IMPLEMENTATION OF JAVA TECHNOLOGIES IN IMAGING

Abhineet Sinha, Vishal Kumar, Mohit Panchal
Student, B.tech, Electronics and Computers Engineering,
Dronacharya College of Engineering, Khentawas, Gurgaon

Abstract- Digital imaging in Java has been supported since its first release, through the java.awt and java.awt.image class packages. Now there are three distinct java imaging APIs namely , the basic AWT imaging API, Java 2D API and Java Advance Imaging (JAI) API. This paper firstly summarizes and compares important features of AWT, Java2D and JAI APIs. Then the Java2D and JAI technologies are compared based on practical results obtained by evaluation of convolution.

Index Terms– **AWT, PixelGrabber, RMI**

I. INTRODUCTION

Image Processing or Imaging has an important role to play in many fields like entertainment, health care, internet and also high tech areas like medical imaging, satellite imaging, and astronomy. Since Java technology gives the flexibility through its “write once run anywhere” concept and provides many advanced image processing features it is well suited for developing image processing applications which can run on almost any machine.

Early versions of the Java AWT provided a simple rendering package suitable for rendering common HTML pages, but without the features necessary for complex imaging. The Java 2D API extended the early AWT by adding support for more general graphics and rendering operations. The Java Advanced Imaging (JAI) API further extends the Java platform (including the Java 2D API) by allowing sophisticated, high-performance image processing to be incorporated into Java applets and applications.

II. THEORETICAL COMPARISON BETWEEN AWT, Java2D AND JAI APIs

The AWT class for image representation is **java.awt.image.** In Java 2D primary image representation class is **java.awt.BufferedImage.**

javax.media.jai.PlanerImage is central.

Image data representation is important in image analysis. In AWT when an image is loaded, pixels are encapsulated in the Image object. The Image class in java.awt.image has no methods for reading or writing pixels directly from an Image object. To read pixels one needs to extract them using the PixelGrabber class. AWT uses a single element to represent the pixel components and color model, and to interprete the pixels. Image data is stored as as array. Java 2D uses SampleModel and ColorModel classes associated with BufferedImage to read and write pixel data. Data is stored in a DataBuffer object. JAI, in addition to SampleModel and ColorModel, has classes that extend both these two classes .

In imaging, an operation is often performed on a set of images. AWT & Java2D does not support image sets. JAI has several sophisticated classes that support image collection. It makes use of **collection** data structure already available in Java.

In case of handling large images , AWT cannot handle large images. Java 2D is not well suited to handle large images because immediate mode model stores the entire image in cache. JAI because its tile based has good support for handling large images.

AWT has no explicit support for network imaging. Java 2D has no explicit support for network imaging. JAI explicitely makes use of RMI(remote method invocation) for server side imaging. It also has operators that support IIP (Internet Imaging Protocol).

For sophisticated image geometry or data manipulation operations imaging operators are very important. AWT has no imaging operators . Java 2D supports only a few basic single input/single output operations in the form of classes like AffineTransformOp, ConvolveOp, LookupOp etc. JAI has a large array of imaging operators that can assist in the I/O, display, manipulation, enhancement, and analysis of images.

Looking into the support for image properties, AWT supports width and height properties. Java 2D supports width and height properties. JAI has extensive support for property management. One can create properties such as Region of Interest(ROI) and save them along with the image.

Finally, AWT supports only GIF and JPEG image loading. Starting with jdk1.3 this has extended to PNG. Java 2D has no explicit support for image loading, saving APIs. The image I/O package in JAI 1.0.2 and 1.1 supports several image formats, including GIF, JPEG, BMP, TIFF, PNG, PPM, and Flash\Pix. Except for GIF and FlashPix images in all of these formats can be saved.

III. PRACTICAL COMPARISON BETWEEN Java2D

AND JAI APIs

The practical comparison is done between Java2D and JAI APIs based on the operation of convolution. Convolutions are useful for a wide variety of digital image processing operations, including smoothing of noisy images (spatial averaging) and sharpening of images by edge enhancement, utilizing Laplacian, sharpening, or gradient filters. In addition, local contrast can be adjusted through the use of maximum, minimum, or median filters, and images can be transformed from the spatial to the frequency domain with convolution kernels.

In the simplest form, a two-dimensional convolution operation on a digital image utilizes a two-dimensional convolution kernel $K = \{k[i][j]\}_{i,j=-m,\ldots,-1,0,1,\ldots,m}$. Convolution kernels typically feature an odd number of rows and columns in the form of a square, with a 3 x 3 pixel mask being the most common form, but 5 x 5 and 7 x 7 kernels are also frequently employed. The convolution operation is performed individually on each pixel of the original input image $F = \{f[i][j]\}_{i,j=0,1,\ldots,n-1}$ so that the pixels in the output image are given by the equation

$$g[x][y] = \sum_{i=-m}^{m} \sum_{j=-m}^{m} k[i][j] \cdot f[x-i][y-j]. \quad (1)$$

To perform a convolution on an entire image, the sum operation must be repeated for each pixel of the original image. Thus, convolution is computationally very intensive and hence evaluation and comparison of total processor time involved to execute this operation in different Java technologies is important.

3.1 Dependency of the Execution Times on the Kernel Data

Firstly, we would like to find the dependency of the execution times on the kernel data. The execution times have been measured for the 3*3 kernels for edge detection, smoothing, and blurring. From Table 1 and 2 we can find that the convolution method in the Java 2D API does depend on the kernel data since the similar method for the JAI API doe not depend. Moreover, the same conclusions have been found for the similar 5*5, 7*7 and 9*9 kernels.

Table 1. Java2D Execution Times for for 3*3 Kernels for Edge Detection, Smoothing, Bluring.

Image Size	11.6	19.2	33.3	47.2	154	275
Edge Detection	17	23	45	78	334	1104
Smoothing	15	18	35	56	265	551
Blurring	10	21	70	102	345	728

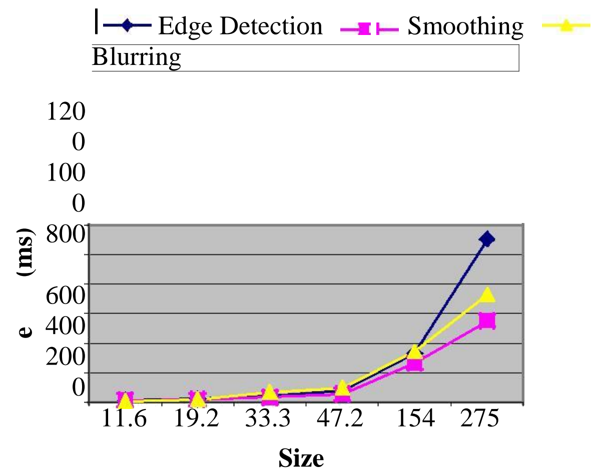


Figure 1.

Table 2. JAI Execution Times for 3*3 Kernels for Edge Detection, Smoothing, Bluring.

Image Size	11.6	19.2	33.3	47.2	154	275
Edge Detection	26	28	42	46	53	70
Smoothing	26	28	42	44	53	69
Blurring	26	29	43	46	54	71

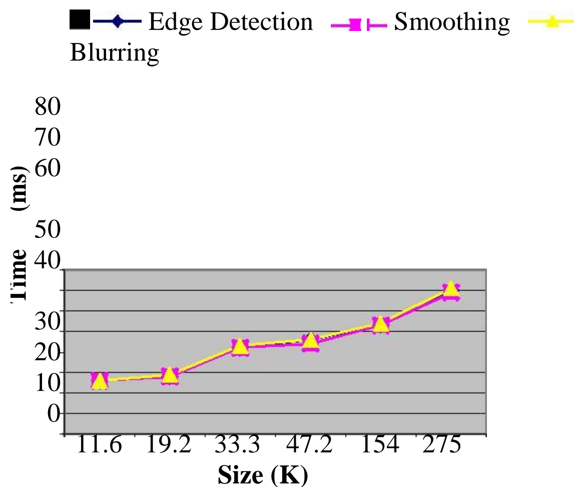


Figure 2.

Table 4. JAI Execution Times for the 3*3, 5*5 and 7*7 Blurring Kernels.

Image Size	11.6	19.2	33.3	47.2	154	275
3*3 Kernel	26	28	42	46	53	70
5*5 Kernel	25	29	42	45	52	72
7*7 Kernel	24	27	40	47	51	71

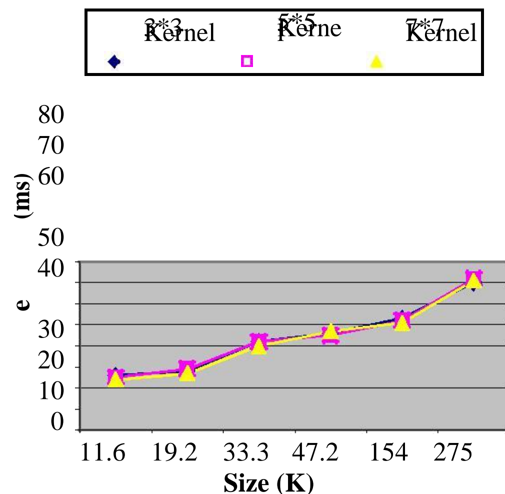


Figure 4.

3.2 Dependency of the Execution Times on the Kernel Size

The second practical comparison tests how the convolution depends on the kernel sizes. In this case the execution times have been found for the 3*3, 5*5, 7*7 blurring kernels (see Table 3, 4). As we expect the Java 2D convolution depends on the kernel size. More importantly, we have found that the JAI convolution does not depend on the kernel size. The transformation only depends on the image size.

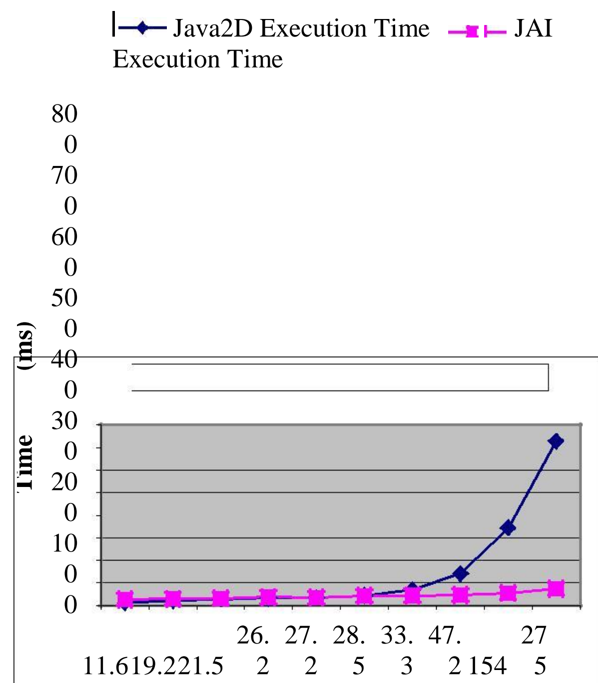
Table 3. Java 2D Execution Times for the 3*3, 5*5 and 7*7 Blurring Kernels.

Image Size	11.6	19.2	33.3	47.2	154	275
3*3 Kernel	12	23	68	110	352	740
5*5 Kernel	56	68	104	328	682	1167
7*7 Kernel	362	451	709	2325	4882	8384

Figure 3.

3.3 Direct Comparison of Java2D and JAI

Finally, a direct comparison between Java 2D and JAI APIs has been done based on a benchmark containing a sequence of 3*3,



Size (K)

Figure 5.

Table 5. Execution Times for Java 2D vs JAI.

Image Size	11.6	19.2	21.5	26.2	**27.2**	28.5	33.3	47.2	154	275
Java2D	10	21	26	34	**35**	45	70	140	345	728
JAI	26	29	31	36	**35**	42	43	46	54	73

IV. CONCLUSION

Results from above tables and graphs indicate that JAI is independent of change in Kernel Data or Kernel Size for Convolution Operation , while Java2D is dependent on change in Kernel Data or Kernel Size used to perform Convolution Operation. Also while comparing Convolution Operation in JAI and Java2D technology one can find a 'Threshold Point' at which JAI and Java2D execution Time is same. Below this threshold point as image size decreases Java2D behaves better than JAI and above this threshold point as image size increases JAI behaves better than Java2D.

REFERENCES

[1] Jonathan Knudsen (1999), JAVA 2D Graphics, O'Reilly Publication.

[2] Laurence H. Rodrigues, Building Imaging Applications with Java™ Technology.

[3] Sun's Java Web Page, java.sun.com

[4] http://micro.magnet.fsu.edu/primer/digitalimaging/index.html

Author Profile

Abhineet Sinha - Research interest is in the area of organization designs that maximize innovative patents. Under the Guidance of senior professor I wish to learn and analytically approach various research fields in depth . At DCE additionally, I also am a Resource Executive for the Society of Innovation Development.

***Vishal Kumar* – a diligent scholar and a hard working individual who continues his focused effort in order to achieve his goals. Working on this paper has been informative and revisiting and has created a curiosity about this field of research**

Mohit Panchal- An Associate researcher and scholar at our university. His contribution to this work is appreciable.