Vision Software for Teaching and Research

Mark W. Powell

Department of Computer Science & Engineering
University of South Florida
Tampa, FL 33620
mpowell @csee.usf.edu

Abstract

I introduce a software framework called the Java Vision Toolkit (JVT) for teaching and research in image processing and computer vision. The toolkit provides over 50 image operations and presents them to the user in a GUI that can render greyscale, color and 3D range images. The software is written in Java, enabling it to be integrated into HTML documents and interactive course materials. The framework is designed for extensibility using a source code template that supports the implementation of any new operation with a minimal amount of supporting code. For students, this framework encapsulates the GUI, file I/O and other trivial programming details and allows them the maximum amount of time to spend on understanding computer vision. For researchers, the process of getting started on a new research project is made more efficient in several important ways. First, JVT integrates many useful computer vision algorithms into a intuitive GUI for quickly generating results, regardless of the user's experience in computer vision. JVT also stimulates code reuse within research groups by providing a foundation upon which to build new algorithms. Additionally, this framework increases the reproducibility and potential for availability of implementations used to create published results, since it the source code is architecture-neutral. Lastly, I compare the JVT with other computer vision software frameworks that are used for teaching and research.

Key words: Java programming, computer vision software

1 Introduction

> "Writing computer vision programs isn't easy. Debugging them (or even telling whether they contain a bug) is harder. Conventional programming courses don't really give students the tools they need to tackle these problems." - Olson [1]

Computer vision algorithms are among the most complicated to implement in software of any area of computer science. New students who are encountering vision methods for the first time find that they must delve into statistics, calculus, physics, linear algebra, artificial intelligence, etc. Students are often expected to implement a number of computer vision concepts in software to better understand the material, to develop skills for performing well in an industry role, and to make it relevant to their own life experience. At this point, in addition to the abundance of diverse material the students must absorb in order to understand the theory, they encounter implementation hurdles in the form of GUI programming, file I/O, and miscellaneous programming details in the process of putting theory into practice. In order to maximize the amount of time available to the student to spend on the road to understanding imaging algorithms, we must eliminate the programming details that are irrelevant to computer vision.

I submit the premise that anything that the student needs to implement before the point of receiving one or more input image data structures and after creating one or more output data structures is irrelevant to understanding image computation. Any time spent in implementing image display, file I/O, or any supporting code that integrates image computation with any particular environment is time wasted for the student. Issues like these are better covered in a more robust fashion in a class such as computer graphics (for rendering images) or software design (for implementing robust file filtering routines). Ideally, all a student requires is a template for image computation in the form of a function header through which source images are provided and destination images and/or resultant output is returned.

Another important part of maximizing student understanding is the presentation of course materials in an interactive media, as in [2][3]. An effective software framework for teaching computer vision should be able to be integrated into web-based presentations and also be accessible to students for building projects. This capability would also allow researchers to provide demonstrations of their recent works in a format which is easily integrated into classroom instruction, as well as making their published work more easily reproducible and extensible for others.

The Java Vision Toolkit (JVT) introduced here provides this framework. Once installed, it can read, write and display images, and perform an assortment of image computations using a GUI called "jv" that is similar in appearance and use to Adobe Photoshop. Image computations are selected from a menu, and the output images are rendered in the GUI. The operations menu is extendible, and allows for the easy addition of customized computations designed by the student or instructor. A source code template is provided that the programmer can use to implement a new image computation, compile the code, and instantly see the output in the GUI context. Image reading, writing, printing and display are handled by the the JVT behind the scenes, so the student need only be concerned with transforming pixel values or higher level vision operations. The JVT is free for educational use, so there is no cost to the student or institution, unlike commercially developed packages such as Matlab [4] or Khoros [5]. JVT is written in Java, so a student can program with it on their PC at home or at school as well as a X-terminal or workstation in a research lab, using the same source code for either platform. Although the performance of Java is notoriously slow compared to equivalent C code or other compiled languages, the majority of the JVT image computations are implemented in native C for Windows or Solaris operating systems and hardware for optimum performance. [6] identified the problem of student projects written in a rapid-prototyping software development environment (in this case, Khoros) not having the performance that real-time robotics applications (and likewise, many industrial vision systems) require. The JVT has the potential to bridge that gap.

For interactive textbooks, the performance advantage of JVT over the original Java imaging paradigm (java.awt.image) removes the major concern of instructors and programmers who are interested in creating such content [7]. Java applets for HTML documents can use the JVT classes to implement image computations with high computational efficiency. JVT also stimulates code reuse within research groups by providing a foundation upon which to build new algorithms. Additionally, this framework increases the reproducibility and potential for availability of implementations used to create published results, since the source code is architecture-neutral. The JVT also makes programming such imaging operations much simpler compared to the standard Java awt.image API.

2 Related Work

Matlab [4] is an effective tool for computing mathematical transformations on arrays of numeric data such as images. The image processing toolkit for Matlab provides functions to read and write images in various formats and a set of image computations such as convolution, frequency domain filtering, and binary image operations. Matlab is very effective for rapidly prototyping chains of image computations to get quick results. As a general-purpose API, however, it has several shortcomings: it is interpreted, it is restricted to running in the Matlab environment, and it lacks the platform independence and web presentation capabilities of the JVT.

Khoros [5] is another commercial image processing package that can be used to implement image computation. It offers a large number of image operations as well as pattern classification and data visualization capabilities. It uses a visual programming paradigm where operators are laid out as icons in a workspace and connected in a directed graph. As a programming tool, it is confined to the Khoros environment, which limits its availability to the Unix and Windows platforms that it supports, and does not support web-based instruction and demonstration.

ImageMagick [8] is a freely available image format conversion and display program which includes a few image processing operations such as histogram equalization, crop, zoom, etc. and has an open source C library that can be extended to include new functionality. The API is very low level and the documentation is quite terse, so it takes substantial time to learn and use effectively. XV [9] is a similar package with similar advantages and disadvantages as ImageMagick, with the additional burden of being shareware, even for educational use.

CVIPtools [10] is an excellent framework for image computation. It is a library of vision algorithms written in C and Tcl/Tk (for the GUI), and provides a large number of computer vision algorithms, including texture measures and color segmentation algorithms as well as standard point, frequency, convolution and binary image operations. The documentation included with CVIPtools is extensive and the software is available for Unix and Windows platforms. The performance of CVIPtools is very good, since it is compiled to native machine code from C. The fact that it is written in C, however, makes it unable to support web-based interaction. Although the CVIPtools API is similar to JVT in terms of providing a template for image computation that specifies the input image, output image and parameters, because it is written in C, making the programmer responsible for memory management. Tracking down memory access errors accounts for a substantial percentage of overall software debugging time, and this wasted time is eliminated through garbage collection in Java. Given that computer vision students typically have only moderate programming experience at best, and novice level experience at worst (they may not even be computer engineering majors), not being responsible for managing memory is a positive boon.

2.1 Other Java Vision Frameworks

Image/J [11] is a pure Java image processing toolkit developed at NIH. It has a GUI which lets the user read, write and display images in a variety of formats and provides a number of image calculations. It has javadoc documentation and has been used and extended by researchers outside of NIH for general computer vision implementations. Since it is pure

Java and does not use native C implementation of image operators, it is less computationally efficient, but no less effective than JVT and its peers. Image/J is an open source library that can be used for web presentation and general vision programming, but it uses the java awt.image API for representing images, which is less intuitive than the JVT API.

JIGL [12] is another Java image processing toolkit developed at BYU. It uses its own class called jigl.image to represent binary, greyscale and color images in integer or float datatypes. The API has a demo applet that demonstrates its available image computations, such as morphology, FFT, connected components and thresholding. Like Image/J, it is a pure Java implementation and therefore suffers in performance. Although it is more intuitive to use the jigl.image API than Image/J, the lack of a cohesive GUI and relatively small number of available operators makes it less useful as a research tool on its own in its current form.

NeatVision [13] (formerly JVision) is a Java image processing API that uses a visual programming environment to read, write, display and perform chains of image computations. It can use either a pure Java implementation for portability, or a native C optimization for high performance. It is shareware, and the API is only available to developers by special upgrade.

Table 1 is a feature comparison between all of the computer vision software frameworks mentioned herein.

3 Framework

3.1 JVT

The JVT framework is implemented in Java and is capable of reading, writing and displaying greyscale, color and 3D range images from files in a variety of formats. JVT uses the Java Advanced Imaging API (JAI), which provides routines for reading and writing 2D images in PNM, JPEG, GIF (read-only), PNG, FPX, TIFF and BMP formats. The Java Advanced Imaging API also provides support classes for implementing additional file format filters that it does not support by default. These classes are used in the JVT to read and write 3D

	Matlab	Khoros	CVIP tools	Image Magick	xv	JIGL	Image /J	Neat Vision	JVT
Histogram operations	Y	Y	Y	Y	Y	Y	Y	Y	Y
Convolution	Y	Y	Y	Y	Y	Y	Y	Y	Y
Image arithmetic	Y	Y	Y	Y	Y	Y	Y	Y	Y
Morphology	Y	Y	Y	N	N	Y	Y	Y	Y
Freq. domain operations	Y	Y	Y	N	N	Y	N	Y	Y
Color space conversion	Y	Y	Y	N	N	Y	Y	Y	Y
Median filter	Y	Y	Y	Y	Y	N	Y	Y	Y
Geometric transforms	Y	Y	Y	Y	Y	Y	Y	Y	Y
Statistical operations	Y	Y	Y	N	N	N	Y	Y	Y
Grad. mag. edge detect	Y	Y	Y	Y	Y	N	Y	Y	Y
RGB thresh.	Y	Y	Y	N	N	N	N	Y	Y
Color segmentation	N	Y	Y	Y	N	N	N	N	Y
Texture Analysis	N	Y	Y	N	N	N	N	Y	Y
ROI support	Y	Y	N	N	Y	Y	Y	Y	Y
Range Image Operations	N	N	N	N	N	N	N	N	Y
Availability	Comm	Comm	Free	Free	Shar	Free	Free	Free	Free
Language	Prop	Prop	C	C	C	Java	Java	Java	Java

Table 1: Feature comparison between computer vision software frameworks. Comm indicates commercial, Prop indicates proprietary, and Shar indicates shareware.

range images with registered intensity from the K^2T GRF-2 structured light scanner and the Cyberware 3030 laser range finder. The JAI class library consists of mainly image operations that would be classified as image processing rather than computer vision, such as histogram equalization, filtering, frequency operations, etc. The JVT adds a number of computer vision operations to this library, such as segmentation, texture measures, morphology, and Hough transform techniques. The JVT also uses the Java 3D API which allows the user to render 3D images in a 2D window and perform real-time zoom, rotation and translation of the 3D

point set for range image visualization.

3.2 JAI

The JAI API currently includes support for over 80 common image computations, such as histogram functions, convolution, Fourier transforms, image zoom, rotate, crop and warp, color space conversion, lookup table modification, etc. [14] Although other Java software libraries [11][12] also provide some or all of this functionality, their are several advantages of the JAI API over its peers. Firstly, and probably most importantly, the image computations that it supports are implemented in C, which is optimized for both Sun hardware (through VIS instructions) and Intel platforms (through MMX instructions), which maximizes their computational efficiency.

The JAI API also includes support for distributed processing using a class called RemoteImage, which allows various processes to communicate via RMI to perform parallel computations. This functionality is made even more useful by the TiledImage class, which allows large images to be split up into an arbitrary number of subimages or tiles. An operation on a TiledImage can be any operation which requires only a single pixel or window of pixels at each iterative step. Such an operation can be performed on each tile individually and the results from the individual tiles are later merged. These tile computations can then be assigned to different processors (at the O/S layer on a multiprocessor machine) for parallel computation on a single image. Finally, because the library is implemented in Java, all of the platform independence characteristics of the language are present, allowing the same code to run on either a Unix or Windows platform. The C implementations of the image computations can be replaced with a pure Java implementation, making the API portable to any platform with a Java 1.2 virtual machine at the cost of computational efficiency. At this time, support for Solaris and Windows is available. With the recent release of JDK 1.2.2 for Linux, a port of the JAI API to that system has been announced, but does not exist at the time of this writing.

As of this writing, the JAI API does not include support for volumetric image data structures or operations on them, but the proposed specification for the version 1.1 API will include this [15]. The JVT includes a RangeImage class which provides a data structure for a range image and operations on them such as cropping, 2D quantization and surface normal estimation.

3.3 JAI API example

The JAI API supports many common image computations directly. For these operations, all that is required is to supply the source image(s), set the parameters (if any) and instantiate the appropriate class. Algorithm 1 is an example of performing Sobel edge detection on an image using the JAI API.

Algorithm 1 Sobel edge detection using the Java Advanced Imaging (JAI) API.

```
//Load the image
PlanarImage source = (PlanarImage)JAI.create(‘‘FileLoad’’, ‘‘lena.ppm’’);

//create the two kernels
float horizontal[] = new float[] { -1., 0., 1.,
                                   -2., 0., 2.,
                                   -1., 0., 1. };
float vertical[] = new float[] { 1., 2., 1.,
                                 0., 0., 0.,
                                 -1., -2., -1. };
KernelJAI kernelH = new KernelJAI(3, 3, horizontal);
KernelJAI kernelV = new KernelJAI(3, 3, vertical);

//create the Gradient operation
PlanarImage dest = (PlanarImage)JAI.create(‘‘GradientMagnitude’’,
                                           source, kernelH, kernelV);
```

In the first statement, the source image is in a file named “lena.ppm” that is loaded from the current directory using the FileLoad operator. In the next four statements, the KernelJAI class is used as a generic data structure for defining a convolution kernel of any particular size. In the last statement, the Sobel edge image is computed using the GradientMagnitude

operator. The JAI class, referenced in the first and last statements, is a convenience class used to instantiate operations such as file I/O and image computation. The JAI API classes make performing this type of operation simple to implement, requiring only the necessary parameters for the operation and hiding the implementation details from the user. This allows more time to be spent on higher-level understanding of vision issues such as analyzing varying performance between similar operators, since the operations themselves are quickly implemented and their results are immediately visible.

3.4 JVT API

The JAI API implements only a subset of the commonly-used operations in image processing and virtually none of the operations used in computer vision. The JVT extends the JAI by implementing operations for segmentation, texture, Hough transforms, morphology and range image processing. A complete listing of all the imaging operations available in the JVT is given in Table 2. Also, it is sometimes desirable to get deeper into the details of programming image operations and work with actual pixel values. This is where the JVT adds value to the JAI API by including more intuitive image classes which extend JAI.

3.5 JVT API example

Algorithms 2 and 3 are the image computation template from the JVT API. This template provides all of the supporting code for image display, I/O, and integration with the new operation. The template consists of two files, TemplateDescriptor and TemplateOpImage. The TemplateDescriptor contains the parameter specification for the operation and includes the "main" method which may be invoked for testing. By default, invoking the operator from the command line will load an image called "test.pgm", perform the custom operation, and display the result in a scrolling window. For quick results, no changes are necessary in the TemplateDescriptor code, but ideally the programmer would change the name from "template" to the name of their custom operation and add commented documentation. The

Category	Name
Arithmetic	Add two images
	Subtract two images
	Multiply two images
	Divide two images
	Max of two images
	Min of two images
Logical	AND two images
	OR two images
	XOR two images
	NOT
Histogram	Histogram Equalization
	Histogram Normalization
Filter	Gaussian
	Mean
	Median
Frequency	Magnitude
	Magnitude Squared
	Phase
Point	Invert
	Log
	Overlay two images
Range	Crop Range
	Cosine Shade
	Remove Intensity Shading
Lookup	Color linearization
	Randomize

Category	Name
Edge	Sobel
	Roberts
	Prewitt
	Frei-Chen
Segmentation	Binary Threshold
	RGB Threshold
	SCT
	HSI
	PCT/Median Cut
	Connected Components
Morphology	Expand
	Shrink
Hough	Line Finder
	Circle Finder
Texture	Laws texture measures (8)
	GLCM statistics
Affine	Zoom
	Rotate
	Scale
	Flip Horizontal
	Flip Vertical
	Crop
Convert	Linear RGB to Grey
	Nonlinear RGB to Grey
Lighting	Locate Light Sources
	Locate Lights using Range

Table 2: Listing of JVT image operations.

TemplateOpImage contains the actual code for the operation. The implementation is in the "compute" method. It takes a source image(s) and destination image as argument, modifying the destination image as required.

Algorithm 4 is an example implementation of a morphological expansion operator using the JVT classes in concert with the JAI API.

The first statement is a package statement, indicating that this class belongs to the JVT toolkit. All classes that implement image operations not included in the JAI API belong to this package, as well as a number of support classes. The import statements following

Algorithm 2 The JVT source code template for image computation (part 1).

```
public class TemplateDescriptor extends OperationDescriptorImpl
implements RenderedImageFactory {
  public TemplateDescriptor() {
    super(resources, 1 , paramClasses, paramNames, paramDefaults);
  }
  public RenderedImage create(ParameterBlock pb, RenderingHints hints) {
    if (!validateParameters(pb))
      return null;
    return new TemplateOpImage(pb.getRenderedSource(0), new ImageLayout());
  }
  public boolean validateParameters(ParameterBlock pb) { return true; }
  private static final Class[] paramClasses = { };
  private static final Object[] paramDefaults = { };
  public static void main(String arg[]) {
    OperationRegistry or = JAI.getDefaultInstance().getOperationRegistry();
    ParameterBlock pb = new ParameterBlock();
    TemplateDescriptor templateDescriptor = new TemplateDescriptor();
    OperationDescriptor odesc = templateDescriptor;
    RenderedImageFactory rif = templateDescriptor;
    or.registerOperationDescriptor(odesc, "template");
    or.registerRIF("template", "edu.usf.csee", rif);
    RenderedOp source = JAI.create("fileload", "test.pgm");
    pb.addSource(source);
    RenderedOp dest = JAI.create("template", pb, null);
    Frame window = new Frame("JVT Template Test");
    window.add(new ScrollingImagePanel(dest,
      dest.getWidth(), dest.getHeight()));
    window.pack(); window.show();
  }
}
```

Algorithm 3 The JVT source code template for image computation (part 2).

```
public class TemplateOpImage extends UntiledOpImage {
  public TemplateOpImage(RenderedImage source, ImageLayout layout) {
    super(source, null, layout);
  }
  protected void computeImage(Raster src, WritableRaster dst,
                              Rectangle destRect) {
    /* COMPUTE IMAGE HERE */
  }
}
```

Algorithm 4 A morphological expansion operator from the JVT.

```
package edu.usf.csee.media.jai;
import java.awt.Rectangle;
import java.awt.image.*;
import javax.media.jai.*;
/** Morphologically expands a binary image. 4-connected regions are assumed.
  * @author Mark Powell
  * @author Mark.Powell@computer.org
  * @version beta */
public class ExpandOpImage extends UntiledOpImage {
/** Constructs ExpandOpImage. Image dimensions are copied from the
  * source image. The tile grid layout, SampleModel, and ColorModel may
  * optionally be specified by an ImageLayout object
  * @param source a RenderedImage
  * @param layout an ImageLayout optionally containing the tile grid layout,
  * SampleModel, and ColorModel or null. */
  public ExpandOpImage(RenderedImage source, ImageLayout layout) {
    super(source, null, layout);
  }
  /** Morphologically expands the image (4-connected).
    * @param src the source raster.
    * @param dst the resultant connected component image.
    * @param destRect the rectangle within the OpImage to be computed */
  protected void computeImage(Raster src, WritableRaster dst, Rectangle destRect) {
    Raster2 source = new Raster2(src);
    WritableRaster2 dest = new WritableRaster2(dst);
    int width = source.getWidth(), height = source.getHeight();
    /* Implement op here */
    //Assume we have a binary image where 0->0 and 1->255
    //Assume 4-connected
    final int white = 255;
    final int black = 0;
    for (int v = 1; v < height-1; v++)
    {
      for (int u = 1; u < width-1; u++)
      {
        if ((source.grey(u,v)) == black)
        {
          if (source.westNeighbor(u,v) == white || source.eastNeighbor(u,v) == white ||
              source.northNeighbor(u,v) == white || source.southNeighbor(u,v) == white)
          {
            dest.setGrey(u, v, white);
            continue;
          }
          dest.setGrey(u, v, black);
        }
        else
        {
          dest.setGrey(u, v, white);
        }
      }
    }
  }
}
```

the package statement identify classes outside of this package that are referenced within this class. The comments beginning with /** and ending in */ are specialized documentation comments which are extracted by the HTML documentation generator called javadoc. The inclusion of these comments both encourages good programming practice and code reuse by the automatic generation of hypertext documentation in HTML form for the library of computer vision operators. The UntiledOpImage class from the JAI API embodies an image computation that transforms one or more source images to a destination image, and taking zero or more input parameters for the image computation. The ExpandOpImage class inherits from the UntiledOpImage class, has its own constructor and overrides compute(), where the image computation takes place. The constructor receives any necessary input parameters, passed in via the header. The compute() method takes a source image(s) as input, represented by a Raster object(s), and stores the destination image in a WritableRaster object. These objects are the first two arguments to the compute() method. The third argument, destRect, is ignored. The Raster2 and WritableRaster2 classes inherit from the JAI API Raster and WritableRaster classes and provide highly intuitive pixel accessor methods. A partial listing of these accessor methods from the Raster2 class is given in Table 3. These methods allow implementations of image computations to refer to greyscale pixel values and color pixel values in an intuitive way and also to compute simple statistics and examine neighboring pixel values easily. The Raster2 and WritableRaster2 classes can be extended further through inheritance and customized for any specific area of computer vision. This flexibility is afforded by the underlying DataBuffer object, which can be instantiated with any required size, number of bands, and pixel data type (byte, int, float, double, complex, etc.) Through inheritance, any default pixel access or image computation method may be overridden with new functionality that is needed for any sort of image computations that break with “conventional” standards, such as 6-connected neighbors.

Method name	Description
public int grey(int u, int v)	Returns the pixel value in band 0 at the given position.
public int red(int u, int v)	Returns the red pixel value in band 0 at the given position.
public int green(int u, int v)	Returns the green pixel value in band 1 at the given position.
public int blue(int u, int v)	Returns the blue pixel value in band 2 at the given position.
public int minGrey()	Returns the minimum pixel value in band 0.
public maxGrey()	Returns the maximum pixel value in band 0.
public int greyMean()	Returns the mean pixel value in band 0.
public int greyCovariance()	Returns the covariance of the pixels in band 0.
public int eastNeighbor(int u, int v)	Returns the pixel value in band 0 at position east of the given position.
public int westNeighbor(int u, int v)	Returns the pixel value in band 0 at position west of the given position.
public int northNeighbor(int u, int v)	Returns the pixel value in band 0 at position north of the given position.
public int southNeighbor(int u, int v)	Returns the pixel value in band 0 at position south of the given position.

Table 3: Some accessor methods of the Raster2 class in the JVT.

3.6 JVT Interface

Figure 1 shows the JVT GUI, called "jv". The GUI can be invoked from the command line or desktop or run from a web browser. The "jv" window is the main window, where images are rendered. Operations such as Open, Save, Zoom, Threshold, etc., are invoked from the jv menu. Jv can render greyscale, color and 3D range images. In Figure 1, a color image of a skin sample under magnification and a 3D color and range image of a toy dumptruck is shown. For range images, the window is split to allow simultaneous viewing of the 2D color or greyscale image, and the corresponding range image. The display is interactive, allowing region of interest selection, pixel value querying, and 3D range image rotation, translation, and zoom via mouse. The "Source" window provides basic image statistics for the currently

selected image, and also allows the selection of a stack of images for performing computations over more than one image, such as image subtraction. The "Histogram" window displays a histogram, either greyscale or color, where appropriate, for the currently selected image. The "Options" window contains parameter fields in a tabbed pane interface where the user may input image computation parameter settings.

4 Discussion

We presented a new software framework for computer vision, JVT, which is available online at http://marathon.csee.usf.edu/~mpowell/jvt/index.html and free for educational use. It is designed with the student in mind, providing support for image display and I/O and putting emphasis on implementing image computation effectively. Although it is a Java-based framework, much of the image computation is coded in C libraries that are compiled for high performance. The JVT also supports publication of research-related implementations and interactive course materials.

Development of JVT is ongoing, and the main future directions for it are the progressive addition of more image computation facilities to the framework, solicitation of contributions from other researchers, and use of the framework in a classroom environment.

References

[1] Tom Olson, "CVPR panel discussion on teaching image computation," http://figment.csee.usf.edu/teach_res/olson.html, 1996.

[2] R. Jordan and R. Lotufo, "Interactive digital image processing course on the world wide web," *IEEE International Conference on Image Processing*, vol. 2, pp. 433–436, September 1996.

[3] R. Lutofo and R. Jordan, "Hands-on digital image processing," *IEEE 26th National Conference on Frontiers in Education*, vol. 3, pp. 1199–1202, November 1996.

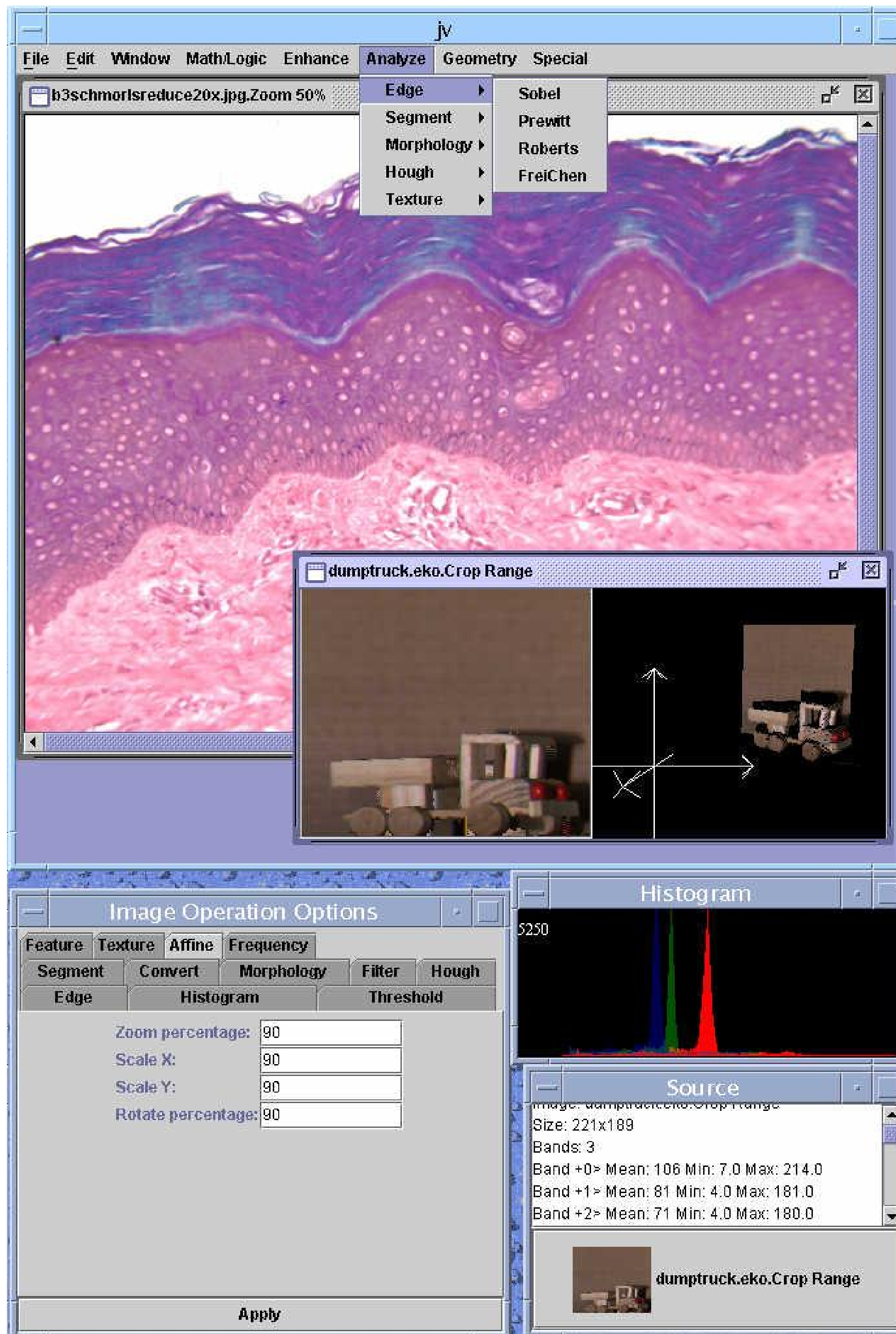

Figure 1: Jv, the JVT graphical user interface, shown here rendering a color image of a magnified skin sample and a 3D range and color image of a toy dumptruck.

[4] *Matlab*, http://www.mathworks.com/.

[5] *KHOROS*, http://www.khoral.com/.

[6] R. R. Murphy, "Teaching image computation in an upper level elective on robotics," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 12, no. 8, pp. 1081–1093, December 1998.

[7] R.B. Fisher and K. Koryllos, "Interactive textbooks; embedding image processing operator demonstrations in text," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 12, no. 8, pp. 1095–1123, December 1998.

[8] *ImageMagick*, http://www.wizards.dupont.com/cristy/ImageMagick.html.

[9] *xv*, http://www.trilon.com/xv/.

[10] S.E. Umbaugh, *Computer Vision and Image Processing: a practical approach using CVIPtools*, Prentice Hall, http://www.ee.siue.edu/CVIPtools/, 1999.

[11] *Image/J*, http://rsb.info.nih.gov/ij/.

[12] *JIGL*, http://rivit.cs.byu.edu/jigl/.

[13] *NeatVision*, http://www.neatvision.com.

[14] Sun Microsystems, Inc., http://www.javasoft.com/products/java-media/jai/, *Programming in Java: Advanced Imaging.*

[15] *JSR-000034 Java Advanced Imaging API 1.1*, http://java.sun.com/aboutJava/communityprocess/jsr/jsr_034_jai.html.

Contents

1 Introduction **2**

2 Related Work **4**

2.1 Other Java Vision Frameworks 5

3 Framework **6**

3.1 JVT 6

3.2 JAI 8

3.3 JAI API example 9

3.4 JVT API 10

3.5 JVT API example 10

3.6 JVT Interface 15

4 Discussion **16**

List of Figures

1 Jv, the JVT graphical user interface, shown here rendering a color image of a magnified skin sample and a 3D range and color image of a toy dumptruck. . 17

List of Tables

1 Feature comparison between computer vision software frameworks. Comm indicates commercial, Prop indicates proprietary, and Shar indicates shareware. 7

2 Listing of JVT image operations. 11

3 Some accessor methods of the Raster2 class in the JVT. 15