Semper 6 *Plus*

ADVANCED

USER

GUIDE

Synoptics

Trademarks:

Semper is a registered trademark of *Synoptics Limited*

Manual number: SEM010
Update number: 0
Printed: June 1990

Table OF CONTENTS

Chapter 1: Overview

Chapter 2: Semper Elements

Chapter 3: The Command Interpreter

Chapter 4: Pictures

Chapter 5: Devices and Storage

Chapter 6: Important Variables

Chapter 7: Indirect Interpretation

Chapter 8: The User Interface System

Appendix A: Sample Semper Programs

Appendix B: Glossary of Terms

Chapter 1

OVERVIEW

1.1 About this manual

Semper can be considered to be organised into four main parts:

- a command interpreter
- a picture filing and display system
- a configurable user interface
- a set of utilities (processing modules)

This manual describes the first two parts in detail, and also the syntax of the commands used to invoke the processing modules. It provides a full account of the Semper working environment, and should be read by any user who expects to do more than imitate a handful of basic commands suggested by other users.

1.2 What else to read?

This should not be the first manual that you read, however the:

Beginners' User Guide

is the document which you should consider first if you are a new user, as it is considerably shorter than this, less formal but more obviously related to the questions you are likely to need answered quickly. If you like to work by adapting examples you may also find that the following is helpful:

Quick Reference List

This manual lists all commands devoting a few lines to each. For a detailed description of the programming of user interfaces, consult the:

User Interface Guide

The above manuals are contained in the *Semper 6 Guide.*

1.3 How this manual is organised

This manual is divided into 8 chapters, 2 appendices and an index. This chapter provides an overview of the Semper environment. The remaining chapters and appendices are as follows:

- *Chapter 2: Semper Elements*

  Introduces Semper's image processing environment, describing input, output and resources. Explains Semper elements such as names, variables and functions.

- *Chapter 3: The Command Interpreter*

  Defines Semper command syntax (keys, options and defaults), describing the most useful Semper commands.

- *Chapter 4: Pictures*

  Explains each aspect of a Semper picture, ranging from picture classes, coordinates and origins to titles and write-protection.

- *Chapter 5: Devices and Storage*

  Tells you how a picture is stored on disc and tape, how the Semper display works and how to alter the viewing conditions.

- *Chapter 6: Important Variables*

  Concentrates on the most important and widely-used variables and explains how to save time by setting global keys and options.

- *Chapter 7: Indirect Interpretation*

  Details how to combine commands into programs, run files and macros and how to write your own startup file.

- *Chapter 8: The User Interface System*

  Provides pointers on how to set up your own menu interface to Semper.

- *Appendix A: Sample Semper Programs*

  Gives practical examples of Semper programming.

- *Appendix B: Glossary of Terms*

  Explains words and phases commonly used in Semper and image processing.

1.4 Conventions used in this manual

Semper commands are shown in the following font:

```
semper
```

except when they are embedded in the text, in which case they are highlighted using a **bold** font. All commands are shown in their *unabbreviated* form, that is, **panel** instead of **pan**.

> Material that can be omitted on a first reading is printed at the right hand side in shorter lines, like this.

Chapter 2

SEMPER ELEMENTS

2.1 Overview

This chapter provides a framework within which to understand Semper. It describes the Semper image processing environment, detailing input, output and resources. It explains the elements that Semper makes use of in its processsing – names, variables and functions.

2.2 The Semper environment

A Semper session runs as a single program (task, load module etc.), with various resources assigned as input/output units (channels, streams, devices, etc). The following are the ones required when Semper starts up:

- **terminal input** — For the user's commands; normally a terminal, but some other source in batch mode.
- **terminal output** — For user prompts, results and error messages; normally the same terminal, but a printer or printer spool file in batch mode.
- **startup run file** — Containing a series of Semper commands that initialise a session to suit the individual user.
- **error message file** — System error message texts; shared by all users.
- **work disc** — A random-access storage for miscellaneous purposes; not usually evident to the user.

Figure 2-1 illustrates the Semper working environment.

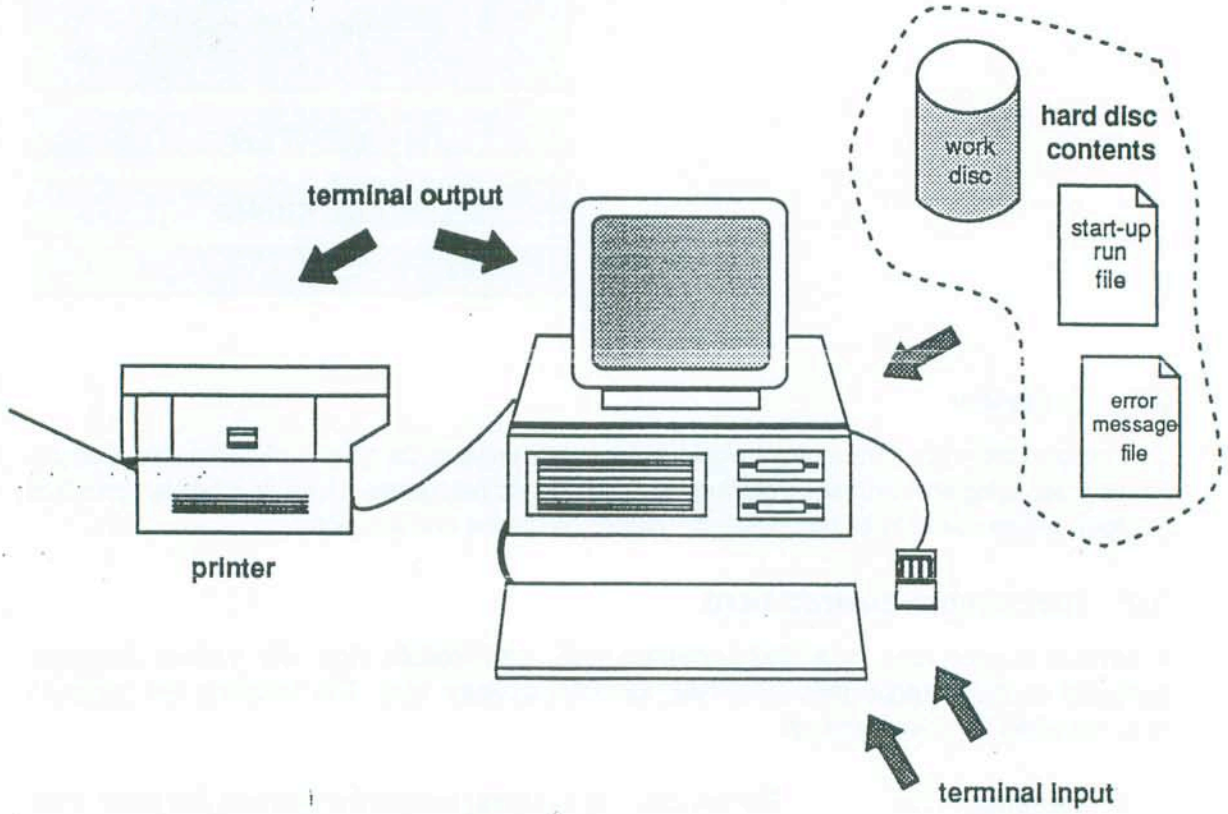

Figure 2-1. The Semper Environment

Further resources may be assigned and deassigned dynamically during a session:

- **picture discs** — Random-access storage for many individual pictures, with an internal directory; size and number chosen by the user.
- **display** — A display device; ideally of the framestore type, but with many possible alternatives.
- **magnetic tapes** — Serial storage for many individual pictures; for back-up or interchange with other computers or installations.
- **program libraries** — Collection(s) of Semper programs (sequences of Semper commands for deferred or indirect execution). Typically one such library is shared by all users, and users assign further libraries of their own.

- **help libraries** — Documentation file(s) available on-line, prepared and maintained by an independent help library manager utility; one such library is shared by all users, but individuals may assign additional libraries of their own.

- **log files** — Files for recording all the various forms of text output from Semper, for example, console output, results, diagnostic messages, etc.

Figure 2-2 illustrates these Semper resources.

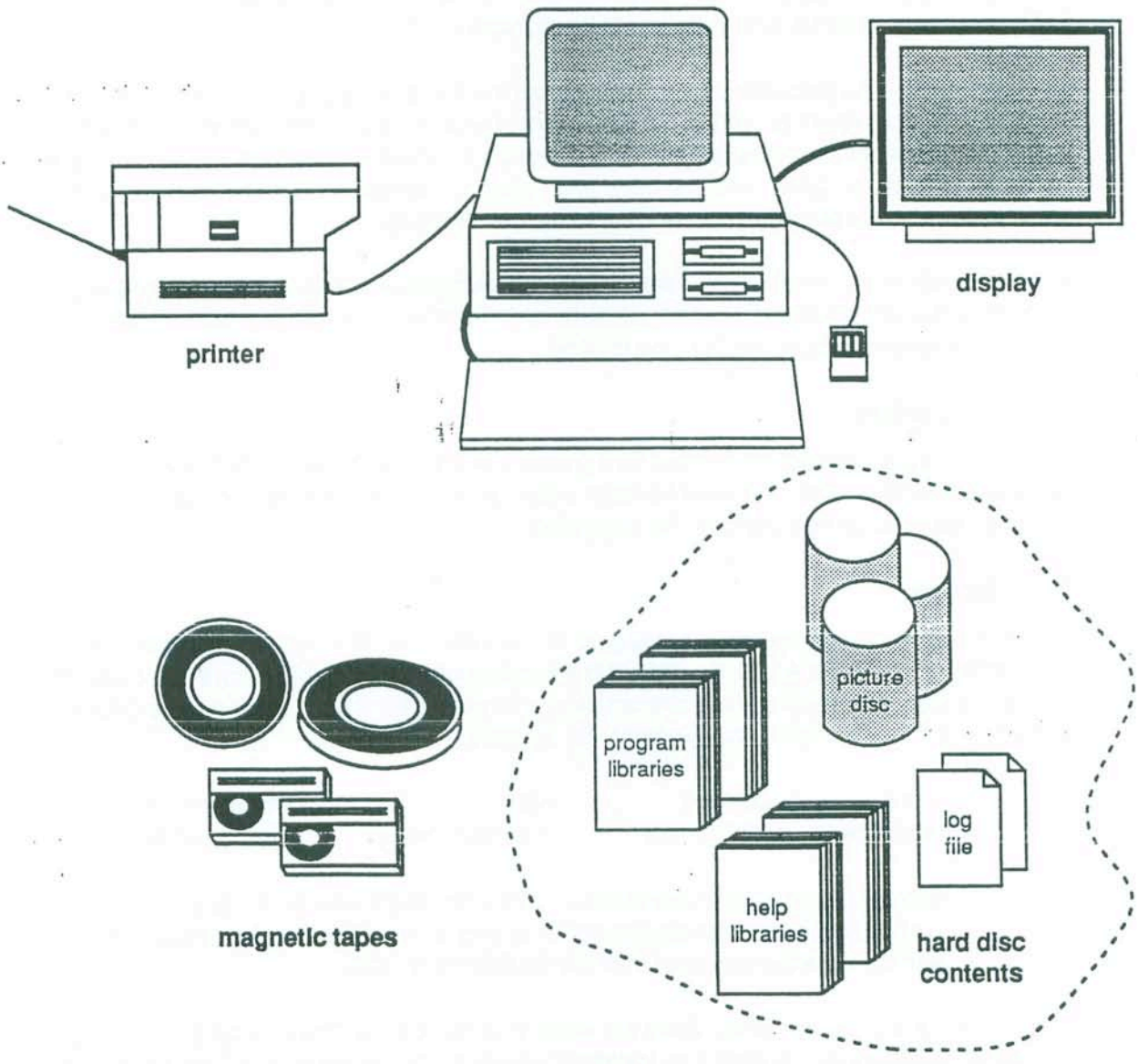

Figure 2-2. Semper Resources

Semper can run in three modes:

- **Interactive**
- **Indirect** or **deferred**
- **batch**

Initially, Semper accepts commands entered at a terminal and executes them immediately (or prints appropriate error messages); this is its *interactive* mode. Some commands cause it to read subsequent commands instead from some other source such as a program in a program library, reverting to terminal input when these have been completed; this is the *indirect* or *deferred* mode. *Batch* mode is like terminal mode, except that the terminal input and output units are assigned to the batch command unit and printer spool which the local operating system may provide. Execution in batch mode may be direct or indirect just as for interactive use.

Most commands – the processing commands – in fact invoke particular processing modules, either supplied with the system or written by local users. These have a simple universal syntactical pattern. A further set of commands combine to provide a complete programming language with variables, arithmetic, loops, routine calls etc., used to organise the individual processing commands into more powerful tools without losing their flexibility.

In most installations, one particular terminal key (for example, <control-c>) can be used to ask Semper to abandon whatever it is doing (whether in interactive or in indirect mode) and revert at once to taking fresh instructions from the terminal.

2.3 Characters

Internally, Semper operates with the 95 ASCII graphic character set, though some of the codes may be reassigned locally. The difference between upper and lower case letters matters only in literal text (used for titles, display lettering, file names etc).

2.4 Names

Names are used extensively when building up Semper commands. A name may consist of any combination of the letters A to Z, the digits 0 to 9 and the dollar character $, except that it may not start with a digit. Only the first three characters of a name are significant: more may be typed for readability, but they are ignored by Semper. For example:

X (=x)	*D3 (=d3)*	**acf**	*minimum (=min)*
next (=nex)	*theta (=the)*	*maximum (=max)*	*$count (=$co)*

Semper makes special use of some names that begin with the dollar character. You are free to use names beginning with $, as long as you make sure that they do not clash with the ones Semper uses (see the file *SEMPER.VDS*).

When it comes to program names, there are virtually no restrictions. They are arbitrary strings of non-blank characters with a maximum, installation-dependent length of at least 8 characters (the command **show system** will tell you what the maximum length is). You use program names to invoke Semper programs stored in a program library (see *7.2, Programs*).

2.5 Numbers

Numbers are used to represent constant numerical values. A number consists of a string of digits with or without a decimal point. A plus or minus sign may optionally precede it and an exponent (as a power of 10) may follow it. The exponent is an integer number preceded by the letter *e*. For example:

```
6          (= 6. = 6.0 = 06.00)
39461      +12          –360
0.61       (= .61 = 6.1e–1)
11.36      (= 1.136E1 = 1136e–2)
–1e–13     (= –.0000000000001)
```

Binary, octal and hexadecimal number representations can also be used. A string of appropriate binary, octal or hexadecimal digits (no decimal point or exponent is allowed) must be prefixed with the characters **0b** for binary numbers, **0o** for octal numbers, and **0h** or **0x** for hexadecimal numbers. For example:

```
0b11001      (= 25)
0o567        (= 375)
0h1de        (= 478)
0xfa2b       (= 64043)
```

Numbers are represented internally in floating point form, typically giving six significant figures of precision. Where only an integer value is appropriate, the number is rounded to the nearest integer.

2.6 Variables

Like most programming languages, Semper makes considerable use of variables. Each variable has a name and an associated numerical value which you can set, reset or unset as required. Variables are used to hold and manipulate values for various purposes. They allow you to pass information to the processing modules and also to get back information. Semper maintains a list of the variables set at any given time, together with their values. A variable not in the list has no value at all, and is said to be *unset*. This is the state of most variables at the start of a Semper session.

There are variables that exist at all times: these are called *protected* and *fixed variables* (See *6.2, Protected and Fixed Variables*). You are not allowed to reset or unset any of the protected variables and you cannot unset any of the fixed variables.

2.7 Indexed names

Names of one or two characters may be followed, with no intervening spaces, by an index of the form *#name*. The name in the index must not itself be followed by an index. When an indexed name is encountered, the value of the variable named in the index is examined, and provided it lies between 0 and 99, its decimal character representation is added to the original name to give the actual name. Thus, if *n* is 32, the names *b#n* and *x1#n* refer to *b32* and *x132*. This allows you to set up short arrays with up to 9 elements. It also makes constructions such as computed jumps possible, for example,

the command **jump dd#n** refers to *dd1*, *dd2*, and so on, according to the value of the variable *n*.

2.8 Functions

Function calls return a numerical value. Each call consists of a function name followed by one or more argument values enclosed in brackets, with commas separating multiple arguments. For example:

exp(3) *root*(x) *sin*(*theta*) *mod*(3.6,–12.3)

The full list of recognised functions is given below:

sin(x)	sine (*x* in radians)
asin(x)	arcsine (result in radians), for *mod(x) <= 1*
cos(x)	cosine (*x* in radians)
acos(x)	arccosine (result in radians), for *mod(x) <= 1*
tan(x)	tangent (*x* in radians)
phase(x)	arctangent (result in radians)
phase(x,y)	*arg(x + iy)* (result in radians), *tan(phase(x,y)) = y/x*
exp(x)	exponent (*e* to the power of *x*)
ln(x)	ln (natural or Napierian log), for *x >= 0*
min(x,y)	lower of the values *x* and *y*
max(x,y)	higher of the values *x* and *y*
mod(x)	modulus (absolute value)
mod(x,y)	modulus(*x + iy*), *mod(x,y) = root(x*x + y*y)*
msq(x,y)	modulus squared, *msq(x,y) = x*x + y*y*
root(x)	square root, with *x >= 0*
rem(x,y)	remainder when *x* is divided by *y*, *rem(x,y) = x – fix(x/y)*y*
round(x)	nearest integer to *x*
fix(x)	next integer towards zero from *x*
p(x,y,z)	value of pixel at *x,y,z* of current picture (see *4.6, Direct Pixel Access*)
re(x,y,z)	real part of pixel, same as *p(x,y,z)*
im(x,y,z)	imaginary part of pixel
and(x,y)	bitwise AND of *round(x)* and *round(y)*
or(x,y)	bitwise OR of *round(x)* and *round(y)*
not(x)	bitwise NOT of *round(x)*
rad(x)	angle in radians equivalent to *x* degrees
deg(x)	angle in degrees equivalent to *x* radians
ifelse(x,y,z)	returns *y* if *x* is non-zero and *z* if *x* is zero.
set(name)	returns 1 if named variable is set, and 0 otherwise (see *3.7, Conditional Clauses*)

Semper will not confuse function calls with variables of the same name. For example, *min*(x,y) and *min* may be used quite independently.

2.9 Expressions

Expressions can be used almost anywhere that Semper expects a simple numerical or logical value. A logical result is represented by the numerical value 1 for true and 0 for false. If a numerical value is encountered where a logical value is expected, it is taken to represent false if zero and true if non-zero.

Numbers, variables and function calls are the basic arguments in expressions. These all supply an immediate numerical value. An expression may also appear as an argument in another expression and must be evaluated first, giving a simple numerical value that is used in evaluating the next expression. Expressions may be nested in this way at least up to six deep.

An expression has either one argument which may be preceded by a unary operator (for example, ^4) or else two arguments separated by a dyadic operator (for example, 4+4). There are three types of expressions:

- **arithmetic**
- **relational**
- **logical**

Arithmetic and relational expressions require numerical arguments while logical expressions require logical arguments. The result for an arithmetic expression is a numerical value, while the result for a relational or logical expression is a logical value. The type of expression depends on the operator specifying the operation to be carried out:

Arithmetical * / + – ^ :

Relational < > = ~= <= >=

Logical ~ & |

The operators +, – and : may be used as unary or dyadic operators. The ~ operator can be used only as a unary operator. The operator ^ is used to denote exponentiation (raising to a power). The : operator is a special one for combining device and picture numbers into a single numerical value (see *5.2, Picture numbers in processing commands*). If the first argument is omitted, the value of the variable *cd* is assumed. The operators + and – when used as unary operators are treated as dyadic operators with the first argument defaulting to zero, that is:

+4 = 0+4
–pi = *0*–pi

The logical operators ~, & and | specify the logical operations NOT, AND and OR respectively. They all require logical arguments, but if a numerical value is encountered, its logical meaning, as described above, is used.

The order in which nested expressions are evaluated depends on the relative priorities of the operators concerned. An expression is evaluated first if the operator it depends on has the higher

priority. Where there is no difference in priority, the expressions are evaluated from left to right. Brackets can be used freely to control the order in which expressions are evaluated. The priority of the operators is as follows, with equal priority operators on the same line:


```
:
^
*   /
+   -
<   >   =   ~=  <=  >=
~
&
|
```

Expressions therefore provide a powerful and general way for combining and manipulating numerical data. Note that function arguments may themselves be entirely general expressions. A number of examples can be given to illustrate the the range of possibilities:

56	–1.5	*x*	192*64
n+3	0.5–*x*	*min**2.4	*mod*(*x*,*y*)/5
(*a*+*b*)/2	*scale**(*r*+.2)	1/(*x***x*+*a***a*)	*exp*(–*msq*(*x*,*y*)/50)
(*x*–*fix*(*x*/*y*)**y*)	*deg*(pi)	*sin*(*rad*(60))	~*set*(*erase*)
x+*y*<=5	*a*<1|*a*>10	*and*(*or*(*not*(*a*),*not*(*b*)),or(*a*,*b*))	

2.10 Literal text

A string of characters enclosed in single quotes comprises literal text in Semper. The single quote character itself is represented by two successive quotes in a string. Examples of valid strings are:

'Original' 'didn''t work' 'SEMPER$DISK' 'Plate #32'

Chapter 3

THE COMMAND INTERPRETER

3.1 Overview

The Semper command interpreter supports both very primitive and very elaborate operations on pictures. It supplements the commands that run the various processing modules with the normal facilities of a high level programming language, so that sophisticated operations can be defined in terms of simpler ones. The command interpreter is also designed to make it easy to extend, allowing you to add your own commands where greater speed or convenience is required.

3.2 Command line structure

Semper commands involve strings of names, numbers and other characters as described in more detail below. Here is an example of a command that you might enter at the terminal after the **S$** prompt string:

```
rank 52 to display
```

Several commands may be placed on a single line, with a semicolon separating them, and are then said to form a *command line*, for example:

```
survey 42; copy to display
```

You can use blank lines and extra spaces freely (except within names, numbers or literal text) to improve the layout of commands.

Lines typed at a terminal or read from a file must not exceed 80 characters in length – characters beyond the 80th may be discarded without warning. However, you can continue a command line over several input lines by adding a plus sign + at the end of any line to be continued. You will then be prompted for more input with the normal prompt followed by a + character. The total length of the command line must not then exceed the size of the command line buffer (the command **show system** will tell you what this is). Note that in the following example, Semper prompts are shown in **bold**:

```
S$ xcf +
S$+ 1 with 2
```

is equivalent to the single line:

```
xcf 1 with 2
```

When entering commands interactively at the terminal, you have the facility to edit the command line and also to recall up to the last 20 command lines. You can move the cursor anywhere within the command line, insert, delete or replace characters and erase or refresh the entire command line. Being able to recall command lines is particularly useful when you mistype a command. You just recall the command you want to change, make the necessary corrections and then re-enter it. The keystrokes for editing the command line may vary from one system to another, but they will usually be the same as those provided by the host operating system. If in doubt, use the command **help.editing** to find out what they are.

3.3 The STOP command

A session is terminated by the simple command **stop**; also by the commands **quit** and **exit**, by the word **end** encountered as a command, and by an *end-of-file* condition in the terminal input where this can be detected.

3.4 Assignments

Assignments set variables to particular numerical values, and have the form *variable=expression*, for example:

```
k3=-11.36
abc=r/3
x=-x; y=-y; z=0
x=-x y=-y z=0
```

Several assignments may appear in a single command as in the last example. They may also appear prefixed to (but not embedded in the middle of) other commands, for example:

```
min=0 max=max/2 rank to display
```

3.4.1 Multi-component assignments

A special feature of assignments allows convenient setting of pairs or sets of values, such as complex numbers or 2-D and 3-D vectors. If an assignment contains more than one expression, separated by commas, the first value is assigned to the named variable and the remainder are assigned to associated variables, whose names are generated by adding 2, 3.. to the end of the first (or replacing the third letter if the name is already three letters long). For example:

```
a=2,3 abc=5,4,3
```

is equivalent to:

```
a=2 a2=3 abc=5 ab2=4 ab3=3
```

Up to 9 variables can be set with a single assignment in this way.

3.5 Processing commands

Most Semper commands are processing commands. These invoke the processing modules that actually process, analyse, combine and generate pictures. An initial set of such modules/commands is a standard component of Semper, but widely different sets may be added later – perhaps developed by local users. All such commands share the common syntactical construction explained here.

Processing commands consist of a *command name*, identifying the action to be performed, commonly (but optionally) followed by a list of *option names* and *keyname-expression* pairs, which identify the operands and/or give details of the action to perform. For example, the command:

```
mask inside radius 35
```

is a use of the command **mask** (which imposes a circular mask on a picture, resetting to a constant all points inside or outside a certain cut-off distance from the origin), quoting the option **inside** and the key **radius** with the value 35.

A command name identifies a particular processing module to be run in response to the command; it also identifies a particular set of options and keys allowed. Quoting an option or a key causes the variable with the same name to be set for the duration of the command; the processing module consults the variable in determining some detail of the process it is effecting. The keys and options may be quoted in any order, but a key and its value must not be separated.

3.5.1 Options

Options refer to features of an on/off or yes/no character – in the example above, whether the points reset are to be those within rather than beyond the cut-off distance. An option name causes the variable of the same name to be set to the value *yes* (= true or 1), forcing the corresponding feature on. The same option name with **no** prefixed causes the variable to be be set to *no* (= false or 0) instead, forcing the feature off when it would otherwise be assumed. When using the **no** prefix, you must still remember to type the first three characters of the option name. For example:

```
survey noverify
```

Note that the command interpreter will accept option names abbreviated to three characters, but only after it has stripped away the **no** prefix. This means that in the above example the option can be shortened to **nover**, but **nov** would not work as this would imply setting the variable *v* to **no**.

3.5.2 Keys

Keys refer to features that involve a numerical value – in the example, how large the cut-off radius is to be. The variable with the same name is simply set to the value given following the key. The value may be given as an expression, for example:

```
mask radius mod(x,y)
```

A few keys accept more than one value expression, separated by commas, exactly like multi-component assignments (see *3.4.1, Multi-component assignments*). A common example is **size** which is widely used to indicate picture or subregion dimensions:

```
extract size 300,250..
```

Although the same variables are involved, quoting options and keys differs from assigning the variables (for example **inside**=yes; **radius**=35) in several ways:

- The setting is local to the command (the original state of the variable – whether set, and if so with what value – is remembered by the interpreter and restored after the command).
- The setting is faulted if it does not involve one of the options or keys expected for the command in question; and the interpreter provides a default setting (specific to the particular command and key) for many keys when they are not otherwise specified.

There are thus several levels at which an option or key may be set:

(a) it may be quoted explicitly in the command:

```
mask radius 20
```

(b) it may be assigned earlier to a value which then serves as a global default:

```
radius=20
mask
```

but may be overridden whenever necessary by an explicitly quoted local value:

```
mask radius 35
```

(c) if (a) and (b) do not apply, a default may be provided by the interpreter:

```
extract size 300
```

causes key **sl2** to be assigned the default value *size*. Some defaults are in fact provided by the module invoked, usually because the default value depends on special factors that the command interpreter is not aware of.

You should assign global defaults with care, as confusing results can very easily be produced. For example, setting the **verify** option to *yes* will have the unexpected effect of turning on the **vertical** option in the command **project**.

This document is concerned only with the syntax or grammatical construction of command lines. Refer to the manual:

Semper 6 Command Reference

for a full description of all the commands and their options and keys, and the on-line help

documentation. For a brief description of each command and examples, see the manual:

Quick Reference List

This manual is contained in the *Semper 6 Guide.*

The command **show commands** lists all of the commands that Semper recognizes and you can use the **syntax** command to find out what options and keys (and their defaults, if any) apply to each command, for example:

```
syntax mask
```

lists the following information at the terminal:

```
Options:
ins out
Keys:
val va2 rad pos po2 wid wit $1=sel from=$1 $2=fro to=$2
Picture opens:
(lp1,old)=fro (lp2,new,lp1)=to
```

The set of processing commands recognised by the interpreter, their options and keys, and some other details, are all established in the *Verb Descriptor* file *semper.vds* from which the system generation program *SEMGEN* generates the main program which is linked together with other system and processing modules to form the complete program actually executed. Details of this process, and how new modules can be written and incorporated as new processing commands, are given in the *following manual:*

Semper 6 Plus Fortran Programmers' Guide.

Some commands accept a number of unkeyed values, for example:

```
erase 1001
```

The interpreter in fact assumes the keynames *$1, $2* ... for such values. A keyname becomes optional if an unkeyed value is defined as its default value, for example, the keyname **number** can be omitted from the command **lut number 1**, because *$1* is the default for number. Keys like **number** are called *assumed keys.*

Some commands, such as **help, type** and **local** have their own special syntax which the **syntax** command will not be able to report. These commands will appear in the *Verb Descriptor* file with just a *$* option beside them. Note that the conditional commands **if** and **unless** are special in that they do not even appear in the *Verb Descriptor* file.

3.6 The TYPE and LOG commands

The **type** command outputs numerical values and text to the terminal (strictly speaking, the console output stream – *see 3.16, Controlling output in Semper*), for example:

```
type pi
type 'Iteration complete'
```

Several items to be typed may be listed, separated by commas. The items may be expressions (which of course includes variable names and numbers) or literal text, for example:

```
type 'Pi squared is ',pi^2
```

Values are typed correct to six significant figures, with no leading or trailing spaces, apart from single spaces to separate adjacent values, for example:

```
type min,max
```

The **log** command outputs messages in exactly the same way as the **type** command, but to the log output stream instead of the console output stream (see *3.16, Controlling output in Semper*), for example:

```
log 'Focus ',z,' and brightness ',b
```

A third kind of item provides a limited form of tabulation control: *#n*, where *n* is an expression (or variable name or number), resets the column number (forwards or backwards as desired) to *n*; thus, for example:

```
type #10,n,#30,m,#20,l
```

produces the values of *n*, *l* and *m* left justified to columns 10, 20 and 30 respectively.

3.6.1 Textstrings

There are in fact several other contexts in which Semper accepts a list of items with the same syntax as **type** requires; for convenience, such lists are called *textstrings*, and they are evaluated in the same way as **type** evaluates them before use.

A few keys to processing commands in fact accept a textual rather than a numerical value, and this is one of the contexts in which a textstring is accepted. Two examples follow:

```
assign new name 'pictures\demo' size 2.5

title text 'at temperature ',temp
```

It is impossible to assign global defaults for textual keys, as variables can only be assigned numerical values. The **syntax** command lists textual keys with a quote character instead of a default value, for example **syntax title types tex='** for the **text** key.

3.7 Conditional clauses: IF and UNLESS

The execution of a Semper command can be made conditional by prefixing an appropriate **if** or **unless** clause, for example:

```
if x>3 type 'X is too large'
unless abs(e)<.1 n=n+1
if ramp type 'Ramp removal requested'
if x<3 | x>10 type 'X is out of range'
```

Syntactically, a conditional clause consists of an **if** or **unless** followed by an expression treated as a logical value (see *2.9, Expressions*). The command governed is executed only if the expression is true in an **if** clause or false in an **unless** clause. Multiple conditional clauses are allowed and are interpreted in turn. If all the clauses are satisfied, the command is executed, otherwise the interpreter moves on to the next command as soon as the first invalid clause is encountered, for example:

```
if ramp unless mean>5*sd ..
```

The function set allows conditional clauses to test whether particular variables are set at all, for example:

```
S$ if set(focus) ..
```

The value of *set(..)* is in fact simply that of the argument expression evaluated using 0 for any variable which is not set and 1 for any variable which is set. For example, *set(plot)|set(out)* and *set(plot|out)* both test whether either of *plot* or *out* is set.

3.8 Simple loops: the FOR command

The **for** command provides a simple way of repeating a sequence of commands for a specified number of times. For example:

```
for x=1,6; type x,root(x); loop
```

causes the numbers 1,2,3,4,5 and 6 to be typed together with their square roots. The inclusion of the equals sign = in the **for** command is optional.

The **for** command cannot be executed until a corresponding **loop** command is encountered. If the **loop** command is not found in the command line, the command interpreter will prompt you for more commands until it finds the **loop** command that it requires. For example, the first example could have been typed in as follows (computer output is shown in **bold**):

```
S$ for i=1,6
(For 1) S$ type x,root(x)
(For 1) S$ loop
```

This is a convenient way to input long or complicated loops. An *abandon request* will get you back to normal command input if, after typing a **for** command, you change your mind and want to break out.

In general, a **for** command causes all subsequent commands, until a corresponding **loop** command, to be executed repeatedly, with the variable named in the **for** command (the loop variable) taking in turn a series of values. You specify an initial value and a final value, optionally followed by an increment value (all separated by commas). The increment value defaults to +1, if the final value is greater than the initial value, and −1, if the final value is less than the initial value, for example:

```
for i=4,0; ... ; loop
```

loops five times, with the loop variable taking the values 4, 3, 2, 1 and 0.

The loop is executed just once, if the initial and final values are equal. If the increment value is of opposite sign to the difference between the final and initial values, that is, the loop variable would be incremented away from the final value, the loop is bypassed altogether. The loop variable is set to the initial value for the first loop. For each subsequent loop the increment value is added to it, until its value exceeds the final value, at which point the loop is terminated. For example:

```
for pic 20,10,-3; type pic; loop
```

types the values 20, 17, 14 and 11.

Loop variables are local to the loop, that is, the original state of the loop variable (whether set, and if so with what value) is noted at the start of the loop and restored on its completion. The variables *x* and *y* might be used freely as loop variables, for example, without interfering with a **position** (*x,y*) previously recorded in these variables.

Resetting the loop variable within a loop, for example:

```
for n 20,30; ... ; n=p; ... ; loop
```

does not alter the value assigned to it at the start of the next loop, that is, its value is restored at the end of each loop to what it was at the start of the same loop.

Where the loop variable takes non-integral values, rounding errors may make execution unpredictable in different installations, for example:

```
for n 0,1,1/7; type n; loop
```

may or may not type the last value (1).

Loops may be nested, up to an installation-dependent maximum depth (the command **show system** will tell you what this is). For example:

```
for p=2,5; type 'Powers ',p; for n=1,6; type n,n^p; loop; loop
```

types four tables, giving the squares, cubes, fourth and fifth powers of the numbers 1 to 6. When loops are typed interactively, their total length is subject to the same limit as a single (continued) command line. When typing commands inside a loop, the terminal prompt reflects the loop level, for example:

```
S$ for p=2,5
(For 1) S$ type 'Powers ',p; for n=1,6
(For 2) S$ type n,n^p; loop
(For 1) S$ loop
S$
```

When loops occur in programs (see *7.2, Programs*), there is no restriction on their length. In run files (see *7.3, Run files*), the interactive limit still applies.

As an aid to making lengthy loops more readable (especially in programs), the relevant loop variable name may be included after the **loop** command, for example:

```
S$ for p=2,5
(For 1) S$ type 'Powers ',p; for n=1,6
(For 2) S$ type n,n^p; loop n
(For 1) S$ loop p
S$
```

Where the name is present, the interpreter checks that it is correct. Conditional clauses should not be prefixed to any **for** commands, as this is liable to lead to **for-loop** mismatch errors.

3.8.1 Interrupting loops: NEXT and BREAK

Two commands are available for interrupting the normal flow of **for** loops. The command **next** *<name>* tells the interpreter to jump immediately to the next cycle (if any) of the named loop and the command **break** *<name>* causes a jump out of the named loop altogether, that is, to the command immediately following the terminating **loop** command. The commands can only be used inside the named loop, of course. The item *<name>* refers to the name of the loop variable. The innermost active loop is assumed if the loop variable name is omitted. For example:

```
S$ for y=-5,5; for x=-5,5
(For 2) S$ if mod(x,y)>5 next; .. ; loop; loop
```

executes the commands .. for all positions (*x,y*) within 5 units of the origin.

3.9 Labels and jumps

jump commands allow the interpreter to be diverted from its normal pattern of executing commands in the order in which they are entered. A position in a command line can be marked with a label consisting of a prefixed name and colon, for example:

```
next: Type 'Beginning next phase'
```

and a **jump** command quoting a given label name, for example **jump next**, causes interpretation to resume at the labelled position. Prefixing a **jump** command with a conditional clause allows the order of execution to be data-dependent, for example:

```
i=0; n: i=i+1; .. ; if i<100 jump n
```

repeats the enclosed commands 100 times with *i* taking the values 1, 2.. in turn.

These constructions are of limited value for interactive use, as **jump** commands can then refer only to labels within the same (continued) command line, and references to labels not found in the command line are faulted. In programs and run files (see *7.2, Programs* and *7.3, Run files*) however, there is no such limitation, and **jump** may transfer control anywhere within a given program or run file.

The **jump** command may be used inside a **for** loop, and it may jump to anywhere outside the loop and also to anywhere inside the loop, except another **for** loop.

Although labels with the same name are not faulted, the effect of jumping to such a label is not clearly defined. The use of repeated labels is not encouraged and may be faulted in a later version of Semper.

3.10 The NULL command

If a command begins with a number rather than a name, the command *null* is assumed by the interpreter. This in turn is normally defined to be synonymous with the **display** command, so that the picture number 2:53, for example, typed on its own is interpreted as if **display 2:53** had been typed, and causes picture 2:53 to be displayed.

3.11 Comments

Commands whose first non-space is an exclamation mark **!** are treated as comments by the interpreter and ignored completely; these are chiefly of use in indirect execution modes (see *7.2, Programs* and *7.3, Run files*).

The **!** applies only to an individual command, not to the command line, that is, a comment is terminated by a semicolon, so do not expect to be able to use semicolons within comments.

3.12 The ASK command

A further feature intended primarily for use in indirect modes of operation is the **ask** command, which causes the interpreter to type a prompt at the terminal and accept one or more numerical values from the user. The command name is followed by an optional textstring defining the prompt, followed by a (non-optional) list of variables to be set separated by commas; a space (not a comma) separates the variable list from any prompt. For example:

```
ask 'Block size for filter? ' b
```

types the quoted text as a terminal prompt, reads an expression (which of course may include a variable name or number) typed by the user in response, and sets the variable *b* to the corresponding value. As a second example:

```
ask 'Circle centre,radius for picture ',n x,y,r
```

types the value of *n* as part of the prompt, and accepts three expressions from the terminal, separated by commas, in response.

If no prompt is provided in the command, the list of variable names itself is used as a prompt.

Any variables listed for which no value is entered are unset by **ask**, and if the user types nothing but a return, all are unset; a program using an **ask** command can then provide defaults if it wishes, for example:

```
ask 'Angle (RETURN=>pi/2): ' t; unless set(t) t=pi/2
```

3.13 The LOCAL command

The last feature intended primarily for indirect execution modes is the **local** command, which allows a set of variables to be declared local to a particular command sequence or program in the sense that their original state (whether set, and the value if set) is restored at the end. The effect is as if fresh copies of the variables were made available that did not conflict with any previous use. The form of the command is simply the command name followed by one or more variable names separated by commas, for example:

```
local n,i,min,max
```

When the command is used interactively or in a run file (see *7.3, Run files*), the variables are restored at the end of the (continued) command line; when it is used in a program (see *7.2, Programs*) they are restored when the program returns.

3.14 Help

Semper provides comprehensive assistance during a terminal session by means of one or more help libraries. These are disc files containing indexed and cross-referenced information prepared by a separate library management utility. A summary of how help is used is typed in response to the simple command **help**. The **help** command accepts one or more strings, separated by spaces, causing any library entries with matching topic or alias names to be typed, for example:

```
help mask
help xwires mark
```

A successful match is established when all of the characters in a topic string are found to match an entry name, ignoring any excess characters in the entry name. You can therefore be as vague or as precise as you want to about the topics you wish to investigate.

For example:

```
help x
```

will list all topics beginning with x, while

```
help dis
```

lists a more restricted range of topics (that is, *disc* and **display**), and

```
help display
```

lists the information about the single topic **display**. Topic names can be broken down into several component parts with dots separating each part, for example:

```
xwires.region mask.syntax
```

If a dot is encountered in a topic string, the matching process continues after the dot in both topic string and entry name. There must be the same number of dots in the topic string and entry name for a successful match. For example, the entry **mask.syntax** would be matched by any of the following topics strings:

```
mask. .syntax . m.s ma.syn
```

but not by the topic strings:

```
mask m m.. m.s.
```

This construction makes it possible to select amongst several entries with the same basic name.

For example:

```
help display
```

would tell you about display devices as well as the display command. However, to obtain help on just the **display** command, you can type:

```
help display.command
```

For help on all the keys and options for a particular command, look for the corresponding **.syntax** entry, for example:

```
help display.syntax
```

The **help** command accepts a number of options, each of which must be preceded by a stroke / to avoid any confusion with topic strings. Options may appear anywhere after the **help** command. They can be shortened to a single character and they do not have to be preceded by spaces.

Information about a particular topic may start with a brief summary which is all that would normally be typed. If there is more information to come, you will see a short message to that effect at the end of the listing. If you repeat the **help** command using the **/full** option, all of the information will be typed. The **/log** option may be used at any time to divert the listing to the log output stream (see *3.16, Controlling output in Semper*). The command **help/log** by itself outputs the entire contents of the help library to the log output stream. The **/header** option suppresses all but the header line for each entry listed. This allows you to see what topics would be listed, without having to step through large volumes of text when a large number of entries are selected.

Each help library has a directory that records all of the entry names and their aliases. The command **help/topics** will type all of the entry names in alphabetical order. If you use the **/full** option as well, each entry is typed on a separate line along with any aliases that are recorded for that entry. The range of topics can be restricted by specifying one or more topic strings, for example:

```
help/topics a b c
```

lists all topics beginning with *a*, *b* or *c*, and

```
help/topics .command
```

lists all topics that describe commands.

Dots in topic names have the same effect as when listing help entries, except that unmatched dots in an entry name are ignored, that is, **mark** as a topic name would match entries **mark**, **mark.command** and **mark.syntax**.

The listings that the **help** command outputs to the terminal can be quite lengthy. If a listing fills the terminal screen, a prompt will appear giving you several options as to how to proceed. Pressing the <return> key will cause another screen of information to be typed.

Pressing the <space bar> causes the next line to be typed. The key **Q** enables you to quit from the **help** command altogether. The prompt string disappears when a key is pressed.

Help libraries are made available to a Semper session by the **assign help** command (see *Chapter 5, Devices and Storage*). When more than one help library is assigned, all are consulted. The normal arrangement is that one standard library supplied with Semper is supplemented by others describing locally written changes or additions.

A separate document, the *Semper 6 Plus Implementation Guide*, explains how the help library management utility is used to create new libraries, update entries, etc.

3.15 Errors and abandon requests

Errors occurring during the interpretation of Semper commands or the execution of the processing modules invoked result in:

- immediate termination of the operation in progress
- the abandoning of any loops in progress
- the abandoning of any indirect program (see *7.2, Programs*) being interpreted
- the typing of a brief error message on the terminal, with a short extract of the offending command where appropriate to clarify the context
- the typing of a 'traceback' of any program calls (see *7.2, Programs*) in effect at the time of error
- a request for fresh commands from the terminal

You can then enter corrected commands, or take whatever action is appropriate in the circumstances.

All errors have an associated numerical code (a positive integer) included in the error message. The command help **?n** types more details about the likely causes of error *n*.

You can force Semper to abandon whatever it is doing by pressing a particular key (typically <control-c>). After perhaps a short delay, Semper should respond in just the same way as if an error had occurred. Errors and abandon requests may both, of course, leave half processed pictures behind them.

You can re-display the text of the last error message (or the error message which would have been displayed if you have used **trap**) using the **report** command. The command **report error** will redisplay the last error message and the command **report trap** will display the last trapped error.

3.16 Controlling output in Semper

Many of the Semper commands output their results to the terminal, for example the **print** command lists picture values onto the terminal. In fact, Semper is completely flexible as to where it directs its output. There are two possible destinations for Semper's output:

- user terminal
- log file

Semper makes use of six logical output streams to distinguish between different classes of output:

• console	normal output
• diagnostics	error and warning messages
• log	log output – output not normally intended for the terminal
• command	command echoing
• input	all other non command input echoing
• monitor	debugging messages

Figure 3-1 illustrates the six logical output streams.

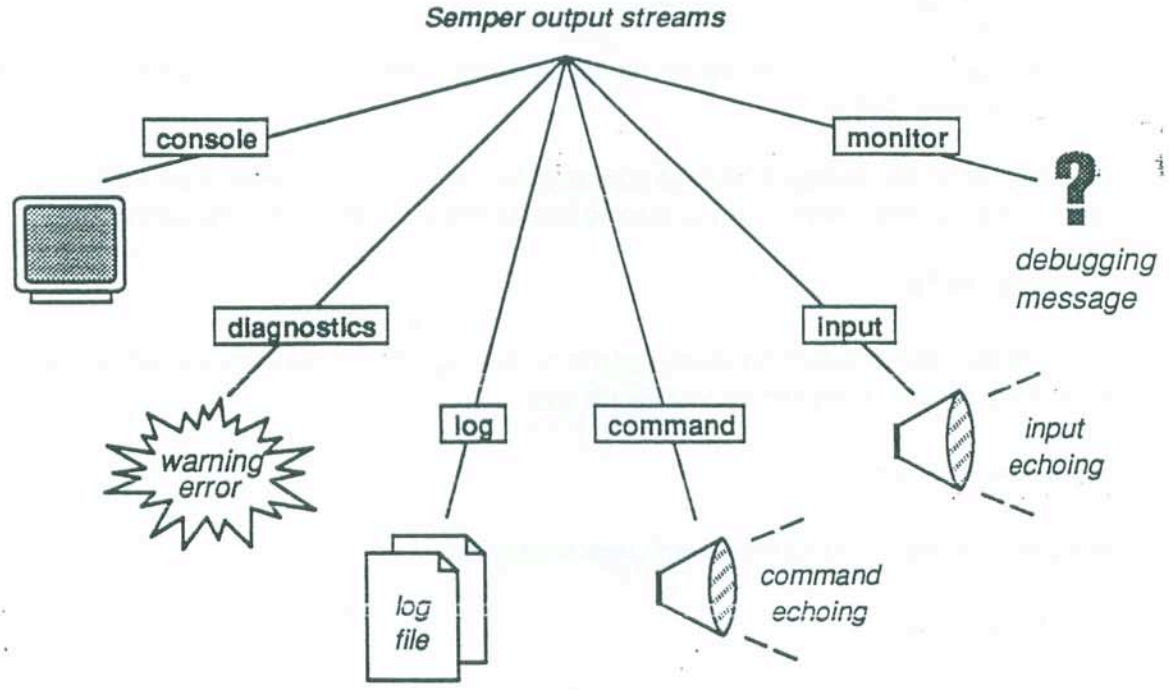

Figure 3-1. Logical Output Streams used by Semper

The **echo** command allows you to direct each of these individually to the terminal and also to any log files that are open. By default, the console, diagnostic and monitor streams are directed to the terminal.

When output is displayed on the terminal, Semper lets you control, in a number of ways, how that output is presented to you. Firstly the output can be paged – so that results do not scroll off the top of the terminal screen (this is the default). When Semper finds that a command has filled the screen with output, it will display a prompt message at the bottom of the screen and wait for you to press a key (or a mouse button if you have a mouse). You may move the display on one line, one page or abandon entirely by pressing the appropriate key (or mouse button).

If you do not want to have your output paged, you should type:

```
page noprompt
```

and to turn paging on again type:

```
page prompt
```

Semper also allows you to redefine the size of your terminal screen. Suppose that you want the output to be displayed in a region 60 characters wide by 10 lines deep, you would inform Semper by typing:

```
page width 60 length 10
```

The **page** command also lets you set up the character aspect ratio, if the default provided by Semper is inappropriate.

You may notice that Semper by default truncates lines which are too long to fit across the current width of the terminal. However, if you want to see the ends of these lines, you should type:

```
page wrap
```

After this, Semper will output the entire contents of each line onto as many lines of the terminal as is necessary. To turn off this facility, you should type

```
page nowrap
```

If you want to find out about the current page settings, type:

```
show page
```

This will give the current terminal dimensions, character aspect ratio, whether prompting is enabled and whether or not lines are truncated.

You can direct any combination of the logical output streams (console, diagnostics, etc.) into log files. Log files are opened by means of the **assign...file** command. For example:

```
assign file name 'results.out'; output = n
```

would assign a file called *results.out* for logging output.

When you assign a log file, the output is limited to a maximum width (the default is 132) but by using the **width** key with the **assign** command, you may alter this as you wish. If you want to add output to the end of an existing file you should specify the **append** option – otherwise Semper will overwrite the contents of the log file.

Having assigned the log file the next step is to direct output to it. This is achieved by means of the **echo** command. Suppose that you wished all console output to go to this file, you would type:

```
echo console device output
```

This does not stop console output from appearing on the terminal. If you want to turn off console output to the terminal, you should type:

```
echo noconsole terminal
```

The **echo** command changes the settings of only the logical output streams that are mentioned. You may use the option **all** to select all logical output streams or the option **none** to exclude all logical output streams. Individual streams are excluded by prefixing the corresponding option with **no**. For example:

```
echo none console terminal
```

disables all output to the terminal except console output, and

```
echo all nolog terminal
```

enables all output to the terminal except log output. If you want to find out about all the echo settings, type:

```
show echo
```

The command **show devices** lists all the currently assigned log files. There is nothing to stop you from having several log files opened at once.

When you have collected all the results required in a log file, use the command **deassign** to close the file. Failing that, Semper will close all log files at the end of a session.

Chapter 4

PICTURES

4.1 Overview

Pictures are obviously Semper's main concern. This chapter explains exactly what Semper means by the term *picture*, and what characteristics pictures may have, noting relevant processing commands where appropriate. A Semper picture has:

- dimensions
- a *class*
- a *form*
- coordinates and an origin
- a title
- a range of data
- a write-protection flag
- a date and time of origin

Figure 4-1 illustrates the elements that make up a typical Semper picture.

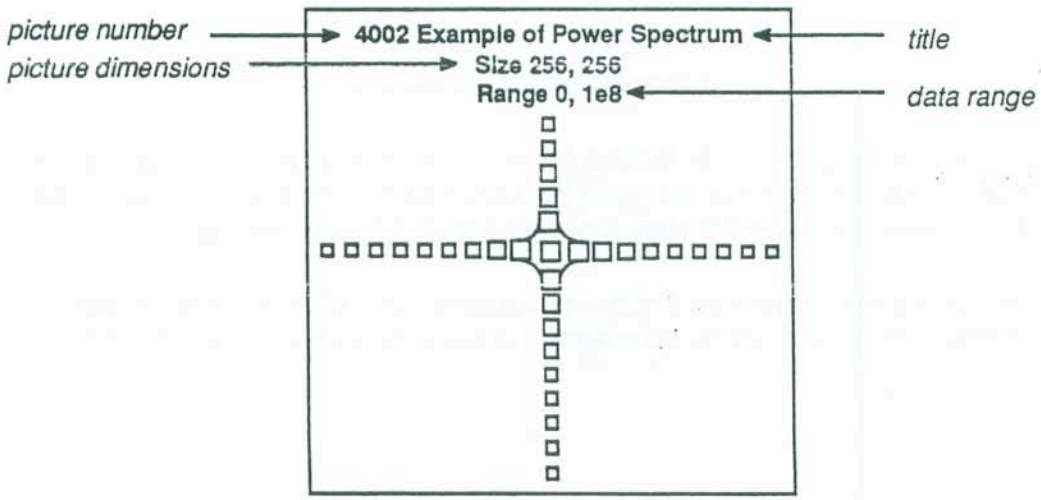

Figure 4-1. A Semper Picture

At the lowest level, Semper pictures are arrays of numbers, called picture elements or *pixels*. Pixels are stored in horizontal rows only, forming one-dimensional pictures (curves, functions, sections or line graphs). Equal length rows may be grouped to form layers with both rows and columns of pixels (2-D pictures), and equal size layers may be stacked together to form three-dimensional arrays (full colour images with three layers, multi-spectral images with any number of layers, or true 3-D objects).

Pixels commonly represent picture brightness values, evenly sampled in each direction. However, different sampling may be used for particular purposes, and other kinds of data altogether may be coerced into the form of a 'picture'. Some examples appear later on.

4.2 Dimensions

Pictures have three dimensions in general, referring respectively to the number of columns (or the row length), the number of rows (or the column length), and the number of layers. The dimensions are usually given as a set of three numbers in that order, for example size *ncol, nrow, nlay*. Each of the three dimensions may be different. A 2-D picture has dimensions *ncol, nrow, 1* and a 1-D picture has dimensions *ncol, 1, 1*. Figure 4-2 illustrates a one, two and three dimensional picture.

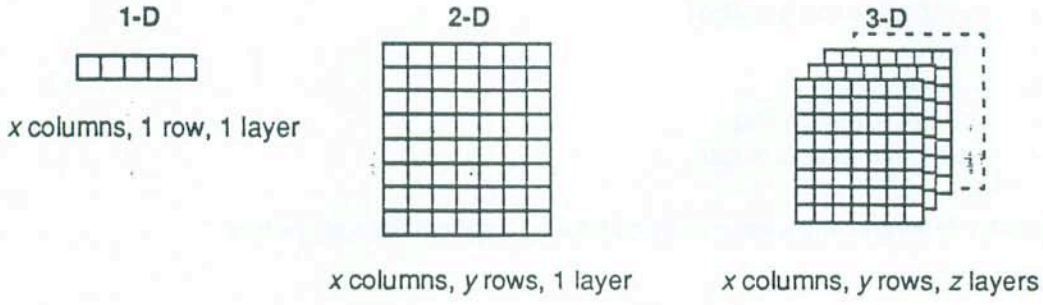

Figure 4-2. Picture Dimensions

The filing system supports pictures that are 2-D or 1-D in different directions (for example, size 1,1,256 or size 1,500,1). However, storage is inefficient whenever the first dimension is 1, and 1-D processing utilities (and some 2-D ones also) will not work with such pictures.

The command **examine** reports the dimensions of a given picture, and the command **pcb** sets the variables *ncols, nrows and nlayers* to the number of columns, rows and layers of a picture.

4.3 Classes

All pictures are assigned to one of a number of classes according to what their data represents and how it is to be used. There are 10 picture classes, numbered from 1 to 10 as follows:

No	Class	Description
1	*Image*	a sampled picture (most data fall into this class)
2	*Macro*	the text of a numbered macro (see *7.5.2, Numbered Macros)*
3	*Fourier*	the Fourier transform (*ft*) of an image
4	*Spectrum*	the intensity (modulus squared) of the *ft* of a picture
5	*Correlation*	the cross or auto-correlation function between pictures
6	*Undefined*	any data not otherwise classifiable
7	*Walsh*	the Walsh/Hadamard transform of a picture
8	*Plist*	a list of position coordinates with associated information
9	*Histogram*	the histogram (of the pixels) of a picture
10	*Lut*	a framestore look-up table

Semper commands may vary their action according to the class of a picture. For example, the command **display**, which displays pictures, will not display the text of a macro, and displays histograms in a different way from images; and the command **ps**, which generates a power spectrum, will not carry out the Fourier transformation before calculating the squared modulus, if the picture is already a *Fourier.*

The **examine** command reports the class of a particular picture, and the **pcb** command sets the variable *class* to the class number. You can change the class of a picture, if it is really necessary, by using the command **reclass.**

A *Macro* picture has only one row, containing the internal codes for the characters of the macro text.

A *Plist* picture contains a 1-D or 2-D array of positions in the form of x, (x,y) or (x,y,z) coordinates. The coordinate information is always stored in the first one, two or three layers of the *Plist.* Supplementary information about each position (such as heights or areas) can also be stored in subsequent layers of the picture. All the *Plist* pictures currently output by standard Semper commands contain (x,y) values but no z values. For example, the **peaks** command will locate local peaks in a picture and output the (x,y) positions of the peaks and their intensity values as a *Plist* with one row and three layers.

You can use the **xwires** command to generate *Plist* pictures interactively with the display cursor. The options **list** (the default) and **curve** determine whether you want to record an arbitrary series of

points or a connected series of points. The **curve** option may be further qualified by the **open** and **closed** options, to say whether the *Plist* describes an open or closed curve. The coordinate values are stored in exactly the same way for all three types of *Plist*. The three types of *Plist* are distinguished by a type code stored in the picture label. The command **examine full**, when applied to a *Plist* will list the *Plist* type. The **reclass** command accepts the same options as **xwires** to allow you to set or change the type code for a *Plist*. Most commands that use *Plist* pictures do not check the type code (the notable exception to this is the **mark** command which checks the type code, if the options **list** or **curve** and **open** or **closed** are used). An example of a *Plist* is given below. In this example, layer 1 of picture 4500 contains the *x* coordinates of a set of points and layer 2 contains the *y* coordinates. Computer output is shown in **bold**.

```
print 4500 layer 1

Layer 1 Z-coord 0

     0       1       2       3       4
0 |  0.000  -1.000  -1.000  -1.000   0.000

print 4500 layer 2

Values printed /10^1
Layer 2 Z-coord 1

     0       1       2       3       4
0 |  1.200   3.000   5.000   6.900   8.300
```

A *Histogram* picture has only one row, containing the various channel frequency counts followed by two final pixel values giving the minimum and maximum datum value spanned by the channels.

A *Lut* picture holds data suitable for use as a framestore look-up table. A *monochrome* lut consists of a single table of values for mapping pixel values to display intensities. A *false* or *full colour* lut has three tables, for mapping pixel values to the red, green and blue display channels in turn. Each table is held in a single picture row as follows:

- **monochrome**: one row only
- **false colour**: three rows; red, green, blue from top to bottom
- **full colour**: three one-row layers; red, green, blue from back to front

An *Undefined* picture may contain data of any kind – this class is provided to accommodate data used by locally written processing modules that do not fit any other existing class.

4.4 Forms

The data in a picture may be represented in four different ways, called *forms* in Semper. Each form offers a different tradeoff between storage and precision requirements. The basic storage unit is a byte (B), which consists of 8 binary digits. These have the following numbers, names and characteristics:

No	Form	Description
0	*Byte*	1byte storage. 0 to 255, unsigned integers only. A compact but restrictive form of storage as negative values can easily arise in processing.
1	*Integer*	2 byte storage. –32768 to 32767 or wider, integers only. *Integer* storage is less restrictive than *Byte* but still compact.
2	*Fp*	4 byte storage usually.1–35 to 1+35 or wider, either sign, integral or fractional (floating point). Typically one part in a million precision over the full range.
3	*Complex*	8 byte storage usually. Pairs of *fp* values representing real and imaginary parts of a complex value.

The **examine** command reports the form of a particular picture, and the **pcb** command sets the variable *form* to the form number. Generally speaking, all commands can be applied to all forms without distinction; in default, the output of most operations has the same form as the source, but a change can be forced by adding the new form name as an option to a command, for example:

```
copy 1 to 2 fp
```

The command **show system** reports the number of bytes actually occupied by each form in your installation.

Fourier pictures require complex pixels; *Spectrum, Correlation, Walsh* and *Histogram* normally require at least *fp*.

The result of forcing a picture into a form its pixels will not fit.(for example, converting a picture with negative values to *byte* form, or a *Fp* picture with values as large as 1e20 into *Integer* form) is not always clearly defined. Such conversions will usually result in hardware-generated errors which would cause Semper to abort. Where possible, such errors are trapped and reported like any other Semper generated error.

4.5 Coordinates and origins

The position of a given pixel in a Semper picture is identified by up to three coordinates, which form a normal right-handed *x,y,z* set: the *x* coordinate counts columns rightwards as a picture is viewed, the *y* counts rows upwards, and the *z* counts layers towards the viewer. We speak of 'the pixel with coordinates *x,y,z*' or simply 'the pixel (*x,y,z*)'.

The position of the coordinate origin is a characteristic of an individual picture. For most picture classes, its default position is the centre, rounded where necessary towards the front bottom right; however, the origin may be moved to any position within the picture using the **origin** command. For class *Histogram, Plist* and *Lut* pictures, the default is at the top left of the first (back) layer instead.

Figure 4-3 shows the typical positioning of a coordinate origin.

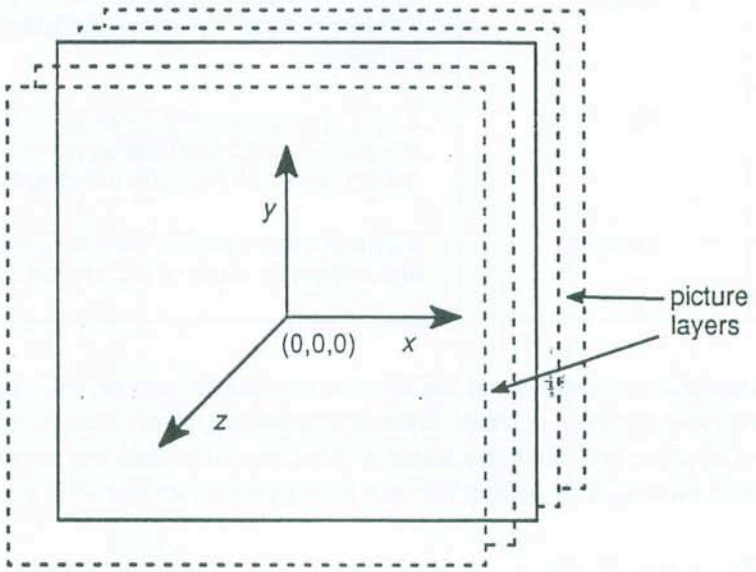

Figure 4-3. An Example of The Coordinate Origin of a Picture

Although almost all commands honour the recorded origin, a very small number, such as **fourier** transformation, insist on restrictions peculiar to the operation being performed.

The Fourier transform of a real image, although complex, is conjugate-symmetric, that is, pixels connected by reflection through the origin are complex conjugates of each other. This makes it unnecessary to store the entire transform plane, and Semper stores only the positive *x* half plane, with the picture origin at the origin of the left hand column in these circumstances. The transform of a complex image is stored in its entirety however, with a central origin. Similar remarks apply to *Spectrum* pictures, for which the data are real and reflected pixels connected by reflection are equal. The full and half-plane forms of such pictures can be interconverted by the commands **fullplane** and **halfplane**.

Layers can also be identified by layer numbers. Picture layers are numbered from the back layer (layer 1) to the front layer. Layer numbers are independent of any movement of the picture origin.

The **examine full** command can be used to check the position of the coordinate origin for any given picture, and the **pcb** command sets variables *x1,x2, y1,y2 and z1,z2* to the ranges spanned by the *x, y* and *z* coordinates of a given picture.

4.6 Direct pixel access

Two features of the command interpreter allow direct manipulation of individual pixels from the command level. Firstly, a function *p(x,y,z)* returns the value of the pixel value at *(x,y,z)* in a picture. Omitted coordinates default to zero, so that the forms *p(x,y)* and *p(x)* are suitable for 2-D and 1-D pictures. When a picture is complex, the functions *re(x,y,z)* and *im(x,y,z)* can be used to return the real and imaginary parts of a pixel (*re(x,y,z)* and *p(x,y,z)* are in fact identical in function).

Secondly, the command **p** or **pixel** resets the value of a pixel in a picture, for example:

```
p x,y,z = 56
```

sets the value of the pixel at *(x,y,z)* to 56. Here too, omitted coordinates default to zero. If several value expressions are given, they are placed in successive pixels rightwards along the row. If a picture is complex, values are taken in pairs for the real and imaginary parts (Semper does not support complex valued expressions).

The picture accessed by these facilities is the current picture, as defined by the variable *select* (see *5.2, Picture numbers in processing commands*). The flexibility that you have here, for example:

```
S$ pcb; for y y1,y2; for x x1,x2
(For 2) S$ p x,y=p(x,y)*1e3/(msq(x,y)+1e3); loop; loop
```

is partly offset by the fact that these facilities are not particularly fast (each use of the **p** command or call to the *p* function requires the entire picture row in question to be accessed). When you want to operate on an entire picture, the processing command **calculate** provides an efficient alternative.

4.7 Titles

A title of up to 150 characters is stored with each picture (with any trailing spaces truncated), providing a convenient way of recording the source, status or purpose of the picture. Most commands that output to a picture will automatically copy the picture title from the source picture.

If a picture has a title, the **examine** command will print it on the terminal and the **display** command will write it on the screen. When you create a new picture with the **create** command, it will have no title. You can add a title to a picture with the **title** command. The same command also allows you to add to or replace an existing title.

4.8 Data range

There are many occasions when a picture's minimum and maximum pixel values (or range) are required. A typical example is the **display** command which scales the picture range to fit the data range of the display. The picture range could be determined each time it is needed, but this is time-consuming, so Semper keeps a record of the range whenever it is calculated. Any subsequent change to the contents of a picture will cause the range to be deleted.

The **examine full** command reports the range if it is recorded, as does the **survey** command. You can use the **survey full** command to force Semper to re-calculate the range (in cases where the recorded range might be corrupt or out of date).

4.9 Write-protection

A write-protect (*wp*) flag may be set for any individual picture. Semper will not delete a write-protected picture or alter any of its data or recorded characteristics. The **examine** command reports when a picture is write-protected, and the **wp** command turns write-protection on or off.

4.10 Date and time of creation

Where the installation is capable of providing the information, the date and time at which each picture was created is recorded. The date and time is also updated when a picture is replaced by a processed version. The **examine full** command reports the creation date of a given picture.

Chapter 5

DEVICES AND STORAGE

5.1 Overview

Semper's picture storage is organised as a number of separate *devices*, each of which may be a disc file or magnetic tape and one of which may be a framestore display device. Each device typically holds many pictures. You establish the devices that are available dynamically at any given time by means of the **assign** command. Main memory is not used explicitly as a storage medium, though installations where large amounts of memory are available usually buffer portions of disc storage in main memory.

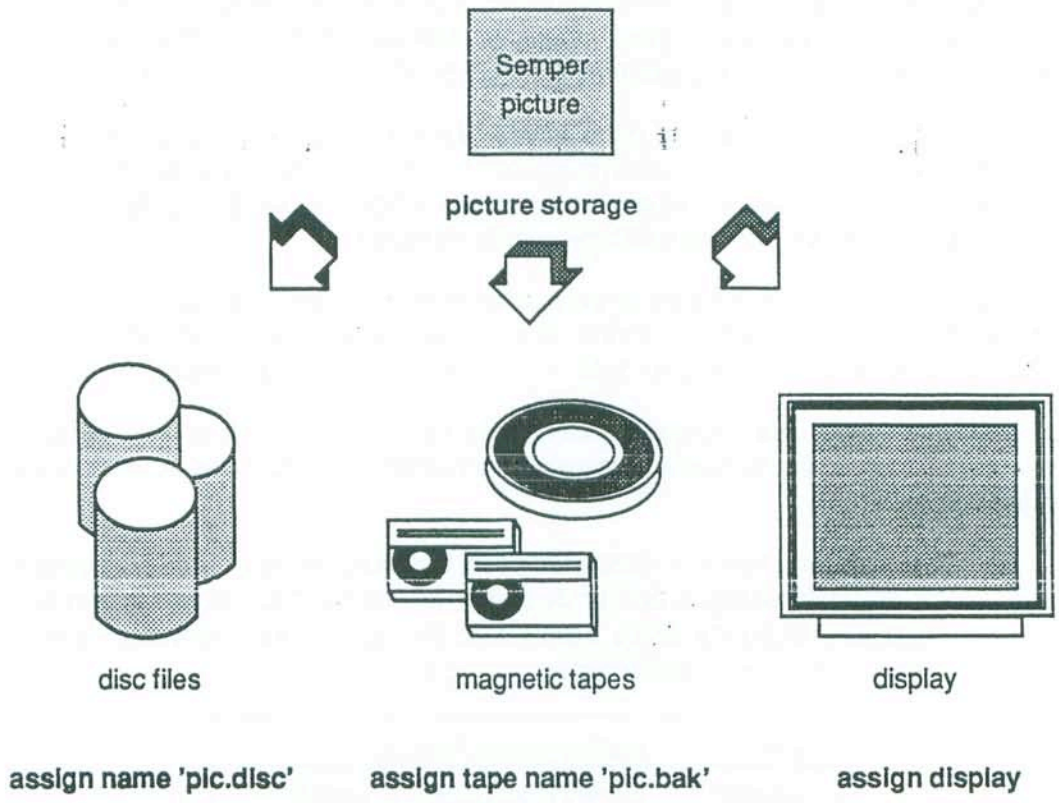

Figure 5-1. Semper Devices

Devices are numbered from 1 up to an installation-dependent maximum (the command **show system** will tell you what this is), and pictures within a given device are numbered from 1 to 999. To identify a picture exactly, you need to specify a device number as well as the picture number. These can be combined in one of two ways. Firstly, you can place the picture number after the device number with a colon to separate them, for example:

```
1:45 3:999 cd:4 dev:pic
```

In fact, the colon is a special arithmetic operator (see *2.9, Expressions*) which combines the device and picture number into a single numerical value. The other way, which corresponds to the way Semper handles and reports picture numbers, is the form *1000*dev+pic*, for example, 2035 is picture 35 on device 2.

> More precisely, *d:p* means [*d*, or its device part if *d>1000*] times 1000 plus [*p*, or the picture part of *p* if *p>1000*]. If *d* is omitted, the device number defaults to the value of the variable *cd*, that is, *:p* is evaluated as *cd:p*.
>
> For mnemonic purposes, you can give pictures at least temporary names by assigning picture numbers to appropriate variables.

Device 1 is at present reserved for the display device. Only one such device is allowed at any one time. Devices 2, 3 and upwards may be assigned to disc files or tapes as desired. Help libraries (see *3.14, Help*), log files (see *3.16, Controlling output in Semper*) and program libraries (see *7.2, Programs*) are also assigned as devices. The command **show devices** types details of the devices currently assigned. The command **examine device** *n* types details of all pictures on device *n*.

The variable *select* is set by Semper to the number of the most recently used picture (the output picture, when a command uses more than one picture). It provides a current picture number, and is the default operand for many commands. You cannot reset its value directly by means of an assignment. However, you can set it with the command **select**.

The variable *display* is set by Semper to the number of the most recently used display picture. It provides a current display number, simpler to use than the full picture number, for example, **sharpen display** rather than **sharpen 1:23**. You cannot reset this value directly.

The variable *cd* is treated as a current device number. Its value is prefixed as a device number to any picture number quoted without an explicit device number. You may reset its value freely by a simple assignment.

> The variable *fs* holds the device number of the display device, and may be used freely, for example, **sharpen fs:3** or **deassign device fs**. This release of Semper only supports one display device, however, so that no purpose is served by resetting *fs*, though users are not prevented from doing so.

select	current picture number
display	current display picture number
cd	current device number
fs	current display device number

5.2 Picture numbers in processing commands

Although a processing command could have any number of source pictures and any number of output (resulting) pictures, most commands in fact have no more than one source picture and one output picture. In these circumstances, the key **from** is almost always used to designate the source picture number, and the key **to** to designate the output. For example:

```
mask from 6 to 7 radius 35
```

takes the image held in picture 6 (which means *cd:6*, as noted in the previous section), masks it and stores the output as a new picture 7 (that is, *cd:7*). Because these two keys are so widely used, the interpreter usually assumes the names when they are omitted, using the mechanism described earlier (see *3.5, Processing Commands*). The key **from** is assumed before **to**. As a result:

```
mask 6 7 radius 35
```

```
mask radius 35 6 to 7
```

and even:

```
mask from 6 7 radius 35
```

are all equivalent to the first example above.

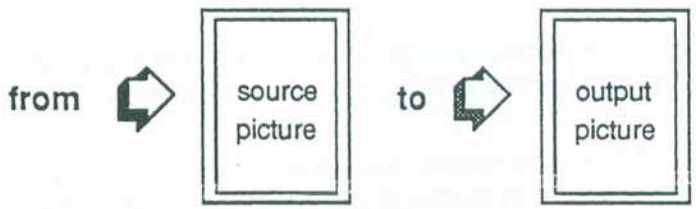

*Figure 5-2. The keys **from** and **to***

Commands are also consistent as to the defaults assumed for these keys if either is not given a value at all. The default source is the current picture number, *select*, and the default output replaces (overwrites) the source, giving apparent *in situ* processing. Formally, the default for **from** is *select*, and the default for **to** is *from*. Thus:

```
mask
```

replaces the current picture (which might or might not be picture 6) by a masked version of itself:

```
mask 6
```

replaces picture 6 (whether or not it is the current picture) by a masked version of itself, and:

```
mask to 7
```

creates a new current picture 7 containing a masked version of the original current picture. Commands never alter the source picture when a separate output picture is indicated.

If you need to override the default selection, for example, to use the **p** command, you can use the **select** command. For example:

```
select 4:53
```

makes picture 4053 the default source to the next command.

Although the **select** command always does exactly what you tell it, Semper does not automatically set *select* to point to display pictures (because of the lower precision often associated with these), unless you set *cd=fs* explicitly, thereby making the display your current device.

5.3 Disc files

The differences between the three types of storage medium used (disc, tape or display) are heavily disguised by Semper, so that pictures appear to have exactly the same characteristics whatever device they are stored on, and processing is as far as possible device-independent. However a few features still need comment.

Disc files provide the most flexible form of storage. Pictures can be stored anywhere within them, and Semper manages the allocation and reclaiming of space as necessary.

Producing a new picture with the same number as an existing picture causes Semper to implicitly delete the latter, that is, to overwrite it. *In situ* processing, where the output has the same number as the source and appears to overwrite it, is possible both when the two pictures are compatible for storage purposes (when they have the same dimensions and form), in which case they do indeed share the same storage, and when they are not, in which case separate storage is provided for the duration of the processing command.

Disc files may have any size, and are organised into blocks, which need not match the physical block size of the actual disc drives being used, and which can be read or written in any order. The first few blocks in a disc file hold a directory of up to 2000 or so slots each recording a group of blocks occupied by a picture or a group of free blocks.

In the interests of speed, most installations buffer substantial parts of disc devices in main memory areas called cache buffers, which are read and written as a whole, and alterations to these are not always copied to the physical disc immediately. You can use the command **flush** (with no arguments) to force an immediate copy to disc, should there be any fear of loss of data due to a subsequent session abort.

Semper has a number of commands for managing disc files:

- **assign** makes a disc file available to a Semper session
- **assign new** creates a new disc file
- **assign scratch** creates a temporary disc file
- **reinitialise** clears the contents of a disc file's directory (discarding all existing data)
- **deassign** releases a disc file
- **compress** regroups free blocks together (in case a disc file becomes too fragmented)
- **directory** types information on the state of the directory for a given device
- **renumber** renumbers individual pictures
- **delete** deletes pictures

5.4 Magnetic tapes

Magnetic tapes are intrinsically serial in nature: writing data at a given position renders data further down the tape inaccessible. Picture numbers correspond directly to the relative position of the picture on the tape; picture 3 is the third recorded picture, etc. You may use tape pictures quite freely as the source to commands (though if access is very non-serial in nature, it may be faster to copy them to disc first), but only the **copy** command, which takes special action to ensure the integrity of the tape in the event of errors of any kind, can output pictures to tape.

Tapes contain no directory, but individual pictures have labels exactly like disc pictures. The **examine** command is used to inspect them in the same way. Use the **assign tape** command to mount a tape (or **assign tape new** for a tape to be treated as blank). The command **deassign** releases and dismounts them. The **rewind** command, in some installations at least, initiates a rewind without waiting for it to complete.

> Tapes may always be transported between installations sharing a common processor and operating system. Between different systems it will usually be possible to exchange byte pictures on tape, as this form (used for picture labels as well as for byte pixels) is universally defined, and it may be possible to exchange integer pictures. Fp and complex data are, however, usually represented in mutually incompatible ways by different manufacturers.

5.5 Displays

The display device plays a dual role, as a viewing device and as a storage medium, and its organisation is correspondingly more complex than that of the other devices. Ideally, it is of the framestore type, with a multiple grey level storage capability in each of several frames. Display frames are numbered from 1 up to the number supported by the current display device (you can use the command **show devices** to find out what the display device can do). Quite different hardware may be used, with appropriate adjustments in functionality.

For most purposes, the display storage space is subdivided by the user into partitions, numbered from 1 up to an installation-dependent maximum (the command **show system** will tell you what this is), each of which can be used independently. Partitions need not be square, they may overlap each

other and they may have any size and position that keeps them within a display frame. There are no partitions defined at the start of a session. Display partitions are defined with the **partition** command, perhaps in the startup file (see *7.4, The startup file*). While defined, each partition may be used to hold one picture, identified by the same number as the partition, that is, display picture *n* is stored in partition *n*. The picture is stored centrally within the partition and it may not fill all of the partition. Redefining or deleting a partition causes any picture stored in it to be deleted. Otherwise, you can use the **partition** command to delete or redefine partitions freely. The command **show partitions** reports all the partitions currently defined.

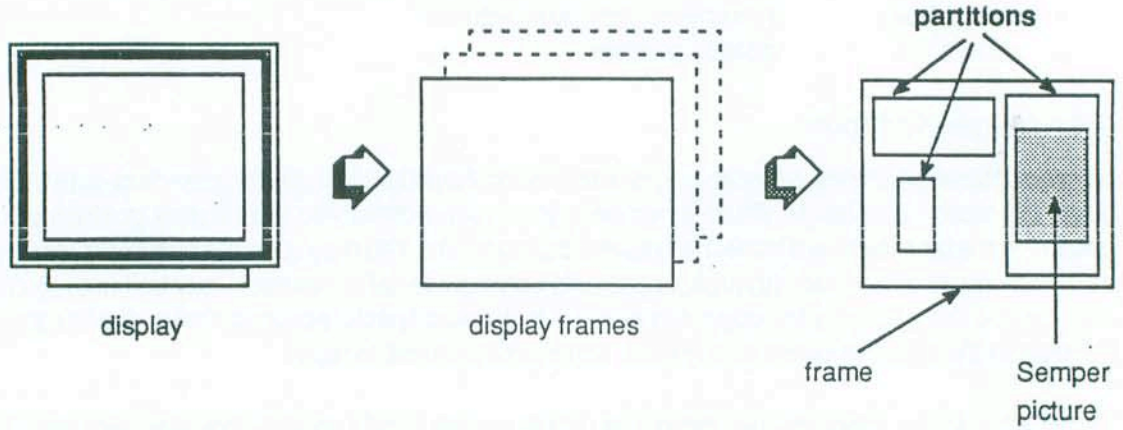

Figure 5-3: Display Partitions

Frame and partition numbers can be prefixed with a device number and colon, just as with picture numbers, for example, 1:5 and *fs*:2.

Storage is primarily two-dimensional. However, if the display device is organised as several distinct frames, successive layers of a multi-layer display are stored in successive display frames, allowing full colour presentation. 1-D display pictures are simply stored as 2-D pictures with only one row.

Some processing commands, such as **display** and **ymod**, generate a different type of display constructed with vector graphics. Such displays cannot be 'read' or re-processed. In this version of Semper there are three types of pictures displayed in this way:

- Line graph presentation of 1-D picture
- Graphical presentation of histogram
- Isometric grid plot (**ymod**) presentation of 2-D picture

Two further characteristics are peculiar to display pictures. The first is an undersampling factor. Where pictures do not otherwise fit their partitions, the data are simply undersampled as necessary, with equal sampling in both directions, and starting from the top left. Obviously, undersampled pictures cannot be 'read' or reprocessed.

Secondly, although a form is recorded for each display picture, most display devices only allow the data to be held in a limited precision form (typically one byte per pixel). To accommodate this, pixels are in general scaled on output to the display and the scaling is reversed when they are recovered, so that the device can appear to hold data with an otherwise impossible range such as 0.1 to 0.15 or 1e6 to 1e9. Precision is thus reduced when *fp* pictures are stored in and later recovered from the display, but there are no range problems. The values scaled to black and white are recorded as additional characteristics of display pictures. They are determined by the values of the variables *min* and *max* at the time a picture is displayed.

> The precision loss is normally too great for *Fourier* or *Spectrum* pictures (which have a very wide dynamic range) to be stored in display pictures, though there is no problem about viewing them.

Setting *min* and *max* before directing a picture to the display is the basic mechanism provided for controlling the grey scaling of a display picture. Moving them apart reduces contrast, exchanging them reverses contrast, and so on. The mechanism is not however always obvious to the new user, since the command **display**, the simplest means of displaying an existing picture, itself sets *min* and *max* to the actual range of the data being displayed (unless option **preset** is used). Look-up tables (see *5.5.3, Look-up tables*) may also be used to alter how the data actually stored in a framestore is presented on the monitor.

> *Complex* display pictures are accommodated by storing their real and imaginary parts side by side.
>
> Since a single picture, and therefore a single set of picture characteristics is associated with a given partition, *in situ* processing of a display picture, for example, **sharpen display**, is possible only where the output has characteristics compatible with those of the source (the same dimensions, both forms complex or both non-complex). The same, that is, original, data range is used for scaling both pictures.

The display is made available to a Semper session by the **assign...display** command (which may, in a given installation, have keys and options selecting any features of the hardware that are not invariable). The **examine** command can be used with display pictures just as with disc pictures, except for pictures displayed in vector graphic form. The **erase** command clears a frame or partition, and the **ramps** command generates full grey scale ramps for viewing monitor adjustment.

5.5.1 Overlays and annotation

Every display frame has, hardware permitting, a separate one bit overlay, which can be written to and erased without affecting the picture data. Semper uses the overlay for displaying text and vector graphics and for annotating display pictures. The overlay is displayed on top of the image stored in the display frame. You use the **erase overlay** command to clear an overlay. There are no facilities for reading back information from the overlay or for writing picture data to it.

Figure 5-4 illustrates the overlays and annotation provided by Semper.

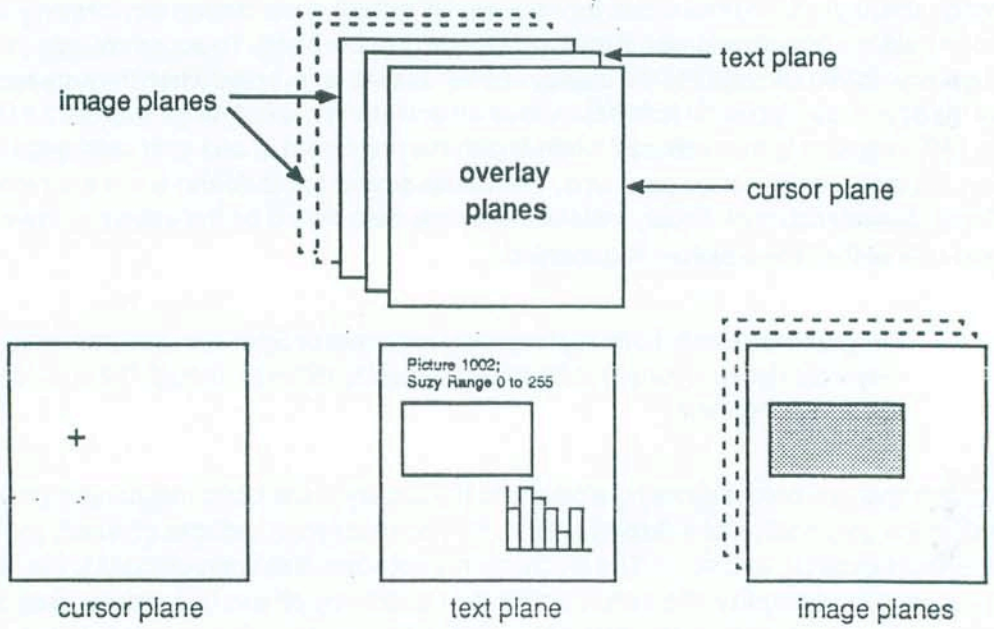

Figure 5-4. Overlays and Annotation

There are a number of commands, like **mark**, **ymod** and **contour**, that explicitly generate graphical output. Some commands also generate optional display annotation that allows you to verify the details of the operation of the command. Such annotation is turned on by setting the **mark** key either to a display picture number or to the value *yes* (to annotate the current display picture). Setting **mark** to *no* or leaving it unset suppresses the output of such annotation. For example:

```
display to dis:3

cut size 250 top left mark dis:3
```

cuts out a subregion of the current picture and marks the subregion border on display picture *dis:3*. Of course, annotating a picture in this way makes no sense unless it is related to the picture being processed. Details about annotation produced by Semper commands are given in the individual descriptions of these commands. See *6.4, Other widely used keys and options* for more information about the **mark** key.

> Where no independent overlay is available, overlaid material overwrites picture data. Suitable settings of keys and options **mark**, **letter**, **border** and **verify** (see *6.4, Other widely used keys and options*) however make it possible to suppress all such material when necessary.

5.5.2 Graphical coordinate systems

Semper uses three types of coordinate systems to identify pixels:

- **picture coordinates**
- **partition coordinates**
- **frame coordinates**

For graphical purposes, positions are most commonly specified in terms of the standard coordinate system used to identify pixels (see *4.5, Coordinates and origins*) – called *picture coordinates*. The two alternative systems, *partition* and *frame* coordinates are 2-D (*x,y*) systems, with the same sense as picture coordinates (that is, the first coordinate increasing to the right and the second upwards), but they measure distances directly in display hardware pixels from an origin at the centre of the partition or frame as appropriate (rounded towards the bottom right if the dimensions are even). They are useful for positioning text outside pictures, for referring to positions when no display picture exists (for example, for initial definition of partitions), for setting viewing conditions that include more than one partition, and so on. Figure 5-5 illustrates the three different types of coordinate systems.

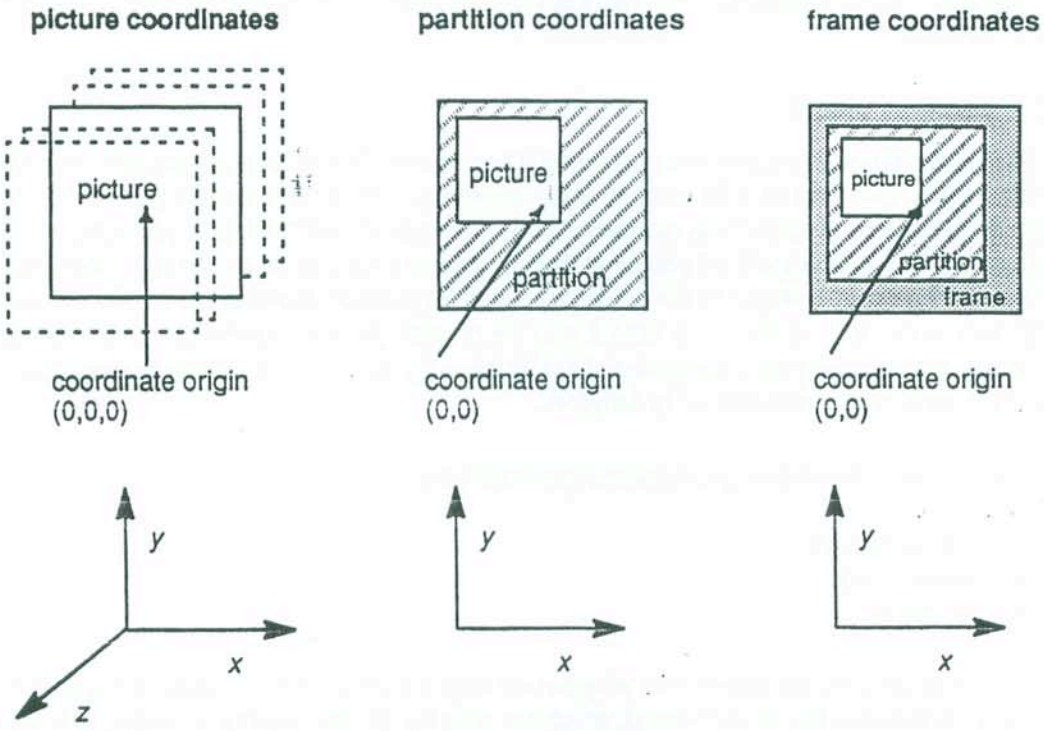

Figure 5-5. Graphical Coordinate Systems

Where relevant, one of the three graphics coordinate modes is selected by one of the options **picture** (the default), **partition** or **frame**, with an unkeyed value indicating the required display picture, partition or frame number.

For example:

```
erase fs:1

erase picture fs:1
```

both clear the area occupied by picture *fs:1*

```
view partition fs:2
```

sets the viewing conditions to make partition *fs:2* visible with the partition centred on the monitor, and:

```
ramps frame fs:2 size 200,40 top
```

fills a rectangular subregion at the top of display frame *fs:2* with a grey scale ramp.

If a picture or partition number is not given, it defaults to the value of the variable *display*. If a frame number is not given, it defaults to the value of variable *cframe* (maintained by Semper as a current frame number).

5.5.3 Look-up tables

In most framestores, look-up tables (*luts*) control how stored pixel values are actually presented on the monitor screen, that is, with what intensity or colour. Semper provides in general for several distinct luts to be defined at once, numbered from 1 up to an installation-dependent maximum (the command **show system** will tell you what this is). Only one lut can be active at a given time. The variable *clut* is set by Semper to the active or current lut number. Semper stores all lut data in the temporary work disc. At the same time it may be possible to store some luts within the display hardware, allowing very rapid switching of luts. If there is just a single lut in the hardware, it will be reloaded every time a different lut is activated.

Luts have one of the following associated characteristics:

- **monochrome**
- **false colour**
- **full colour**

The active lut accordingly determines whether viewing is in monochrome, false or full colour form. Luts are managed by the **lut** command, which provides for generating luts, adjusting existing ones (perhaps interactively via terminal keys), loading luts from lut pictures, and storing them permanently as lut pictures. The command **show luts** reports all luts currently defined. No luts are defined at the start of a session, and it is necessary to establish at least one before any pictures can be displayed, perhaps by **lut** commands in the startup file (see *7.4, The startup file*).

There are two further commands that allow you to create or modify the contents of look-up tables: **lset** and **ladjust**. They both accept a wide range of parameters, such as brightness, hue and

saturation. The values you specify are applied on a linear scale over a given range of output levels. The two commands differ only in that **lset** takes its parameters from the command line, while **ladjust** allows you to vary selected parameters according to the movements of the mouse or other pointing device. With the **hue** and **saturation** parameters, you can create all kinds of colour scales. You can highlight a particular range of output levels by modifying those entries in the look-up table.

A *monochrome* lut consists of a single array of intensity transformation values, tabulating the intensity with which each possible stored pixel value is to be displayed. When a monochrome lut is active, a single display frame is fed through the table to each of the three monitor colour channels.

A *false colour* lut consists of three separate tables, for red, green and blue respectively. When one is active, a single display frame is fed to each of the monitor colour channels through the appropriate table.

A *full colour* lut similarly consists of three separate tables for red green and blue. When one is active, three successively numbered display frames are fed to the monitor colour channels through the appropriate table.

Storing luts in the work disc means that you can always access the lut data, even when it is not possible to read the lut data stored in the hardware.

Luts are selected explicitly by the **view** command described in the next section. They can also be selected when viewing a display partition or picture because each display partition has a default lut number established by the **partition** command. The command **show partitions** lists the default lut number for every partition that exists.

5.5.4 Viewing conditions

The **view** command provides comprehensive control of the monitor viewing conditions which are described in terms of the following parameters:

- Display frame number
- Look-up table number
- Zoom factor
- Pan centre position

All of these can be specified directly with the **view** command, for example:

```
view frame fs:1 lut 2 zoom 4 pan 20,50
```

causes frame *fs:1* to be displayed on the monitor screen with lut 2 and scaled up by a factor of 4. The pan centre position specifies that the frame should be displayed so that the frame position (20,50) appears at the centre of the monitor screen. Standard viewing conditions (current frame, current lut, unit zoom factor and pan centre position (0,0)) are re-established with the command:

```
view frame
```

The current frame and lut numbers (as defined by the last **view** command) are stored in the variables *cframe* and *clut* (see *6.2, Protected and fixed variables*).

You can also use the **view** command to view display partitions and pictures by substituting a **partition** or **picture** option for the **frame** option. If you do not specify an option, the **picture** option is assumed. For example:

```
view partition fs:3
view picture fs:5
view fs:5
```

The default for the partition or picture number is taken from the variable *display*, so the **view** command by itself is equivalent to:

```
view picture display
```

The **view** command can also view a number of frames, partitions or pictures in turn, for example:

```
view frame 1,2,3
view display:4,display:5 zoom 2
```

This is a very useful way for comparing pictures. There is a default lut number associated with each partition when it is created (the command **show partitions** will list all partition information). If no lut number is specified when viewing a partition or picture, it defaults to that specified for the corresponding partition. This means that the view switching facility just mentioned can switch to a different lut for each picture or partition.

The pan centre position specifies a position relative to the origin of the frame, partition or picture, as appropriate. This point will be displayed (hardware permitting) at the centre of the monitor. You use the **pan** key to specify this position explicitly in the **view** command. If the **pan** key is not used, you may use any of the standard 2-D subregion keys and options (see *6.5, Specifying picture subregions*) instead. The pan centre position will be set to the centre of the given subregion, that is, the view is centred on the subregion. Here are a few examples:

```
view frame top left
view partition position x,y
xwires region; view @region zoom 2
```

The last example uses the **@region** macro (see *7.5.1, Named macros*) to pick up the subregion input on the display by means of the **xwires region** command.

> How much of the general capability described is implemented may vary considerably with the local display hardware. With the simplest of framestores, the **view** command may have no effect at all.
>
> It is important to appreciate that the changes apparently caused by **view** and **lut** are indeed only apparent, and do not affect the stored data themselves, which can be read or altered in the usual way whether visible or not.

In installations using a device such as a laser printer as the display device, Semper retains all the display output internally, so that nothing can be seen, until the command **close** is used. This makes it possible to use several partitions, and superpose text and graphics on them etc., before the displayed material is finally output for viewing and can no longer be accessed from Semper.

The **view** command allows you to set the viewing conditions directly. You may also cause the viewing conditions to change in any command which accesses or creates a display picture. To view the display picture in question, you include the **view** option, for example:

```
transpose display view
display 1 to fs:3 view
```

If **zoom** and/or **lut** have been assigned global defaults, these are honoured when the **view** option is used, for example:

```
zoom=2; lut=5
sharpen to display view
```

The commands **xwires**, **mark**, **ramps** and **erase** also recognise the **view** option. These commands provide general display input and output facilities, in that they will work with respect to frames and partitions as well as pictures. Here, the **view** option will cause the part of the display in question to be viewed, for example:

```
ramps frame 2 size 150 bottom right view
```

The **xwires** command assumes that the **view** option is set, and allows you override this by prefixing the option with **no**. The default for the **view** option is different for **xwires** to ensure that the cursor will normally be visible when it is displayed.

5.6 Write-protect and write-only flags

When using the **assign** command to make a device available to Semper, you may use the option **wp** to set a write-protect flag for the entire device. This prevents any subsequent alteration to the contents of the device, providing at the device level the same sort of protection that is available for individual pictures within a device. Quoting **wp** may also allow at least read access to devices belonging to other users. The **wp** option is ignored when assigning a display device.

The display device in some installations is marked write-only if it is not of the storage type, for example, a laser printer.

5.7 Getting images into Semper

Before you can begin image processing, you must first make images available to Semper. There are a number of standard routes provided for doing this. Your version of Semper may also provide extra commands to provide access to images, for example, camera input via the display device, access to images stored by other image processing systems, etc.

The **save** command provides the simplest and most efficient way to get images into Semper. Its main purpose is for transferring images between Semper users and for saving images for long term storage. The command simply writes one or more pictures to a new, dynamically created disc file.

For example:

```
save 2:3,2:5 name 'results'
```

is equivalent to the series of commands:

```
assign new name 'results' size ...
copy 2:3 n:1; copy 2:4 n:2; copy 2:5 n:3
deassign device n
```

where the size of the disc file is just large enough for the pictures that are to be copied into it and *n* is the device number assigned to the new disc file. The pictures can be recovered subsequently like any other pictures stored in a Semper disc file, that is:

```
assign name 'results'
examine device n
```

> The size of the file created by the **save** command is rounded up to the nearest multiple of the unit in terms of which all disc transfers are actually made (use the **show system** command to find out what this is). This may result in much wasted space if very small pictures are stored individually.

A more flexible but slower route is provided by the **read** and **write** commands. These make use of standard Fortran READ and WRITE statements. The files can be unformatted if the option **unformatted** is specified. The **write** command allows you to specify a format for writing out the picture data to a formatted file. The format used is recorded in the first line of the file so that the **read** command can read the file without any further ado. For example:

```
write 2:71 name 'pic71' format '(10F8.2)'
```

would write the picture data to file *pic71*. The data can then be retrieved simply with the **read** command, for example:

```
read 3:4 name 'pic71'
```

Formatted files are very portable: it should be possible to read them on any system. They also provide a straightforward route for transferring picture data from user programs into Semper. Unformatted files are much faster, but it usually will not be possible to read them on another system.

The **read** and **write** commands are sometimes specially extended to provide access to local picture file formats. These extra facilities would appear as extra keys and options for **read** and **write**, that is, try the commands **syntax read** and **syntax write**. You can find a precise description of the file formats in the document:

Semper 6 READ/WRITE file format

which can be obtained from *Synoptics*.

The display device, if it is of the framestore type, could provide you with a number of ways of getting images into Semper. Firstly, you can gain access to any image stored on the display if you first create a display picture which corresponds exactly with the image in size and position. Once the display picture is created, it can be accessed like any other display picture. Secondly, the framestore may have a camera or other input facility, in which case you may find some extra commands are provided to control this facility. Here, the end result is normally a display picture which you can then process in the usual way. The command **help framestore** should tell you about any extra display commands.

The table below gives a summary of the commands described in this section.

save	writes a picture to a newly created disc file
read	reads a picture from a non-Semper program or a picture created using the **write** command
write	writes a picture to file to be transferred to a non-Semper program – you can specify the file format
copy	copies a picture to another device

Chapter 6

IMPORTANT VARIABLES

6.1 Overview

This chapter concentrates on the variables that Semper uses to process commands. It also details keys and options that are common to many commands and some that are common to all.

6.2 Protected and fixed variables

A small number of Semper variables have special meanings and properties. The first set of these are said to be *protected*, and cannot be changed by assignments:

ric	release identification code (see *7.3, Run files*)
yes	= 1; used as logical value true and for answering questions
no	= 0; used as logical value false and for answering questions
pi	= π = 3.1415927
select	the current picture number (see *5.2, Picture numbers in processing commands*)
display	the current display number (see *Chapter 5, Devices and Storage*)
cframe	current frame number (see *5.5.4, Viewing conditions*)
clut	current lut number (see *5.5.4, Viewing conditions*)
rc	error code after command is executed (see *7.6, Error trapping*)

A second set are said to be *fixed*. These may be changed by assignments, but not unset:

min,max,mean,me2,sd	last determined pixel data range, mean, standard deviation
trap	used in user controlled error handling (see *7.6, Error Trapping*)
cd	current picture device number (see *Chapter 5, Devices and Storage*)
fs	current display device number (see *Chapter 5, Devices and Storage*)

To provide for complex data, a complex mean (*mean,me2*) is calculated in general, and *sd* is the rms modulus deviation from this.

6.3 General keys and options

There are keys and options used sufficiently widely and consistently by processing commands for them to be recognised directly by the command interpreter. These are called *general* keys and options. They do not appear in the list of keys and options listed by the **syntax** command but are nevertheless accepted (though quite often ignored) by all processing commands:

byte,integer,fp,complex These options force the form of a picture that is output by a command. Except where particular output forms are essential, commands produce output with the same form as the source unless overriden by one of these options.

erase This option causes the display partitions to be erased before pictures are displayed in them. The default is **no**.

mark The **mark** key causes some commands to annotate the display picture indicated (see *5.5.1, Overlays and annotation*), for example, **mask 42 mark dis:2**. The value **yes** causes the current display picture to be annotated. The default is **no**.

view This option causes the viewing conditions to be set so as to view the display region in question. The default is **yes** for **xwires** and **no** for other commands.

re,im These options limit the annotation on a complex display picture to the real or imaginary part. The default is to annotate the real part and to repeat the annotation on the imaginary part.

mkmode The **mkmode** key determines the way in which positions are annotated on display pictures with values as follows:

1. upright cross (default)
2. diagonal cross
3. upright box
4. diagonal box
5. single pixel

The default is **mkmode 1**.

mkmode 1 **mkmode 2** **mkmode 3** **mkmode 4** **mkmode 5**

mksize This key determines the size (radius) of annotation made according to **mkmode**. The total width of the mark is *2*mksize+1* (**mksize** is ignored if **mkmode=5**)

6.4 Other widely used keys and options

In addition to the general keys and options, there are a number of widely used options that are defined for some commands. They control facilities that you may want to enable or disable for a time by setting the corresponding variable to *yes* or *no*:

preset — This option causes the current value of the variables *min* and *max* to be used, instead of the actual pixel range, for scaling purposes. The default is **no**.

letter — This option causes commands such as **display** to add lettering to displays giving size, title and grey scale information. The default is **yes**.

border — This option causes commands such as **display** to mark the border outlining display pictures. The default is **yes**.

verify — This option causes commands to 'verify' their operation or results to the console output stream (or the display if **xwires** or **pdraw**). The default may be **yes** or **no**, depending on whether or not a command provides verification output by default.

The variables *zoom*, *lut* and *mtpass* are consulted by Semper system routines that may be called by many Semper commands. When the **view** option is used (see *5.5.4, Viewing conditions*), the variables *zoom* and *lut*, if set, determine the zoom factor and lut number for viewing. If *zoom* is unset, a value of 1 is assumed and if *lut* is unset, the value of the variable *clut* is used. If *mtpass* is set to *yes*, hardware errors detected during the reading of magnetic tape blocks are ignored, allowing Semper to continue with no more than an isolated picture row being affected. This kind of failure is unfortunately all too common, especially on interchanging tapes between computers. If *mtpass* is unset, *no* is assumed.

6.5 Specifying picture subregions

Subregions of a given picture can be specified by means of a relatively flexible set of keys and options, in which each of the three directions figures largely independently.

The subregion dimensions are indicated via the key **size**, which has three components. If none are quoted, they default to the source picture dimensions, but if *size* alone is quoted, *si2* defaults to *size* and *si3* defaults to the source picture *z* dimension. Thus:

```
size 300,200
```

describes a subregion 300 pixels across by 200 down, and

```
size 250
```

describes one 250 pixels square.

By default, the subregion is positioned so that its centre (rounded towards the near bottom right if necessary) coincides with the source picture origin. However options **left** or **right** cause it to be placed with the left or right sides coinciding with those of the source instead, and similar options **bottom** or **top** and **far** or **near** have a similar effect on the other two directions. Thus:

```
size 100 bottom left
```

refers to the bottom left 100 pixels square. A three-component key position can be used to impose an additional displacement on the default position otherwise established, for example:

```
size 100 position x,y
```

describes a 100 pixel square region centred on the point *x* to the right and *y* above the source picture origin. Figure 6-1 illustrates subregion dimensions and positioning. Note that a standard Semper picture size is 768 by 512.

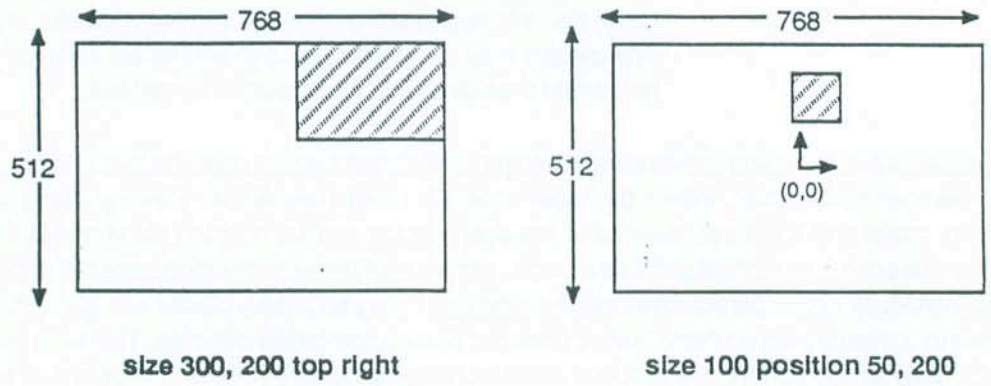

Figure 6-1. Subregion Examples

You can specify the range of layers to be processed (where the processing module has multi-layer capabilities) in two ways. Firstly, you may use the previously mentioned keys and options. For example:

```
size 300,200,1 near
```

indicates the front layer of a picture with five layers 300 by 200 in size, with the origin in the central (third) layer.

You indicate the central three layers by:

```
size 300,200,3 position 0,0,0
```

or simply:

```
size 300,200,3
```

Secondly, you can omit the last component of the **size** key and then specify a range of layers with the **layer** key. If you omit the second component of the **layer** key, it defaults to the value for the first component. For the first example this gives three possibilities:

```
size 300,200 layer 5,5
size 300,200 layer 5
layer 5
```

and for the second example, two possibilities:

```
size 300,200 layer 2,4
layer 2,4
```

Any attempt to use the **layer** key in combination with any of the subregion keys and options that relate to the *z* direction (keys *si3* and *po3* and options **near** and **far**) will be faulted.

> For the sake of brevity, Semper documentation on processing commands often does not list all the keys and options described here, but simply refers to the *standard 2-D subregion* or *standard multi-layer subregion keys and options*.

Chapter 7

INDIRECT INTERPRETATION

7.1 Overview

This chapter describes the forms in which Semper command text can be held for later indirect or deferred interpretation, as programs, run files and macros (named and numbered), and explains a few remaining features of the system relevant to their use.

7.2 Programs

The principal way of storing Semper commands for indirect execution is as a *program.* A program consists simply of a sequence of commands, such as you might type interactively at the terminal, between special header and terminator commands. Two simple examples follow:

- a program *ctop* converting cartesian coordinates *x, y* to radial polars *r,theta*
- a program *smm* determining the minimum and maximum values found in a series of pictures *n1* to *n2.*

```
ctop(); r=mod(x,y); theta=phase(x,y); end

smm()
! Sets min,max to data range for pictures n1 - n2
local n,b,t
for n=n1,n2; survey n; if n=n1 b=min t=max
   b=min(b,min) t=max(t,max); loop
min=b max=t; return
end
```

The header command gives the program a name. It consists of a program name (see *2.4, Names*) followed by an empty pair of round brackets. The terminator command is the simple word **end.** Execution begins at the start of the program, and ends when Semper encounters a **return** or **end** command. Within a program, all the language features described earlier may be used freely, with the following restrictions and extensions:

- Programs must be complete in themselves, in the sense that **for** commands must be matched by **loop** commands within the program, **next** and **break** commands must refer to loops within the program, and **jump** commands must refer to labels within the program.

- The lengths of **for** loops are unrestricted in programs, as is the relative positions of **jump** commands and the labels that they refer to.

- Labels within each program are local to the program, and the same name may be used in several programs safely.

- The values of variables named in **local** commands are restored when the program exits (recommended for all variables local to a program, thus avoiding any clash with variable names in use outside a program).

Programs are invoked or called by the **library** command, for example:

```
library ctop
library smm
```

Note that all the letters of the program name are significant, and that you cannot abbreviate the name like most Semper names. You can also use the **library** command quite freely within a program as long as you do not exceed the installation-dependent maximum depth for the nesting of programs (use the **show system** command to find out what this is). If a **return** command in a program is executed, it causes interpretation to resume immediately after the **library** command that invoked the program.

Programs are stored in program libraries. These are created and assigned in much the same way as picture disc files, for example:

```
assign program name 'semper'
assign program name 'temprogs' new size 30
```

Figure 7-1 illustrates the way in which program libraries are held by Semper.

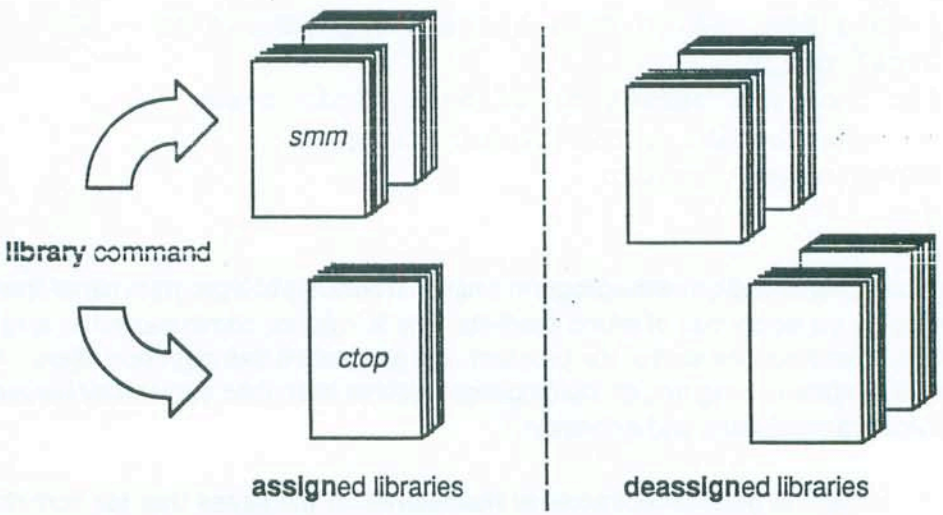

Figure 7-1. Program Libraries

The command **show programs** lists the names of all the programs in any libraries currently assigned.

> You may apply the **deassign, directory** and **compress** commands to program libraries as to other disc devices. Within a program library, a basic directory of program names is supplemented with tables of **for** command and label positions for each program, making interpretation of large programs possible without significant overheads.

You load a program into a program library using the **add** command and it replaces any existing program with the same name. You can also input a program directly from the terminal. Here you must specify the program name as part of the **add** command and not input the header command. For example, the program *ctop* would be input via the terminal like this (computer output is shown in **bold**):

```
add program 'ctop'
Program input-> r=mod(x,y)
Program input-> theta=phase(x,y)
Program input-> end
```

If the **program** key is not used, a prompt appears asking you to supply the program name. The **end** command terminates program input. You can also abort program input at any time with an abandon request. This form of program input is simple and convenient when loading small programs. If you make a mistake, you simply input the program again. For large programs this would become rather tedious. The **add** command lets you load programs from a text file instead. You specify the filename with the **name** key. For example, if the program *smm* is stored just as it appears above, in a text file called *smm.spl*, it can be loaded with the following command:

```
add name 'smm.spl'
```

A file can contain more than one program. Each program must start with a header command and finish with the **end** command.

> The **add** command can also load a program from a numbered macro (see *7.5.2, Numbered macros*). There are a number of restrictions in using numbered macros and they are likely to be dropped in later versions of Semper. You can avoid any problems by turning numbered macros into programs. The following example turns macro 2:100 into a program called *macro100*:
>
> ```
> add macro 2:100 program 'macro100'
> ```

Programs can be renamed, copied and deleted by means of the commands **rename, copy** and **delete**, for example:

```
rename program 'oldprog' as 'newprog'
copy program 'prog1' as 'prog2'
delete program 'duffprog'
```

The **list** command prints the contents of a program on the terminal, for example:

```
list program 'smm'
```

The **list** command also accepts a **name** key to allow you to copy the contents of a program into a text file (the reverse of the **add name** command). For example:

```
list program 'smm' name 'smm.spl'
```

is the reverse of the example using the **add name** command. The **list** and **add** commands in combination allow you to edit programs outside Semper using whatever editor is preferred locally. There is no facility within Semper for editing programs (although some Semper systems have a **spawn** command that creates a subprocess within Semper to enable you to communicate with the operating system and therefore edit files from within Semper).

Typically, one program library is shared between users at an installation and others are used by individuals for their own purposes. When more than one library is assigned, Semper maintains a search order amongst them, which determines which program is accessed when the same name is found in more than one library and to which library programs are added in default. The last assigned program library comes first in the search order by default. You can change the search order by using the **order** command. For example:

```
order 3,6,4
```

places devices 3,6 and 4 (which must of course be program libraries) first, second and third in the search order. With the commands **add** and **list** you can override the search order by means of the device key, for example:

```
add program 'smm' name 'smm.spl' device 4
list all device 3
```

The **order** command by itself lists the current search order.

Further examples of complete programs are given in *Appendix A, Sample Semper Programs.*

7.3 Run files

A series of commands stored in a text file can be executed by means of the **run** command. The interpreter executes each command in turn, starting with the first command in the file, until Semper encounters a **return** or **end** command or it reaches the end of the file. In the event of an error, a **return** is executed immediately. The **end** command should be the last command in the file. For example:

```
run name 'commands.run'
```

executes the commands in the text file called *commands.run*.

jump commands in run files may refer to labels anywhere in the file (the interpreter simply searches forward from the beginning for any label not found within the current command line). The maximum length of **for** loops, and the scope of **local** commands, are the same as they are interactively. **Library** commands may be used freely in run files.

The **run** command may not be executed from within a program or from within a **for** loop.

The **run** command should always be the last command in a command line because when Semper reads in the first line from a run file, the rest of the command line after the **run** command is lost.

A variety of mechanisms can be used for truly off-line (batch mode) running of programs, and their details are necessarily dependent on the local operating system. The simplest is to put the commands of real interest in a program or run file, and provide in the batch input stream (the 'terminal' input to Semper) a single **library** or **run** command initiating this. Semper behaves identically in batch and interactive environments, and an error in the program or run file causes it to revert to the batch input stream just as if it were a terminal.

7.4 The startup file

The most important use of the run file mechanism is the initialisation of a session via a file called the *startup file*. The name of this is installation-dependent, but where possible the name *semper.run* is used. Semper automatically executes a **run** command using this file at the start of each session. This allows you to tailor the initial environment to suit particular needs. The contents of the file obviously vary widely, but the following is a typical example:

```
! Semper 6 personal startup file
assign name 'pictures'                  [main picture storage device]
cd=n                                    [initialise cd (n is set by assign)]
assign help name 'helplib'              [shared help library]
assign program name 'proglib' wp        [shared program library]
assign program name 'myprogs'           [personal program library]
assign display                          [display device]
lut 1 create                            [define at least one lut]
partition 1 size 512                    [define at least one partition]
erase=yes                               [set default option]
end
```

The actual file names are, of course, installation-dependent.

7.5 Macros

Macros in Semper provide a simple textual substitution mechanism that allows much used strings or short command sequences to be invoked without being typed in full each time. Formally, they are names or numbers with an associated replacement text, and are called by constructions of the form ***@name*** or ***@number***, which are interpreted as if the replacement text had appeared instead.

7.5.1 Named Macros

Named macros are used to provide shortened versions of fixed combinations of keys and options. These are quite common because of the way in which information is passed from one Semper command to another. There are commands that store values in certain variables and these values can be picked up by other commands via keys and options.

For example, all installations have a named macro **region** defined as:

```
size r,r2 position x,y
```

This is used in conjunction with the command **xwires region**, which allows you to define a rectangular subregion of a displayed picture via the display cursor. The **xwires** command will set the variables *r* and *r2* to record the subregion dimensions and the variables *x* and *y* to record the centre position of the region. The **@region** macro can subsequently be used with any of the commands that use the standard 2-D subregion keys, for example:

```
display @region
cut display 3:7 @region
```

The command **show macros** lists all the named macros that are defined in Semper.

> Named macros are established in the *Verb Descriptor* file that defines the processing commands for the installation.
>
> Macro calls may be nested as desired, that is, the text of a macro may itself contain @ constructions. However, the total length of the command line with all calls expanded, is still subject to the normal limit.

7.5.2 Numbered Macros

Numbered macros differ from named macros in that you can define and redefine them at will within a Semper session. They are stored in *Macro* class pictures.

The **macro** command, which takes a single picture number expression as its argument, causes any commands remaining on the current input line (optionally continued as usual) to be stored as a *Macro* 'picture', with the number indicated. For example:

```
macro 90; xwires region; survey @region; +
+ type 'rms contrast ',100*sd/mean,'%'
```

This command defines a macro for reporting the fractional contrast level within a region marked on the display with the cursor. A later command **@90** is then interpreted exactly as if the stored text had been encountered instead. Note that you cannot use the **macro** command within a program.

The **list** command lists the text of macros, and edit invokes a simple editor for altering them without retyping them entirely.

Because macros are handled as pure text, they need not contain complete commands. For example:

```
macro 32; text 'Image at defocus ',d#n
for n 1,4; title 40+n @32; loop
```

is a convenient way to set the titles for the pictures 41 to 44.

Macros invoked from a program must not contain labels or any of the commands **for**, **loop**, **break**, **next** or **jump**. Numbered macros will also cease to be supported in later versions of Semper. All in all, the use of numbered macros is not recommended. Existing numbered macros can be converted into programs (provided they contain complete commands) by means of the **add macro** command (see *7.2, Programs*).

7.6 Error trapping

Semper allows experienced users to trap errors if they wish, by-passing the normal error handling process (see *3.15, Errors and abandon requests*) and allowing command interpretation to continue. The command **show errors** lists all error messages produced by the system, together with a numerical code for each. If the fixed variable *trap* is set to a particular error number at the start of a given command, the interpreter ignores the error in question should it occur. If *trap* is set to –1, all errors are ignored.

In any case, the error code returned by an extension routine is always transferred to the fixed variable *rc* for testing if required. Thus, the following sequence reports all unused picture numbers in the current device (30 is the error code returned for a non-existent picture):

```
for n=1,999; trap=30 select n; if rc=30 type n,' is free'; loop
```

Error trapping is a dangerous pastime, so *trap* is reset to zero by the interpreter at the outset of each new command, and must be set by a prefixed assignment to be effective.

Unless the local operating system provides a means of suppressing them, numerical errors such as division by zero will usually end in the session being aborted entirely by the system, perhaps with some diagnostic information. These errors should occur only rarely, as Semper takes reasonable care to trap them before they happen; it would however require unreasonable care (and reduce processing speed significantly) to guarantee their avoidance in all circumstances, and processes such as repeated

squaring of a picture will usually lead to exponent overflow or underflow errors eventually.

If, after trapping an error, you want to output the error message after all, you may do so with the **report** command, that is:

```
report trap
```

Chapter 8

THE USER INTERFACE SYSTEM

8.1 Overview

Semper 6 *Plus* contains nine commands which are provided for creating and manipulating user interfaces. The user interface system allows the Semper command line interface to be hidden completely and replaced by a menu system which can still execute all the standard Semper commands.

A demonstration user interface is provided and is documented in the following manual:

Tutor User Guide

This interface does not attempt to hide Semper but is meant to be an aid to learning about Semper itself – which commands to use, how to use them, etc. The programming of user interfaces is described in the manual:

User Interface Guide

This gives several examples of how to program with the user interface commands as well as a description of the different parts of the user interface. These two manuals are contained in the *Semper 6 Guide*.

8.2 User Interface Commands

The following commands are concerned solely with the user interface system:

- **uif** controls the user interface system.
- **device** examine/control the display
- **mouse** examine/control the mouse
- **execute** commands executed before/after every command
- **textfield** user interface object
- **cell** user interface object
- **menu** user interface object
- **panel** user interface object
- **justification** controls object positioning

These commands are described in the *Semper 6 Command Reference*, the *Quick Reference List* and the on-line help. With these commands it is possible to provide several different styles of user interface since they have various keys and options which allow their behaviour to be modified.

Figure 8-1 illlustrates the elements of a typical user interface.

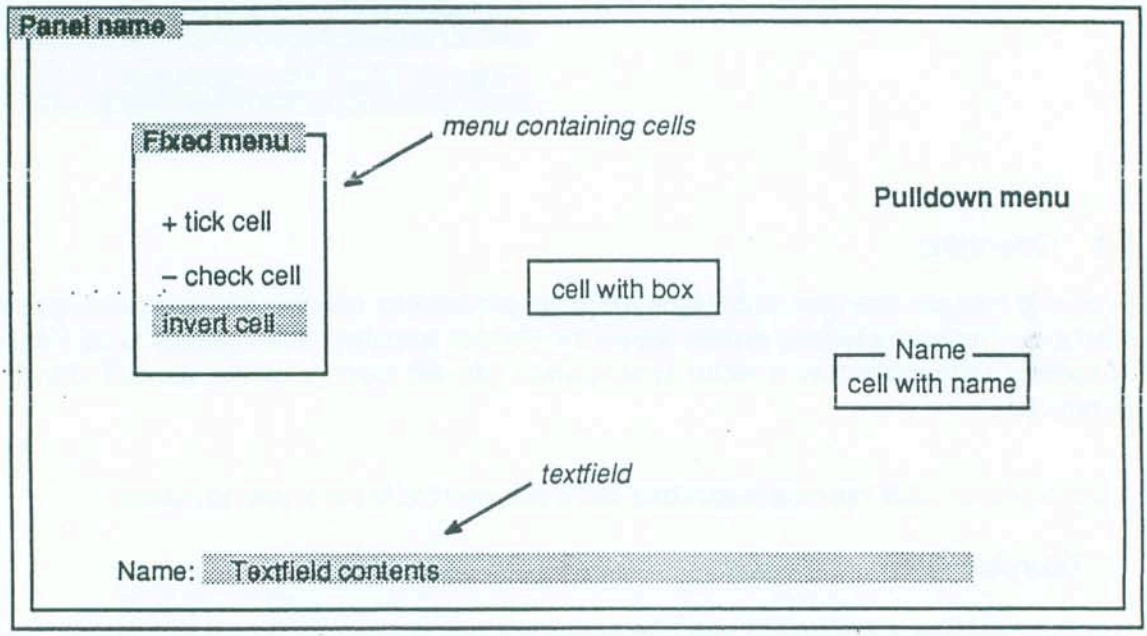

Figure 8-1. A Typical User Interface

Appendix A

SAMPLE SEMPER PROGRAMS

A.1 Overview

The sample programs that are given below reconstruct a 2-D picture from a set of simulated 1-D projections in different directions. They illustrate how the basic processing commands can be combined to form relatively sophisticated procedures.

The reconstruction is performed using the multiplicative ART algorithm: for each direction in turn, the projection columns are re-scaled so as to match the experimental data, and the whole process is repeated as often as necessary.

To run the programs, type the command **!library prepare** at the terminal, followed by **!library recon**. The first of these uses a standard library program molecule to prepare a test field, and then projects the picture in a series of directions, 0.3 radians apart. The variable *p* is used for the output picture number of a projection, and *t* for the angle at which it is made.

The reconstruction is carried out by *recon*. An evenly bright picture is set up to start the algorithm, and two cycles of ART are performed (counted by *ncycle*). Each cycle consists of taking each of the projection directions in turn, and scaling the columns of the reconstruction so that they are consistent with the corresponding projection data. This is achieved for a given direction by projecting the current reconstruction, dividing the projection data by those thus obtained (with 0.1 added to prevent division by zero), and multiplying the reconstruction columns by the resulting ratios.

A.2 Sample Program 1

```
Prepare(); ! Prepare model in 901, projections in 903, 904, etc.
in=901 dt=0.3; library molecule
display times 4; mark=yes
p=903; for t=-pi/2,pi/2,dt; project in to p angle t; p=p+1; loop
end
```

A.3 Sample Program 2

```
Recon(); ! Reconstruct projections into 902 (Multiplicative
ART)
rec=902 dt=0.3
create rec size 32 value 1; title text 'ART reconstruction'
min=0 max=2; display rec times 4 preset; mark=yes
```

```
for ncycle=1,2; p=903
for t=-pi/2,pi/2,dt; project rec to 999 angle t
calculate :p/(:999+0.1) to 999
backproject multiply rec with 999 angle t
display rec times 4 preset
p=p+1
loop t
type 'Reconstruction after cycle', ncycle
loop ncycle
end
```

Appendix B

GLOSSARY OF TERMS

B.1 Overview

This glossary provides definitions of the technical terms that are used to describe Semper 6 software. The terms are arranged in alphabetical order.

Abandon

(see *2.2, The Semper environment* and *3.15, Errors and abandon requests*)
Abandon request; request made by a special keyboard character that Semper should abandon the current process and seek fresh instructions.

Annotations

(see *5.5.1, Overlays and annotations* and *6.3, General keys and options*)
Text and graphics marked in a display overlay.

Argument

Item qualifying some basic item, for example, an option or key to a processing command, or an input value to a mathematical function such as *sine*.

Arithmetical expression

(see *2.9, Expressions*)
Combination of numbers, variables, function calls and expressions with arithmetic operators, resulting in a numerical value.

Arithmetical operator

(see *2.9, Expressions*)
*, /, +, –, ^ or :

Assignment

(see *3.4, Assignments*)
Construction of the form *name=expression* assigning a value to a variable; may be prefixed to commands or stand alone.

Assumed key

(see *5.2, Picture numbers in processing commands*)
Key whose name is assumed by the interpreter when omitted.

Batch mode

Mode of running with no user interaction or terminal at all.

Block

Unit in terms of which Semper manages disc storage; typically 64 to 512B; need not match block size used by actual peripherals.

Byte

(see *4.4, Forms*)
Pixel form using 8 bits, with possible values in the range 0 to 255.

Cache buffer

(see *5.3, Disc files*)
Memory area used to buffer disc data; unit in terms of which all physical disc transfers are made.

Cell

(see *Chapter 8, The User Interface System*)
The basic building block of the user interface system; used in creating menus, buttons, etc.

Character

(see *2.3, Characters*)
Letter, digit or punctuation sign used to make up Semper commands – the allowed set is that of the ASCII standard.

Class

(see *4.3, Classes*)
Name and number coding the kind of data held by a picture.
or
The clustering of pixel values about a point in multi-spectral space.

Classification

The process of labelling pixels in an image according to their class. Supervised classification uses *training regions* to identify the characteristics of the separate classes.

Colon operator

(see *2.9, Expressions* and *Chapter 5, Devices and Storage*)
Special arithmetical operator combined with numerical expressions representing a device and picture numbers, resulting in a numerical value. The device number expression may be omitted.

Column

Group of pixels arranged in a vertical line as viewed.

Conditional clause

(see *3.7, Conditional clauses: IF and UNLESS*)
If or **unless** construction prefixed to a command making its execution conditional.

Continuation line

(see *3.2, Command Line Structure*)
An input line appended to part-line already entered but terminated with a plus (+) sign.

Command

see (see *3.2, Command Line Structure*)
Single instruction to Semper, involving a command name with optional arguments, label, conditional clause and/or assignment.

Command line

(see *3.2, Command Line Structure*)
One or more commands separated by semicolons; possibly formed from several continuation lines.

Comment

(see *3.11, Comments*)
A command introduced by an exclamation mark, ignored by the interpreter.

Complex

(see *4.4, Forms*)
Pixel form using two *fp* values, used for values with imaginary as well as real parts, such as *Fourier* transforms.

Console

(see *3.16, Controlling output in Semper*)
Logical output stream for normal Semper output.

Coordinates

(see *4.5, Coordinates and origins*)
Set of three numbers giving the pixel position in each of the three directions, relative to the origin.

Current device

(see *Chapter 5, Devices and Storage*)
Default device number, determined by the variable *cd*.

Deferred

(see *2.2, The Semper environment* and *Chapter 7, Indirect Interpretation*)
Indirect; mode of operation in which executed commands have been stored previously instead of coming directly from the terminal.

Device

(see *Chapter 5, Devices and Storage*)
Disc file, magnetic tape or display, identified by number 1, 2... within which Semper stores 1–999 pictures.

Dimensions

(see *4.2, Dimensions*)
Set of three values giving row length, column length and number of layers in a picture.

Disc

(see *5.3, Disc files*)
Disc file, or picture disc file; disc file within which Semper stores 1–999 pictures with an internal picture directory.

Display

(see *Chapter 5, Devices and Storage* and *6.2, Protected and fixed variables*)
Protected variable identifying last display picture used.

Dyadic expression

(see *2.9, Expressions*)
An expression which includes a dyadic operator and more than one argument, resulting in a numerical or logical value, for example, 2+2.

Dyadic operator

(see *2.9, Expressions*)
An operator which takes more than one argument, for example, +,–,*,/.

Echo

(see *3.16, Controlling output in Semper*)
Reflection of output to the terminal or log file.

Error code

(see *3.15, Errors and abandon requests* and *7.6, Error trapping*)
Number used internally by Semper to indicate the nature of some error. The number is included by Semper in the displayed error message.

Expression

(see *2.9, Expressions*)
Combination of numbers, variables, function calls and expressions with arithmetical, relational and logical operators, resulting in a numerical value.

False colour

(see *5.5.3, Look-up tables*)
View generation mode in which a single frame is presented with different values mapped to different colours.

Frame

(see *5.5, Displays*)
Single picture store within a framestore; typically 256 to 1024 pixels square, with 8 bits/pixel.

Frame coordinates

(see *5.5.2, Graphical coordinate systems*)
Distances rightwards and upwards from the display frame centre.

Framestore

(see *5.5, Displays*)
Multiple grey-level picture store with 2-D organisation and continuous TV viewing capability; may contain more than one frame.

Form

(see *4.4, Forms*)
Name and number coding pixel representation (**byte**, **integer**, **fp** or **complex**).

Fp

(see *4.4, Forms*)
Floating point pixel form allowing positive or negative integral or fractional values, very small or very large.

Full colour

(see *5.5.3, Look-up tables*)
View generation mode in which three separate frames are presented as red, green and blue components of a colour image.

Function call

(see *2.8, Functions*)
Instance of mathematical function such as *sine*, involving function name followed by argument expression(s) in brackets.

General key/option

(see *6.3, General keys and options*)
Key/option accepted by all processing commands.

Help library

(see *3.14, Help*)
Disc file from which Semper retrieves information for the **help** command; maintained by the independent *Help Manager.*

Implicit deletion

(see *5.3, Disc files*)
Deletion of picture/program by creating new one with the same number/name; in contrast to the explicit deletion of the **delete** command.

In situ

In place; processing for which output shares storage of source.

Indirect

(see *2.2, The Semper environment* and *Chapter 7, Indirect Interpretation*)
Deferred; mode of operation in which executed commands have been stored previously instead of coming directly from the terminal.

Indexed name

(see *2.7, Indexed names*)
Name with # and variable name appended, interpreted as if the name has the decimal representation of the appended variable.

Installation-dependent

Variable between Semper installations, because dependent on the type of computer or because of the choices made when compiling the system.

Integer

(see *4.4, Forms*)
Pixel form normally using 16 or 32 bits, allowing positive or negative integer values.

Interactive mode

Typing commands directly at a terminal for immediate execution.

Key

(see *3.5, Processing commands*)
Argument to a processing command with a numerical value.

Label

(see *3.9, Labels and jumps*)
Name with a colon appended, at the start of a command, marking a position in a command line, program or run file.

Layer

(see *Chapter 4, Pictures*)
2-D picture; comprises pixels organised in rows and columns; may be stacked to form a multi-spectral or 3-D picture, with layers numbered 1, 2... from the back.

Literal text

(see *2.10, Literal text*)
Arbitrary sequence of characters enclosed in single quotes (a quote itself can be represented by two successive quotes).

Local setting

(see *3.5, Processing commands*)
Setting of variable made for the duration of a command only, with the original state restored afterwards.

Log file

(see *3.16, Controlling output in Semper* and *Chapter 5, Devices and Storage*)
Assigned text file for recording Semper results.

Logical expression

(see *2.9, Expressions*)
Combination of numbers, variables, function calls and expressions, all treated as logical values, with logical operators, resulting in a logical value. An expression with the result treated as a logical value.

Logical operator

(see *2.9, Expressions*)
~, & or |

Logical output stream

(see *3.16, Controlling output in Semper*)
One of six different output streams used to distinguish between the different types of Semper output.

Logical value

(see *2.9, Expressions*)
True or false, represented by numerical values 1 and 0 respectively. A numerical value, encountered where a logical value is expected, is treated as false if zero and true if non-zero.

Look-up table, lut

(see *5.5.3, Look-up tables*)
Intensity transformation table mapping the pixel values stored in the framestore to the colours/intensities presented by the view generator.

Loop

(see *3.8, Simple loops: the FOR command* and *3.9, Labels and jumps*)
Sequence of commands interpreted repeatedly, under control of **for** or **jump** commands.

Macro call

(see *7.5, Macros*)
Command item of the form ***@name*** or ***@number***, equivalent to the replacement text of the specified macro.

Menu

(see *Chapter 8, The User Interface System*)
A matrix of cells.

Monitor

Facility providing internal debugging information (not supported).

Monitor view

(see *5.5.2, Graphical coordinate system* and *5.5.4, Viewing conditions*)
The part of framestore memory visible on the video monitor.

Multi-component assignment/key

(see *3.4.1, Multi-component assignments* and *3.4, Assignments*)
Use of an assignment/key, involving two or more expressions, and setting two or more variables.

Name

(see *2.4, Names*)
Begins with a letter, continues with letters and/or digits, and identifies commands, variables, keys, options, functions, labels, named macros and programs. First three characters are significant except for program names where all characters are significant.

Named macro

(see *7.5.1, Named macros*)
Name with a fixed associated replacement text.

Numbered macro

(see *7.5.2, Numbered macros*)
Number with a dynamically assigned associated replacement text. Stored in a *Macro* class picture.

No

(see *6.2, Protected and fixed variables*)
Protected variable with value 0 (false).

Number

(see *2.5, Numbers*)
Digit string with optional sign, decimal point and decimal exponent, making up a numerical value.

Numerical expression

(see *2.9, Expressions*)
Expression with the result treated as a numerical value.

Numerical value

(see *2.9, Expressions*)
Any numerical quantity capable of being stored in floating-point form by the host system.

Operator

(see *2.9, Expressions*)
Arithmetical (+, –, *, /, ^, :), *relational* (<, >, =, ~=, <=, >=) or *logical* (~, & or |) operator combined with one or two numerical or logical values to give a resulting numerical or logical value.

Option

(see *3.5, Processing commands*)
Argument to processing command expecting value *yes* or *no*.

Origin

(see *4.5, Coordinates and origins*)
Nominal picture centre position, relative to which the picture coordinates are defined.

Overlay

(see *5.5.1, Overlays and annotation*)
Additional display memory 1 bit deep, allowing display text and graphics to be defined independently of the picture storage.

Paged output

(see *3.16, Controlling output in Semper*)
Temporary suspension of output to the terminal to prevent text from scrolling too fast.

Panel

(see *Chapter 8, The User Interface System*)
The foundation on which user interfaces are created.

Partition coordinates

(see *5.5.2, Graphical coordinate systems*)
Distances rightwards, upwards from the display partition centre

Panning, scrolling

(see *5.5.4, Viewing conditions*)
Movement relative to the display frame of a subregion presented by the view generator.

Picture

(see *Chapter 4, Pictures*)
Array of numerical values, 1-D, 2-D or 3-D, containing picture brightness values or other data.

Picture coordinates

(see *5.5.2, Graphical coordinate systems*)
Distances rightwards, upwards from the picture origin.

Pixel

(see *Chapter 4, Pictures*)
Picture element; one of the numerical values making up a picture.

Prefix

Label, conditional clause or assignment placed at the beginning of a command.

Processing command

(see *3.5, Processing commands*)
Command defined in the *Verb Descriptor* file, calling a particular processing module.

Program

(see *7.2, Programs*)
Self-contained sequence of Semper commands invoked from the terminal by a **library** command.

Program library

(see *7.2, Programs*)
Disc file containing programs, assigned as a device.

Prompt

(see *3.2, Command line structure*)
Short message on the terminal inviting the user to type a command or other information.

Relational expression

(see *2.9, Expressions*)
Combination of numbers, variables, function calls and expressions with relational operators, resulting in a logical value.

Relational operator

(see *2.9, Expressions*)
<, >, =, ~=, <= or >=.

Return code

(see *6.2, Protected and fixed variables* and *7.6, Error trapping*)
Error code produced by preceding command, available in the variable *rc*.

Row

Group of pixels organised in a horizontal line as viewed; the first (lowest, closest) level of grouping in Semper storage.

Run file

(see *7.3, Run files*)
File of Semper commands executed in response to a **run** command.

Save file

(see *5.7, Getting images into Semper*)
Dynamically created file in which a group of Semper pictures is saved by the **save** command.

Scroll

Displacement of monitor viewing window relative to the display frame.

Select

(see *5.2, Picture numbers in processing commands*)
Protected variable identifying the last used picture.

Startup file

(see *7.4, The startup file*)
File of commands used to initialise the environment for a Semper session.

Syntax

Grammatical construction of a command, as opposed to the meaning.

Tab

(see *3.6, The TYPE and LOG commands*)
Construction of the form *#expression*, establishing position at which the next item of type – item list is output.

Terminal

Keyboard-cum-screen/printer used for typing commands and receiving responses.

Textstring

(see *3.6.1, Textstrings*)
List of items of literal text, tabs and/or expressions, used to construct text.

Title

(see *4.7, Titles*)
Short descriptive message stored with a picture.

Training region

An area of an image which has been identified as belonging to a particular class.

Unary expression

(see *2.9, Expressions*)
Number, variable, function call or expression preceded by a unary operator, resulting in a numerical or logical value.

Unary operator

(see *2.9, Expressions*)
: +, –, ~ or :

Variable

(see *2.6, Variables*)
Name with associated (usually changeable) numerical value.

Verb Descriptor file

(see *3.5, Processing commands*)
Collection of syntax definitions and other information defining processing commands; user-extendable.

View generator

(see *5.5.4, Viewing conditions*)
Part of the framestore hardware generating TV image from stored data; may provide for zoom, scroll, look-up-tables.

Work disc

(see *2.2, The Semper environment*)
Scratch storage used for various purposes, especially display picture labels and luts.

Write-protected, wp

(see *4.9, Write-protection* and *5.6, Write-protect and write-only flags*)
State of picture or device in which data can be inspected and used, but not altered.

Yes

(see *6.2, Protected and fixed variables*)
Protected variable with the value 1 (true).

Zoom

(see *5.5.4, Viewing conditions*)
Magnification factor relative to the sub-region of a frame presented by the view generator.

1-D picture

(see *4.2, Dimensions*)
Picture with one row only; graph, function, etc.

2-D picture

(see *4.2, Dimensions*)
Picture with more than one row but only one layer; image etc.

3-D picture

(see *4.2, Dimensions*)
Picture with more than one layer; colour or multi-spectral image, 3-D density distribution etc.

Index

Numbers

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z