ALPHA
Implementation of a
subset of PHIGS

by
Ylva Gullestad
ALPHA

Implementation of a subset of PHIGS

by

Ylva Gullestad

December 1988

SICS, Swedish Institute of Computer Science
BOX 1263
S-164 28 KISTA
SWEDEN
Abstract

ALPHA is a support system for applications using interactive computer graphics. The ALPHA system is a subsystem of the PHIGS graphic standard (the PHIGS version presented in dpANS X3.144-198x of October 1986), specifically tailored to the Xerox/Interlisp-D environment.

The Programmer's Hierarchical Interactive Graphics System (PHIGS) is a functional specification of the interface between an application program and its graphic support system. PHIGS supports hierarchical graphics, user interaction and 3-D modelling.

ALHPA is a detached CommonLisp package in the Interlisp-D environment. The package exports all ALPHA functions to be used and only imports necessary Interlisp-D functions.

Thoughts about this master thesis work

This work has been the most exiting and stimulating experience during my study time. One reason for this has been the knowledge that my work does not stop at producing a thesis. It gives a special satisfaction to know that the thesis, as a subproject, will be used in actual production.

The master thesis work has made me understand that reality is not the way I would like it to be. Programming environments are not perfect, i.e. they do not always work the way they are intended, things happening outside the area in which you are working do affect your work even if they should not.

The knowledge I have collected from the course Programming Environments and Interactivity has made this work possible. The experience from about 20 different courses has made it possible for me to learn and use new conditions, and last but not least, to learn that things most often will work out even though it looks dark at times.
Contents

Abstract
Thoughts about this master thesis work

Contents

References

Chapter I PHIGS in general

Chapter II ALPHA a PHIGS implementation

Chapter III Technical documentation of ALPHA

Supplement

1 Summary of the ALPHA functions
   Control functions
   Editing functions
   Structure manipulating functions
   Graphic output functions
   Attribute changing functions
   Miscellaneous structure element creating functions
   Functions for supporting modelling
   Input functions
   Archive functions
   Inquire functions

2 Functional specification of non-standard functions
   new
   different

3 How to start ALPHA

4 The structure function and struct

ALPHA - QUICK REFERENCE GUIDE

A full index is available in the beginning of every chapter.
References


Chapter I
PHIGS in general

The Programmer's Hierarchical Interactive Graphics System (PHIGS) is a functional specification of the interface between an application program and its graphic support system. PHIGS supports hierarchical graphics, user interaction and 3-D modelling. This chapter is a general presentation of PHIGS.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>4</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>5</td>
</tr>
<tr>
<td>2 The goals of PHIGS</td>
<td>5</td>
</tr>
<tr>
<td>3 The PHIGS model</td>
<td>5</td>
</tr>
<tr>
<td>4 How to build an image</td>
<td>6</td>
</tr>
<tr>
<td>Graphical primitives</td>
<td>6</td>
</tr>
<tr>
<td>Attributes</td>
<td>6</td>
</tr>
<tr>
<td>View specific indices</td>
<td>6</td>
</tr>
<tr>
<td>Modelling matrices</td>
<td>6</td>
</tr>
<tr>
<td>Labels</td>
<td>6</td>
</tr>
<tr>
<td>Namesets</td>
<td>6</td>
</tr>
<tr>
<td>Execute structure</td>
<td>7</td>
</tr>
<tr>
<td>Editing</td>
<td>7</td>
</tr>
<tr>
<td>Posted</td>
<td>7</td>
</tr>
<tr>
<td>The bindings of attribute</td>
<td>8</td>
</tr>
<tr>
<td>5 Workstation</td>
<td>8</td>
</tr>
<tr>
<td>6 Transformation pipeline</td>
<td>8</td>
</tr>
<tr>
<td>Composite modelling transformation</td>
<td>9</td>
</tr>
<tr>
<td>View transformation</td>
<td>9</td>
</tr>
<tr>
<td>Clipping operation and projection</td>
<td>9</td>
</tr>
<tr>
<td>Workstation clip and transformation</td>
<td>9</td>
</tr>
<tr>
<td>7 Filter</td>
<td>10</td>
</tr>
<tr>
<td>Example</td>
<td>10</td>
</tr>
<tr>
<td>8 Interaction - input</td>
<td>12</td>
</tr>
<tr>
<td>Modes</td>
<td>12</td>
</tr>
<tr>
<td>9 Example</td>
<td>13</td>
</tr>
<tr>
<td>10 PHIGS today</td>
<td>15</td>
</tr>
<tr>
<td>11 Criticism of PHIGS</td>
<td>15</td>
</tr>
</tbody>
</table>
Chapter I PHIGS in General

1 Introduction
Why is a standard needed at all? A standard makes the programmers able to concentrate on the application program and not worry about what kind of system they are working with. Common rules can be taught, improved and reused in different projects, they can be coordinated with other organizations. The standard makes it easy to mix small and large systems. It is easy to move applications from one environment to another.

Why PHIGS? PHIGS is using an hierarchical modelling system rather than extended I/O libraries. It is an improvement of graphic packages in three ways:
- hierarchical defined graphical objects
- high degree of interactivity
- dynamic modifying of objects

PHIGS is also well suited for object oriented design technique, and it is addressing both 3D and 2D.

2 The goals of PHIGS
The goals of PHIGS are to provide an interactive toolkit specially made for highly interactive, hierarchical systems. PHIGS is specified for making and modifying objects. Since computer graphics can render application data, it has become an important tool to make models, real or simulated, more understandable. Applications with large databases, like CAD, CAM, scientific modelling and simulation, all try to interact with the databases and to reflect these changes in some form of graphical rendering. The key advantages of PHIGS is that it makes it possible to describe the application model effectively and that it updates the graphical model and rendering quickly.

3 The PHIGS model
PHIGS consists of four parts:
- the control system
- the data definition system
- the data display system
- the interaction handler

![Diagram of PHIGS model](image)

*Fig 1:1 The PHIGS model*
Implementation of a subset of PHIGS

The control system gives the application program access to the different parts mentioned. The data definition system controls defining and modifying objects, i.e. changes to structures, the data display system keeps track of the changes made and updates the rendering. The interaction handler is responsible for the input contact with the application operator.

4 How to build an image
The fundamental parts of PHIGS, where the graphical information lies are called structures. The structures consist of structure elements, these are graphical primitives and their attributes, view specific indexes, modelling matrices, elements for building of hierarchical structures, labels, namesets and application dependent data. PHIGS is equipped both with 2D and 3D. The 2D functions are simply shorthand versions of their 3D counterparts, PHIGS handles this internally by setting the z component to zero.

Graphical primitives
The graphical primitives provided in PHIGS are:
- polyline (a set of connected lines)
- polymarker (a set of positions, each indicated with a marker)
- text
- fill area (filled polygons without edges)
- fill area set (filled polygons with edge control)
- cell array (a rectangular grid of equal sized rectangular cells with the same colour)

Attributes
PHIGS provides primitive segregated attributes. That means that every graphical primitive has its own attributes and that no other primitive is affected by them. The fact that the colour of text is set does not affect the colour of polyline and vice versa. The attributes in PHIGS are set either individually or as a bundle. A bundle holds in some sense global values. E.g. the bundle to line contains the values of LINE WIDTH SCALE FACTOR, LINE TYPE and POLYLINE COLOUR INDEX.

E.g. the predefined line bundles in PHIGS are:

<table>
<thead>
<tr>
<th>BUNDLE</th>
<th>LINETHICKNESS SCALE FACTOR</th>
<th>LINETYPE</th>
<th>POLYLINE COLOUR TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>solid</td>
<td>some</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>dashed</td>
<td>some</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>dotted</td>
<td>some</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>dashed-dotted</td>
<td>some</td>
</tr>
</tbody>
</table>

The Aspect Source Flag (ASF) decides if bundles or individuality are used. If ASF is set to INDIVIDUAL, the attributes are set one by one, if it is set to BUNDLE, all attributes are read from the bundle table. It is not possible to erase the bundle values by writing individual values.

View specific indexes
The function SET VIEW INDEX decides in which view a structure is displayed.

Modelling matrices
See Transformation pipeline.

Labels
LABEL inserts a label in the open structure. It is possible to set the element pointer at a label and to delete structure elements between labels.

Namesets
See Filter
Execute structure

To make hierarchies the `EXECUTE STRUCTURE` element is used. When the structure is
traversed for displaying, the `EXECUTE STRUCTURE` element makes another structure
execute in the context of the original structure, i.e. a
hierarchical organization
has been created, like the
one in figure I:2.
In this example the structure
linefellow contains
`EXECUTE STRUCTURE` elements referring to head,
lums and body. The
structure linefellow is
called parent to head,
lums and body. The
structures head, limbs
and body are called
children to the structure
linefellow. The structure
linefellow is called the
root of the hierarchy.

![Diagram of linefellow structure hierarchy]

**Fig 1.2 The structure hierarchy of linefellow**

All structures may execute a structure, and there is no limit given to the number
of structures that each structure may execute. The depth of the hierarchy is also unlimited.

Editing

It is possible to modify structures at any time. This is done with the structure editor. Only
one structure can be edited at the time. To be made ready for editing the structure has to be
opened. If `OPEN STRUCTURE` is activated
when the specified structure does not
exist, it will be created automatically.
When a structure is open it is possible to
insert, modify and delete elements.
Where the modifications take place is
determined by the element pointer.
The modification takes place at the
structure element that the element pointer
is pointing at. The element pointer can
be moved back and fourth in the
structure, it can be set to point at a label,
moved to a specific element number or
moved a given number of steps from the
current element it is pointing at.

![Diagram of structure editor]

**Fig 1.3 The open structure**

Posted

To make a structure visible it has to be posted to a workstation. To make a structure
hierarchy visible only the root is to be posted. When the structure is posted it is traversed,
i.e. executed. A structure may be posted to several workstations. The `UNPOST STRUCTURE`
function makes a structure disappear where it is displayed.
Implementation of a subset of PHIGS

The bindings of attribute
The attributes, which define the appearance of the graphical objects, are the same as in other graphical systems. The big difference with PHIGS, is that the attributes are bound by execution and not when the primitive is created. When an EXECUTE STRUCTURE element is executed during structure traversal the following actions are taken:
1. The traversal of the parent structure is interrupted
2. The value of the attributes are saved
3. The executed structure is traversed (and all structures it executes)
4. The attributes values are restored to the values saved before the EXECUTE STRUCTURE element
5. The traversal of the parent continues
These rules implies that a child structure inherits the value of the attributes from its parent when it is executed. Since the values of the attributes are restored after a EXECUTE STRUCTURE element, a structure only affects its children. Posted structures inherit their attributes from a global default value. One way to look at this is that a structure always pushes the attribute registers on a stack before calling another structure and then restores them from the top of the stack when the called structure has returned.

5 Workstation
PHIGS is based on the concept of abstract workstations. These can for instance be implemented as windows. The workstations provide the logical interface through which the application program controls physical devices. Connection to a workstation is established by the function OPEN WORKSTATION. It is possible to have more than one workstation on the same physical computer or even on different physical computers. Every graphical primitive has a number of attributes related to it. The workstation determines how to display a given primitive by examining the attribute registers available for each primitive. The attributes in PHIGS are workstation dependent. The application may choose when an updating of the displayed data should take place. The values (in increasing order of delay) are:
- ASAP: The display on the workstation becomes visually correct As Soon As Possible. PHIGS takes all the steps necessary to update the display before control is returned to the application program.
- BNIG: The display on the workstation becomes visually correct Before Next Interaction Globally, i.e. before the next input device gets underway on any workstation.
- BNIL: The display on the workstation becomes visually correct Before Next Interaction Locally, i.e. before the next input device gets underway on that workstation.
- ASTI: The display on the workstation becomes visually correct At Some Time.
- WAIT: the display on the workstation becomes visually correct When the Application requests IT.

6 Transformation pipeline
PHIGS provides general tools for describing transformations and for making dynamic modifications to them. The PHIGS transformation pipeline is conceptually a 3D pipeline. PHIGS also provides 2D tools, they are shorthands of the 3D form with the operation applied on z=0, i.e. a plane in the 3D. The orientation of this plane in 3D world coordinates are controlled by model transformations. The transformations available and their relation are shown in the transformation pipeline in figure I.4.
Chapter I PHIGS in general

Composite modelling transformation

The modelling transformation locates and orients object in space and in relation to other objects. Structure elements describe model transformations as 4×4 matrices. Besides, there are utility functions which produce matrices related to the most common transformations, such as scaling, rotation and translation. Model transformations can be replaced or concatenated with new transformations by matrix multiplication. There are two kinds of modelling transformations:

- Global modelling transformations. When a EXECUTE STRUCTURE is found during structure traversal, the executed structure inherits a current transformation matrix from its parent. This matrix becomes the global modelling transformation of the structure. SET GLOBAL TRANSFORMATION modifies it.
- Local modelling transformations. These are concatenated with the global value in the order GXL to produce a composite current transformation matrix, but does not affect the value of the global transformation matrix. Note that the local modelling transformations are applied before the global modelling transformation.

The model transformation converts the model coordinate system to the world coordinate system.

View orientation transformation

The view table entry on the workstation describes the view transformations, as 4×4 matrices. They describe the orientation of the view of the scene. The view transformation maps the world coordinate system onto the view coordinate system.

Clipping operation and projection

The clipping and projection part of the transformation pipeline maps the view coordinates onto the Normalized Projection Coordinate system (NPC). These window-to-viewport transformations are stored in the workstation view table, not as structure elements. Components in this transformation are i.a. the distance of the view plane, the window, the projections reference point and whether the projection is parallel or perspective.

Workstation clip and transformation

The NPC space can be regarded as an abstract image composition space that is workstation dependent. The workstation can select any part of its NPC space to be displayed on any part of its physical display space. A workstation transformation is a mapping from NPC space onto the device coordinate space.
Implementation of a subset of PHIGS

7 Filter

Many applications need to be able to group objects together according to invisibility, highlighting and detectability. The application objects do not have to be related in structure hierarchies, they can be in different objects and structures. PHIGS provides a powerful toolkit to handle these problems. Every graphical primitive has a `NAMESET` defining which classes it belongs to. And each of the three traversal operations visibility, highlighting and detectability has two filters associated with it. These filters, inclusion and exclusion filters are workstation dependent. To be able to execute one of the operations visibility, highlighting and detectability there are two conditions a primitive’s `NAMESET` must fulfil, these are: (1) `NAMESET` must have at least one member in the inclusion filter and (2) must not have any member in the exclusion filter. Figure 1:5 shows how `NAMESET A` fulfils the conditions and how `NAMESET B` does not. The executed structure inherits its members of the `NAMESET` from its parent, just like other attributes.

Example:

A car dealer wants to be able to classify his cars available in four classes: type, color, #of doors, equipment. The structure to be able to set the `NAMESET` would look like:

```
OPEN STRUCTURE (new-car)
LABEL (TYPE)
ADD NAMES TO SET (type)
LABEL (COLOUR)
ADD NAMES TO SET (colour)
LABEL (NROFDOORS)
ADD NAMES TO SET (number)
LABEL (EQUIP)
ADD NAMES TO SET (radio)
POLYLINE(...) draw the car
CLOSE STRUCTURE
```

At the moment there are seven cars available for sale, they have the following `NAMESETS`:

- `car1` (volvo red five radio)
- `car2` (volvo blue four spoiler)
- `car3` (fiat green five radio)
- `car4` (ford gray two sunroof)
- `car5` (fiat blue four radio)
- `car6` (saab white three radio)
- `car7` (saab white four none)

Fig 1:5 Sample of NameSets

Fig 1:6 Seven cars
Now the car dealer wants to highlight the pictures of all cars that are blue, this is done by:

\[
\text{SET HIGHLIGHTING FILTER (workstationnr, include, exclude)}
\]
\[
\text{where}
\]
\[
\text{include} = \text{(blue)}
\]
\[
\text{exclude} = \text{null}
\]

The result is depicted below.

![Figure 1.7 All blue cars highlighted](image1)

If he only wants his volvos to be visible this instruction would give him the expected result.

\[
\text{SET INVISIBILITY FILTER (workstation, inclusion, exclusion)}
\]
\[
\text{where}
\]
\[
\text{inclusion} = \text{all possibly kinds of TYPE}
\]
\[
\text{exclude} = \text{volvo}
\]

![Figure 1.8 All volvos visible](image2)

This example shows the use of filters and namesets. As default the inclusion and exclusion filter are empty, i.e. the objects are not highlighted, not invisible (i.e. visible) and not pickable.
Implementation of a subset of PHIGS

8 Interaction - input

PHIGS specifies six kinds of logical classes of interaction input. These are:

- Locator
- Stroke
- String
- Choice
- Valuator
- Pick

Every logical input device is mapped to a physical device. Locator and Stroke returns position information to the application program and the graphical system. It is returned on the form (x, y) in 2D and (x, y, z) in 3D. Locator returns a single position and Stroke returns several positions. They are normally mapped on mouse, lightpen or joystick.

String returns a string and is usually mapped to the keyboard.

Choice is used to choose among several choices, for example to select an item in a menu or to select which buttons to press on a mouse. Choice returns information on the form "button number tree is pressed".

Valuator is like a light dimmer. It returns a real number between defined minimum and maximum. Typically mapped on a dialog box (analogous control with lightpen).

Pick returns information about the part of an object of which the device is pointing. The Pick device is often mapped on the mouse, lightpen or joystick. It returns information of the form "you are now pointing at the head of a line fellow", obviously written in a more cryptical form.

Modes

The logical input devices are divided into three modes:

- Request
- Sample
- Event mode

REQUEST mode: demands the application operator to fire the trigger to get the input report. For example, a Locator Request demands the operator to press a button on the mouse to return the position (x, y).

SAMPLE mode: does not request the operator to take any action, it simply reads the current value of the device and returns it. For example, Sample Locator returns the position where the mouse happened to be.
EVENT mode: is the most complex and powerful mode of the three. When the application operator makes a move to fire the trigger to get an input report, this action - an input event - is placed in a first-in-first-out queue. The elements in the queue contain the type of the input event and a package that contain the information that type of input event returns. The application fetches elements from the event queue, checks the type and makes appropriate actions.

Event mode is used to make applications event-driven rather than procedure-driven. Event-driven systems are known to be more user friendly than procedure-driven, since the operator controls the order of input actions and not the application developer.

9 Example
This example tries to explain the hierarchical model of PHIGS, the inheritance of attribute and the pick interaction.

Let’s draw a garden.

OPEN STRUCTURE (garden)
SET LINE WIDTH SCALE FACTOR (2)
EXECUTE STRUCTURE (buildings)
SET LINE WIDTH SCALE FACTOR (6)
EXECUTE STRUCTURE (plants)

OPEN STRUCTURE (buildings)
SET LOCAL TRANSFORMATION ((TRANSLATE (50 75))
EXECUTE STRUCTURE (house)
SET LOCAL TRANSFORMATION (…)
EXECUTE STRUCTURE (fence)
SET LINE WIDTH SCALE FACTOR (1)
SET LOCAL TRANSFORMATION (…)
EXECUTE STRUCTURE (gate)

OPEN STRUCTURE (plants)
SET LOCAL TRANSFORMATION (…)
EXECUTE STRUCTURE (bush)
SET LOCAL TRANSFORMATION (…)
EXECUTE STRUCTURE (bush)
SET LOCAL TRANSFORMATION (…)
EXECUTE STRUCTURE (bush)
SET LOCAL TRANSFORMATION (…)
EXECUTE STRUCTURE (tree)
SET LOCAL TRANSFORMATION (…)
EXECUTE STRUCTURE (tree)

OPEN STRUCTURE (bush)
POLYLINE ((0 0) (5 5) (8 2)…)

The structure garden is created and opened
Set the line width to two
Make a structure hierarchy by calling plants
Change the modelling coordinate system

Draw a bush
Implementation of a subset of PHIGS

OPEN STRUCTURE (tree)
POLYLINE (...)

OPEN STRUCTURE (house)
FILL AREA SET ((0 0) (75 0) (75 100) (0 100))

OPEN STRUCTURE (fence)
POLYLINE (...)

OPEN STRUCTURE (gate)
FILL AREA SET (...)
POLYLINE (...)
POLYLINE (...)
FILL AREA SET (...)

Now all structures used in this example are defined, they form the hierarchical organization shown in figure I:12.

However none of them are yet visible. To make a structure visible it has to be posted.

POST STRUCTURE (garden)
Makes the picture visible:

Figure I:13 shows how fence inherits its LINEWIDTH from garden and how gate inherits its LINEWIDTH from buildings.
The pick device is initiated with:

\text{INITIALISE PICK (WORKSTATIONNAME DEVICENUMBER)}

and if \text{REQUEST} mode is set, which is set by \text{SET-PICK-MODE}, PHIGS waits for a pick when

\text{REQUEST PICK (WORKSTATION DEVICENUMBER)}\text{ is called.}

If the left tree in the garden is chosen, the pick device will return the information (garden, plants, tree). If the structure tree is posted separately and picked, the pick device will return the information (tree). Note that the \text{SET LINE WIDTH SCALE FACTOR} does not affect the edge around the \text{FILL AREA SET} primitive, used in for instance gate.

10 PHIGS today

The specification of the PHIGS standard started by ANSI in the end of the eightieth decade. The formalisation was then overtaken by ISO. PHIGS is today adopted as a standard by both ANSI and ISO.

Graphical functions addressing shading and lightning are not included in the PHIGS specification. These facilities are specified by an ad hoc committee called PHIGS+. The PHIGS+ committee has been working since November 1986. This committee develops compatible extensions to the ANSI PHIGS draft proposal. Besides the already mentioned extensions they also specify parametric definition of curves and curved planes. PHIGS+ primitives and attributes follows the philosophy of PHIGS. The new primitives and attributes are structure elements just like those in PHIGS.

PHIGS systems implemented so far are among others:

- ALPHA implemented on XEROX 1186/1109 at SICS
- PHIGS 1.0b6 - NMP-PHIGS developed at NMP-CAD [7].
- TGS Figaro, fully supported on VAX/VMS, IBM VM/CMS, MVS/TSO, Apollo Aegis and many major Unix workstations, including Sun, Silicon Graphics, Hewlett-Packard etc [3].
- Sun microsystem has as commercial PHIGS implementation available on their SUN 3 hardware.

11 Criticism of PHIGS

PHIGS structure hierarchy is a complex system and it can be hard to implement quickly and without loss of much performance. This can make the decision to use PHIGS instead of GKS for instance, hard to make.

When the \text{ASF} is set to \text{BUNDLE} it is not possible to change an individual attribute, it is a major disadvantage not to be able to use parts of the defined bundles.

None of the inquire functions specified in PHIGS return the current coordinates of graphical primitives. Hence coordinate data has to be stored by the application.
Implementation of a subset of PHIGS

This page is intentionally left blank
Chapter II
ALPHA a PHIGS implementation

ALPHA is a support system for applications using interactive computer graphics. The ALPHA system uses many of the concepts developed for the PHIGS graphic standard (the PHIGS version presented in dpANS X3.144-198x of October 1986), but is smaller and specifically tailored to the Xerox/Interlisp-D environment. When possible, ALPHA interface functions are identical to the corresponding PHIGS functions and can therefore be found in any PHIGS manual.
# Contents

Contents .............................................. 18

1 Why PHIGS .................................. 19

2 Where is ALPHA used ...................... 19

3 Where, when, how ............................ 19

4 General presentation of the ALPHA machine .... 20

5 Extentions ................................. 21

6 Constraints ................................. 21

7 Parts of the ALPHA system
   Workstations ............................. 22
   Bundles ...................................... 22
   Attribute inheritance .................. 22
   Transformation pipeline ......... 23
   Structures ................................. 23
   Graphical primitives ............... 24
Chapter II  ALPHA a PHIGS implementation

1 Why PHIGS?
The following elements of the PHIGS system attracted us particularly:

- Hierarchical graphical structures
  - symbolical names on these
  - functions for editing of structures
  - local coordinate systems
- Updating independent of application
- Interaction directly related to instances of graphical objects

This chapter describes an implementation of a subset of PHIGS which is called ALPHA.

2 Where is ALPHA used?
LOGGIE is a system for construction of design environments in which the design process is based on formal techniques given an interactive graphic syntax. The LOGGIE user can interactively specify and use meta notations, specification/programming languages, specifications/programs, program transformations, etc.

LOGGIE is divided into derivation layer and a structure presentation layer and includes advanced graphical support. See figure II:1. Since LOGGIE itself is an interactive tool several types of editors will become state-of-the-art when interactive graphics and human interaction is concerned.

![Diagram](image)

The derivation layer is specified in terms of a hierarchical graph model based on typed nodes connected by typed links.

The structure specification layer sees the nodes as nodes in a tree.

The graphical support system sees the nodes as boxes, circles, ellipses etc, and is ready to display these objects.

In the derivation layer a graph can represent a meta-notation for specification of attribute grammar formal language, an attribute grammar (including syntax tree, attributes and inherit/synthesis functions), or a program written according to a specific grammar.

The purpose of the structure presentation layer is to visualise structures generated by the derivation layer. It integrates both text and graphical symbols gracefully.

The graphical support system is a versatile system for interactive graphics implemented in an object oriented manner. The fundamenal in the system is the graphic standard PHIGS, a sub set has been implemented, this is ALPHA.

3 Where, when, how?
ALPHA is made on XEROX 1186/1109, LYRIC release, CommonLisp.
ALPHA is made by three persons, and it took approximately 24 weeks to accomplish.
4 General presentation of the ALPHA machine

The ALPHA-Machine is the virtual architecture of the ALPHA system, specifically designed to perform interactive graphics. The machine consists of a memory to hold all active structures, an editor and a number of workstations (see fig 11:2).

![Diagram of the ALPHA machine architecture]

**Fig 11:2 Architecture of ALPHA**

The ALPHA machine stores all defined structures and it has a structure editor. Every ALPHA machine can hold several workstations and every Lisp machine can hold several ALPHA machines.

To control the machine, the user/using program calls a number of interface functions. These can be divided into four groups:

- Control functions: These immediately perform certain operations such as creating new structures, opening new workstations etc.
- Editing functions: Editing functions are used to modify the content of already existing structures, such as deleting an element or moving the editing pointer.
- Structure content functions: This group of functions creates and inserts instruction elements in the structure being edited. When the structure is executed, these instructions perform various kinds of graphic output and attribute changing.
- Input functions: These are used to prompt the user for different kinds of values. To facilitate user interaction, four different kinds of input are provided.

All instructions/functions follow the PHIGS standard as far as possible. One exception is that all functions have a key-parameter machine. If no machine name is sent to the function, it uses the machine name "GLOBALMACHINE" as default. All machines are stored in an association list in the variable "MACHINETABLE" , which is global in the ALPHA package.
5 Extensions
The viewports in PHIGS are implemented by the XEROX window system. It is in these windows posted structures are displayed. The windows can be opened and closed with OPEN-VIEW and CLOSE-VIEW.

The ALPHA implementation has skipped the generalized drawing primitive, instead the following graphical primitives are implemented:

- arc
- arrow
- box
- circle
- curve
- ellipse
- filled-circle

The function BITMAP replaces the CELL ARRAY, this is made to make Xerox bitmaps used instead.

The FILL-AREA primitive in ALPHA, is corresponding to the FILL-AREA-SET primitive in PHIGS.

The POLYMARKER primitive is not implemented.

In ALPHA it is not only possible to make new bundles (the bundles differ from those in PHIGS too, see Bundles) out of existing attributes. For the graphical primitives LINE, EDGE, TEXT and PATTERN it is also possible to make bundles with completely new content, eg a line with three points and one dash.

The text primitive is simplified, only the SET-TEXT-FONT, SET-FONT-FACE and SET-FONT-SIZE are available this simplification is made to follow the Interlisp-D functions in the text area.

6 Constraints
As mentioned before only a subset of PHIGS is implemented. The major parts not present in ALPHA are:

- 3D
  ALPHA is the 2D part of PHIGS, the 3D functions are not implemented. However, 2D structures can be put in planes with different priority.

- Colour
  No colour is implemented since XEROX 1186/1109 only has black and white.

- Coordinate system
  The transformation pipeline is not fully implemented see Transformation pipeline.

- Inquire functions
  Only a few inquire functions are implemented.

- Miscellaneous
  Simple error handling
  No hidden lines
  No meta file
7 Parts of the ALPHA system

Workstations
A workstation consists of a number of attribute registers, a collection of windows, a number of input devices and a memory to hold references to posted structures. Windows are implemented as Interlisp-D windows. Beside the ALPHA package there is a package which makes it possible to use all Interlisp-D window properties. To display a structure, it must be posted to a workstation. The workstation will then execute the structure at a time determined by the workstations update mode, and the graphics will show in the window(s) belonging to the workstation. If a posted structure is changed (edited) the changes will show when the workstation reexecutes the structure. The time when this happens is determined by the update mode of the workstation. In ALPHA a workstation has four update modes:
- **ASAP** (As Soon As Possible): The structure is reexecuted immediately after it is changed.
- **BNIG** (Before Next Interaction Globally): The structure is reexecuted before any interaction on any workstation takes place.
- **BNIL** (Before Next Interaction Locally): The structure is reexecuted before any interaction on this workstation takes place.
- **WAIT**: Do not execute until explicitly told (with the UPDATE function)

To change the update mode of a workstation, use the SET-DISPLAY-UPDATE-STATE function.

Bundles
A bundle is a collection of values belonging to a graphic output operation. When a bundle is called, eg. "SET-LINE-INDEX 2" is executed in a structure, the values of bundle #2 are copied to the attribute registers of Line (Linetype and Linewidthscalefactor). These values can subsequently be overwritten by individual attribute changing instructions such as "SET-LINE-TYPIE", but the bundle values can always be restored with another "SET-LINE-INDEX".

The idea of bundles is to provide convenient grouping of attributes, eg. a dot-dashed line is defined as line bundle #4.

Attribute inheritance
Every graphic output operation has a number of attributes, eg. "Line" have the attributes Linetype and Linewidthscalefactor. The values of these attributes are fetched from the corresponding attribute registers of the workstation which the structure is executed in. The values of the attributes are dynamically bound at the time of execution. This means that if structure FOO sets Linetype and then calls structure BAR, then BAR will inherit the register value set by FOO. If BAR sets Linetype to a different value, then, upon returning from BAR, the value will be restored to the value it had when FOO called BAR. One way to look at this is that a structure always pushes the attribute registers on a stack before calling another structure and then restores them from the top of the stack when the called structure has returned.
Transformation pipeline
All coordinates given in graphic output operations are transformed via three transforms: the modelling transformation, the global transformation and the workstation translation and scaling, i.e. only the modelling coordinate system, the world coordinate system and the device coordinates are used, see figure II:3.

![Diagram of transformation pipeline]

The modelling and global transformations are 3x3 homogeneous matrices, which can perform arbitrary rotation, translation and scaling. The workstation scaling and translation are view (window) specific and consist of multiplications and additions, respectively, with scalars. When a structure calls another structure, the inherited global transformation becomes the product of the current modelling and global transformations, and the new modelling transformation is set to the identity matrix. The consequence of this is that every structure operates in its own local coordinate system which can be arbitrarily translated, scaled and rotated by a calling structure.

Structures
A structure can be viewed as a program, consisting of instructions understood by the ALPHA-workstation. The instructions usually set the values of attribute registers, perform graphic output operations such as drawing a line, or calls other structures. Structures are stored partly as lisp structs and partly as runnable lisp functions (see Chp III 7 Structures) The lisp structs stores information about children to the structure, which structures it is called by, to which workstations it is posted, etc. The lisp function generates the code necessary to display the primitives in the structure.
When structures are executed they build a tree structure in which all graphical information lies (see Chp III 10 Tree structure).
Graphical primitives
The graphical primitives implemented are adapted to the graphical package in the Interlisp-D environment.

Predefined line and edge types are:
• 1 straight
• 2 dashed
• 3 dotted
• 4 dash/dotted
The edge flag is set to T, i.e. the predefined edgetypes are visible.

Predefined text attributes are:
• 1 Gacha 10 MRR
• 2 Helvetica 5 MRR
• 3 Helvetica 10 BRR
• 4 Helvetica 12 MRR
• 5 Timesromand 36 MRR
The tripples "MRR"/"BRR"are Interlisp-D abbreviations for how the font will be displayed (see [15] 27.12 Fonts). The text attributes are adapted to the Interlisp-D environment. Other fonts available to use are those defined in the Interlisp-D package.

Predefined patterns are:
• 1 whiteshade
• 2 grayshade
• 3 blackshade
The Interlisp-D function for editing shades are ment to be used by the user. The Interlisp-D function for editing bitmaps are also possible to use, although they resist outside ALPHA.

ALPHA’s user needs inquire functions for reading data, since functions like that don’t exist in PHIGS (or in ALPHA), this implies that important data has to be stored by the application.
Chapter III
Technical documentation of the
PHIGS implementation ALPHA

ALHPA is a detached CommonLisp package in the Interlisp-D environment. The package exports all Alpha functions to be used and only imports necessary Interlisp-D functions. The fundamental conceptual objects (machine, workstation ...) are implemented as CommonLisp structs. All ALPHA instructions are implemented as functions that operate on these objects.