard way of entering commands is by pointing on the menue, but it is also possible to use the keyboard.

4.2.2. Point digitizing

The standard way of entering coordinates is by the digitizer, but it is also possible to switch to other input devices like: the keyboard, the alphacursor, the screen etc. Default mode for the echo is LINE MODE with a solid line connection between the points.

The user may also specify other types of line connection included end points, where only the digitized point is visible.

4.2.3. 2 1/2 D digitizing

Different layers of an object may be digitized and stored in pseudo segments.

The distance between the layers may be specified by the user.

![Diagram](image-url)

Fig. 4.3. Object to be digitized.

4.2.4. Orthogonal projections

The orthogonal projections are digitized according to the European standard (see fig. 4.4).
Fig. 4.4. Orthogonal projections according to European standard.

4.2.5. 3D reconstruction

Software tools for reconstructing the 3D object from either the orthographics projection (mechanical drawings) or perspective view (photographs) will be provided. The mathematics of going from two 2D-positions to the corresponding 3D-position of a point both for orthographics and perspective views are described in [22], tested and will be implemented in IGWS for the reconstruction process.

4.2.6. Additional information

Additional information is normally entered by the keyboard, but it is also possible to use a special purpose menu, or specify that the info is the same as the previous one.
4.2.7. Digitizing of standard symbols

Standard symbols or user defined symbols can be "digitized" by identifying the symbol on the menu and giving the position where the symbol is to be included.

4.2.8. Interface to IDIGS

The data entered can be stored as segments identical to the IDIGS pseudo segments [23]. The IDIGS file system will therefore be used for storing and retrieving of digitized or partly digitized picture segments.

4.3. Editing

Editing of the digitized image is possible both during the digitizing process and as a stand alone activity.

4.3.1. Point editing

A point may be deleted, moved or inserted anywhere within the digitizing area. The attributes such as colour, connecting line type etc. may also be changed.

4.3.2. Symbol editing

A standard or user defined symbol can be deleted, moved or inserted anywhere within the digitizing area. Both groups of symbols and single symbols may be changed.

4.3.3. Information editing

Additional information connected to any element can be deleted, changed or added.

The Data Recording Subsystem has been implemented [24], and is now in production at RUNIT.
5. THE IMAGE PROCESSING SUBSYSTEM

Computer Graphics and Image Processing have traditionally been treated as disjoint subjects. However, since both deal with pictures and have a number of elements in common, it is natural to include Image Processing into the Integrated Graphics Workstation.

![Diagram of image processing subsystem]

Fig. 5.1 Relation between computer graphics and image processing.

Computer graphics can be characterized as a system that are accepting data as input and generating pictures as output. The most advanced types of graphic images take the form of shaded or coloured simulations of three dimensional scenes. These pictures are generated by scanning a database that describes the objects of interest.

Image Processing and Pattern Recognition, on the other hand, begin with an already existing image, typically a digitized photograph. There are two main types of processing that are performed. Firstly there is the image enhancement function. In this case, the desired output is simply another picture that has better or other quality in some sense than the original. Either the contrast is enhanced, or various types of noise or geometric distortions are removed.

The second, and more ambitious function, is to extract some information from the image concerning the picture content.
This can simply be a matter of identifying regions of common texture or it may concern determination of whether a certain object appears in the picture or not. The most advanced image "understanding" systems attempt to reconstruct a complete three dimensional description of the object from just a few images of it. In effect, this second function of image processing takes pictures as input and produces a computer internal object model as output [25].

IGWS therefore makes use of an Image processing subsystem for enhancing existing images and data capturing (automatic digitizing). The image processing subsystem may on the other hand use Computer graphics for manipulating the pictures, enhance the output and generate controlled test images.

Finally the two techniques can be combined for making very powerful foreground, background or overlay pictures.

The central item in either of the two operations is the data model. Many aspects of computer graphics concern geometric operations on this data, often beyond that which is strictly necessary to draw pictures. Similarly, geometric modelling is important to many of the geometric reconstruction tasks of image processing.

Another reason for including Image Processing into the Integrated Graphics Workstation is for utilization of common hardware.

The development of inexpensive random access memories has made it possible to create random access refresh memories that are large enough to devote an entire word of up to 24 bits to each pixel. The result is a display device called a random access frame buffer that has capabilities which are attractive to both computer graphics and image processing [26].

5.1. Data input from image processing

An important application of the techniques of image processing is as mentioned object model generation. This object model may be either a three dimensional representation of some shape or an image of the surface features on the objects. The ideal general purpose image understanding system, can take a few photographs of an object and build a database describing the three dimensional shape.
Fig. 5.2. An ideal image understanding system.

Except for relatively simple shapes, the full use of this system is some time ahead. However, combined with a human operator, digitizing of objects and drawings can be done quite efficient [61].

5.1.1. 3D Reconstruction of scanned images

By means of for instance a TV-camera, several views of an object are scanned into the computer. Edge detection techniques are used to find the silhouette edge of each image. Each image may then be defined as generalized cone with the apex at the camera position and the sides intersecting the silhouette edge. The entire object lies within this cone. The intersection of several such generalized cones produces a shape approximating the initial object.

This technique is developed by Baumgarten [27] and the corresponding data-structure "winged edge", is also used by the data recording subsystem of IGWS.

Another automated technique that can be used in certain cases begins with stereo pairs of images. Corresponding regions of the two images are identified by correlation techniques. The relative displacements in the horizontal direction can be used to calculate the distance to the object. This is used primarily for ranging of single objects but it may also be used for shape determination.

The preceding methods make use of the geometric properties of several images used. Since the eye and brain can extract three dimensional interpretation out of an image from just the lighting information, this process may also be performed by the computer. Computer graphics must calculate shading intensities by some model of light reflection from the surfaces. Such models are based on the direction of the surface normal and the light source direction. This process can be inverted by taking a photograph as the source for input of intensities and then calculating the normal vector directions that must have been on the object to generate
those intensities. The solution can be exact if there are three identical images with only the light source direction changed. A more heuristic approach is necessary if only one view is available, but some success has been obtained by Horn [28].

5.2. Combined applications

If we have access to a frame buffer, a program may combine two images from totally disjoint sources with no knowledge of where they came from. A computer graphics algorithm known as the painters algorithm [29] utilizes this capability by drawing all visible objects in order back to front. Each image overwrites that which is currently in the buffer, thus hiding the further parts. Thus entirely different algorithms may be used for different objects in the scene. This is aided by the fact that the program can read back portions of an image from the frame buffer memory and do some local image processing on it.

Real images can be mixed with raster graphics in two ways:

1. On the screen by superposition

2. In the frame buffer so that the image can be manipulated frame by frame.

The first method brings new possibilities for digitizing, data extraction and annotation. The second has opportunities in for instance animation.

5.3. Capturing of data from the creative process

Collection of informations created by the user as he works such as sketches, symbols, characters and numbers is the second category of data capturing. This can be done by directly capturing the user's drawing actions as input to the system [30].

As an example let us watch a designer creating a new mechanical part. Traditionally he will use a pen and a sheet of paper for sketching the rough ideas of how the object should look like. Then he will use the same tools for refining the sketches before he submit it to the drawing department for working out the final work drawing.

By switching the pen with a stylus and the paperpad with a tablet, the designer's drawing actions can be captured directly by the computer. Utilizing pattern recognition methods, the system may now extract the necessary information for building up the internal model of the object to be designed:
Fig. 5.3 A sketch of the Y-Z projection of a mechanical object.

This model can then be manipulated, modified and displayed from any point of view with perspective and shading.

Fig. 5.4 Prospective Projection With Shading.

The recognition subsystem consists of the tablet, a line following algorithm, a three dimensional object reconstruction system and a symbol and alpha/numeric recognizer.

The line following algorithm tests continuously a given set of points, the direction change within these points is compared to two given
tresholds \( t_1 \) and \( t_2 \). If the direction change is less than \( t_1 \), it is a part of a straight line, if it is between \( t_1 \) and \( t_2 \), it is part of a curve and if it is larger than \( t_2 \) it is an angle. The symbol and alpha/numeric character recognizer extracts the features of the pattern such as straight lines, rotations, inflection points, invisible strokes etc. and use a pattern matching technique for the classification [30].

![Diagram of handwritten symbols]

\[ \text{L : straight line} \quad \text{IS: invisible stroke} \]
\[ \text{LR: left rotation} \quad \text{P : cusp} \]
\[ \text{RR: right rotation} \quad \text{I : inflection point} \]

**Fig. 5.5** Feature descriptions of handwritten symbols.

The recognition process consists of a main and a back-up step. The main recognition process comprises the initial filtering and local processing of measured data, the rough feature extraction, and the decision making. The back-up step includes the refining and correction of the previous extracted features and is used when the previous decision making step cannot give a definite answer.
6. BASIC GRAPHICS SOFTWARE

As described in chap. 2 one of the most powerful tools for man/computer conversation is computer graphics, because the dominating information channel into the human brain is the visual system [8].

A major problem with this tool, up to recently, has been the poor accessibility because of the software complexity and lack of standardization.

To succeed with the development of an integrated graphics workstation, the basic graphics software must be powerful enough to support advanced devices, modular enough to be distributed, and standardized.

![Diagram of device-independent graphic system](image)

Fig. 6.1. Device Independent graphic system.
Internationally a lot of effort is going into standardization of basic graphics software (the Viewing subsystem):

The German DIN proposal: Graphical Kernel System (GKS) [32] is now in the process of being officially accepted by ISO as an international standard for a two dimensional graphics system.

The American proposal, built on the three dimensional ACM SIGGRAPH GSPC "Core" [33], has been withdrawn. It has, however, been very influential on the total standardization process.

The Norwegian three dimensional graphics proposal "Interactive Device Independent Graphics System" (IDIGS), which is a candidate for an international 3D-graphics standard is one part of this thesis [23].

6.1. Standardization of Graphics Software

The strongest reason for standardization is to achieve portability, i.e. the ability to transport graphics applications from one installation to another with minimal program change. [34]

A fundamental strategy for achieving portability is to provide features which shield the application program from specific hardware characteristics. This shielding is at the functional level and need not imply expensive software overheads. The programmer describes his graphical world to the system in device-independent world coordinates. He also specifies, in normalised device coordinates, where on the available logical view surfaces the view of his object is to be placed.

Similarly, he can specify which logical input devices he wishes the operator to use at what time during the execution of the application program without concern for the hardware-dependent protocol of the actual devices. The mapping from the logical input and output devices (view surfaces) to the physical devices for a specific configuration is handled by the system implementation for that configuration, and need not concern the application programmer.

Another strong justification for standardization is the user portability which means a considerable saving in education effort for students and users moving from one site to another.
On the other hand we have the problem with too early standardization which leads to standards which nobody want to use because the field is moving so fast that the methodology and technology may be outdated when a standard is reached. In the case of basic graphics software, I think the timing is about right, because of the clear boundaries identified at the SEILLAC I conference in France 1970 (see chap 6.1.2) and the results achieved by Graphical Standards Planning Committee (GSPC) which is reported in [35] and [33]. Another strong reason for standardization now, is the variety of graphics equipments which is continuously pouring on the market. Both the vendors and the users need guidelines in order to avoid complete chaos.

6.1.1: The impact of the IFIP WG 5.2

The standardization work really got started in 1976 with the IFIP WG 5.2 workshop on methodology in computer graphics [35].

Two major results emerged from the workshop:

1) The fundamental division between viewing and modelling. The importance of this division is due to the solution of two problems:

   . The clear-cut limitation of the standardization task

   . The confusion which arises because transformation may be used for two purposes, viewing and modelling.

As the report from the SEILLAC meeting puts it [35]:

'The solution appears to lie in dividing the required facilities into two parts, the viewing system and the modelling system. The modelling system allows objects to be defined in their own local coordinates; it provides functions for composing and combining transformations, and for applying these transformations to local coordinates. The result of the transformations is typically used to produce a definition of the viewed transformation in a world-coordinate system, suitable for application of the viewing transformation. Using the viewing system, we set up the appropriate viewing transformation, and then generate the picture using the definition in world coordinates; this will typically be done by calling line-drawing functions with world-coordinate arguments.'
2) At the conclusion of the workshop it was clear to a number of the attendees that considerable progress could now be made towards the design of a standardised graphics system. The GSPC (Graphics Standards Planning Committee) took up the task of developing the methodology further by designing the 'Core' system which is described in the GSPC 1977 Status Report [36] and the revised 1979 report [33]. These reports have since been the reference for further standardization work in the USA, Germany, (GKS) [32], Norway (IDIGS) [23], ISO etc.

6.1.2. Two-dimension versus three-dimension

A graphics subroutine package for viewing two-dimensional (2D) objects is well suited for many applications such as drafting mapping etc. On the other hand, many applications are concerned with displaying representation of parts in the real, three-dimensional (3D) world. The programmer of a 3D application can use a 2D package, but must explicitly program the appropriate projection from 3D to 2D.

GKS is a pure 2D system. The main argument for this decision is that the majority of today's applications of computer graphics is in 2D. It is therefore critical that the process of describing 2D objects is not complicated by the need to accommodate 3D. The programmer must be able to use the familiar concepts of windows, viewports and 2D output primitives specified in the XY-plane. The size and speed of the programs also played a role in the decision. IDIGS supports both 2D and 3D, a goal which can be achieved either by disjoint 2D and 3D output primitives and viewing capabilities, or with intermixed 2D and 3D output and a unified viewing capability. In ILIGS I have chosen the latter alternative in order to have one integrated system. What we are loosing in speed and space for pure 2D applications is gained by user friendliness. The 2D output primitives use as their z-coordinate the z of the Current Position (CP). This also opens the possibilities for 2 1/2 D applications.

6.1.3. Output primitives

One of the dominating issues in the discussion of graphics standardization is the CP.

One problem can be illustrated by the following example [35]:

```
INIT
MOVE (0,0)
DRAW (1,1)
TRANS (3,3)
DRAW (2,2)
```
Initially we have the A coordinate system:

The sequence MOVE (0,0) and DRAW (1,1) produces the following situation.

Note that the current position is now at (1,1) in the A coordinate system. We then have the operation TRANS (3,3) which is seen by the graphic system as providing a new local coordinate system within the same sheet of paper which is used as a temporary coordinate system for subsequent line drawing.

The new coordinate system B is given as follows:

The translate command has defined a new coordinate system B which has its origin at the position (3,3). The problem is that we have defined a transformation operation within the drawing commands. What happens to our 'current position'? The new current position must be defined with respect to the new B coordinate system. However, it is unclear what its value should be.
This CP problem was the initiating factor for the subdivision into a viewing and a modelling system. However the CP issue is still controversial. Essentially there are three possibilities:

1. No CPs.
2. One CP.
3. Many CPs.

GKS has chosen the first alternative with no CP. Instead of the usual LINE and MOVE primitive, GKS offers a POLYGON command which generates a polygon defined by a point sequence. Traditionally CP has been used extensively in graphics applications. IDIGS has therefore used common practice as the strongest argument for choosing the second alternative — one CP.

Raster graphics has during the last years become a viable alternative to linedrawing graphics and the increasing pressure for including primitives for handling raster graphics in the standard has been taken into consideration.

GKS has expanded the output primitives with two raster graphics primitives:

1. FILL POLYGON: a polygon is filled with a uniform colour.
2. PIXEL ARRAY: an array of pixels with individual colours is drawn.

IDIGS offers polygons of four types: contour, plain, shaded and patterned with possibilities for hidden surface removal included.

Attributes define characteristics of primitive elements, segments and workstations such as linestyle, colour, visibility, text representation etc.

Only GKS offers workstation attributes which control the appearance of output primitives on a specific workstation; the quality and size of the display space and the immediate visibility of primitive generation and picture manipulation. All attributes have a default value which may be changed by invoking the set-attribute routines. GKS offers also a PEN attribute which overrides the colour, intensity, linestyle and linewidth attributes in a device-dependent fashion. IDIGS offers an attribute stack for pushing and popping the attributes. The last feature is included in order to utilize advanced hardware and nested segments.
6.1.4: Input

The goal of the input primitives is to provide a framework for interaction which is independent of the particular physical input devices available on a given configuration. This may be achieved by the concept of logical input devices, which are abstractions of typical physical input devices.

At least five classes of logical input devices have been identified and accepted by the different proposals (even though the names may differ):

1. PICK - identify a segment and a primitive within a segment.
2. TEXT - provide alphanumeric information.
3. CHOICE - provide integer numbers specifying alternatives.
4. LOCATOR - provide position information.
5. VALUATOR - provide scalar values.

In general we have two main groups of input devices: Event and sampled devices. Event devices are used by the operator to signal an event to the application program. An event report, containing data related to the state of the event device when the event occurred, is placed on a first-in/first-out event queue. This report may later be removed from the queue by the application program.

Sampled devices have values which may be sampled by the application program any time during a session.

IDIGS provides a 'trigger' mechanism for building up compound input devices to any level. [37]

A large number of applications do not require simultaneous use of multiple input devices, or for other reasons do not need asynchronous input.

Synchronous input facilities are therefore also provided in order to avoid needless overheads and complexity.

6.1.5: Text handling

Even if text is an important part of many graphical applications, little effort has been made in the past to make it professional, except for special applications where the text is playing the leading part. The standard recommendations take a step forward in this direction. IDIGS has for text handling adapted the model of the 'Core' system which has a number of useful attributes. [33].
1. FONT which gives the user the option to choose between different character fonts like Standard, Roman, Italic, Greek etc.

2. CHARPRECISION specifies the precision for text appearance:
   
   - 'String precision' is used to select the 'best fit' among available hardware character sizes for the whole string
   
   - 'Character precision' is used to select the 'best fit' among available hardware character sizes for character by character adaption
   
   - 'Strokeprecision' text must be displayed as if each character where made up by short lines (strokes) exactly according to the attribute specifications.

3. CHARSIZE determines the width and height of the character drawn.

4. CHARPLANE defines the plane on which characters appear. (Figure 6.2.).

5. CHARUP determines the up direction on the character plane. (Figure 6.2.).

6. CHARPLANE is used to determine the string direction within the character plane (right, left, up, down). (Figure 6.2.).

These facilities makes it in principle equally natural to write for instance Arabic or Chineese as English.

---

Fig. 6.2. Several values of CHARPLANE and CHARUP [33].
6.1.6. Viewing

The subdivision of graphical systems into viewing and modelling was the origin of the concept of the synthetic camera which takes a 'snapshot' of an object, in the graphical world, massaged and modelled by the modelling system.

The application program describes the view of an object by specifying a viewing transformation. A viewing transformation selects a region of the graphical world that is to be displayed and specifies how objects in the selected region are mapped to the view surface. The view surface is a rectangular area of the two-dimensional output surface of the device in use. The coordinate system that specifies positions on a view surface is called the normalised device coordinate system.

In GKS, which is a 2D system, the viewing transformation is completely specified by a window in world coordinates and a viewport on the view surface. The window may be any rectangle in the xy world coordinate plane. The viewport is a rectangle specified in normalised device coordinates and its edges are parallel to the vertical and horizontal edges of the view surface. A window is used to clip an object, that is, it is used to determine the portion of the view surface bounded by the viewport.

In GKS, the model of the synthetic camera has been relaxed and it is possible to generate primitives outside segments, directly on the viewing surface.

The three-dimensional viewing in IDIGS are more complicated since the viewing transformation includes a projection from three dimensions to two dimensions in addition to clipping and mapping to the view surface. A projection is specified by a view plane within the graphical world and either a centre of projection (an 'eye'-point') for a perspective projection, or a projection direction for a parallel projection. Because of conceptual clarity, IDIGS has chosen to use the synthetic camera model.

Two types of clipping compose the three-dimensional viewing system. The first, depth clipping, clips objects to the region between two planes parallel to the projection plane. The second clips objects so that their projections lie within a window specified in the projection plane. These two clipping rules define a visible region or view volume in three dimensions. For perspective projection the view volume is a truncated rectangular pyramid. (See figure 6.3.) For parallel projection, the view volume is a finite parallelepiped. The mapping from the window to the viewport on the view surface is similar to the mapping in the two-dimensional case.

As mentioned earlier, IDIGS treats the two-dimensional and three-dimensional viewing in an integrated fashion; the former is a subset of the latter. The viewing capabilities are such that the programmer of a two-dimensional application need not know about the three-dimensional viewing constructs.
6.1.7. Storage of graphical information

In GKS graphical output primitives may be generated outside segments or they may be grouped in segments. In IDIGS primitives can be generated only in logical segments. The argument for this approach is conceptual clarity (the synthetic camera).

The status of the segments may be temporary if the application program does not require picture part-modification capabilities. The segment may be retained (with an identifying segment name) if the application program needs the capability to modify parts of the whole picture without interfering with the rest of the graphical data structure. The segments may also be stored away in pseudo buffers or libraries for short or long time storage for later retrieval and insertion into the viewing pipeline (IDIGS). (See figure 6.4.)

The primitive elements are run through the viewing pipeline and then routed to any of the active device drivers (including the pseudo driver). From the pseudo driver the segments can be stored in the pseudo buffer for short time storage or in the library for long time storage from where it can be entered into the pipeline again.

For the purpose of long term filing of graphical information and transferring of pictures among different computing facilities with different host computers and different graphical capabilities, the output primitives may be stored on a device independent intermediate display file called a metafile.

Figure 6.4. Pseudo buffers and library.
Output to a metafile may be thought of as simulating a passive output device (eg, plotter), while reading and interpreting a record from the metafile does not correspond to an input function. Reading has the same effect as calling the function which previously generated the record. For instance reading and interpreting a POLYGON record will cause the same effect as if the sequence of LINE functions were called.

6.2. System Design

The design and testing of an interactive graphics system like IDIGS [23] is a time- and work consuming research activity involving:

- systematic collection of user requirement
- system requirement specification
- research in order to create new methods and techniques.
- functional specifications and testing

The testing includes:

- user reactions
- system performance
- acceptance for international experts

It is obvious that such a huge task can't be covered by one person only, I have therefore drawn heavily on my research group at the Computing Centre at the University of Trondheim (RUNIT) [38], [24], [40], a number of diploma students at The Norwegian Institute of Technology (NTH) which I have been supervising [38], [41], [42], [43], and the international graphics community such as ACM SIGGRAPH [36], [33] and ISO/TC97/SC5/WG.2-Graphics [32] where I have been heavily involved.

6.2.1. Collection of user requirements

GPGS-F [38] was implemented under my supervision and became an informal Norwegian graphics standard. In the first trial period we collected experiences and reactions from the graphics community at NTH/SINTEF and improved the system until it reached a relatively stable state, then GPGS-F was introduced on the Norwegian market. The Norwegian Graphics Interest Group (NORSIGD) was founded with myself as chairman. One of the major tasks was to collect experience from a wider audience for further improvement of the system.
The collected data and experiences with GPGS-F was used as basis for the user requirement specification for IDIGS.

6.2.2. System requirement specification

The system requirement specification has been a long procedure, studying available literature such as [2], [35], [36], [39], [44], [45] etc., testing available graphics system such as GPGS-F [38], GINO-F [46], PLOT 10 [47], IGS [48], GHOST [49] etc., participate in international workshops on graphics methodology [35] etc. and utilizing the user requirement specifications from chap. 6.2.1.

6.2.3. Research in order to create new methods and techniques

Several detailed research projects of different complexity was undertaken in order to create and test new methods and techniques such as viewing, attributes, raster techniques, segment hierarchy, metafiles and input.

The input research is the only one which is reported in detail in this connection (see [37] which is a part of this thesis), partly because the research resulted in a complete new concept and partly because that concept influenced the input model of the proposed 2-dimensional international standard, GKS [32] considerably. The other research results are briefly reported here or baked into the design of IDIGS (see [23] which is also a part of this thesis) and will not be reported explicitly.

6.2.4. Functional specification and testing

Based on user requirements, the system requirements and the research results, the functional specifications was worked out according to the following procedure:

1. Identify possible desirable functions
2. Formulate the issues precisely
3. Work out the possible alternatives
4. Make a list of pro and con arguments for the different alternatives
5. Perform tests and discuss the specifications with national and international experts (NORSIGD, ISO etc.)
6. Make a resolution and include the result in the IDIGS specification
7. If necessary go to pt. 1 again
It will lead to far to go into detail of all the arguments and evaluations in this thesis, only a sample of the issues is therefore included in order to illustrate the method:

6.3. Examples of Some Issues

ISSUE: PRIMITIVES OUTSIDE SEGMENT

DESCRIPTION: Primitive elements such as lines, points and characters may be generated both inside and outside an open segment.

ALTERNATIVES:

1. Primitives can be generated outside segment
2. Primitives can not be generated outside segment

ARGUMENTS

a) Pro 2: Encourage structured programming
b) Pro 2: Better portability
c) Pro 1: Save redundant open/close calls if segments is not retained
d) Con 1: The useful synthetic camera analogy is violated

RESOLUTION: Alternative 2: Primitives can not be generated outside segment.

ISSUE: CALL OF WINDOW AND VIEWPORT WITHIN AN OPEN SEGMENT

DESCRIPTION: Window and viewport may or may not be changed while a segment is open.

ALTERNATIVES:

1. Window and Viewport can be called within an open segment
2. Window and Viewport can not be called within an open segment

ARGUMENTS

a) Pro 2: Encourage better programming habits
b) Con 1: Violate the synthetic camera approach
c) Pro 2: Easier to implement
d) Pro 1: More flexible
e) Con 1: Bad consistency with viewport transformations etc., which are performed on the entire segment
RESOLUTION: Alternative 2: Window and Viewport can not be called within a segment.

ISSUE: RESETTING OF THE ATTRIBUTES TO DEFAULT VALUES WHEN A NEW SEGMENT IS OPENED

DESCRIPTION: The attribute setting, which is a mode operation, may be set to default when opening a new segment in order to maintain the control of the attributes when stored segments are to be inserted. If the attributes are not set to default, the attribute values must be stored together with the segment.

ALTERNATIVES:

1. The attributes are not reset to default values
2. The attributes are reset to default values

ARGUMENTS

a) Pro 2: The attribute values are known when inserting, transmitting etc. a segment
b) Pro 2: Use of an attribute stack will easily set the attributes to the values of the previous segment
c) Pro 1: Easier to use for simple applications
d) Pro 2: Well defined segments

RESOLUTION: Alternative 2: The attributes are reset to default when a new segment is opened.

ISSUE: NUMBER OF WORKSTATIONS ACTIVE SIMULTANEOUSLY

DESCRIPTION: The number of simultaneous active workstations is an important issue for workstation flexibility and implementation complexity. It will also influence both system size and programming complexity. An alternative to several simultaneously active workstations is the use of one active and several background workstations (a background workstation produce output only).

ALTERNATIVES:

1. Unlimited number of workstations active simultaneously
2. One workstation active and other workstations acts as background workstations (e.g. output only)
3. Only one workstation active simultaneously
4. All three alternatives dependent on level
ARGUMENTS:

a) Pro 1: Flexible and powerful  
b) Con 1: Difficult to implement in most timesharing environment  
c) Pro 2: A good compromise of flexibility and complexity  
d) Pro 3: Simple  
e) Con 3: Too limited for advanced applications  
f) Pro 4: The flexibility and power of the system can be chosen dependent of need.

RESOLUTION: Alternative 4: All three alternatives dependent on level.

ISSUE: STRUCTURING OF THE LEVELS

DESCRIPTION: The level approach opens the possibilities for choosing the kind of graphics system which best suits the current needs. To achieve this, the subdivision into levels should be clean and functional i.e. the user can choose the capabilities he needs for each major class of functions without introducing unwanted complexity of other functions.

It is for instance not necessary that raster is needed together with input. A functional subdivision may be as follows: OUTPUT, INPUT, DIMENSIONALITY, RASTER/NO-RASTER (ev. also MULTI-WORKSTATION) which constitute a 4-(5) dimensional feature space with discrete entries, each of which is a valid subset of the Graphics System.

The actual system level may be checked by INIT (OUTPUT, INPUT, DIMENSIONALITY, RASTER [MULTI]) where each parameter describes the requirements of the actual system.

ALTERNATIVES

1. Functional unstructured levels (as GKS)  
2. Functional structured levels

ARGUMENTS:

a) Pro 2, Con 1: Dimensionality, raster, input etc, are quite independent features which are not naturally given a level scalar number  
b) Pro 2: Full control of introduced features  
c) Con 2: Too many valid subsystems  
d) Pro 2, Con 1: It does not seem convenient that programs which needs some simple input (crosshair etc.) should be classified together with programs using raster and retained segments

RESOLUTION: Alternative 2: Functional structured levels.

ISSUE: TEXT CAPABILITIES
ALTERNATIVES:

1. Medium level text capabilities (GKS proposal [32])
2. Extended text capabilities (GSPC proposal [33])
   (String justification etc. included)

ARGUMENTS:

a) Pro 1: Simple to use and implement
b) Con 1: Text is an important and timeconsuming part of graphics and needs
   therefore powerful functions

c) Pro 2: Powerful. String justification for instance is very desirable for
   diagram plotting etc.
d) Con 2: Too complex

RESOLUTION: Alternative 2: Extended text capabilities

ISSUE: POSITION OF THE WORKSTATION VIEWPORT

DESCRIPTION: The positioning of the viewport on the workstation
surface may or may not be controlled by the user.

ALTERNATIVES:

1. A default position is used
2. An application programmer specified position is used

ARGUMENTS:

a) Pro 2: Control of the viewing area
b) Con 1: Multiple viewports can not be utilized

RESOLUTION: Alternative 2: An application programmer specified position is
used

ISSUE: COMPOUND INPUT DEVICES

DESCRIPTION: In order to achieve sufficiently powerful input, it may be
possible to combine input devices into high performance compound input
devices.

In this case the necessary tools for building compound input devices like:
ASSOCIATE, TRIGGER, ETC. should be provided (see the IDIGS specification)
[23] which is a part of this thesis).
ALTERNATIVES:

1) Only input primitives are provided  
2) Tools for building compound input devices are provided 

ARGUMENTS

a) Con 1: Too limited for fast highlevel input 
b) Pro 2: Powerful tools can be built for fast and structured input 
c) Con 1: Difficult to build high level tools on top of GKS 
d) Pro 2: Well suited for hardware/firmware implementation

RESOLUTION: Alternative 2: Tools for compound input devices should be provided.

ISSUE: DYNAMIC ATTACHMENT OF PHYSICAL AND LOGICAL DEVICES

DESCRIPTION: In order to utilize available physical devices, it may be useful to dynamically attach and detach logical and physical devices. Installation dependent default attachment can be used.

ALTERNATIVES:

1. Static attachment of logical and physical devices 
2. Dynamic attachment of logical and physical devices 

ARGUMENTS:

a) Pro 2: Good utilization of available physical input tools 
b) Pro 1: Simple to use and implement 

RESOLUTION: Alternative 2: Dynamic attachment of physical and logical devices.

The two last issues are concerned with input, which from my point of view is a special weak part of both GKS and GSPC. A research report on the Compound logical input devices of IDIGS is given in [37] which is a part of this thesis. The compound logical input devices have also been implemented and tested by two diploma students, which I have been supervising, with very promising results when flexibility and device utilization is concerned [41], [43].
7. THE COMMAND LANGUAGE

All computer systems need to be monitored.

A well adapted command language therefore plays an important role in the Integrated Graphics Workstation.

The user must be provided with a set of commands by which to control and communicate with the computer systems.

These commands serve four functions: start and stop the system, control which processes are to be activated, contain the data which are to be passed to these processes and inquire information from the system.

By defining the range of commands and their syntax, we define the command language. This is the language "spoken" by the user as he operates the programs.

Every time a programmer writes an interactive program, he must provide at least one command language because this is generally very user dependent. New users need a lot of guidance through the system, while experienced users want to go straight on with the problem solving. Specialists want to use technical terms, while naive operators need simpler languages adapted to their level of competence.

But in all cases the user needs assistance from the program to discover input errors, and help to correct these errors. That means that the system programmer must spend much of his time in designing and implementing the command language(s). It's therefore important to provide a tool, which can simplify the work and improve the result.

7.1. 'A user oriented' command language

Command languages may be general purpose or user oriented and may be looked upon as a communication bridge between the user and the system. Because the users are very individual, it is from my point of view important that the design and implementation of the command language is adapted to the particular user in order to enable him to utilize his knowledge and ideas without being disturbed by a dialogue with the system which feels unnatural to him.

An ability to construct a good dialogue between the user and the program can generally be gained only by a highly developed understanding of human behaviour, and close cooperation with the user during the design phase. Available computer systems shows that these characteristics are frequently underdeveloped among most programmers [50].
Even with highly qualified programmers, the result will seldom be quite satisfactory until the user himself has done the finishing polishing, such as: incorporating his own commands connected to special mnemonics, time-saving abbreviations, including synonyms, or writing his own macros etc. [5].

To make this possible, the first commandment to the designer of the command language is SIMPLICITY, without which the user will seldom be quite satisfied.

The simplicity may appear as structured programming, simplified code, simple syntax definitions, separating the command descriptions from the rest of the program etc. These efforts make it possible even for a non-programmer to make the necessary modifications.

Another problem in designing the command language is that the user gradually gains experience and therefore needs less and less support from the program. On the other hand even experienced users may sometimes get lost and will need detailed instructions to get on the track again.

In order to fully support this commandment, the command language must both keep track of where it is and take care of the operating sequence such that back-tracking is possible. That means that the command language ought to be ADAPTIVE to the user. A framework of such an adaptiv command language is reported by the author [53].

ERROR HANDLING is the third important point in the interaction between the user and the computer system. The main principle must be that the user should never be penalized unduly for an erroneous input. The essential task of error handling is to discover as many errors as possible in the input stream and give correction and easily understandable error messages by means of a good recovery system and then guide the user in correcting his errors.

7.2. Some important aspects of the command language

It is true that many command languages have been developed with English words and near English syntax, but they never seem to fulfill their designer's claims. A training course, or a big thick manual beside one at the terminal, always seems to be necessary. The reason for this is maybe the language guage is only a covering despite its sophistication. Underneath is still a system with the characteristics of mathematics, namely:

- One-to-one correspondence between symbols and their meanings
- Nothing assumed except what is expressly stated
- Deductions made from an absolute invariant frame of references
Everyday thought and conversation have very different characteristics:

- Many words with the same meaning
- Innumerable unstated and even unconscious assumptions
- Meaning strongly dependent on immediate context

Artificial Intelligence people are working on this kind of problems and they will eventually solve them, but not for a long time [51].

In the meantime, we must utilize our knowledge as far as possible to find principles, concepts, standards and notations which will improve the command languages.

Four major principles of man/computer dialogue have been identified [52]:

1. **Expectation and prediction**

   The user's expectations of performing a dialogue with the computer should be met in such a way that he can utilize his experience in communicating with humans. This is only possible if the dialogue is made reasonably predictable.

2. **Context**

   At any point in time, there is a selected set of previous experiences which we are currently using to interpret the messages and to generate our expectation of the next experience.

3. **Experimentation**

   For some reason or another, the designers of man/machine dialogues tend to assume that a user can learn once and for all a sequence of correct patterns of activity and from then onwards never make a mistake. But this is not the case, human nature is to experiment. The system must therefore be able to handle trial and errors without letting the penalty of making errors become excessive.

4. **Motivation**

   The computer dialogue should do everything it can to generate confidence. This may be done by making the user feel that his commands will be obeyed and that the machine is going to help him.
8. VOICE INPUT/OUTPUT

A command language consists usually of written or menu driven commands, but as voice input/output technology is improving that alternative has become more and more attractive.

Fig. 8.1. Voice input/output

Voice is particularly advantageous in the situations where one or more of the following conditions apply:

- The operator's hands are busy
- Mobility is required during the session
- The operators eyes must remain fixed upon a display, an optical instrument, or some object to be tracked
- The environment is too harsh to allow use of a keyboard

Voice is suitable for these applications because it requires neither hands nor eyes for its operations.

Mobility can be provided by telephone or a dedicated radio link where the only parts of the voice data system that must be available at the work place is a telephone or a microphone and some devices for feedback.

8.1. Voice input

Voice is direct data input to the computer without any intermediate data preparation such as keypunching. Up to now good results in speech recognition have been obtained only under certain restrictions, such as limitation of vocabulary, utterance with pauses between words (isolated words) and certain specified speakers [54]. In systems with such constraints, the voice recognition can be performed relatively simple: By storing the speech pattern of each word for each speaker as a reference pattern, individual
difference problems can be solved. An unknown spoken word is matched against the reference patterns, when the best fit is obtained, the word is recognized as the category of the corresponding reference. Limited word recognition can therefore be performed by simple pattern matching. But also here are problems:

Even if the same person pronounces the same word several times, his speaking rate is not always the same. His speech pattern is therefore subject to nonlinear time fluctuations which creates a lot of difficulties for the word recognition systems.

However, several word recognition systems are on the market today such as Threshold Technology, Inc, Centrigram Corp, Heuristics etc.

In these systems, the words must be spoken in isolation. This restriction forces the operator to speak in a stilted, unnatural style that increase the fatigue of the user and limits the operation speed of the system.

There are, however, some systems on the marked that claims to be user independent and recognize continuous speech.

The most impressive (from my point of view) is the DP-100 delivered by Nippon Electric Co. Ltd. [53]. Among the major problems in speech recognition are the time fluctuation and the syllable segmentation. People naturally speak the same word at slightly different speed in different contexts, and since they run their words together, it is difficult for the computer to determine where one word stops and the next word begins.

The human brain is able to perform the necessary time normalization and segmentation with great efficiency. The DP-100 system has solved these problems reasonable well by a recognition principle based on time normalizing pattern matching using a dynamic programming algorithm. The vocabulary of the system are limited to 120 connected words and 900 isolated words.

![Fig. 8.2. Progress of voice input technology.](image)
8.2. Voice output

Voice output, or speech synthesizing, is a promising tool for man/computer conversation because the computer can prompt the operator while he is performing some other tasks that might require his full attention.

Today's talking computers typically utilizes one of two techniques to produce recognizable human speech:

1. Direct speech synthesis. Here sound signals are shaped according to parameters that attempt to capture the essence of phonemes, the smallest fundamental units of speech.

   These parameters derive from both the physics of speech, the nature of obstructions in the vocal tract and the rules of linguistics.

   Because this type of speech output is constructed according to a set of rules, it is also named synthesis by rule.

   A typical example of synthesis by rule voice synthesiser is the Microspeech from Costronics Electronics, which uses a model known as a "serial 3 formant synthesiser" with parallel fricative formant, three formant frequencies and nasal formants.

2. Playback of recorded speech. Here are utterances recorded in small pieces which are selected by the controlling processor. Playback occurs sequentially, stringing together coherent messages. This method is also named synthesis by concatenation.

   The problem with this method is that it requires a lot of storage because the information stored by this method contains a great deal of redundancy. The redundancy problem, however, is avoided by the linear predictive coding used for instance in Texas Instruments "Speak and spell" [56].
8.3: Voice recognition

It turns out that the individual human voice has a unique voiceprint which is as characteristic as the fingerprints. These facts can be utilized as a convenient security key for access control to the workstation.

A security "contest" was arranged by the American Air force between voice, fingerprints and signature to find the best security key for access to military installations. The best result was achieved by use of voice recognition (a not published report from TI).

8.4: Experiences

We categorize the voice operated systems into three types. The first is a pure substitute of function- and alpha/numeric keys. Therefore the utterance of a word or a phrase is corresponding to pushing a key and is easy to introduce into IGWS. The second type is where the voice input device recognizes sentences and understands tasks. A "regular language" which is fairly natural should therefore be used. In today's systems the vocabulary of the last type is smaller than of the first one.
The third type, which belongs more to the future, is one for multiple channel interaction.

![Diagram of a workstation from NPC utilizing voice input.](image)

Fig. 8.5. A workstation from NPC utilizing voice input.

Some preliminary results from studying Voice versus Key stroke interaction in a CAD system is available [55]:

<table>
<thead>
<tr>
<th></th>
<th>trained (more than a week)</th>
<th>untrained (less than an hour)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>key</td>
<td>voice</td>
</tr>
<tr>
<td><strong>data entry</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>o speed</strong></td>
<td>30 connections per min.</td>
<td>50 connections per min.</td>
</tr>
<tr>
<td><strong>o accuracy</strong></td>
<td></td>
<td>△</td>
</tr>
<tr>
<td><strong>o preference</strong></td>
<td>&lt;</td>
<td></td>
</tr>
<tr>
<td><strong>interactive operation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>o speed</strong></td>
<td>30 commands per min.</td>
<td>60 commands per min.</td>
</tr>
<tr>
<td><strong>o accuracy</strong></td>
<td>&lt;</td>
<td></td>
</tr>
<tr>
<td><strong>o preference</strong></td>
<td>&lt;</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 8.6. Voice versus Keystroke interaction is a CAD system.
9. THE TEXT HANDLING SUBSYSTEM

Text constitutes a considerable part of the graphics data presentation such as work drawings, maps, business graphics, documents, artworks, reports etc., even on advanced graphics workstations.

In addition to the standard text processing capabilities as described in for instance [57], IGWS is providing tools for combining text and graphics. Especially for the preparation of documents and reports, it is valuable to handle text and pictures together. This enables the storing of the entire document digitally and eliminate the expenses and the potential errors of hand collating figures into text after printing.

Once text and pictures can be edited and printed simultaneously with sufficient quality, updated copies of documents can be provided on demand. This eliminates the expence of shelf storing of for instance old drawings which is a major problem for a number of manufacturers today. Since text and picture data is stored in a common digital data base, selective document preparation is possible. That is, each copy is an edited and selected copy that contains only the information relevant to the recipient. Such systems are also described in [58] and [59].

9.1. Representation for document production

A document can be represented statically or dynamically dependent on how it is to be treated later:

1. Static representation.
   Here the whole document may be represented as a video image and stored directly on a video disc, or as a raster image stored on more traditional bulk storage devices. Unless advanced methods for pattern recognition is introduced, the electronic storing of the documents in this case serves only as an archive for later retrieval and copying of the finished documents.

2. Dynamic representation
   Here the text and pictures are usually stored on separate accessable layers in such a way that the relations between the items are preserved. This approach gives the possibilities for both editing the text and manipulating the picture before the document is generated. Then the text and image is merged into the requested document and printed.
Fig. 9.1. Merging of text and pictures.

For high level artwork, IDIGS [23] provides fonts and text capabilities as a part of the graphics system.
10. THE DRAWING PRODUCTION SUBSYSTEM (DPS)

Production of high quality drawings is a major problem at most installations because the equipment is very expensive and the local utilization is usually poor.

The increased use of computer graphics in most fields of computing and the wide variety of graphics hardware has therefore created a need for an efficient and easy way to generate the hardcopies based on graphical information. Traditionally there has also been too many problems in utilizing these devices: Some of the obstacles that must be overcome in order to make an efficient drawing production system are listed:

- The various systems use different format for geometric, textual and other data describing a graphical drawing. That complicates the production of plot and drawings on graphic devices since the data have to be treated different dependent of where it comes from and where the drawing is to be produced [40].

- Existing drawing systems such as Versatec and Calcomp programs tend to put a heavy burden on the host computer on which they are running. This stems from the lack of effective drivers for the various graphical devices. Most drivers are implemented solely in software on the host computer, which especially for incremental and raster graphics devices results in a high load on the host computer [60].

- Each system provides a different pile of more or less well written manuals which makes it difficult and expensive to train operators to handle all the different devices.

- The operator control facilities of today's drawing systems are not satisfactory either. It must be a requirement that a drawing system can operate as far as possible automatically with little or no operator interaction. On the other hand, the system shall possess facilities which enables an operator to direct and control the production of drawings at his own choice.

- The operator is usually confronted with a number of facilities that he can not adequately handle, because of too many differences.

- The utilization of this often expensive drawing devices are frequently quite low, because too few users have easy access to the facilities.

The main problems is therefore to find a uniform way to utilize this large variety of systems and to make them available for larger user communities.
10.1. Functional specifications of DPS

From the user's point of view, the access to DPS will be equivalent to the access to any local device driver in Digi. However, the actual drawing production is controlled by the operator of DPS.

The user wants sometimes to be present or to give special instructions when a particular drawing is produced. The reason may be that the drawing is to be produced on special material such as maps, forms, plastic etc. It is therefore possible to specify whether the drawing should be produced automatically as soon as there are time available on the chosen device, or if it should be stored for later manual activation.

The same holds for pictures which the user wants to control on a "quick-look" device before it is produced.

All instructions to the operator concerning a particular drawing such as pen switching, request for special forms etc. are written on the operator's console. The drawing production system then comes to a halt until the necessary action is performed and the operator restarts the drawing process. Each drawing will have inserted a standard heading which identify the user who produced the drawing and the drawing generation time which makes the delivery of the produced drawing to the customer fast and simple. A back-space facility is provided to avoid unnecessary redrawing if something unforeseen happens during the drawing procedure such as the pen dries out etc.

10.1.1. Drawing layout

DPS provides various automatic layout modes. The modes can be individually set for each output device:

Mode 0: Sequentiel (default):

In sequential mode the drawings are produced sequentially in the longitudinal direction of the "paper".

Fig. 10.1. Sequential mode
Mode 1: Optimal:

In optimal mode the drawings are produced such that an optimal utilization of the "paper" is achieved.

![Diagram](image1)

**Fig. 10.2. Optimal mode**

Mode 2: Formated:

Produce the drawings to fit onto a standard drawing format AB, scaled such that it fits within a rectangle given by the parameters C, D, E, F scaling should be the same in all directions to preserve shape.

![Diagram](image2)

**Fig. 10.3. Formated drawings.**

10.1.2. Drawing frame

If the frame mode is on (1), a solid frame (rectangle) will be drawn following the viewport boundary.

Default value for frame is off (0).

10.1.3. Viewport transformation

Viewport transformation indicates how the image to be drawn is scaled, rotated and translated when it appears on the view surface.
The image to be drawn is first scaled by $S_x$, $S_y$, then rotated an angle $A_z$ and finally shifted by $D_x$, $D_y$.

10.1.4. Number of copies

Automatic production of a certain number of copies for each drawing may be specified.

10.1.5. Logging capabilities

A Drawing Production System will usually administrate quite expensive devices such as big flat-bed plotters, colour plotters etc. which have to be charged for.

DPS is therefore providing extensive logging facilities for supporting invoicing, statistics and user support.

"Pen" movements and consumption of "paper" is logged for every drawing together with user identification and drawing generation information. This information may be used for automatic billing, statistics for device utilization, searching for missing drawings etc.

10.2. Picture editor

In order to save time and resources, a picture editor is provided. The picture editor has facilities both for "quick-look" and simple on-line editing of the pictures to be drawn.

The picture code is stored in DPS in device independent format which the editor can interpret and display on a screen for control and simple editing, even if the picture is generated for another device.

For more complicated picture manipulation, the image must be rerun through the IDIGS pipeline [23].

The user must have the opportunity to apply a window freely on the picture both for overview and for looking at details.

A facility for scrolling the picture is also provided.

Editing can only be done on drawings stored for manual initialization.
10.3. Operators command language

The system provides an operators command language which enables him to interact with the system in an efficient, flexible and simple manner. This includes:

- Defining and changing administrative data
- Defining and changing operational data
- Initiate plotting in various modes
- Stop (exit from) the system

The command syntax is designed to be easy to use and easy to remember. A DPS command has the form:

<OPERATOR><ARGUMENTS>

The <OPERATOR> is sometimes sufficient to form a command.

The commands are divided into three classes:

A. OPERATIVE commands:

- BEGIN Begin automatic plotting
- STOP Temporary stop
- CONTINUE Continue after stop
- MODE Give drawing mode
- BACKSPACE Backspace a given number of instructions
- TRANSFER Transfer a drawing from one device to another
- END Terminate automatic plotting

B. ADMINISTRATIVE commands:

- DEFINE Define administrative data
- CHANGE Change " --- ""
- DELETE Delete " --- ""
- INSPECT Display " --- ""

C. EDITING commands:

- EDIT Enter edit mode
- CHANGE Change Picture primitives and attributes
- INSERT Insert "- " "- "- 
- DELETE Delete "- " "- "- 
- WINDOW Set a window by lower left and upper right corner
- TRANSFORM Set translate, rotate and scale parameters for window transformation.
- PLOT Plots a given number of the picture on a given device
- OMIT Leave edit mode and disregard all editing
- EXIT Leave edit mode

Editing can only be performed on pictures which is marked "manuel".

The OPERATIVE commands are concerned with the control of the administrative data connected to the drawings and the drawing process and works on such parameters as: Account number, logging information, (time, distance, price etc.) priority, picture file tables etc.

The EDITING commands are concerned with the drawing itself and works on parameters like: Line, point, character, window coordinates, transformation parameters, plotting device, number of copies etc.

10.4. Technical specification

The basic graphics programs for DPS is IDIGS which is running with the device independent front end on the local user computer and the particular drivers on the graphical machine. However, the front end may also be running on the graphical machine enabling it to work stand alone.
The architecture given in fig. 10.4. is possible only if the graphics software is machine- and device independent (IDIGS) and the graphics network protocol is well defined and reasonably compressed. The last requirement may not be particular important because of the broad bandwidth optical communication links are providing. However, the communication link may for a long time be the telephone line.
10.4.1. The Application Computers

Drawings which are routed to remote devices from the Application computer are handled by the Network drivers. The Network drivers have a common core which are optimized for transport on the communication link.

The Network driver may be activated directly or indirectly by specifying a remote device.

Each driver contains tables with the necessary information of the corresponding device or mechanisms for inquiring this information by means of the IDIGS routine:

```
CALL INQOUT (OUTPUT-ARRAY)
```

which returns the output capabilities of the current device driver such as:

- raster capability
- colour capability
- intensity capability
- line width capability
- line style capability
- character capability (size, font, etc.)
- highlighting capability
- image transformation capability
- size of the view surface
- resolution of the view surface (e.g., number of addressable elements per centimeter for each dimension)
- number of marker symbols defined

If the drawing is to be started manually from the console, 100 is added to the actual device code in IDIGS.

Example:

```
CALL NITDEV (112)
```

The parameter 112 specify that the drawing should be stored until it is started from the console. Then the drawing will be produced on device number 12.

Manually initiated drawings is used in connection with editing, drawing on special forms, control of the drawing process etc.

When the Network driver is activated an IDIGS Metafile with one picture is created.

The metafile is used for long term storage of graphics information and for sequential transmitting of pictures from one site to another.
The Metafile consists of headings containing necessary information related to the picture and records of device independent code or non graphical user generated information. The File header consist of an identifier and a sequence of data related to picture generation. The Picture header consists of a sequence of administrative data.

```
METAFILE
| HEADER | PICTURE-1 | --- | PICTURE-N | EOI |
```

```
PICTURE
| HEADER | RECORD-1 | --- | RECORD N | EOF |
```

```
RECORD
| TYPE | LENGTH | DATA |
```

Each of the headings consist of 80 characters:

**10.4.1.1. Header information**

File header:

```
I  N  F  WL  M  E  D
```

The fields of the file header have the following meanings:

- **I(8=)** Metafile identifier: IDIGS-M
- **F(14)** File number
- **N(8=)** Name of the application computer
- **WL(i4)** Word length of the application computer
- **M(14)** Length of mantissa and exponent to be used for converting purposes
- **E(14)** Metafile coordinates and values
- **D(48=)** Reserved for future use

(= means alpha/numeric information and i means integer information. The accompanying number means the width of the field).

Picture header:

```
P  D  U  A  DA  R
```

The fields of the picture header have the following meanings:

- **P(i4)** Picture ID
- **D(i4)** Device number
- **U(8=)** User number
- **A(8=)** Account number
- **DA(24=)** Date and time of picture generation
- **R(36=)** Reserved for future use
10.4.1.2. Picture code

The picture code consist of compressed device independent IDIGS code as a stream of ASCII characters.

The general format of the picture code is as follows:

The bits are numbered from right (least significant = 0) to left. All numbers in the code are octal. The command header has 16 bits and two possible formats:

A.

```
  15   3   0
   MODIFIER  CMD
```

B.

```
  15   7   3   0
   MODIFIER  SUB  CMD
```

Fig. 10.5. Command headers

The necessary block of data follows immediately. The number of data words is dependent on the type of command.

Ex. 3D absolute line

```
  15   7   3   0
   MODIFIER  06  05
   X-coordinate
   Y-coordinate
   Z-coordinate
```

Fig. 10.6. Command header and data block
The coordinates are given as real numbers in the application computer dependent format (described in the file header).

However if the exponent or mantissa is equal to the previous one a bit in the command header is set and the exponent or mantissa is not transmitted. Negative numbers are transmitted as positive numbers with the sign bit set.

For strings the ASCII characters are sent as they are.

10.4.2. The graphics computer

A Real Time (RT) program in the graphics computer is continuously polling the communication links from the application computers.

When data is transmitted on one of the links, the RT program reads the File header to check if the first record is a Metafile identifier (IDIGS-M). If it is not a Metafile identifier, the data is ignored. Otherwise the File header is read and the parameters for data conversion etc. is set up and a program to receipt the Pictures is initiated.

This program reads the Picture header, sets up the administrative parameters, picks off all redundant information, converts the data to device independent code for the graphics computer and builds up a picture file which is identical to the standard IDIGS pseudosegments except for the heading with administrative data which is added (see fig. 10.7).
Fig. 10.7. Software in the graphics network.

The information in the Header of the Device independent picture code on the graphics computer is identical with the picture header that is transmitted.
Dependent on if the picture is to be drawn automatically or stored for control, editing and/or manual initiation from the console, the further treatment can be divided into two classes:

10.4.2.1. Automatic drawing

The drawings which are marked "automatic" are stored on a device specific queue (for instance N versions of a file or element) and a ringtable is established for controlling the entries in the queue.

![Ringtable for a device queue](image)

Fig. 10.8. Ringtable for a device queue

0 - The corresponding entry is available
1 - The corresponding entry is occupied

These tables are available both for the RT program and the drivers. When an "automatic" picture is transmitted from an Application Computer, the corresponding table is searched.

If a free entry is found (0 in the table) the entry is marked busy (1 in the table), the "file" is opened and the picture code is stored on the "file".

If the table is full, a message is given to the operator and the transmitting is stopped until space is available.
When the transmission of the picture code is terminated, the "file" is closed and a program for drawing the picture is activated.

Each drawing device or class of drawing devices are assigned a background process for quasi-parallell drawing on the available output devices.

Fig. 10.9. The background drawing process.

In order to make the system more efficient the spooling capabilities of the Graphic Computer can also be utilized.

10.4.2.2. Drawing started from the console

When the drawing is marked "manual", it is given a unique name composed from the user- and account identification and a serial number dependent on the number of drawings that particular user has stored for manual drawing initiation.
These drawings may now be retrieved and manipulated by the command language described in chapter 10.3, previewed or edited by the picture editor described in chapter 10.2 and routed to any of the output devices available for immediate drawing.

If any of the "automatic" drawings are to be manipulated or edited, it must first be transferred from the picture queue into the "manual" drawing area.

10.5. Experiences

DPS which was designed by myself and implemented by Jens Erik Torgersen, RUNIT which was a colleague of mine [40] has now been in active use at RUNIT for some time.

The experiences from the pilot phase is very positive seen both from the operators and the users point of view according to the reports RUNIT has received: The improvements reported (compared to the old system) are as follows:

- easier access to the devices from the users point of view
- a standard way of generating drawings
- faster and more reliable production of drawings
- simpler operation of the drawing devices
- better control of both the drawing process, and the system and device utilization
11. CONCLUSION

The Integrated Graphics Workstation (IGWS) is designed for fast and efficient data capturing in an interactive environment which is utilizing the human senses for communication in a natural way. The heavy drawing production is separated and concentrated at a central location with a permanent operator. In the future IGWS will most likely consist of a number of special purpose microcomputers controlled by what I will call an "Interaction Machine" for handling the different interaction devices and serve as a Simultaneous Interpreter between the user and the computer system. The computer systems will most probably be a distributed multiprocessor system.

IGWS can be envisioned as a flexible pilot's cockpit which is combining effectiveness and pleasure of work because it provides an efficient and natural way of utilizing the computer.

![Diagram of Interaction Machine]

*Fig. 11.1. The Interaction Machine*
References

(1) Several papers in "Progress in Psychobiology". Readings from Scientific American 1975.


(4) Computer Vision, Systems manuals 201 Burlington Road, Route 62, Bedford, Massachusetts 01730.

(5) Auto-Trol, Systems manuals 5650 North Pecos St, Denver Colorado 80221.

(6) Tektronix, System manuals, Information Display Division, P.O.Box 500 Beaverton, Oregon 97005.


Tektronix-4954 System manual. P.O. Box 500, Beaverton, Oregon 97008.


Sutherland, I.E.: "Three Dimensional Data input by Tablet" Proceeding of the IEEE, April 1974.


(57) "NOTIS-1". Brukerhåndbok, Norsk Data 1980.

(58) "ALTO: A Personel Computer System." XEROX, Palo Alto Research Centre, 333 Coyote Hil Road, Palo Alto, California 94304.


(60) VERSATEC Graphics Programming Manual. 10100 Bubb Road Cupertino California 95014.

PROPOSAL OF GRAPHICS STANDARD

IDIGS

INTERACTIVE DEVICE INDEPENDENT
GRAPHIC SYSTEM

BY

KETIL BØ

RUNIT
7034 TRONDHEIM-NTH
NORWAY

- VERSION 3.0 -

NORSIGD
PREFACE

IDIGS (Interactive Device Independent System) is the Norwegian proposal for a 3-dimensional international standard.

The proposal is strongly influenced by GPGS-F [1] which is the preliminary norwegian graphics standard, the GSPC "Core" [2] which is the ACM/SIGGRAPH Graphics recommendation, GKS [3] which is the DIN proposal for a 2-dimensional international standard and a number of discussions with colleagues both national and international.

Trondheim, December 1980

Ketil Bø
# CONTENTS

## PREFACE

**CONTENTS**

### 1. INTERACTIVE DEVICE INDEPENDENT GRAPHICS SYSTEM (IDIGS)

- 1.1. Portability  
- 1.2. The modular Model  
  - 1.2.1. Modularity  
  - 1.2.2. Device Independence  
  - 1.2.3. Machine Independent  
  - 1.2.4. Requirement Independent

### 2. ROUTINE NAMING CONVENTION

- 2.1. The action routines  
- 2.2. The attribute routines

### 3. OUTPUT PRIMITIVES

- 3.1. Move  
- 3.2. Line  
- 3.3 Polylin  
- 3.4. Polygon  
- 3.5. Draw  
- 3.6. Marker  
- 3.7. Text

### 4. ATTRIBUTES

- 4.1. Primitive attributes  
  - 4.1.1. Line style  
  - 4.1.2. Linewidth  
  - 4.1.3. Intensity  
  - 4.1.4. Colour  
  - 4.1.5. Marker symbol size  
  - 4.1.6. Tool  
  - 4.1.7. Pick-id  
  - 4.1.8. Font  
  - 4.1.9. Text quality
4.1.10. Character spacing in 2D 20
4.1.11. Character spacing in 3D 21
4.1.12. Character size 21
4.1.13. Character plane 22
4.1.14. Character-path 22
4.1.15. Character-justification 23
4.1.16. Save attributes 25
4.1.17. Set attributes 25

5. PICTURE SEGMENTATION AND NAMING 26
  5.1. Overview 26
  5.1.1. Begin segment 27
  5.1.2. Close segment 28
  5.1.3. Delete segment 28
  5.1.4. Rename 29

6. SEGMENT ATTRIBUTES 30
  6.1. Overview 30
  6.1.1. Visibility 30
  6.1.2. Highlighting 30
  6.1.3. Detectability 31
  6.1.4. Image-transformation 31

7. VIEWING 33
  7.1. Overview 33
  7.2. 3D viewplan transformation 37
  7.2.1. View reference point 37
  7.2.2. View plane normal 37
  7.2.3. View plane distance 37
  7.2.4. Parallel projection 38
  7.2.5. Perspective projection 38
  7.2.6. View up in 3 dimension (3D) 38
  7.2.7. Window in 3D 39
  7.2.8. View depth 40
  7.3. 2D viewplane transformations 40
  7.3.1. View up in 2 dimension (2D) 41
  7.3.2. Window in 2D 41
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4. 3D NDC transformation</td>
<td>42</td>
</tr>
<tr>
<td>7.4.1. Viewport in 3D NDC</td>
<td>42</td>
</tr>
<tr>
<td>7.4.2. Viewport in 3D NDC</td>
<td>43</td>
</tr>
<tr>
<td>7.5. Workstation transformation</td>
<td>44</td>
</tr>
<tr>
<td>7.5.1. Workstation window in 3D</td>
<td>45</td>
</tr>
<tr>
<td>7.5.2. Workstation viewport in 3D</td>
<td>45</td>
</tr>
<tr>
<td>7.5.3. Workstation window in 2D</td>
<td>46</td>
</tr>
<tr>
<td>7.5.4. Workstation viewport in 2D</td>
<td>46</td>
</tr>
<tr>
<td>7.6. Model transformations</td>
<td>46</td>
</tr>
<tr>
<td>7.6.1. Model transformation in 2D</td>
<td>47</td>
</tr>
<tr>
<td>7.6.2. Model transformation in 3D</td>
<td>47</td>
</tr>
<tr>
<td>7.7. Default values</td>
<td>47</td>
</tr>
<tr>
<td>7.8. Polygons in the viewing system</td>
<td>48</td>
</tr>
<tr>
<td>8. INPUT PRIMITIVES</td>
<td>51</td>
</tr>
<tr>
<td>8.1. Overview</td>
<td>51</td>
</tr>
<tr>
<td>8.2. Functional Capabilities</td>
<td>53</td>
</tr>
<tr>
<td>8.2.1. Logical Input Facilities</td>
<td>53</td>
</tr>
<tr>
<td>8.2.2. Event devices</td>
<td>54</td>
</tr>
<tr>
<td>8.2.2.1 Pick</td>
<td>54</td>
</tr>
<tr>
<td>8.2.2.2. Text</td>
<td>55</td>
</tr>
<tr>
<td>8.2.2.3 Choose</td>
<td>55</td>
</tr>
<tr>
<td>8.2.2.4 Distance</td>
<td>56</td>
</tr>
<tr>
<td>8.2.2.5 Timer</td>
<td>56</td>
</tr>
<tr>
<td>8.2.3. Sampled devices</td>
<td>56</td>
</tr>
<tr>
<td>8.2.3.1. Locator</td>
<td>56</td>
</tr>
<tr>
<td>8.2.3.2. Valuator</td>
<td>57</td>
</tr>
<tr>
<td>8.3. Access to the physical Input Devices</td>
<td>57</td>
</tr>
<tr>
<td>8.3.1. Attach device</td>
<td>58</td>
</tr>
<tr>
<td>8.3.2. Detach device</td>
<td>58</td>
</tr>
<tr>
<td>8.4. Input levels</td>
<td>58</td>
</tr>
<tr>
<td>8.4.1. Request input</td>
<td>58</td>
</tr>
<tr>
<td>8.4.1.1. Request PICK data</td>
<td>58</td>
</tr>
<tr>
<td>8.4.1.2. Request TEXT data</td>
<td>59</td>
</tr>
<tr>
<td>8.4.1.3. Request CHOISE data</td>
<td>59</td>
</tr>
<tr>
<td>8.4.1.4. Request LOCATOR data</td>
<td>60</td>
</tr>
<tr>
<td>8.4.1.5. Request VALUATOR data</td>
<td>60</td>
</tr>
</tbody>
</table>
8.4.2. Input based on event queue 60
  8.4.2.1. Await event 61
8.4.3. Compound input devices 62
  8.4.3.1. Associate devices 62
  8.4.3.2. Disassociate device 63
  8.4.3.3. Disassociate all devices 64
8.5. Reading sampled devices 64
  8.5.1. Read locator 64
  8.5.2. Read valuator 64
8.6. Input device control 65
  8.6.1. Enable device 65
  8.6.2. Disable device 65
  8.6.3. Flush device 65
  8.6.4. Flush all devices 66
8.7. Accessing event report data 66
  8.7.1. Get PICK data 67
  8.7.2. Get TEXT data 67
  8.7.3. Get CHOISE Data 68
  8.7.4. Get LOCATOR Data 68
  8.7.5. Get VALUATOR Data 69
8.8. Setting input device characteristics 69
  8.8.1. Set Echo 69
  8.8.2. Set Echo Position 70
  8.8.3. Set Echo Surface 70
  8.8.4. Set PICK 71
  8.8.5. Set LOCATOR 71
  8.8.6. Set VALUATOR 72
  8.8.7. Set CHOISE 72

9. CONTROL 73
  9.1. Overview 73
  9.2. System Initialization and Termination 73
    9.2.1. Initialization of IDIGS 73
    9.2.2. Termination of IDIGS 74
  9.3. Device initialization and selection 74
    9.3.1. Initialize Device 75
    9.3.2. Select device 75
9.3.3. Deselect device 76
9.3.4. Release device 76
9.3.5. Select background device 77
9.3.6. Deselect background device 77
9.4. Device Control 77
9.4.1. Clear device 77
9.4.2. Special control function 78
9.4.2.1. Begin special function 78
9.4.2.2. End Special Function 79
9.4.2.3. Enable hidden lines removal 79
9.4.2.4. Enable hidden surface removal 80
9.4.3. The R, B, G color model 80
9.4.3.1. Define Color Index table 81
9.4.4. Define Intensity Index table 82
9.4.5. Define Hatching Index table 82
9.4.6. Define Workstation pen Index table 82
9.4.7. Send message to workstation 82
9.5. Viewing Control 83
9.5.1. Clipping control 83
9.6. Polygon Control 84
9.6.1. Set Polygon mode 85
9.6.2. Set PLAIN Polygon filling index 86
9.6.3. Set SHADEx polygon vertex index 87
9.6.4. Set PATTERNED polygon pixel array 87

10. STORAGE OF GRAPHICAL INFORMATION 89
10.1. Overview 89
10.2. Buffer handling 91
10.2.1. Initialize primary buffer 91
10.2.2. Select Buffer 92
10.2.3. Release Buffer 92
10.3. Secondary Storage Buffers (Library) 92
10.3.1. Initialization of library 93
10.3.2. Select library 93
10.3.3. Release library 93
10.4. Library handling 94
10.4.1. Save picture segments in the library 94
10.4.2. Restore picture segment from the library 94
10.5. Pseudo Picture Segments 95
10.5.1. Insert meta segment from buffer 95
10.5.2. Insert meta segment from library 95
10.5.3. Copy a frame from library 96
10.5.4. Reference to meta segments 96
10.6. Metafile handling 97
10.6.1. Initialize the metafile 97
10.6.2. Select meta file 97
10.6.3. Release meta file 98
10.6.4. Write user information to the meta file 98
10.6.5. Read Record type from the meta file 98
10.6.6. Read next record from meta file 99
10.6.7. Read and interpret next record from the meta file 99
10.6.8. Skip next N records on the meta file 99
10.6.9. Backspace N records on the meta file 100
10.7. Subpicture 100
10.7.1. Begin subpicture 101
10.7.2. End subpicture 101
10.7.3. Delete subpicture 101
10.7.4. Call subpicture 102

11. INQUIRY 103
11.1. Overview 103
11.2. Inquire Current Position 103
11.3. Inquire Viewing Parameters 103
11.3.1. Inquire Viewing Transformation 104
11.3.2. Inquire Viewing Position 104
11.3.3. Inquire Projection Parameters 105
11.4. Inquire max. surface area 105
11.5. Inquire segment attributes 106
11.6. Inquire Output Status 106
11.7. Inquire Input Device Information 107
11.7.1. Inquire Echo parameters 107
11.7.2. Inquire Device Association 107
11.7.3. Inquire Device Enabled 107
11.8. Inquire Output Capabilities 108
11.8.1. Inquire Input Capabilities 108
11.9. Inquire Library content 109
11.9.1. Inquire Library-Segment generation environment 109
11.10. Inquire Raster 109
11.10.1. Inquire Pixel Array 109
11.10.2. Inquire Polygon Mode 110

12. ERROR HANDLING 111
12.1. Users error routine 112
12.2. Users own error messages 113
12.3. Initialize error file 113
12.4. Report most recent error 114

REFERENCES 115

APPENDIX A - Alfabetlic list of routines in IDIGS

APPENDIX B - Errors
1. INTERACTIVE DEVICE INDEPENDENT GRAPHICS SYSTEM (IDIGS)

1.1. Portability

The strongest justification for the development of a standard is the promotion of program portability, i.e., the ability to transport graphics applications from one installation to another with minimal program changes.

As a fundamental strategy for achieving portability, IDIGS provides features which shield the application programmer from specific hardware characteristics. This shielding is at the functional level and need not imply expensive software overhead. The programmer describes his graphical world to the System in device-independent world coordinates. He also specifies, in normalized device coordinates, where on the available logical view surfaces the view of his object is to be placed.

Similarly, he can specify which logical input devices he wishes the operator to use at what time during the execution of the application program, without concern for the hardware-dependent protocol of actual devices. The mapping from logical input devices and output devices (view surfaces) to the physical devices for a specific configuration is handled by the System implementation for that configuration, and need not concern the application programmer. Also, while the physical input devices of a display system are associated with the physical output devices of the system, this need not be true for logical input devices and view surfaces.

Another strong justification is the user portability which means a considerable saving in education effort for users moving from one site to another.

1.2. The modular Model

IDIGS consists of a set of procedures and data comprising
the basis functions required for the programming of graphical
devices and for their respective management.

The definition of IDIGS considers the following focal points:

1.2.1. Modularity

IDIGS is to be modular. This has been achieved by dividing
the system both vertically and horizontally in modules with
well defined interfaces.

The IDIGS is a menu of modules with a pick and choose
strategy for building up the graphics system.

The modules are chosen partly at system generation time
and partly at load time. (Fig. 1)

The major reasons for this approach are:

1. Make programming easier for the user (ex. 2D,
   2 1/2 D, 3D or combined).

2. Make the system available also for special purpose
   applications with limited space available.

3. Prepare for firmware or microcomputer implementation
   of particular modules.

4. Prepare for running the system on Computer Networks

5. Easy extention of the system.
Fig. 1. MODULAR GRAPHICS SYSTEM
But, in order to avoid arbitrary combination of capabilities and thus introduce an almost unlimited number of different standard dialects which could result in making program portability difficult, only a subset are defined as standard combinations of the IDIGS system. (See fig. 2)

However, if the programs are not planned to be moved to other installations, any useful combination of capabilities are possible.

The valid subsets are made up from the entries in a five dimensional feature space where the axis represent INPUT features, OUTPUT features, DIMENSIONALITY, TYPE and active WORKSTATIONS with increasing capabilities along the axis in such a way that the lower level on each axis is a subset of the higher level on the same axis.

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
<th>DIRECT INPUT</th>
<th>COMPLETE INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NONE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DIRECT INPUT</td>
<td></td>
<td>COMPLETE INPUT</td>
</tr>
<tr>
<td></td>
<td>QUEUE INPUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BASIC</td>
<td>INITIALIZATION</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VIRTUAL SEGMENT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WINDOW/VIEWPORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CLIPPING</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ONE DRIVER</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2D/3D-CALLIGRAPHIC/RASTER</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BUFFERED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RETAINED SEGMENT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STORED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PSEUDO BUFFER</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PSEUDOLIBRARY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>META FILE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>SIB PICTURE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IMAGE TRANSFORMATION</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 2. Standard combinations of IDIGS.
Next two axis have only two values each:

**DIMENSIONALITY** = 2D or 3D
**TYPE** = CALLIGRAPHIC only or RASTERGRAPHICS added

The Active Workstation axis may have three values:

One active, no background workstations
One active, multiple background
Multiple active, multiple background

(An active workstation may handle both output and input, a background workstation handle only output.)

The level of a particular IDIGS system is thus given by a five dimensional vector: \((I, J, K, L, M)\)

where:
- \(I\) denotes the output capability \(I = 1, 2, 3, 4\)
- \(J\) denotes the input capability \(J = 1, 2, 3\)
- \(K\) denotes the dimensionality \(K = 1, 2\)
- \(L\) denotes the type \(L = 1, 2\)
- \(M\) is the number of active workstations \(M = 1, 2, 3\)

1.2.2. Device Independence

By a device-independent graphics software system (package) we mean a system that can be used in conjunction with two or more different graphical devices (such as two different displays with different graphics input devices) without the need to modify the application program that use the system. (The performance of the system may of course be degraded when it is moved to a new configuration, but it should work.)

IDIGS is to be device independent. This will be achieved by definition of standard reactions for all devices which can provide a certain function and standard error reactions for all devices which cannot provide this function. In order to allow effective programming for the different graphical output devices, IDIGS contains functions by which device
oriented data from the workstations can be requested.

A workstation is the operators work place that includes a display device and/or also one or more input devices such as an alphanumerical keyboard, function keys, joy stick, light pen or tablet.

A fully equipped workstation:

a) has one addressable display surface;
b) permits the use of smaller display spaces that the maximum while guaranteeing that no display is generated outside the specified display space;
c) supports several linestyles, text fonts, character sizes, etc;
d) has one or more input devices for each class of input primitive;
e) permits request type, event type, and sample type input.

In actual installations the workstation may or may not be equipped with all of these capabilities. For example, the input capabilities of a workstation may range from no input to several input devices of every class. The application program may inquire which capabilities are available and adapt its behaviour accordingly. If capabilities are requested which a particular workstation does not provide, a standard error reaction is defined.

1.2.3. Machine Independent

By machine-independence we mean the ability of a software system to run on a variety of different computers, because it may be accessed from commonly available machine independent programming languages.

IDIGS is to be machine independent in the sense that no structural modifications of the program is necessary when moved from one machine to another. This implies, also that its functions, at the user interface, are supported by most operating systems and do not require expansions or modifications of operating systems.
1.2.4. Requirement Independent

IDIGS is to be independent of the requirements of specific applicational fields (e.g. cartography, computer aided design). Such application oriented systems may be built on top of the basic IDIGS system.

2. Routine naming convention

In general will the names of the routines will follow the FORTRAN standard with the following conventions:

2.1. The action routines

The names of the subroutines are divided in two main parts:

OPERATION OPERATOR

in "free" format.

That means both the OPERATION and the OPERATOR may be from 0-6 characters. If the OPERATION appear in the name, it will be the first part. The OPERATOR may be given some postfixed attributes such as addressing, argument type and dimension.

Default values for this attributes are absolute address, floating point in 2 dimensions.

The postfix:

R: means relative addressing

3: means 3-dimension

2.2. The attribute routines

The names of the attribute routines are describing the FUNCTION which is performed by the particular routine.
3. OUTPUT PRIMITIVES

The graphical world which the programmer describes to IDIGS consists of one or more objects. Each graphical object is described in world coordinates by invocations of two- or three-dimensional output primitive functions. The primitive functions in IDIGS are: draw, polygon, polyline, line, move, polynmarker, (to designate points on plots) and text strings. An invocation of an output primitive function results in an output primitive. The appearance of output primitives is affected by the values of the primitive attributes of linestyle, linewidth, intensity, colour, character font, character size, character spacing, character quality, and character plane. Each output primitive may also have an associated name, called a PICK-ID, for selection purposes.

To achieve a drawing in correct scale, the metric equivalent of the normalized coordinates can be inquired from the attached work station thus allowing a proper setting of the viewport.

Graphical output functions may be called only when a segment is open.

The graphical output primitives are:

3.1. Move

MOVE(X,Y)
MOVER(DX,DY)
MOVE3(X,Y,Z)
MOVER3(DX,DY,DZ)

The CP is set to the value \((X,Y,(Z))\) where \((X,Y,(Z))\) is a position in the world coordinate system.

Errors:
1. No segment is open.
3.2. Line

\begin{align*}
\text{LINE}(X,Y) \\
\text{LINER}(DX,DY) \\
\text{LINE3}(X,Y,Z) \\
\text{LINER3}(DX,DY,DZ)
\end{align*}

This function is used to describe a line of an object in world coordinates. This line runs from CP to the position specified. The primitive attributes of COLOUR, INTENSITY, LINEWIDTH, LINESTYLE, and PICK-ID are meaningful for lines. The CP is updated according to the arguments of the function involved.

Errors:
1. No segment is open.

3.3. Polyline

\begin{align*}
\text{PLIN}(X-\text{ARR},Y-\text{ARR},N) \\
\text{PLINR}(DX-\text{ARR},DY-\text{ARR},N) \\
\text{PLIN3}(X-\text{ARR},Y-\text{ARR},Z-\text{ARR},N) \\
\text{PLINR3}(DX-\text{ARR},DY-\text{ARR},DZ-\text{ARR},N)
\end{align*}

These functions are used to draw an array of lines, with the endpoints given in world-coordinates. The first line starts at the first coordinate in the array (i.e. an implicit move is included), and lines are drawn sequentially to the following coordinates.

3.4. Polygon

\begin{align*}
\text{POLY} (X-\text{ARR}, Y-\text{ARR}, N) \\
\text{POLYR} (DX-\text{ARR}, DY-\text{ARR}, N) \\
\text{POLY3} (X-\text{ARR}, Y-\text{ARR}, Z-\text{ARR}, N) \\
\text{POLYR3} (DX-\text{ARR}, DY-\text{ARR}, DZ-\text{ARR}, N)
\end{align*}
This function is used to define a closed polygon with a vertex list in world coordinates. The polygon is the region enclosed by the connected sequence of lines that starts at the first vertex in the array, joins all the other vertices in order, then returns to the first vertex to close the defined area. Because closure is implied, the CP is set to the coordinates of the first vertex at the completion of the primitive.

The interior of the polygon will depend on the current polygon type and the corresponding set of primitive attributes.

Errors:
1. No segment is open
2. N < 2

3.5. Draw

\[
\text{DRAW} \ (X-\text{ARR}, Y-\text{ARR}, N, \text{ITYPE})
\]
\[
\text{DRAWR} \ (DX-\text{ARR}, DY-\text{ARR}, N, \text{ITYPE})
\]
\[
\text{DRAW3} \ (X-\text{ARR}, Y-\text{ARR}, Z-\text{ARR}, N, \text{ITYPE})
\]
\[
\text{DRAWR3} \ (DX-\text{ARR}, DY-\text{ARR}, DZ-\text{ARR}, N, \text{ITYPE})
\]

A generalized primitive of the type indicated by the parameter ITYPE is generated on the basis of the data in the arrays. The routines start drawing at the position specified by the parameters in the function call. N is the number of entries in the arrays.

The generalized primitives are to be specified in a separate standard. At present the following primitives are anticipated:

- interpolating curve (points)
- circle, (circle parameters)
- circular arc (arc parameters)
- ellipse (ellipse parameters)
- elliptic arc (elliptic arc parameters)

Errors:
1. No segment is open
2. ITYPE has not a valid value.

3.6. Marker

MARK(NR,X-ARRAY,Y-ARRAY,N)
MARKR(NR,DX-ARRAY,DY-ARRAY,N)
MARK3(NR,X-ARRAY,Y-ARRAY,Z-ARRAY,N)
MARKR3(NR,DX-ARRAY,DY-ARRAY,DZ-ARRAY,N)

These functions draw N system defined markers NR, centred at the positions given by the coordinate arrays.

A limited number of standard markers (number 1 through N1) will be included in the specification of IDIGS. Another set of numbers (from N1+1 to N2) are set aside for expansion of the set of standard markers. All marker-numbers greater than N2 and less than 32767 will be legal and set aside for installation-defined markers. Calls for unsupported markers will result in a warning message and a default marker defined by each installation will be displayed.

The effect of clipping on the marker is that the marker will not be drawn if part of the marker box is outside the clipping box.

MARKERS are just a means by which points uniquely manifest themselves on the output display surface. Consequently, "point plotting" is done by plotting (for example dot or bullet shaped markers) rather than by using a POINT primitive.

NR - Is is a discrete scalar ranging from 0 to 32767.

The values 0-9 will produce the marker symbols in the following table. The values above 9 produce implementation-dependent marker symbols. The default marker symbol value is a dot.
0 = dot
1 = plus
2 = star (x and +)
3 = circle (or octagon)
4 = cross
5 = square
6 = diamond
7 = nabla
8 = delta
9 = plus in a square

Errors:
1. No segment is open
2. N < 0.

3.7. Text

CHARS (STRING, LENGTH)

A character string in the actual text font is generated. The lower left corner of the first character box coincides with the starting point. The text quality orientation, size and spacing are determined by the current values of the character attributes.

LENGTH is a parameter specifying the STRING length in characters. LENGTH < 0 means that the STRING is terminated with a terminator (.*).

Errors:
1. No segment is open.
2. The string contains an illegal character.
4. ATTRIBUTES

The purpose of attributes in IDIGS is to specify general characteristics for segments and output primitives: these characteristics are represented by the attribute values of the attributes. For example, the LINTYP attribute is a characteristic of the LINE primitive and has values such as "solid" and "dashed". Similarly the VISPIC attribute is a characteristic of a segment and has values "visible" or "invisible". There are two orthogonal partitionings of the set of attributes into disjoint subsets, as explained below. The first partitioning is based on whether an attribute applies to a segment or to an output primitive. To prevent conflicts and avoid complexity, a single attribute cannot apply to both. The second partitioning is based on whether an attribute specifies a static or a dynamic (changeable) characteristic.

Segment attributes specify characteristics of segments. Every segment is created with an attribute value for each of the segment attributes. Primitive attributes specify general characteristics of output primitives. Some primitive attributes are meaningful for more than one type of output primitive, but most are not meaningful for all types of output primitives. Every individual output primitive is created with an attribute value for each primitive attribute that is meaningful for that primitive.

Static attributes specify immutable characteristics of individual segments or primitives and are fixed throughout the lifetimes of those entities. Dynamic attributes specify characteristics of entities that can be altered during the lifetimes of those entities. IDIGS includes static primitive attributes and a single static segment attribute. It also includes dynamic segment attributes, but does not include any dynamic primitive attributes because alteration of primitives within a segment is not supported (the segment is the unit of modification). An application program can only change a given attribute value of a particular primitive by recreating
the segment that contains the primitive and generating a similar primitive using a different attribute value.

However, intensity and colour are semidynamic attributes because even if the given index to the intensity/colour look up table is static, the look up table itself can be dynamically changed and thus create sequential (typical plotter devices) or retroactive (typical refresh devices) changes in the intensity/colours without regenerating the segment.

A third classification of attributes can be made, but does not result in a clear-cut partitioning. Some attributes define characteristics of objects in their object space, such as the orientation of text. Other attributes define characteristics related to the views of objects (images), such as the style and width of line primitives. (Thus foreshortening of "dashed" segments within a line will not occur in a perspective view of the line.) Other attributes, such as COLOUR, can be interpreted either as characteristics of the conceptual object, or of its view. Identical behavior results from either interpretation. Only those attributes having different behaviors depending upon their classification must be formally identified. CSIZE, CSPACE, CPATH, CPLANE and CJUST are all defined in terms of conceptual text objects; LINTYP, and LINWID are only related to the views of objects.

IDIGS will provide a standard set of default attribute values for both static and dynamic attributes. IDIGS also maintains a set of current attribute values. These values are initialized to the corresponding default values, but can be explicitly altered by the program. When a primitive or segment is created, it is automatically assigned current attribute values for all of its static attributes. Typically several primitives or several segments will be created between changes to one or more of the current values, with their static attribute values specified modally. IDIGS provides also a mechanism for stacking and destacking of attribute values.
4.1. Primitive attributes

4.1.1. Line style

LINTYP (N)

This is a discrete scalar attribute, with values ranging from 0 through an implementation-determined maximum. IDIGS implementations must support a maximum of at least 9999. The single-digit values 0-9 are reserved for hardware-generated line styles, and the multi-digit values are interpreted, with each digit dictating the appearance of a portion of the line, as the table shown:

| DIGIT VALUE | LINE SEGMENT
|-------------|-------------
| 0,1         | point       | (invis., vis.) |
| 2,3         | very short  | (invis., vis.) |
| 4,5         | short       | (invis., vis.) |
| 6,7         | medium      | (invis., vis.) |
| 8,9         | long        | (invis., vis.) |

Dash and space lengths are based on device dimensions, and are not scaled by viewing transformations. The default value for line style is 0, representing a solid line.

4.1.2. Linewidth

LINWID (IWIDTH)

Linewidth indicates the relative width of the image of a visible line (does not apply to Text and Markers).

IWIDTH is a continuous scalar, ranging from 0 (narrowest line possible) to 1, representing a percentage of the smallest device dimension (width or height of NDC space). The maximum line width is 1% of the smallest device dimension; a IDIGS implementation need not provide more than one width. The default IWIDTH is 0. Values may be mapped to the nearest
available hardware line width.

4.1.3. Intensity

INTENS (INDEX)

INDEX is an entry into a table describing the intensity which is a continuous scalar, ranging from 0 (device minimum intensity) to 1 (device maximum intensity). The default INDEX correspond to intensity 0.5.

The intensity range is to be divided into equal divisions, based on the number of intensities available for the display device; intensity values are rounded down to the next lower available setting.

4.1.4. Colour

COLOUR (INDEX) 1 ≤ INDEX ≤ N

The colour attribute for output primitives is specified as an INDEX to a table consisting of a ordered set of tripples \( (R_i, G_i, B_i) \) in the Red, Green, Blue colour system ranging from 0-1.

This tripple table is called the active colour set. The entries in the colour table may be changed (see ch. 9). This concept can be extended for use with colour look-up tables.

4.1.5. Marker symbol size

MSIZE (M-UP, M-WIDTH)

The marker symbol size is given in world coordinates.

The M-UP and M-WIDTH are 2 dimensional vectors describing a parallelogram into which the marker symbol is to be placed with the marker position at the center of the parallelogram. These values are used for creating subsequent polynmaker
output primitives.

The M-UP and M-WIDTH vectors are transformed by all applicable transformations.

4.1.6. Tool

TOOL (TOOL-NUMBER)  \(+\ N1 \leq\ TOOL-NUMBER \leq\ N2\)

Indicates a table driven logical and/or physical tool selection mechanism.
Entries in the table is workstation defined. For each workstation two tool table exists.

A positive TOOL-NUMBER indicates an entry in a table which specify a set of workstation attributes (Clour, Intensity, Linestyle and Linewidth) which is used instead of the current line attributes (A logical pen). Entries in this table may be redefined by the DEFTOL routine.

A negative TOOL-NUMBER indicates an entry in a table that defines workstation dependent physical tools. This results in the loading of knives, photoheads etc. Entries in this table are installation defined and may not be modified.

4.1.7. Pick-id

NAME (N)  \(0 \leq N \leq 32767\)

N indicates the name of a primitive which is returned for use by the application program whenever the primitive is selected by the operator using a PICK input device.

4.1.8. Font

CFONT (IFONT)

Used for selecting the style of a visible characters

IFONT is the number of the text font:
IFONT = 0 Standard
IFONT = 1 Roman
IFONT = 2 Italic
IFONT = 3 Greek
IFONT = 4 Cyrillic
IFONT = 5 Arabic
IFONT = 10 Norwegian

Further text fonts are implementation dependent. For not
defined text fonts the default value is 0 (standard).
"Standard" is an implementation dependent Latin text font.

Font values only apply to Text primitives.

Errors:
1. The font is invalid

4.1.9. Text quality

CQUAL (IQUAL)

Indicates the quality of the text string.

IQUAL = 0 Low quality (string precision)
IQUAL = 1 Medium quality (character precision)
IQUAL = 2 High quality (stroke precision)

Low quality = 0 - The effect of the character attributes
on the appearance of the string is
implementation-dependent. CP and the
current value of CJUST are used to deter-
mine the placement of the text, string,
and CHARSIZE, after being scaled by
the viewing operation, is used to select
a "best-fit" among available hardware
character sizes. However, the values
of character attributes CPLANE, CPATH
and CSPACE may be ignored.
Medium quality = 1 - The placement of individual character
corresponds to the character attributes,
but the orientation of individual
characters is implementation-dependent.
The value of CSIZE, after being scaled
by the viewing operation, is used to
select a "best-fit" among available
hardware character sizes. The position
of individual characters in world
coordinates is transformed by the current
viewing operation. It is implementation-
dependent whether perspective projection
of the characters is implemented.

High quality = 2 - All character attribute specifications
are strictly applied. In general, "high
quality" text must be displayed just
as if each character were made up of
short lines (strokes). The strokes
comprising the character are subject
to the current viewing operation.

Default value is low quality (0).

4.1.10. Character spacing in 2D

CSPACE (XSPACE, YSPACE)

Indicates the desired spacing in world coordinate units
between adjacent characters in a text string. (The
"Character box") XSPACE and YSPACE gives also the direction
of the text on the character plane (see fig. 4.1). CSPACE
values only apply to Text primitives.
Fig. 4.1. Result of CSIZE and CSPACE.

Default value is \((0.01, 0)\) which gives 100 character boxes along a default window.

4.1.11. Character spacing in 3D

CSPAC3 \((XSPACE, YSPACE, ZSPACE)\)

The primitive attribute CSPAC3 defines a vector relative to CP in the world coordinate system which gives both the direction and the desired spacing between adjacent characters in a text string.

This direction and size vector together with CPATH and CP defines the world coordinate system plane and how the characters appear on that plane.

Default value is \((0.01, 0, 0)\) which gives 100 character boxes along a default window.

4.1.12. Character size

CSIZE \((XSIZE, YSIZE)\)

CSIZE indicates the desired size, in world coordinate units,
of a character. CSIZE has components for both width and height, and thus also determines the aspect ratio of a square in font coordinates. For low- and medium quality text, after the viewing operation maps the desired size of a character to normalized device coordinate units, the system will select the largest hardware character size that does not exceed the desired size. (If all available sized exceed the desired size, then something visible must appear, but it is implementation-dependent exactly what technique is used -- be it to display a special marker, to choose the smallest character size, etc.) For high quality text, a font designer defined value in font coordinates is scaled to the width and height components of CSIZE in world coordinates. CSIZE values only apply to TEXT primitives.

Default value is (0.01,0.01) which will give 100 lines a 100 characters in a default window.

4.1.13. Character plane

CPLANE (DX, DY, DZ)

defines a vector relative to the CP in the world coordinate system. This vector and the string spacing vector establish the world coordinate system plane in which three-dimensional text characters are created and the placement of the character boxes in this plane. The character plane is normal to CPLANE and passes through CP. CPLANE value applies only to Text primitives. Default value is (0,0,-1).

4.1.14. Character-path

CPATH (IDIR)

The primitive attribute CPATH is used in determining the string direction. CPATH specifies the position of the character box relative to CP and the position of CP afterwards. The character box is positioned adjacent to CP in the direction specified by IDIR. The valid values of IDIR and their effects are:
0 = right - Initially, the character box is positioned such that CP lies on the intersection of the horizontal axis with the left edge of the character box. Afterwards, CP is moved along the horizontal axis to the right edge of the character box.

1 = up - Initially, the character box is positioned such that CP lies on the intersection of the vertical axis with the bottom edge of the character box. Afterwards, CP is moved along the vertical axis to the top edge of the character box.

2 = left - Initially, the character box is positioned such that CP lies on the intersection of the horizontal axis with the right edge of the character box. Afterwards, CP is moved along the horizontal axis to the left edge of the character box.

3 = down - Initially, the character box is positioned such that CP lies on the intersection of the vertical axis with the top edge of the character box. Afterwards, CP is moved along the vertical axis to the bottom edge of the character box.

If necessary, the edges of the character box are extended to the axes.

0 = right is default.

4.1.15. Character-justification

CJUST (ITYPE)

The primitive attribute CJUST is used to specify the mode of string justification. A string extent vector is defined by the current values of CSPACE, CPLANE, CSIZE, CPATH, and the characters from the current font specified in the string. This vector connects the CPs before and after a text primitive before the effects of string justification. ITYPE specifies
a dislocation of this vector longitudinally with respect to the starting CP. The valid values of ITYPE and their effects are:

ITYPE = 0 = begin - The beginning of the string extent positioned at the CP; CP is updated to coincide with the end of the string extent. (This is the default.)

ITYPE = 1 = end - The position of the beginning of the string extent is obtained by subtracting the string extent vector from CP. There is no net effect on CP by the text primitive.

ITYPE = 2 = center - The position of the beginning of the string extent is obtained by subtracting one half of the string extent vector from CP. CP is updated to coincide with the end of the string extent. (This corresponds to adding one half of the string extent vector to CP.)

---

a) begin (0)  c) end (1)  b) center (2)

Fig. 4.2. Character justification

Note that in all cases CP is updated to the so-called "concatenation position" so that another text primitives following immediately.

Default value is 0 (begin)

Errors for all attributes:
1. Invocation when no open segment
2. Invalid attribute value

4.1.16. Save attributes

SAVATR (IARRAY,RARRAY)

The attribute values of LINTYP, LINWID, INTENS, COLOUR, NAME, CFONT, CQUAL, CSPAC3, CSIZE, CPLANE, CPATH, CJUST, MSIZE(M-UP+M-WIDTH) is saved in the array ARRAY.

Errors:
None

4.1.17. Set attributes

SETATR (ARRAY)

The attribute values of LINTYP, LINWID, INTENS, COLOR, NAME, CFONT, CQUAL, CSPAC3, CSIZE, CPLANE, CPATH, CJUST, MSIZE(M-UP + M-WIDTH) is set to current attributes.

Errors:
None.
5. PICTURE SEGMENTATION AND NAMING

5.1. Overview

All the output primitives for a graphical object are placed by IDIGS in an application program specified segment. Each segment defines an image, which is a view of the object, and which is part of the picture displayed on the view surface. An application program describes an object by creating (opening) a segment, invoking output primitive functions and then closing the segment. The application program can specify whether primitives become visible as the output primitive functions are invoked, or at some later time. By grouping an object's primitives in a named segment, the programmer can selectively modify pieces of the picture by deleting and then recreating segments (effectively replacing them) so that their images change.

In the same way that output primitives are affected by primitive attributes, the images defined by segments are affected by dynamic segment attributes. The visibility and highlighting attributes are used to control the appearance of the image defined by the segment. The detectability attribute makes the image selectable by a PICK input device, such as a lightpen.

Two types of segments can be created. Virtuell segments are used primarily for plotters, on which their images are displayed once, but no record is kept of the segments or their contents. Retained segments are used primarily for buffered displays, on which their images are displayed and records of the segments are kept until they are explicitly deleted or IDIGS is closed.

In addition to being identified by the name of the segment into which they have been placed, output primitives may be identified by a PICKID name, specified as a primitive attribute. Thus within the single level of segmentation, an additional level of naming is provided. This feature
is useful, for example to allow all the character strings for a particular light button menu to be defined in a single segment. Identification (segment name, PICKID) of which light button was selected by the operator is returned to the application program. Note that one or more primitives may be assigned the same PICKID to make them selectable as a unit.

When a segment is closed, graphical information of the individual primitives cannot be modified nor can primitives be added to or deleted from the segment. However if the segment is stored, transformations and modifications of the segment are possible. Most of the display information is fixed when a BGNPIC occurs. For example window, viewport and attributes have to be set before a segment is opened. If the programmer does not set these segment attributes, the default values listed in (7.6.) will be valid.

5.1.1. Begin segment

BGNPIC (NAME)

NAME = 0 virtual segment

0 < NAME < 32767 retained segment with name NAME.

BGNPIC creates a new, empty segment. The type of the new segment is determined from the NAME parameter (which is 0 for virtual segments) and the level of the system. If the type is virtual the NAME parameter is ignored. If the type is retained, the NAME parameter determines the name of the segment.

The CP is set to the origin of the world coordinate system, and the current primitive attribute values are set to their system defined default values. Also the new segment becomes the open segment.

While a segment is open, the viewing transformation may not be altered. (In the synthetic camera analogy, this would
correspond to moving the camera while the shutter is open.) Also view surfaces may not be selected or deselected.

Errors:
1. There already is an open segment.
2. A retained segment named NAME already exists.
3. 32767 < NAME < 0
4. 

5.1.2. Close segment

ENDPIC

The currently open segment becomes closed, i.e., ceases to be open. It does not matter whether the segment is retained or virtual. There is no open segment after ENDPIC is called and a new BGNPIC is invoked. Thus output primitive functions may not be called until another segment is created. Closing the open segment has no effect on its VISIBILITY or other dynamic segment attributes.

Error:
1. There is no open segment.

5.1.3. Delete segment

DELPIC (NAME) 1 ≤ NAME ≤ 32767

The segment named by the parameter NAME is deleted.

If the open segment is deleted, it is first automatically closed, so that there is no open segment after DELPIC returns. If it is not the open segment which is being deleted, the open segment remains open.

Errors:
1. No retained segment named NAME exists.
5.1.4. Rename

RENAME (OLD-SEGMENT, NEW-SEGMENT)

The existing retained segment named by the parameter OLD SEGMENT is henceforth known by the name specified by the NEW-SEGMENT parameter. After RENAME returns, there is no segment with the name OLD SEGMENT. This function has no visible effect.

This function is provided to make it easier for an application program to use the human factors technique of "double buffering": preparing a replacement image invisibly while the image to be replaced remains visible, then deleting the old one and making visible the new one. The final step in this technique is to rename the new segment to have the old name, so that the same process can be repeated.

Errors:
1. There is an existing retained segment named NEW-NAME.
2. There is no retained segment named OLD-SEGMENT.
6. SEGMENT ATTRIBUTES

6.1. Overview

There are four dynamic segment attributes in IDIGS: VISIBILITY, HIGHLIGHTING, DETECTABILITY and IMAGE-TRANSFORMATION.

These are only available for retained segments.

Sending of the graphical primitives to a work station may be performed in several ways which are controlled by the TYPE in the special function BGNFUN/ENDFUN.

- either concurrently with the definition, that is each output primitive is sent to the work station immediately after definition

- or in a batch type way such that all graphical information defined in one segment are sent to the work station when the IDIGS performs the closing of the segment.

The display of the segment on the display space of the work station is controlled by the "visibility"-attribute of the segment. Invisible segments may be sent to a work station in order to be stored for later display.

6.1.1. Visibility

VISPIC (IDENT, ISWITCH)

ISWITCH = 0  Invisible
ISWITCH = 1  Visible

Control visibility of the picture segment IDENT. The default of the current value for ISWITCH is visible.

6.1.2. Highlighting

HILPIC (IDENT, ISWITCH)
ISWITCH = 0  Non-highlighted
ISWITCH = 1  Highlighted

HIGHLIGHTING indicates whether or not a segment image is to be distinguished from other segment images. If it is possible, HIGHLIGHTING shall be implemented with a hardware blinking feature; otherwise IDIGS uses other techniques, such as intensity variation. HIGHLIGHTING has two values ("highlighted" and "non-highlighted") and affects the appearance of all output primitives in the segment. Default value is non-highlighted.

6.1.3. Detectability

DETPIC (IDENT, ISWITCH)

ISWITCH = 0  Non-detectable
ISWITCH = 1  Detectable

DETECTABILITY indicates whether or not a segment can be detected by a PICK input device. It has two values ("detectable" or "non-detectable") and affects all output primitives in the segment. The system-provided initial default for the current value for DETECTABILITY is "non-detectable".

6.1.4. Image-transformation

IMAGE-TRANSFORMATION indicates how the image of a segment is translated, scaled, and rotated when it appears on the view surface.

IMTRA (IDENT, ARR2)

ARR2(1) = XO            fix-point
ARR2(2) = YO
ARR2(3) = SX
ARR2(4) = SY
ARR2(5) = AZ
ARR2(6) = DX
ARR2(7) = DY translate

IMTRA3 (IDENT, ARR3)

ARR3(1) = XO
ARR3(2) = YO fix-point
ARR3(3) = ZO
ARR3(4) = SX
ARR3(5) = SY scale
ARR3(6) = SZ
ARR3(7) = AX
ARR3(8) = AY rotate
ARR3(9) = AZ
ARR3(10) = DX
ARR3(11) = DY translate
ARR3(12) = DZ

The segment specified is first scaled by Sx, Sy, (Sz) then rotated by (Ax, Ay,) Az about fixpoint xO, yO, (zO) and finally shifted by dx, dy (dz). The transformations are executed in exactly this sequence. (A different sequence would lead to other results.)

All parameters are given in NDC coordinates.

A new set of parameter values will not be concatenated with old ones. The system-provided initial default for the current attribute value for the IMAGE-TRANSFORMATION attribute is the identity transformation.

A clipping mechanism will ensure that the drawing does not exceed the workstation viewport.

Errors: (for the segment attributes)
1. Invalid (i.e., non-existing) segment.
2. Invalid attribute value.
3. Invalid image transformation for the segment.
7. VIEWING

7.1. Overview

The separation of graphics operations into modelling and viewing functions provides a useful paradigm, that of a synthetic camera taking a view (snapshot) of an object. For 2D objects, the synthetic camera's viewing transformation is specified by a window in world coordinates and a viewport on the NDC space. The window may be inclined with respect to the principal axes. It is used to clip the object and to determine the window to viewport mapping. This mapping takes the portion of the 2D object bounded by the window into the portion of the normalized device coordinates space bounded by the viewport.

A window in the NDC space is then mapped on to the view surface by a workstation transformation. The workstation transformation will map the workstation window with uniform scaling onto as large a rectangle that will fit into the workstation viewport.

If no workstation window has been specified explicitly the default (unit square) will be used. If no workstation viewport has been specified explicitly the workstation transformation will map the workstation window under uniform scaling onto the available display surface.

Both the size and the position of the workstation viewport may be specified. A clipping mechanism will ensure that the drawing does not exceed the workstation viewport.

The viewing transformation in 3D corresponds to the specification of the synthetic camera's position, its type of projection (e.g., perspective or parallel), and where on the specified view surface the view of the object (the image) is to appear. Since the viewing transformation allows arbitrary positioning of the camera, IDIGS need not support transformations (translation, rotation and scale) of objects. Thus, to get a different view, the camera is moved into a new position.
Fig. 7.1. Elements in the 3D viewing transformation system.

In 3D, the window is specified in an arbitrary view plane (projection plane). If a perspective projection is specified, the object is clipped to a pyramid and projected onto the view plane in such a way that objects close to the camera appear larger than objects further away.

The world coordinate system is right-handed.

The viewing transformation is sufficiently general that the camera need not be centered above the window. For perspective projections, this means that the centre line of the pyramid need not be perpendicular to the view plane. For parallel projections, this means that the center line of the parallelepiped need not be perpendicular to the view plane. Orthographic, cavalier, and cabinet projections are special cases of parallel projections. In all cases, the portion of the view plane bounded by the window is mapped into the portion of normalized device coordinate space bounded by the viewport.
If a parallel projection is specified, the object is clipped to a parallelepiped and is projected onto the view plane in such a way that objects of the same size appear to be the same size no matter how far away they are positioned with respect to the camera.
Subsequent to the viewing transformation, the resulting image can be translated, scaled, rotated using image transformations and mapped from the NDC Window to the workstation viewport. In terms of the synthetic camera paradigm, this facility corresponds to positioning, rotating, and enlarging or reducing the snapshot when it is placed on the view surface.

In summary: IDIGS regards three levels of coordinate systems.

a) One world coordinates system (WC)
b) One normalized device coordinate system (NDC)
c) A number of device coordinate systems (DC)

Two transformations are applied to an output primitive enroute from the application program to the display surface:

- A window/viewport transformation that maps the world coordinate space onto the NDC space,

- A workstation transformation that is used to map the NDC space onto the DC space individually for every workstation.
7.2. 3D viewplan transformation

7.2.1. View reference point

VREFPT (X,Y,Z)

The three parameters give the view reference point, in world coordinates. The initial default point is (0, 0, 0).

Errors:
1. A segment is open.

7.2.2. View plane normal

VPNORM (DX,DY,DZ)

The parameters determine a world coordinate point by giving its position relative to the view reference point. The view plane normal is the directed line segment from the view reference point to the given point. The default view plane normal specification is (0, 0, -1).

Errors:
1. A segment is open.
2. DX, DY, and DZ are all zero: no direction can be established.

7.2.3. View plane distance

VPDIST (VIEW-DISTANCE)

The view (or projection) plane is specified. It is perpendicular to the view plane normal, and is VIEW-DISTANCE from the view reference point. Distances are measured in world coordinate units from the view reference point, with positive values corresponding to the direction of the view plane normal and negative values corresponding opposite direction. The initial default view plane is located at the view reference point, corresponding to a VIEW-DISTANCE of 0.
Errors:
1. A segment is open.

7.2.4. Parallel projection

AXON (DX, DY, DZ)

A parallel projection is selected. The parameters specify a world coordinate point by giving its position relative to the view reference point. The projection lines for the projection are parallel to the line running through the view reference point and this point.

The initial default projection is parallel with parameters (0, 0, +1).

Errors:
1. A segment is open.
2. DX, DY, and DZ are all zero: the projection direction cannot be determined.

7.2.5. Perspective projection

PERS (DX, DY, DZ)

A perspective projection is selected. The center of projection is the world coordinate point specified by the parameters relative to the view reference point. In the synthetic camera analogy, the camera is at the center of projection, aimed parallel to and in the direction of the view plane normal. The initial default projection is parallel.

Errors:
1. A segment is open.

7.2.6. View up in 3 dimension (3D)

VUP3 (DX-UP, DY-UP, DZ-UP)
The three parameters determine a world coordinate point by giving its position relative to the view reference point. The view up directed line segment runs from the view reference point to this point. This directed line segment, when projected onto the view plane (parallel to the view plane normal), specifies the positive V-axis of the UV coordinate system in the view plane. The U-axis is in the view plane also, such that the U-axis, the V-axis and the view plane normal form a left-handed coordinate system. The V-axis will be vertical when the view plane is mapped to the view surface. In the synthetic camera analogy, the camera is oriented such that the projection of the view up vector appears vertically in the viewfinder.

The default view up vector is \((0, 1, 0)\), which causes the Y-axis to be "up". Note that this default will not work if the view plane normal is parallel to the Y-axis.

**Errors:**
1. A segment is open.
2. DX-UP, DY-UP, and DZ-UP are all zero: no view up direction can be established.

7.2.7. Window in 3D

**WINDW3 (UMIN, UMAX, VMIN, VMAX)**

The window, defined by the four parameters in the UV coordinate system, is in the view plane. The left and right sides of the window are specified by UMIN and UMAX and are parallel to the V-axis, and the top and bottom of the window are specified by VMIN and VMAX and are parallel to the U-axis.

If a perspective projection has been selected, the centre of projection and window define a semi-infinite viewing pyramid, the contents of which are projected onto the window in the view plane. If a parallel projection has been selected, the projection direction and the window define an infinite parallelepiped, the contents of which are projected onto the window in the view plane.
If window clipping is disabled, the entire view plane is mapped to the NDC space. The image appearing in the NDC space is well-defined only for that portion of the view plane which, when mapped to the NDC space, falls within the range of the specified normalized device coordinate space. Objects whose views in the NDC space cross or exceed the bounds of normalized device coordinate space will be displayed in a hardware- or installation-dependent manner. The initial default value for the window is \((0, 1, 0, 1)\), and clipping is initially disabled.

Errors:
1. A segment is open.
2. UMIN is not less than UMAX, or VMIN is not less than VMAX.

7.2.8. View depth

\textbf{VDEPTH (FRONT-DISTANCE, BACK-DISTANCE)}

This function specifies planes for depth clipping, but does not affect whether depth clipping is performed. The initial default condition is that depth clipping is off.

Errors:
1. A segment is open.
2. FRONT-DISTANCE is greater than BACK-DISTANCE, so that the back clipping plane is in front of the front clipping plane.

7.3. 2D viewplane transformations

One additional function, included in IDIGS to simplify viewing of planar objects which lie in the XY plane, is described in this section. The capability it provides is useful to implementors of 2D applications. In reality the viewing does occur in a 3D environment. The defaults have simply been arranged to specify an orthogonal (parallel) projection onto the view plane, which is defaulted to the XY plane.
If the U- and X-axes are coincident, as are the V- and Y-axes, thus, a window can be specified in the XY plane. The one additional function VUP allows is a 2D view up direction to be specified, so that rotated windows can be specified. As usual IDIGS implies the current z value for specifying the 3D world coordinate position.

7.3.1. View up in 2 dimension (2D)

VUP (DX-UP, DY-UP)

This function specifies the world coordinate "up" direction, so that the world coordinate Y-axis need not be "up" on the view surface. The synthetic camera is rotated about its line of sight so that the vector from the origin to (DX-UP, DY-UP) would appear to be vertical in the camera's viewfinder. The default for (DX-UP, DY-UP) is (0, 1). Thus the world coordinate Y-axis direction is the default view up direction.

VUP can be used interchangeably with VUP3 (DX-UP, DY-UP, CURRENT-Z).

Errors:
1. A segment is open.
2. DX-UP and DY-UP are both zero: no view up direction can be established.

7.3.2. Window in 2D

WINDW (XMIN, XMAX, YMIN, YMAX)

The parameters to WINDW specify a rectangle in world coordinates. The rectangle's left and right sides are vertical; its top and bottom are horizontal. This rectangle is rotated about the origin so that the sides running from the bottom to the top of the rectangle run in the direction of the view up vector. The rotated rectangle defines the window. In the default case, the view up vector is the positive Y direction, so that no rotation need to be performed.
The window and the viewport define the transformation which will be used to map from world coordinates to normalized device coordinates. When enabled, clipping is performed at the window boundary. That is, points on or inside the window boundary are visible; points outside the boundary are not. If window clipping is disabled, then the entire XY plane is mapped to the NDC space. The image appearing in the NDC space is well-defined only for that portion of the XY plane which, when mapped to the NDC space, falls within the range of the specified normalized device coordinate space. Any object whose image is partially or completely outside these bounds will be displayed in an installation- or hardware-dependent way. The default window specification is (0, 1, 0, 1), and clipping is initially disabled.

Errors:
1. A segment is open.
2. XMIN is not less than XMAX, or YMIN is not less than YMAX.

7.4. 2D/3D NDC transformation

This section describes the transformation from the view volume to one or more viewports which lays within a defined Normalized Device Coordinate Space (NDC). The NDC space is defined to be in range 0-1 in x, y and z.

7.4.1. Viewport in 3D NDC

NDCVP3 (XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX)

The parameters give the minimum and maximum bounds of the viewport in 3D normalized device coordinate space. The parameters correspond to he left, right, bottom, top, front, and back surfaces of a parallelepiped, all for whose sides are either vertical or horizontal. The 3D viewport will be used for displaying all picture primitive until a new 3D viewport or 2D viewport is specified. The initial default 3D viewport specification is the same as the 3D normalized device coordinate space.
The view volume is defined by either the semi-infinite pyramid for a perspective projection or by an infinite parallelepiped for a parallel projection, as limited by the front and back clipping planes. The contents of the view volume are mapped into the viewport such that the front clipping plane coincides with the plane defined by ZMIN, and the back clipping plane coincides with the plane defined by ZMAX. The sides of the view volume then correspond to the sides of the 3D viewport.

If either window or depth clipping is turned off, it is possible that some output primitives, when transformed into normalized device coordinate space, will exceed the range specified in NDCSP3. The results of such an occurrence are hardware dependent, and will vary from installation to installation.

Errors:
1. View volume corner(s) are outside of normalized device coordinate space.
2. A segment is open.
3. XMIN is not less than XMAX, or YMIN is not less than YMAX, or ZMIN is not less than ZMAX.

7.4.2. Viewport in 2D NDC

NDCVP (XMIN, XMAX, YMIN, YMAX)

The parameters give the extent, in 2D normalized device coordinate space, of the current NDC viewport. The viewport's side are vertical, and its top and bottom are horizontal. The viewport cannot exceed the bounds of normalized device coordinate space. This viewport will be used for displaying all output primitives until a new viewport is specified. The initial default viewport specification is the entire normalized device coordinate space, as specified by NDCSP3 or NDCSP, or by the default if neither of these functions has been invoked. If VPORT is invoked, rather than VPORT3, then ZMIN and ZMAX are set to 0.
Errors:
1. Viewport extent is outside of normalized device coordinate space.
2. A segment is open.
3. XMIN is not less than XMAX, or YMIN is not less than YMAX.

7.5. Workstation transformation

The normalized device coordinate system will be mapped onto the individual device coordinate systems, that describe the display space of each workstation. The device coordinates are measured in metres. This workstation transformation can be set individually for each workstation, allowing the same graphical primitive to appear on different display surfaces at different, application program controlled, scales. In this way it is possible, for example, to use the full display space of an interactive workstation and simultaneously to have the same drawing in correct scale on a plotter. The workstation transformation is set by specifying a workstation window in NDC and the workstation viewport in DC space.

The workstation transformation always preserves aspect ratio. The workstation window is mapped with uniform scaling onto as large a rectangle as will fit into the workstation viewport size.

The default setting of the workstation transformation will map the unit NDC square under uniform scaling onto a square defined by the shorter side of the workstation viewport. The same workstation transformation is valid for all graphical output primitives on an individual workstation. A resetting of this transformation changes the scale of all segments stored on the respective workstation.

IDIGS takes care that the result of the window/viewport transformation does not exceed the unit square and that the result of the workstation transformation does not exceed the workstation viewport.
7.5.1. Workstation window in 3D

WSWIN3 (WIDENT, XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX)

The WIDENT workstation window is set to the box defined by the given points in the normalized device space.

The part of the NDC space defined by this box will be mapped with uniform scaling on the current workstation viewport. Default window is the whole NDC space.

Errors:

1. Window extent is outside of NDC
2. A segment is open
3. XMIN is not less than XMAX, or YMIN is not less than YMAX, or ZMIN is not less than ZMAX
4. WIDENT is not a valid workstation

7.5.2. Workstation viewport in 3D

VPORT3 (WIDENT, XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX)

The WIDENT workstation viewport is defined by the given points in the device coordinate space. The viewport is a box with the sides parallel to the x-, y- and z-axis. All points are given in metres. Default workstation viewport is the whole view surface.

Errors:

1. The viewport exceeds the display surface
2. A segment is open
3. XMIN is not less than XMAX, or YMIN is not less than YMAX, or ZMIN is not less than ZMAX
4. WIDENT is not a valid workstation identifier
7.5.3. Workstation window in 2D

WSWIN (WIDENT, XMIN, XMAX, YMIN, YMAX)

The WIDENT workstation window is set to the rectangle defined by the given points in the NDC space.

The part of the NDC space defined by this rectangle will be mapped with uniform scaling on the current workstation viewport. Default window is the NDC space.

Errors:

1. Window extent is outside of NDC
2. A segment isopen
3. XMIN is not less than XMAX, or YMIN is not less than YMAX
4. WIDENT is not a valid workstation identifier

7.5.4. Workstation viewport in 2D

VPORT (WIDENT, XMIN, XMAX, YMIN, YMAX)

The WIDENT workstation viewport is defined by the given points in the device coordinate space. The viewport is a rectangle with the sides parallel to the x- and y-axis. All points are given in metres. Default value is the whole view surface.

Errors:

1. The viewport exceeds the display surface
2. A segment is open
3. XMIN is not less than XMAX, or YMIN is not less than YMAX
4. WIDENT is not a valid workstation identifier

7.6. Model transformations

Modelling functions are not considered to be a point of
the IDIGS system. However to allow an efficient implementation of such a system on top of IDIGS, functions are added to set the modelling transformation. Model transformations are specified by a matrix which is composed with the composite viewing matrix to obtain a composite viewing operation matrix.

7.6.1. Model transformation in 2D

MODTR (TRANS-MATRIX)

The parameter defines a 3x3 transformation matrix that is to be combined with the viewing matrix. It MODTR is used in a 3D system, the unspecified components of the 4x4 matrix are set to the identity matrix.

7.6.2. Model transformation in 3D

MODTR3 (TRANSF-MATRIX)

The parameter defines a 4x4 transformation matrix that is to be combined with the viewing matrix. A model transformation matrix is composed as follows:

$$\begin{pmatrix}
  r & r & r & p \\
  r & r & r & p \\
  r & r & r & p \\
  t & t & t & s \\
\end{pmatrix}$$

- \( r \) = rotation and shear elements
- \( t \) = translation
- \( p \) = perspective
- \( s \) = overall scale

7.7. Default values

The following table lists the defaults taken by all the viewing parameters whenever initialized. Some default values depend upon the current value of other viewing parameters: the default front distance equals the current view distance.
the IDIGS system. However to allow an efficient implementation of such a system on top of IDIGS, functions are added to set the modelling transformation. Model transformations are specified by a matrix which is composed with the composite viewing matrix to obtain a composite viewing operation matrix.

7.6.1. Model transformation in 2D

MODTR (TRANS-MATRIX)

The parameter defines a 3x3 transformation matrix that is to be combined with the viewing matrix. It MODTR is used in a 3D system, the unspecified components of the 4x4 matrix are set to the identity matrix.

7.6.2. Model transformation in 3D

MODTR3 (TRANSF-MATRIX)

The parameter defines a 4x4 transformation matrix that is to be combined with the viewing matrix. A model transformation matrix is composed as follows:

```
| r r r p |
| r r r p |
| r r r p |
| t t t s |
```

\( r = \) rotation and shear elements  
\( t = \) translation  
\( p = \) perspective  
\( s = \) overall scale

7.7. Default values

The following table lists the defaults taken by all the viewing parameters whenever initialized. Some default values depend upon the current value of other viewing parameters: the default front distance equals the current view distance.
Once the front distance is specified explicitly, it will not again implicitly be set equal to the view distance, until IDIGS is again initialized. The same is true for the relationship between back distance and view distance, and between NDC viewport and normalized device coordinate space.

<table>
<thead>
<tr>
<th>Viewing Parameter</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normalized Device Coordinate Space</td>
<td>(1, 1, 0)</td>
</tr>
<tr>
<td>NDC Viewport</td>
<td>Normalized device coordinate space</td>
</tr>
<tr>
<td>Window</td>
<td>(0, 1, 0, 1)</td>
</tr>
<tr>
<td>View Reference Point</td>
<td>(0, 0, 0)</td>
</tr>
<tr>
<td>View Plane Normal</td>
<td>(0, 0, -1), relative to the View Reference Point</td>
</tr>
<tr>
<td>View Up Vector</td>
<td>(0, 1, 0), relative to the View Reference Point</td>
</tr>
<tr>
<td>NDC window</td>
<td>NDC space</td>
</tr>
<tr>
<td>Workstation Viewport</td>
<td>The whole view surface</td>
</tr>
<tr>
<td>Type of Projection</td>
<td>Parallel (0, 0, 1), (perpendicular to the XY plane)</td>
</tr>
<tr>
<td>View Distance</td>
<td>0</td>
</tr>
<tr>
<td>Front Distance</td>
<td>View Distance</td>
</tr>
<tr>
<td>Back Distance</td>
<td>View Distance</td>
</tr>
</tbody>
</table>

**IMAGE-TRANSFORMATION**

| Scale                                  | 1, 1, 1 (no scaling) |
Rotation 0, 0, 0 (no rotation)
Translation 0, 0, 0 (no translation)

These defaults produce a parallel projection along the Z-axis onto the XY plane. Also, by default, window clipping is enabled while the front- and back plane clipping is disabled. The world coordinate system is right-handed.

The defaults are arranged so that the programmer who wishes to deal with a 2D world need use only NDCSP, VPOR, VUP, and WINDW. The 2D world coordinate points given by these functions always have a Z coordinate of the current value added implicitly. Similarly, the programmer who does not wish to deal with 3D image transformations need not understand 3D normalized device coordinate space.

7.8. Polygons in the viewing system

The viewing transformation specifies a transformation from 3D world coordinates into a 3D rectangular viewport in 3D normalized device coordinates (NDC). The display on the view surface is then a parallel projection of the entire contents of 3D NDC space (the unit cube) onto the front plane of the unit cube. Two segments with identical 3D viewports then have their output primitives in the same part of 3D NDC space. A hidden surface algorithm will then operate on all of 3D NDC space, producing a single unified image or photograph on the view surface. Thus there is no need to be concerned about the priority of several overlapping photographs on the view surface. If two output primitives have identical coordinates in 3D NDC space, then temporal priority can be used to determine visibility. The most recently created output primitive would take priority over others.

If clipping is enabled while the output primitives are defined to IDIGS, then in 3D NDC space all output primitives will be contained within 3D viewports. Hidden surface algorithms can use this knowledge to avoid unnecessary comparisons
Consider first the 2D temporal priority discussed earlier. For many simple 2D applications of raster graphics such as displaying charts and graphs, 2D temporal priority is an attractive and useful concept for a programmer to use. This concept is in fact the default for IDIGS, because all 2D viewports (which is the typically-used default) lie in the front face of 3D NDC space (the Z = 0 plane). Hence output primitives in NDC space have no depth, so the hidden surface algorithms have no basis for deciding which output primitives are visible. It is in just this case that temporal priority is used.

As a second case, consider the 2 1/2 D case, wherein each segment has an explicit priority for visibility. All output primitives, however are 2D.

The effect of 2 1/2 D can be achieved by using segment priorities in the range 0-1. A segment with priority P and 2D viewport defined by corner points \((X_{\text{min}}, Y_{\text{min}}), (X_{\text{max}}, Y_{\text{max}})\) would in fact have a 3D viewport defined by corner points \((X_{\text{min}}, Y_{\text{min}}, P), (X_{\text{max}}, Y_{\text{max}}, P)\). That is, the 3D viewport is in the \(Z = P\) plane. Hence the effect of the depth priority for a 2D segment is achieved by putting the image of the segment on the appropriate constant Z plane in 3D NDC space.
8. INPUT PRIMITIVES

8.1. Overview

The basic goal of the IDIGS input primitives is to provide a framework for application program interaction which is independent of the particular physical input devices that are available on a given configuration.

The input system should be so flexible and powerful that it can get a maximum utilization of a given configuration and allow higher level input routines to be efficiently implemented.

A set of logical input devices are defined and controlled by the application program. The logical input devices are divided into disjoint classes such that each primitive device may be either sampled or cause an event to occur, but not both. IDIGS dynamically maps the logical device on to whatever available physical devices most conveniently implement the desired function. The primitive input device are low level and are defined to allow a natural and efficient mapping to commonly available physical input devices and to be used as primitive elements for building up more complicated compound input devices.

Logical compound input devices are characterised by having:

1. An identifying number

2. An associated physical device or devices from which the input actually came, including a TRIGGER with a TRIGGER-ATTRIBUTE which actually causes the dispatch of the input data to the input queue.

3. A value or a set of values generated by a sampled device and/or the terminal operator's action.

4. A prompt to the operator to announce the program's readiness
for logical input.

5. An echo to the operator to announce the recognition of his input.

Primitive event devices may be regarded as compound devices with the device itself as the trigger.

IDIGS provides the facilities for building up more complicated compound input devices like DIGITIZERS, GROUPERS, DRAGGERS etc.

To make this compound input device efficient, the data handling is done on a low level in order to avoid spending time popping up to the surface for every single action. This may be specially time consuming when working on a satellite system or on a graphic network.

A large number of applications do not require simultaneous use of multiple input devices. Also, some time sharing systems prohibit such simultaneity. For these applications and systems, the input facilities described above requires needless overhead, both in program complexity and in programmer sophistication. IDIGS offers therefore also a simple REQUEST input function that simplify the programming task for many application programs. Furthermore, since this function constitute the input of Level 2 of IDIGS, an implementation at input Level 2 can be greatly simplified.

Input primitives provide the following general facilities:

- ATTACH and DETACH physical and logical devices;
- REQUEST input values directly.
- queuing/dequeuing of events caused by event generating devices;
- sampling of current settings of sampled devices;
- Association of devices such that one device will serve as a trigger for another device.

To each trigger is associated a trigger attribute specifying
the threshold of the trigger. The values of all associated devices are read when the trigger is activated. The set of values generated is saved on the event queue and the control is returned to the program when the trigger on the highest level is activated.

Note that although physical input devices are normally connected to a particular physical output device, this information is not normally directly available to the application program using IDIGS. The conceptual framework of IDIGS treats input and output devices as orthogonal concepts. In particular, the selection of output surfaces has no effect on issues related to input, such as where echoing or device simulation is performed.

8.2. Functional Capabilities

8.2.1. Logical Input Facilities

The logical input devices supported by the IDIGS are divided into two disjoint sets -- those that only cause events and those that may only be sampled. Event devices are used by the operator to signal an event to the application program. An event report is created for each event caused by an event device and contains data related to the state of the event device when the event occurred. Event reports are placed on an event queue by the IDIGS and remove by the application program. Event devices may not be sampled by the application program. The event devices are:

- **PICK** - identify a position, a segment and a primitive within a segment
- **TEXT** - provide alphanumeric information
- **CHOISE** - provide logical "buttons" or combination of "buttons" (i.e., the ability to choose among alternatives)
- **DISTANCE** - cause an event when a cursor, etc. has moved a certain distance.
TIMER - cause an event when a certain time has elapsed.

Sampled devices have values which may be sampled by the application program. Sampled devices do not cause events. The sampled devices are:

LOCATOR - provide position information in world coordinates from a N dimensional space. N is specified by the user.

VALUATOR - provide scalar values within a program specified value range.

The IDIGS automatically echoes operator interaction in a simple manner, unless the application program deactivates the echo.

Note that events and the setting of sampled device values are defined as logical actions. The physical action taken by an operator in order to cause a particular logical event or define a particular value may be different on different systems. Furthermore, if one physical device is being used to simulate two or more logical devices, it is possible that the same physical action may generate different logical events. For example, when using a storage tube terminal, it is likely that both the TEXT and the CHOISE devices will be mapped to the physical keyboard. There will have to be conventions established by the storage tube device driver to allow it to determine whether a given operator keystroke is intended to act as the logical CHOISE or the logical TEXT.

8.2.2. Event devices

8.2.2.1. Pick

A PICK device is an event device that returns the position of the "pick" and the identification of the segment and output primitive "picked" by the PICK device. A typical
PICK device is the lightpen. A common simulation technique is to use a data tablet or a joystick plus a hardware or software coordinate comparator. A pick by the operator using the PICK device causes a logical event to occur. An event report is created and added to the event queue. The PICK event report contains the position of the hit in world coordinates and an identifier for the segment that contains the picked primitive and the PICK-ID associated with that primitive. One possible echo for a PICK operation may be to highlight the picked segment.

8.2.2.2. TEXT

TEXT is an event device that allows the operator to enter individual characters or entire text strings. A typical device is the normal alphanumeric keyboard. Simulation is possible using light buttons and a lightpen or data tablet. The TEXT device allows the operator to enter an entire text string as a single entity. The text string is limited to a maximum length defined by the program. The trigger is activated when a string terminator (system-defined carriage return) is typed or when the maximum length is reached. The TEXT event report contains the string typed by the operator and the number of characters in the string. The normal echo is to display the individual characters at a program-defined position as they are typed. IDIGS supports simple editing features allowing the operator to delete the previous character and the entire string. The specific characters used to invoke these features and the echo will remain installation-dependent and should normally coincide with the conventions of the operating system being used.

8.2.2.3. CHOISE

A CHOISE is an event device that allows the operator to choose between alternatives in a program.

The typical CHOISE is the physical button or combination of buttons on a program function keyboard (PFK), but the CHOISE device can be simulated by light buttons or by a
keyboard device. The CHOOSE event report identifies which logical CHOOSE device caused the event. The echo for a CHOOSE device is very dependent on the available physical devices. For example, a normal implementation of CHOOSE using a PFK might utilize the lamps physically within the pfk device to echo a CHOOSE. (Note that the physical lamps can also be controlled by the application program directly).

8.2.2.4. Distance

A DISTANCE is an event device that allows the operator to generate an event by moving a cursor etc. by a certain distance. A typical DISTANCE device is the cursor used for digitizing.

The DISTANCE device is usually used as a trigger.

A possible echo for a DISTANCE event is to draw a brighter spot on the position where the event occurred.

8.2.2.5. Timer

A TIMER is an event device that causes an event when a certain amount of time has elapsed.

A typical TIMER device is the real time clock.

The TIMER is used as a pure event device or as a trigger.

A possible echo for a TIMER is a blinking spot.

8.2.3. Sampled devices

8.2.3.1. Locator

A LOCATOR is a sampled device that provides coordinate information in the world coordinate space representing a location on the view surface. Typical LOCATORS are data tablets and joysticks. The most common LOCATOR simulation uses a lightpen tracking cross. When sampled the LOCATOR returns N-dimensional coordinates of the current location.
in world space coordinates. \( N \) is specified by the user.

The LOCATOR echo may consist of a cursor (typically a small cross) displayed on the display surface at the current LOCATOR position. (Note: Many physical devices that are normally used as LOCATORS have switches. These switches are accessed by IDIGS as logical CHOISE. The application program can then associate the LOCATOR with the CHOISE in order to provide the normal effect of having that physical device serve as a TRIGGER for the LOCATOR.

8.2.3.2. Valuator

A VALUATOR is a sampled device that allows the operator to enter numeric values within a range specified by the application program. A typical VALUATOR is the control dial, but multi-position switches and toggle switches can be utilized as low-precision VALUATORS. VALUATORS can be simulated using lightpens, data tablets, gas pedals, light handles, etc. or a keyboard (by typing the desired number). VALUATOR values are constrained to lie within a program-specified value range. IDIGS maps the values received from the physical device in a linear fashion onto this range. In addition, there is a program-defined initial value which is used when appropriate. For example, when a VALUATOR is being simulated, the program-specified value is used as the initial value for the simulation. Also, continuous-turn control dials could utilize this value as a base for determining incremental values. On such dials, the number of turns which corresponds to the specified range is installation-dependent, as is whether exceeding the maximum number of turns will result in wraparound or clamping. The echo of a VALUATOR is a numeric value displayed on the view surface at a program-defined position and is constantly changed to reflect the current value of the VALUATOR.

8.3. Access to the physical Input Devices

IDIGS provides a mechanism for dynamic attachment of physical and logical devices by the application program in order
to utilize existing physical devices to a maximum.

8.3.1. Attach device

ATTACH (TOOL-ID, DEVICE-CLASS, DEVICE-ID)

This function will attach the physical device TOOL-ID to the logical device class DEVICE-CLASS and create the device DEVICE-ID. If multiple workstations are active for input the TOOL-ID must contain information of ON which workstation the tool is located.

Errors:
1. TOOL-ID or DEVICE-CLASS is invalid.

8.3.2. Detach device

DETACH (DEVICE-ID, TOOL-ID)

The specified device is detached. TOOL-ID becomes free for other use.

Errors:
1. DEVICE-ID is invalid.

8.4. Input levels

8.4.1. Request input

In order to make primitive input simple, IDIGS offers functions for direct request of primitive input data.

The operator at the workstation had the possibility to enter an input primitive of the requested class. IDIGS waits until the requested input has been entered. The REQUEST input behaves in this way similar to a FORTRAN READ.

8.4.1.1. Request PICK data

REQPIC (DEVICE-ID, DATA-ARRAY, POSITION)
A PICK event from DEVICE-ID identifies the primitive chosen by the name of the segment containing it and the primitive's PICK-ID attribute value. This information is returned in the DATA-ARRAY of this function together with the PICK position in world coordinates. IDIGS waits until the PICK data is entered.

Errors:
1. DEVICE-ID is invalid

8.4.1.2. Request TEXT data

REQTXT (DEVICE-ID, MAX, DATA-ARRAY, NUMB)

The parameter DATA-ARRAY represents the space for storing the string entered. MAX is the number of characters that fit in that space. If DATA-ARRAY is not large enough to hold all the characters from the TEXT buffer, IDIGS only returns what fits. The parameter NUM, however, is always set to the number of characters entered into the TEXT buffer by the operator. By specifying a buffer size of one, the application program can receive separate events for each character entered by the operator. (If an operating system or hardware prevents IDIGS from using a buffer in this fashion, a convention will have to be established at that installation to allow the operator/application program interface to achieve a similar effect. At worst, this may require the operator to enter a carriage return after each individual character is typed when the program is in this mode.) IDIGS waits until the TEXT data is entered.

Errors:
1. DEVICE-ID is invalid
2. DATA-ARRAY is not large enough to hold the string entered by the operator.

8.4.1.3. Request CHOISE data

REQCHO (DEVICE-ID, CHOISE-NUMBER)
A CHOOSE event identifies the particular choice activated.

The CHOOSE data is returned in CHOOSE-NUMBER. IDIGS waits until CHOOSE data is entered.

Errors:
1. DEVICE-ID is not valid

8.4.1.4. Request LOCATOR data

REQLOC (DEVICE-ID, DATA-ARRAY, NDIM)

The LOCATOR data (N-dimensional position) is returned in DATA-ARRAY.

IDIGS waits until LOCATOR data is entered NDIM is the LOCATOR dimension which is output from the device.

Errors:
1. DEVICE-ID is invalid

8.4.1.5. Request VALUATOR data

REQVAL (DEVICE-ID, VALUE)

The VALUATOR value is returned in VALUE.

Errors:
1. DEVICE-ID is not valid.

8.4.2. Input based on event queue

An event report is created for each event caused by an event device and contains data related to the state of the event device when the event occurred. Event reports are placed on an event queue by the IDIGS and removed by the application program. There is no pre-defined limit to the number of event reports from each device that can be placed on the event queue at any given time. However, physical memory requirements for
the event reports on the queue will certainly impose installation dependent limitations. Note that event reports for one event device may contain varying amounts of information due to changes in associations or enabling of associated devices.

Although many graphics hardware systems allow events to be generated asynchronously, IDIGS interface to the application program is purely synchronous. The application program must poll IDIGS to determine whether an event has occurred. This is the only approach that ensures operating system independence and is required by all commonly used programming languages. Event information is available to the application program only from the current event report, which is the event report that has been most recently removed from the event queue by the EVENT function.

8.4.2.1. Await event

EVENT (TIME, DEVICE-ID)

This function is a request by the application program to wait for the specified TIME or until there is at least one element on the event queue, then remove this element of the queue. The dequeued report becomes the current event report.

The only input argument is TIME which indicates the number of seconds the system should wait. The time parameter may be zero, indicating that the queue should be simple tested, the element dequeued (if one exists), and an immediate return to the application program made. If TIME < 0, EVENT will wait until there is at least one element on the queue. DEVICE-ID is set by IDIGS to identify the device that generated the event.

A "zero" device code is possible, indicating that no events occurred during the specified time. If TIME is set negative, the routine will wait until an event report is entered into the queue. (Note that some operating systems may not provide a reliable time-out facility. If this is the case, IDIGS will have to treat all non-negative TIME values as a TIME
Errors:
1. DEVICE-ID is invalid.

8.4.3. Compound input devices

IDIGS provides a mechanism by which the application program can associate any event generating device with any sampled or compound input device. The event device will in this case serve as a TRIGGER for the associated devices and read all the corresponding data of the compound device when activated.

An event is caused by a logical "action" taken by the operator or program by "pulling" the TRIGGER an amount specified by the TRIGGER-ATTRIBUTE. This action is indicating explicit input for the application program. An event caused by a logical action will result in the creation of an event report by IDIGS: The event report is saved by IDIGS on an event queue when the trigger is activated. The control is not returned to the program until the trigger on the highest level is activated.

8.4.3.1. Associate devices

ASSOC:(DEVICE-ID, TRIGGER-ID, ATTRIBUTE, NEW-DEVICE-ID)

DEVICE-ID: Logical input device identification
TRIGGER-ID: Identifier for the logical event device used as a trigger
ATTRIBUTE: Threshold for the trigger.
NEW-DEVICE-ID: Identifier for the compound device.

This routine logically associate a logical device with a TRIGGER.

When the trigger is activated all data corresponding to the devices associated with the trigger will be stored on an event queue.
Several occurrences of the report from a device may be stored on the event queue without the control returned to the application program. This will not happen until the TRIGGER on the highest level is activated.

All the sampling devices may, however, be directly interrogated anytime by the program.

If no trigger is specified for an event device, the event of the device itself is the trigger. Default trigger attribute type is the number of occurrences. Note that associations are many to many, so an invocation of this function for a particular pair of devices does not alter any previous associations for these devices.

Errors:
1. DEVICE-ID is active.
2. The specified association already exist.
3. The DEVICE-ID is invalid.
4. The TRIGGER-ID is invalid.

8.4.3.2. Disassociate device

DASDEV (DEVICE-ID, OLD-DEVICE-ID, TRIGGER-ID)

DEVICE-ID = Identifier for the compound device
output:
OLD-DEVICE-ID = Identifier for the logical device in DEVICE-ID
TRIGGER-ID = Identifier for the trigger in DEVICE-ID

Delete the association on the highest level of a compound logical input device. The identifiers for the old device and the trigger-id are returned in OLD-DEVICE-ID and TRIGGER-ID.

Errors:
1. DEVICE-ID is invalid.
8.4.3.3. Disassociate all devices

`DASALL (DEVICE-ID)`

Delete all existing associations (if any) for the specified device.

Errors:
1. DEVICE-ID is invalid.

8.5. Reading sampled devices

Whenever enabled, the sampled devices LOCATOR and VALUATOR, can be interrogated by the application program to determine their current state. The dimension of the LOCATOR device is N-dimensional.

8.5.1. Read locator

`REALOC (LOCATOR-ID, POS-ARRAY, N DIM)`

- LOCATOR-ID = LOCATOR identifier
- POS-ARRAY = Position in world coordinates
- N DIM = Dimension of the world coordinate space.

Returns the current position in world coordinates of the specified LOCATOR device.

Errors:
1. LOCATOR-ID is invalid.
2. The specified LOCATOR device is not enabled.

8.5.2. Read valuactor

`REAVAL (VALUATOR-ID, VALUE)`

Returns the current VALUE of the specified VALUATOR device.

Errors:
1. VALUATOR-ID is invalid.
2. The specified VALUATOR device is not enabled.

8.6. Input device control

An input device must be enabled before it can cause an event or be sampled.

8.6.1. Enable device

ENABLE (DEVICE-ID)

This function causes the specified device to become enabled. If the device is a sampled device, it becomes eligible for sampling.

Errors:
1. Specified device already enabled.
2. DEVICE-ID invalid.

8.6.2. Disable device

DISABL (DEVICE-ID)

The specified device is disabled. A sampled device may not be sampled and further use of an event device will not result in events being generated until the device is enabled again. This function has no effect on event reports for this device that are already on the event queue.

Errors:
1. Specified device is not enabled
2. DEVICE-ID invalid

8.6.3. Flush device

Unwanted event reports can be removed from the event queue.

FLUSHD (DEVICE-ID)
This function removes all events from the event queue that was generated by the specified device.

Errors:
1. DEVICE-ID is not valid

8.6.4. Flush all devices

FLUSHA ( )

This function removes all events from the event queue.

Errors:
None

8.7. Accessing event report data

As a first step in processing events, the application program invokes the EVENT function which dequeues an event report and returns the identification of the device causing the event. This identification is used to determine which functions in this section are appropriate to obtain the data in the event report.

At a minimum, event reports contain the data for the classes of event devices. Additionally, device associations can cause information from sampled devices to be included in the event report. Because of the considerable variability of amount and types of data in event reports, there are separate functions for each class of devices. These functions access data in the current event report. The data remains in the current event report until another event report is dequeued.

(Note that there is no GET-DISTANCE and GET-TIME DATA functions since there is no data directly related to those events. On the other hand, it is still possible for a DISTANCE and TIME event to have associated sampled data.)
8.7.1. Get PICK data

GETPIC (MAX, DATA-ARRAY, NUMB)

A PICK event identifies the primitive chosen by the name of the segment containing it and the primitive's PICK-ID attribute value. This information is returned in the DATA-ARRAY of this function together with the PICK position in world coordinates.

A series of input primitives can be entered. The maximum number of items is specified as the parameter MAX. The number of items entered is returned in the parameter NUMB.

If NUMB is returned with value 0, the current event report does not contain any PICK reports.

Errors:
1. The DATA-ARRAY is not large enough.

8.7.2. Get TEXT data

GETTXT (MAX, DATA-ARRAY, NUMB)

The parameter DATA-ARRAY represents the space for storing the string entered. MAX is the number of characters that fit in that space. If DATA-ARRAY is not large enough to hold all the characters from the TEXT buffer, IDIGS only returns what fits. The parameter NUMB, however, is always set to the number of characters entered into the TEXT buffer by the operator. By specifying a buffer size of one, the application program can receive separate events for each character entered by the operator. (If an operating system or hardware prevents IDIGS from using a buffer in this fashion, a convention will have to be established at that installation to allow the operator/application program interface to achieve a similar effect. At worst, this may require the operator to enter a carriage return after each individual character is typed when the program is in this mode.)
If NUMB is returned with value 0, the current event report does not contain any TEXT reports.

Errors:
1. DATA-ARRAY was not large enough to hold the string entered by the operator.

8.7.3. Get CHOISE Data

GETCHO (MAX, DATA-ARRAY, NUMB)

A CHOISE event identifies the particular choice activated.

The CHOISE data is returned from the current event report in the DATA-ARRAY. A series of input primitives can be entered.

The maximum number of items is specified as the parameter MAX. The number of items is returnes in the parameter NUMB.

If NUMB is returned with value 0, the current event report does not contain any CHOISE reports.

Errors:
1. The DATA-ARRAY is not large enough.

8.7.4. Get LOCATOR Data

GETLOC (MAX, DATA-ARRAY, NDIM, NUMB)

If a LOCATOR is associated with a trigger it's position data (N-dimensional) is read every time the trigger is activated.

The LOCATOR data in the current event report is returned in the DATA-ARRAY in world coordinates.

A series of input primitives can be entered. The maximum number of items is specified by the parameter MAX. The number of items is returned in the parameter NUMB.
NDIM is the LOCATOR dimension, which is output from the device.

If NUMB is returned with value 0, the current event report does not contain any LOCATOR reports.

Errors:
1. The DATA-RAY is not large enough.

8.7.5. Get VALUATOR Data

GETVAL (MAX, DATA-ARRAY, NUMB)

If a VALUATOR is associated with a trigger, it's value is read every time the trigger is activated. The VALUATOR data is returned from the current event report in the DATA-ARRAY.

A serie of input primitives can be entered. The maximum number of items is specified as the parameter MAX. The number of items entered is returned in the parameter NUMB.

If NUMB is returned with value 0, the current event report does not contain any VALUATOR reports.

Errors:
1. The DATA-ARRAY is not large enough.

8.8. Setting input device characteristics

The logical input devices have various characteristics that can be altered by the application program. These characteristics are properties that are independent of the enable/disable status of the device. If the device is enabled, the effect of the function below is immediate; otherwise the effect occurs the next time the device is enabled. These functions may be invoked at any time.

8.8.1. Set Echo

ECHCTL (DEVICE-ID, ON/OFF)
This function switch the echo on/off. The parameter ON/OFF contains a value indicating whether the echo for this device is to be turned on or off. All devices are initialized with echo on.

Errors:
1. DEVICE-ID is invalid.

8.8.2. Set Echo Position

ECHPOS (DEVICE-ID, ECHO-X, ECHO-Y, ECHO-TYPE)

This function set the echo start position on the surface. ECHO-X and ECHO-Y are values in world coordinates indicating a position on the view surface at which the echo is to appear. For some physical devices, it is unnecessary to specify a position for the echo. For example, the echo for a PFK used as a set of CHOISE is normally implemented using the lamps in the hardware, and a PICK device normally echoes by blinking the picked segment. In such cases, the ECHO-X and ECHO-Y values are ignored. IDIGS uses a default position on the view surface for echoing if it is necessary.

ECHO-TYPE is an implementation defined Echo type such as blink, rubber band etc.

Errors:
1. DEVICE-ID is invalid.
2. ECHO-X or ECHO-Y specify positions outside normalized device coordinate space.
3. ECHO-TYPE is not defined.

8.8.3. Set Echo Surface

ECHSUR (DEVICE-ID, WORKSTATION-ID)

This function select the workstation where the echo is to appear.
The echo for the specified device will be performed on the specified view surface. The default view surface for each input device is determined by the implementation.

Errors:
1. DEVICE-ID specify a non-existent device.
2. The specified view surface is not initialized.

8.8.4. Set PICK

SETPIC (DEVICE-ID, POS-ARRAY, NDIM)

This routine provides the possibility for the program to position a software or hardware comparator (Hit-mechanism). If the comparator (when positioned), hits a detectable segment a PICK-event will occur.

Errors:
1. DEVICE-ID is not a valid PICK device.

8.8.5. Set LOCATOR

SETLOC (DEVICE-ID, LOC-ARRAY, NDIM)

The LOCATOR cursor is set to the N-dimensional (NDIM) position specified by LOC-ARRAY in world coordinates. If the LOCATOR is not enabled, the specified coordinate position is utilized the next time it is enabled. Note that this is especially useful when a lightpen with a tracking cross is used as a LOCATOR. Other physical devices, such as a data tablet, behave in such a way that the initial position is immediately replaced by the actual position of the stylus. The default LOCATOR position is origo.

Errors:
1. DEVICE-ID does not specify a valid LOCATOR device.
2. LOC-ARRAY specify a position outside normalized device coordinate space.
8.8.6. Set VALUATOR

SETVAL (DEVICE-ID, INIT-VAL, LOW-VAL, HIGH-VAL)

Each VALUATOR has an application defined initial value and range associated with it. The initial value is set to INIT-VAL at the time the device is enabled. As with the LOCATOR position, the initial value setting will be immediately replaced when used with certain physical devices such as fixed range control dials. The range is specified to be between LOW-VAL and HIGH-VAL. The values from the physical device will be scaled linearly to the specified range. The default range is 0,1 and the default initial value is 0.

Errors:
1. DEVICE-ID is not a valid VALUATOR device.
2. LOW-VAL is greater than HIGH-VAL.
3. INIT-VAL does not lie in the range defined by LOW-VAL and HIGH-VAL.

8.8.7. Set CHOISE

SETCHO (DEVICE-ID, BUTTON-NUMB, PROMPT-SWITCH)

The physical devices that are most commonly used to implement the logical CHOISE device normally has a built-in prompting capability. For example, the lamps which are found on most PFKS are often used as prompts to identify which buttons are current. This function allows the application program to invoke this prompting capability. If the value of PROMPT-SWITCH is 0, prompting for the specified button is turned off. Any other value of PROMPT-SWITCH indicates that prompting for that button is turned on. Furthermore, a non-zero value may be used by the device driver to control the prompting. For example, for function key boxes with back-lit lamps, the value of PROMPT-SWITCH could be used to choose which prompt string should be projected onto the button.

1. DEVICE-ID is not a valid CHOISE device.
2. BUTTON-NUMB is not valid.
9. CONTROL

9.1. Overview

IDIGS has several functions for initialization and control of the different processes in the system.

9.2. System Initialization and Termination

9.2.1. Initialization of IDIGS

IDIGON (I, J, K, L, M)

This function is used if initialization or reinitialization of the system is needed. IDIGON is also used for checking the capabilities of the given IDIGS system. The parameters I, J, K, L, M states the lowest level possible for running the program.

I is the level of output features I = 1, 2, 3, 4
J is the level of input features J = 1, 2, 3
K is the level of dimensionality K = 1, 2
L is the type (calligr. or raster) L = 1, 2
M is number of active workstations M = 1, 2, 3

If the function is called, it must be the first IDIGS function called during application program initialization. It guarantees that the IDIGS is in a standard state with the default settings of all the IDIGS parameters established. The particular operations performed by this function will vary from one operating system to another. Whatever initialization is needed in that operating environment will be performed.

Errors:
1. IDIGS has already been initialized.
2. The level of the system is too low.
9.2.2. Termination of IDIGS

IDIGOF ( )

This function is used if explicit termination of the system is necessary.

The function terminates all devices close all files and releases all other resources being used by IDIGS.

Errors:
None

9.3. Device initialization and selection

Most interactive application programs are designed to respond to the inputs of a single operator and a single view surface. However, some application programs control more than one view surface. As an example, an operator might direct the same output to a microfilm recorder and a refresh display. As a second example, a user wants to save the pseudocode for a picture which is generated on a screen in order to redraw it later on a hardcopy device. In this case the "pseudodevice" is looked upon as a special view surface.

Multiple view surfaces are supported by IDIGS within a restricted conceptual framework. For each segment there is only one image which may be displayed on multiple view surfaces. No extra level of naming is introduced to allow differentiation between the separate versions of the image by the logical view surfaces on which they appear.

Multiple view surfaces are supported by the BACDEV function. The image of a segment appears on every view surface that is selected at the time the segment is created. IDIGS accept only input from the main device such that the background device is passive. A view surface cannot be selected or deselected (with BACDEV) while there is an open segment. All selected surfaces display the segment's image as defined
by the viewing transformation in effect when the segment was created. Also, when the segment's dynamic attributes are changed the image displayed on all its view surface changes appropriately.

9.3.1. Initialize Device

\textit{NITDEV (IUNIT, COLOUR)}

This function must be called before any graphics elements are produced. It performs whatever operating system functions are required to obtain access to the logical device IUNIT and to initialize that surface. The surface is cleared and prepared for graphic output.

IUNIT is the logical name of the device.

COLOUR specifies the background colour for the device. Only one device can be selected as the "current device" at a time. Such that a new call to SELDEV deselects the "current device" before it selects the new one. However several devices may be initialized (NITDEV) simultaneously. Echo and prompts may be routed to initialized devices even if they are not selected.

A call to NITDEV implies a call to SELDEV such that the last device which is initialized is selected as the current device if SELDEV is not explicitly activated in the mean time.

Errors:
1. The device has already been initialized (since it was last terminated).
2. No physical device is associated with IUNIT.

9.3.2. Select device

\textit{SELDEV (IUNIT)}

This function selects the logical device IUNIT for all sub-
sequent graphic output and input until the device is deselected with a DESDEV or RLSDEV function call.

A call to NITDEV implies a call to SELDEV.

Errors:
1. The specified device has not been initialized.
2. The specified device is already selected.
3. A segment is open.

9.3.3. Deselect device

DESDEV (IUNIT)

This function deselects the logical device IUNIT for all subsequent graphic output and input until the device is selected with a SELDEV for both input and output or SELBAC for only output.

Errors:
1. The specified device has not been initialized
2. The specified devices is already selected
3. A segment is open

9.3.4. Release device

RLSDEV (IUNIT)

This function terminates access to the logical device IUNIT. Segments whose images appear on only this device are deleted. Segments whose images appear on multiple surfaces are marked to indicate that their images no longer appear on this surface. The view surface may or may not be cleared, depending on the particular implementation and operating system constraints.

Errors:
1. The specified device has not been initialized.
9.3.5. Select background device

SELBAC (IUNIT)

This function select IUNIT as a BACKGROUND device, such that the image on the current device also appear on the background device. The background device is passive i.e. no input is accepted from a background device. More than one BACKGROUND device may be current simultaneously. IUNIT must be initialized before SELBAC is activated.

Errors:
1. There is no current device
2. IUNIT is not initialized
3. A segment is open

9.3.6. Deselect background device

DESBAC (IUNIT)

This function deselect the background device for all subsequent graphic output until it is reselected with a SELDEV or SELBAC function.

Errors:
1. IUNIT is not a background device

9.4. Device Control

9.4.1. Clear device

CLRDEV (IUNIT, COLOUR)

This function will erase all the currently created display elements from both the device and the buffer(s) of a buffered device and create an entirely new display with COLOUR as the background colour.

For hard copy devices a call to this routine provides a
new picture making surface (sheet of paper).

Errors:
1. IUNIT is invalid
2. COLOUR is not valid

9.4.2. Special control function

IDIGS provides a special control function to solve particular problems like Delay of segments, Batch of updates, Hidden line, Hidden surface etc.

9.4.2.1. Begin Special Function

BGNFUN (ITYPE)

This function denotes the beginning of a special function. Within a BGNFUN, visual changes to the picture may or may not take place, but the state of the picture's corresponding segments, and of other aspects of IDIGS (e.g., attribute value settings), will be kept current. However, at the end of a special function, all the specified operations are guaranteed to have taken place.

ITYPE = 0: Immediate visibility (Default)

All primitives are sent individually or output buffer by output buffer (size is implementation dependent) to the current device.

ITYPE = 1: Delay of Segments

All primitives of a segment are collected and sent to the work station when the segment is closed.

ITYPE = 2: Batch of Updates

There are two primary motivations for providing a capability for batching updates. First, for storage tube devices, it enables an application program to minimize the number of
screen erasures. If several segments are to be deleted at one time, the screen can be erased once for all the segments rather than once per segment. Second, for satellite systems over networks, the "end batch of updates" function guarantees that the operating system output message buffer has been sent to the satellite. This is typically used just before waiting for input.

**ITYPE = 3: Hidden Line Removal**

If this mode is set, nothing is displayed until the ENDFUN function occurs, at which time the Hidden Line algorithm is initiated.

**ITYPE = 4: Hidden Surface Removal**

If this mode is selected, nothing is displayed until the ENDFUN function occurs, at which time the Hidden Surface algorithm is initiated.

**Errors:**
1. There has been no ENDFUN since the last BGNFUN.

**9.4.2.2. End Special Function**

**ENDFUN ( )**

This functions denotes the end of a Special Function to the picture. All the changes to the picture that are specified prior to this function call are guaranteed to have taken place before control returns from this function. This function can be invoked at any time after a BGNFUN function call.

**Errors:**
1. There has been no corresponding BGNFUN.

**9.4.2.3. Enable hidden lines removal**

**HIDLIN (POL-DIRECTION, SOLID)**
This routine must be called before the hidden line removal may be activated. The parameters are used by the algorithm to speed up its performance.

POL-DIRECTION indicates the direction of polygons to be drawn.

+1: Polygons are counterclockwise
+1: Polygons are clockwise
0: Direction of polygons is not known

SOLID indicates if all polygons describes solid figures. If solid figures are drawn and the polygon direction is known, polygons facing away from the viewport may be identified and quickly removed. SOLID = 1 if solid figures are drawn, 0 else.

HIDLIN may be called multiple times to change options.

Errors:
1. POL-DIRECTION or SOLID is invalid

9.4.2.4. Enable hidden surface removal

HIDSUR (POL-DIRECTION, SOLID)

This routine must be called before the hidden surface removal may be activated. Parameters and logical functioning is the same as HIDLIN.

9.4.3. The R, B, G color model

Most existing color devices use an RGB (red, green, blue) specification, with a certain number of discrete intensity values for each color component. These colors are added together to give the color which appears on the view surface.

The color model used when RGB color specification is used is the color cube shown in the figure 1. Black is represented by (0, 0, 0) on the color cube and white by (1, 1, 1). The three additive primaries red, green and blue are along
the axes from the origin. Cyan, magenta, and yellow occupy the other three vertices of the color cube. All colors are specified by giving their red, green and blue components. (Some hardcopy devices specify color with the subtractive primaries cyan, magenta, and yellow rather than the additive primaries red, green, and blue. This suggests a color model consisting of a color cube with white at the origin and cyan, magenta, and yellow along the axes from the origin. The transformation between the two models is very simple: \( \text{RGB} = 111 - \text{CMY} \).)

![RGB color space](image)

9.4.3.1. Define Color Index table

**DEFCOL** (WORKSTATION-ID, I, R, G, B) \( 1 \leq I \leq N \)

This function defines the entry I in the workstation index table, with the given color parameters R, G, B. The R, G, B is rounded by IDIGS to the nearest color displayable on the currently selected view surface. If the value of the current color attribute has the index I, then the current color attribute is also changed.

By this mechanism the color attribute is semidynamic (i.e. the table content may change dynamically while the index is static).

Errors:
1. Invalid color index
2. Invalid color attribute value
3. No current view surface
4. Invalid WORKSTATION-ID
9.4.4. Define Intensity Index table

DEFINT (WORKSTATION-ID, I, INTENSITY)

This function defines entry I in the workstation Intensity index table. The intensity is rounded by IDIGS to the nearest intensity displayable on the currently selected view surface.

If the value of the current intensity attribute has the index I, then the current intensity attribute is also changed.

By this look up table the intensity attribute is semidynamic (i.e. the table content may change, but the index is static).

Errors:
1. Invalid intensity index
2. Invalid intensity attribute value
3. No current view surface
4. Invalid WORKSTATION-ID

9.4.5. Define Hatching Index table

DEFHTC (WORKSTATION-ID, I, DIRECTION, DISTANCE)

This function defines the workstation hatching pattern whose index is I.

DIRECTION specifies the angle and DISTANCE the distance between the lines in the hatching pattern given in world coordinates.

Errors:
1. Invalid hatching index.
2. Invalid WORKSTATION-ID

9.4.6. Define Workstation pen Index table

DEFPEN (WORKSTATION-ID, I, LINTYPE, LINWIDTH, COLOUR)
This function defines an entry in the workstation pen table who's index is I. The primitive attributes for Linetype, Linewidth and colour are entered into the table. These will override the current attributes for linetype, linewidth and colour when routine TOOL(I) is called.

Errors:
1. Invalid pen Index
2. Invalid workstation ID

9.4.7. Send message to workstation

MESSAG (Workstation-ID, MESSAGE, LENGTH)

This function may be used e.g. to demand a certain paper quality on a plotter.

MESSAGE is a text to display at an installation dependant location on the workstation viewport. LENGTH is the length of the text. If a negative length is given, the message is terminated by '*'.

Errors:
1. Invalid workstation-ID
2. The message contains an illegal character.

9.5. Viewing Control

In many applications, the entire picture is known to fit on the view surface without clipping. In other applications, all of the clipping may occur in the modelling system. In either case, considerable processing in IDIGS can be avoided if the application program notifies IDIGS that the objects described to the system need not be clipped.

9.5.1. Clipping control

CLICTL (TYPE)
TYPE = 0  All clipping turned off (default)
TYPE = 1  Clipping against the window only
TYPE = 2  Clipping against all planes
TYPE = 3  Clipping against the window and the front plane
TYPE = 4  Clipping against the window and the back plane
TYPE = 5  Clipping against the front and back plane
TYPE = 6  Clipping against the front plane only
TYPE = 7  Clipping against the back plane only

This function is used to enable or disable clipping against the window in the view plane and the front and back clipping plane. When clipping is turned off, objects described to IDIGS are not checked for possible clipping. If a line or a portion of a line appears outside the normalized device coordinate space, the appearance of that line is undefined. Wraparound may occur. When clipping is turned on, objects described to IDIGS are clipped, according to TYPE (when necessary). The default state is all clipping turned off.

Errors:
1. A segment is open.
2. TYPE is invalid

9.6. Polygon Control

There is four major reasons for introducing polygons into IDIGS.

First the convenience of drawing polygons directly, secondly hidden line/surface removal, third the requirement for hatching, greyscale and colouring and fourth the trend of raster graphics hardware coming out of its traditional use in image processing and process control into the world of interactive graphics. This has occurred because of rapid change in the economics of large scale integration and because of user demand for colour and shaded areas which is relatively easy to provide with raster graphics. In many applications, raster graphics is a natural complement to line-drawing graphics, while in other applications, raster graphics will sooner or later replace line-drawing graphics. Still other applications,
never considered practical with line-drawing graphics for reasons of either economics or display aesthetics, suddenly become feasible with raster graphics.

9.6.1. Set Polygon mode

Polygon drawing is performed in one of the following modes: Contour, plain, shaded or patterned.

POLMOD (TYPE)

This function sets the current polygon type.

TYPE = 0  CONTOUR (Default)
A contour polygon is drawn simply by its contour.
The current line drawing attributes is in effect for the contour drawing.

TYPE = 1  PLAIN
A plain polygon is filled with one colour, a grayscale or hatching dependent on the capability of the current device. The filling is defined by the function POLIND

TYPE = 2  SHADED
The interior of the polygon image is filled with pixel values of the vertices of the polygon. (The vertex indices are specified by using POLVTX.) The interpolation is a two step process. First the edge values are determined by linear interpolation along the edges using the vertex values. Next the interior values are determined by linear interpolation from edge to edge along the scan lines that cover the interior of the polygon image. If the current device is not a raster device, the polygon will be hatched by an installation specified pattern.

TYPE = 3  PATTERNED
Each pixel of the interior of the polygon image
is assigned an index value by copying the value from one element of an array of index values. The Pixel array describing the pattern is made current by the function POLPAT. If the current device is not a raster device, the polygon will be hatched by an installation specified pattern.

Errors:
1. TYPE is invalid
2. A segment is open
3. For TYPE = 2 and 3 the current view surface is not raster.

9.6.2. Set PLAIN Polygon filling index

POLIND (INDEX).

INDEX is an entry into a table which content depends on the capability of the device:

1) If colour is available, INDEX is an entry into the colour table (R, G, B) defined by the function DEFCOL.

   The colour attribute is by this mechanism semidynamic. (The index table may change dynamically, while the index is static.)

2) If greyscale is available INDEX is an entry into an intensity table which is defined by the function DEFINT.

3) Otherwise INDEX is an entry into a hatching table, defined by the function DEFHTC. Default value is INDEX = 1.

Errors:
1. A segment is open.
2. INDEX is out of range.
9.6.3. Set SHADED polygon vertex index

POLSHD (VERTEX-INDEX-ARRAY, NUMB)

This function sets the array containing the colour or intensity indices of the vertices of a polygon. The sequence of the vertices in the VERTEX-INDEX-ARRAY correspond to the list of vertices given by the corresponding POLYGON command.

NUMB is the number of indices in the array. If NUMB is less than the number of vertices in the corresponding polygon the sequence of indices will repeat until all vertices have been assigned an index.

The default array is of size NUMB = 1 with index value = 1.

If the current view surface is not a raster device, the polygon will be hatched in an installation specified pattern.

Errors:
1. NUMB < 1
2. Invalid index values.

9.6.4. Set PATTERNED polygon pixel array

POLPAT (PIXEL-INDEX-ARRAY, COLOUMNS, ROWS, XPOS, YPOS)

This function sets the current PIXEL-INDEX-ARRAY which is to be used for filling the corresponding polygons.

The array is rectangular with size given by COLOUMNS and ROWS.

The mapping is a view surface operation where each pixel of the pixel array is mapped onto a pixel on the view surface such that adjacent pixels in the pixel array become adjacent pixels on the view surface. The origin of the pixel array may be positioned to an arbitrary position in NDC space using the parameters XPOS and YPOS.
For the purpose of displaying, the pixel array is conceptually replicated in both the X and Y dimensions until each pixel of the polygon image has a corresponding pixel array element. In this manner, the interior of a polygon will always be entirely filled.

Default is a one by one pixel array with index value = 1. If the current view surface is not a raster device, the polygon will be hatched in an installation specified pattern.

Errors:
1. Invalid index values in the index array.
2. COLUMNS < 1 or too big
3. ROWS < 1 or too big
10. STORAGE OF GRAPHICAL INFORMATION

10.1. Overview

There is a number of reasons for including storage of graphical information: Use of Retained segments for image manipulation has already been specified. In order to work with retained segments, space must be allocated for storage of the code. Stored segments can also be utilized for increasing the efficiency of the graphics system by inserting them into the picture generation pipeline during a graphic session.

IDIGS provides aids for space allocation for retained segments and storing of Pseudosegments. Pseudosegments consists of graphics elements coded in device independent format. Storing of the Pseudosegments may be done in two ways:

a) Storing of the Pseudosegments in primary storage for reuse in the same run.

But even if the user has a large amount of primary storage at some point of time this is less than he actual needs.

b) To help solving this problem, the provision for storing picture segments in buffers on secondary storage (LIBRARY) has been made.

The other natural use of secondary storage buffers is to keep frequently needed picture segments or Pseudo segments in pre-stored form so they can be quickly retrieved and inserted into the pipeline. This also saves the time that otherwise would be required to re-create the picture segments during every run of the program.

In nearly all implementations it is expected that the secondary storage buffers for Pseudosegments will be on direct access devices. Thus the routines provided for defining the secondary storage buffers closely parallel the routines
for defining devices.

Although space allocation is in general language dependent, IDIGS suggests a standardized allocation mechanism which can be used for most languages.

For the purpose of long term filing of graphics information, IDIGS provides also an access to a sequential file called the IDIGS Metafile. The IDIGS Metafile may also contain non-graphical data.

The IDIGS Metafile can also be used for:

- Transferring of "pictures" between subsystems

- Transportation of graphical information from one site to another (e.g. by magnetic tape or on a graphical network).

The IDIGS Metafile can be thought of as a passive output device. Input from the Metafile will be processed by a special IDIGS interpreter under the control of the application program. Reading and interpreting the Metafile has the same effect as if the same IDIGS functions where called, that previously generated the file.
10.2. Buffer handling

Primary storage must be allocated for retained segments, display files and Pseudosegments and pixel arrays.

10.2.1. Initialize primary buffer

NITBUF (ARRAY, LENGTH, TYPE)

This routines allocate and initialize a buffer ARRAY with length LENGTH for the purpose given by TYPE.

TYPE specify what kind of buffer is initialized.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Retained segment (default)</td>
</tr>
<tr>
<td>1</td>
<td>Pseudo buffer</td>
</tr>
<tr>
<td>2</td>
<td>Display file</td>
</tr>
<tr>
<td>3</td>
<td>PIXEL ARRAY</td>
</tr>
</tbody>
</table>

A call to NITBUF implies a call to SELBUF.
Errors:
  1. TYPE is not valid

10.2.2. Select Buffer

SELBUF (ARRAY)

Select the buffer ARRAY as the current buffer.

The array becomes the "current" buffer for the purpose specified by TYPE.

Only one buffer of each TYPE can be "current" simultaneously. When a new buffer of the same TYPE is selected, the current one is closed.

Errors:
  1. ARRAY is not initialized

10.2.3. Release Buffer

RLSBUF (ARRAY)

The buffer ARRAY is released and may be used for other purposes.

Errors:
  1. ARRAY is not initialized

10.3. Secondary Storage Buffers (Library)

The library is used mainly for three purposes.

1) Provide additional storage for picture segments.

2) Keep frequently needed picture segments in pre-stored form so they can be quickly retrieved for use.

3) Long term storing of pictures segments.
10.3.1. Initialization of library

NITLIB (UNIT)

This routine initiates a secondary buffer for use by IDIGS, or clear an already existing one.

IUNIT is the name of the file or data set on secondary storage. A call to NITLIB implies a call to SELLIB.

Errors:
1. IUNIT is invalid.

10.3.2. Select library

SELLIB (IUNIT)

This routine select a particular secondary storage buffer.

IUNIT is the name of the file or data set on secondary storage.

A selected library becomes the "current library".

Errors:
1. IUNIT is not initialized or made available.

10.3.3. Release library

RLSLIB (IUNIT)

This routine disconnects the secondary storage buffer from the current program. IUNIT is the name of the file or data set on secondary storage.

Errors:
1. IUNIT is not initialized or made available.
10.4. Library handling

The library most recently selected by NITLIB or SELLIB will be the current library. There are two methods of storing segments in a picture library.

1. The pseudo picture segments are built directly in the current picture library if a current library is selected (NITLIB, SELLIB).

2. The picture segments may be copied from a primary picture buffer into the current library (SAVPIC).

10.4.1. Save picture segments in the library

SAVPIC (OLD, NEW)

This routine copies the picture segment OLD in the current pseudo buffer into the current library and gives the segment identifier the name NEW which must be unique for the library.

Errors:
  1. There is no current library
  2. There is no current buffer
  3. There is no segment named OLD

10.4.2. Restore picture segment from the library

RESPIC (OLD, NEW)

This routine restore the picture segment OLD from the current library into the current pseudo buffer and gives it the name NEW.

Errors:
  1. There is no current library
  2. There is no current buffer
  3. There is no segment named OLD
10.5. Pseudo Picture Segments

Pseudo-picture segments are graphic elements stored in device independent form. By specifying the Pseudo Device driver to NITDEV, SELDEV or BCKDEV the programmer requests loading and initialization of the pseudo-picture segment driver, which will store pseudo code in the pseudo Picture Segment if there is a current pseudo buffer or library.

Until a different device is selected all picture segments will be stored in the current pseudo buffer or library in a device independent format. At a later time the pseudo-picture segments can be copied or inserted in new picture segments for any display device.

The content of the pseudo segments are the primitives of the segment transformed according to the current window/viewport transformation and clipped if clipping was arranged for during segment generation.

10.5.1. Insert pseudo segment from buffer

    INSERT (IDENT)

In order to make the graphic elements stored in the pseudo segment IDENT in the current buffer available for display, this routine must be activated.

Errors:
1. There is no current buffer
2. There is no pseudo segment called IDENT

10.5.2. Insert pseudo segment from library

    INSLIB (IDENT)

In order to make the graphical elements stored in the pseudo-segment IDENT on the current Library UNIT available for display, this routine must be activated.
Errors:
1. There is no current library
2. There is no pseudo segment IDENT

10.5.3. Copy a frame from library

COPDEV (IUNIT)

Instead of inserting segment by segment from a library, this functions copies all the segments in the library UNIT directly into the system after the viewing transformations.

Errors:
1. There is no library named UNIT

10.5.4. Reference to pseudo segments

REFER (IDENT2)

Instead of physically including all elements from a subsegment IDENT2 when creating a new pseudo picture segment IDENT1, just a symbolic reference to IDENT2 may be included by a call to this routine.

The reference to IDENT2 is not evaluated until IDENT1 is referenced by INSERT or INSLIB.

If a segment is INSERT'ed, all segments must reside in the current buffer. If the segment is INSLIB'ed, all segment must reside on the current library file.

Errors:
1. There is no segment open
2. There is no segment named IDENT2. (This error message will not be activated until INSERT or INSLIB is activated).
10.6. Metafile handling

Meta file is a sequential storage of graphics and non-graphics information. Functions are provided to both read from and write to metafiles.

For output a meta file driver is used. This driver is controlled by NITDEV, SELDEV etc. and behaves like a passive output device (e.g. plotter), but in addition records of user specified data can also be included.

For input the meta file is interpreted by a special metafile interpreter under the control of the application program. See figure 10.1.

10.6.1. Initialize the metafile

NITFIL (IUNIT, ACCESS-MODE)

This routine initiates a metafile for input or output and makes it available for IDIGS.

IUNIT is the name of the file or data set. A call to NITFIL implies a call to SELFIL.

If ACCESS-MODE 'read' is given the metafile is initiated for input to the metafile interpreter. For output the ACCESS-MODE may be either 'write' or 'append'.

Errors:
1. IUNIT is not valid.
2. ACCESS-MODE is not valid.

10.6.2. Select meta file

SELFIL (IUNIT)

This function select a particular file and makes it the current meta file for input or output.
Errors:
1. IUNIT is not valid.
2. IUNIT is not initialized.

10.6.3. Release meta file

RLSFIL (IUNIT)

This routine disconnects the meta file from the current program. IUNIT is the name of the meta file.

Errors:
1. IUNIT is not initialized or made available

10.6.4. Write user information to the meta file

WRIFIL (TYPE, LENGTH, DATA-ARRAY)

writes a record of type TYPE and length LENGTH with user data (DATA-ARRAY) to the current meta file.

No segment can be open when user information is written to the meta file.

Errors:
1. There is no current meta file
2. There is an open segment

10.6.5. Read Record type from the meta file

RECTYP (TYPE, LENGTH)

IDIGS inspect the type and length of the next record in the metafile and delivers TYPE and LENGTH back to the application program.

Errors:
1. No record left
2. Content of record is illegal
3. There is no current meta file

10.6.6. Read next record from metafile

REAREC (DATA-ARRAY, LENGTH)

Reads a user information record (DATA-ARRAY) with length LENGTH from the current meta file.

Errors:
1. No record left
2. Content of record illegal
3. There is no current meta file

10.6.7. Read and interpret next records from the metafile

INTPRT (N)

The next N records (which should be segments) are read and interpreted by IDIGS. The effect is the same as that of the function calls would be that previously wrote the records. If N<0 all subsequent graphics records will be read and interpreted.

Errors:
1. Less than N records left
2. Content of record illegal
3. There is no current meta file

10.6.8. Skip next N records on the metafile

SKPREC (N)

IDIGS positions the record pointer in the Meta file after the next N records.

Errors:
1. No records left
2. There is no current meta file
10.6.9. Backspace N records on the meta file

**BKSREC (N)**

IDIGS position the record pointer in the meta file before the previous N records.

**Errors:**
1. N > number of records read
2. There is no current metafile.

---

**FILE**

<table>
<thead>
<tr>
<th>HEADER</th>
<th>PICTURE1</th>
<th>PICTURE2</th>
<th>— — —</th>
<th>PICTUREN</th>
<th>EOI</th>
</tr>
</thead>
</table>

**PICTURE**

<table>
<thead>
<tr>
<th>HEADER</th>
<th>RECORD1</th>
<th>RECORD2</th>
<th>— — —</th>
<th>RECORDM</th>
<th>EOF</th>
</tr>
</thead>
</table>

**RECORD**

<table>
<thead>
<tr>
<th>TYPE</th>
<th>LENGTH</th>
<th>DATA</th>
</tr>
</thead>
</table>

**SEGMENT**

**OR**

**USER INFO**

Fig. 10.2. The structure of metafile.

10.7. Subpicture

Many displayed pictures include repeated symbols such as logic circuit elements, architects' symbols, chemical pipework symbols and so forth. It is convenient to define each such symbol only once and be able to jump to this definition whenever a reference to the symbol occurs in the display file. This is made possible by the use of subpictures. A subpicture is a single segment of display code that can be used several times in the display file without being recreated. A subpicture is assumed used mainly for advanced refresh displays.

The possibilities for manipulating the subpicture (transformations etc.) are dependent on the current device. However,
translation should always be included. The attribute handling
is as if the subpicture was a sequence of instructions within
a segment. The action when screen boundary violation occur
is implementation dependent. A subpicture and a segment
can not be open simultaneously.

10.7.1. Begin subpicture

BGNSUB (NAME)

This function starts the creation of a subpicture named
NAME. All subsequently generated display code is stored
in this subpicture until ENDSUB is encountered.

Errors:
1. NAME is not unique
2. There is an open segment

10.7.2. End subpicture

ENDSUB

The currently open subpicture becomes closed.

Closing the subpicture has no effect on the visibility of
the subpicture which must be explicitly linked to the display
file to become visible (CALSUB).

Errors:
1. There is no open subpicture.

10.7.3. Delete subpicture

DELSUB (NAME)

The subpicture named by the parameter NAME is deleted.

Errors:
1. No subpicture named NAME exist.
10.7.4. Call subpicture

CALSUB (NAME)

The subpicture identified with NAME is linked to the display file. The subpictures can be nested.

Errors:
1. No subpicture named NAME exist.
2. No picture segment is open.
11. INQUIRY

11.1. Overview

Inquiry functions allow an application program to determine the status of the picture generation process, the status of IDIGS and the physical capabilities of the particular device being used. The requested information is copied into the supplied output parameter. Inquiry functions provide a debugging capability, the capability for an application program to tailor its interactions to the devices being used and an aid for a portable way of getting around problems due to device insufficiency (i.e. by simulation). To achieve a drawing in correct scale, the metric equivalent of the normalized coordinates can be inquired from the device and thus allowing a proper setting of the viewport.

11.2. Inquire Current Position

INQPOS (X, Y, Z)

The current drawing position is copied into the variables X, Y and Z.

Errors:
None

11.3. Inquire Viewing Parameters

INQVIW (VIEWING-ARRAY)

The values of the current viewing parameters are copied into the VIEWING-ARRAY in the following order:

NDC-Viewport
Window
View Plane Normal
View Reference Point
View Up Vector
View Distance
Front Distance
Back Distance

Errors:
None

11.3.1. Inquire Transformation

INQTRN (TYPE, NDIM, TRAN-MATRIX)

This function is used to inquire the current viewing transformation, model transformation, and composit viewing transformation matrix (model * viewing).

TYPE is the type of matrix to return:
1 = viewing transformation
2 = model transformation
3 = composite viewing transformation

NDIM is the number of dimensions (2D or 3D) of the matrix to be returned

TRAN-MATRIX is a 3*2 matrix if 2D was required or a 4*4 matrix if 3D.

Errors:
1. Type or NDIM is illegal
2. Information was lost because a 2D matrix was Inquired when a 3D matrix was set

11.3.2. Inquire Viewing position

INQIMG (NDC-X, NDC-Y, NDC-Z, X, Y, Z)

The world coordinates corresponding to the position (NDC-X, NDC-Y, NDC-Z) are calculated using the current viewing transformation. These coordinates are then written into the variables X, Y, and Z. If Z-MIN = Z-MAX, NDC-Z is ignored, and since no information about depth can be inferred, the arbitrary assumption is made that the desired world coordinate position lies on the front clipping plane.
11.3.3. Inquire Projection Parameters

INQPRS (TYPE, PX, PY, PZ)

The current projection parameters are copied into the variables: TYPE (parallel or perspective), PX, PY, PZ which is the direction of projection (for parallel projection) or centre of projection (for perspective projection).

Errors:
None

11.3.4. Inquire Workstation transformation

INQWTR (WORKSTATION-ID, NDIM, WSWIND, VPORr)

The current values of the workstation transformation of the specified workstation is returned.

NDIM marks if 2D or 3D data is wanted.

Values of the NDC-window (WSWIND) and workstation viewport (VPORr) are returned.

Errors:
1. WORKSTATION-ID is invalid
2. NDim is invalid

11.4. Inquire max. surface area

INQMTR (WORKSTATION-ID, LENGTH, WIDTH, HEIGHTH)

The maximum size of the WORKSTATION-ID device surface are given in meters by meters and copied into the parameters LENGTH, WIDTH and HEIGHTH.

This values may be used for setting the viewport to achieve a drawing in correct scale.
Errors:
1. WORKSTATION-ID is invalid or not current.

11.5. Inquire segment attributes

INQSEG (SEG-NUMBER, MAX-IARR-LEN, IARR, MAX-RARR-LEN, RARR)

The current values of the segment attributes of the specified picture segment (SEG-NUMBER) is copied into the output arrays as follows:

IARR : Visibility, Detectability, Highlighting, Image Transformation type,
Number of workstations associated to segment,
List of associated workstations

RARR : Image Transformation parms

Errors:
None

11.6. Inquire Output Status

INQOST (MAX-LENGTH, OUTPUT-STATUS-ARRAY)

The values of the following output status parameters are copied into the array OUTPUT-STATUS-ARRAY:

current clipping mode
workstation
segment name of open segment, if any
special control function mode
number of selected workstations
list of selected workstations

Errors:
None
11.7. Inquire Input Device Information

11.7.1. Inquire Echo parameters

INQECO (DEVICE-ID, WORKSTATION-ID, POS-X, POS-Y)

The values of the DEVICE-ID's Echo parameters: surface and Echo position are copied in the parameters WORKSTATION-ID, POS-X, POS-Y).

   Errors:
   1. DEVICE-ID is invalid or not initialized.
   2. WORKSTATION-ID is invalid.

11.7.2. Inquire Device Association

INQASO (DEVICE-ID, MAX-NUMB, DEVICE-ARRAY, NUMB)

Identifiers for input devices and triggers associated with the specified device (DEVICE-ID) are copied into the DEVICE-ARRAY in hierarchical order starting with the TRIGGER for DEVICE-ID and followed by identifiers and triggers level by level. MAX-NUMB is the maximum number of entries. NUMB is the actual number of entries.

   Errors:
   1. DEVICE-ID is invalid
   2. MAX-NUMB is too small

11.7.3. Inquire Device Enabled

INQENA (DEVICE-ID, FLAG)

Returns the enabled/disabled status of DEVICE-ID

   FLAG = 0: Device disabled
   FLAG = 1: Device enabled

   Errors:
1. DEVICE-ID is invalid

**11.8. Inquire Output Capabilities**

**INQOUT** (WORKSTATION-ID, OUTPUT-ARRAY, NUMB)

The following output capabilities of the specified workstation are copied into the array OUTPUT-ARRAY:

- raster capability
- colour capability
- intensity capability
- line width capability
- line style capability
- character capability
- highlighting capability
- image transformation capability
- resolution of view surface (e.g., number of addressable elements per centimeter for each dimension)

Included in the capability information is some encoding of how each capability is provided, e.g., hardware, software simulation, or not provided.

NUMB is the number of entries.

Errors:
1. Workstation-ID is invalid

**11.8.1. Inquire Input Capabilities**

**INQINP** (WORKSTATION-ID, DEVICE-ID, INPUT-ARRAY, NUM)

The input capabilities of this implementation of the workstation are copied into the array INPUT-ARRAY as follows:

Name and type of physical input devices, name and type of simulated input devices, timing capability for EVENT.
NUM is the number of entries.

Errors:
1. WORKSTATION-ID is invalid
2. DEVICE-ID is invalid or not initialized.

11.9. Inquire Library content

INQLIB (LIBRARY-ID, MAX, SEGMENT-ARRAY, NUM)

The name of all segments in the library LIBRARY-ID is copied into the array SEGMENT-ARRAY with maximum number of segments MAX and actual number of segments NUM.

Errors:
1. LIBRARY-ID is invalid

11.9.1. Inquire Library-Segment generation environment

INQSG (LIBRARY-ID, SEGMENT-NAME, WINDOW, CLIPPING)

The window used for generating SEGMENT-NAME which resides on LIBRARY-ID is copied into the array WINDOW and the flag CLIPPING is set to 0 if clipping was not in effect and 1 if clipping was in effect during the segment generation.

Errors:
1. LIBRARY-ID does not exist
2. SEGMENT-NAME does not exist

11.10. Inquire Raster

11.10.1. Inquire Pixel Array

INQPIX (COLUMNS, ROWS)

The size of the current Pixel Array is copied into the parameters COLUMNS and ROWS.

Errors:
1. No Pixel array is current.

11.10.2. Inquire Polygon Mode

INQPOL (TYPE)

The current polygon mode is given in the parameter TYPE.

TYPE = 0: CONTOUR
TYPE = 1: PLAIN
TYPE = 2: SHADED
TYPE = 3: PATTERNED
12. ERROR HANDLING

Several different types of errors may be generated by an application program as it uses IDIGS. Examples include programming errors such as an attempt to delete a segment which does not exist and environment errors such as running out of buffer space.

Improper or inconsistent use of IDIGS requires the system to inform the user of his misdeeds. The input parameters passed to IDIGS is checked to see if they fit the definition of the subroutine called, and to see if they are consistent with the routines previously called. If these necessary conditions are not met, an error message is printed on the error message file. Default error message file is the Echo Surface.

IDIGS provides the following ERROR-REPORT to tell the user what the error was and what the system did about it.

1. Error number - indicates the type of error which occurred.

2. IDIGS subroutine code - each user callable IDIGS routine has a unique number. The name of the corresponding routine may be found in the index of routines.

3. Parameter number - the number of the parameter in the parameter list which has the bad value. The parameter numbers are counted from left to right.

4. Error Severity Code - Error severity as determined by IDIGS when the error is detected.

5. Number of IDIGS subroutines called since the beginning of the program.

6. Bad value - value of integer parameter which is in error.
The IDIGS error severity codes indicate the action IDIGS has taken (will take) to avoid or recover from an error condition:

1: Warning
   Program execution continues.

2: Recovery from error
   Recovery action is taken and program execution continues.

3: Severe error
   IDIGS cannot perform requested action - run will be stopped if critical value is not changed by the user.

4: Terminal error
   IDIGS will stop run.

If there are sufficient space available, all the error messages will be given in clear text (map number to message table).

12.1. Users error routine

Instead of just passively sitting by and watching his program die, the user may have the error information passed to a routine which he has provided.

ERROUT is the name of the routine which IDIGS will pass control to when error is detected. The user supplied routine will be called by:

```
ERROUT (IARR)
```

The ERROR-REPORT passed to "ERROUT" contains the follow integer data:
IARR(1) = Error Code
IARR(2) = IDIGS Subroutine Code
IARR(3) = Number of Parameter causing Error
IARR(4) = Error Severity
IARR(5) = Number of IDIGS Routines called
IARR(6) = Bad value

The ERROUT routine may attempt to recover from error by changing the severity code (IARR(4)) to something less severe so the program will continue and supply a new integer parameter value (IARR(6)) to be used in place of the bad value which caused the error. ERROUT may also communicate with the terminal user so that he may change the state of the program to allow it to continue processing.

Errors:
None

12.2. Users own error messages

To obtain an error message from his own routine the user may include the routine.

ERRLOG (IARR)

where "IARR" contains the ERROR-REPORT described above.

Errors:
None

12.3. Initialize Error file

ERRFIL (ERROR-FILE)

The error message file ERROR-FILE is initialized and all subsequent ERROR-REPORTS goes to this file. Default error message file is the Echo Surface.

Errors:
12.4. Report most recent error

ERRPT (ERROR-REPORT)

This function copies the error report for the most recently detected error into ERROR-REPORT, and flushes the report from IDIGS. If no error has been detected since IDIGS was initialized or since ERRPT was last invoked, the function will return a null error report.

If the ERROR-REPORT is not automatically written out, the simplest way for an application program to use the error-handling capabilities of IDIGS is through repeated invocation of the ERRPT function. Whenever it is appropriate for the program to check whether an error has occurred, this function can be invoked. The error report can then be compared with the null report to determine if an error has occurred.

Errors:
None
REFERENCES


## Alfabetical list of routines in IDIGS

<table>
<thead>
<tr>
<th>Routine</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSOC (DEVICE-ID, TRIGGER-ID, ATTRIBUTE, NEW-DEVICE-ID)</td>
<td>62</td>
</tr>
<tr>
<td>ATTACH (TOOL-ID, DEVICE-CLASS, DEVICE-ID)</td>
<td>57</td>
</tr>
<tr>
<td>AXON (DX, DY, DZ)</td>
<td>38</td>
</tr>
<tr>
<td>BGNFUN (ITYPE)</td>
<td>78</td>
</tr>
<tr>
<td>BGNPIC (NAME)</td>
<td>27</td>
</tr>
<tr>
<td>BGNSUB (NAME)</td>
<td>101</td>
</tr>
<tr>
<td>BKSREC (N)</td>
<td>100</td>
</tr>
<tr>
<td>CALSUB (NAME)</td>
<td>102</td>
</tr>
<tr>
<td>CFONT (IFONT)</td>
<td>18</td>
</tr>
<tr>
<td>CHARS (STRING, LENGTH)</td>
<td>13</td>
</tr>
<tr>
<td>CJUST (ITYPE)</td>
<td>23</td>
</tr>
<tr>
<td>CLICTL (TYPE)</td>
<td>86</td>
</tr>
<tr>
<td>CLRDEV (IUNIT, COLOUR)</td>
<td>77</td>
</tr>
<tr>
<td>COLOUR (INDEX)</td>
<td>17</td>
</tr>
<tr>
<td>COPDEV (UNIT)</td>
<td>96</td>
</tr>
<tr>
<td>CPATH (IDIR)</td>
<td>22</td>
</tr>
<tr>
<td>CPLANE (DX, DY, DZ)</td>
<td>22</td>
</tr>
<tr>
<td>CQUAL (IQUAL)</td>
<td>19</td>
</tr>
<tr>
<td>CSIZE, (XSIZE, YSIZE)</td>
<td>21</td>
</tr>
<tr>
<td>CSPACE, (XSPACE, YSPACE)</td>
<td>20</td>
</tr>
<tr>
<td>CSPAC3 (XSPACE, YSPACE, ZSPACE)</td>
<td>21</td>
</tr>
<tr>
<td>DASALL (DEVICE-ID)</td>
<td>64</td>
</tr>
<tr>
<td>DASDEV (DEVICE-ID, OLDE-DEVICE-ID, TRIGGER-ID)</td>
<td>63</td>
</tr>
<tr>
<td>DEFocol (WORKSTATION-ID, I, R, G, B)</td>
<td>81</td>
</tr>
<tr>
<td>DEFHTC (WORKSTATION-ID, I, DIRECTION, DISTANCE)</td>
<td>82</td>
</tr>
<tr>
<td>DEFINT (WORKSTATION-ID, I, INTENSITY)</td>
<td>82</td>
</tr>
<tr>
<td>DEFPEN (WORKSTATION-ID, I, LINTYPE, LINWIDTH, COLOUR)</td>
<td>82</td>
</tr>
<tr>
<td>Delpic (NAME)</td>
<td>28</td>
</tr>
<tr>
<td>DELSUB (NAME)</td>
<td>101</td>
</tr>
<tr>
<td>DESBAC (IUNIT)</td>
<td>77</td>
</tr>
</tbody>
</table>
DESDEV (IUNIT) 76
DETACH (DEVICE-ID, TOOL-ID) 58
DETPIC (IDENT, ISWITCH) 31
DISABL (DEVICE-ID) 65
DRAW (X-ARR, Y-ARR, N, ITYPE) 11
DRAWR (DX-ARR, DY-ARR, N, ITYPE) 11
DRAW3 (X-ARR, Y-ARR, Z-ARR, N, ITYPE) 11
DRAWR3 (DX-ARR, DY-ARR, DZ-ARR, N, ITYPE) 11
ECHCTL (DEVICE-ID, ON/OFF) 70
ECHPOS (DEVICE-ID, ECHO-X, ECHO-Y, ECHO-TYPE) 70
ECHSUR (DEVICE-ID, WORKSTATION-ID) 70
ENABLE (DEVICE-ID) 65
ENDFUN ( ) 79
ENDPIC 28
ENDSUB 101
ERRFIL (ERROR-FILE) 113
ERRLOG (IARR) 113
ERROUT (IARR) 112
ERRPT (ERROR-REPORT) 114
EVENT (TIME, DEVICE-ID) 61

FLUSH ( ) 66
FLUSHD (DEVICE-ID) 65

GETCHO (MAX, DATA-ARRAY, NUMB) 68
GETLOC (MAX, DATA-ARRAY, NDIM, NUMB) 68
GETPIC (MAX, DATA-ARRAY, NUMB) 68
GETTXT (MAX, DATA-ARRAY, NUMB) 68
GETVAL (MAX, DATA-ARRAY, NUMB) 69

HIDLIN (POL-DIRECTION, SOLID) 79
HIDSUR (POL-DIRECTION, SOLID) 80
HILPIC (IDENT, ISWITCH) 30

IDIGOF ( ) 74
IDIGON (I, J, K, L, M) 73
IMTRA (IDENT, ARR2)  31
IMTRA3 (IDENT, ARR3)  32
INQASO (DEVICE-ID, MAX-NUMB, DEVICE-ARRAY, NUMB)  107
INQECO (DEVICE-ID, WORKSTATION-ID, POS-X, POS-Y)  107
INQENA (DEVICE-ID, FLAG)  107
INQIMG (NDC-X, NDC-Y, NDC-Z, X, Y, Z)  104
INQINP (WORKSTATION-ID, DEVICE-ID, INPUT-ARRAY, NUM)  108
INQLIB (LIBRARY-ID, MAX, SEGMENT-ARRAY, NUM)  109
INQMTR (WORKSTATION-ID, LENGTH, WIDTH, HEIGHT)  105
INQOST (MAX-LENGTH, OUTPUT-STATUS-ARRAY)  106
INQOUT (WORKSTATION-ID, OUTPUT-AAY, NUMB)  108
INQPIX (COLUMNS, ROWS)  109
INQPOL (TYPE)  110
INQPOS (X, Y, Z)  103
INQPR5 (TYPE, PX, PY, PZ)  105
INQSEG (SEG-NUMBER, MAX-IARR, LEN, IARR, MAX-RARR-LEN, RARR)  106
INQSG (LIBRARY-ID, SEGMENT-NAME, WINDOW, CLIPPING)  109
INQTRN (TYPE, NDIM, TRAN-MATRIX)  104
INQV1W (VIEWING-ARRAY)  103
INQWTR (WORKSTATION-ID, NDIM, WSWIND, VPOR)  105
INSERT (IDENT)  95
INSLIB (IDENT)  95
INTENS (INDEX)  17
INTPRT (N)  99

LINE (X, Y)  10
LINER (DX, DY)  10
LINE3 (X, Y, Z)  10
LINER3 (DX, DY, DZ)  10
LINTYP (N)  16
LINWID (IWIDTH)  16
MARK (NR, X-ARRAY, Y-ARRAY, N)  12
MARKR (NR, DX-ARRAY, DY-ARRAY, N)  12
MARK3 (NR, X-ARRAY, Y-ARRAY, Z-ARRAY, N9) 12
MARKR3 (NR, DX-ARRAY, DY-ARRAY, DZ-ARRAY, N) 12
MESSAG (WORKSTATION-ID, MESSAGE, LENGTH) 83
MODTR (TRANS-MATRIX) 47
MODTR3 (TRANS-MATRIX) 47
MOVE (X, Y) 9
MOVER (DX, DY) 9
MOVE3 (X, Y, Z) 9
MOVER3 (DX, DY, DZ) 9
MSIZE (M-UP, M-WIDTH) 17

NAME (N) 18
NDCVP (XMIN, XMAX, YMIN, YMAX) 43
NDCVP3 (XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX) 42
NITBUF (ARRAY, LENGTH, TYPE) 91
NITDEV (IUNIT, COLOUR) 75
NITFIL (IUNIT, ACCESS-MODE) 97
NITLIB (IUNIT) 93

PERS (DX, DY, DZ) 38
PLIN (X-ARR, Y-ARR, N) 10
PLINR (DX-ARR, DY-ARR, N) 10
PLIN3 (X-ARR, Y-ARR, Z-ARR, N) 10
PLINR3 (DX-ARR, DY-ARR, DZ-ARR, N) 10
POLIND (INDEX) 86
POLMOD (TYPE) 85
POLPAT (PIXEL-INDEX-ARRAY, COLUMN, ROWS, XPOS, YPOS) 87

POLSHD (VERTEX-INDEX-ARRAY, NUMB) 87
POLY (X-ARR, Y-ARR, N) 10
POLYR (DX-ARR, DY-ARR, N) 10
POLY3 (X-ARR, Y-ARR, Z-ARR, N) 10
POLYR3 (DX-ARR, DY-ARR, DZ-ARR, N) 10

REALOC (LOCATOR-ID, P^S-ARRAY, NDIM) 64
REAREC (DATA-ARRAY, LENGTH) 99
REAVAL (VALUATOR-ID, VALUE) 64
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECTYP (TYPE, LENGTH)</td>
<td>98</td>
</tr>
<tr>
<td>REFER (IDENT2)</td>
<td>97</td>
</tr>
<tr>
<td>ENAME (OLD-SEGMENT, NEW-SEGMENT)</td>
<td>29</td>
</tr>
<tr>
<td>RESPIC (OLD, NEW)</td>
<td>94</td>
</tr>
<tr>
<td>REQCHO (DEVICE-ID, CHOICE-NUMBER)</td>
<td>59</td>
</tr>
<tr>
<td>REQLOC (DEVICE-ID, DATA-ARRAY, NDIM)</td>
<td>60</td>
</tr>
<tr>
<td>REQPIC (DEVICE-ID, DATA-ARRAY, POSITION)</td>
<td>58</td>
</tr>
<tr>
<td>REQTXT (DEVICE-ID, MAX- DATA-ARRAY, NUMB)</td>
<td>59</td>
</tr>
<tr>
<td>REQVAL (DEVICE-ID, VALUE)</td>
<td>60</td>
</tr>
<tr>
<td>RLSBUF (ARRAY)</td>
<td>92</td>
</tr>
<tr>
<td>RLSDEV (IUNIT)</td>
<td>76</td>
</tr>
<tr>
<td>RLSFIL (IUNIT)</td>
<td>98</td>
</tr>
<tr>
<td>RLSLIB (UNIT)</td>
<td>93</td>
</tr>
<tr>
<td>SAVATR (IARRAY, RARRAY)</td>
<td>25</td>
</tr>
<tr>
<td>SAVPIC (OLD, NEW)</td>
<td>94</td>
</tr>
<tr>
<td>SELBAC (IUNIT)</td>
<td>77</td>
</tr>
<tr>
<td>SELBUF (ARRAY)</td>
<td>92</td>
</tr>
<tr>
<td>SELDEV (IUNIT)</td>
<td>75</td>
</tr>
<tr>
<td>SELFIL (UNIT)</td>
<td>97</td>
</tr>
<tr>
<td>SELLIB (UNIT)</td>
<td>93</td>
</tr>
<tr>
<td>SETATR (ARRAY)</td>
<td>25</td>
</tr>
<tr>
<td>SETCHO (DEVICE-ID, BUTTON-NUMB, PROMPT-SWITCH)</td>
<td>72</td>
</tr>
<tr>
<td>SETLOC (DEVICE-ID, LOC-ARRAY, NDIM)</td>
<td>71</td>
</tr>
<tr>
<td>SETPIC (DEVICE-ID, POS-ARRAY, NDIM)</td>
<td>71</td>
</tr>
<tr>
<td>SETVAL (DEVICE-ID, INIT-VAL, LOW-VAL, HIGH-VAL)</td>
<td>72</td>
</tr>
<tr>
<td>SKPREC (N)</td>
<td>99</td>
</tr>
<tr>
<td>TOOL (TOOL-NUMBER)</td>
<td>18</td>
</tr>
<tr>
<td>VDEPTH (FRONT-DISTANCE- BACK-DISTANCE)</td>
<td>40</td>
</tr>
<tr>
<td>VISPIC (IDENT, ISWITCH)</td>
<td>30</td>
</tr>
<tr>
<td>VPDIST (VIEW-DISTANCE)</td>
<td>37</td>
</tr>
<tr>
<td>VREFPT (X,Y,Z)</td>
<td>37</td>
</tr>
<tr>
<td>VPNORM (DX, DY, DZ)</td>
<td>37</td>
</tr>
<tr>
<td>VPORT (WIDENT, XMIN, XMAX, YMIN, YMAX)</td>
<td>46</td>
</tr>
<tr>
<td>VPORT3 (XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX)</td>
<td>45</td>
</tr>
</tbody>
</table>
VUP (DX, UP, DY, UP)  41
VUP3 (DX-UP, DY-UP, DZ-UP)  38

WINDW (XMIN, XMAX, YMIN, YMAX)  41
WINDW3 (UMIN, UMAX, VMIN, VMAX)  39
WRIFIL (TYPE, LENGTH, DATA-ARRAY)  98
WSWIN (WIDENT, XMIN, XMAX, YMIN, YMAX)  46
WSWIN3 (WIDENT, XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX)  45
ERRORS

3.1. 1. No segment is open
3.2. 1. No segment is open
3.4. 1. No segment is open
   2. N < 2
3.5. 1. No segment is open
   2. ITYPE has not a valid value
3.6. 1. No segment is open
   2. N < 0
3.7. 1. No segment is open
   2. The string contains an illegal character

4.1.14. 1. Invocation when no open segment
         2. Invalid attribute value

5.1.1. 1. There already is an open segment
        2. A retained segment named NAME already exists
        3. 32767 < NAME < 0
        4. Illegal combination is viewing parameters

5.1.2. 1. There is no open segment
5.1.3. 1. No retained segment named NAME exists
5.1.4. 1. There is an existing retained segment named NEW-NAME
         2. There is no retained segment named OLD-SEGMENT

6.1.4. 1. Invalid (i.e., non-existing) segment
         2. Invalid attribute value
         3. Invalid image transformation for the segment

7.2.1. 1. A segment is open
7.2.2. 1. A segment is open
         2. DX, DY, and DZ are all zero: no direction can be established
7.2.3. 1. A segment is open
7.2.4. 1. A segment is open
2. DX, DY, and DZ are all zero: the projection direction cannot be determined

7.2.5. 1. A segment is open

7.2.6. 1. A segment is open
2. DX-UP, DY-UP, and DZ-UP are all zero: no view up direction can be established

7.2.7. 1. A segment is open
2. UMIN is not less than UMAX, or VMIN is not less than VMAX

7.2.8. 1. A segment is open
2. FRONT-DISTANCE is greater than BACK-DISTANCE, so that the back clipping plane is in front of the front clipping plane

7.3.1. 1. A segment is open
2. DX-UP and DY-UP are both zero: no view up direction can be established

7.3.2. 1. A segment is open
2. XMIN is not less than XMAX, or YMIN is not less than YMAX

7.4.1. 1. View volume corner(s) are outside of normalized device coordinate space
2. A segment is open
3. XMIN is not less than XMAX, or YMIN is not less than YMAX, or ZMIN is not less than ZMAX

7.4.2. 1. Viewport extent is outside of normalized device coordinate space
2. A segment is open
3. XMIN is not less than XMAX, or YMIN is not less than YMAX

7.5.1. 1. Window extent is outside of NDC
2. A segment is open
3. XMIN is not less than XMAX, or YMIN is not less than YMAX, or ZMIN is not less than ZMAX
4. WIDENT is not a valid workstation

7.5.2. 1. The viewport exceeds the display surface
2. A segment is open
3. XMIN is not less than XMAX, or YMIN is not less than YMAX, or ZMIN is not less than ZMAX
4. WIDENT is not a valid workstation identifier

7.5.3. 1. Window extent is outside of NDC
2. A segment is open
3. XMIN is not less than XMAX, or YMIN is not less than YMAX
4. WIDENT is not a valid workstation

7.5.4. 1. The viewport exceeds the display surface
2. A segment is open
3. XMIN is not less than XMAX, or YMIN is not less than YMAX
4. WIDENT is not a valid workstation identifier

8.3.1. 1. TOOL-ID or DEVICE-CLASS is invalid
8.3.2. 1. DEVICE-ID is invalid

8.4.1.1. 1. DEVICE-ID is invalid
8.4.1.2. 1. DEVICE-ID is invalid
2. DATA-ARRAY is not large enough to hold the string entered by the operator

8.4.1.3. 1. DEVICE-ID is not valid
8.4.1.4. 1. DEVICE-ID is invalid
8.4.1.5. 1. DEVICE-ID is not valid
8.4.2.1. 1. DEVICE-ID is invalid
8.4.3.1. 1. DEVICE-ID is active
2. The specified association already exist
3. The DEVICE-ID is invalid
4. The TRIGGER-ID is invalid

8.4.3.2. 1. DEVICE-ID is invalid
8.4.3.3. 1. DEVICE-ID is invalid

8.5.1. 1. LOCATOR-ID is invalid
2. The specified LOCATOR device is not enabled

8.5.2. 1. VALUATOR-ID is invalid
2. The specified VALUATOR device is not enabled

8.6.1. 1. Specified device already enabled
2. DEVICE-ID invalid
8.6.2. 1. Specified device is not enabled  
       2. DEVICE-ID invalid

8.6.3. 1. DEVICE-ID is not valid

8.7.1. 1. The DATA-ARRAY is not large enough

8.7.2. 1. DATA-ARRAY was not large enough to hold the string  
       entered by the operator

8.7.3. 1. The DATA-ARRAY is not large enough

8.7.4. 1. The DATA-ARRAY is not large enough

8.7.5. 1. The DATA-ARRAY is not large enough

8.8.1. 1. DEVICE-ID is invalid

8.8.2. 1. DEVICE-ID is invalid  
       2. ECHO-X or ECHO-Y specify positions outside normalized  
          device coordinate space
       3. ECHO-TYPE is not defined

8.8.3. 1. DEVICE-ID specify a non-existent device
       2. The specified view surface is not initialized

8.8.4. 1. DEVICE-ID is not a valid PICK device

8.8.5. 1. DEVICE-ID does not specify a valid LOCATOR device  
       2. LOC-ARRAY specify a position outside normalized  
          device coordinate space

8.8.6. 1. DEVICE-ID is not a valid VALUATOR device  
       2. LOW-VAL is greater than HIGH-VAL
       3. INIT-VAL does not lie in the range defined by  
          LOW-VAL and HIGH-VAL

8.8.7. 1. DEVICE-ID is not a valid CHOOSE device  
       2. BUTTON-NUM is not valid

9.2.1. 1. IDIGS has already been initialized  
       2. The level of the system is too low

9.3.1. 1. The device has already been initialized (since  
       it was last terminated)
       2. No physical device is associated with IUNIT

9.3.2. 1. The specified device has not been initialized  
       2. The specified device is already selected
       3. A segment is open
9.3.3. 1. The specified device has not been initialized
        2. The specified device is already selected
        3. A segment is open
9.3.4. 1. The specified device has not been initialized
9.3.5. 1. There is no current device
        2. IUNIT is not initialized
        3. A segment is open
9.3.6. 1. IUNIT is not a background device
9.4.1. 1. IUNIT is invalid
        2. COLOR is not valid
9.4.2.1. 1. There has been no ENDFUN since the last BGNFUN
9.4.2.2. 1. There has been no corresponding BGNFUN
9.4.2.3. 1. POL-DIRECTION or SOLID is invalid
9.4.3.1. 1. Invalid color index
        2. Invalid color attribute value
        3. No current view surface
        4. Invalid WORKSTATION-ID
9.4.4. 1. Invalid intensity index
        2. Invalid intensity attribute value
        3. No current view surface
        4. Invalid WORKSTATION-ID
9.4.5. 1. Invalid hatching index
        2. Invalid WORKSTATION-ID
9.4.6. 1. Invalid pen Index
        2. Invalid WORKSTATION-ID
9.4.7. 1. Invalid WORKSTATION-ID
        2. The message contains an illegal character
9.5.1. 1. A segment is open
        2. TYPE is invalid
9.6.1. 1. TYPE is invalid
        2. A segment is open
        3. For TYPE = 2 and 3 the current view surface is not raster
9.6.2. 1. A segment is open
        2. INDEX is out of range
9.6.3.  1. NUMB < 1
        2. Invalid index values

9.6.4.  1. Invalid index values in the index array
        2. COLUMNS < 1 or too big
        3. ROWS < 1 or too big

10.2.1. 1. TYPE is not valid
10.3.1.  1. IUNIT is invalid
10.3.2.  1. UNIT is not initialized or made available
10.3.3.  1. UNIT is not initialized or made available
10.4.1.  1. There is no current library
        2. There is no current buffer
        3. There is no segment named OLD
10.4.2.  1. There is no current library
        2. There is no current buffer
        3. There is no segment named OLD
10.5.1.  1. There is no current buffer
        2. There is no pseudo segment called IDENT
10.5.2.  1. There is no current library
        2. There is no pseudo segment IDENT
10.5.3.  1. There is no library named UNIT
10.5.4.  1. There is no segment open
        2. There is no segment named IDENT2.  (This error
           message will not be activated until INSERT or
           INSLIB is activated)
10.6.1.  1. IUNIT is not valid
        2. ACCESS-MODE is not valid
10.6.2.  1. IUNIT is not valid
        2. IUNIT is not initialized
10.6.3.  1. IUNIT is not initialized or made available
10.6.4.  1. There is no current meta file
        2. There is an open segment
10.6.5.  1. No record left
        2. Content of record is illegal
        3. There is no current meta file
10.6.6. 1. No record left
         2. Content of record illegal
         3. There is no current meta file
10.6.7. 1. Less than N records left
         2. Content of record illegal
         3. There is no current meta file
10.6.8. 1. No records left
         2. There is no current meta file
10.6.9. 1. N > number of records read
         2. There is no current meta file
10.7.1. 1. NAME is not unique
         2. There is an open segment
10.7.2. 1. There is no open subpicture
10.7.3. 1. No subpicture named NAME exist
10.7.4. 1. No subpicture named NAME exist
         2. No picture segment is open

11.3.1. 1. Type of NDIM is illegal
         2. Information was lost because a 2D matrix was
            Inquired when a 3D matrix was set
11.3.4. 1. WORKSTATION-ID is invalid
         2. NDIM is invalid
11.4.   1. SURFACE-NAME is invalid or not current
11.7.1. 1. DEVICE-ID is invalid or not initialized
         2. WORKSTATION-ID is invalid
11.7.2. 1. DEVICE-ID is invalid
         2. MAX-NUMB is too small
11.7.3. 1. DEVICE-ID is invalid
11.8.   1. WORKSTATION-ID is invalid
11.8.1. 1. WORKSTATION-ID is invalid
         2. DEVICE-ID is invalid or not initialized
11.9.   1. LIBRARY-ID is invalid
11.9.1. 1. LIBRARY-ID does not exist
         2. SEGMENT-NAME does not exist
11.10.1. 1. No Pixel array is current
COMPOUND LOGICAL INPUT DEVICE

KETIL BØ
SIU
7001 Trondheim, Norway
COMPOUND LOGICAL INPUT DEVICES

Current design of Man/Machine Communication systems suffer from a mis-match of levels between input and output. Output is handled at a high level, while input is regarded from a much lower level.

A concept for building logical high level input devices is proposed in order to overcome some of the problems arising from this lack of symmetry. The concept includes a mechanism for better utilization of available physical input devices and prepares for interaction with multidimensional machine tools and robots.

INTRODUCTION

The interaction software is the communication bridge between the user and the computer. It handles information which has been organized by the mind of the user in a for him natural way and translates it into actions which can be accepted by the input devices of the computer. It also accomplishes transfer of information in the opposite direction.

In order to properly design interactive software, it is important to understand the fundamentals of the communication process in general and the human/computer interaction in particular.

The basic goal of a communication system is to communicate information, ideas and meanings from one body to another.

If we look into the basic mechanisms of such systems, we find that they involve transfer of primitive components across an interface.

To convey the meaning requires a process of analysis through semantic, syntactic and lexical phases to define the stream of primitive components. The transfer of the stream through a suitable medium and a process of synthesis to re-assemble the meaning.
through lexical, syntactic and semantic phases in the receiving body.

Fig. 1. Human/computer communication.

In the natural man/man communication processes such as conversation or correspondence, we find symmetry between the input and the output both in primitive components and in the processing of these components. Furthermore the processing is controlled by "familiar" rules supported by a given context.
In computer graphics the language is not a familiar mode of expression such as spoken or even written words, but rather one of images and actions such as stylus movements, button pushes and light pen indications. Despite of the obvious differences in the manifestation of the communication process in these two examples, the basic structure on corresponding levels is surprisingly equal.

However, one of the problems which frequently occurs in human/machine interaction is that the dialog is performed on different level for output and input. The reason for this is partly due to the fact that output is fairly well understood with well defined primitive components and standardized tools for building them together into complex structures, while input is still in a consolidation phase. The main effort on input has been put on to the primitive components and there exist hardly any equivalent to the output tools for building compound structures which is necessary for symmetry in the dialogue.

1. FUNDAMENTALS OF INTERACTION

Human/computer interaction may be classified into three main sub-groups:

1. Input of raw data
2. Data manipulation
3. Presentation of results

At the present only the last one is pretty much settled and will not be discussed further in this paper.

Manipulation of data and results as "real time" interactions between the human and the computer may be improved considerably by better understanding of the interaction process both from the users - and the system designers point of view.
Symmetry is one of the important parameters for an efficient, natural and unambiguous dialogue.

The distinction between knowing and doing is another:

"The flavor of interaction is to achieve agreement on what to do (knowing), then pass the control to task performance (doing). When the task is completed, or when disagreement occurs in task performance (errors, unexpected events etc.), then control is returned to knowing, so that agreement may be reestablished before doing is resumed" (2).

The condition for a symmetric dialogue is access to equally powerful tools for handling interaction on the lexical, syntactic and semantic level both for output and input. (See fig 3)
Fig. 3. Levels of control on a human/computer communication session.

The lexical level is related to the binding of specific hardware capabilities to primitive components of the input and output "languages". For input this means associating actions or sequences of actions using input devices as words of the "language". For output the primitive elements such as lines and characters as well as attributes such as line style and colour are used to encode the context.

The syntactic level is related to the way "words" are put together to form "sentences".

For input this means decisions of the action sequence. For output it means the display layout such as how the primitive elements are organized on the display surface, prompt position, error messages and menu control.

The semantic level is related to the meaning of the "language". For input it has to do with the selection of information which
the user conveys to the computer. In output it is concerned with how the information will be displayed to the user exemplified by the presentation form of geometrical objects (wire frame, hidden surface, shading etc.).

The efficiency of entering raw data into the computer is at the present far behind both interaction and data presentation. The procedure for inputting raw data to the computer is traditionally both slow and error prone. It may take days to prepare input to a program which needs only minutes or seconds to solve the actual problem and present the results.

One way to make this process more effective is to build high-level input tools for supporting the operator and utilizing more advanced input equipments.

2. HARDWARE CONSIDERATION

The flora of hardware input devices is growing wild and leads to a number of problems both for compatibility and portability of graphic programs. These problems are simplified by introducing logical input devices which can be mapped on to the physical input devices (1). However, if we look to an actual graphical workstation, the number of physical devices is limited.

It is therefore also important to maximize the utilization of available physical devices. A mechanism for dynamic attachment of physical and logical devices by the application program should therefore be provided.

3. THE INPUT CONCEPT

3.1. HISTORY

The concept of Compound Logical Input Devices is built on a considerable basis of earlier work (3, 4, 5, 6, 7, 8), the most influential in this respect is the work by Wallace (4).

Wallace enunciated the concept of virtual input devices as a mean whereby interactive graphics application programs could be made independent from the peculiarities of the input devices of a particular terminal, and then become portable.
The idea was that the application programmer had available a range of virtual input devices, the only aspect of which was the type of value they returned. It became, however, soon evident that the pure virtual device concept was inadequate. Virtual devices needed other aspects controlling details of the operator interface such as prompt and echoing. This logical device concept formed, however, the basis for the GSPC Core System (6).

By closer examination also the logical device concept of the "Core" has some severe shortcomings:

- The lack of possibility to utilize a given configuration of physical devices in a dynamic way.

- The lack of a clear distinction between the concepts of:
  
  - creating a logical device using particular types of hardware
  - prompting an operator for input
  - echoing an operator's action
  - acknowledging an operator's generation of an event

- The difficulty of relating any of these "output concepts" to logical input devices

- The lack of powerful input tools to construct high level input devices

Moreover the trend in system development is integration and extent of computer graphics into related areas such as verification of N/C tapes, simulation of tool path in Computer Aided Manufacturing (CAM), interaction with machine tools where handling of more than three dimensions is necessary, robotics etc. For such purposes the input handling in a graphical software package must be able to handle N-dimensional information and include important parameters such as time and distance. Tools for supporting advanced facilities like programmable cursors and hit mechanisms should also be provided.

It seems clear that these problems are all related to deficiencies in the underlying input models which are currently available.
3.2. THE CONCEPTUAL INPUT MODEL

In our input model a set of logical input devices are defined and controlled by the application program. The primitive input devices are divided into seven disjoint classes. Logical devices are dynamically mapped to whatever available physical device which most conveniently are implementing the desired function. The primitive input devices are low level and defined to allow efficient mapping to commonly available physical input devices and to form building blocks for higher level input devices.

If a device is to be used for input, it must first be attached, that is, a connection must be made between the identifier known to the application program and the external name(s) for the physical device(s). Once this connection is made, the device becomes accessible and interactions may take place using the device. The device may also be detached, breaking the connection between the identifier and the physical devices. The device is then inaccessible until it is re-attached.

There are three generally accepted ways in which application programs can obtain data from logical input devices; they are known as different modes of operation (8):

- In SAMPLE mode the application program invokes a function to obtain the current logical data value from the device.

- In EVENT mode the operator's actions create event reports containing data from the actual devices at a specific moment in time. The system preserves these reports in one or more event queue(s), the contents of which the application program can process at its convenience.

- In REQUEST mode the application program invokes a function permitting the operator to adjust the logical data value of the device and then indicate that the value is satisfactory; the function waits until this has been done and then returns the value.

All compound logical input devices are considered to be created by an implementation even if the mapping from the physical device to the logical device is quite direct. The compound logical input device has five major parts:
1. An identifying number

2. An associated physical device or devices from which the input actually came, including a TRIGGER with a TRIGGER-ATTRIBUTE which actually causes the dispatch of the input data to the program.

3. A value or a set of values generated by the logical device.

4. A prompt to the operator to announce the program's readiness for logical input.

5. An echo to the operator to announce the recognition of his action.

6. An acknowledgement to announce the recognition of a generated event.

Primitive event devices may be regarded as compound devices with the device itself as the trigger.

The value or set of values generated by a logical event device is the result of a process which is active only when the particular device is taking part in the interaction. The value generating process of a logical sample device is conceptually active all the time, and the application may obtain the data values from the process whenever it pleases.

The trigger part of a compound logical device respond to changes in the state of the physical input devices either by changing its internal state or by firing. When the trigger of a logical input device fires, it sends a message to the processes for the devices which is associated with the trigger.

Several logical input devices may have their trigger in common.

The firing of the trigger; signals activation to all processes associated with that particular trigger. It is a requirement that a single operator action cause the firing of no more than one trigger. When a device process receives a signal from the trigger, the action it takes depends upon the device mode as follows:
- In REQUEST mode, the device process returns its data value to the application, and then goes to sleep again.

- In SAMPLE mode, signals from triggers are always ignored. The application may obtain the data value from the device process whenever it pleases.

- In EVENT mode, the device process attempts to add an event record, containing its identification and data value, to the appropriate input queue. Other device processes using the same trigger may also be attempting to add records to the same queue at the same time. The resulting group of simultaneous event records must either all be added to the queue, or none must be added to the queue. If the device process fails to add its events to the queue, input queue overflow must be reported for that queue.

Groups of simultaneous event records are marked as such; when an event record is dequeued, the application must be able to discover whether more events in the same group remain in the queue.

When accessible to the application program, a logical input device has certain characteristics or attributes that distinguish it, in a general fashion, from other logical input devices of the same input class. Depending on the particular graphics system, some attributes will be under application control, while others will have been fixed by the implementor. Some of the attributes of logical input devices are:

- Current mode of operation.

- How the implementation simulates the logical device using physical devices.

- How the operator is informed that a device process has come into existence, and thus that its associated physical devices are available for manipulation. This is called the prompt.

- How the operator is informed of the logical device's logical data value. This is called the echo.

- How the operator is informed of a significant firing of the input device's trigger; this is called acknowledgment. A
significant trigger firing is one satisfying a REQUEST function invocation, or adding events to the queue.

- An initial value, of the type appropriate to the class, for use by the device process when it comes into existence.

- A switch turning echo on or off.

The attributes may also contain extra information used, for example, by particular simulation, prompting and echoing techniques.

It is now possible to outline the sequence of operations that corresponds to a specific logical input device taking part in an interaction:

- A device process is created for the device, and its value is set to the initial value in the device's state.

- If the trigger for the device is not in existence, it is created.

- The operator is prompted for input, using the selected technique.

- If the echo switch in the device's state is on, echoing is commenced, using the selected technique.

- As the operator manipulates the physical input devices, the trigger may fire, causing the appropriate actions to take place, depending on the device's mode.

- If a trigger firing is significant, it is acknowledged.

- Eventually, either because in REQUEST mode the trigger fires, or because the device leaves SAMPLE or EVENT mode, the device process goes to sleep.

Such an approach provides a facility for building up compound input devices like DIGITIZING, GROUPING, etc.

To make this compound input devices efficient, the data handling is done on a low level in the program hierarchy in order to avoid spending time popping up and down in the program hierarchy for
every single action. This may be specially time consuming when working on a satellite system or in a graphic network.

Input primitives provide the following general facilities:

- ATTACH and DETACH physical and logical devices;
- queueing/dequeueing of events caused by event generating devices;
- sampling of current settings of sampled devices;
- association of devices, such that one device will serve as a TRIGGER for another device.

To each TRIGGER is associated a TRIGGER-ATTRIBUTE specifying the threshold of the trigger. The values of all associated devices in event mode are read when the TRIGGER is activated. The set of values generated is saved on an event queue and the control is not returned to the program until the TRIGGER on the highest level is activated.

3.3. INPUT PRIMITIVES

Event devices are used by the operator to signal an event to the application program.

An event is caused by a logical "action" taken by the operator or program by "pulling" the TRIGGER an amount specified by the application program. An event caused by a logical action will in event mode result in the creation of an event report. The event report is saved on the event queue when the TRIGGER on the highest level is activated.

Sampled devices have values which may be sampled by the application program any time during a session.

The primitive logical input devices in our model are:

PICK - identify a graphics segment and a primitive within a segment

TEXT - provide alphanumeric information
CHOICE - PROVIDE LOGICAL "buttons" or combination of "buttons" (i.e., the ability to choose among alternatives)

DISTANCE - cause an event when a cursor, etc. has moved a certain distance

TIMER - cause an event when a certain time has elapsed

LOCATOR - provides position information

VALUATOR - provides scalar values

Whenever enabled, the sampled devices can be interrogated by the application program to determine their current state. The dimension of the LOCATOR is N-dimensional which makes it possible to simulate or control muliaxis machine tools and robots. The system automatically echoes operator interaction in a functional manner, unless the application program deactivates the echo. By this philosophy, dragging, rubber banding etc, is regarded as echo equivalent to traditional echoes like blinking, arrows, appearance of characters etc. The system also allows the operator to select the surface where the echo is to appear.

4. FUNCTIONAL CAPABILITIES

4.1 ATTACH PHYSICAL AND LOGICAL DEVICES

A mechanism for dynamic attachment of physical and logical devices is provided for the application program in order to utilize existing physical devices to a maximum:

ATTACH (TOOL-ID, DEVICE-CLASS, DEVICE-ID)

This routine will attach the physical device TOOL-ID to the logical DEVICE-CLASS and create the logical device DEVICE-ID.

To detach the device and make the physical device available for other purposes the following routine may be invoked:

DETACH (DEVICE-ID, TOOL-ID)
4.2. TRIGGER

A mechanism by which the application program can associate any event generating device with any logical input device is provided. The event device will in this case serve as a TRIGGER for the associated device and when the trigger is activated, all the corresponding data of the compound device will be read (see fig. 15).

\[
\text{ASSOC (DEVICE-ID, TRIGGER-ID, ATTRIBUTE, NEWDEVICE-ID)}
\]

- **DEVICE-ID**: The logical input device identifier
- **TRIGGER-ID**: The identifier for the logical event device used as a trigger
- **ATTRIBUTE**: The threshold value for the trigger
- **NEWDEVICE-ID**: The identifier for the compound device

This routine logically associate a logical device with a trigger. When the trigger is activated, all data corresponding to the devices associated with the trigger will be stored on a scratch or an event queue.

Several occurrences of the event report from a device may be stored on the event queue without the control being returned to the application program. This will not happen until the TRIGGER on the highest level is activated.

However, sampling devices may be directly interrogated anytime by the program because the buffer is currently updated.

If no trigger is specified for a device, the event from the device itself serves as the trigger. Default trigger attribute is the number of occurrences. Note that associations are many to many, so an invocation of this function for a particular pair of devices does not alter any previous associations for these devices.

The compound input device may be disassociated by

\[
\text{DASDEV (DEVICE-ID, OLD-DEVICE-ID, TRIGGER-ID)}
\]
Which delete the association on the highest level of the compound logical input device (DEVICE-ID). The identifiers for the old device and the trigger-id are returned in OLD-DEVICE-ID and TRIGGER-ID.
4.3. EVENT PROCESSING

As a first step in processing events, the application program invokes

EVENT (TIME, DEVICE-ID)

which dequeues an event report and returns the identification of the logical device causing the event. This identification is used to determine functions for getting the data from the input devices.

The EVENT function is requested to wait for the specified TIME or until there is at least one element on the event queue. A "nill" DEVICE-ID is a possible indication that no event occurred during the specified time.

At a minimum, event reports contain the data for the activated device. Additionally, device associations can cause information from other devices to be included in the event report. Because of the considerable variability of amount and types of data in event reports, there are separate GET-functions for each class of devices. These functions access data in the dequeued event report.

Let me exemplify how the functions for accessing data in the event report is built up by means of the routine for getting PICK data:

GETPICK (MAX, DATA-ARRAY, NDIM)

A PICK event identifies the selected primitive element by the name of the segment containing it and the primitive attribute value, PICK-ID. This information is returned in the DATA-ARRAY of this function.

A serie of input primitives can be entered. The maximum number of items is specified as the parameter MAX. The number of items actually entered is returned in the parameter NDIM ≤ MAX (for the other input primitives see (7)).
5. SYSTEM DESIGN

One of the advantages with the concept of compound logical input devices is the possibilities to support distributed computer systems and to utilize custom designed compound input devices built in hardware. The system is therefore designed modular with well defined interfaces. The major modules of a graphics supported system seen from a graphics point of view is:

- The application program which communicates with a graphics system based on standard subroutine calls (i.e. IDIGS) supported by stored knowledge of the I/O devices available.

- The front end of the graphics system which is device independent and communicates with the application program by standard calls and with the back end of the graphics system by means of synchronized standard messages and reports.

- The device independent back end of the graphics system which communicates with the front end by means of synchronized standard messages and reports, and with the device dependent part by means of interpreted commands and reports supported by stored knowledge.

- The device dependent part, which may be realized in hardware or software, communicates with the device independent part by interpreted commands and reports and with the physical device by means of the particular device dependent instructions and reports supported by knowledge of the capabilities of the devices and the possible simulations.

- The physical devices which communicates with the device dependent part by means of the device dependent instructions and reports and with the operator by means of the physical device itself.

Usually the modules will be grouped such that the application programs and the graphics front end runs on one or more application computer and the back end part of the graphics system runs on the workstation. All combinations are, however, possible:
5.1. THE INTERNAL DATA STRUCTURE

In order to achieve the flexibility of the input system as described in the functional requirements, a fast and easy to handle data structure must be designed.

Based on the described mappings, associations, and attributes of the compound input devices which have to be named, connected, stored, and manipulated, an internal data structure which supports the critical functions best possible must be chosen. In this case the choice will be between a relational and a hierarchical structure.

The main advantage of a relational structure in this case, is the simplicity of the tables which are both easy to understand and easy to build by the operators themselves. On the other hand the compound logical input devices are built up in a typical hierarchical way which is easy to support and easy to implement by a hierarchical structure. Our choice is a hierarchical structure, partly because the underlying structure is typical hierarchical
and partly because we can utilize the necessary structuring tools for other purposes such as administrating the event queue.

![Diagram of input device hierarchy]

**Fig. 5. Hierarchy of input devices.**

In order to support the implementation of the datastructure we have developed a small and simple "Free Storage" administration package. (9). The package is administrating a data array by allocating and returning blocks on request and by keeping track of the occupied and free blocks in the array. The package has the following subroutines:

**OPEN (ARRAY, LENGTH)**

which initializes an array of arbitrary size.

**ALLOC (LENGTH, IADR)**

which allocates a block of length LENGTH from the free storage area.

**RETURN (LENGTH, IADR)**

which returns a block of length LENGTH back to the free storage area.
As we will see later, this package is also used for administrating the event queue.

In order to give flexibility to the structure and to add the necessary attributes and information to the associations, link-objects have been introduced.

The chosen data structure is shown in fig. 6.

Fig. 6. The datastructure for Compound Logical input devices.
In fig. 6 the logical device LD1 is attached to the physical device PD1 and is associated with the compound logical device CLD1 and the Logical device LD2. CLD1 is again associated with CLD3 and the Logical device LD2 on the next level. CLD4 which is an association between LD2 and LD3 in an intermediate compound logical device is again part of an association on a higher level. LD4 is a direct attachment between a physical and logical device where the device itself has to serve as the trigger.

5.2. PROCESSING OF THE INPUT DATA

The data picked up from the physical devices are put into "packages" corresponding to the class of the attached logical device.

Such a data package consists of:

- Name of the logical device which generated the data

- The data itself corresponding to the logical device class

Dependent on if it is the highest level input device or not, the data are processed in two different ways:

If it is the highest level, the data package is linked directly on to the event queue. If not, the data package is stored as a scratch report until the trigger on the next higher level is activated. All reports on the event queue may therefore be regarded as unique and belonging to a particular, named logical input device. The data package on the event queue consists of:

- Name of the compound logical input device on the highest level which generated the report

- Name and class of the logical devices on the lowest level which "actually generated" the data

- The data itself corresponding to the logical device class

- Administrative data for manipulating the queue.

Since the scratch reports are created by input devices which may be associated with several higher level compound input devices, the reports contains a counter which tells how many higher level
input devices are still interested in the report. When the counter is zero, the report will be removed. The scratch report contains the following data:

- The name of the logical input device creating the report
- The name and class of the logical input devices on the lowest level
- The data itself
- A counter which tells the number of owners of the report
- Administrative data

| Name of the logical device creating this report |
| Length of the report |
| Numbers of owners |
| Name of the first logical input device on lowest level |
| : data from the 1. device : |
| Name of the 2. device on the lowest level |
| : data from the 2. device : : |
| : : |
| Name of the n'th device on the lowest level |
| Data from the n'th device on the lowest curd |
| Adm.data |
| EOR |

Fig. 7. The layout of a scratch report.
At a given moment we may have the following situation:

<table>
<thead>
<tr>
<th>Compound logical input device</th>
<th>Link object</th>
<th>Scratch report</th>
<th>Number of owners</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td># owners = 1</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td># owners = 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td># owners = 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td># owners = 1</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td># owners = 1</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td># owners = 1</td>
</tr>
</tbody>
</table>

Fig. 8. The organization of scratch reports.

In fig. 8 the compound logical input devices A and B are built on the same low level input devices, which may also be compound devices. A and B may have different triggers or trigger thresholds on the higher level associations.

C has no common lower level device with A, B or D.

D has at the moment no scratch reports.
5.3. THE EVENT QUEUE

As described earlier, when an input device is activated, it generated small data packages which are building up the input reports. The reports may be stored intermediately in a scratch area or they may be put directly on the input queue. The generation and processing of the data packages are done asynchronous and independent of the application program, while the application program reads the current reports from the input queue in a synchronous way.

There are many ways of organizing such an input queue. The most common solution is to utilize a ring buffer, where it is easy to manage both the synchronous and the asynchronous aspect. Such an approach makes the input of data at the end of the queue totally independent of the reading of the data in the front of the queue.

In this case, however, a "free storage" solution is chosen (9) because of several reasons: The Flush functions in IDIGS (4) are violating the ringbuffer concept of a producer generating data at the end of the queue while a consumer is eating the data from the front of the queue.

In the communication process between the main parts of the system (see fig. 4), it is important to minimize the administration and the data flow in order to gain speed. From this point of view the application program is interested in the current event report only, which means that the current event report and the event queue may be split in such a way that the current event report is the only common data.

The last reason for using this concept is that a "free storage" administration package is already included in the system (see 5.1).

Garbage collection in connection with the event queue is another problem because of the Flush-function. Holes will occur, and the utilization of the empty areas are difficult to handle because the sequence of the reports are important and because the length of the reports are all different.

When a "free storage" concept is used, the reports are linked freely within the free storage area and the open spaces are
continuously made available for the new reports. Only if all the free space slots are too small for a new report, a garbage collection must be performed.

![Diagram of event queue in a free storage area]

**Fig. 9. The event queue in a free storage area.**

The reports in the event queue have the following lay-out.

<table>
<thead>
<tr>
<th>Highest level logical device</th>
<th>Lowest level logical device</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Length of report</td>
</tr>
<tr>
<td></td>
<td>&lt;data&gt;</td>
</tr>
<tr>
<td></td>
<td>Next report pointer</td>
</tr>
</tbody>
</table>

**Fig. 10. Lay-out of an event report.**
When the EVENT process is evoked, the oldest data records on the event queue, generated by the given logical input device, are collected and placed in the current event report. The current event report may therefore consist of not only one oldest report, but a sequence of "equally oldest" reports generated by the same highest level logical input device.

The reason for such a solution is efficiency. A serie of locator reports, generated by for instance a logical digitizer, can then be picked up by the GETLOC routine and handed over to a POLYLINE routine in one operation for immediate drawing of complicated polygons. Each current event report may be read zero or several times and will not be destroyed until a new EVENT process is activated.

5.4. SYNCHRONIZATION

As described earlier, the communication between the front-end and the back-end of the graphics system consists of messages (specs) and event reports in such a way that:

- the front end produces the specs and the back end is the consumer

- the back end produces the answer which is consumed by the front end and the necessary actions are activated

- the back end produces the event reports which are consumed by the front end.

If the various processes are performed in parallell utilizing common resources, synchronization is necessary in order to avoid inconsistency or dead locks.

There are several ways to handle this problem such as Brinch-Hansen's critical regions and signal/wait directives (10) or Dijkstra's semaphores and P,V operations (11). For this particular case, Dijkstra's semaphores and P,V operation will solve the problem nicely.
Fig. 11. Communicating messages (SPEC'S).

When the event reports are concerned, the roles of the Front- and Back-end system are reversed. In order to achieve symmetry the Front end is logically returning the empty buffer asking for new informations.

Fig. 12. Communicating Event report.

While the Front- and Back-end of the graphics system are cooperating processes utilizing common resources, the Back-end and the interrupt system are competing for their common resources.

Fig. 13. The total model.
In order to control the competing processes, we need one semaphore for each common resource to avoid deadlocks.

If the semaphore is a small look up table containing information about from which tools on the lowest level are data to be accepted, special features such as echo etc., the synchronization programs may become quite simple:

**Back-end**
- \( P \) (sem)
- \( P \) (scratch)
- \( P \) (event queue)
  - <execute order>
  - \( V \) (event queue)
  - \( V \) (scratch)
  - \( V \) (sem)

**Interrupt system**
- \( P \) (sem)
- \( P \) (scratch)
  - \( V \) (event queue)
  - \( V \) (scratch)
  - \( V \) (sem)
  - <generate scratch report>
  - <create event report>
  - <echo>

Fig. 14. The necessary synchronization directives between the Back-end and the Interrupt system.

6. IMPLEMENTATION AND TESTING

In order to test the concept of the compound logical input devices a relatively simple test version has been implemented on a NORD-100 and a Tektronix 4014 terminal.

Because the terminal has no local computer power, all the different processes has to be simulated on the host computer in such a way that all the different processes are run in quasi parallel by utilizing the real time system of the computer.

Each of the processes are given a priority which are tuned during
the test phase in order to find the best possible response time.

The implementation proved that the concept is viable and all the specified functions can be performed.

The response time for simple logical devices was acceptable (i.e. \( \leq 1/3 \) second), while for more complex compound devices the response time on this particular configuration grew fast too slow.

Timing each of the processes as if they were running in true parallel environment, indicated that an advanced hardware system could handle very complicated compound logical input devices with acceptable performance.

Because of the current hardware shortcomings and the human sensitivity for time delays, a meaningful test of the system on operators was not possible. The specified functionality of the system, however, has been discussed with users and very positive responses and feedbacks has been received.

7. EXAMPLES OF COMPOUND INPUT DEVICES

Ex. 1. Suppose we want to build a logical DIGITIZER device having access to the physical device cursor with function buttons (FB) on top of it.

The DIGITIZER is supposed to sample the position each time the cursor has been moved a certain distance. The data is put on the event queue when FB-1 is pushed:

```
ATTACH ("CURSOR", LOCATOR, ID1)
ATTACH ("FB-1", CHOICE, ID2)
ASSOC (ID1, DISTANCE, LENGTH, ID3)
ASSOC (ID3, ID2, NUMBER, ID4)
```

ID4 is now the Device-id of a logical DIGITIZER built up from a LOCATOR (Cursor) and a TRIGGER (Distance). When the cursor has moved a distance LENGTH, its position will be sampled.

This set of samples will be entered into the event queue when FB-1 is pushed. The corresponding event report is made available for the application program by calling EVENT:
500  EVENT (TIME, DEVICE-ID)
     ID (DEVICE-ID .EQ. 0) GO TO 500
GETLOC (MAX, DATA-ARRAY, NDIM)
POLYLINE (DATA-ARRAY, NDIM)

The whole procedure may be regarded as a hierarchy of input processing:

Fig. 15. Logical DIGITIZER device.

This is particular efficient on a satellite configuration. All the sampling is done in the satellite, but the set of samples is not sent to the host until FB-1 is pushed.

Ex. 2: We have a lightpen (LP) with a lightpen switch (LPS) and function buttons (FB) as physical devices and we want to build the logical compound input device IDENTIFIER.

ATTACH (LP, PICK, ID1)
ATTACH (LPS, CHOICE, ID2)
ATTACH (FB1, CHOICE, ID3)
ASSOSIATE (ID1, ID2, 1, ID4)
ASSOSIATE (ID4, ID3, 1, ID5)
We may now use the lightpen (LP) with the switch (LPS) to identify a sequence of elements without leaving the lowest level of interrupt handling. This operation is therefore very fast. By pushing FB1 the whole sequence of identifiers are made available for the EVENT routine which may reside in the host machine.

![Diagram](image)

**Fig. 16. Logical IDENTIFIER device.**

By using this approach, logical input devices of any complexity can be built to provide very powerful input mechanisms for the user. The interaction speed may be considerably increased by this possibility for optimizing the data handling at the different levels.

The concept is also well suited for utilization of special purpose input micro processors running in parallel.
8. CONCLUSION

The concept presented in this paper is a contribution to make more general and efficient man/machine communication procedures.

By means of compound logical input devices the system designer is able to better utilize existing equipments and build logical input tools which is as powerful as the equivalent output devices. That means better symmetry between input and output which will help the user to a more natural and less error prone dialogue.
9. REFERENCES

(1) G. Butlin: "A Conceptual framework for Man-Computer Interaction"
Dep. of Engineering, The University of Leichesten 1976

(2) F.R.A. Hopgood et.al.: "SEILLAC II. Workshop on Man-
Machine Interaction"
North-Holland Publishing Company 1980

(3) J.D. Foley and V.L. Wallace: "The Art of Natural Man-
Machine Communication"

(4) V.L. Wallace: "The Semantics of Graphics Input Devices"
Computer Graphics 10 (1) 1976

(5) R.A. Guedj (eds.) "IFIP Workshop on Methodology and In-
action"
North-Holland Publishing Company 1980

(6) SIGGRAPH-ACM "Status Report of the Graphics Standards Plan-
ing Committee"
Volume 13, Number 3, August 1979

(7) K. Bø: "IDIGS, Interactive Device Independent Graphic Sys-
tem" NORSIGD 1979

(8) ISO "Graphical Kernal System (GKS) - Functional Description"
DP 7942, 1982

(9) K. Bø, S.H. Moen: "Grafisk Databehandling"
Tapir 1982

(10) P. Brinch-Hansen: "Operating System Principles"
Prentice-Hall 1973

(11) E.W. Dijkstra: "Cooperating Sequential Processes" In Pro-
gramming Languages
Academic Press Ny. 1968

32