Vista: A Software Environment for Computer Vision Research

Arthur R. Pope David G. Lowe
Department of Computer Science, University of British Columbia
Vancouver, B.C., Canada V6T 1Z4
{pope,lowe}@cs.ubc.ca

Abstract

Vista is a software environment supporting the modular implementation and execution of computer vision algorithms. Because it is extensible, portable, and freely available, Vista is an appropriate medium for the exchange of standard implementations of algorithms. This paper, an overview of Vista, describes its file format, its data abstraction, its conventions for UNIX *filter programs and library routines, and its user interface toolkit. Unlike systems that are designed principally to support image processing, Vista provides for the easy creation and use of arbitrary data types, such as are needed for many areas of computer vision research.*

1 Introduction

As a science concerned with creating artificial systems, computer vision ought to emphasize the evaluation and comparison of alternative methods [1] [2]. It should be simple to compose large systems from components created by other researchers. That this is seldom done is partly due to the lack of a suitable, common software environment. Without a common environment, implementations and data can be exchanged only with great difficulty.

What is required of a software environment for computer vision research? It should support a large variety of data types, including not only images of various kinds but also other objects such as edge vectors, object models, and camera calibration parameters. It should be readily extensible in terms of both the data structures supported and the algorithms implemented. So that it can be understood easily, it should be modular, lightweight, and straightforward. It should have reasonable efficiency. Finally, to encourage its widespread adoption, the environment should be based on a popular language, be written for portability, and be made freely available.

Of these requirements, extensibility deserves special mention because it is difficult to provide and often overlooked. No design can anticipate the tremendous variety of data structures that researchers may wish to create; the design must, instead, provide a mechanism by which new structures can be defined, and then stored and manipulated alongside standard structures like images. It must be possible to define new structures out of existing ones (e.g., build an image pyramid from images) and to augment existing structures with additional information (e.g., annotate an image with information about how it was acquired). Moreover, extensions like these should not require modifications to existing software.

Several existing software environments provide some support for computer vision research, yet fail to meet all of the requirements stated above. Often these environments focus too narrowly on image processing (e.g., Khoros [3] and LaboImage [4]); while providing excellent support for image manipulation, they make few provisions for storing and manipulating other kinds of information (e.g., edge vectors). Some environments must be purchased, which limits the ability to distribute code developed within them (e.g., HIPS [5], HVision [6], and KBVision [7]), or they bear copyrights restricting how code may be copied, modified, and redistributed, which limits their extensibility (e.g., Khoros [3]). One promising system, the ARPA Image Understanding Environment [8], remains under development.

Faced with these shortcomings, we began three years ago to develop a new environment whose particular goals were to support computer vision research and to foster the exchange of research products. That environment, Vista, is the subject of this paper. Vista is now freely distributed with extensive documentation, a collection of image and edge vector manipulation routines, tools for printing and viewing images and edge vectors, and a toolkit to aid in developing interactive applications. As a foundation for developing and exchanging algorithms, Vista is designed to be simple, lightweight, and accessible; it requires no special hardware, nor is it intended for real-time applications. It is written in ANSI C, and it runs on most UNIX platforms and X Window System displays.

Unlike most environments for image processing, Vista emphasizes easy extension. It uses a flexible, self-describing file format so that objects of many types, including custom ones, can be represented for storage and interchange. In section 2 we describe the file format. Vista already includes support for some commonly needed types, such as one for representing images described in section 3. The environment can be extended readily to support new data types.

Vista operations are packaged both as stand-alone UNIX programs and as library routines. Section 4 illustrates how the programs can be used from a UNIX shell to manipulate files containing images and other objects. Section 5 describes the library, which can be used to construct custom applications. One component of the library is a toolkit for rapid development of X Window System applications. Using a few toolkit routines, and without knowledge of X programming, one can create interactive, menu-driven applications that display images and vectors. The toolkit is described further in section 6. An appendix summarizes operations implemented in the spring 1994 release of Vista.

2 File format

Vista employs a data file format that is flexible, designed for extensibility, and largely self-describing. It is based on a simple structure called an *attribute list*—a sequence of attributes, each having a name and a value. An attribute's value may be a number, a string, or a keyword; it may be a nested attribute list; or it may be an object, such as an image, with an explicitly-named type and a value that, in turn, is specified by a nested attribute list. Attribute lists provide a natural representation for a wide variety of data structures, as the example below will illustrate.

When an attribute list is stored in a data file, it is encoded primarily in ASCII. However, information that must be stored compactly (such as an array of image pixel values) is expressed using a machine-independent, binary encoding. Within a file, ASCII and binary portions are kept separate so that the ASCII portion can be examined directly by a human.

Figure 1 shows a data file that contains two images, plus a third object whose type is identified as `camera`. The first image, named `ufo`, contains several attributes describing it's size, pixel representation, and the location of its binary-encoded pixel data within the file. One of `ufo`'s attributes is actually another image, named `icon`. The `camera` object contains both numeric attributes and binary-encoded data. This example illustrates some key properties of Vista's data file format:

- Related objects can be grouped, either by nesting one object within another (as `icon` is nested within `ufo`) or by listing them sequentially within one file (as the `ufo` image is paired with the `camera` object).
- Because the format is self-describing, new types of objects can be introduced without changing or obsoleting existing software. Each object includes enough information about itself to permit it to be copied from one file to another unmodified, even by programs not equipped to interpret the object. In this example, the `camera` object is a custom one, newly created; however, it will be carried transparently by any existing programs used to transform the images.
- Existing types can be extended as needed by inserting additional attributes. For example, if some application requires that images be annotated with some extra information, we can simply include that information as additional attributes within image objects. The application can check for these new attributes while existing programs, not programmed to interpret them, will simply copy them unchanged.
- Much of a data file's contents can be interpreted directly by a human. Using a text editor one can inspect attributes, edit or delete them, and insert new ones. (Offsets specifying the location of binary data remain valid despite such edits because they measure from the end of the ASCII portion of the file.)

```
V-data 2 {
  ufo: image {
    data: offset of 1st image's pixel values
    length: length of 1st image's pixel values
    nbands: 1
    nrows: 256
    ncolumns: 256
    repn: float
    comment: "UFO sighted over Seattle"
    pixel_aspect_ratio: 1.25
    icon: image {
      data: offset of 2nd image's pixel values
      length: length of 2nd image's pixel values
      nbands: 1
      nrows: 32
      ncolumns: 32
      repn: ubyte
    }
  }
  parameters: camera {
    data: offset of camera object's binary data
    length: length of camera object's binary data
    focal_length: 30.0
    center_x: 0.5
    center_y: 0.51
  }
}
^L
1st image's pixel values
2nd image's pixel values
camera object's binary data
```

Figure 1: Example of a Vista data file. A list of attributes is encoded primarily in ASCII to allow human interpretation. Large volumes of data are encoded in binary, in a separate part of the file, for efficiency.

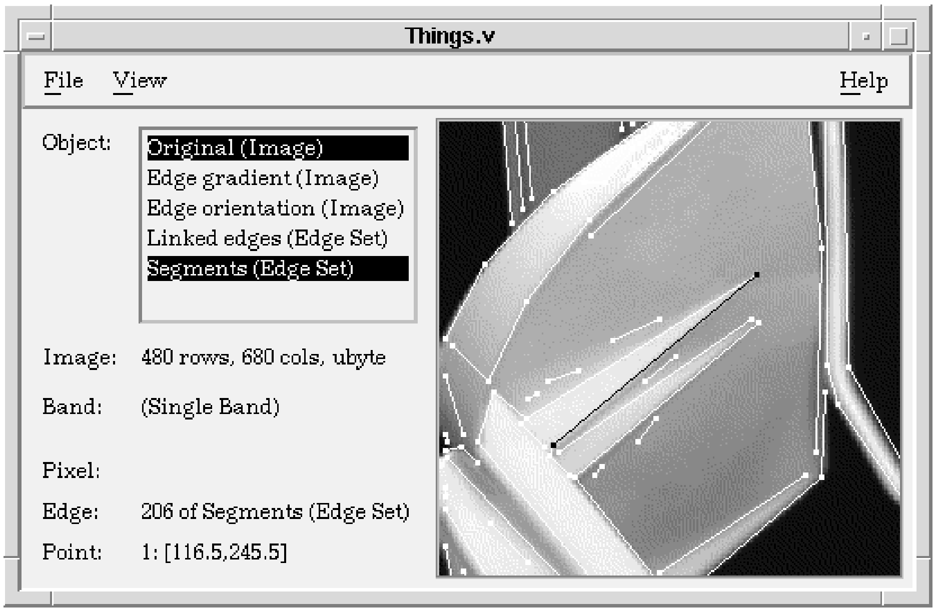

Figure 2: One of Vista's programs for viewing images and edge vectors.

3 Image format

Images are one type of object for which Vista includes built-in support. The image type is flexible enough to represent many of the kinds of images most commonly encountered, yet it remains simple and efficient. A Vista image includes a three-dimensional array of pixels indexed by row, column, and "band" number. The third dimension, band, can be used to represent any of four "virtual" dimensions: the discrete frames of a motion sequence, various camera viewpoints, multiple color channels, or the real and imaginary components of complex pixel values. It can also be used to represent some combination of these four dimensions according to a fixed hierarchy—allowing, for example, a single Vista image to represent a motion sequence of stereo pairs of RGB color images. Seven different pixel representations are supported, from a single bit to a 32-bit signed integer and a 64-bit floating point number. Vista's image processing routines and programs manipulate multi-band images as readily as a single-band ones, and most support all pixel representations equally well.

A Vista image also includes an attribute list that may contain any number of attributes describing the image. Among these are standard attributes, such as `nrows` and `ncolumns`, specifying the dimensions of the image and indicating how its various bands should be interpreted. However, the list may include any other, custom attributes needed to support particular applications (e.g., for a photometric stereo application, images may include attributes specifying light source locations). Custom attributes are carried transparently by Vista software—whenever a new image is created by processing an old one, the new image automatically inherits the attributes of the old. Consequently, existing software does not need to be modified whenever custom attributes are employed.

Although Vista's image representation is unique, it is easily mapped to other formats. Vista includes software for translating images in both directions between Vista's format and Portable Pixmap (PPM) format. The PBMPLUS package [9] can be used to reach a large variety of other formats from PPM format.

4 Using Vista programs

Most of Vista's data manipulation operations are available in the form of UNIX filter programs. Following the UNIX philosophy, each program is designed to perform just one particular task well. All programs follow a convention governing their command line arguments, allowing most options to be omitted or abbreviated, and all provide some on-line help. Some, like the viewer shown in figure 2, are interactive and have graphical user interfaces.

A typical program reads objects from a file or stream, manipulates the objects somehow, and writes the result to another file or stream; thus, programs can be "piped" together to perform a sequence of operations. Here, for example, is a sequence that detects edge pixels, links neighboring edge pixels into chains, and breaks the chains into approximating straight line segments:

```
vcanny < image.v -sigma 1.0 -noise 0.5 | \
  vlink -minlength 10 | vsegedges > segments.v
```

A program that processes one type of object will copy any other objects unchanged from input to output; thus, when files contain a mix of both standard objects (e.g., images) and custom ones (e.g., the `camera` object in figure 1), existing programs can still be used to process the standard objects. A program will also copy unchanged any unrecognized attributes, allowing even standard objects to carry customizing information.

5 Programming with the Vista library

Most of Vista's data manipulation operations are also available in the form of C-callable library routines that can be combined to build new modules. These routines, and the data structures they manipulate, are organized in an object-oriented fashion to hide nonpertinent implementation details. An image, for example, is represented in memory in a way that optimizes common operations; it includes an array of pixel values, various indexing arrays to speed pixel access, a structure with fixed fields for recording commonly-accessed image attributes, and a list containing any other image attributes. However, the programmer refers to this data structure by means of a single pointer and accesses it using a collection of macros and routines that shield most details. This is illustrated by the following code fragment, which zeros pixels in band 0 of an image:

```
VImage image; /* handle to the image */
int r, c;     /* row and column indices */

for (r = 0; r < VImageNRows(image); r++)
  for (c = 0; c < VImageNColumns(image); c++)
    VPixel(image, 0, r, c, UByte) = 0;
```

A typical image processing routine accepts one or two source images and returns a destination image. By convention, the caller has a choice of either supplying the destination image, or letting the routine create and return a suitable destination image. This gives the programmer full control over how memory is allocated for images.

When an image is read from a data file like the one illustrated in figure 1, it is automatically converted to the more-efficient, internal representation; when written to a file, it is converted back again to the file format. Using facilities of the library, the programmer can define a new, custom type and register it so that objects of that type are automatically converted between the file format and an efficient, internal format. The library can thus be readily extended to recognize and support new types of objects. Among the library's other general facilities are routines for handling errors, parsing command line arguments, and generating PostScript output depicting images and edges.

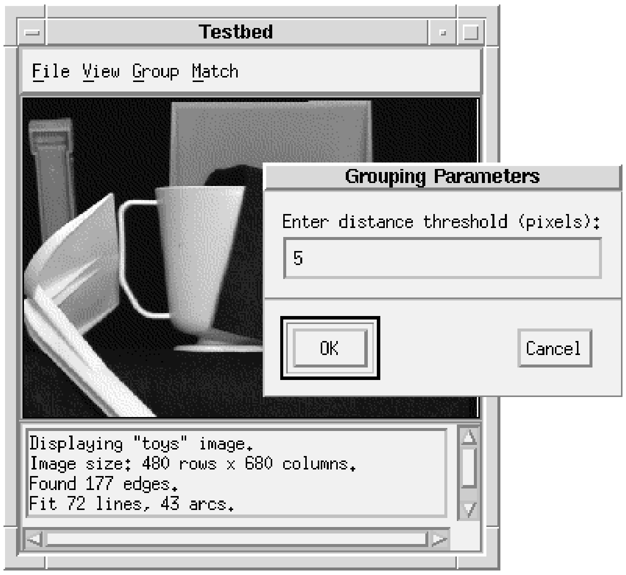

Figure 3: Appearance of a typical program constructed using Vista's VX package.

6 Creating interactive programs

When one is experimenting with an algorithm it is often helpful to be able to embed it within an interactive program that allows parameters to be adjusted and results viewed quickly. To aid this style of experimentation, the Vista library includes a package, called VX, for simplifying the task of creating programs that display images, have graphical user interfaces, and run under the X Window System. Using the VX package, and without knowledge of X Window System programming, one can create a program that will accept commands, parameters, and filenames through pull-down menus and pop-up dialog boxes, report messages in a scrolling text field, and display images with overlaid text and graphics; the displayed images can be updated by the program, and zoomed or panned either by the program or directly by the user through mouse actions. Although basic, these capabilities appear sufficient for many prototyping tasks in computer vision. Figure 3 shows the appearance of a typical VX-based program.

Like most programs implementing graphical user interfaces, a VX-based program has two phases. During the initialization phase, various calls are made to VX routines to parse program arguments and to define the commands that will appear on the menu bar. Each menu command is associated with a callback routine supplied by the programmer; callback routines can also be attached to the image display. During the execution phase, the program's operation is entirely driven by the user who, by selecting menu commands or image pixels, invokes various callback routines. The callback routines, in turn, use VX routines to pop up new dialogs, solicit parameter values, issue text messages, or update the displayed image. Programs organized this way are relatively easy to construct and modify. Moreover, where a VX-based program requires some user interface feature not supported by VX, underlying facilities of the X Window System can still be accessed directly.

7 Conclusion

Vista is available as a software environment for computer vision research. While it includes support for common data types such as images and edge vectors, it is also easily extended. Existing data types can be augmented and new ones added within the framework Vista provides. The environment is modular, lightweight, and easily mastered; it is written for portability and in a standard language; and it is freely available with no significant copyright restrictions. For these reasons, computer vision algorithms and data structures implemented in the context of Vista can be readily shared.

Not all of Vista's features can be described here. It does, however, come with extensive documentation. The most recent distribution of Vista can be obtained by anonymous FTP from the `/pub/local/vista` directory of `ftp.cs.ubc.ca`.

Acknowledgments

Vista was developed at the University of British Columbia's Laboratory for Computational Intelligence. In addition to the authors, those who made significant contributions were Ralph Horstmann, Johnny Kam, Daniel Ko, Richard Pollock, and Dan Razzell. Many others at UBC and elsewhere have participated in its testing and refinement.

References

[1] R. M. Haralick, "Computer vision theory: The lack thereof," *CVGIP* **36** (1986), pp. 372-386.

[2] K. Price, "Anything you can do, I can do better (no you can't). . . ," *CVGIP* **36** (1986), pp. 387-391.

[3] J. Rasure, D. Argiro, T. Sauer, and C. Williams, "Visual language and software development environment for image processing," *Int. J. of Imaging Systems and Technology* **2** (1990), pp. 183-199.

[4] A. Jacot-Descombes, M. Rupp, and T. Pun, "LaboImage: A portable window-based environment for research in image processing and analysis," in *SPIE Proc. Vol 1659: Image Processing and Interchange: Implementation and Systems* (1992), pp. 331-340.

[5] M. S. Landy, Y. Cohen, and G. Sperling, "HIPS: A Unix-based image processing system," *CVGIP* **25** (1984), pp. 331-347.

[6] P. Hallinan, M. Nitzberg, and G. Gordon, "The HVision Package: Image Processing Software at the Harvard Robotics Lab," Harvard Univ. Robotics Lab. Tech. Rep. 88-1, Jan. 1990.

[7] "KBVision," Amerinex Artificial Intelligence, Inc., Amherst, Mass. Software.

[8] J. Mundy, "The Image Understanding Environment program," In *Proc. CVPR* (1992), pp. 406-416.

[9] J. Poskanzer, "PBMPLUS," Dec. 1991. Software available on Internet.

Appendix: Summary of Vista features

Each of the following operations is available both as a stand-alone UNIX program and as library routine unless stated otherwise.

Viewing data: interactive programs for viewing images and edge vectors; toolkit for generating such programs; widget for displaying images with overlaid edges and other graphics; routines for allocating a standard palette of display colors; routine for popping up a window displaying an image.

Basic image processing: scale, crop, flip, transpose or rotate an image; adjust image brightness and contrast; perform an arithmetic or logical operation on each pixel of an image; convert an image from one pixel representation to another; convert a color image to gray-scale; compute pixel statistics; generate test images; corrupt an image with Gaussian noise.

Image filtering: convolve an image with a 2D or 3D kernel; convolve an image with a separable filter, such as a Gaussian or its derivative.

Fourier analysis: build a complex image from real and imaginary components; compute the Fourier transform of an image; compute the magnitude or phase of a complex image.

Edge detection: estimate image gradient in 2D or 3D; decompose gradient into magnitude and orientation; mark zero crossings; Canny's edge detector.

Edge organization: link edge pixels into curves; decompose curves into straight line segments.

Data file manipulation: combine multiple files into a single, multi-object file; combine multiple images into a single, multi-band image; select specified objects from a file; select specified bands from an image.

Format conversion: convert images to/from Portable Graymap/Pixmap format; read image from a text file.

Printing: render images, edge vectors, and flow fields as Encapsulated PostScript (EPS) documents; program for arranging multiple EPS documents on a page; routines for generating PostScript documents.

Other: estimate optical flow; camera calibration.