

VisionX V4 Users Guide

Anthony P. Reeves
School of Electrical Engineering
Cornell University

©2000 by A. P. Reeves. All rights reserved.

March 20, 2000

1 Introduction

The VisionX system provides computer tools and programs for the analysis and visualization of image data. It is suitable for a wide range of image analysis applications and is designed to address the processing needs of multidimensional image sets that arise both from temporal image sequences and from image modalities that involve 3D data collection.

VisionX has been used in a wide range of research applications including: multispectral image analysis, 3D object recognition, multi-frame image analysis, target tracking, neural networks, biological cell analysis and 3D biomedical image analysis.

Important features of the VisionX system include: the ability to handle multidimensional image sets, the wide range of available processing functions and a flexible tagged data format that facilitates the automatic recording of the history of the file. VisionX performs image analysis and related visualization functions; as such it has capabilities for processing video image sequences and rendering animations. However, it does not attempt to duplicate or replace other types of image processing applications such as video editors or computer graphics animation that are well supported by other mature systems.

The VisionX system has benefited from over 20 years of development. The original system was developed in the UNIX environment and exhibits the standard features of a UNIX package. A VisionX interactive data manager (VXM) has been developed to provide a more familiar environment for Windows users. However, these primary environments are interchangeable. VXM may be used on UNIX platforms and the UNIX like environment may be used on Windows DOS command terminals and with the more UNIX-like packages that are available for Windows.

VisionX and Windows

VisionX V4.0 is available for Windows 95 and Windows NT platforms. Program development is done with Microsoft Visual C++ version 5.0. The older VisionX V3 commands discussed below are currently not available for Microsoft Windows. VisionX takes advantage of a number of free software packages for utility operations: gnuplot for graph plotting, and a www browser for documentation. The tcl/tk8.0 package is used for several visual interfaces and command scripts.

VisionX and UNIX

VisionX was originally developed on UNIX and has been compiled on many different UNIX platforms. Command documentation is available in the UNIX man style format. A number of commands in the old VisionX V3 system have not yet been ported to V4. These commands are available for UNIX systems but they use the old v3 file format; in general, these commands can be used with V4 files but not all the V4 file facilities, such as automatic history logging, are supported.

VisionX takes advantage of a number of free software packages for utility operations: gnuplot for graph plotting, a www browser for documentation, and the pbmplus or netpbm package for image format conversion. The tcl/tk8.0 package is used for several visual interfaces. In general program development may be done using either the local C compiler or with the gnu C compiler.

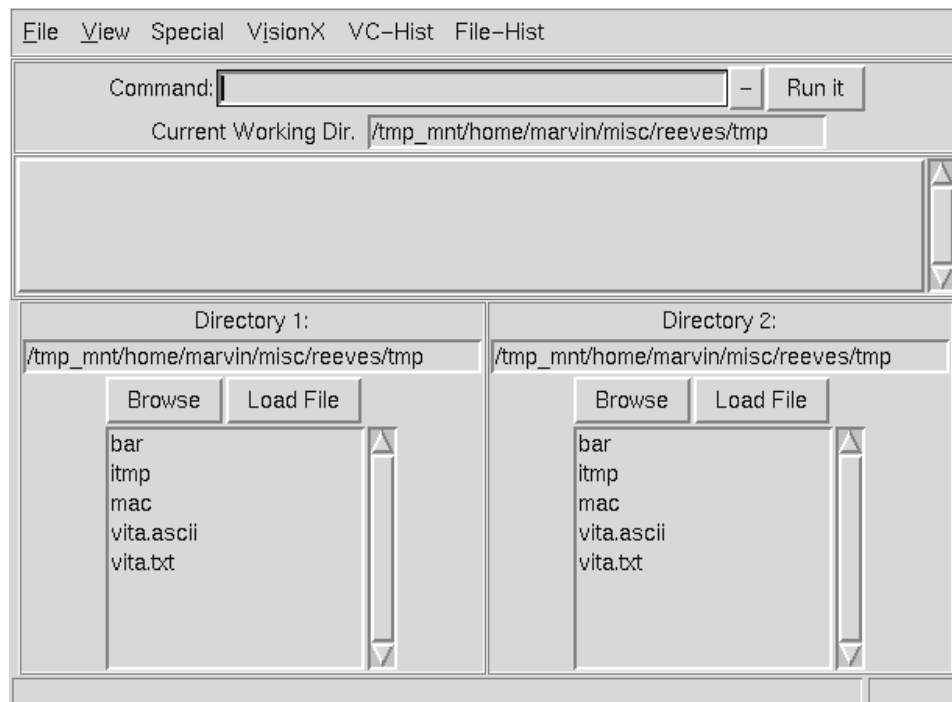


Figure 1: Snapshot of vxm

2 The VisionX Data Manager (VXM)

The VisionX data manager provides interactive access to both image data files and VisionX commands. In addition it also provides access to the VisionX documentation. A snapshot of the vxm interface screen is shown in Figure 1. Images and graphics visualized by vxm are displayed in their own external windows.

Image Display

The two lower panels labeled "Directory 1" and "Directory 2" permit the user to browse and select files from two different parts of the file system. Changing the directory can be achieved by typing a different name into the directory or by clicking on the "Browse" button and selecting a file from a different location.

The list below the Browse button lists all the files and directories in the current directory. An image may be displayed by double clicking on its name or by selecting it and then clicking on the "Load File" button. The two directory panels are independent; however if you select a file in one panel and click on the "Load File" then the second panel will be set to the same directory as the first.

VisionX image files will be displayed by the *xvx* command and which will create a separate image display window for each image. A display window may be deleted by typing the letter "q" in it. If the file selected is not a VisionX image file then *vxm* will attempt to identify the file type and select an appropriate display command. Polygon files are rendered as a wireframe image, graph files are plotted if an appropriate plotting program can be found, and tex files are presented in a

text view. On UNIX systems many foreign image format files will be displayed if an appropriate conversion filter can be found in the system.

Executing Commands

The main panel above the directory panels contains a command input line and a small panel for logging the results of executed commands. Any command may be typed at the “Command:” prompt; commands may include pipes and file indirections. The command is executed on hitting return or by clicking on the “Run it” button. The response from the command, if any, will be shown in the log window. A history of commands may be accessed by selecting the “-” button at the end of the command line. Selecting a previous command from the history list will load it into the command line where it may be edited and executed. The “Current Working Dir.” shows in which directory the command will be executed. The current directory may be manually set at any time. The directory is also automatically set to one of the directory panel directories when a directory change is made or when an image is displayed. This automatic capability may be inhibited by selecting the menu item “Special=>Fix Working Dir.”. Vxm has an interactive facility for exploring VisionX commands which is described in section 4.

Alternative Image Display (VDM)

A second flavor of vxm called *vdm* (or alternatively invoked by vxm -i) employs a reformulated visual interface that includes two additional 512 x 512 pixel image display panels; each of which is associated with the directory panel above it. In this version images and graphics are displayed, by default, in the window below the directory panel; they may also be displayed in external windows by “VisionX=>Display” menu selections.

3 Command Line Parameters

An advanced protocol to specify command line parameters is employed by VisionX commands. It permits both positional and position-independent parameter specification. Furthermore, it supports the pipelining of commands such that the output from one command can be fed directly into the next command without an intermediate file.

Information Parameters

This protocol is illustrated by means of an example command “vsobel” which is a simple edge enhancing filter command that takes an image input and produces an image result. Two enquiry parameters “-h” and just “-” provide information about the command. For example, to find out what parameters a command can take give

```
vsobel -H
Usage: vsobel [-H] [if=<inputfile>] [of=<outputfile>] [-f]
[-d] [-i] [-e]
```

The -H parameter lists all the possible parameters for a command. Further information can be obtained with the “-” parameter, for example

```
vsobel -
Usage: vsobel [-H] [-] [-help]
[if=] infile vsobel: edge operator
[of=] output file
[-f ] float computation option
[-d ] provide direction information
[-i ] isotropic flag
[-e ] euclidian flag
```

The “-” option provides short descriptions of each parameter. It may be specified as the last parameter in a list as a shortcut for situations (often encountered) where a user has already typed several parameters but wishes to then be reminded of the full command syntax.

Value Parameters

There are two types of parameters *option* parameters which are preceded by a “-”, such as “-i” above, and *value* parameters which have a “=”, such as “if=” above, and require an additional value argument.

By convention, the primary input file for a command is usually associated with an “if=” prefix and the primary output file is associated with an “of=” prefix; such is the case with vsobel. Consider that we wish to process a file called “image.vx” and store the result in a file called “edge.vx” using the isotropic “-i” option. This may be specified by either

```
vsobel if=image.vx of=edge.vx -i
```

or

```
vsobel of=edge.vx -i if=image.vx
```

This is an instance of position-independent value parameters: the prefix identifies the parameter. This is the standard and most safe method for parameter specification.

Alternative methods for parameter specification are supported which are especially useful when the user is interactively typing commands. Value parameters may use a positional dependent syntax in which they are satisfied in order. The only exception to this is the of= parameter which may be specified by a “-o” prefix. This is very convenient when a number of value parameters are to be specified. Therefore, the above command could also be specified by

```
vsobel image.vx -o edge.vx -i
```

The mixing of value parameter types is also permitted so any of the following are equivalent

```
vsobel if=image.vx -o edge.vx -i
vsobel -i of=edge.vx image.vx
vsobel -o edge.vx if=image.vx -i
```

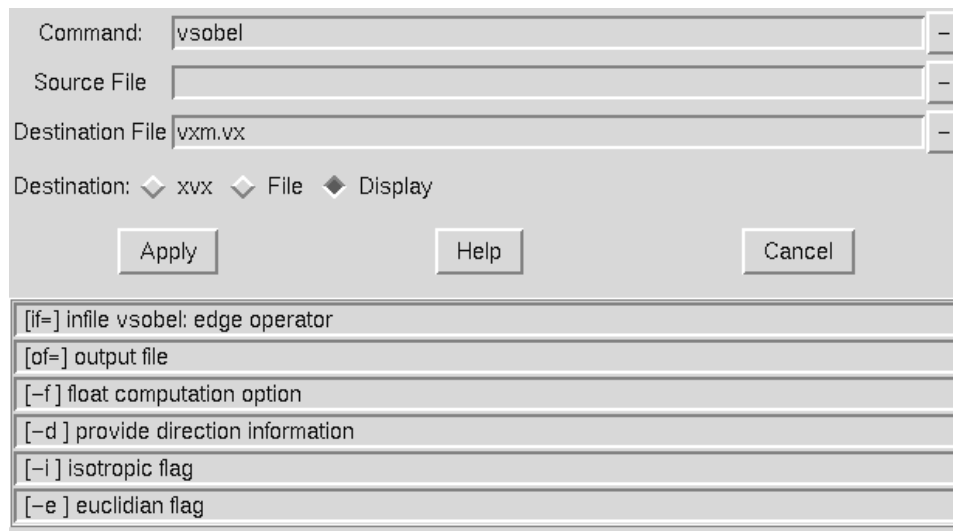


Figure 2: vdm interactive command window

File Redirection

Standard file direction primitives such as `<`, `>`, and `|` are supported such that the following are also equivalent to the above:

```
vsobel -o edge.vx -i < image.vx
vsobel -i of= if= < image.vx > edge.vx
vsobel -i < image.vx > edge.vx
vsobel if=image.vx -i > edge.vx
```

The syntax “if=” without an argument is used to explicitly state that the value of the argument is the standard input (second example above). Since `vsobel` only has one input value argument, there is no need to explicitly specify it (third example above).

4 VXM interactive command execution

Vxm has a convenient facility to allow the user to interactively explore the use of the VisionX commands. The facility is accessed by the *VisionX* menu. A command is selected by dragging the *VisionX* menu to the correct command and then selecting the desired command. For example, selecting “*VisionX*=>Edge Operators=>`vsobel`” will display the window shown in Figure 2.

The first entry in this window is the command that you are constructing; it starts with the command name. Typically there is one or two input files and an output file associated with a command. The next two or three entries are for these file names. Command options need to be set. The available command options for the specified command are listed in the lower part of the window; clicking on one of these will add it to the command line that you are building.

A user constructs a command in the top entry, selects input and output files (if= and of= are automatically provided as needed) and then clicks on the “Apply” button to execute the command. Clicking on the “Help” button will bring a window with the complete documentation for the specified command. The “-” menu buttons at the end of each entry provide access to a history list of

command or file names that have been used in the past. The destination buttons allow the result of the command to be visualized or sent to a specific file.

5 VisionX Program Development

The programming tools for VisionX are described in the VisionX V4 Programmers Manual.

6 VisionX Commands

The commands for VisionX are described in the VisionX V4 Command Guide.

7 VisionX File Data Structures

VisionX files use a tagged type system. That is, each component of the file consists of a tag which contains a type and a length, followed by the data which has the specified length. The philosophy of this organization (used in many image file formats) is that a large number of diverse data types can easily be accommodated. Many applications modify just those components that they are designed to operate on and ignore any others. In this way, the file structure is easily extended to add new component types. In addition, some higher level structures are required that require multiple tags. For example, an image consists of a bounding box component followed by a pixel data component; the bounding box provides the structure information while the pixel data provides the image contents.

An advantage of the above design is that new features can be easily added to the system without modifying all the commands to accommodate them (of which VisionX has over 200). A disadvantage is that not all commands treat new data structures in the expected manner.

There are several VisionX commands for viewing the components of a VisionX file. Every file has a title component and a cumulative history component; these can be viewed with the `vls` command or from the View menu of `vxm`. A summary of the components in a file can be obtained with the `vps` command (with a `-t` option) or with the View menu of `vxm`. The contents of a file with each component listed can be viewed with the `vpr` command.

In the following the main basic component organizations currently used in VisionX are outlined.

The 2D image

The 2D image is the fundamental building block of vision systems. In VisionX it consists of two components: a bounding box, and pixel data. The bounding box specifies the index range of the image and the pixel data tag specifies the base type and the size of the data. Multi-component pixels (e.g., color pixels) are indicated by the pixel data length being a multiple of the size specified by the bounding box. Originally the bounding box specified 4 elements (x-low, x-high, y-low and y-high); more recently they often have six elements including an additional (z-low and z-high). For 2D images these last two values are usually set to zero. For color index images the image structure is usually preceded by a color look-up-table component.

Programming tools are available for treating 2D images like 2D arrays.

The 3D image

The 3D image is a relatively recent feature of VisionX. In VisionX files it is realized by a set of 2D images in which the z-specification of the bounding box is consistent. For example, a 3D image with 3 voxels in the z-direction would consist of 3 2D images which have the same x and y direction bounding box values. In addition if for the first image the bounding box has z-low = 0 and z-high = 1 (indicating a 0-1 range in z) then the box for the second image must have z-low = 1, and z-high = 2 and the box for the third image must have z-low = 2 and z-high = 3. Most VisionX commands designed to operate primarily on 3D images have a v3 prefix in their command name. Note, many older VisionX commands will just treat 3D images as a set of 2D images and will frequently perform the correct function without requiring any modification. Note also that sets of 2D images which do not conform to the 3D convention are still valid VisionX data files but will not be treated as 3D entities.

Programming tools are available for treating 3D images like 3D arrays.

Frames

A file may be organized in frames. A frame consists of a start frame component, the frame contents components, and an end frame component. There are many cases when a whole image file is not to be read into memory in one step (when the file is a movie for example); the frame provides a mechanism for "chunking" a file so that a file read operation may read just one frame at a time as a unit. The frame end element prevents read-ahead into the next frame when this capability is required. A temporal sequence of images (movie) is usually represented by a set of 2D images with similar x and y specifications in separate frames. Prior to the introduction of the 3D convention above, 3D images were also represented by this structure. In fact it is still appropriate to store large 3D image sets in this format.

There are two fundamental programming tools for reading data files: the first reads the whole file in a single operation and the second reads a file one frame at a time. Most commands use the latter form as this enables them to process files of an arbitrary large size. Programming tools are available for processing a moving window of frames.

The 4D image

A 4D image in VisionX is represented by a framed sequence of 3D images. That is, each frame contains one 3D image. In the future, an extended version of the 3D image structure could be used for 4D images; however, while such a file can be created now, there are currently no programming tools to directly support this data structure.

Commands are available for reorganizing the dimensions of 4D images so that they can be processed with standard 2D and 3D commands.

Objects

Objects are collections or groups of components. Objects are delineated by the object component. Components between two object components are considered to comprise of a single object. Object groupings are nested within and do not cross frame boundaries.

Objects are very useful in grouping say a set of polygons to a single entity. Other attributes (such as color) may then be included in the group.

3D Graphical Representations

3D graphics in VisionX is based on files containing a set of 3D polygons (preceded by a single 3D bounding box). A set of polygons may be grouped into a single "object" using the object component mentioned above such that a file may contain a collection of "objects". In addition a polygon or set of polygons may be preceded with other attributes such as a face color and a boundary color. An important feature of the VisionX system is the matching of coordinate systems of both images and polygons which makes possible the mixed rendering of both image and polygon surface data.

Programming tools are available for the rendering of polygon files that contain just 3 and 4-sided polygons. A number of utility commands are available for manipulating 3D polygon files in the above format.

3D Image Structures

There are two 3D image structures outlined above, 3D images and image sequences. At this time, some commands will only operate on one not both of these formats. However, the "vdim" command will convert between these formats. In general, it is possible add a vdim pipe between two incompatible commands.