A Hands-on Guide to VisiQuest

To Mary and Valéria

Preface

This tutorial is a practical overview of the entire VisiQuest system. It is not intended to replace the manuals. Instead, it was designed to be the quickest way to learn about the system.

This tutorial takes the user step by step through the different VisiQuest subsystems. Its hands-on approach is similar to a laboratory guide. For best results, the user should work through the material in a linear fashion, without skipping any section or chapter. We carefully chose simple and practical experiments to present the functionality and capability of the VisiQuest system.

Chapter 1 gives an overview of the VisiQuest system, explaining how it is used in data processing, visualization, and software development. It introduces the principal VisiQuest components. It also presents the organization of the VisiQuest system toolboxes. Lastly, it introduces the internal data model and the ways the user community can interact with the VisiQuest developers.

Chapter 2 presents the three types of user interfaces: command line, graphic, and visual programming. The goal of this chapter is to introduce you to the VisiQuest visual programming environment. It also describes how the information is stored, processed, and visualized. Beginning with this chapter you will start experimenting with the system.

Chapter 3 continues to present `VisiQuest` programming features to solve practical problems of importing and exporting data, and how to interface with other software packages. More advanced features are presented, such as `VisiQuest` variables and control connections.

Chapter 4 makes extensive use of the VisiQuest data model to represent True Color and Indexed Color images. The validity mask concept is presented.

Chapter 5 presents advanced visual programming features, such as procedures, flow control, and data transport mechanisms.

Finally, Chapter 6 contains a brief introduction to the software development tools. In this chapter you will create a toolbox and develop two illustrative data processing operators.

Throughout the book, many interesting and practical problems are presented. Our intention is to use the VisiQuest system as a tool to assist you in learning and thinking in the field of data processing and visualization. VisiQuest does not solve problems by itself.

You should have at least some experience of the UNIX operating system, to be able to navigate through the file system, copy, delete, move files, and use a text editor. We assume that you have some practice with window managers and the X Window System so that you are comfortable with window operations such as move, raise, lower, focus, refresh, destroy, and iconify. We also assume that you know how to program a computer.

Several registered trademarks appear in this tutorial: UNIX is a trademark of X/Open Company, Ltd.; X Window System is a trademark of X Consortium, Inc.; `xv` is copyrighted by John Bradley; `gnuplot` is copyrighted by T. William and C. Kelly; PostScript is a trademark of Adobe Systems, Incorporated.

Typographical conventions

In this tutorial the following conventions are used:

Convention	**Description**
Note:	This is a note
Tip:	This is additional useful information
`%command` `result`	This is a command and its result
`VisiQuest`	Binary executable names
Width	A parameter input
`Display Image`	A glyph
`my input`	A string that the user has to type in
Run	A button
Menu, submenu	The menu and submenu names

Description of the practical experiments is mixed within the text. Paragraphs on which you have to make an action with the software are specified by numbers as illustrated below.

1. Here is described an action you must take.

   A general comment or an explanation may follow the action.

2. Here is another action.

Contents

Chapter 1

Introduction

In modern problem solving there is a need to process large data sets, multidimensional and time varying data. Applications that require modeling, simulation, animation, and algorithms to extract information from the data demand computer-intensive operations. Finally, problem solving often requires collaboration among interdisciplinary teams.

The area of data processing has seen a significant increase in the level of interest from many disciplines. Data processing is continuously expanded by areas such as neural nets, wavelet theory, mathematical morphology, data compression and recognition, and artificial intelligence. With the dramatic drop in the cost of digital systems and the equally dramatic increase in the level of performance, the use of data processing in other disciplines is feasible. Data processing plays a crucial role in the areas of document processing, remote sensing, medical imaging, scientific visualization, telecommunications, robotics, biology, and environmental engineering, among others.

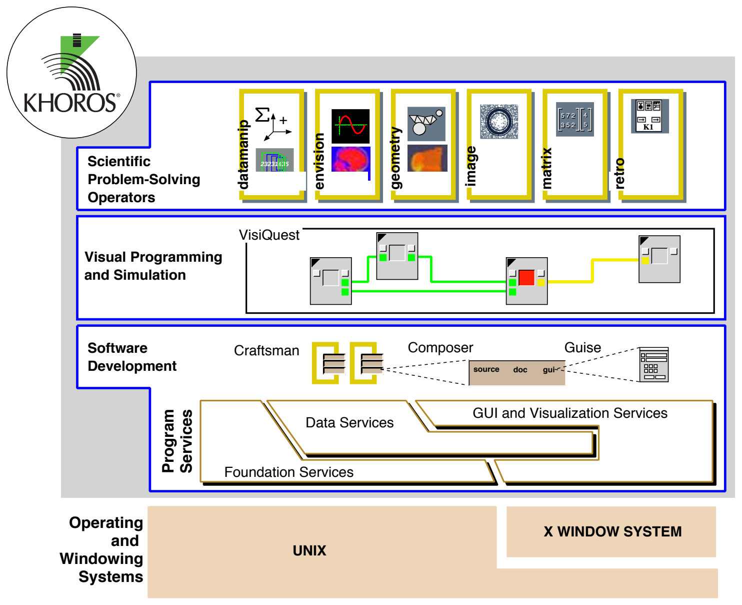
In data visualization, we take advantage of the advances in technology (computers, data networks, user interfaces, data processing algorithms) and combine them with the powerful human visual system to solve problems. A principal goal in data visualization is to gain understanding and insight into the data under observation. To facilitate this process, a fast prototyping environment and data processing and visualization operators are needed.

What is VisiQuest?

VisiQuest is a software integration and development environment that emphasizes information processing and data exploration. The objective of the ongoing work of VisiQuest development is to build a complete application development environment that redefines the software engineering process to include all members of the work group, from the application end user to the infrastructure programmer, in the productive creation of software. The VisiQuest environment accomplishes this objective

- by providing a broad set of program services (libraries) that can be used as a source of reusable code for visual program development;
- by providing visual software development tools that can be used to quickly prototype new software and maintain developed software;
- by being portable and extensible;
- by supporting different levels of user interaction, from non-technical end user to expert programmer;
- and by motivating a broad community to collaborate utilizing both synchronous and asynchronous exchanges of resources and knowledge.

The figure shown below illustrates the different levels of user interaction.



For end user solutions to scientific problems, VisiQuest may be used as it stands, providing a rich set of programs for information processing, data exploration, and data visualization. Multidimensional data manipulation operators include pointwise arithmetic, statistic calculations, data conversions, histograms, data organization, and size operators. Image processing routines and matrix manipulation are also provided. Interactive data visualization programs include an image display and manipulation package, an animation program, a 2D/3D plotting package, a colormap alteration tool, and an interactive image/signal classification application. In addition, 3D visualization capabilities are offered: a number of data processing routines for 3D visualization are provided, along with a software rendering application. The VisiQuest operators are generalized, so that each can solve problems in a variety of specific areas such as medical imaging, remote sensing, process control, signal processing, and numerical analysis.

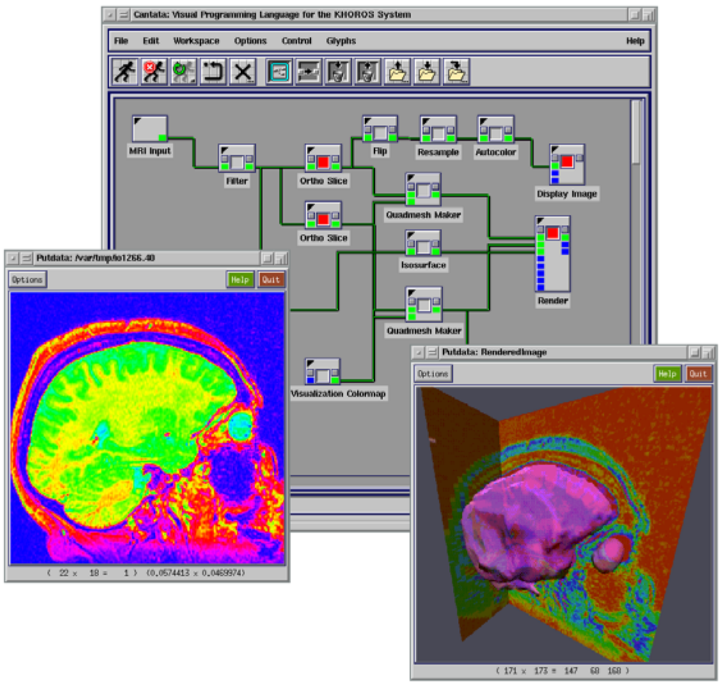
Visual programming environments provide iconic elements which can be interactively manipulated according to some specific spatial grammar for program construction. Data flow is a naturally visible approach in which an algorithm is described as a directed graph where each node represents an operator or function and each directed arc represents a path over which data flows. By connecting the data paths between nodes, users can interactively draw out a solution in an intuitive manner that matches their mental representation of the problem. Thus, a visual environment for problem solving introduces a level of abstraction which provides accessibility to the functionality represented by the underlying operators, regardless of the developer's programming experience; moreover, it increases the productivity of both experienced and novice developers.

Programs from the VisiQuest system are represented as visual objects called *glyphs*. The program which corresponds to the glyph is called an *operator*. To create a visual program, the user selects the desired operator, places the corresponding glyphs on the workspace, and connects the glyphs to indicate the flow of data from operator to operator, forming a *network* within a *workspace*.

Visual hierarchy, iteration, flow control, and expression-based parameters extend the data flow paradigm to make `VisiQuest` a powerful simulation and prototyping system. Data and control-dependent program flow is provided by *control structure glyphs* such as if/else, while, count, and trigger. Visual subroutines, or *procedures*, are available to support the development of hierarchical data flow graphs. Variables may be set interactively by the user, or calculated at run time via mathematical expressions tied to data values or control variables within the visual network.

By combining a natural environment of visual constructs with the programming features typically found in textual languages, `VisiQuest` provides a powerful problem-solving and prototyping system. Visual hierarchy, iteration, flow control, and expression-based parameters augment the traditional data flow paradigm so that `VisiQuest` can be used effectively in a number of domains, including process control, simulation, and system integration. When combined with the data processing and data visualization programs available in the VisiQuest system, `VisiQuest` is particularly well suited for scientific data processing and visualization.

The flexibility of the visual network scheduler allows this single visual programming environment to simultaneously address the needs of diverse domains. Furthermore, the visual programming paradigm is enriched by the ability to combine operators from different domains as desired in a single visual program, allowing the visual programmer to solve more complex and diverse problems than would otherwise be possible.



For application developers, the VisiQuest system consists of programming services and software development tools that support all aspects of developing new engineering and scientific applications. Applications written to VisiQuest can take advantage of the same capabilities offered by the VisiQuest data processing and visualization routines, including the ability to transparently access large data sets distributed across a network, operate on a variety of data and file formats without conversion, and maintain a consistent presentation with a standardized user interface. The software development environment provides developers with a direct manipulation graphical user interface design tool, automatic code generation, standardized user interface and documentation, and interactive configuration management. The VisiQuest software development system can be used for software integration, where existing programs can be brought together into a consistent, standardized, and cohesive environment.

VisiQuest provides a powerful working environment for the academic, engineering, and scientific communities, addressing many of the issues associated with quickly developing X Window-based applications, prototyping solutions to complex problems, and utilizing the resources of a distributed network. The layered approach of the VisiQuest infrastructure and the concept of program services provide developers with the flexibility to create complex applications, while at the same time hiding the intimidating details of operating systems and X Window Systems.

Organization of the VisiQuest software

A common misperception is that “VisiQuest” is a single application. In contrast to previous versions, VisiQuest does include a front-end application named “VisiQuest”, from which the VisiQuest visual programming environment, the manual, and demos are accessible. However, the VisiQuest system incorporates hundreds of programs and thousands of library calls. The VisiQuest system is distributed as a number of *packages*. Each package contains one or more *toolboxes*. VisiQuest is a complete data exploration and software development environment that reduces time in solving complex problems, allows free sharing of ideas and information, and promotes portability.

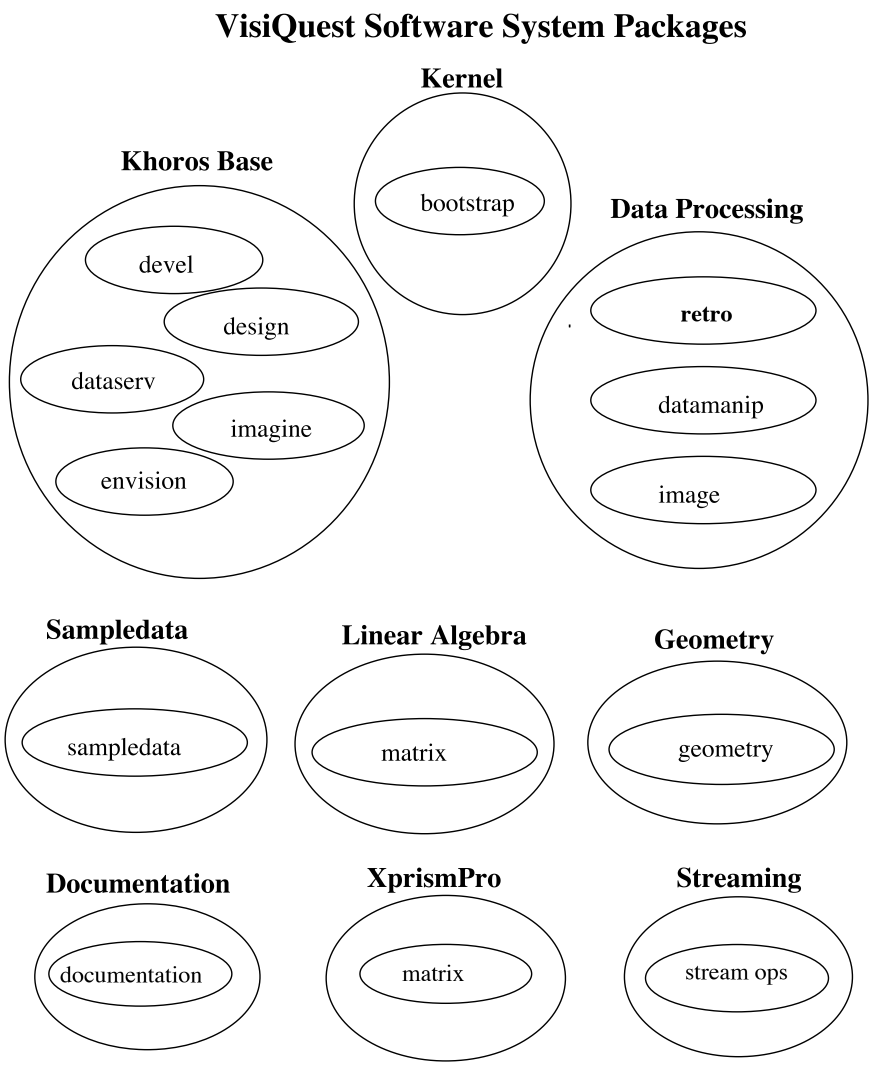
Each program of the VisiQuest system is located in a toolbox. A toolbox is a collection of programs and libraries that are managed as *objects*. A toolbox object is an encapsulation of programs and libraries normally related to a specific application. The toolbox object has a predefined directory structure in which its software objects are located.

A software object is composed of source code, documentation, and user interface. The different categories of software objects are: library objects, kroutine object, xvroutine objects, pane objects, script objects, and workspace objects. Except for workspace objects, software objects have one or more User Interface Specification (UIS) files that serve to specify the Graphical User Interface (GUI) and Command Line User Interface (CLUI) of the software object.

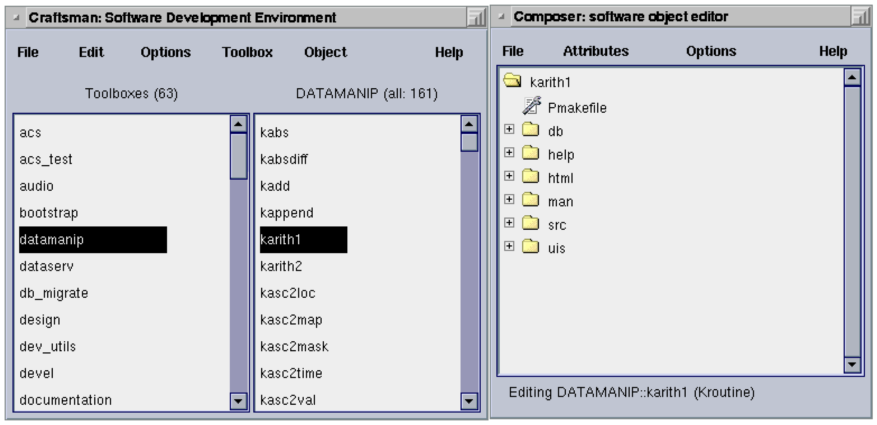
A toolbox contains programs and libraries which have a similar function or common objective. Similarly, the various VisiQuest packages group toolboxes according to purpose. The packages and toolboxes which make up the VisiQuest system include:

- The *Kernel* package, which contains the *bootstrap* toolbox (installation and make utilities, Foundation services).
- The *VisiQuest Base* package, which contains: the *devel* toolbox (software development environment), the *design* toolbox (the VisiQuest application, GUI & Visualization services), the *imagine* toolbox (the `VisiQuest` visual programming environment), the *dataserv* toolbox (data services), and the *envision* toolbox (scientific visualization programs).
- The *Data Manipulation* package, which contains: the *datamanip* toolbox (data processing routines), the *image* toolbox (image processing routines), and the *retro* toolbox (additional image processing routines).
- The *Geometry* package, which contains the *geometry* toolbox (3D visualization programs).
- The *Sampledata* package, which contains the *sampledata* toolbox (multi-dimensional data sets).
- The *Migration* package, which contains The *migration* and *db_migrate* toolboxes (utilities to assist in migration from an older version to a new version of VisiQuest).

- The *Linear Algebra* package, which contains the *matrix* toolbox (linear algebra and matrix computation routines).
- The *Streaming* package, which contains the *streamops* toolbox (streaming data services and operators).
- The *Plotting* package, which contains: the *XprismPro* toolbox (3D Plotting package using OpenGL).
- The *Documentation* package, which contains: the *documentation* toolbox (PDF files for VisiQuest online manual).

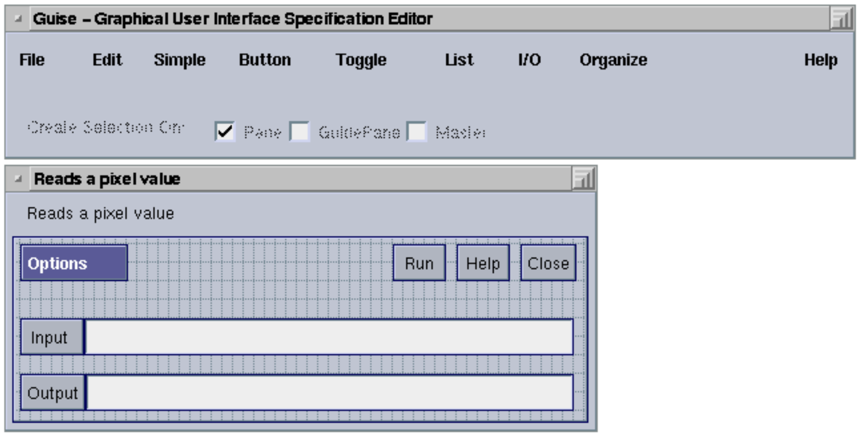


The VisiQuest system provides the `craftsman` tool for managing a toolbox. It allows the user to create, delete, and copy toolbox objects as well as software objects. The second tool that comes with this system is `composer`. `composer` is the user interface to the software objects and invokes all the operations needed to manage them.



Editing DATAMANIP::karith1 (Kroutine)

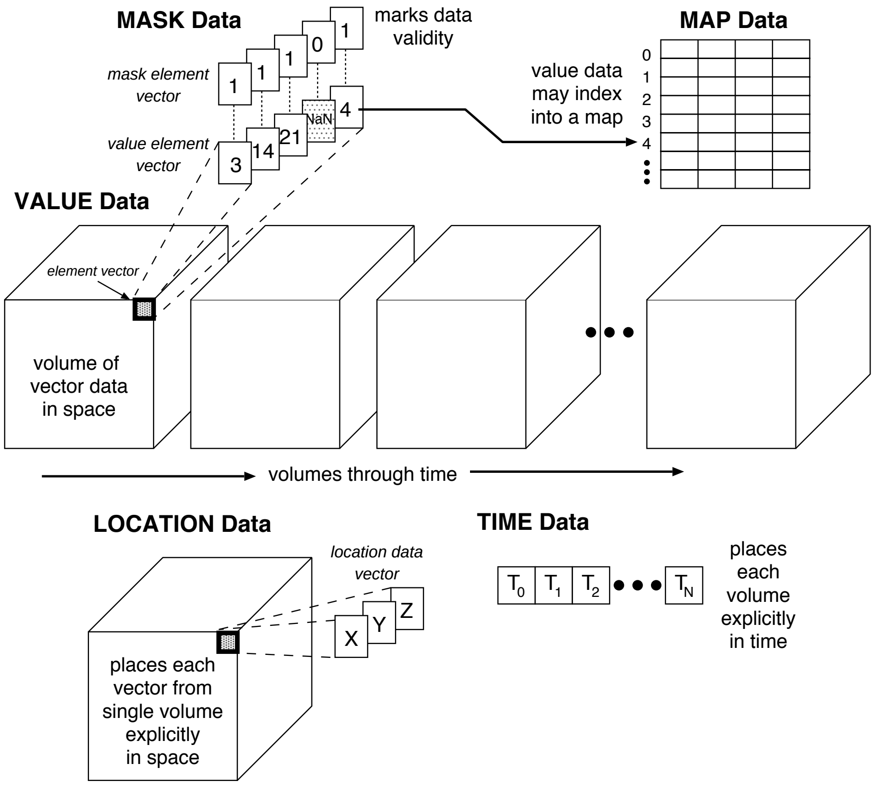
The third tool provided `guise`, is an interactive graphical user interface design tool.



Polymorphic Data Model

The VisiQuest system has its own data structure designed to manipulate multidimensional data. This data structure facilitates the processing of 1D signals like audio or speech, 2D signals such as images, and 3D data such as volume information. In addition to these three dimensions, VisiQuest has a time dimension to represent the data in time and an element dimension to represent more than one attribute to each data point. This data structure within the VisiQuest system is known as the *Polymorphic Data Model.*

The polymorphic model consists of data that exists in three-dimensional space and one-dimensional time. You can picture the model most easily as a time-series of volumes in space. This time-series of volumes is represented by five different data segments. Each segment of data has a specific meaning dictating how it should be interpreted.



An overview of the Polymorphic Data Model is shown above. The polymorphic model consists of five data segments, with each segment serving a specific purpose: *value*, *location*, *time*, *mask*, and *map*. All of these segments are optional; a data object may contain any combination of them and still conform to the polymorphic model.

VisiQuest community interaction

The VisiQuest system evolution has thrived on real-time user enhancement, on-line training, and permanent feedback from its broad spectrum of users.

The user community can interact with the VisiQuest developers via email and newsgroups:

- Send email to `info-request@accusoft.com` to subscribe to the VisiQuest mailing list.
- The web URL for VisiQuest is `http://www.accusoft.com`.
- The `kbugreport` utility is used to report bugs and facilitate interaction with the developers.

Chapter 2

Getting Started

The best way to get started with VisiQuest is to use the software. You benefit most from this tutorial if you have access to VisiQuest so you can apply each step as it is in this guide. This chapter walks you through a general overview of VisiQuest.

Installation guide reference

VisiQuest has a detailed guide on how to install the software system called the VisiQuest Installation Guide. If you are not responsible for the system installation, you may still want to read the section "Setting Up Your Environment" (section 2.5). The easiest way to set up the VisiQuest environment is by executing `kconfigure` as explained in that manual. VisiQuest is installed when the binaries are accessible and the file *.kri/KP2001/Toolboxes* is present in your home directory. The Toolboxes file dictates which toolboxes are available to you. Note that not all toolboxes mentioned in this guide will be available to you unless all the VisiQuest 2001 packages have been installed. All these issues are described in the VisiQuest Installation Guide.

Interfaces to Major VisiQuest Toolbars

VisiQuest is an environment for data processing and visualization and software development. The two principal tools of VisiQuest through which the user can interact with the system are `VisiQuest` and `craftsman`.

`VisiQuest` is a visual programming environment from which over three hundred data processing and visualization operators can be invoked. It can executed from the command line as follows:

```
%VisiQuest
```

`craftsman` manages the maintenance and development of software within the system, specifically the management of all the system toolboxes. It can be executed with the following command:

```
%craftman
```

Another VisiQuest tool, `composer` , deals with software objects individually. It can be executed from `craftsman` or from the command line. For example, to run `composer` on the "test1" software object of the TESTS toolbox, the following command would be issued:

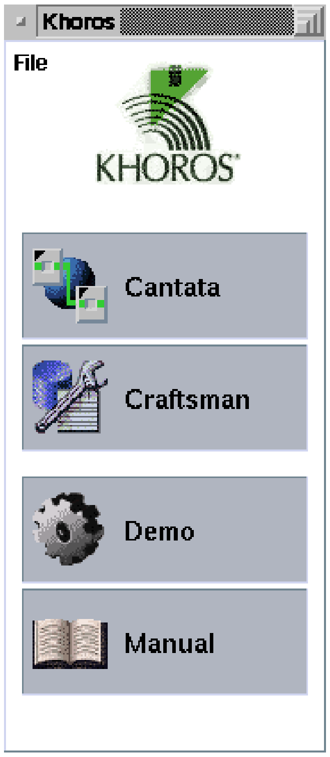
```
%composer -tb tests -oname test1
```

¿From `VisiQuest` it is also possible to invoke the `craftsman` and `composer` software development tools.

The VisiQuest Toolbar

Both `VisiQuest` and `craftsman` are accessible from the `VisiQuest` application, a new feature of VisiQuest. The `VisiQuest` application also offers access to the online VisiQuest manual, as well as the standard VisiQuest demos. To execute the `VisiQuest` application, type:

```
%VisiQuest
```



Selecting the $\boxed{\text{VisiQuest}}$ button displays `VisiQuest` , and selecting the $\boxed{\text{Craftsman}}$ button displays `craftsman` . A $\boxed{\text{Demo}}$ button provides access to the standard VisiQuest demos.

VisiQuest Version

To check the version of VisiQuest you are executing when running the `VisiQuest` application, select $\boxed{\text{Version Information}}$ from the *File* menu of the `VisiQuest` application.

VisiQuest Manuals

To access the online VisiQuest manuals, click on the $\boxed{\text{Manual}}$ button of the `VisiQuest`application. Note that the DOCUMENTATION toolbox contains the manual files and must be installed for this option to work.

Interfaces to VisiQuest Operators

VisiQuest operators can be executed through three different types of user interfaces:

1. CLUI, the Command Line User Interface,
2. GUI, the Graphical User Interface, and
3. `VisiQuest`, the Visual Programming Environment.

2.1 Command Line User Interface

To understand the structure of VisiQuest and the relationship among the three types of user interfaces, it is best to start with the Command Line User Interface (CLUI). Follow the instructions below to familiarize yourself with VisiQuest.

VisiQuest version

To check the version of VisiQuest you are executing, issue the following command at the UNIX prompt line:

```
%kversion

VisiQuest version 3 3.2.0.0 (Mar 31, 2002)
```

Note: If the `kversion` command is not found, there may be a problem with your PATH environment variable.

Listing toolboxes

The VisiQuest software system is composed of several *toolboxes*. A toolbox is a collection of programs and libraries managed as a single entity, called objects. The organization of the system in toolboxes greatly improves the modularity of the system and encourages contribution from the user community.

To list the toolboxes included in the standard system, type in the following command:

```
%kecho -echo toolboxes
AUDIO BOOTSTRAP DATASERV DESIGN IMAGINE DATAMANIP ENVISION
GEOMETRY IMAGE ..,
```

Note: If an error message is received, there may be a problem with the file *.kri/KP2001/Toolboxes* in your directory.

Toolbox as a collection of objects

Typically, a toolbox contains programs and libraries which have a similar function or common objective. The toolbox DESIGN, for instance, contains libraries that comprise GUI and visualization services, and the `VisiQuest`Toolbar. To list the objects contained in the toolbox DESIGN, type

```
%kecho -tb design -echo objects

xvwidgets xvobjects VisiQuest xvutils xvforms khelp xvrun
```

If you know the object, you can also search for its toolbox. For instance, to find which toolbox the object `kadd` is associated with, type

```
%kfindobj -oname kadd

  Toolbox    Object         Path
 ----------------------------------------
 DATAMANIP    kadd    objects/pane/kadd
```

You have verified that the `kadd` object is in the DATAMANIP toolbox.

CLUI syntax

As you can see above, all the operators in the CLUI interface have a consistent format. The program name is typed in, followed by as many arguments as necessary. The arguments can appear in any order and they are identified by an associated *-variable* tag.

Object usage

The options `-U` and `-usage` can be used with any command to describe its command line syntax. The `-usage` option gives the full list of arguments, including standard VisiQuest arguments, while the `-U` option gives only the arguments that are specific to that command. The option `-U` can be used with any command to describe its command line syntax:

```
%kversion -U

Usage for kversion:  This script will print out the version
of VisiQuest

 [-tb]         (string) toolbox name
               (default = none)
 [-oname]      (string) Program or Library Object Name
               (default = none)
 [-date]       (flag) Display the date only
               ignored if -tb and/or -oname specified
 [-name]       (flag) Display the name only
               ignored if -tb and/or -oname specified
 [-version]    (flag) Display the numerical version only
               ignored if -tb and/or -oname specified
```

Object information

Once you know to which toolbox an object belongs, you can find relevant information associated with it by using `kecho`. To show the information of the object `kadd` in the toolbox DATAMANIP, execute the following command:

```
%kecho -echo object-info -oname kadd -tb datamanip

begin DATAMANIP::kadd
binary-name=
category=Arithmetic
subcategory=Two Operand Arithmetic
short-description=Output = Input1+(Input2 or Constant)
icon-name=Add
in-VisiQuest=YES
end DATAMANIP::kadd
```

From this output, it is installed in `VisiQuest` under the icon name `Add`, and found under the category *Arithmetic*, sub-category *Two Operand Arithmetic*. The short description briefly explains the operation of the object.

Man pages

For a full description of an object, access its man pages. For instance, to know more about the `kecho` command, execute

```
%kman kecho
```

The CLUI interface has many more properties and rules. The full description is found in the Introduction, section E, of the VisiQuest End Users Guide. The End User Guide, along with the other VisiQuest manuals, are available online from the `VisiQuest` application. Start the `VisiQuest` application by typing

```
%VisiQuest
```

then click on the $\boxed{\text{Manual}}$ button to access the online manuals.

Aliases

Aliases in VisiQuest provide a convenient shorthand for long pathnames to input files without having to remember the location of files. It also makes programs in VisiQuest (either visual programs or scripts) more portable across installations, since the alias mechanism determines the exact location of the data files.

Aliases are specified by words separated by colons. In the example below, a data file stored in the DATASERV toolbox is copied to your directory under the name `gull.kdf` by using the VisiQuest copy command `kcp`. The alias `image:gull` corresponds to the path where the data file is actually located.

```
%kcp -i image:gull -o gull.kdf
```

Creating aliases

Each toolbox in VisiQuest can define its own aliases. The file where the aliases definitions are stored is located in each toolbox (`toolbox-path/repos/Aliases`). Each line defines an alias. Below is an excerpt of the line of the file `.../VisiQuestbase/dataserv/repos/Aliases` where the `image:gull` is declared.

```
image:gull $DATASERV/data/images/gull.kdf.gz
```

The aliases mechanism is available both in CLUI and GUI interfaces. You can learn more about the aliases syntax and capabilities in the Introduction, section G.3 of the VisiQuest Users' Guide.

2.2 Graphical User Interface

While the Command Line User Interface (CLUI) is sufficient to execute any VisiQuest program, a Graphical User Interface is also available for every program. There is an exact correspondence between the CLUI and GUI interfaces. An advantage of using GUI interfaces is that you do not need to remember all the options that each object provides. Another advantage is that you can access it from the `VisiQuest` visual programming environment.

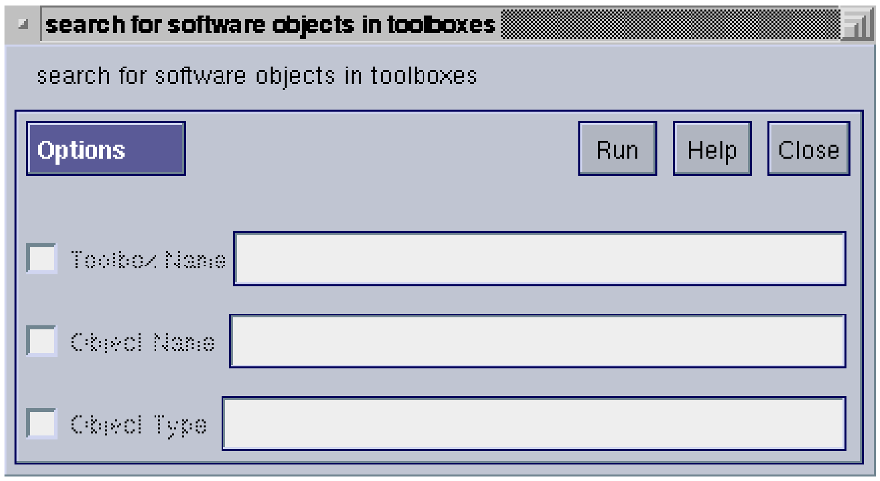
To better illustrate the equivalence between the CLUI and GUI interfaces, you will work with the `kfindobj` object. From its usage output shown below, you can see that `kfindobj` has three optional parameters: `-tb`, `-oname`, and `-type`.

```
%kfindobj -U

Usage for kfindobj:  search for software objects in toolboxes

% kfindobj
 [-tb]      (string) Toolbox to search
            (default = none)
 [-oname]   (string) Software object name
            (default = none)
 [-type]    (string) Software object type
            (default = none)
```

To invoke the GUI of an object, execute the object with the `-gui` option.

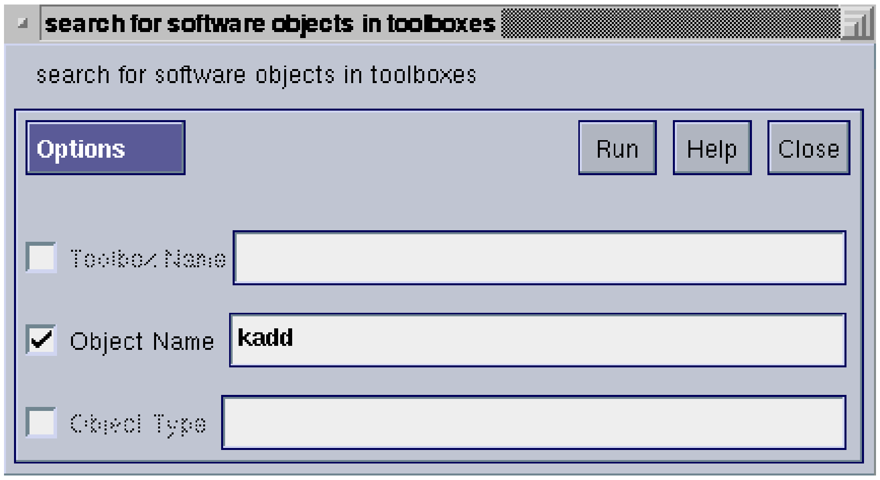
```
%kfindobj -gui
```

This displays the *pane* of the operator `kfindobj`, with its associated parameters. The pane is the GUI representation of an object.


search for software objects in toolboxes
search for software objects in toolboxes
Options
Run
Help
Close
Toolbox Name
Object Name
Object Type

There are three parameters in the `kfindobj` pane: *Toolbox Name*, *Object Name* and *Object Type*. Each is related to one of the three optional parameters in the CLUI interface. When a parameter is optional, there is a button on its left to activate or deactivate it. In the case of `kfindobj` alone, the three parameters are deactivated.

Activate the *Object Name* parameter and type in `kadd`. To type in a parameter, the pane window must be in focus and the mouse pointer must be inside the box when the parameter is entered.


search for software objects in toolboxes
search for software objects in toolboxes
Options
Run
Help
Close

Toolbox Name [ ]

Object Name [✓] kadd

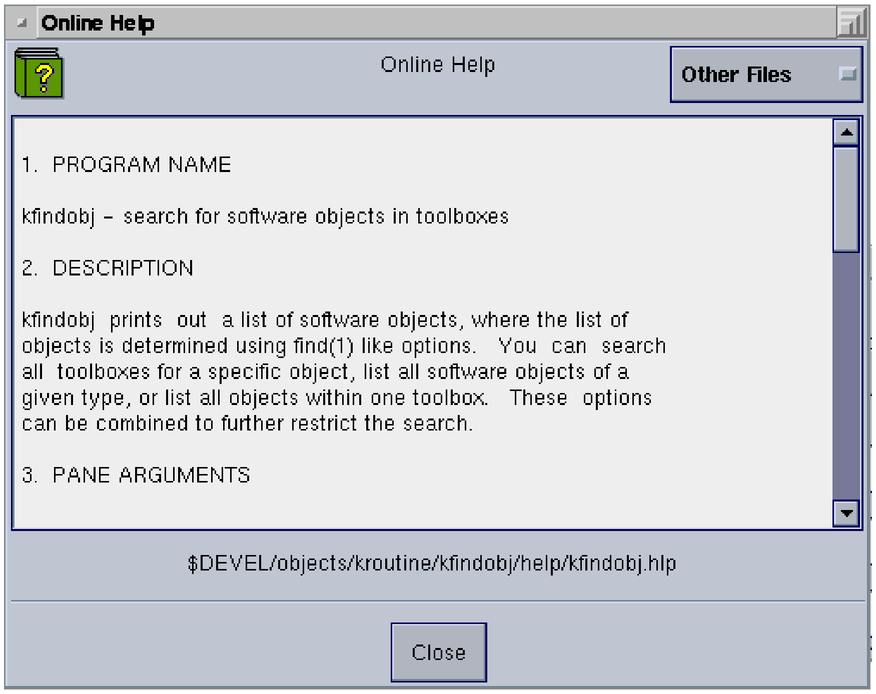
Object Type [ ]

To execute the object, click on the Run button in the pane and observe its output in the UNIX prompt line. The CLUI equivalent command is shown followed by the output of the command execution. Note the GUI and CLUI equivalence. The *Object Name* parameter is associated with the option `-oname` in the CLUI interface.

```
$DEVELBIN/kfindobj -oname "kadd"

  Toolbox    Object          Path
 DATAMANIP    kadd    objects/pane/kadd
```

On-line help

There is an on-line help available in every GUI interface. To invoke the on-line help, click on the Help button of the pane object.



1. PROGRAM NAME

kfindobj - search for software objects in toolboxes

2. DESCRIPTION

kfindobj prints out a list of software objects, where the list of objects is determined using find(1) like options. You can search all toolboxes for a specific object, list all software objects of a given type, or list all objects within one toolbox. These options can be combined to further restrict the search.

3. PANE ARGUMENTS

$DEVEL/objects/kroutine/kfindobj/help/kfindobj.hlp

Close the on-line help window and the `kfindobj` pane by clicking on their Close buttons.

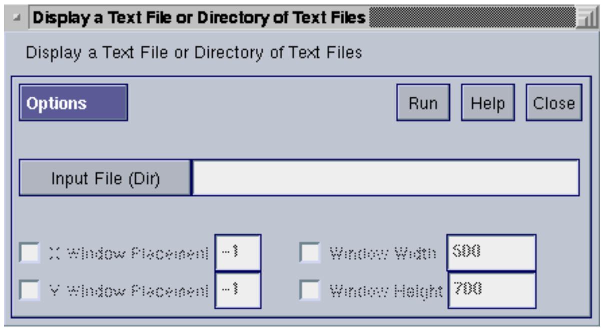
Directory Browser

VisiQuest provides the *Browser* tool as an alternative to typing in the filename specification. Any input or output file parameter in a GUI pane has a button to invoke the browser tool.

To experiment with this tool, use the `khelp` command, which displays on-line help or ASCII files.

1. Type in

```
%khelp -gui
```



This command requires an ASCII file as input. You can type the filename directly into the *Input File* parameter field or you can invoke the browser tool.

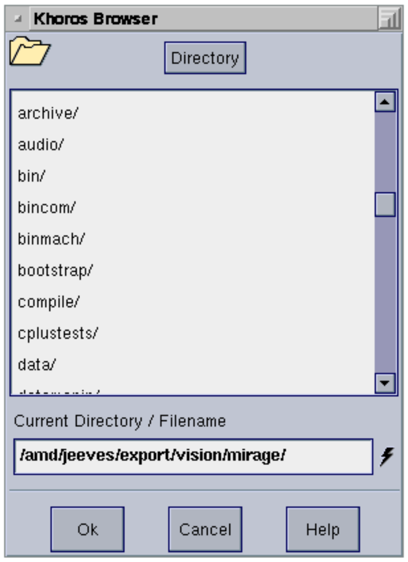
2. Invoke the browser tool by clicking on the $\boxed{\text{Input File (Dir)}}$ button.

   This displays the *Directory Browser* tool listing all the files and subdirectories found in the current directory. By double-clicking on a directory name, or by typing the appropriate path in the Directory/Filename box you can traverse directory structures and select filenames.

   The browser tool also works with the alias functionality.

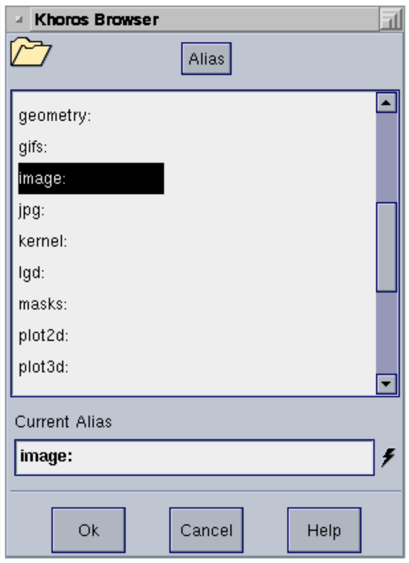
3. Click on the $\boxed{\text{Directory}}$ browser button or on the folder icon to toggle between the *Directory Browser* and the *Aliases Browser*.

4. Select `kernel:` by clicking on it, followed by a click on the $\boxed{\text{Ok}}$ button, or by double-clicking on it. Then scroll down the list, select `sobel_y`, and click on the $\boxed{\text{Ok}}$ button.



archive/
audio/
bin/
bincom/
binmach/
bootstrap/
compile/
cplustests/
data/

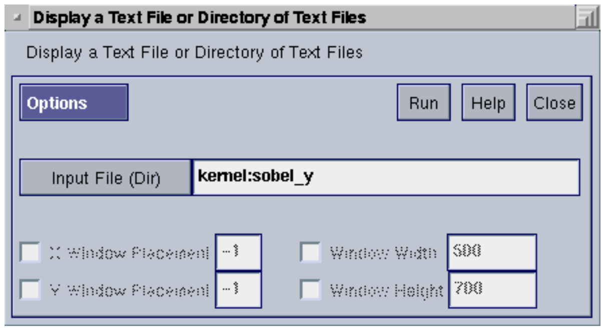
Current Directory / Filename
/amd/jeeves/export/vision/mirage/



geometry:
gifs:
image:
jpg:
kernel:
lgd:
masks:
plot2d:
plot3d:

Current Alias
image:

The Aliases Browser closes and the selected filename appears in the file-name parameter field.

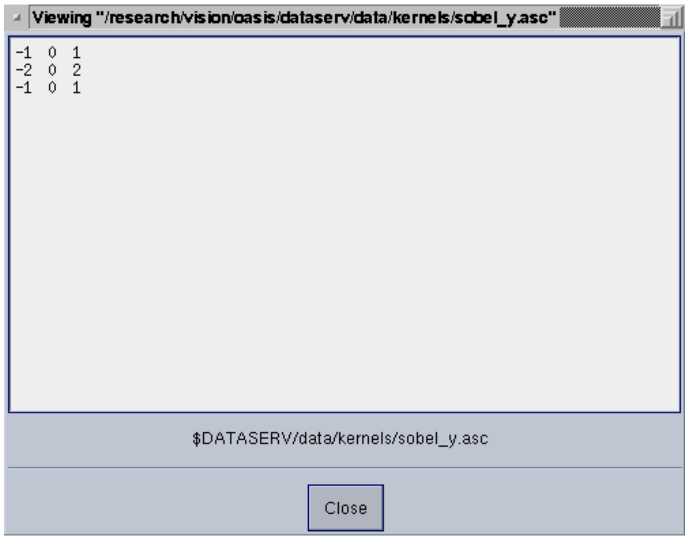


Display a Text File or Directory of Text Files

Input File (Dir) kernel:sobel_y

X Window Placement -1
Y Window Placement -1
Window Width 500
Window Height 700

5. Click on the Run button to execute the **khelp** command to view the ASCII file `kernel:sobel_y` in the on-line text viewer.


Viewing "/research/vision/oasis/dataserv/data/kernels/sobel_y.asc"

```
-1  0  1
-2  0  2
-1  0  1
```

$DATASERV/data/kernels/sobel_y.asc

6. Close the text viewer and close the `khelp` pane.

Before introducing `VisiQuest`, the third and most widely used VisiQuest interface, it is important to understand the VisiQuest data model because most operators process the data stored in this model. The data model is used for storing, processing, and visualizing information. We will discuss the data model using the CLUI and GUI interfaces. Later, the same example will be carried out with `VisiQuest` to see the advantages of visual programming.

2.3 Data Model

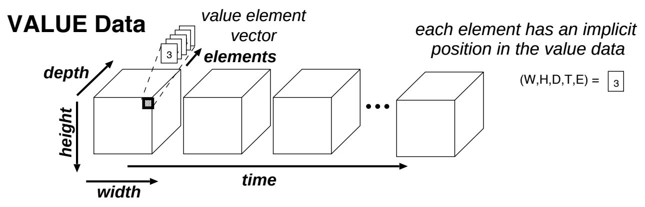
A good understanding of the internal data structures used in a software environment is necessary to process and display data. The VisiQuest system was designed to manipulate multidimensional data, using a *Polymorphic Data Model*.

The polymorphic model consists of five data segments: *value*, *location*, *time*, *mask*, and *map*.

To set the groundwork for the discussion of `VisiQuest`, this section will explore the value segment of the Polymorphic Data Model. The full discussion of this important concept is in Chapter 4.

The value segment: signals, images, and animations

The value segment can store data in five dimensions: *width*, *height*, *depth*, *time*, and *element*.



Data file header information

In data processing, it is critical to understand the nature of the data file being processed. Basic information concerning the size, the data type, the dimensionality of the data, data segments associated with the data, and file format must be known before processing the data.

The routine `kfileinfo` allows you to find the data information stored with the data on a disk file. Accessing `kfileinfo` gives you file header information. Using the example of the data file `gull.kdf`, which is aliased to `image:gull`, execute

```
%kfileinfo -i image:gull

# Name:  image:gull
# Comment:  # End Comment # Machine Architecture:  Big Endian
IEEE
# Date Created:  Tue Aug 09 17:15:37 1994
# Storage Format:  kdf
# Format Description:  VisiQuest 2.0 Data Format (kdf)
# Sub-Object Position:  0 , 0 , 0 , 0 , 0
# World Coordinate Point Size:  1 , 1 , 1 , 1 , 1
#
# Color Space Model:  0 (invalid)
# Has Alpha Channel:  0
#
# maskedValuePresentation:  0
#
# -- Value Data --
# Data Type:  Unsigned Byte
# Size:  Width=256, Height=256, Depth=1, Time=1, Elements=1
#
```

Interpreting kfileinfo

¿From the output produced by the operator `kfileinfo`, you obtain several pieces of information stored as attributes: the full path location of the data file in the system, the machine that created the file and its internal data storage type, date of creation, the file format, data segments associated with the data, and its dimensions. The VisiQuest data model supports 5 dimensions in the "Value Data" segment: *width*, *height*, *depth*, *time*, and *elements*.

The gull data file stores a two-dimensional (2D) signal or image with a *width* of 256 points, known as *pixels* (abbreviation for picture elements), and a *height* also of 256 pixels. The total number of points is then 65,536 (256 x 256) of data type *Unsigned Byte*, as reported in the output under the "Value Data" attribute depicted below.

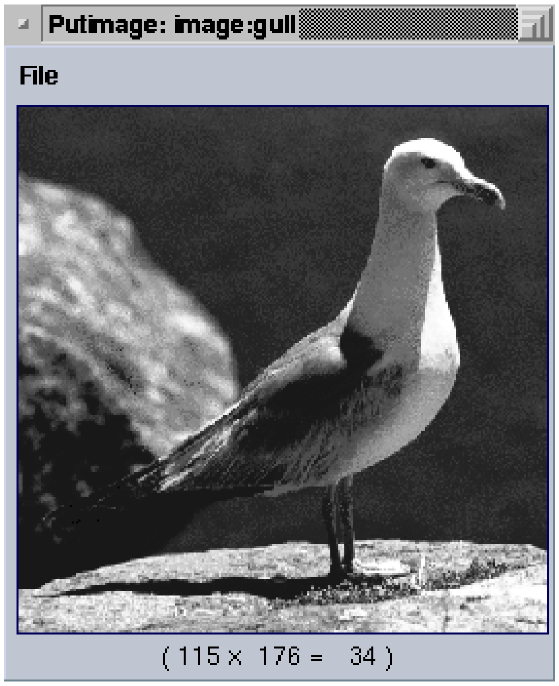
```
# -- Value Data --
# Data Type:  Unsigned Byte
# Size:  Width=256, Height=256, Depth=1, Time=1, Elements=1
```

Digital image

A digital image is a two-dimensional matrix of pixels. The process of displaying an image creates a graphical representation of this matrix where the pixel values are assigned a particular grey-level (monochromatic image) or a particular color.

You may display this image on your screen by using the `putimage` operator.

```
%putimage -i image:gull
```



Coordinates

Above is a graphical representation of the image produced by the `putimage` operator available in VisiQuest. Pixels with low values are assigned dark grey-levels, and high pixel values are assigned bright grey-levels. When you point the cursor at a particular pixel, the small position window below the image gives the actual pixel value and its coordinates. The top-left pixel is located at coordinate (0,0). The lower right-hand corner pixel has coordinates (255,255).

You can exit `putimage` by (1) selecting *Quit* from the *File* menu, by (2) double-clicking on the displayed image, or by (3) typing the <`q`> key when the displayed image is in focus.

Editimage

Another visualization operator, `editimage`, gives you full interactivity to explore and gain insight from the data. The steps below show how to use `editimage` and how to better understand the digital image representation by zooming to a small portion of the image, by displaying the pixel values, and by changing the color assigned to them.

Invoke `editimage` from the command line, without listing parameters.

```
%editimage
```

Now follow these steps:

1. Select *Open...* from the *Files* menu to access the *Input Files* pane.

2. Access the browser tool to read and display the `image:gull` data file. Click on the $\boxed{\text{Input Image}}$ button. When the browser pane is displayed, click on the $\boxed{\text{Directory}}$ button or folder icon. Use the scroll bar to find the `image` aliases directory and double-click on it. Use the scroll bar again to locate the data file `gull` and double-click on it. At this point the browser pane disappears. Note that in the $\boxed{\text{Input Image}}$ box the string `image:gull` is displayed. Alternatively, you could type in `image:gull` directly in the *Input Image* parameter.

3. Close the *Input Files* pane by clicking on the $\boxed{\text{Close}}$ button.

4. Select *Zoom...* from the *View* menu and place the zoom window on your screen. Note that as you move the mouse over the image displayed by `editimage`, a small portion of the image around the mouse is displayed in the zoom image window, revealing the discrete nature of the digital image.

5. Select *Pixel Values...* from the *View* menu to display the Pixel Values window. Similarly to the zoom window, this shows a zoom of the image but with the actual pixel values.

6. Select $\boxed{\text{Colormaps...}}$ from the $\boxed{\text{Select}}$ menu to access the $\boxed{\text{Colormaps}}$ subform. Use the scroll bar in the *Autocolor Procedures* table to access the colormap `SA Pseudo`. This changes the previous grey-scale colormap used to display the image to a new colormap, which is called a pseudo colormap. You can use a pseudo colormap to enhance finer details of the image that cannot be seen in the grey-scale colormap.

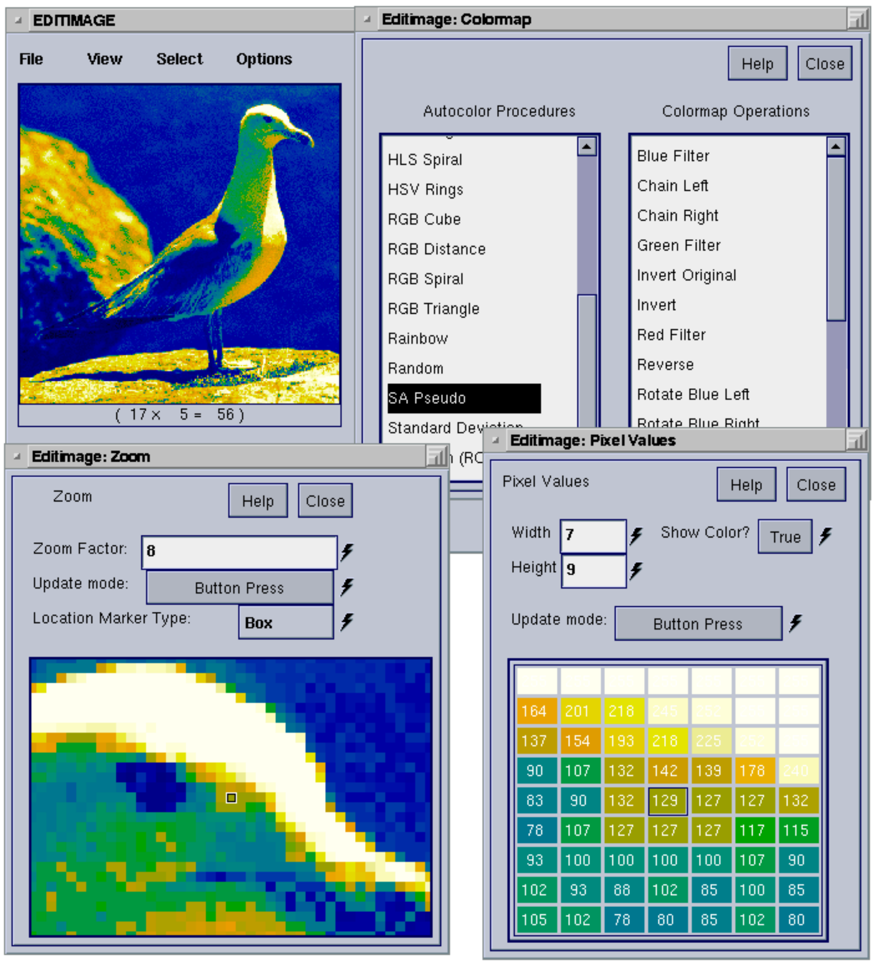
7. Use the mouse to navigate the `editimage` window and note that the pixel values have not changed by applying the pseudo colortable. The pseudo colortable affects only the color representation assigned to the pixel values.

8. Open the *Input Files* again and change the *Input Image* parameter to `image:head` and press the <`Enter`> key. The new image is displayed in the `editimage` windows. Verify that in this image, the pixel values are not restricted to the interval 0-255 as many other visualization systems are. You can change the *Input Image* to `image:ankle_MRI` image and verify that the pixel values in this image are complex.

   Note: It may take a while for the pixel values be updated.

9. Exit `editimage` by selecting *Quit* from the *File* menu. You will be prompted for a confirmation.

You can find detailed information regarding the functionality of `editimage` in Chapter 4 of the Envision Toolbox Manual, which is part of the VisiQuest Users' Guide.



Zoom Factor: 8

Update mode: Button Press

Location Marker Type: Box

Width 7

Height 9

Show Color? True

Update mode: Button Press

255	255	255	255	255	255	255
164	201	218	245	252	255	255
137	154	193	218	225	252	255
90	107	132	142	139	178	240
83	90	132	129	127	127	132
78	107	127	127	127	117	115
93	100	100	100	100	107	90
102	93	88	102	85	100	85
105	102	78	80	85	102	80

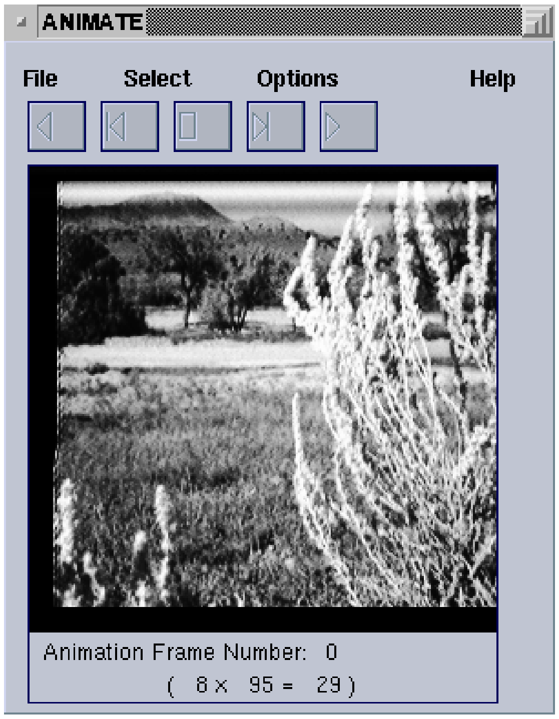
Visualizing an image sequence

The `animate` operator is an interactive image sequence display tool. This tool visualizes 3D or higher dimensional data by sequencing 2D frames. You can use it to visualize volume data where the depth dimension of the value segment is used, or to visualize a data set where the different frames are stored in the elements or time dimensions. Execute the `kfileinfo` to display the header information associated with the file `sequence:bushes`.

```
%kfileinfo -i sequence:bushes

# Name:  /home/VisiQuest/Datamanip/data/sequences/bushes.kdf
# Comment:
This is an image sequence provided by Sandia National Labs,
taken by G.Donohoe and E.Hoover in the Sandia Labs playground.
# End Comment
# Machine Architecture:  Big Endian IEEE
# Date Created:  Sat Aug 27 19:22:30 MDT 1994
# Storage Format:  kdf
# Sub-Object Position:  0 , 0 , 0 , 0 , 0
# World Coordinate Point Size:  1 , 1 , 1 , 1 , 1
#
# Color Space Model:  0 (invalid)
# Has Alpha Channel:  0
#
# -- Value Data --
# Data Type:  Unsigned Byte (4)
# Size:  Width=256, Height=256, Depth=1, Time=20, Elements=1
#
```

¿From the `kfileinfo`, you can observe that this data is stored as a sequence of 20 images in the time dimension of size 256 x 256 pixels. To move through the data, execute the `animate` operator by typing

```
%animate -i sequence:bushes
```



You can move through the frames forward or backward one frame at a time or automatically through all of them by using the five buttons above the image. Selecting *Attributes* from the *Options* menu, you can control the speed of the sequencing, changing the attribute *Animation Speed.*

Exit `animate` by selecting *Quit* from the *File* menu.

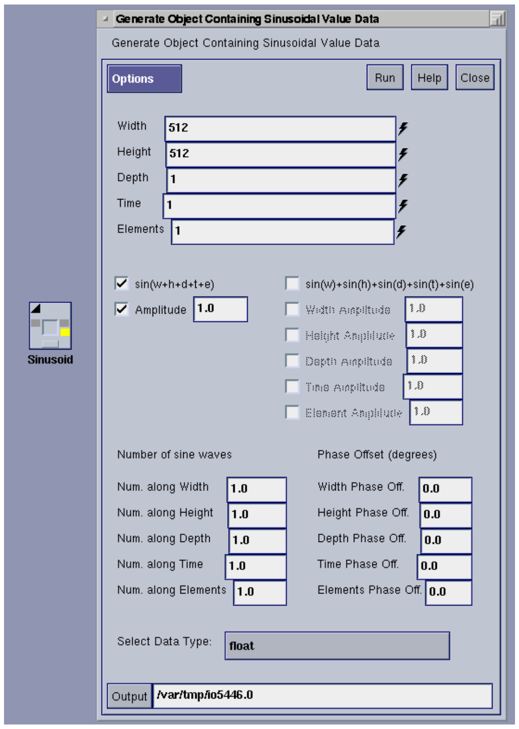
Generating 1D data

You used `putimage` and `editimage` to visualize 2D images and `animate` to visualize image sequences. You will now use the plotting facilities in VisiQuest to visualize 1D data.

First generate a sinusoidal signal of size 50 samples by using the `kgsin` operator.

1. Execute the following:

```
%kgsin -gui
```



Generate Object Containing Sinusoidal Value Data

Generate Object Containing Sinusoidal Value Data

Width 512
Height 512
Depth 1
Time 1
Elements 1

[✓] sin(w+h+d+t+e)
[✓] Amplitude 1.0

[ ] sin(w)+sin(h)+sin(d)+sin(t)+sin(e)
[ ] Width Amplitude 1.0
[ ] Height Amplitude 1.0
[ ] Depth Amplitude 1.0
[ ] Time Amplitude 1.0
[ ] Element Amplitude 1.0

Number of sine waves

Num. along Width 1.0
Num. along Height 1.0
Num. along Depth 1.0
Num. along Time 1.0
Num. along Elements 1.0

Phase Offset (degrees)

Width Phase Off. 0.0
Height Phase Off. 0.0
Depth Phase Off. 0.0
Time Phase Off. 0.0
Elements Phase Off. 0.0

Select Data Type: float

Output /var/tmp/io5446.0

2. In the pane of the `kgsin` command, set the *Width* parameter to 50 and the *Height* to 1.

   This will generate a single sinusoidal wave with 50 floating point samples in the *width* dimension with an amplitude of 1.0.

3. Type in the filename `sine50w.kdf` in the *Output* parameter.

   The kdf extension is a convention used for the VisiQuest Data Format (KDF) files.

4. Click on the $\boxed{\text{Run}}$ button to execute the command and then on $\boxed{\text{Close}}$ to close the pane.

   Verify that the generated file was created with the proper size by using `kfileinfo`. The *width* dimension should be 50 and all other dimensions should be 1. Verify that the *data type* is *Float*.

5. Execute

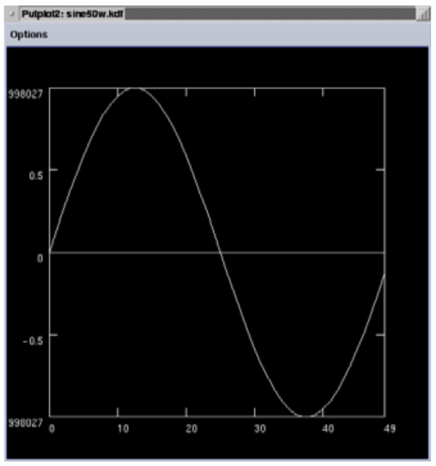
```
%kfileinfo -i sine50w.kdf

# Name:  ../experiments/sine50w.kdf
# Storage Format:  kdf
#
# -- Value Data --
# Data Type:  Float
# Size:  Width=50, Height=1, Depth=1, Time=1, Elements=1
```

Putplot2

Use the `putplot2` operator to plot the sinusoidal data.

6. Execute

```
%putplot2 -i sine50w.kdf
```



7. Select *Plot Options...* from the *Options* menu to access the parameters of this visualization operator.

8. Change the *Plot Type* pulldown menu to `Line Marker` to reveal the discrete nature of the data.

9. Change the *Color Origination* pulldown menu to `Use Data Values`. This assigns a different color to different sample values.

10. Exit `putplot2` by selecting *Quit* from the *Options* menu.

In this section you learned how data is stored in a file. With the introductory knowledge of VisiQuest' CLUI and GUI interfaces, you are ready to use `VisiQuest` and to understand its connection with GUI and CLUI interfaces. In the next section, you will repeat most of the experiments used in this chapter, only using the `VisiQuest` interface.

2.4 VisiQuest

The third and most important user interface in VisiQuest is the visual programming environment called `VisiQuest`. For many VisiQuest users, `VisiQuest` is considered the *showcase* of the system, since you can access virtually all the resources of the system. Everything you have learned so far is accessible via `VisiQuest`, so indirectly you have studied the `VisiQuest` components.

Visual language programming

`VisiQuest` is a graphically expressed, data flow visual language. In `VisiQuest`, a program is described as a direct graph, where the operators are represented by the nodes and the data are represented by the arcs. A visual language provides a visual programming environment for quick prototyping, testing, and creation of new operators.

VisiQuest versus CLUI/GUI

We will reexamine the sinusoidal generation and plotting example using `VisiQuest`. To illustrate how `VisiQuest` increases the productivity of scientists, engineers, and application developers. By providing a natural environment which resembles the block diagrams that are already familiar to practitioners in the field, the visual language supports both novice and experienced programmers.

Invoking VisiQuest

1. There are two ways to execute `VisiQuest`. The first is to use the `VisiQuest` application. Display the `VisiQuest` application by executing

   ```
   %VisiQuest &
   ```

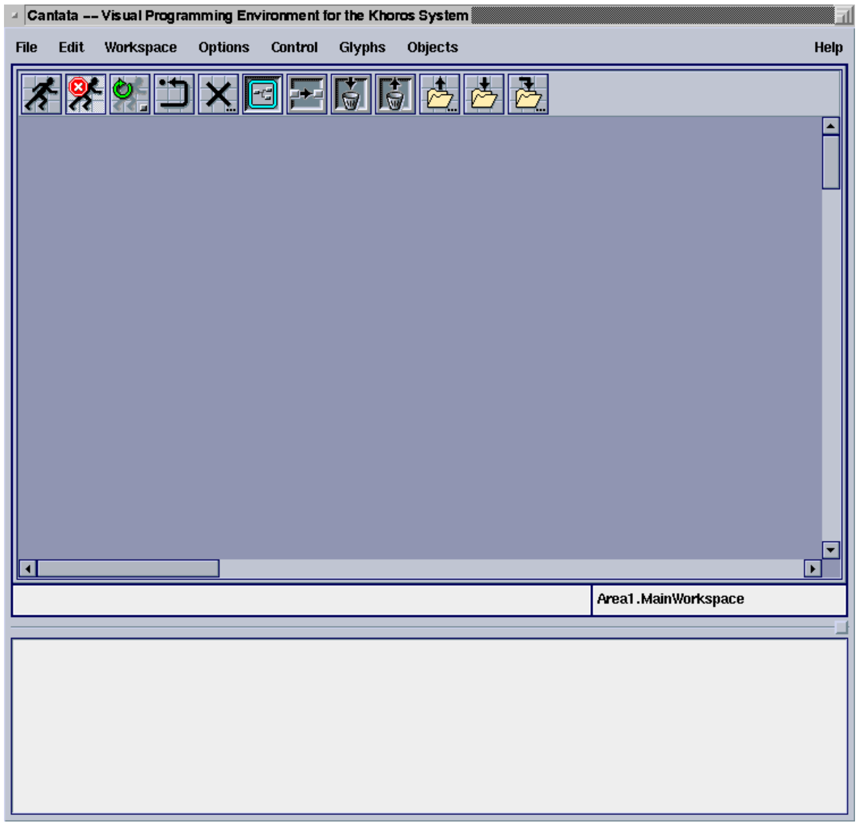
   Now, bring up `VisiQuest` by selecting $\boxed{\text{VisiQuest}}$ from the `VisiQuest` interface.

The second way to display VisiQuest is to execute `VisiQuest` it directly from the command line by typing

```
%VisiQuest &
```

The "&" symbol at the end of the UNIX command line puts `VisiQuest` in the background execution mode. In this mode, the command line is not locked and you can invoke other commands. It is convenient to run `VisiQuest` in background mode since it will run while you are performing other tasks.

Once `VisiQuest` is running, the window of a visual programming *workspace*, made up of a viewport containing a canvas, is displayed.

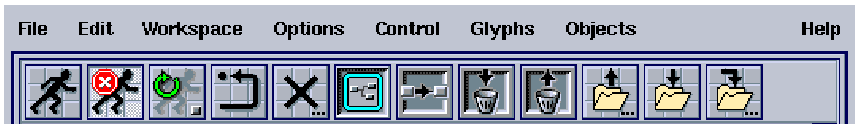


VisiQuest menus and Command Bar

`VisiQuest` has seven pulldown menus through which the user can access a variety of programs and utilities. The menus are *File*, *Edit*, *Workspace*, *Options*, *Control*, *Glyphs*, *Objects*, and *Help*.

In addition, there is a workspace *command bar* which contains icons representing the most commonly used commands from the `VisiQuest` inventory. The icons are displayed just below the main `VisiQuest` menu bar.

Note: The set of icons is variable: you can choose to display only those you wish or you can make all of them disappear using the *Preferences* subform of the *Options* menu.

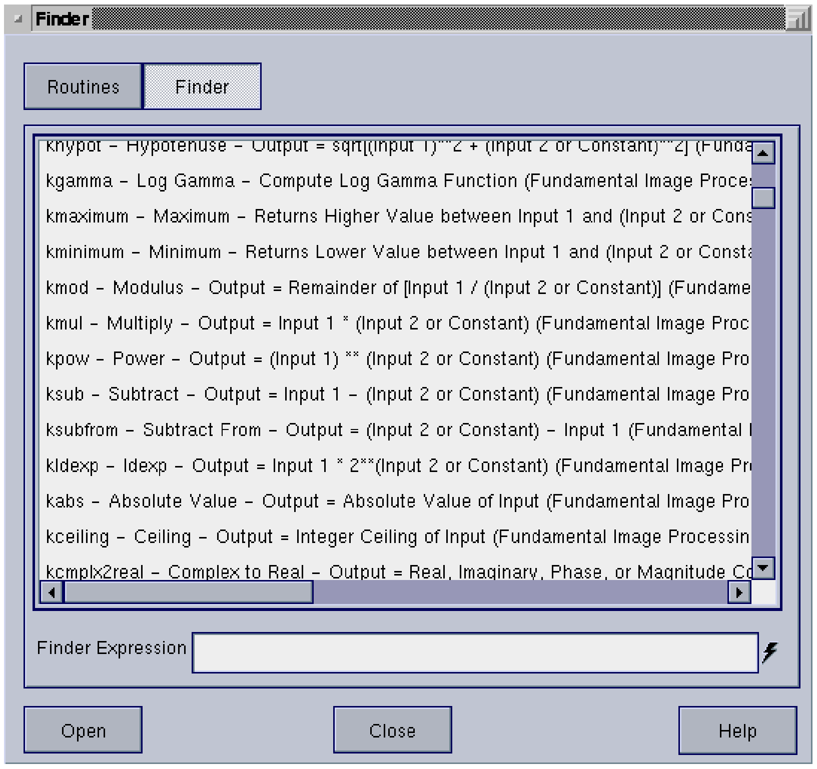


Locating glyphs with Finder

Each operator in `VisiQuest` has a graphical representation called a *glyph*. All CLUI objects that can be accessed via `VisiQuest` have a glyph associated with them. A visual program consists of a number of glyphs placed in a workspace and connected together, forming a network.

The glyphs can be brought into the visual programming workspace via the *Glyphs* menu or the *Finder* tool. First, we will use the *Finder* tool to access the glyphs, followed by a discussion of the *Glyphs* menu.

2. Open the *Finder* tool by clicking on the $\boxed{\text{Edit}}$ pulldown menu with the left mouse button and drag it to the very end where it reads *Find...*. Place the *Finder* pane on your screen and in the *Finder Expression* parameter box, type in a keyword that may be related to the operator you wish to locate.



```
khypot - Hypotenuse - Output = sqrt[(Input 1)**2 + (Input 2 or Constant)**2] (Funda
kgamma - Log Gamma - Compute Log Gamma Function (Fundamental Image Proce
kmaximum - Maximum - Returns Higher Value between Input 1 and (Input 2 or Cons
kminimum - Minimum - Returns Lower Value between Input 1 and (Input 2 or Consta
kmod - Modulus - Output = Remainder of [Input 1 / (Input 2 or Constant)] (Fundame
kmul - Multiply - Output = Input 1 * (Input 2 or Constant) (Fundamental Image Proc
kpow - Power - Output = (Input 1) ** (Input 2 or Constant) (Fundamental Image Pro
ksub - Subtract - Output = Input 1 - (Input 2 or Constant) (Fundamental Image Pro
ksubfrom - Subtract From - Output = (Input 2 or Constant) - Input 1 (Fundamental I
kldexp - ldexp - Output = Input 1 * 2**(Input 2 or Constant) (Fundamental Image Pr
kabs - Absolute Value - Output = Absolute Value of Input (Fundamental Image Pro
kceiling - Ceiling - Output = Integer Ceiling of Input (Fundamental Image Processin
kcmplx2real - Complex to Real - Output = Real, Imaginary, Phase, or Magnitude Co
```

Finding kgsin

3. Find the operator `kgsin` by typing the keyword `sinusoid` followed by the <`Enter`> key.

   This produces an output which corresponds to the routines that deal with sinusoids. Note that by moving the scrollbar at the bottom of the window to the right, the Category, Subcategory, and Name of the `kgsin` operator appear in parentheses "(Input/Output: Generate Data: Sinusoid)". These can be used to find the `kgsin` operator in the Glyph menus.

4. Select the desired routine either by (1) clicking on its line description with the left mouse button and then clicking on the $\boxed{\text{Open}}$ button, or (2) by double-clicking on its line description with the left mouse button. Move the cursor to the `VisiQuest` workspace and place the glyph on it.

5. Close the Finder pane.

Locating glyphs with the Glyph Menus

Glyphs can also be created using the Glyph Menus. Here, we know the category, subcategory, and name of the glyph we are looking for.

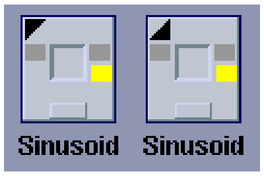
6. Select *Input/Output* from the Glyphs menu.
7. Holding the button down, move the mouse to the right of *Input/Output* and select *Generate Data* .
8. Holding the button down, move the mouse to the right of *Generate Data* and select *Sinusoid.*
9. Finally releasing the mouse, move the cursor into the `VisiQuest` workspace and place the glyph on it.

Moving the Sinusoid glyph

The `Sinusoid` glyph is the `VisiQuest` interface to the object `kgsin`. You can move the glyph around the workspace by clicking on it with the left mouse button, and while keeping the mouse button down, drag the glyph to the desired location. Releasing the mouse button leaves the glyph in the new position.

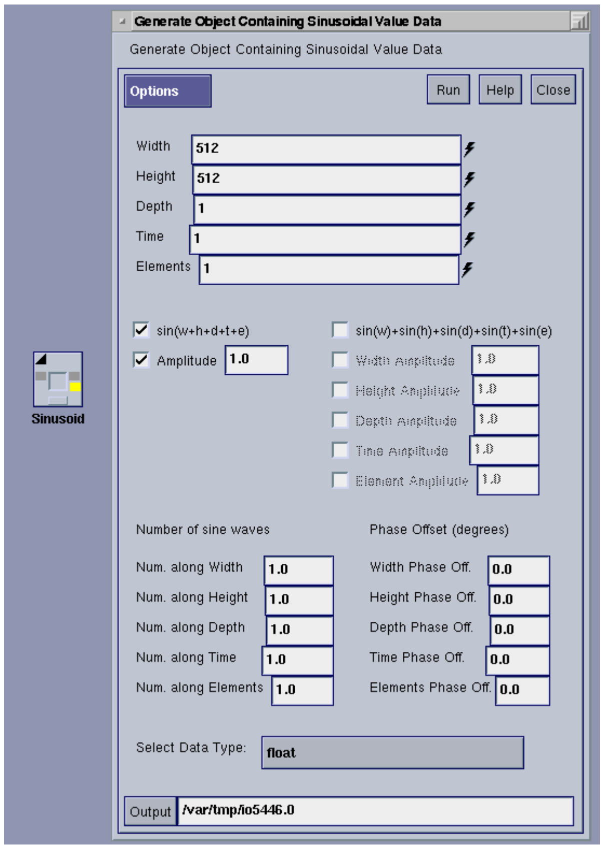
Glyph pane access

10. Open the pane associated with the glyph by clicking on its upper left-hand corner. Observe the change of the direction of the pane access icon in the glyph. Its state indicates if the pane is open or closed.



Notice that the pane you opened is exactly the GUI interface of the `kgsin` operator used before.

11. In the same manner as before, modify the number of samples along the *Width* dimension to `50` and along the *Height* dimension to `1`.
12. Close the pane by clicking on the Close button or by clicking again on the left-hand corner of the glyph.



Generate Object Containing Sinusoidal Value Data

Generate Object Containing Sinusoidal Value Data

Options | Run | Help | Close

Width	512
Height	512
Depth	1
Time	1
Elements	1

[✓] sin(w+h+d+t+e)

[✓] Amplitude 1.0

[ ] sin(w)+sin(h)+sin(d)+sin(t)+sin(e)

[ ] Width Amplitude	1.0
[ ] Height Amplitude	1.0
[ ] Depth Amplitude	1.0
[ ] Time Amplitude	1.0
[ ] Element Amplitude	1.0

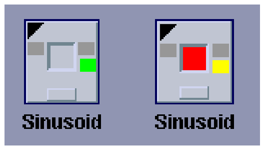
Number of sine waves		Phase Offset (degrees)	
Num. along Width	1.0	Width Phase Off.	0.0
Num. along Height	1.0	Height Phase Off.	0.0
Num. along Depth	1.0	Depth Phase Off.	0.0
Num. along Time	1.0	Time Phase Off.	0.0
Num. along Elements	1.0	Elements Phase Off.	0.0

Select Data Type: float

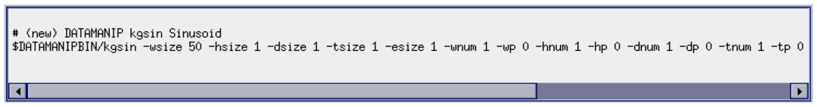
Output /var/tmp/io5446.0

How a glyph executes

13. Execute the operator by clicking on the middle square of the glyph. You will notice that it changes to red when it is executing.

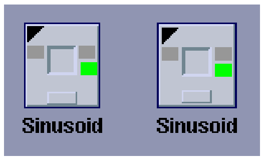


When the glyph is executing, its equivalent CLUI command is displayed in the console area of `VisiQuest` (its lower window).



```
# (new) DATAMANIP kgsin Sinusoid
$DATAMANIPBIN/kgsin -wsize 50 -hsize 1 -dsize 1 -tsize 1 -esize 1 -wnum 1 -wp 0 -hnum 1 -hp 0 -dnum 1 -dp 0 -tnum 1 -tp 0
```

After execution, the glyph changes back to grey and the small colored box on the right of the glyph changes from yellow to green.

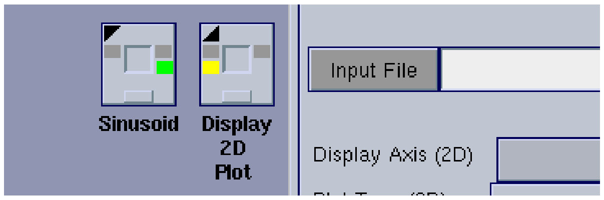


This small colored box is called an *output port*. It represents the output file parameter of the `Sinusoid` glyph. The color of the port indicates the availability of the data associated with it. After execution, it turns to green to indicate that the output file is available.

Connecting two glyphs

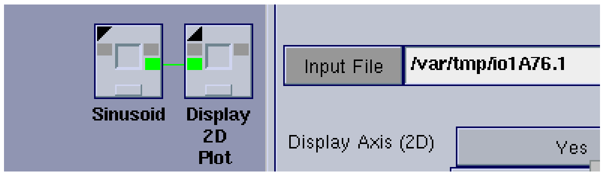
14. Locate the `putplot2` operator by using the *Finder* tool. Follow the same steps used to locate the `Sinusoid` glyph. The `VisiQuest` interface to the `putplot2` operator is the `Display 2D Plot` glyph. Place the glyph to the right of the `Sinusoid` glyph.

15. Open the pane of the `Display 2D Plot` glyph and place it to the right of its glyph. Observe that the parameter *Input File* needs to be filled out with the name of the file generated by the `Sinusoid` glyph. This will be done automatically when you connect the glyphs.



16. Connect the output of the `Sinusoid` glyph to the input of the `Display 2D Plot` glyph by clicking on the output box of the `Sinusoid` glyph, followed by another click on the yellow input box of the `Display 2D Plot` glyph.

    As soon as you do this, the filename automatically appears in the *Input File* box of the *2D Plot* pane. Note that the color of the connection is green, indicating that data is available. The name of the file is generated randomly and it is stored in the *temporary* directory indicated by the environment variable TMPDIR.



Input File /var/tmp/io1A76.1
Display Axis (2D) Yes

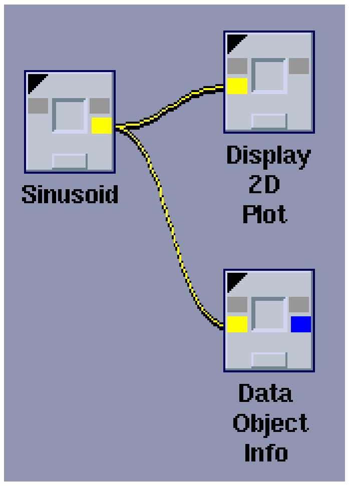
Running 2D Plot inside VisiQuest

17. Close the pane of the `Display 2D Plot` operator and execute its glyph by clicking on the center box. The result of the execution is the plot window of the sinusoidal signal generated by the `Sinusoidal` glyph, exactly as before when you were using the CLUI and GUI interface.

18. You can stop the `Display 2D Plot` glyph either by clicking on the red center box of the glyph or by clicking on the $\boxed{\text{Quit}}$ button of the plot window.

Data Object Info

19. Locate `kfileinfo` by using the *Finder* tool. The `VisiQuest` interface to this operator is the `Data Object Info` glyph. Place the glyph below the `Display 2D Plot` glyph and connect its input to the output of the `Sinusoid` glyph by clicking on their input/output colored boxes.

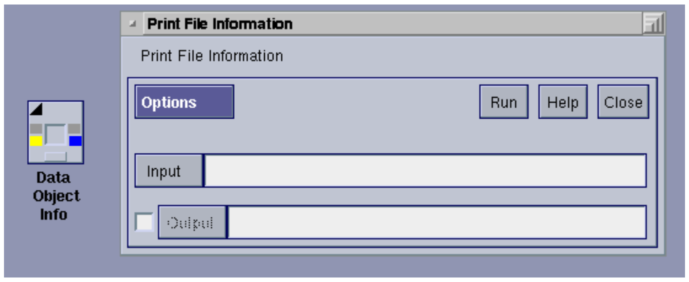


Note: The style of the glyph connections can be changed to spline as in the figure above by accessing the *Preferences* form located in the `VisiQuest` *Options* pulldown menu. Click on the Glyphs button preferences and select `spline` in the *Set Connection Type To* field.

Optional input/output parameter

20. Open the pane of the `Data Object Info` glyph and note that it has two parameters, one *Input* and one *Output*, which correspond to the input and output connections in the glyph. Observe that the *Output* parameter has a grey box associated with it. This indicates an optional parameter. If this output parameter is enabled, the ASCII file generated by this operator is written to it. If it is disabled, the file is sent to the standard UNIX output *stdout*.

21. If the Output parameter is enabled, disable it by clicking on the associated grey box and close the pane by clicking on the Close button. Observe that the output box connection changed its color to blue, indicating that the output is optional and it is not enabled.

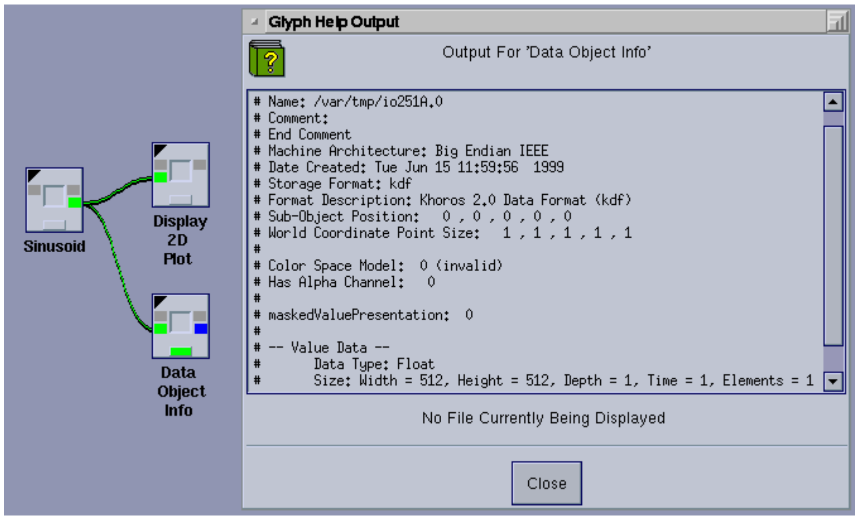


Information/error dialog boxes

22. Execute the `Data Object Info` glyph.

    The result of this operator is sent to the standard output, indicated by the *Console Button*. The console button appears at the bottom of the glyph. When it is lit up in green, this indicates that the operator associated with the glyph has sent a message to the standard output (*stdout*).

23. Click on the *Console Button* to see the message.


Sinusoid
Display 2D Plot
Data Object Info
Glyph Help Output
Output For 'Data Object Info'
No File Currently Being Displayed
Close

```
# Name: /var/tmp/io251A.0
# Comment:
# End Comment
# Machine Architecture: Big Endian IEEE
# Date Created: Tue Jun 15 11:59:56  1999
# Storage Format: kdf
# Format Description: Khoros 2.0 Data Format (kdf)
# Sub-Object Position:   0 , 0 , 0 , 0 , 0
# World Coordinate Point Size:   1 , 1 , 1 , 1 , 1
#
# Color Space Model:  0 (invalid)
# Has Alpha Channel:   0
#
# maskedValuePresentation:  0
#
# -- Value Data --
#       Data Type: Float
#       Size: Width = 512, Height = 512, Depth = 1, Time = 1, Elements = 1
```

If the console button lights up in red, it means that the glyph has sent an error message to the standard error (*stderr*). In this case, you would click on the same *Console Button* to see the error.

24. Close the text window.

Saving a workspace as a file

Congratulations, you have just created your first `VisiQuest` visual program that consists of a *network* of three glyphs.

25. Save this program into a file, by accessing the *Save File* option located on the *File* menu. A short-cut is available by typing <`Ctrl-S`>. Enter the filename `sine.wk` for the workspace file to be saved on disk, and click on the $\boxed{\text{Ok}}$ button. Typically, workspace filenames are given the extension "wk," but you are not restricted from creating your own convention.

Clearing and restoring a workspace

26. Clear the workspace by accessing this option in the *Workspace* menu. Click on the $\boxed{\text{Yes}}$ button to reconfirm your wish to clear the workspace area.

27. Now, to verify that you have saved the `sine.wk` workspace correctly, restore it by accessing the *Open File* option found in the *File* menu (short-cut <`Ctrl-O`>. This brings up the *File Browser* tool through which you locate the file `sine.wk` and load it back into the `VisiQuest` environment.

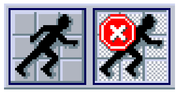
VisiQuest as an event-driven scheduler

Once the workspace is restored, run the entire visual program by accessing the *Run* option in the *Workspace* menu. Notice that each glyph is executed as soon as its input data is available. `VisiQuest` interprets the visual program and schedules glyphs, dispatching them as processes. It is for this reason that the program execution in `VisiQuest` is data-driven. Next, you will see that `VisiQuest` is actually an event-driven scheduler. It responds to parameter changes and supports mechanisms for glyph synchronization.

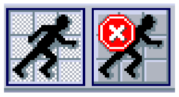
VisiQuest execution

`VisiQuest` can be in one of two modes of execution: stop or run. There are two buttons in the command bar which are short-cuts for the *Run* and *Stop* options in the *Workspace* menu. Together they display the two modes by which `VisiQuest` can schedule glyphs: stop and run. When you execute a workspace either by selecting the *Run* option or by clicking on the $\boxed{\text{Run}}$ icon button, `VisiQuest` enters in the run mode. When you stop the execution of the workspace by selecting the *Stop* option or by clicking on the $\boxed{\text{Stop}}$ icon button, `VisiQuest` enters the stop mode.

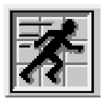
Stop mode This is the default `VisiQuest` execution mode. In this mode you can execute each glyph individually by clicking on the glyph middle square button. In the stop mode, and any changes to a glyph's parameters will not rerun the workspace. For the changes to take effect the workspace must be rerun. This status is shown in the $\boxed{\text{Run}}$ and $\boxed{\text{Stop}}$ icon buttons in the command bar.



Run mode In the run mode, `VisiQuest` executes all the glyphs in the network following a data-driven and an event-driven scheduler until they reach a stable condition. In this mode, `VisiQuest` is ready to schedule any glyph that receives an event. There are many events, such as individual execution of a glyph, a change on its input data or control connection or a change on a *live* pane parameter. The status of the run mode is shown below. Observe the bitmap color differences between the stop and run modes. The color difference is very subtle.



In either mode, while there are glyphs running, the $\boxed{\text{Run}}$ icon button shows this by changing to



Experimenting with VisiQuest run modes

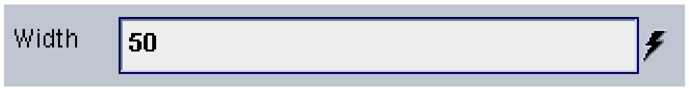
To better understand the `VisiQuest` execution modes, use the `sine.wk` workspace.

1. Put the workspace in stop mode by clicking on the $\boxed{\text{Stop}}$ button in the `VisiQuest` command bar menu. This allows you to execute each glyph manually. Try running the `Sinusoid` glyph. Observe that the network enters in run mode and back to stop mode.

2. Execute the workspace, by clicking on the workspace $\boxed{\text{Run}}$ icon of the command bar. This schedules the `Display 2D Plot` and the `Data Object Info` glyphs. Note that the workspace enters in the run mode. After the network reaches a stable condition, it stays ready to schedule any glyph if a new event occurs.

3. Manually execute the `Sinusoid` glyph to see that the workspace is re-executing again.

4. Now, open the pane of the `Sinusoid` glyph. Notice that there are parameters which are live, as shown by the lighting bolt to the right of the parameter input. An example of a live parameter is *Width*.



Width 50

5. Change this parameter to `100` and, as soon as you press the <`Enter`> key, the `Sinusoid` glyph is rescheduled and the other glyphs which depend on its output are also rescheduled.

   If you change a non-live parameter, like the parameter *Amplitude*, the glyph is not scheduled when you press the <`Enter`> key, but it is triggered when you close its pane.

6. Change the *Amplitude* to `10` and close the pane to verify this condition.

Resetting a workspace

Another command related to the workspace execution mode is *Reset*, available both in the *Workspace* menu and in the command bar. Resetting a workspace has the effect of making all the data connections invalid, changing their color to yellow. If the workspace is in stop mode, this is the only effect. If the workspace is in ready mode, the workspace is rescheduled.
Tip: Reset is useful when you want to rerun all the glyphs in a workspace after changing parameters in many glyph panes.

VisiQuest as a prototyping environment

The basic features of creating and executing a visual program in `VisiQuest` have been introduced. As a user interface, `VisiQuest`, compared to the CLUI and GUI interfaces in VisiQuest, has numerous advantages. `VisiQuest` makes it easier to find operators/glyphs. Input and output parameters are created automatically from the network connections. Glyphs are executed harmonically in an event-driven approach. And a workspace can be stored in a file for later recovery.

Advanced `VisiQuest` Programming (Chapter 5) will conclude this discussion by introducing the concept of glyph synchronization, flow control, and procedures.

Conclusion

In this chapter, you have been introduced to the interfaces to VisiQuest tools. You have learned how `VisiQuest` is related to the CLUI and GUI interfaces. You have also built the first `VisiQuest` workspace program. In addition, you were introduced to the VisiQuest data model, using 1D, 2D, and 3D data sets.

In the next chapter, you will explore additional features of `VisiQuest` as you learn how to get data in and out of the VisiQuest environment.

Chapter 3

Import and Export Data

VisiQuest accepts most standard data file formats. Its operators can read all supported formats without the need for an explicit conversion. When reading a file, VisiQuest determines its file format and converts it to the Polymorphic Data Model.

Operators by default write the data in the VisiQuest KDF file format. To change the default mechanism, use the environment variable KDMS_FORMAT with the name of the desired file format chosen from the supported formats.

3.1 Importing ASCII Data

The simplest supported format in VisiQuest is ASCII, which can represent 2D (width x height) data. You will practice with this format first since it gives you another intuitive way to understand the nature of digital images and digital data in general.

By using a regular text editor, like `xedit`, you first create the file `myimage.ascii`, consisting of a small two-dimensional matrix or image. After the matrix is created, use the interactive image operator `editimage` to visualize the ASCII file.

Creating a 2D file using xedit

1. Invoke the `xedit` text editor by typing

```
%xedit myimage.ascii
```

2. Enter the following pattern:

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

3. Now save it and quit `xedit`.

4. In `VisiQuest`, clear the previous workspace.

   Tip: `VisiQuest` can work with many workspaces simultaneously. To understand how this works, select the *New* option under the *File* menu in `VisiQuest` (or type <`Ctrl-N`>). This creates a new workspace area without deleting the previous one. Observe that you can delete a workspace area by selecting the *Close...* option, or type <`Ctrl-W`>.

Glyphs menu

In the new visual program you will need an operator to read the data file `myimage.ascii`. In the last chapter you used the *Finder* tool to locate glyphs. Another way to find an operator is through the *Glyphs* menu. All the operator glyphs available from the installed toolboxes appear under this menu. These glyphs are classified by category and sub-category.

User defined

5. Pick the `User defined` operator under the *Glyphs* menu in the category *Input/Output*, sub-category *Data Files*.

6. Open the pane of the operator and use the *File Browser* tool to select the file `myimage.ascii`.

7. Close its pane.

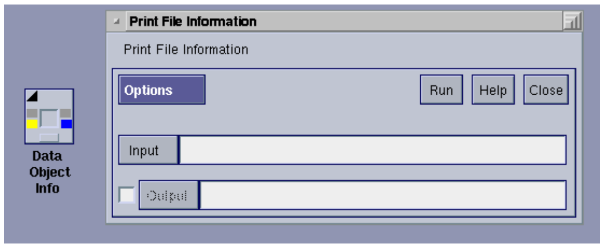
Data Object Info

8. In the same way, choose the `Data Object Info` glyph, which is located in the *Glyphs* menu under *Input/Output* and *Information*. Place it to the right of the `User defined` glyph.

Category and sub-category of a glyph

Tip: To find which category and sub-category a glyph belongs to, open its pane and click on the $\boxed{\text{Options}}$ pulldown menu. Select *Object Info...*.

A pop-up window shows the attributes related to the glyph. Click in the window to position the cursor, and use the down-arrow key to see the text at the bottom.



Running Data Object Info on an ASCII file

9. In the pane of the `Data Object Info` glyph, check to see that the *Output* parameter is diabled and close the pane.

10. Connect the two glyphs together.

11. Execute the `Data Object Info` glyph to extract the header information of the ASCII data file format `myimage.ascii`.

12. Open the *Green Console Button* and verify that its dimensions are indeed 10 pixels in the *width* dimension and 5 pixels in the *height* dimension. Also observe that the *Storage Format* is recognized as ASCII and that each pixel *Data Type* is `Double`, which is a double-precision floating point format. Below is an excerpt of the output of the `Data Object Info` glyph.

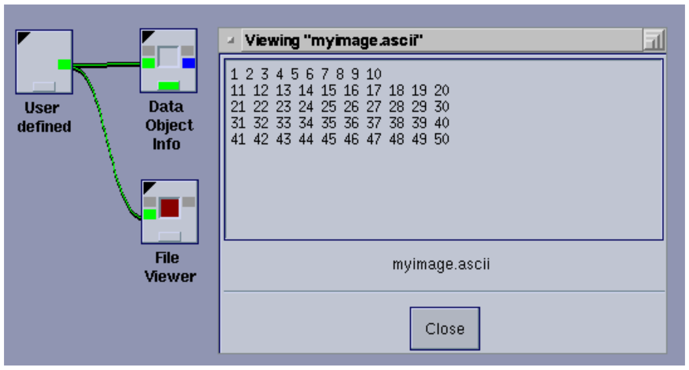
```
# Name:  myimage.ascii
# Storage Format:  ascii
# -- Value Data --
# Data Type:  Double
# Size:  Width=10, Height=5, Depth=1, Time=1, Elements=1
```

File Viewer

The `File Viewer` glyph, which corresponds to the `khelp` operator, is used to visualize ASCII files.

13. Pick up the `File Viewer` glyph, located in the *Glyphs* menu under *Input/Output* and *Information* and place it below the `Data Object Info` glyph.

14. Connect its input to the output of the `User defined` glyph and run it. By doing this you display the actual ASCII contents of the `myimage.ascii` file.



```
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
```

File Viewer in the output of Data Object Info

Recall that the `Data Object Info` glyph has an optional output file. When disabled, it sends its output to the *Green Console Button*. Now use another `File Viewer` glyph to view the output file generated by the `Data Object Info` glyph when it is enabled.

Duplicating a glyph

To duplicate a glyph, first select it by clicking on it and observe a color change from light to dark grey. Then, select the duplicate option in the *Edit* menu. A short cut for this operator is <`Ctrl-D`> or the correspondent icon button located in the command bar.

15. Duplicate the `File Viewer` glyph and place it to the right of the `Data Object Info` glyph.

16. Connect the output of the `Data Object Info` glyph to the input of the `File Viewer` glyph.

    Note: This connection is yellow, meaning that you have to execute the `Data Object Info` glyph again. The last time you ran it, it sent the file output to *stdout* as the output parameter was disabled.

17. Rerun the `Data Object Info` glyph and run the `File Viewer` glyph.

Editimage on ASCII file

To verify that the ASCII data can be displayed as an image, use the object `editimage` in `VisiQuest`.

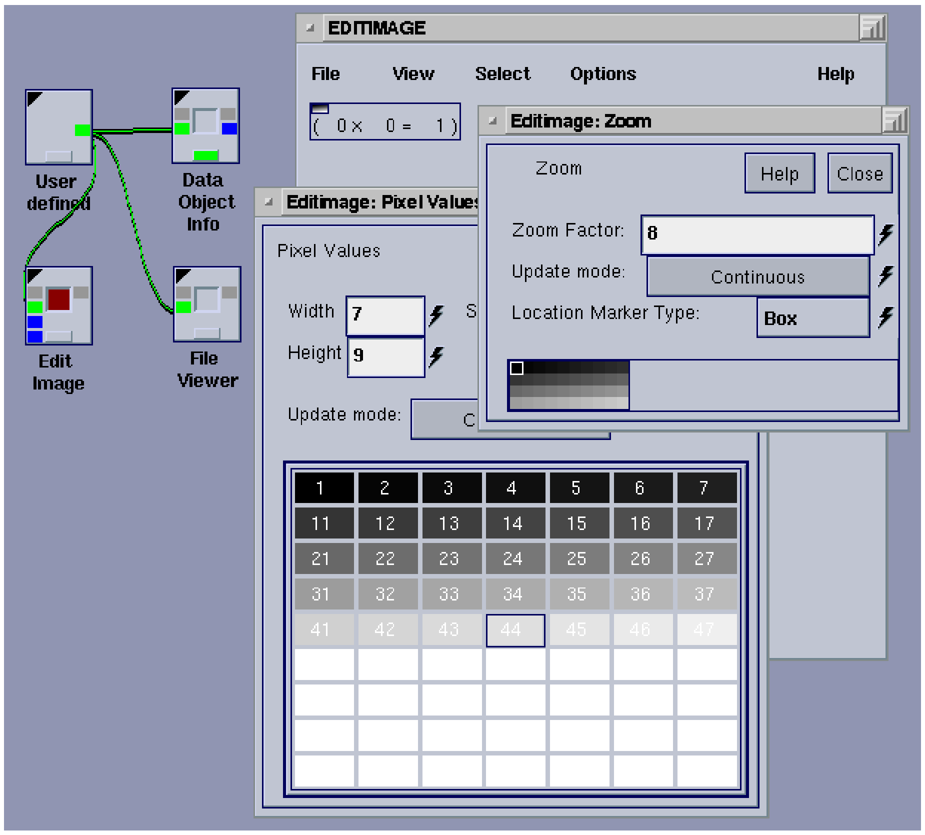
18. Choose the `Edit Image` glyph, which is found under *Visualization* and *Interactive Image Display.*

19. Place it below the `User defined` glyph and connect its top input connection to the output of the `User defined` glyph and run it.

    Observe the small size of the image in the `editimage` window, due to the fact that there is a one-to-one mapping between the pixel stored in the image and the pixel displayed.

20. As you have done before, select the View $\boxed{\text{Zoom}}$ and View $\boxed{\text{Pixel Values}}$ buttons to perform a detailed visualization of the image.

21. Save this workspace under the name `importing-ascii.wk`.



(0 x 0 = 1)

Zoom

Zoom Factor: 8

Update mode: Continuous

Location Marker Type: Box

Pixel Values

Width 7

Height 9

Update mode: C

1	2	3	4	5	6	7
11	12	13	14	15	16	17
21	22	23	24	25	26	27
31	32	33	34	35	36	37
41	42	43	44	45	46	47

3.2 Exporting Data to ASCII

This section explains how to export data to the ASCII format for re-use in other programs. First, you will generate a 1D Gaussian data file, using the `2D Gaussian` glyph, and visualize the data by using the `Display 2D Plot` glyph. Then, you will be shown how to convert data to the ASCII file format, visualize the ASCII data, and save it in the file `gaussian.txt` in the home directory.

Generating 1D Gaussian data

1. Clear the previous workspace and find the `2D Gaussian` glyph (category *Input/Output*, sub-category *Generate Data*), and fill in the following parameters in its pane:

Parameter	**Value**
Width	`1`
Height	`21`
Peak location along Width (X)	`0`
Peak location along Height (Y)	`10`
Variance along Width	`1`
Variance along Height	`4`
Correlation coefficient	`0`
Peak Value	`1.0`
Select Data Type:	`float`

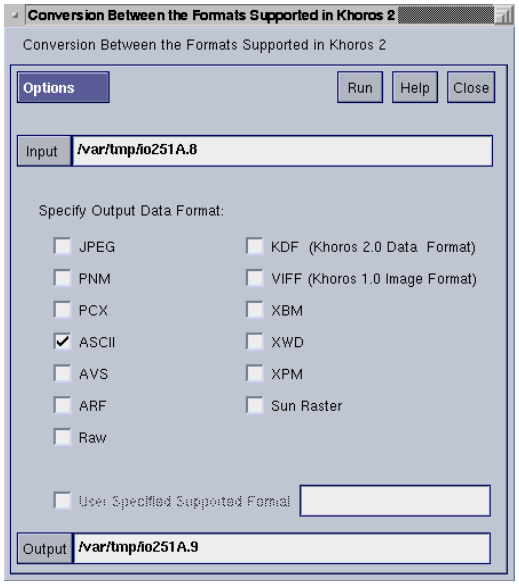
   This generates a Gaussian data file as a 1D signal in the *height* dimension with 21 samples. The maximum position of the Gaussian bell (peak) is at position 10 with value 1, and the spread of the Gaussian data around the peak is given by its variance of 4.

2. Retrieve the `Display 2D Plot` glyph (*Visualization*, *Plot Display*) and connect its input to the output of the `2D Gaussian` glyph.

3. Execute the `2D Gaussian` and the `Display 2D Plot` glyphs to see the shape of the Gaussian data created.

Converting to ASCII

The file conversion routines are integrated in the `Supported Formats` glyph. Below is its pane, from which you can verify the formats supported by VisiQuest. Remember that all these formats are accepted directly as input to any operator.


Conversion Between the Formats Supported in Khoros 2

Conversion Between the Formats Supported in Khoros 2

Input /var/tmp/io251A.8

Specify Output Data Format:

- [ ] JPEG
- [ ] PNM
- [ ] PCX
- [✓] ASCII
- [ ] AVS
- [ ] ARF
- [ ] Raw
- [ ] KDF (Khoros 2.0 Data Format)
- [ ] VIFF (Khoros 1.0 Image Format)
- [ ] XBM
- [ ] XWD
- [ ] XPM
- [ ] Sun Raster

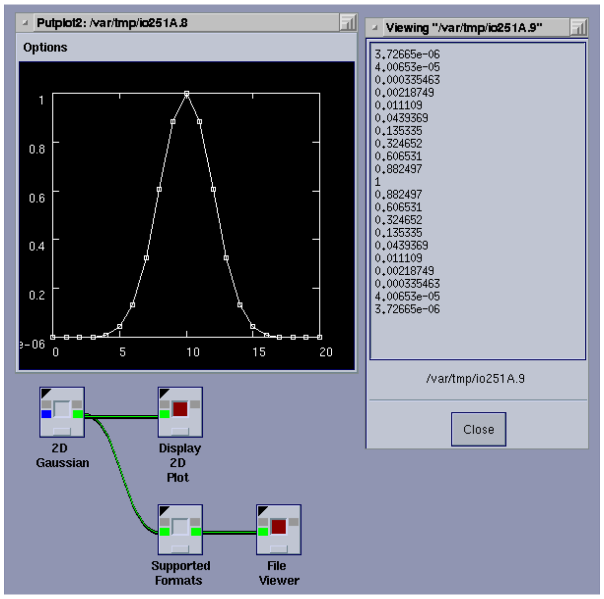
- [ ] User Specified Supported Format

Output /var/tmp/io251A.9

4. Retrieve the `Supported Formats` glyph from *Input/Output*, *Export Data*, connect its input to the `2D Gaussian` glyph, and select `ASCII` format in its pane.

5. Now, choose the `File Viewer` glyph from *Input/Output*, *Information* and connect it to the output of the `Supported Formats` glyph.

6. Run the `Supported Formats` and the `File Viewer` glyphs to verify the converted ASCII data file.

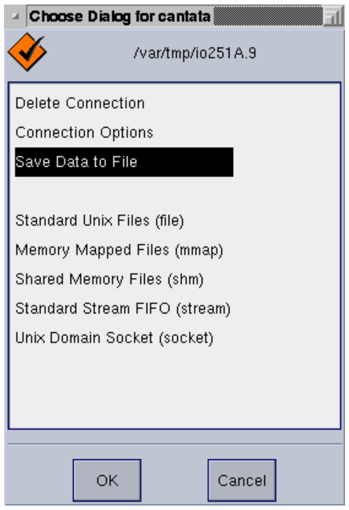


```
3.72665e-06
4.00653e-05
0.000335463
0.00218749
0.011109
0.0439369
0.135335
0.324652
0.606531
0.882497
1
0.882497
0.606531
0.324652
0.135335
0.0439369
0.011109
0.00218749
0.000335463
4.00653e-05
3.72665e-06
```

Saving the data to an explicit file

In the `VisiQuest` workspace, data is represented by connections between the glyphs, and their names created automatically in a temporary directory.

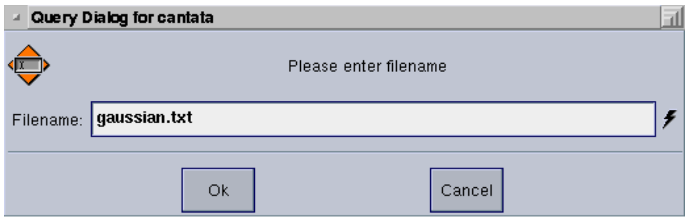
7. Save the ASCII Gaussian data by clicking on the connection between the `Supported Formats` and `File Viewer` glyphs. A connection window pops up. Select `Save Data to File` and click on the $\boxed{\text{Ok}}$ button.


Choose Dialog for cantata
/var/tmp/io251A.9
Delete Connection
Connection Options
Save Data to File
Standard Unix Files (file)
Memory Mapped Files (mmap)
Shared Memory Files (shm)
Standard Stream FIFO (stream)
Unix Domain Socket (socket)
OK
Cancel

Another window to create filenames appears.

8. Fill in the filename `gaussian.txt`.

   Note: Don't forget to click on the $\boxed{\text{Ok}}$ button, otherwise the file will not be saved.



Filename: gaussian.txt

9. Save this workspace under the name `export-data-gaussian.wk` and clear the workspace.

ASCII data and spreadsheets

The created file `gaussian.txt` can be exported to many other software packages that read ASCII data, the simplest file format for data exchange. Most spreadsheets can further process ASCII data and plot it.

Using gnuplot to generate PostScript output

For instance, we use `gnuplot` to plot data and generate *PostScript* files suitable for hardcopy material. PostScript is a standard file format for printers. There are two types of PostScript file formats: regular PostScript (PS) and encapsulated PostScript (EPS). The first is used for final output and EPS is used for insertion of data in other files. In VisiQuest and 2001, PostScript is only supported for the generation of PostScript output images.

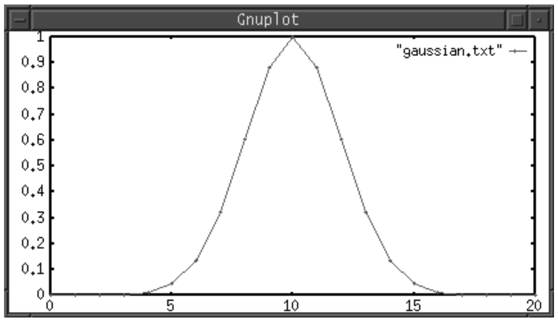
Gnuplot

`gnuplot` is a command-line-driven, interactive plotting utility widely available in UNIX systems. If you don't have it installed, you can download it from `http://www.cs.dartmouth.edu/gnuplot_info.html`.

The following `gnuplot` commands plot the ASCII file `gaussian.txt` and write the output in EPS PostScript in the file `gaussian.eps`.

At the command line, type in the following commands: [1]

```
%gnuplot

gnuplot> plot "gaussian.txt" with linespoints
gnuplot> set terminal postscript eps
gnuplot> set output "gaussian.eps"
gnuplot> replot
gnuplot> quit
```



You can view the PostScript plot with the tool `ghostview`, a popular PostScript visualizer. If you don't have it available in the system, you can download it from `http://www.cs.wisc.edu/~ghost/ghostview/`.

```
%ghostview gaussian.eps
```

Most word processing packages support the insertion of PostScript data within the text. All the figures in this tutorial were inserted as PostScript

[1]For version VisiQuest 1.0.0.0, use an editor to remove the last line of file gaussian.txt which contains the spurious string "<br>".

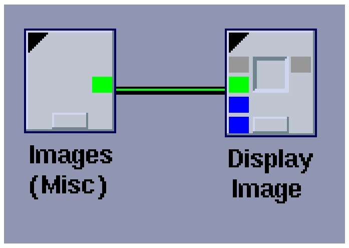
files.

3.3 Exporting Images to PostScript Format

PostScript is a hardcopy format and, as such, it does not make sense to use this file format for data exchange. VisiQuest includes an operator to convert images to PostScript format.

In this section you will learn how to create a workspace to convert an image to PostScript format and then visualize it with the `ghostview` tool inside `VisiQuest`.

1. Clear the previous workspace and retrieve the `Images (Misc)` glyph (*Input/Output*, *Data Files*). Open its pane, choose the `Spanish Sea Gull` image, and close the pane.
2. Choose the `putimage` operator, which corresponds to the `Display Image` glyph in `VisiQuest`. The `Display Image` glyph is in category *Visualization*, sub-category *Non-Interactive Image Display*.
3. Place the glyph to the right of the `Images (Misc)` glyph and connect its first input to the `Images (Misc)` glyph. Run the `Display Image` glyph to visualize the gull image.



4. Now, find the `Postscript` glyph (*Input/Output*, *Hard-Copy Output*) and place it below the `Display Image` glyph. Connect it to the `Images (Misc)` glyph and open its pane.
5. Select `Encapsulated Postscript`, close its pane, and execute its glyph. This makes the output suitable for inclusion in other documents.

Interfacing VisiQuest with third-party programs

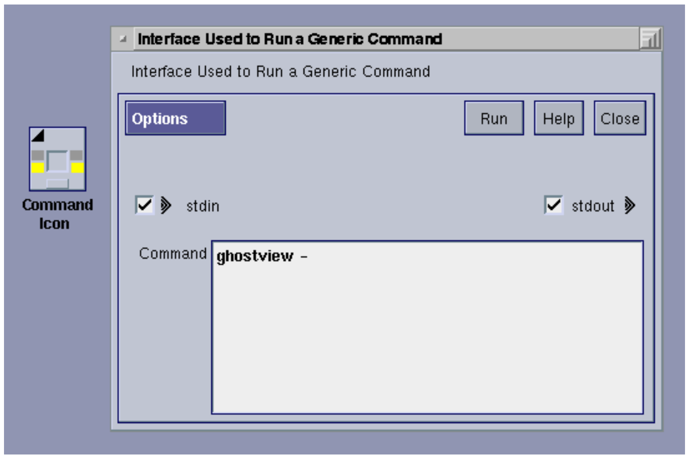
To display the generated PostScript file, we use the `ghostview` tool that we used to visualize the PostScript file of the Gaussian plot. This time, however, we use `ghostview` directly from `VisiQuest`. There are many ways to interface third-party programs into VisiQuest. This example is the simplest.

6. Find the `Command Icon` glyph (*Program Utilities*, *General*) and place it to the right of the `Postscript` glyph.

   This glyph is a general interface program that uses *stdin* and *stdout* standard file connections. Any program which can read from *stdin* and write to *stdout* can be interfaced with the `Command Icon` glyph. In this case, to make `ghostview` read from *stdin* you have to use the syntax `ghostview -` which is an option in `ghostview` for reading from *stdin*.

7. Open its pane and disable the *stdout* option, as there is no output from this glyph.

8. In the *Command* field, type in `ghostview -` and close the pane.



Interface Used to Run a Generic Command

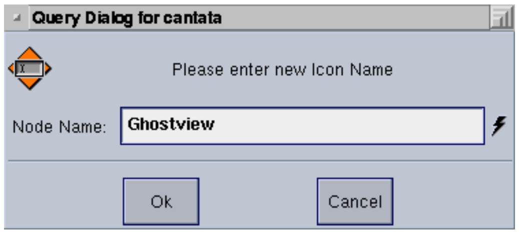
☑ stdin ☑ stdout

Command: ghostview -

Changing the name of a glyph

`VisiQuest` allows you to change the name of a glyph icon to better document your workspace.

9. Change the name of the `Command Icon` glyph to `Ghostview` by clicking on the glyph name, and entering the name `Ghostview` in the *new Icon Name* window.



Query Dialog for cantata

Please enter new Icon Name

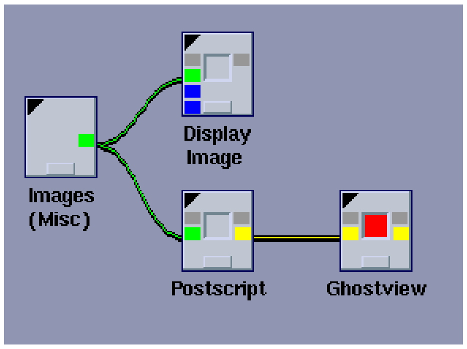
Node Name: Ghostview

10. Connect the input of the `Ghostview` glyph to the output of the `Postscript` glyph.

11. Execute the `Ghostview` glyph and verify that the PostScript file was generated properly.

12. Save the EPS file under the name `gull.eps` by clicking on the output connection of the `Postscript` glyph. This EPS file is ready for insertion in most word processor packages.

13. Save the workspace under the name `gull-postscript.wk`. A screen dump of this workspace is shown below.



3.4 How To Export Images to Other Formats

You normally export images with the `Supported Formats` glyph used in section 3.2. The formats understood by VisiQuest are listed in the `Supported Formats` pane. VisiQuest does not support every data file format. For those that are not supported, there are software packages available that allow you to create an interface with `VisiQuest`. In this section, we will examine how `xv` can be used to generate other file formats.

3.5 Interfacing to xv

`xv` is a widely used software package to manipulate and visualize images. It supports a variety of file formats such as GIF, JPEG, TIFF, and BMP that are not supported by VisiQuest. GIF and JPEG are used in many HTML documents for World Wide Web (WWW) publishing and TIFF is used for desktop publishing. `xv` is available at `ftp://ftp.cis.upenn.edu/pub/xv/`.

PNM image format

Among those formats supported by VisiQuest, PNM and Sun Raster are widely used. PNM, one of the simplest image file formats, was created in association with the software PBMPLUS (which can be found at `http://www.acme.com/software/pbmplus/`). PNM supports three pixel data types: PBM for bit-type images, PGM for unsigned integer images, and PPM for color images.

xv and gif

1. Clear the previous workspace. Start by finding the `Images (Misc)` glyph under *Input/Output*, *Data Files* and in its pane select the image `Spanish Sea Gull`.

2. Find the `Supported Formats` glyph (*Input/Output*, *Export Data*). Place it to the right of `Images (Misc)` and select the `PNM` file format in its pane. Connect its input to the `Images (Misc)` glyph.

3. Now, find the `Data Object Info` glyph (*Input/Output*, *Information*) and place it to the right of the `Supported Formats` glyph. Open the pane, make sure its *Output* option is disabled, and close the pane. Connect its input to the `Supported Formats` glyph and execute it.

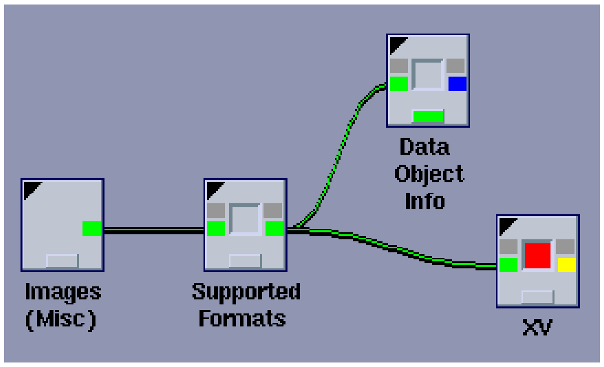
4. Click on the *Green Console Button* of the `Data Object Info` glyph after it runs and verify that the *Storage Format* of this file is PNM.

```
# Name:  /tmp/io94.1c
# Storage Format:  pnm
# Color Space Model:  KGREY
# -- Value Data --
# Data Type:  Unsigned Byte (4)
# Size:  Width=256, Height=256, Depth=1, Time=1, Elements=1
```

5. Place the `Command Icon` glyph (*Program Utilities*, *General*) below the `Data Object Info` glyph. Open its pane, disable the *stdout* option, and type in the string `xv -` in the *Command* field, and close the pane.

6. Change the `Command Icon` glyph name to `xv` by clicking on the icon name.

7. Connect the input of the `xv` glyph to the output of the `Supported Formats` glyphs.



Using xv to save other image data file formats

8. Run the `xv` glyph.

   Once `xv` is running, the gull image is displayed in its window.

9. Click on the image with the right-hand mouse button to display the `xv` control window. Click on the $\boxed{\text{Save}}$ button and then select the GIF file format. Enter the filename `gull.gif` and click on the $\boxed{\text{Ok}}$ button. This GIF file is ready to be included in an HTML document.

10. Exit `xv` by clicking on the $\boxed{\text{Quit}}$ button of the `xv` control window.

11. Save the workspace under the name `xv.wk` and execute the entire workspace.

Importing other image data file formats

You can also use `xv` to import data formats that are not supported by VisiQuest. You can display the image you want to import by using `xv` and then save the file in PBM format (select the color, grey, or bitmap option). This PBM file is supported by VisiQuest.

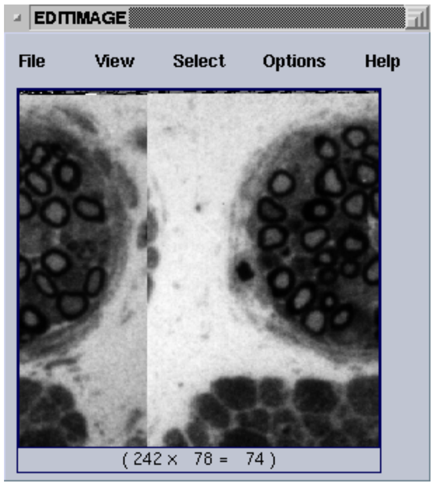
3.6 Reading Data in Binary Format

For reading data files in binary format where the complete specification is unknown, VisiQuest provides a very flexible tool, the `Import Raw` glyph. To use this glyph, adjust parameters until you find what you want on a trial-and-error basis.

In this experiment, we will illustrate how to read a file with a partially known raw binary data format. You will read the file `Nerve Cell` by using the `Import Raw` glyph. Although we know that this file is in the KDF file format with size 256 x 256, we will suppose that you know that the file is in binary raw format with 256 x 256 pixels stored as unsigned bytes in raster order, i.e., from left to right and from top to bottom. You also know that the file has a header stored in the beginning of the file, but you don't know its size. The header of an image file contains information about the file. This scheme is common to many standard file formats.

1. Clear the previous workspace and find the `Images (Misc)` glyph (*Input/Output, Data Files*), then select the image *Nerve Cell* in its pane.
2. Choose the `Import Raw` glyph (*Input/Output, Import Data*) and connect its first input to the `Images (Misc)` glyph.
3. Find the `Edit Image` glyph (*Visualization, Interactive Image Display*) and connect its first input to the output of the `Import Raw` glyph.
4. Open the `Import Raw` pane and set both parameters *Width* and *Height* to 256. Leave *Offset or skip (bytes)* as 0.

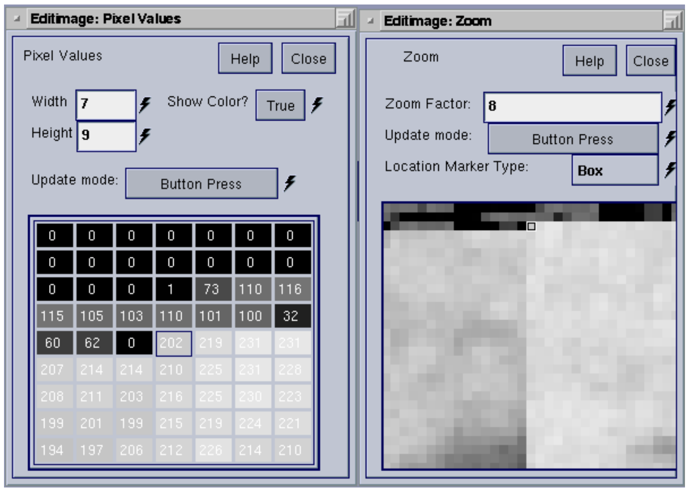
   This operator is very flexible. You specify the number of pixels to read, and the order they are stored in each dimension. You also specify the format each pixel is stored in and at which offset the pixels are stored. In this case the `Import Raw` glyph will read the file as 256 x 256 pixels stored as bytes starting from the very first byte of the file.
5. Run the glyphs. The display output in `Edit Image` is shown below.



Note that the image is wrapped shifted to the right. Also note that the top few lines on top of the image appear as black with noise. This is the typical effect found in reading raster data where the offset to skip the header information is not correct. You need to estimate how many pixels there are in these noise lines. Next, you will use the `Edit Image` tool to zoom the image and find the coordinates of the first valid pixel in the image.

6. Open the View *Zoom* and View *Pixel Values* tools of `Edit Image`. Position the mouse at coordinate (91 x 2) and verify that this is where the pixel data starts. A screen dump of the *Zoom* and *Pixel Values* tools is shown. The first pixel value is 202.

   If the pixel data starts at coordinate 91 x 2 and the pixels are stored in the width first, it means that the header has a size of 2*256 + 91.



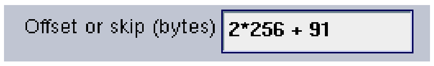
Width 7
Show Color? True
Height 9
Update mode: Button Press

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	73	110	116
115	105	103	110	101	100	32
60	62	0	202	219	231	231
207	214	214	210	225	231	228
208	211	203	216	225	230	223
199	201	199	215	219	224	221
194	197	206	212	226	214	210

Zoom Factor: 8
Update mode: Button Press
Location Marker Type: Box

7. In the pane of the `Import Raw` glyph, change the *Offset or skip (bytes)* to `2*256 + 91`.

   Note: The numerical parameters can be entered as expressions. Later you will see that `VisiQuest` supports the use of variables in the expressions.



Offset or skip (bytes) 2*256 + 91

8. Rerun the glyphs and note that the image is displayed correctly. Verify that the pixel (0,0) of the image has value 202.

9. Save this workspace under the name `importing-raw.wk`.

In this section you learned how to import raw data in binary file format. VisiQuest has a very flexible operator to read this kind of file. You used a known data file format to illustrate the input of binary raw data format. In the next section, you will learn how to export data to other formats that do not have the same features as the VisiQuest Data Format.

3.7 Data Normalization

When exporting data, you must be aware that not all file formats fully support the features of the VisiQuest Polymorphic Data Model. In this model, a pixel value can be stored in many data types, such as bit, unsigned byte, floating point, etc. Most image file formats do not support this flexibility. In this section you will export an image which has negative and positive pixel values to the Sun Raster file format, which supports pixels in the range 0 to 255.

1. Clear the previous workspace, retrieve the `Images (Misc)` glyph (*Input/Output, Data Files*), and select the *Big Truck (signed short)* image.

2. Find the `Data Object Info` glyph (*Input/Output, Information*), connect it to the `Images (Misc)` glyph. Disable its *Output* option in its pane, and run it to verify that the pixel *Data Type* is *short.* Short is a signed 16-bit format capable of representing pixels from values -32,768 up to 32,767.

```
# Comment:
Image produced by a long wave infrared sensor of
an M35 truck.
# End Comment
# Storage Format:  kdf
#
# -- Value Data --
# Data Type:  Short (8)
# Size:  Width=596, Height=356, Depth=1, Time=1, Elements=1
```

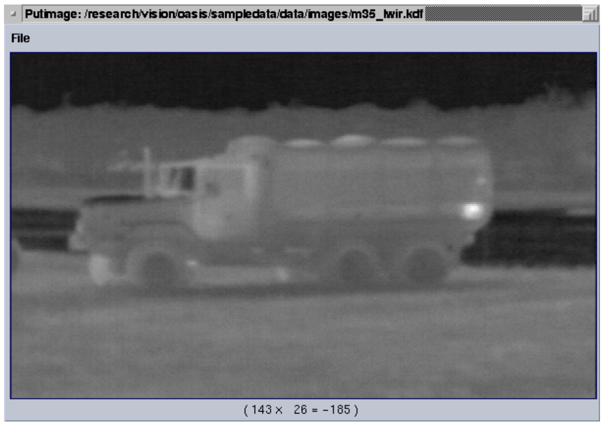
3. Select the `Statistics` glyph (*Data Manip, Analysis & Information*). Connect it to the `Images (Misc)` glyph and run it to verify the minimum and maximum pixel values within the image. An excerpt of the output of the *Green Console Button* of the `Statistics` glyph is shown next.

```
FILE: m35_lwir.kdf
Object Dimension:  w=596 h=356 d=1 t=1 e=1
Mean:  490.116
Std Dev:  374.813
Minimum:  -539
Maximum:  2329
```

Notice that the minimum value is negative (-539) and that the maximum value of 2,329 is well above the usual 255 scale.

4. Select the `Display Image` glyph (*Visualization*, *Non-Interactive Image Display*) and connect its first input to the `Images (Misc)` glyph. Execute the glyph and move the mouse pointer over the image to verify that dark areas are assigned to negative pixel values and white areas are the highest positive values.

   In this way `Display Image` assigns grey-tones to pixel values. To the minimum pixel value in the image, it assigns the black color, while it assigns the white color to the maximum pixel value, creating a linear grey-scale colortable that assigns intermediate grey-tones to all other pixel values.


Putimage: /research/vision/oasis/sampledata/data/images/m35_lwir.kdf
File
(143 × 26 = -185)

   Next, you will convert this image directly to the Sun Raster format and verify that it does not work as expected.

5. Access the `Supported Formats` glyph (*Input/Output, Export Data*) and select *Sun Raster* format. Connect its input to the `Images (Misc)` glyph.

6. Run the `Supported Formats` glyph and observe that an *Green Console Button* appears. Open the *Green Console Button* and verify its contents

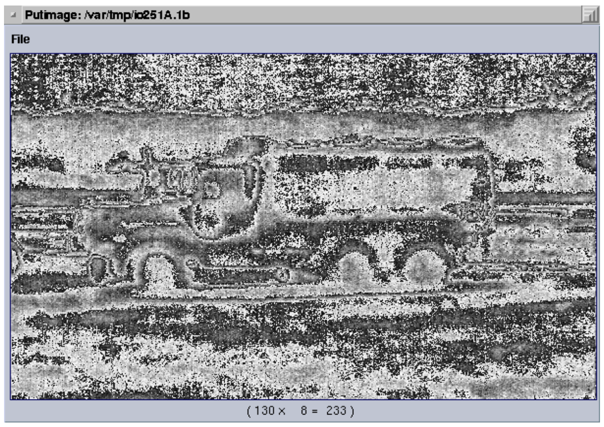
```
Sun Raster format output :
- casting value data to Unsigned Byte for storage.
```

   The conversion routine states that the pixel values are stored in the `Unsigned Byte` in the Sun Raster format. Casting a value to a smaller data type storage means that a truncation operation was done. In this case pixel values outside the range 0 to 255 may be lost.

   Verify how the images were converted in the Sun Raster format by displaying it.

7. Duplicate the `Display Image` glyph and connect it to the output of the `Supported Formats` glyph.

8. Run the `Display Image` glyph and verify with the mouse that the image has pixels ranging from 0 to 255, however the truncation process corrupted the image.



To adjust for the truncation process in the exported data file, it is possible to normalize the data in a range that fits in the unsigned byte data storage.

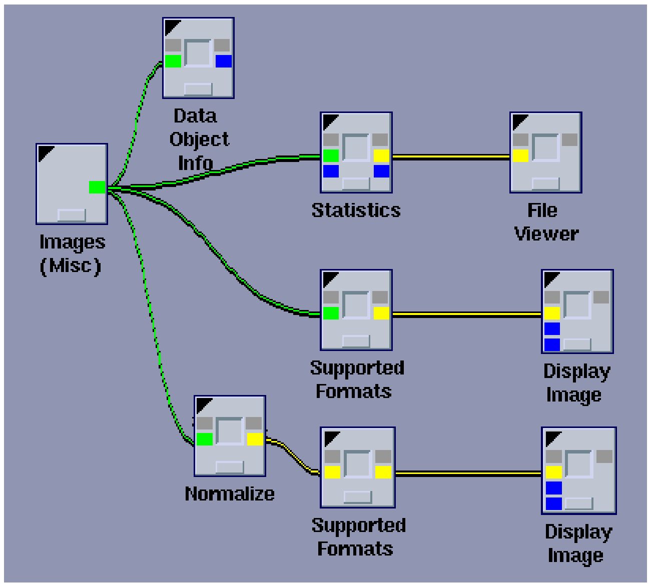
9. Select the `Normalize` glyph (*Data Manip, Data Conversion*) and connect its input to the `Images (Misc)` glyph.

10. Duplicate the `Supported Formats` glyph and connect it to the `Normalize` glyph.

11. Duplicate the `Display Image` glyph and connect it to the `Supported Formats` glyph.

12. Execute the glyphs and verify that the image is displayed correctly.

13. Save the workspace under the name `normalization-truck.wk`.



To repeat, it is important to realize that in exporting data to other formats, information can be lost. In the case of the normalization example above, the original pixel values were lost as well as the dynamic range of the data. Originally, the data ranged from -539 to 2,329 (range of 2,869); after conversion the data ranged from 0 to 255 (range of 256).

3.8 More About the Data

Statistics tool

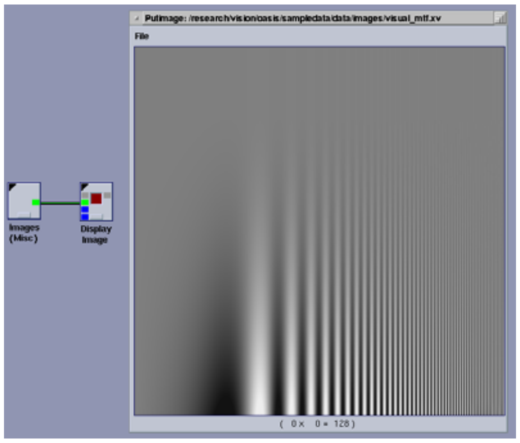
An important tool with which to gain insight on the data is the statistic measurement operator, with basic information on minimum, maximum, and mean values. VisiQuest provides the `kstats` operator, which is activated by the `Statistics` glyph.

Using variables in VisiQuest

This section will explore the `Statistics` tool and the use of variables in `VisiQuest`. Variables are an important feature of a visual program, since they can replace numeric arguments in the panes. The workspace, described below, thresholds an image by its mean value. A binary image is generated from a grey-scale image based on the criterion that each pixel value above the mean value is assigned the value 1. All other pixels are assigned the value 0.

1. To begin, clear the previous workspace and find the `Images (Misc)` glyph (*Input/Output*, *Data Files*). In its pane, select the image `Modulation Transfer Function`.

2. Find the `Display Image` glyph (*Visualization, Non-Interactive Image Display*) and place it above and to the right of the `Display Image` glyph. Connect its first input to the `Display Image` glyph and execute it.


PutImage: /research/vision/oasis/sampledata/data/images/visual_mtf.xv
File
Images (Misc)
Display Image
(0x 0= 128)

This image is rather interesting, serving to measure how the contrast sensitivity of the human eye depends on the spatial frequency. The image is composed of a 2D signal with increasing frequency on the width direction and increasing amplitude (contrast) in the height direction. Looking at this image, we realize that our perception is attuned to medium frequencies, as opposed to low or high frequencies.

3. Choose the `Statistics` operator *Data Manip, Analysis & Information*. Place it below the `Images (Misc)` glyph and connect its first input to the output of the `Images (Misc)` glyph.

4. Execute the `Statistics` glyph. Its output goes to the *Green Console Button*. Click on it to reveal the information. Its result is reproduced below. Note that this image has minimum and maximum pixel values of 0 and 255, respectively, and a mean value of 124.14.

```
Object Dimension:  w=512 h=512 d=1 t=1 e=1
Mean:  124.142
Variance:  1004.15
Std Dev:  31.6883
RMS: 128.122
Skewness:  -0.00614449
Kurtosis:  3.38875
Minimum:  0
Maximum:  255
First minima at location:  w=125 h=511 d=0 t=0 e=0
First maxima at location:  w=173 h=511 d=0 t=0 e=0
Total Integral:  3.2543e+07
Positive Integral:  3.2543e+07
Negative Integral:  0
Total Contributing Points:  262144
Contributing Positive Points:  262134
Contributing Negative Points:  0
Contributing Zero-Valued Pts:  10
```

5. Close the `Statistics` text window.

Variables

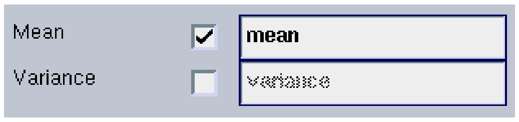
The `VisiQuest` visual language supports the use of expressions in the parameter fields of the operators. Valid expressions may include variables, standard arithmetic operators, and logicals, as well as predefined constants and functions. The variables can be calculated at run time via mathematical expressions tied to data values or control variables, or interactively set by the user.

Assigning to a variable: Print Stats

Use the `Print Stats` glyph to assign the mean value of the data file to a variable named `mean`.

6. Find the `Print Stats` glyph (*Data Manip*, *Analysis & Information*) and place it below the `Statistics` glyph in the workspace.

7. Open its pane and enable the *Mean Calculation* option. The default variable associated with this evaluation is `mean`.


Mean ✓ mean
Variance ☐ variance

8. Close the pane and connect its input to the output of the `Images (Misc)` glyph and execute the glyph.

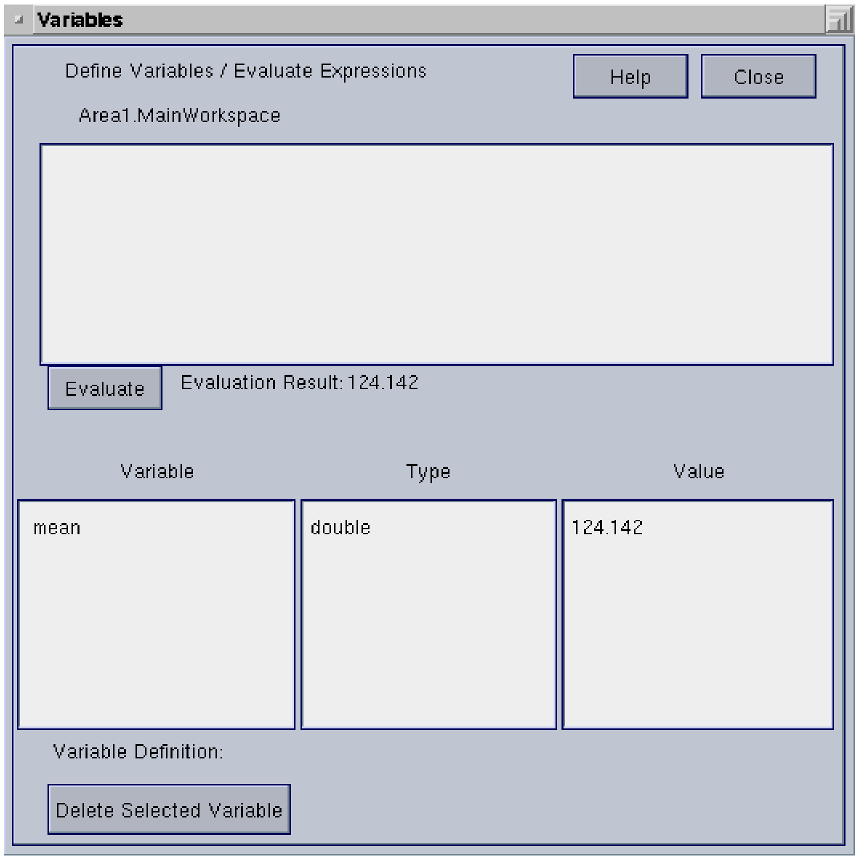
Variable subform

The effect of the execution of the `Print Stats` glyph is on the value of the variable `mean`. To verify that the glyph has executed correctly, check for the new value of the `mean` variable.

The *Variables* subform is used to display and define variables and evaluate expressions.

9. Open the *Variables* subform (*Workspace, Variables...*).

10. Notice that the `mean` that you are working on is listed in the *Variable* list at the bottom left of the *Variables* subform, that it has been defined as a double precision numer and that its value is displayed (mean = 124.142).



Define Variables / Evaluate Expressions

Area1.MainWorkspace

Evaluation Result: 124.142

Variable	Type	Value
mean	double	124.142

Variable Definition:

11. Close the *Variables* subform.

Using variables in expressions

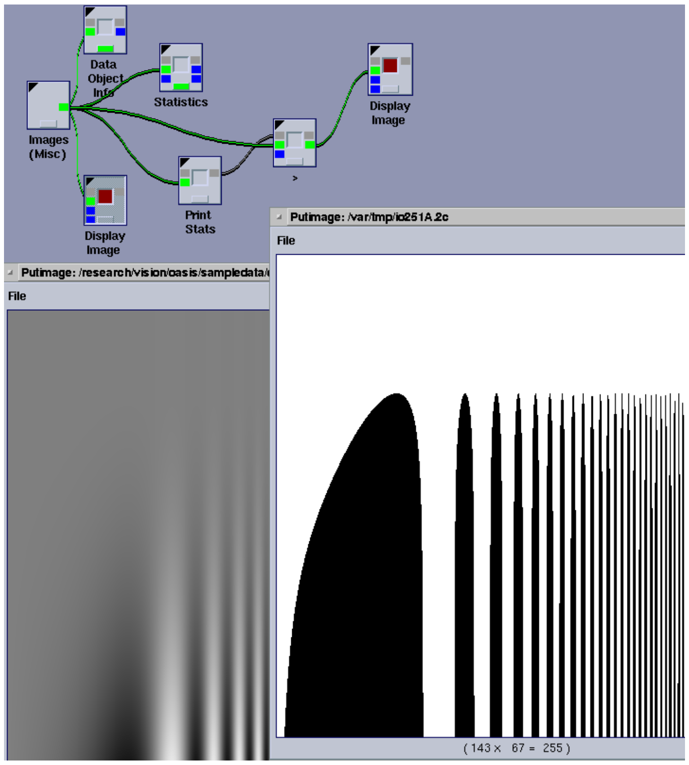
With the `mean` variable you can now do a thresholding operation on the image, by using its value as the threshold parameter. Thresholding is a widely used technique to binarize data. In this example, all pixel values of the image that are above the threshold are assigned the value 1 and all others, the value 0. In VisiQuest, many operators implement this thresholding function. The ">" operator is being used in this example.

12. Select the ">"operator, which is located in *Arithmetic*, *Comparison Operators* and place it below the `Print Stats` glyph. Open its pane, type in the string `mean` in the *Constant* parameter box, change the *TRUE value* to `255`, and close its pane.

13. Connect the output of the `Images (Misc)` glyph to the input of the ">"glyph and execute it.

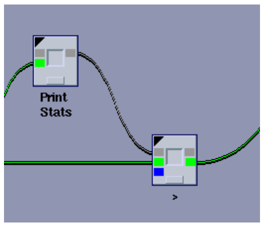
14. Visualize the result of thresholding by using another `Display Image` glyph connected to the output of the "$>$"glyph. You may use the duplicate facility to get another `Display Image` glyph.

    The pixels in this thresholded image are white wherever the pixel values are below the value 124.142 in the original image.



Glyph control connection

Note that the "$>$"glyph has to be executed after the execution of the `Print Stats` glyph. Observe that there is no connection between these two glyphs. VisiQuest provides a control connection mechanism between glyphs to synchronize their execution. These control connections are created by clicking on the small grey boxes above the data connections in the glyphs.



15. Connect the output control connection of the `Print Stats` glyph to the input control connection of the ">"operator. Note that the control connections have a different color than the data connections.

16. Change the image in the `Images (Misc)` pane to `Spanish Sea Gull`, then reset and rerun the entire workspace by clicking on the `VisiQuest` $\boxed{\text{Run}}$ button.

    Note: It is important to reset the workspace when using VisiQuest before rerunning the workspace.

17. Save the workspace under the name `threshold-mean.wk`.

Conclusion

In this chapter, you have seen how to import data to and export data from the VisiQuest environment. You worked with the ASCII data format and generated hardcopy output using PostScript files. You learned how to interface VisiQuest to third-party software packages as well as the basic concepts of inserting images and plotting in text documents.

You have advanced your knowledge of the data model and image file formats, working with raw binary images and data normalization issues.

In `VisiQuest`, you were introduced to variables and to the glyph synchronization mechanism.

Next, you will continue to explore the data model and learn how to represent colored images. You will work with the map and mask segments of the Polymorphic Data Model.

Chapter 4

Polymorphic Data Model

The experiments in earlier chapters required only the value segment of the Polymorphic Data Model. This chapter will expand the discussion of the data model to include the map and mask segments. First, we will explore how color images are built and represented with and without the map segment, followed by the use of the mask segment to define regions of interest in an image.

4.1 RGB Color Model

Color images can be represented in many color models. The color model used in monitors is the RGB, where each pixel has three components: Red, Green and Blue, which when combined can generate most of the existing colors. VisiQuest stores the color model as an attribute associated with the data. There are many color models supported by the system and the selection of the appropriate model is application dependent.

Elements dimension: RGB

In the VisiQuest Polymorphic Data Model, the color components are stored in the "elements" dimension of the value segment: Element 0 is for red, Element 1 for green, and Element 2 for blue. To experiment with color, generate an RGB color image by stacking in the element dimension three monochrome images of translated circles.

In this section, you will learn how to create a colored image using the RGB model. In the workspace described in the following pages, three circle images, decentralized in reference to each other, are stacked together in the elements dimension of the Polymorphic Data Model. The image created has eight different colors.

Circle Image

1. Clear the previous workspace and place the `Circle Image` operator (*Input/Output*, *Generate Data*) in the workspace. Open its pane and fill in the following parameters:

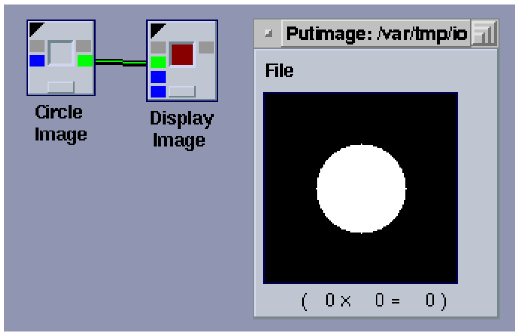
Parameter	Value
Rows:	`128`
Columns:	`128`
Diameter of circle:	`60`
X coordinate of center:	`64`
Y coordinate of center:	`64`
Background level:	`0`
Foreground level:	`255`
Output data type:	`UNSIGNED BYTE`

Older version of VisiQuest

This operator was developed for VisiQuest 1.0.5 (K1) which used a different data structure than VisiQuest. In the older version, the data model was different: *Rows* corresponds to *height* and *Columns* to *width* in the new system.

2. Close and run the glyph.

3. Find the `Display Image` glyph (*Visualization*, *Non-Interactive Image Display*), place it above and to the right of the `Circle Image` glyph, connect and execute them to visualize the circle image.

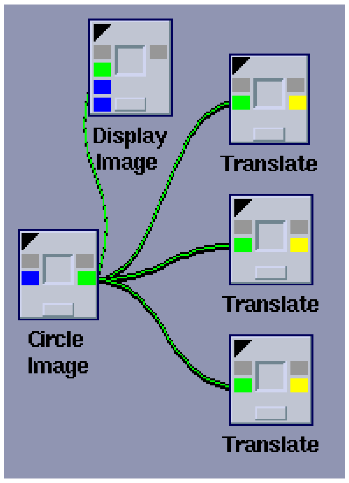


Generating three translated circles

We need three translated images, each translated by 20 pixels in the following directions: 0, +120, and -120 degrees, which correspond to 0, 2*pi/3, and -2*pi/3 in radians. The following table shows the offsets for translating the circle in the *width* and *height* dimensions.

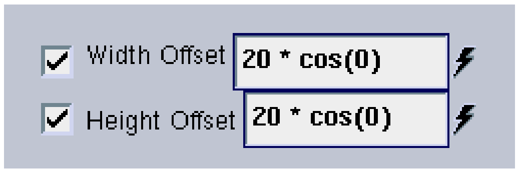
	First image	**Second image**	**Third image**
Width Offset:	`20*cos(0)`	`20*cos(2*pi/3)`	`20*cos(-2*pi/3)`
Height Offset:	`20*sin(0)`	`20*sin(2*pi/3)`	`20*sin(-2*pi/3)`

4. Find the `Translate` operator (*Data Manip,Reorganize Data*). Duplicate the glyph twice and place all three glyphs one above the other to the right of the `Circle Image` glyph. Connect their inputs to the output of the `Circle Image` glyph.



Expressions and functions in panes

5. Open their panes and modify the offset parameters according to the table shown above. Remember that the parameter field can accept expressions involving functions such as `cos` and `sin` and special constant values such as `pi` ($\pi = 3.1416$).

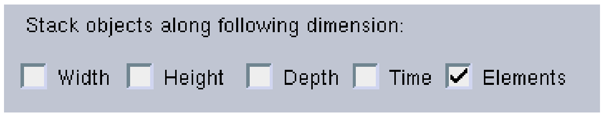


[✓ Width Offset] 20 * cos(0)
[✓ Height Offset] 20 * cos(0)

Appending 2D images

Next you will stack together the monochrome images, by using the `Append` glyph.

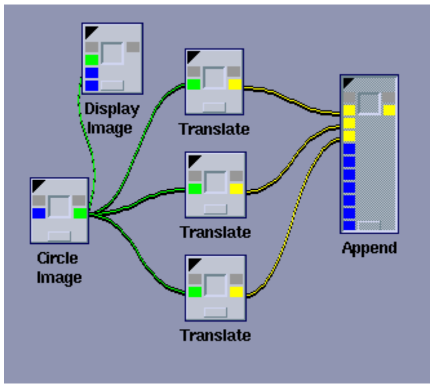
6. Find the `Append` glyph (*Data Manip, Size & Region Operators*) and place it to the right of the three `Translate` glyphs. In its pane, select *Elements* to stack the images along the elements dimension.



Stack objects along following dimension:

Width Height Depth Time [✓ Elements]

7. Connect the output of the three `Translate` glyphs to the first three inputs of the `Append` glyph. Run the three `Tranlate` and the `Append` glyphs.



Stacking images

8. Use the `Data Object Info` glyph (*Input/Output, Information*) to verify that the output image of the `Append` glyph has three elements of size 128 x 128 and no color model attribute. An excerpt of the output of the `Data Object Info` glyph is shown below.

```
# Color Space Model:  0 (invalid)

# -- Value Data --
# Data Type:  Unsigned Byte (4)
# Size:  Width=128, Height=128, Depth=1, Time=1, Elements=3
```

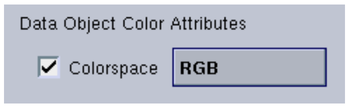
Visualizing with Animate

9. Choose the `Animate` glyph (*Visualization, Interactive Image Display*), connect it to the `Append` glyph, and sequence through the three image frames along the elements dimension. Since there is no color model yet, the image is treated as three independent grey-scale images.

Setting the color model attribute

To interpret this three-element data set as an RGB image, the color model attribute must be set to the *RGB Color Model*.

10. Locate the `Set Attribute` glyph (*Data Manip, Object Attributes*). Place it to the right of the `Append` glyph and connect its input to the output of the `Append` glyph. Open the pane and at its lowest portion, enable the *Colorspace* option and select `RGB` in the associated pulldown menu. Close the pane and run it.


Data Object Color Attributes

☑ Colorspace RGB

Tip: The size of the `Set Attribute` pane is large. If you have difficulty in closing its pane because the $\boxed{\text{Close}}$ button is not on the screen, click again on the top-left corner of the glyph.

Primitive color combination

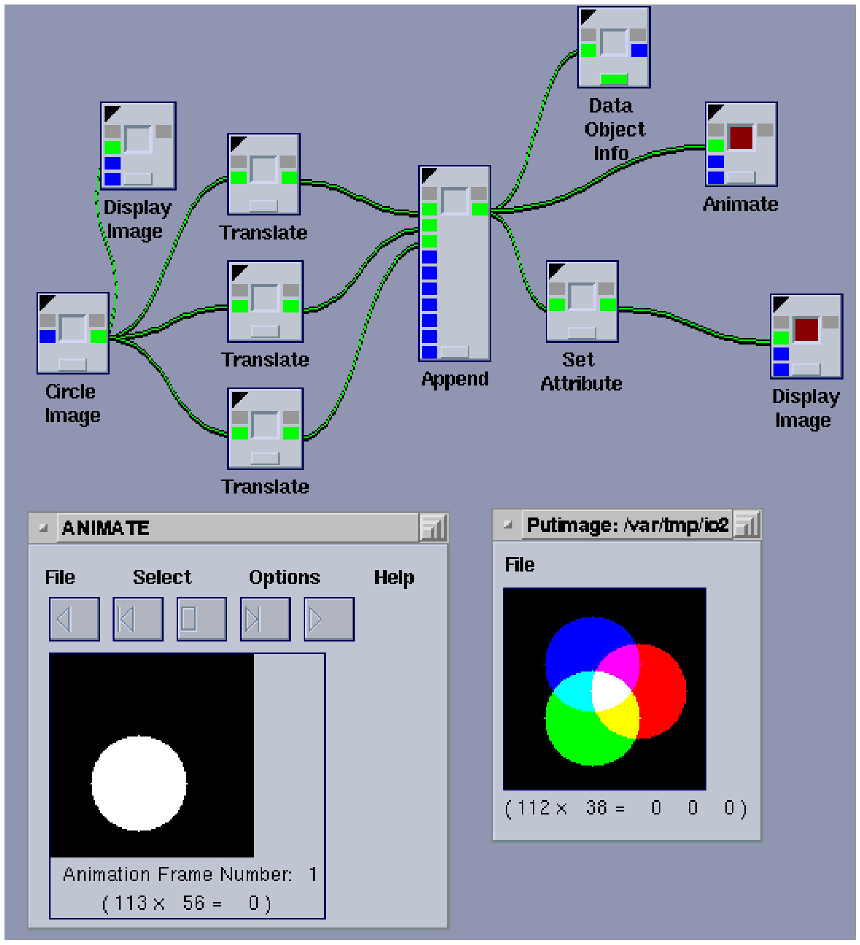
11. Visualize this new image with a duplicate copy of the `Display Image` glyph connected to the append glyph. Navigate with the mouse over the displayed image and observe that there are three pixel values (R, G, and B) for each coordinate. Each color can be a point in the RGB color model. Red, green, and blue are known as the primary colors. These colors can

be combined to produce secondary colors. By placing your cursor over the image, you can identify the following colors:

Color	**Red**	**Green**	**Blue**
Black	0	0	0
Blue	0	0	255
Green	0	255	0
Red	255	0	0
Yellow	255	255	0
Magenta	255	0	255
Cyan	0	255	255
White	255	255	255

Tip: If the text window of the `Display Image` window is not wide enough to show the pixel values, resize the window.

The final workspace for this exercise is shown below.



12. Save the workspace under the name `color-circles-RGB.wk`.

4.2 Other Segments in the Polymorphic Data Model

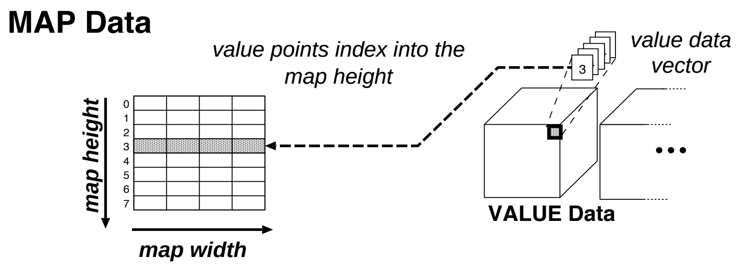
As mentioned before, earlier examples of the Polymorphic Data Model focused on the “value” data segment. We will now examine the “Map” segment. In other image data models, the map segment is often referred to as the colortable.

4.3 Map Segment

The pixel values of a digital image can be a single scalar value or a vector with its values normally stored in the elements dimension. Recall that in RGB color images, each pixel has three elements, corresponding to the colors red, green, and blue.

True Color and Indexed Color

There are two data schemes used to represent the pixel values within an image. One is called "True Color," when it uses the explicit color values in the value segment, and the second method is called "Indexed Color," when each pixel value is an index to a table in the map segment. Each entry in the table can be associated with one or more values. The length of the table can be any size, depending on the index values within the image. This table, stored in the map segment, has many different names, such as colortable, color palette, LUT (look-up table), mapping table, colormap, etc.



The advantages of using the map segment are mainly three:

Image Compression. In a True Color image in the RGB model, each pixel requires three values for storage. If a colortable is used, each pixel requires one index value and the map segment requires a table of *width* 3 and *height* given by the number of different colors in the image. An Indexed Color image in the RGB model, represented by a map segment, is approximately 1/3 the size of the correspondent image stored in the True Color format.

Processing Speed. If an image is represented by using the map segment, all pointwise operations that use a single operand can be executed by processing the table in the map segment. The map segment is in general much smaller than the image.

Hardware Support. Most color monitors and frame grabbers have a colortable in hardware so that the colortable structure closely matches the hardware model, allowing single operand pointwise operations to be processed in hardware.

Experimenting with map segments

In the following experiment, we will create an Indexed Color image, by modifying the previous `color-circles-RGB.wk` workspace. The idea is to build the three monochrome images to a single index image with 8 colors. We then manually build an ASCII colortable to be used as the map segment of the Indexed Color image.

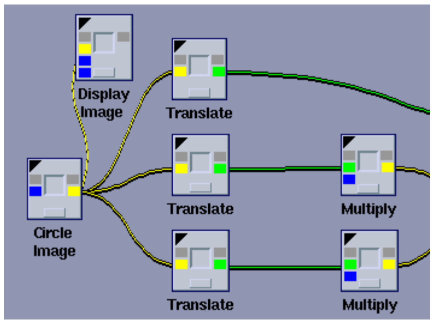
The workspace described in the following steps is shown at the end of this section. You may refer to it to help in its creation.

1. Reload the `color-circles-RGB.wk` workspace after clearing the previous workspace.
2. Delete the `Data Object Info`, `Animate`, `Set Attribute`, and `Append` glyphs. You can delete a glyph by selecting the glyph and then clicking *Delete* in the *Edit* menu (short-cut <`Ctrl-B`>).
3. In the pane of the `Circle Image` glyph, change the *Foreground Level* to `1`.

Multiplying the images

Multiply the pixel values of the second translated image by 2 and the third translated image by 4. The reason is to get a final combined image with 8 different pixel values given by the combination of weighting the three circles by 1, 2, and 4.

4. Select the `Multiply` glyph (*Arithmetic, Two Operand Arithmetic*) and place it to the right of the second `Translate` glyph. In its pane, set the parameter *Real Constant* to 2, and connect the first input of the `Multiply` glyph to the second `Translate` glyph.
5. Duplicate the `Multiply` glyph, place it below the previous `Multiply` glyph, and connect it to the third `Translate` glyph. Change its *Real Constant* parameter to `4`.



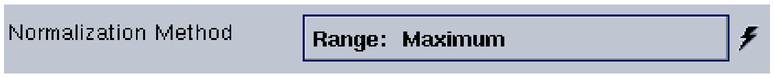
Adding the images together

6. Find the `Add` glyph (*Arithmetic, Two Operand Arithmetic*) and place it above and to the right of the first `Multiply` glyph. Connect the first input

to the first `Translate` glyph and the second input to the first `Multiply` glyph.

7. Duplicate the `Add` glyph and place it to the right of the first `Add` glyph. Connect its first input to the `Add` glyph and the second input to the second `Multiply` glyph. Connect the output of this `Add` glyph to the `Display Image` glyph that was already in the workspace.

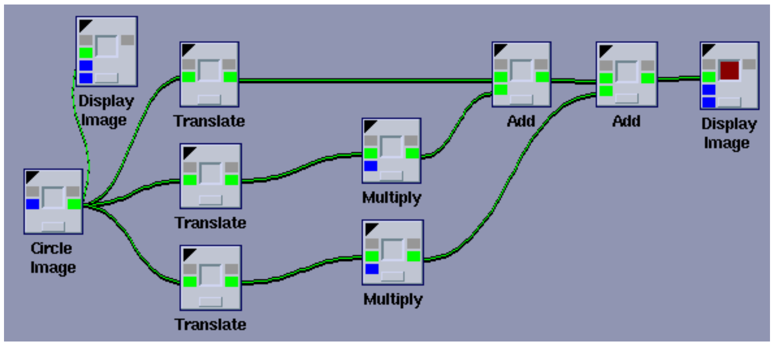
8. Open the pane of the `Display Image` glyph, and set the *Normalization Method* to `Range: Maximum`.



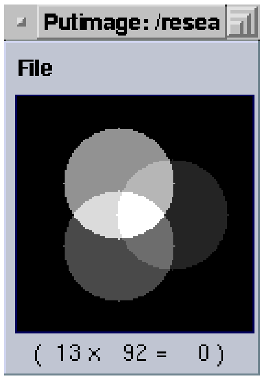
Normalization Method Range: Maximum

9. Save the workspace as `greyscale-circles.wk` before going on to the next part of the experiment.

The grey-scale image with eight grey-levels



10. Run the workspace. The final displayed image is a grey-scale image with 8 different pixel values: 0 to 7. To enhance the displayed image, click on its $\boxed{\text{Options}}$ button and select `-3*(std.dev) <= values <= 3*(std.dev)` as the *Normalization Method*.



The purpose of this experiment is to use the pixel values of the grey-scale image just created as an index to a colortable. The colortable must have 8 entries and each entry must have three values, the red, green and blue components of the indexed color. Next, the colortable will be created and then used in the grey-scale image.

Creating an ASCII colortable

The colortable will be built from an ASCII image with size of *width = 3* and *height = 8*.

11. Create an ASCII colortable file named `map8.txt` by using a text editor such as `xedit`.

```
%xedit map8.txt
```

12. Enter the following pattern:

0	0	0
255	0	0
0	255	0
255	255	0
0	0	255
255	0	255
0	255	255
255	255	255

13. Save it and quit `xedit`.

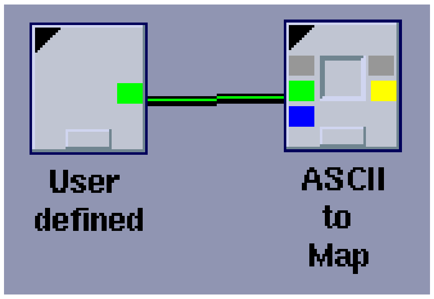
Reading the ASCII colortable in the map segment

14. Find the `User defined` glyph (*Input/Output, Data Files*) and place it below the `Circle Image` glyph. In its pane use the file browser tool to select the ASCII file `map8.txt` created by the text editor.

15. Find the `ASCII to Map` glyph (*Input/Output, Import Data*), place it to the right of the `User defined` glyph and connect to it. In its pane change the following parameters:

Parameter	**Value**
MAP WIDTH:	3
MAP HEIGHT:	8

This `ASCII to Map` glyph reads the ASCII file and stores the table in the map segment of the VisiQuest Polymorphic Data Model.



Checking the data header of a map segment data file

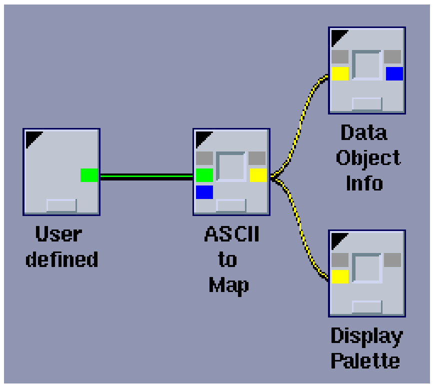
16. Use the `Data Object Info` glyph (*Input/Output, Information*) to verify how this segment is stored in the KDF data.

    An excerpt of the data file header is given below. Note that the data file has just one segment, identified as "Map Data" with *width = 3* and *height = 8*.

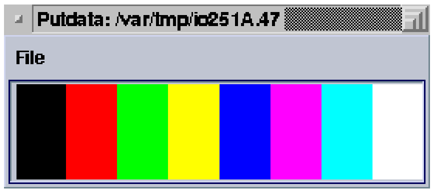
```
# Storage Format:  kdf
# -- Map Data --
# Data Type:  Double (1024)
# Size:  Width = 3, Height = 8, Elements = 1,
# Depth Dimension = 1, Time Dimension = 1
```

Displaying a map segment

17. To display the colors of a map segment, use the `Display Palette` glyph (*Visualization, Color Map Display & Manipulation*) connected to the `ASCII to Map` glyph.



The display shows the 8 colors stored in the map segment. The color indexes are implicit and range from 0 to 7.



Color images and map segment

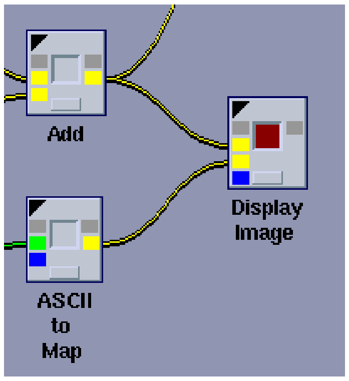
There are three basic ways of dealing with images and map segment:

- Use the map segment as a color map in the display. This is an Index Color image with an external map segment.
- Store the map segment in the same file together with the value segment. This is a *stand-alone* Index Color image.
- Explicitly map the image pixel into the colortable, resulting in a True Color image, with no map segment.

In the following steps, each one of these three ways is used.

Displaying an image with an external map segment

18. Duplicate the `Display Image` glyph and place it below the second `Add` glyph. Connect its first input to the output of this `Add` glyph and connect the second input, which is the *Input Color Map* parameter, to the output of the `ASCII to Map` glyph.



19. Execute the glyph and verify in the displayed image that the image is now colored with the imported map segment as the table for the indexed image. Note that the pixel values displayed by the `Display Image` glyph are the pixel values stored in the value segment. Connecting it directly to the *input colormap* of the visualization tools is the first use of the map segment.

    Tip: You may need to rerun the `Display Image` glyph if the workspace is in the ready mode.

Creating an image with value and map segment

20. Find the `Insert Segments` glyph (*Data Manip, Segment Operators*) and place it below and to the right of the `ASCII to Map` glyph. Connect its first input to the output of the second `Add` glyph and the second input to the `ASCII to Map` glyph. This copies the value segment of the grey-level image with the map segment, generating a new data file which contains a value segment and a map segment.

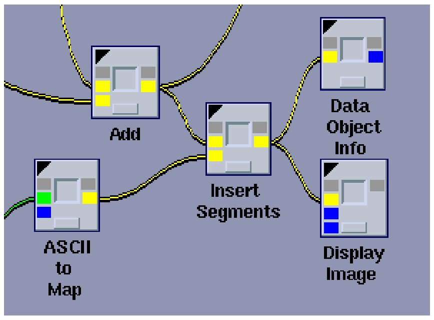
Data Header of an image with value and map segment

21. Use a duplicated copy of the `Data Object Info` glyph to confirm the header information of the output data of the `Insert Segments` glyph. The file is stored in the KDF format, its value segment is of size (128 x 128), and its map segment is of size *width = 3* and *height = 8*.

```
# Storage Format:  kdf
# -- Value Data --
# Data Type:  Unsigned Byte (4)
# Size:  Width=128, Height=128, Depth=1, Time=1, Elements=1
# -- Map Data --
# Data Type:  Double (1024)
# Size:  Width = 3, Height = 8, Elements = 1,
# Depth Dimension = 1, Time Dimension = 1
```

Displaying an image with the map segment

22. Duplicate the `Display Image` glyph and place it to the right of the `Insert Segments` glyph. Connect the first input to it.



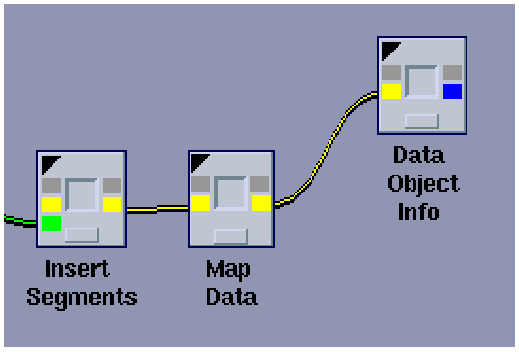
23. Execute the `Display Image` glyph and move the mouse over the displayed image. You will notice that the pixel values come from the value segment as before. That is because the map segment is being used to dictate the displayed color. An image with the value and the map segment together is the second use of the map segment. The image created is an Indexed Color image.

Creating a True Color image from an image with the map segment

It is possible to apply the table stored in the map segment of an image to create another image in which the pixel indexes are explicitly mapped through the map table. This converts an Indexed Color image into a True Color image.

24. Find the `Map Data` glyph (*Data Manip, Map Operators*) and connect its input to the `Insert Segments` glyph.

25. Duplicate the `Data Object Info` glyph and connect it to the `Map Data` glyph.



26. Execute the `Data Object Info` glyph to print the header information of the True Color image to understand the difference of this new data file. The image has only a value segment with a new size of 128 x 128 by 3 elements. Note that storing this image requires nearly three times the space of the image with value and map segments.

```
# Storage Format:  kdf
# Color Space Model:  0 (invalid)
# -- Value Data --
# Data Type:  Double (1024)
# Size:  Width=128, Height=128, Depth=1, Time=1, Elements=3
```

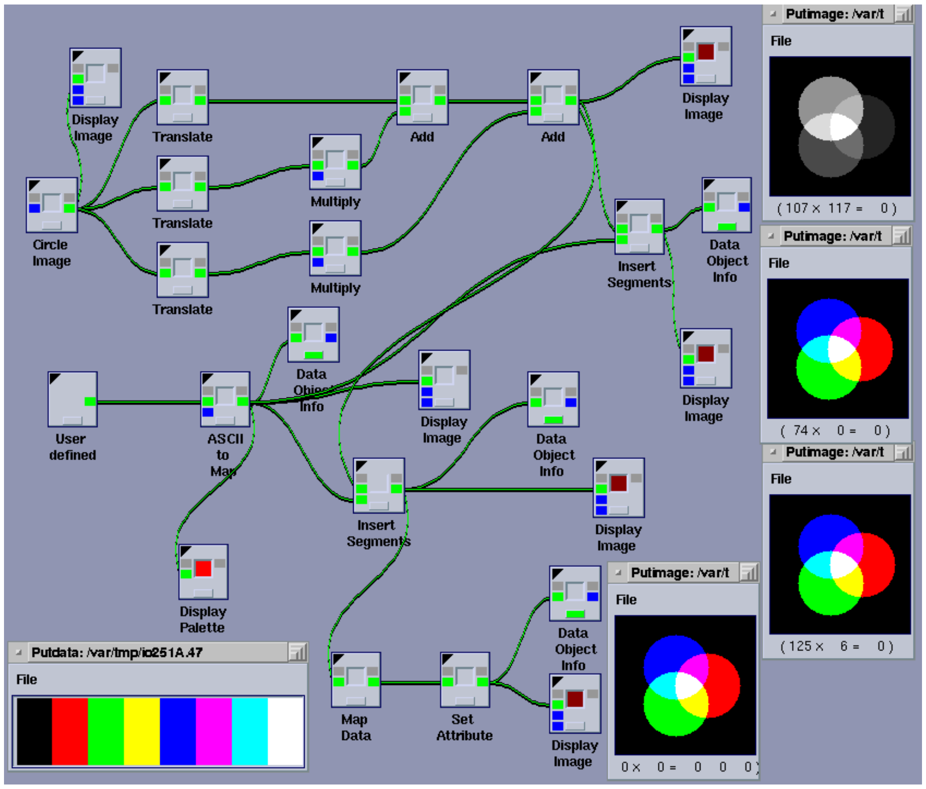
Displaying the True Color image

In the header information of the map image above, the color model is invalid. To display this image as an RGB image, you set the attribute color to the RGB model, following the same procedure used in section 4.1.

27. Find the `Set Attribute` glyph (*Data Manip, Object Attributes*). Place it to the right of the `Map Data` glyph and connect its input to the `Map Data` glyph. In its pane, enable the *Colorspace* option and select `RGB`.

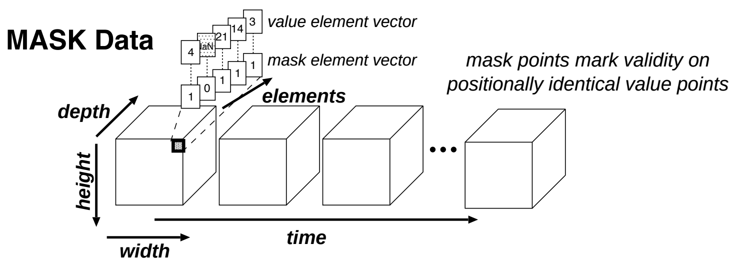
28. Duplicate the `Display Image` glyph and display the output image of the `Set Attribute` glyph. It is a True Color image. Move the mouse over the pixels of the displayed image and note that there are three pixel values (R, G, and B) associated with its pixel coordinates.

29. Save the workspace under the name `color-circles-RGB-map.wk`. A screen dump of the complete workspace and the created images can be seen below.



4.4 Mask Segment

Another segment in the Polymorphic Data Model is the mask segment. Its function is to identify pixels in the value segment which are invalid. The size of the mask segment is the same as the value segment because there is a one-to-one correspondence between every pixel in the value segment and every pixel in the mask segment. A mask pixel value of 0 indicates that the correspondent pixel in the value segment is invalid; otherwise the pixel is valid. The mask segment can be used whenever the image is not rectangular. Typical examples of non-rectangular images are found in ultrasound and CT (computer tomography) systems. Another application of the mask segment is when a non-rectangular *Region Of Interest* (ROI) is selected in an image.



Building the workspace to experiment with the mask segment

In the workspace described below, the tool `Extract ROI` is used to manually extract a Region Of Interest in an ultrasound image. This tool masks the data outside of the delimited area. Compare the mean value computed by the `Statistics` tool when applied to the whole image and to the ROI image with a mask associated with it.

1. Clear the previous workspace and find the `Images (Misc)` glyph (*Input/Output, Data Files*). In its pane, select the image `Ultrasound of Infant Head`.

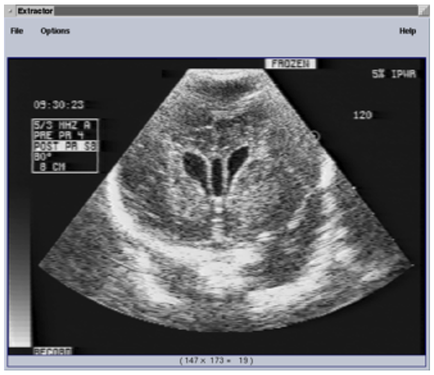
Interactive ROI extraction tool

2. Choose the `Extract ROI` glyph (*Visualization, Interactive ROI Extraction*), place it to the right of the `Images (Misc)` glyph, and connect its input to it. In its pane, select the parameter *ROI Shape* as `Freehand` to activate the Region Of Interest that is defined interactively by freehand drawing.

3. Find the `Data Object Info` glyph (*Input/Output, Information*) and connect it to the output of the `Extract ROI` glyph.

Selecting a Region Of Interest

To extract a freehand Region Of Interest, run the `Extract ROI` glyph and, on the displayed image, begin by clicking the left mouse button on a start point of the shape to be drawn. While holding the button down, move the mouse around the desired shape. A line is drawn as the mouse moves. When the mouse button is released, a line connecting the start point automatically closes the contour and the extracted region is displayed in the *extracted* window and an output file created.



4. Run the `Extract ROI` glyph to extract the ultrasound cone-shaped image with the freehand drawing tool. If you are not satisfied with the contour you can redraw it and the Region Of Interest will be updated in the *extracted* window.

5. Exit the `Extract ROI` tool by clicking on the $\boxed{\text{Quit}}$ button.

   Note: Do not exit by clicking on the glyph $\boxed{\text{Run}}$ button again, as this makes the extract file invalid.

Verifying the header of the extracted image

6. Run the `Data Object Info` glyph to see how the extracted data is stored. An excerpt of the data header output is shown below.

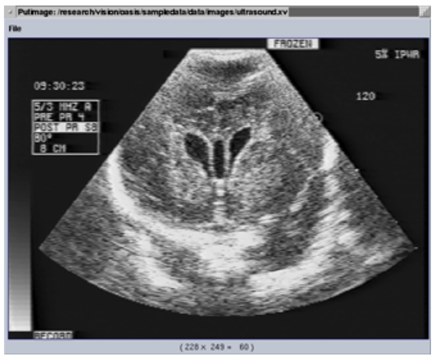
```
# Storage Format:  kdf
# Color Space Model:  AccuSoftGB
# -- Value Data --
# Data Type:  Unsigned Byte (4)
# Size:  Width=564, Height=404, Depth=1, Time=1, Elements=1
# -- Mask Data --
# Data Type:  Unsigned Byte (4)
# Size:  Width=564, Height=404, Depth=1, Time=1, Elements=1
# Masked Value Presentation:  Use Actual Values
# -- Map Data --
# Data Type:  Unsigned Byte (4)
# Size:  Width = 3, Height = 256, Elements = 1,
# Depth Dimension = 1, Time Dimension = 1
```

Note that the data file has three data segments: value, mask, and map. The mask segment has the same size as the map segment, which is 564 x 404 pixels[1]. The reason this output data file also has a map segment is because the ROI extractor tool copies the image that is actually displayed and every displayed image has a color map. In this case the color map is a grey-level color map. The size of the map is *width = 3* and *height = 256*, to represent the 256 grey-tones of an image with pixel data type `Unsigned Byte`.

However, grey-level images do not need a map segment; the shade of grey is implied by the pixel value. Therefore, remove the map segment by creating a `Remove Segments` glyph (*Data Manip, Segment Operators*) and opening its pane to specify that it is the map segment that is to be removed.

Displaying a masked image

7. Find the `Display Image` glyph (*Visualization, Non-Interactive Image Display*) and connect its first input to the output of the `Extract ROI` glyph. Execute it and verify that there are pixels whose values are displayed and pixels whose values are not displayed, by moving the mouse over the displayed image. The pixels that are not shown are the masked pixels.



Making measurements on masked images

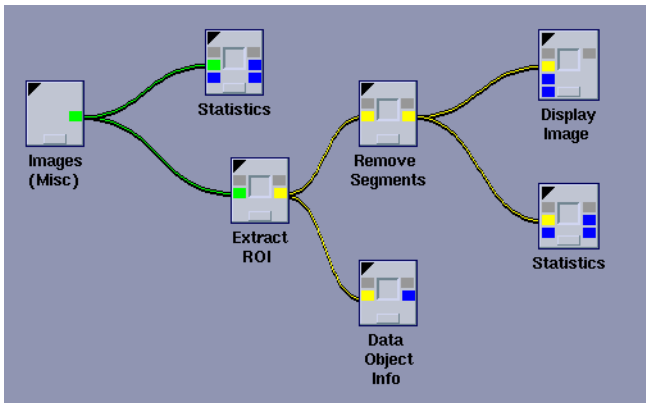
Compare the mean grey-value of the original ultrasound image and of the ROI extracted image.

8. Use two copies of the `Statistics` glyph (*Data Manip, Analysis & Information*), one connected to the original image (`Images (Misc)`) and the other connected to the `Extract ROI` glyph.

[1]This size may be different depending on how large you extract the ROI

The mean value of the original image is 80.9253, while the mean value of the masked image is around 130. The reason the mean value is higher in the masked image is that the black portion of the image around the cone-shaped ultrasound image is not considered in the mean computation of the masked image.

9. Save the workspace under the name `mask-ultrasound.wk`.



Conclusion

In this chapter you have experimented with the map and the mask segments of the Polymorphic Data Model. You created a colored image in the RGB model to understand the advantages of representing a colored image using the map segment. You worked with the interactive `Extract ROI` glyph and learned how a mask segment is used to define a non-rectangular Region Of Interest in images.

The next chapter introduces you to advanced aspects of `VisiQuest` programming such as procedures and flow-control glyphs.

Chapter 5

Advanced VisiQuest Programming

5.1 Procedures

Similar in concept to a subroutine in a textual programming language, a visual programming procedure allows you to modularize a visual program in a hierarchical manner. Procedures promote the readability of a visual program. The use of procedures is more effective with large and complex workspaces.

Creating a procedure

In this section you will learn the fundamentals of procedures. The experiment will be based on the `threshold-mean.wk` workspace. The purpose of this procedure is to threshold an image by a value given by the image mean added by a parameter K multiplied by the image variance:

```
threshold = mean + K * variance
```

where `K` is the procedure input parameter.

1. Clear the previous workspace and restore the `threshold-mean.wk` workspace created in section 3.8. In the pane of the `Print Stats` glyph, enable the *standard deviation variable* to be `std_dev`. With this change, the workspace has two variables: `mean` and `std_dev`.

   The threshold value will depend on the two variables above and on the input parameter `K` of the procedure. First you will use a fixed value for `K` of 0.1. Later you will modify the workspace to incorporate the parameter `K`.

2. In the ">"pane, change the *constant* value to `mean - 0.1 * std_dev`.

3. Reset and run the workspace to verify if it is working properly. You can verify that the workspace is working if the parameter used for the ">"glyph is 80.6146.

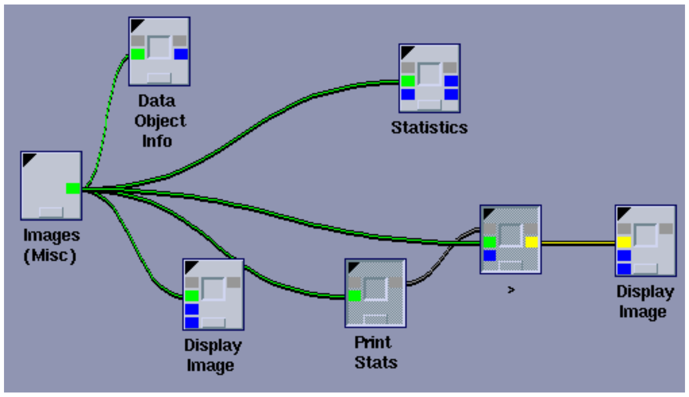
Tip: You can verify the value used in the ">"glyph, by looking at its equivalent command line that appears in the `VisiQuest` output console as the glyph is executed. The command line equivalence of the ">"glyph is `kcompare -gt`. The output should be something like

`$DATAMANIPBIN/kcompare -gt ... -real 80.6146` [1].

4. Put the workspace in stop mode by clicking on the $\boxed{\text{Stop}}$ VisiQuest icon.

Selecting glyphs for the procedure

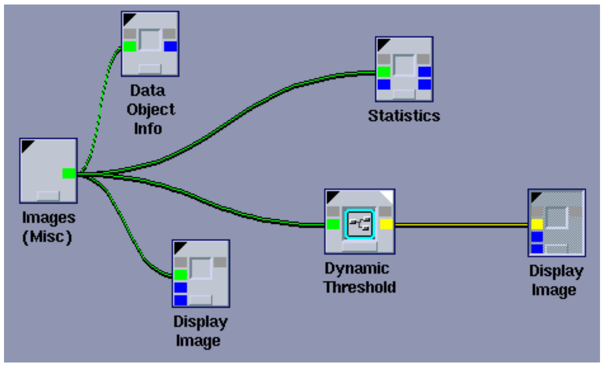
5. Create the procedure by selecting the `Print Stats` and the ">"glyphs. To select more than one glyph simultaneously, either press the <`Shift`> key of the keyboard while clicking on the glyph, or create a rubber-band rectangle around the glyphs by holding the left mouse button down while dragging the mouse to cover the area containing the glyphs.



Procedure creation

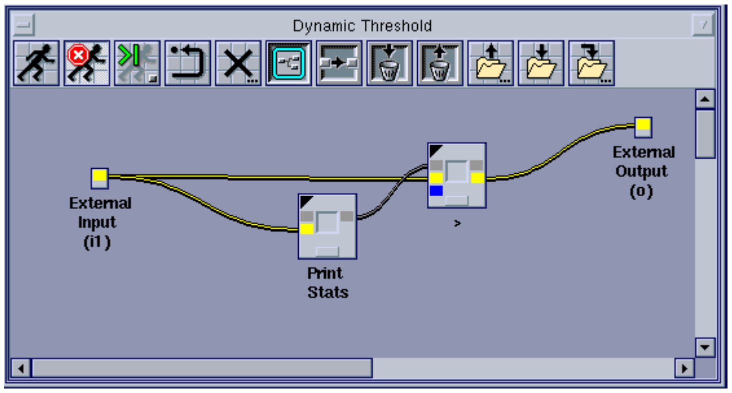
6. Select *Create Procedure* from the *Control* pulldown menu. The selected glyphs are now replaced by the procedure. Change the name of the procedure glyph to `Dynamic Threshold` by clicking on the procedure name.

[1]This value is obtained when processing the gull image



7. Run the procedure glyph by clicking on the middle square button as you run the normal glyphs.

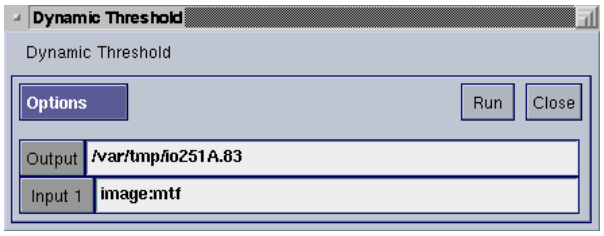
8. View the contents of the procedure glyph by clicking on the white triangle at the top-right corner of the procedure glyph. The glyphs of the procedure appear in an inner level of the workspace. The external input and output connections are represented by small boxes. **Tip:** You may want to move the external input and output connections to the right and left, respectively, if they are bunched up together or obscured by the glyphs when the procedure is created.



9. Close the procedure by clicking on the top-left icon in the procedure workspace.

   Notice that the procedure glyph has a pane like any other glyph.

10. Open its pane to see that the input and output file parameters are present in the procedure pane.



/var/tmp/io251A.83

image:mtf

11. While the input and output file parameters are present, they may not have been placed on the procedure pane the way you would like. Put the procedure pane in *edit mode* by holding down the shift key while you click the left mouse button in the procedure pane. Use the mouse to drag the output file parameter to the bottom, and arrange the procedure pane so that the input file and the output file are in order.

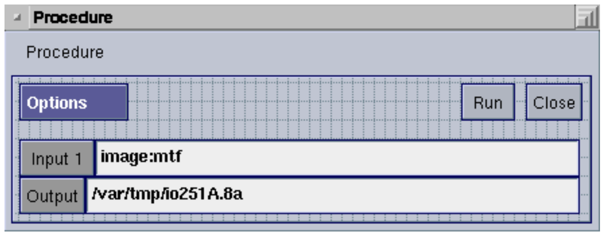
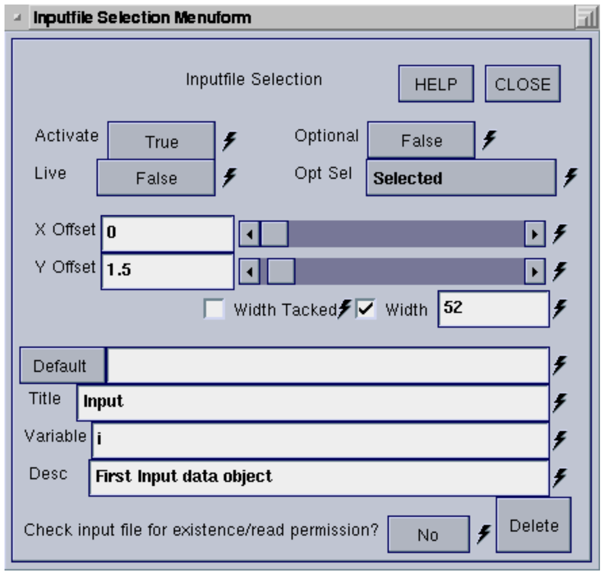


image:mtf

/var/tmp/io251A.8a

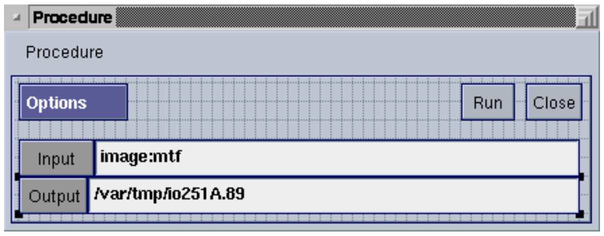
12. Now, display the *menuform* of the input file parameter by clicking on it.



Activate: True
Optional: False
Live: False
Opt Sel: Selected
X Offset: 0
Y Offset: 1.5
Width Tacked: ☐
Width: ☑ 52
Title: Input
Variable: i
Desc: First Input data object
Check input file for existence/read permission? No

13. Change the variable of the input file from `i1` to `i` and the title of the input

file parameter from Input1 to Input.



Input: image:mtf

Output: /var/tmp/io251A.89

Inserting parameters in the procedure pane

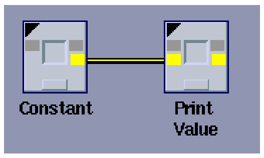
So far, the procedure implemented the equation `mean - 0.1 * std_dev` for the thresholding. To change the equation so that the constant `0.1` is replaced by a procedure input parameter, use the *Export GUI* mechanism. With this mechanism, it is possible to export any parameter of any glyph in the procedure to the procedure pane.

Before using this mechanism, rearrange the procedure so that the parameter `-0.1` can be given as a parameter in a glyph. The easiest process is to create a single pixel image whose pixel value is given by the parameter that will be exported to the procedure pane. When this is done, you read this pixel value and assign it to a variable, such as `K`, that will be used in the new expression for the ">"glyph.

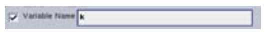
14. Open the procedure by clicking on the top-right corner of the procedure glyph.

15. Find the `Constant` glyph (*Input/Output, Generate Data*) and place it below the `Print Stats` glyph. In its pane, set the *Real Constant Level* to `-0.1`. When this glyph is executed it generates a single pixel image with the value `-0.1`. This is the parameter that will be exported to the procedure pane.

16. Find the `Print Value` glyph (*Data Manip, Analysis & Information*) and place it to the right of the `Constant` glyph, then connect its input to the output of the `Constant` glyph.



17. In the pane of the `Print Value` glyph, set the parameter *Variable Name* to `K`. When this glyph is executed it reads the first pixel of the input image and assigns its value to the variable `K`.



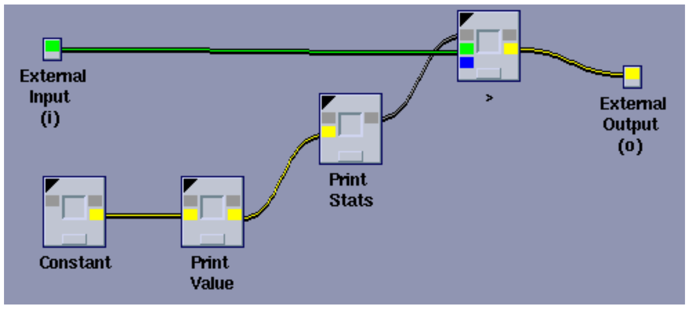
Variable Name K

18. Now, change the expression in the ">"glyph pane to include the variable `K`. In its parameter *Constant*, change the expression to `mean + K * std_dev`.

    To finish the procedure, the control connections should be rearranged so that the variables `K`, `mean`, and `std_dev` have their values properly assigned before the execution of the ">"glyph.

19. Connect the output control connection of the `Print Value` glyph to the input control connection of the `Print Stats` glyph.

    The final procedure workspace is shown below.



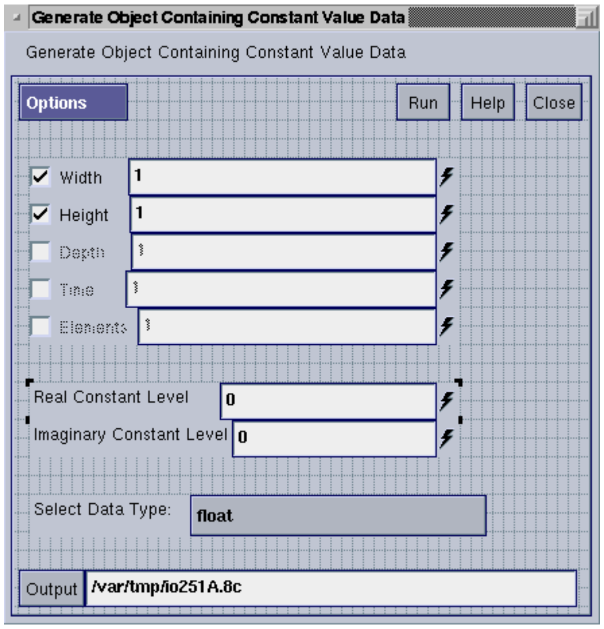
Exporting a parameter to the procedure pane

At this stage the procedure is properly modified so that the parameter *Real Constant Level* of the `Constant` glyph can be exported to the procedure pane.

20. Open the pane of the `Constant` glyph.

21. Put the pane in "edit mode" by selecting the *GUI Editing* in the *Options* pulldown menu in the pane. The pane background changes to a grid, indicating that it is in "edit mode."

22. Select the *Real Constant Level* parameter by clicking on it. Its four corners indicate that the parameter is selected.



Generate Object Containing Constant Value Data

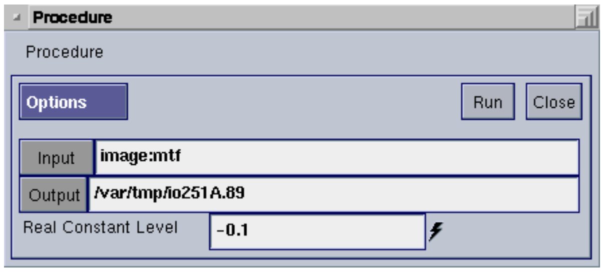
Generate Object Containing Constant Value Data

✓ Width	1
✓ Height	1
Depth	1
Time	1
Elements	1
Real Constant Level	0
Imaginary Constant Level	0
Select Data Type:	float
Output	/var/tmp/io251A.8c

23. In the *Options* pulldown menu in the pane, select *Export to Workspace GUI*. This operation exports the selected parameter to the procedure pane. Finally, close the pane.

24. Close the procedure by clicking on the top-left corner icon of the procedure workspace.

25. Open the procedure pane and verify that the *Real Constant Level* was properly exported.



Procedure

Procedure

Input	image:mtf
Output	/var/tmp/io251A.89
Real Constant Level	-0.1

26. Run the workspace and verify that it is working properly.

27. Save this workspace under the name `threshold-procedure.wk`.

In the software development chapter you will learn how to create a compiled workspace based on the procedure created here. The compiled workspace is a glyph installed in `VisiQuest` which has all the functionality of a procedure.

5.2 Flow Control

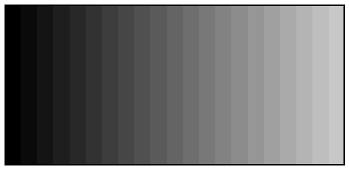
`VisiQuest` has other types of glyphs that are used for data-flow control in the workspace. With the flow-control glyphs it is possible to create programs with repetitive loops.

Loop example in VisiQuest

A prototype example of a repetitive loop is available in the SAMPLEDATA toolbox under the alias `workspaces:Feedback`. This workspace is a model to create loops in `VisiQuest`. You need a loop in the workspace whenever you generate data iteratively. In the `Feedback` workspace an image is generated by iteratively concatenating vertical bars of increasing grey-values.

1. Clear the previous workspace and load the `workspaces:Feedback` workspace.

   The output generated by this workspace is an image with 21 vertical constant rectangles of increasing values varying from 0 on the first rectangle on the left side of the image to 200 on the rectangle on the right side of the image. This image is shown below.

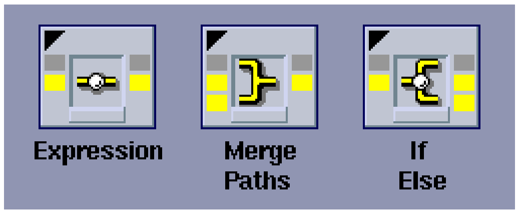


   The general idea of the algorithm implemented in the workspace is to iteratively prepend a rectangular constant image to the right side of the image. The constant value used is increased at each iteration by 10 until 200 is reached.

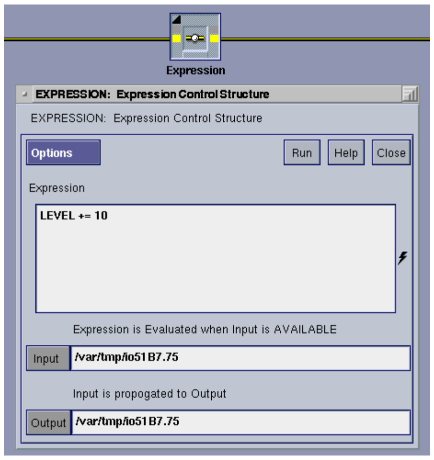
   The regular processing glyphs required in the workspace are two: the `Constant` glyph, to generate a constant rectangle image, and the `Append` glyph, to attach two images sideways.

Flow-control glyphs

The required flow-control glyphs are the `Expression` glyph, the `Merge Paths` glyph, and the `If Else` glyph.



The `Expression` glyph copies the input data to the output data and evaluates its associated expression given in its pane. This glyph is used in two places in the `Feedback` workspace. One place is in the initialization part of the workspace to set the iterate variable LEVEL to 0, and the other is in the loop itself to increment LEVEL by 10.



EXPRESSION: Expression Control Structure

Expression

```
LEVEL += 10
```

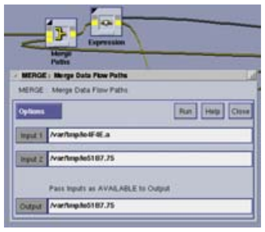
Expression is Evaluated when Input is AVAILABLE

Input: /var/tmp/io51B7.75

Input is propogated to Output

Output: /var/tmp/io51B7.75

The `Merge Paths` glyph is the point where the data cycles back in the loop. It has two inputs and one output. The output data is copied from the most recent input. In the workspace, the first input comes from the initialization expression and the second input comes from the feedback data flow, where the data is accumulated iteratively.



MERGE : Merge Data Flow Paths

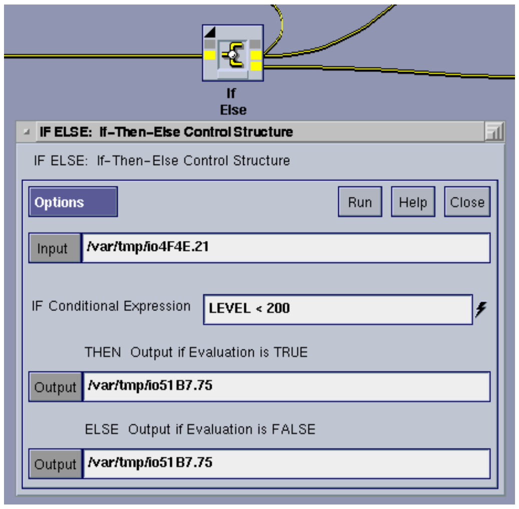
Input 1: /var/tmp/io4F4E.a

Input 2: /var/tmp/io51B7.75

Pass Inputs as AVAILABLE to Output

Output: /var/tmp/io51B7.75

The `If Else` glyph is at the end of the loop. It has one input, two outputs, and a conditional expression. If the conditional expression is true, the input is copied to the first output; otherwise it is copied to the second output. In the case of the `Feedback` workspace, the conditional expression is `LEVEL` < `200`, making the iterative process stop when LEVEL reaches the value 200.



IF ELSE: If-Then-Else Control Structure

IF ELSE: If-Then-Else Control Structure

Input /var/tmp/io4F4E.21

IF Conditional Expression LEVEL < 200

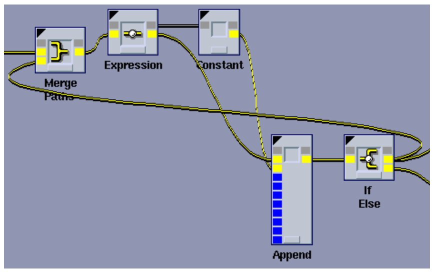
THEN Output if Evaluation is TRUE

Output /var/tmp/io51B7.75

ELSE Output if Evaluation is FALSE

Output /var/tmp/io51B7.75

The part of the `workspaces:Feedback` workspace related to the feedback flow-control loop was rearranged and shown below to highlight the functionality of each glyph.

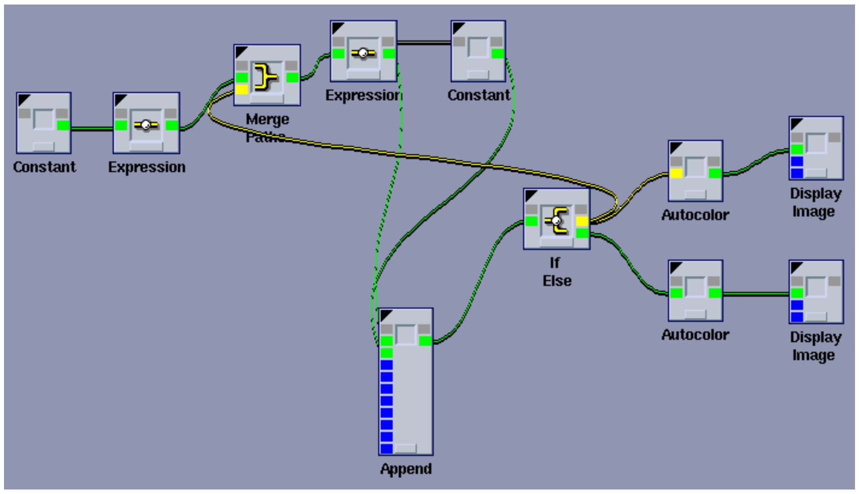


The regular glyphs used in the inner part of the loop are the `Constant` glyph and the `Append` glyph. The `Constant` glyph creates a rectangle with constant value given by the variable LEVEL. The `Append` glyph places the rectangle image generated by the `Constant` glyph on the right of the image that comes from the loop path.

Two control connections in the loop path are required in this example. The

Constant glyph must be executed after the variable LEVEL is iterated by the Expression glyph and the Append glyph must be executed after the Constant glyph is executed.

The workspaces:Feedback in the SAMPLEDATA toolbox is shown below.



The other glyphs presented in this workspace are used to visualize the data in the inner part of the loop and in its output.

2. Execute the workspace and follow the iterative process of the image creation from the visualization glyphs connected to the inner part of the loop.

5.3 Data Transport/Distributed Processing

Data transport refers to the method used to transfer data between processes (operator/glyphs). Earlier, data files were used as the data transport mechanism. VisiQuest supports different mechanisms that can be local or remote. Local transport mechanisms include files, shared memory, memory mapped files, and streams. With the use of remote data transports, the capability of getting input from and output to remote computers is implemented and distributed processing of that data is possible. The only remote transport mechanism supported by VisiQuest is TCP/IP sockets. For detailed information, please refer to the VisiQuest Foundation Services Manual.

Distributed Processing is the ability to specify remote computers on which to execute individual programs. The capability of performing distributed processing of data is implemented via the TCP/IP socket remote data transport

mechanism. This capability is specifically useful with large data sets and for problems that require the number-crunching capability of a supercomputer.
Note: These two features are currently not supported by VisiQuest in the Windows NT operating system; however, all UNIX and Linux version do.

Your computer system dictates which of the data transport mechanisms are available for your use in `VisiQuest`. You can choose the data transport mechanism via `VisiQuest` or the CLUI of each operator. If no data transport mechanism is specified by the user, the VisiQuest transport distribution routines negotiate the transport mechanisms automatically. The syntax used to select a specific transport mechanism is

```
identifier = token
```

The identifier is one of the data transport mechanisms listed earlier. The token is an identifier for that transport. For a file, it is simply the filename. For shared memory, it is the shared memory key. For a pipe, it is the input and output file descriptors, as in `pipe = [3,4]`. For a socket, it is the number of the socket, as in `socket = 5430`.

Data transport mechanisms

This section shows how to change the data transport mechanism of a connection in a workspace.

1. Clear the previous workspace and load the `export-data-gaussian.wk` workspace.

   Select different transport mechanisms between the glyphs.

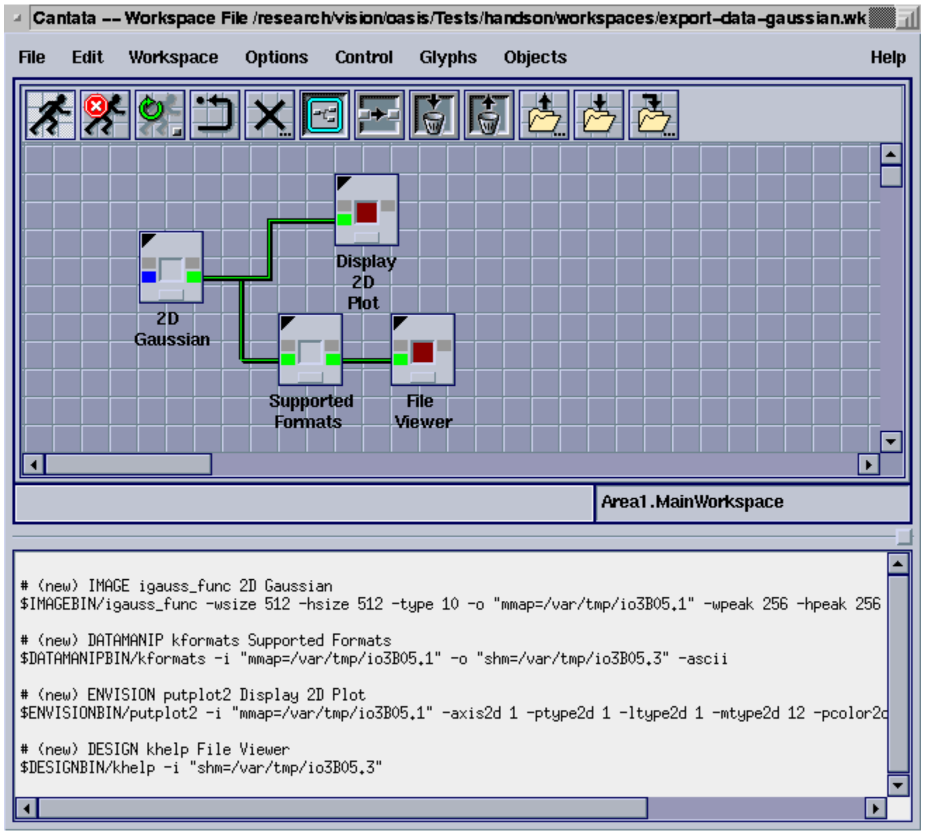
2. Click on the connection that links the `2D Gaussian` and the `Display 2D Plot` glyphs, to bring up a connections menu pane. Select the `Memory Map(mmap)` option and click on the $\boxed{\text{Ok}}$ button. These two glyphs will now exchange data via memory map.

   Note: You can see what data transport mechanism is being used between two glyphs by turning ToolTips on (select *Activate Tooltips* from the *Help* menu) and resting the cursor on a connection.

3. Repeat this procedure for the connection between the `Supported Formats` and `File Viewer` glyphs, but this time select the transport mechanism `Shared Memory(shm)`. The two glyphs will now exchange data via shared memory.

4. Run the workspace and observe the console portion of `VisiQuest` to verify the transport mechanisms being utilized.

Note: The default data transport mechanism for connections in `VisiQuest` is files. You can change this setting in your Preferences file. Select *Preferences* from the *Options* menu to display the Preferences subform. Click on the $\boxed{\text{Workspace}}$ button to display the workspace attributes pane. At the bottom of the pane, you can set the default transport type for new connections. Clicking on Apply Changes installs the new setting for this session of `VisiQuest`. If you then click on Save Preferences, the current preference settings will be re-installed for the next VisiQuest session.



```
# (new) IMAGE igauss_func 2D Gaussian
$IMAGEBIN/igauss_func -wsize 512 -hsize 512 -type 10 -o "mmap=/var/tmp/io3B05.1" -wpeak 256 -hpeak 256

# (new) DATAMANIP kformats Supported Formats
$DATAMANIPBIN/kformats -i "mmap=/var/tmp/io3B05.1" -o "shm=/var/tmp/io3B05.3" -ascii

# (new) ENVISION putplot2 Display 2D Plot
$ENVISIONBIN/putplot2 -i "mmap=/var/tmp/io3B05.1" -axis2d 1 -ptype2d 1 -ltype2d 1 -mtype2d 12 -pcolor2d

# (new) DESIGN khelp File Viewer
$DESIGNBIN/khelp -i "shm=/var/tmp/io3B05.3"
```

Conclusion

In this chapter you created a procedure and learned how to create loops inside `VisiQuest`. The loop example given is a prototype model to be used whenever you program an iterative process inside `VisiQuest`. You have also seen that the files linking the glyphs are not the only data transport mechanisms available. It is an important feature that makes VisiQuest suitable for distributed computing.

In the next chapter, you will develop software in the VisiQuest environment. You will create a toolbox and install an object based on the procedure created in this chapter. You will build a kroutine object written in the C programming language using the software development phases of interface design, coding, installation, and testing.

Chapter 6

Introduction to Software Development in VisiQuest

This chapter introduces the software development tools available in the VisiQuest system. These tools facilitate the development of programs and applications. The software development environment supports the iterative process of developing, documenting, maintaining, delivering, and sharing software. The two principal tools are `craftsman` and `composer`.

`craftsman` is responsible for managing a toolbox. It can create, delete, and copy toolbox objects as well as software objects. `composer` is the user interface to the software objects and can invoke all the operations needed to edit and manage them. For a detailed description of toolbox programming, please refer to the VisiQuest Toolbox Programming Manual.

Setting up your preferred text editor

Before going through the software development experiments, it is important to select a text editor of your choice. The editor will be invoked whenever `craftsman`, `composer`, or `guise` require a file to be edited. The `kconfigure` routine, used to configure the VisiQuest system, does not configure the editor. The editor must be set up later in the `.VisiQuest_env` file generated by `kconfigure`. You can include a line in the `.VisiQuest_env` file defining the VisiQuest_EDITOR environment variable or you can execute the following command at the UNIX prompt to choose the editor as `xedit`:

```
%setenv VisiQuest_EDITOR ¨xedit %f¨
```

If you want another editor or want more information on setting up the text editor, refer to the VisiQuest Installation Guide, section 2.5, “Setting Up Your Environment.”

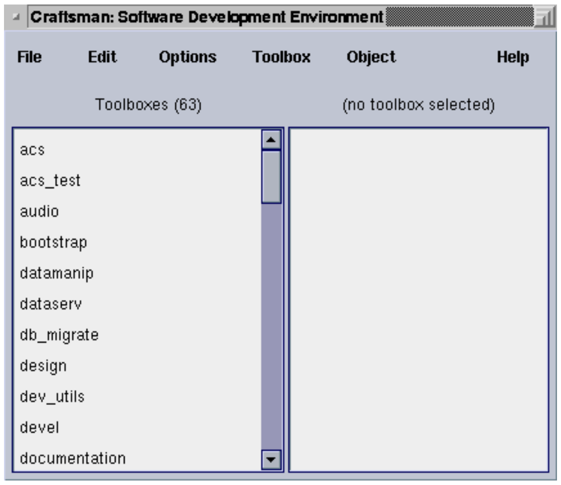
6.1 Creating a Toolbox

Each program of VisiQuest is located in a toolbox. A toolbox is a collection of programs and libraries that are managed as an entity, also known as toolbox objects. It is necessary to have written access to a toolbox to develop software in VisiQuest. In this section, you will create a toolbox under the name HANDSON.

Creating a toolbox

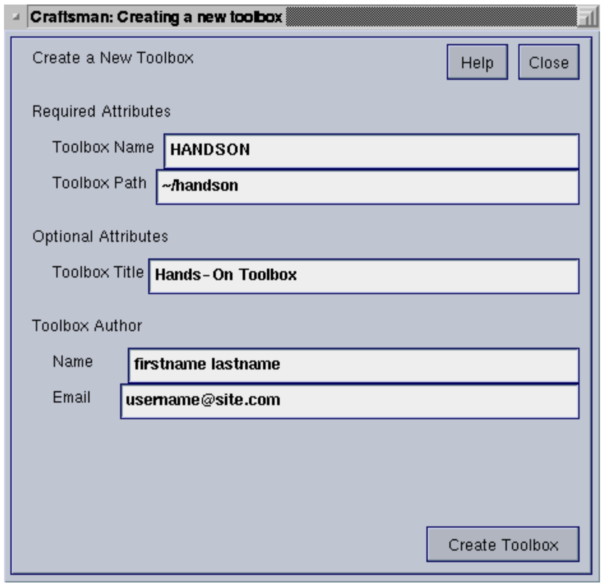
1. Exit `VisiQuest`, if it is running, and execute `craftsman`:

```
%craftsman
```



2. Select the *Toolbox* pulldown menu and choose *Create Toolbox...*. Give your toolbox a name and a path. Specify `handson` as the *Toolbox Name* and ∼`/handson` as the *Toolbox Path*. The " ∼ " symbol indicates the top level of the user home directory structure.

   Note: You need to have written permission to the *Toolbox Path* directory; otherwise an error will be reported.



Create a New Toolbox

Required Attributes

Toolbox Name: HANDSON

Toolbox Path: ~/handson

Optional Attributes

Toolbox Title: Hands-On Toolbox

Toolbox Author

Name: firstname lastname

Email: username@site.com

3. Click on the $\boxed{\text{Create Toolbox}}$ button. A notify window pops up and remains until the toolbox has been created. When `craftsman` has finished creating your toolbox, close the *Create Toolbox* subform and the toolbox just created appears in the scrolled list of *Toolboxes*, preselected for you.

4. Finally, exit `craftsman`.

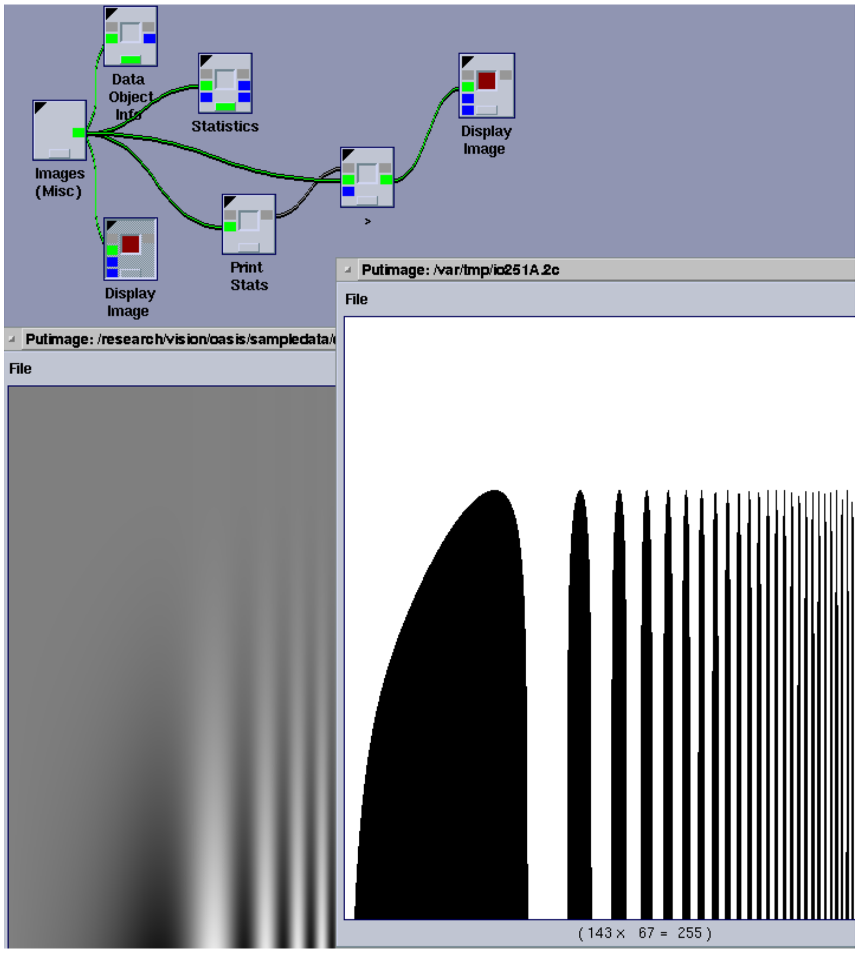
6.2 Creating a Compiled Workspace

In section 5.1 you created a visual language procedure. Here, you will learn how to create an *compiled workspace*. A compiled workspace has a number of advantages over procedures. First, it allows you to install the compiled workspace glyph in `VisiQuest` and access it as you would any other glyph. Second, it can be executed through the command line as with any other operator. Third, it eliminates a great deal of overhead by eliminating the visual aspect of program executing, making the program run faster. Finally, you can deliver a compiled workspace solution to a customer independently of VisiQuest, provided that you also include the binaries associated with the glyphs in the workspace as part of the delivery.

In this section, you will create a compiled workspace based on the thresholding procedure saved in section 5.1.

Creation of a compiled workspace is very similar to the creation of a procedure, as explained in section 5.1. Follow these steps to build the compiled workspace.

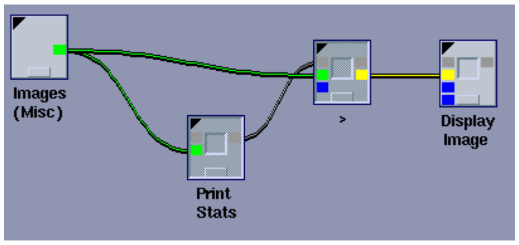
1. If `VisiQuest` is still running, exit it and restart it again so that the new HANDSON toolbox will be visible in `VisiQuest`.
2. Load the workspace created in section 3.8 under the name `threshold-mean.wk`.



Creating the compiled workspace

Like the procedure in section 5.1, the compiled workspace will consist of the `Print Stats` glyph and the “>”glyph.

3. You will not need the `Data Object Info`, the first `Display Image` glyph, or the `Statistics` glyph. Delete them.
4. Select the `Print Stats` glyph and the “>”glyph.

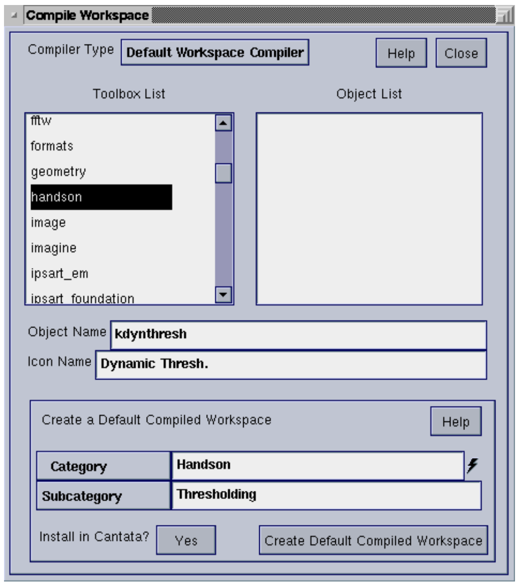


5. Select *Compile Workspace* from the *Workspace* menu.
6. Verify that *Compiler Type* is set to *Default Workspace Compiler*.
7. Select *handson* from the *Toolbox List*.
8. Fill in the parameters

Parameter	**Value**
Object Name:	`kdynthresh`
Icon Name:	`Dynamic Thresh.`
Category:	`Handson`
Sub-category:	`Thresholding`

9. Verify that *Install In VisiQuest* is set to *True*.

   These parameters indicate that the compiled workspace will be installed in your toolbox and will appear in `VisiQuest` as the `Dynamic Thresh.` glyph in the (*Handson*, *Thresholding*) menus.



Compiler Type: Default Workspace Compiler
Object Name: kdynthresh
Icon Name: Dynamic Thresh.
Category: Handson
Subcategory: Thresholding
Install in Cantata? Yes

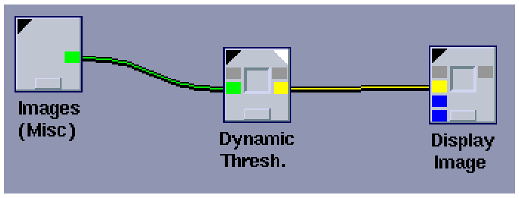
10. Click on the $\boxed{\text{Create Default Compiled Workspace}}$ button.

    This action builds the software object `kdynthresh` in the HANDSON toolbox. You will see the output in the `VisiQuest` console window, as the compiled workspace files are created and updated, Pmakefiles are generated, and the compiled workspace executable is compiled and installed in the HANDSON toolbox.

```
Executing workstation compiler as follows:
    '$IMAGINEBIN/kdefcmpl -cat "Handson" -wksp "/var/tmp/wksp1069.3" -pane ...
Creating $HANDSON/objects/kroutine/kdynthresh/uis/kdynthresh.pane
Creating $HANDSON/objects/kroutine/kdynthresh/wksp/kdynthresh.wksp
Creating $HANDSON/objects/kroutine/kdynthresh/html/kdynthresh.htm
Creating $HANDSON/objects/kroutine/kdynthresh/src/Pmakefile
Creating $HANDSON/objects/kroutine/kdynthresh/Pmakefile
Generating $HANDSON/objects/kroutine/kdynthresh/src/main.c
:
:
```

11. Place the compiled workspace glyph in the workspace.

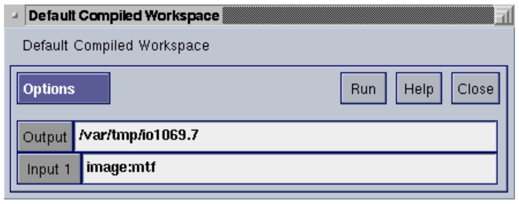
12. Close the *Compile Workspace* subform.



13. Run the workspace and verify that it is working properly. It should produce results identical to the `threshold-mean.wk` workspace.

Editing the compiled workspace glyph's GUI

14. Open its pane to see the input and output file parameters that were automatically exported to the glyph's pane.

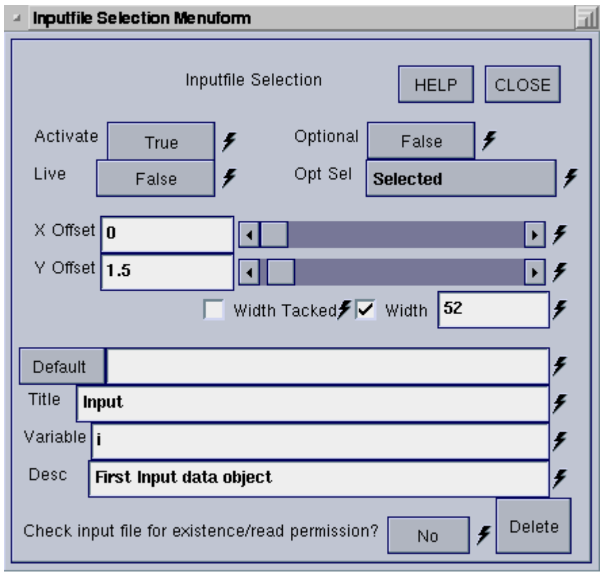


Default Compiled Workspace

Output /var/tmp/io1069.7

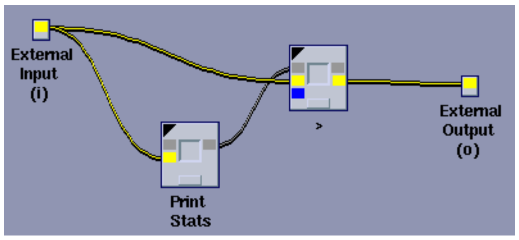
Input 1 image:mtf

15. Notice that the input and output parameters need rearranging. Put the pane in edit mode by selecting *GUI Editing Off* from the $\boxed{\text{Options}}$ menu. Use the mouse to drag the output file parameter to the bottom and the input file parameter to the top.
16. Change the variable name of the input parameter. Display the *menuform* of the input file parameter by clicking the middle mouse button on it.
17. Change the *Title* from `Input 1` to `Input` and the *Variable* from `i1` to i.



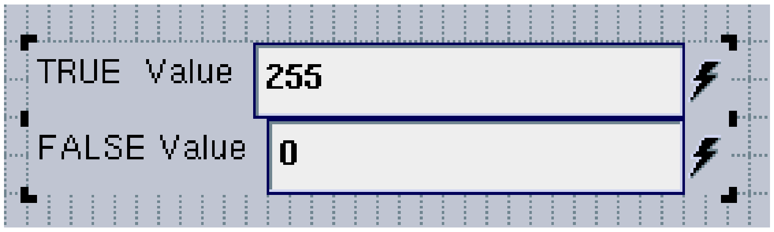
Activate: True
Optional: False
Live: False
Opt Sel: Selected
X Offset: 0
Y Offset: 1.5
☐ Width Tacked ✓ Width: 52
Default:
Title: Input
Variable: i
Desc: First Input data object
Check input file for existence/read permission? No

18. Now, open the compiled workspace glyph to display its contents. Click on the white triangle at the upper right hand corner of the glyph. **Tip:** Often, compiled workspaces and procedures are created with their external input and output ports bunched up together so that they are difficult to see. In this case, use the mouse to re-position the input and output ports by selecting the border og the box.

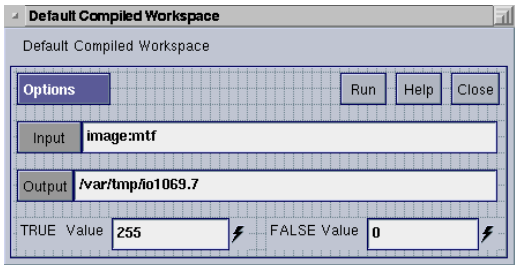


Exporting parameters to the compiled workspace pane

19. Open the pane of the ">"glyph, and put it in edit mode by selecting *GUI Editing Off* from the Options menu.
20. Select the TRUE and FALSE Value parameters.


TRUE Value 255
FALSE Value 0

21. Select *Export to Workspace GUI* from the Options menu to export the selected parameters to the GUI of the compiled workspace glyph.
22. Close the pane of the ">"glyph.
23. Close the compiled workspace by clicking on the top-left corner icon of the compiled workspace.
24. Go back to the pane of the compiled workspace. Resize and position the TRUE and FALSE Value parameters on the pane as shown below.



Input image:mtf
Output /var/tmp/io1069.7
TRUE Value 255
FALSE Value 0

25. Take the compiled workspace pane out of edit mode by selecting *GUI Editing On* from the Options menu.
26. Any time you make changes to the compiled workspace, either in the GUI or in the glyphs in the workspace, *you must regenerate the compiled workspace.* Select *Save Changes* from the Options menu. This will regenerate the compiled workspace with the modifications to its GUI. **Tip:**

You can also save changes by clicking on the *Compile Workspace* icon that appears on the far right hand side of the `VisiQuest` command bar.

27. Close the compiled workspace pane.

Test the compiled Workspace

Verify that the compiled workspace operates correctly with its new parameters.

28. Open the compiled workspace pane and provide values of 175 and 150 for TRUE and FALSE, respectively.

29. Run the workspace. Check the console window for the arguments provided to the `kdynthres` routine. You should see `-tval 175` and `-fval 150`. The output image should also reflect the difference.

6.3 Creating a Kroutine

In this section, you create a glyph based on a software object written in the C programming language (a kroutine). For illustrative purposes, this glyph will perform the simple operation of reading a data file and printing to *stdout* (*Green Console Button*) the pixel value of a specified coordinate. The name of this object will be `kpixel`.

You develop software using VisiQuest by creating software objects. To create an operator software object, you need to:

- Design its GUI pane interface.
- Use the code generator to create the user interface code.
- Add the code that implements the operator.
- Compile and install the operator.
- Test the operator using `VisiQuest` or the CLUI interface.

Creating a kroutine object

1. Exit `VisiQuest`, if it is running, and invoke `craftsman` in the background:

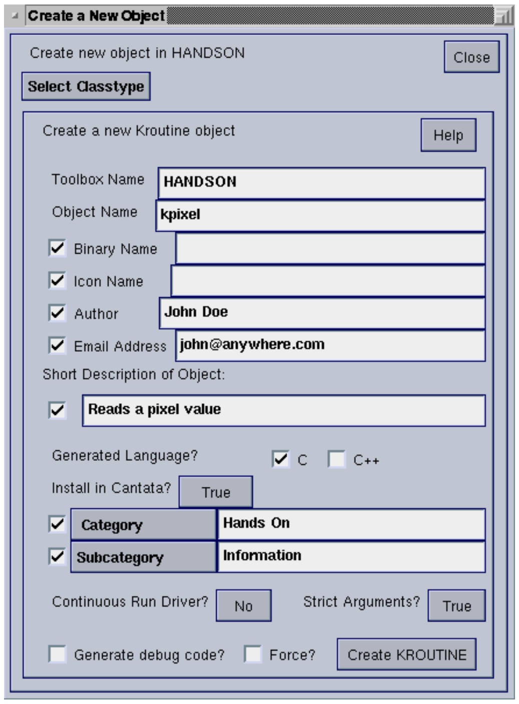
```
%craftsman &
```

2. Select the `handson` toolbox from the list of *Toolboxes*.

3. Select *Create Object* in the *Object* pulldown menu.

4. In the *Create a New Object* subform, fill in the information for the software object `kpixel` as follows. Note that the information below is regarding the creation of a kroutine type of object. A kroutine is an object written in a compiled programming language such as C or C++. The kroutine is the default class type for creating a new software object; software objects of other class types can be created by selecting the desired class type from the *Select Classtype* pulldown menu.

Parameter	**Value**
Toolbox Name	`handson`
Object Name	`kpixel`
Binary Name	`kpixel`
Icon Name	`Pixel`
Author	`your name`
Email Address	`your email`
Short Description	`Reads a pixel value`
Generated Language	`C`
Install in VisiQuest?	`Yes`
Category	`Handson`
Sub-category	`Information`
Continuous Run Driver	`No`
Strict Arguments	`True`



Create a New Object

Create new object in HANDSON

Select Classtype

Create a new Kroutine object

Toolbox Name	HANDSON
Object Name	kpixel
✓ Binary Name	
✓ Icon Name	
✓ Author	John Doe
✓ Email Address	john@anywhere.com

Short Description of Object:

✓ Reads a pixel value

Generated Language? ✓ C ☐ C++

Install in Cantata? True

✓ Category	Hands On
✓ Subcategory	Information

Continuous Run Driver? No   Strict Arguments? True

☐ Generate debug code? ☐ Force?

5. When the parameters are all typed in, click on the $\boxed{\text{Create AccuSoftOUTINE}}$ button. A notifier informs you of the object creation process.
6. When `craftsman` has finished creating your object, close the subform.

Editing the kroutine object

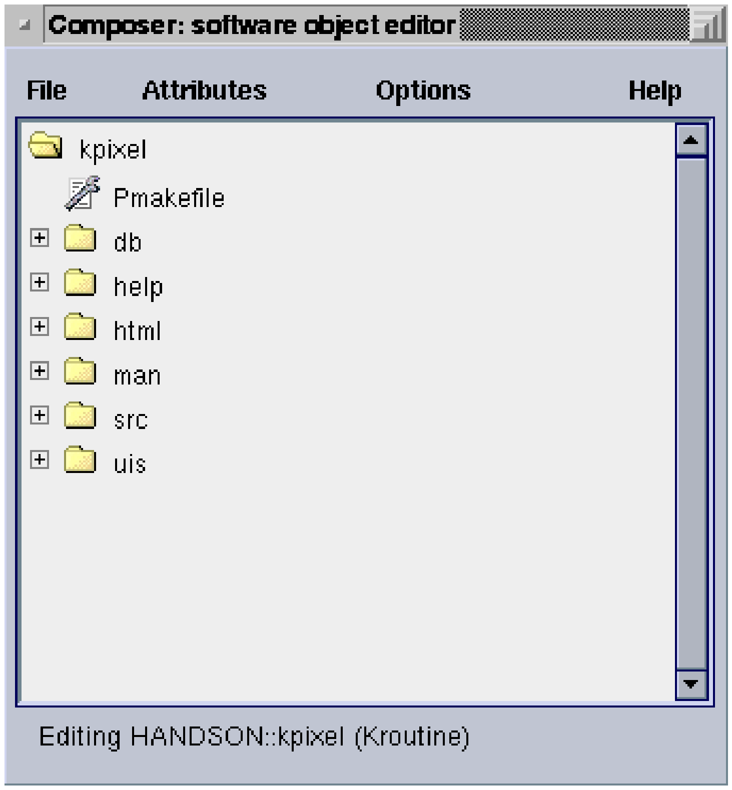
Once the object is created, you can edit to your specifications. The VisiQuest tool used to edit an object is `composer`. Using `composer`, you edit:

- The *User Interface Specification* UIS, the file which dictates the parameters in the GUI (pane) and CLUI interfaces.
- The programming code itself.

VisiQuest simplifies these two tasks. The UIS can be edited graphically directly on the pane interface by using the tool `guise`. Once the pane is edited, `composer` can automatically generate code to deal with the user interface. This means that you do not have to write any code related to user interface.

Both `composer` and `guise`are invoked automatically and transparently from `craftsman`.

1. With the `kpixel` object selected in the `craftsman` subform, select *Edit Object (Composer)* in the *Object* pulldown menu. This executes `composer`.



Editing HANDSON::kpixel (Kroutine)

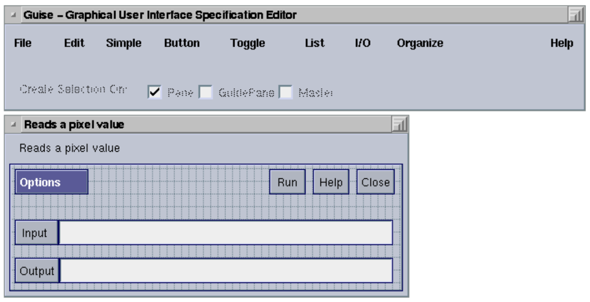
Editing the pane interface

Specify the GUI/CLUI interface by graphically editing the pane of the object by using the `guise` tool.

2. Click on the $\boxed{+}$ to reveal the `kpixel.pane` UIS file.

3. Double-click on the `kpixel.pane` file to execute `guise`.

   In a few moments `guise` will create two windows. The first is the `guise` subform where you select the appropriate commands in the object pane. The second is the pane of the object in the edit mode. Note the gridded background in the pane object. Also note that the pane object, by default, has already one *Input* and one *Output* file parameter, and $\boxed{\text{Run}}$, $\boxed{\text{Help}}$, and $\boxed{\text{Close}}$ buttons.

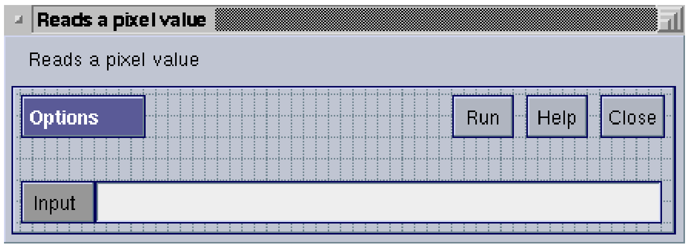


Deleting the output file parameter

As the `kpixel` operator requires a single file input parameter, delete the output file parameter.

4. Select the *Output* parameter in the pane object by clicking on it with the left mouse button. Its four corners are highlighted indicating the selection made.

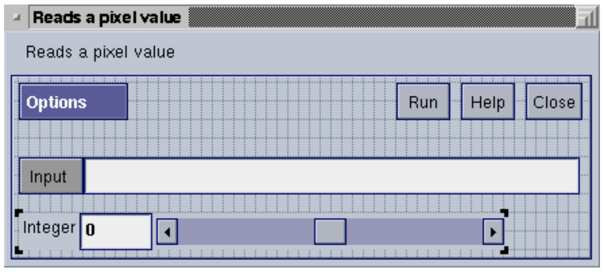
5. In the `guise` subform, select *Delete* in the *Edit* pulldown menu. This will delete the *Output* parameter from the object pane.



Creating five integer parameters

The object must have five integer parameters that correspond to the *width*, *height*, *depth*, *time*, and *element* coordinates.

6. Select *Integer* in the *Simple Variables* pulldown menu in the `guise` subform. This brings up an integer parameter to the pane object.



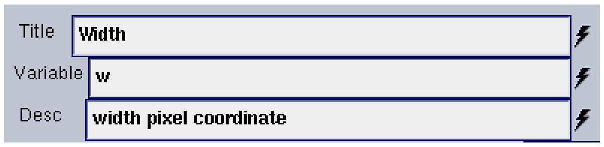
Now change this input parameter to reflect the desired title, variable, and description.

7. Click with the middle mouse button on the *Integer* parameter in the object pane. This shows the *Integer Selection* subform for editing the parameter.

8. Set the following parameters in the *Integer Selection* subform. Verify the changes in the object pane as you enter the new values.

Parameter	**value**
Bounds	`Value >= 0`
Title	`Width`
Variable	`w`
Desc	`width pixel coordinate`

It is important to understand the meaning of the *Variable* parameter, set here as `w`. This variable is associated with the parameter in the CLUI interface. For instance, to specify `width = 100`, use the syntax `-w 100` in the command line. This variable is also used in the C programming code, when you access this integer parameter for width.

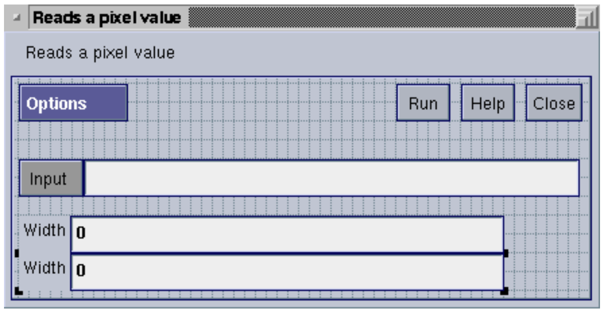

Title: Width
Variable: w
Desc: width pixel coordinate

9. Close the *Integer Selection* subform.

   Create the *height* parameter by duplicating the width integer parameter.

10. With the *Width* selected, select *Copy* in the `guise` *Edit* pulldown menu.

11. Move the duplicated parameter below the first integer parameter by holding down the left mouse button on the parameter and moving it to the desired location.



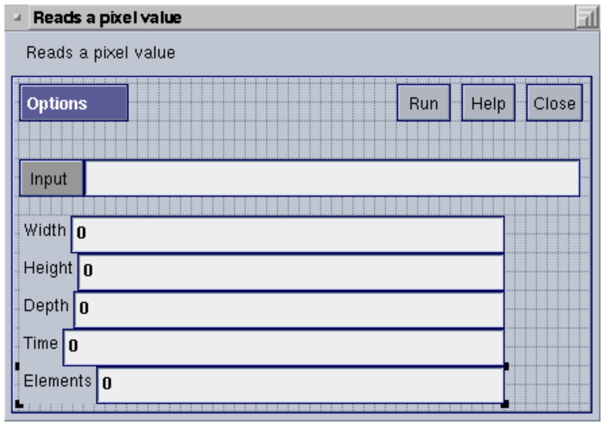
12. Click on the duplicated parameter with the middle mouse button and change the parameters.

Parameter	Value
Title	`Height`
Variable	`h`
Desc	`height pixel coordinate`

13. Create the last three parameters, *Depth, Time,* and *Element*, by duplicating and then editing each one. Use the following titles, variables, and description for each parameter.

Title	Variable	Desc
`Depth`	`d`	`depth pixel coordinate`
`Time`	`t`	`time pixel coordinate`
`Element`	`e`	`element pixel coordinate`

At the end of editing, the pane object appears as follows:



Reads a pixel value

Width 0
Height 0
Depth 0
Time 0
Elements 0

Saving the UIS file

Once the pane has been edited, it must be saved. Note that the edited pane UIS specification is stored in the file `kpixel.pane`. Its path can be found by selecting *Open* from the *UIS* pulldown menu in `guise` .

14. Save the UIS file by selecting $\boxed{\text{Save}}$ from the the `guise` $\boxed{\text{File}}$ menu. You can confirm the overwriting operation on the previous UIS file.

15. Exit `guise` by selecting $\boxed{\text{Quit}}$ from the $\boxed{\text{File}}$ menu.

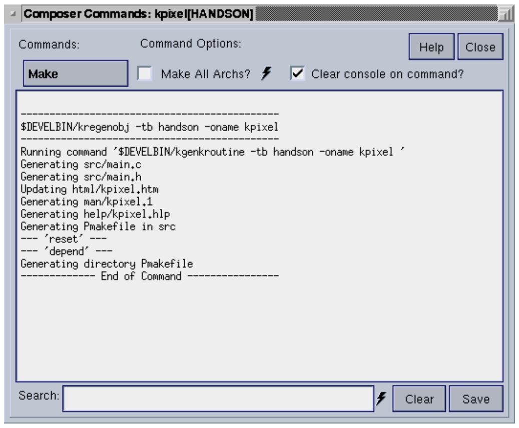
Source code generation

The source code required to read the parameters of the object is generated automatically. Every time you edit the UIS file with `guise` you must update the source code files. This is a crucial step that should never be forgotten.

16. On the *Composer* subform, select $\boxed{\text{Commands}}$ from the *Options* menu.

17. On the *Commands* subform, select $\boxed{\text{Generate Code}}$ from the *Make* menu.

    Note that several files are generated, corresponding to the C files, header files, and documentation files of the object.



Make All Archs? [ ] Clear console on command? [✓]

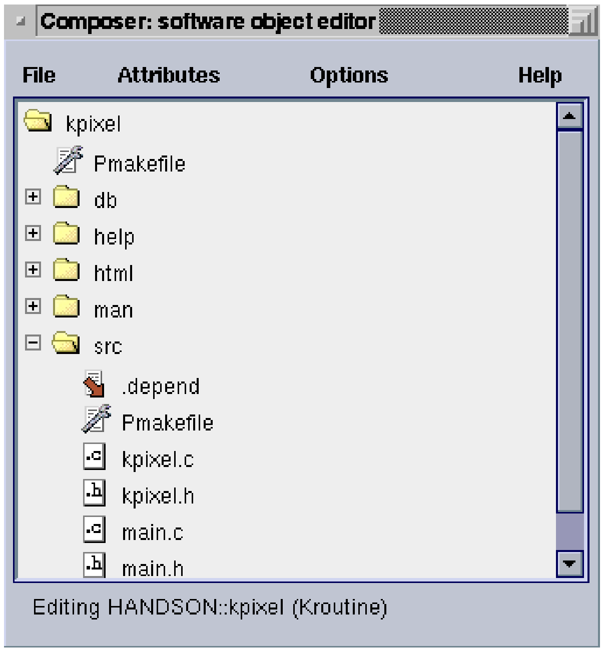
```
----------------------------------------------
$DEVELBIN/kregenobj -tb handson -oname kpixel
----------------------------------------------
Running command '$DEVELBIN/kgenkroutine -tb handson -oname kpixel '
Generating src/main.c
Generating src/main.h
Updating html/kpixel.htm
Generating man/kpixel.1
Generating help/kpixel.hlp
Generating Pmakefile in src
--- 'reset' ---
--- 'depend' ---
Generating directory Pmakefile
------------- End of Command ----------------
```

18. Close the *Commands* subform.

Editing the source C files

Once the source code has been automatically generated, include the code that will actually perform the operation. This object will read the input file and will pick the pixel value at the specified pane coordinate parameters.

19. Click on the $\boxed{\text{src}}$ directory in the *Composer* list. Three source code files appear in the list.



20. To start the editor, either double click on the `kpixel.c` file, or click once on the `kpixel.c` file to select it and then choose *Edit* from the *File* pulldown menu.

    This displays the `kpixel.c` source file in your preferred text editor.

    The next step is to add your code to the `run_kpixel()` routine.

Variables declaration

21. Declare the variables `src_obj`, to hold the input data structure as a VisiQuest software object and `data` to hold the pixel value, in the variable list declaration section:

```
kobject src_obj;
double  *data = NULL;
```

Opening the input file as a data object

Here, the data file is open and stored in the `src_obj`. The filename specified by the user in CLUI or GUI interface is available in the variable `clui_info->i_file`. VisiQuest automatically generated user interface code to parse the input filename string and stored it in this variable.

22.
```
src_obj = kpds_open_input_object(clui_info->i_file);
if (src_obj == KOBJECT_INVALID)
{
  kerror(NULL,"kpixel",
         "Unable to open input object %s",
         clui_info->i_file);
  kexit(KEXIT_FAILURE);
}
```

Implementing the Algorithm

Now, add the code to read the pixel value at the location specified by the user and assign it to the variable `data`. Note the way the input parameters are related to the variables specified in the UIS pane. The variable `w`, associated with the integer parameter `Width`, is stored in `clui_info->w_int` and the same rule is applied for the other parameters. After the pixel value is assigned to the variable `data`, it is printed to *stdout*.

23. Enter the code

```
if (!kpds_set_attributes(src_obj,
    KPDS_VALUE_DATA_TYPE, KDOUBLE,
    KPDS_VALUE_OFFSET,
    clui_info->w_int, /* w input parameter */
    clui_info->h_int, /* h input parameter */
    clui_info->d_int, /* d input parameter */
    clui_info->t_int, /* t input parameter */
    clui_info->e_int, /* e input parameter */
    NULL))
{
    kerror("main","kpixel",
           "unable to set source object value attr.");
    return(FALSE);
}
data = (double *)kpds_get_data(src_obj,
                               KPDS_VALUE_POINT,
                               data);
kprintf("Pixel[%d,%d,%d,%d,%d]=%g\n",
          clui_info->w_int,
          clui_info->h_int,
          clui_info->d_int,
          clui_info->t_int,
          clui_info->e_int,
          data[0]);
```

Finish & Cleanup

As the last steps, the data and the object variables are deallocated from memory. The routine must return TRUE for success.

24. Enter the code

```
kfree_and_NULL(data);
kpds_close_object(src_obj);
```

Compiling your code

Once the code is edited, save the file and exit the editor.

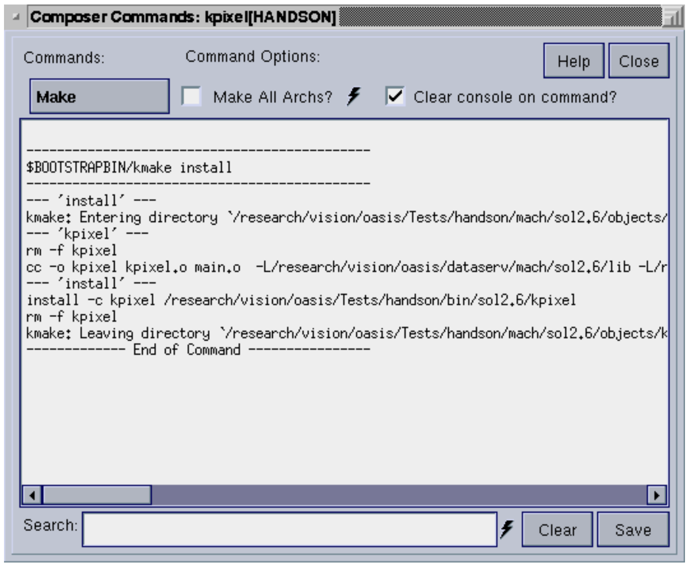
The next step concerns the compilation and the installation of the object. `composer` has integrated tools to compile your programs using the configuration set up in the VisiQuest installation. You need to have access to the ANSI C compiler described in the VisiQuest Installation Guide in section 1.5.1.

25. Select the $\boxed{\text{Commands}}$ button from the *Options* menu in `composer` .

26. In *Commands*, choose *Kmake Install* from the *Make* pulldown menu.

    It is likely that there are syntax errors introduced in the process of copying the code into the C file. If that is the case, invoke the editor again by choosing *Edit* in the *File* menu of `composer` . Correct the errors and repeat the *Kmake Install* process.

    If everything works, your object has been compiled and installed successfully. There may be several warning messages, but you can verify that the object has been installed if the line before the last in the *Commands* subform is the `install` command or a message saying that `kpixel` is installed.



Make All Archs? [✓ Clear console on command?]

```
---------------------------------------------
$BOOTSTRAPBIN/kmake install
---------------------------------------------
--- 'install' ---
kmake: Entering directory `/research/vision/oasis/Tests/handson/mach/sol2.6/objects/
--- 'kpixel' ---
rm -f kpixel
cc -o kpixel kpixel.o main.o  -L/research/vision/oasis/dataserv/mach/sol2.6/lib -L/r
--- 'install' ---
install -c kpixel /research/vision/oasis/Tests/handson/bin/sol2.6/kpixel
rm -f kpixel
kmake: Leaving directory `/research/vision/oasis/Tests/handson/mach/sol2.6/objects/k
------------- End of Command ----------------
```

27. Close the *Commands* window, quit `composer` and `craftsman`.

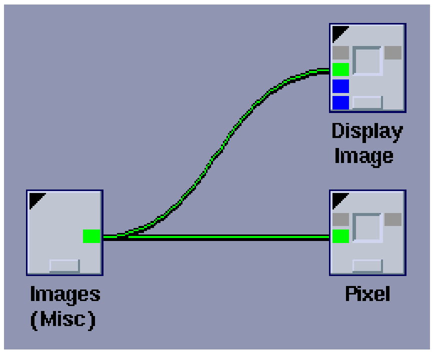
Testing the object in VisiQuest

28. If you have `VisiQuest` running, exit the program and invoke it again, because `VisiQuest` can only see the new changes in the HANDSON toolbox when it starts up.

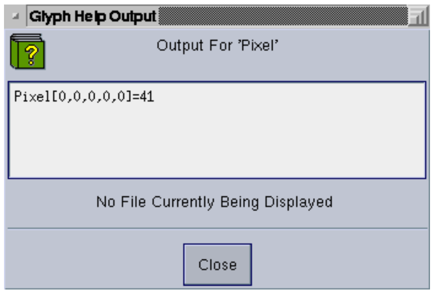
29. Find the glyphs:

    - `Images (Misc)` glyph (*Input/Output, Data File*)
    - `Pixels` glyph (*Handson, Information*). Note that this is the new glyph you have just created using `composer`.
    - `Display Image` glyph (*Visualization, Non-Interactive Image Display*)

30. Connect them together as indicated below.



31. Select the `Spanish Sea Gull` image in the `Images (Misc)` glyph.
32. Run the `Pixel` glyph and verify its output. The first pixel of the gull image has value 41.



```
Pixel[0,0,0,0,0]=41
```

33. Test the `Pixel` glyph in other pixel coordinates and compare the output values with the `Display Image` glyph.
34. Quit `VisiQuest`.

Conclusion

In this chapter, you have learned how to create a toolbox, create and install a compiled workspace in `VisiQuest`, and build a complete kroutine object. You have designed the pane interface of the kroutine and coded an operator that prints the pixel value at specified coordinates of an image data file.