Computer Physics Communications 137 (2001) 300–311

Computer Physics Communications

www.elsevier.nl/locate/cpc

DataScan: An extensible program for image analysis in Java

K.A. Ritley [a,*], M. Schlestein [b], H. Dosch [a,b]

[a] *Max-Planck-Institut für Metallforschung, Heisenbergstrasse 1, D-70569 Stuttgart, Germany*
[b] *Institut für Theoretische und Angewandte Physik, Pfaffenwaldring 57, Universität Stuttgart, D-70550 Stuttgart, Germany*

Received 15 November 2000; accepted 3 January 2001

Abstract

The ability to analyze the topography of two-dimensional images is essential for many applications in the physical, chemical, and biological sciences. We describe a computer program to graphically display and perform numerical operations on large two-dimensional arrays of double-precision numbers. It contains an intuitive graphical user interface, and it includes a suite of useful image analysis and data reduction features. The program has been implemented in Java, a recent programming language whose "write once, run anywhere" philosophy enables graphical capabilities without modification on a wide variety of computers, operating systems and environments. DataScan has been optimized for ease of modification and extensibility, to enable straightforward customization to perform new tasks or include new analysis options. Diverse applications including X-ray and neutron diffraction, as well as microscopy are described. © 2001 Elsevier Science B.V. All rights reserved.

PACS: 02.30.Nw; 07.05-t; 07.05.Pj; 07.79.-v; 07.85.-m

Keywords: Image analysis; Java; STM; SPM; AFM; Fractal; Correlation; Fourier; GUI

PROGRAM SUMMARY

Title of program (32 characters maximum): DataScan

Catalogue identifier: ADOB

Program Summary URL: http://cpc.cs.qub.ac.uk/summaries/ADOB

Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland

Computer for which the program is designed and others on which it has been tested: Any computer which offers the Java Runtime Environment (JRE), version 1.3 or later

Operating systems or monitors under which the program has been tested: Windows95/98/NT, Tru64 Unix (v. 4.0F)

Programming language used: Java 2 (SDK 1.3)

Memory required to execute with typical data: Typically 2 Mb, plus approximately 8 bytes per pixel of loaded image data

Number of bits in a word: 8

Number of processors used: 1

Has the code been vectorized or parallelized? no

Number of bytes in distributed program, including test data, etc.: 1 365 834

Distribution format: zip file

CPC Program Library subprograms used: none

[*] Corresponding author. Present address: Hewlett-Packard Consulting, Posener Str. 1, D-71065 Sindelfingen, Germany.
E-mail address: kenneth_ritley@hp.com (K.A. Ritley).

Keywords (descriptive of problem and method of solution): Image analysis, linescans, convolution, filtering, scanning probe microscopy, atomic force microscopy, scanning tunneling microscopy, X-ray/neutron diffraction, graphical user interface, Java

Nature of physical problem
DataScan provides a graphical user interface for performing image analysis tasks and mathematical/logical operations on large (typically 500 × 500 or 1000 × 1000) arrays of double precision numbers. It includes basic image analysis tasks, sequential processing of hundreds images, plus suites of routines useful for data reduction in X-ray/neutron diffraction and scanning probe microscopy. New routines may be easily included by the user.

Method of solution
The internal program section which defines the graphical user interface (GUI) is clearly separated from program section which performs image analysis tasks, to simplify adding new computational extensions to the program. The use of the Java programming language ensures portability between all operating systems and computers which offer a Java Virtual Machine (JVM).

Restrictions on the complexity of the program
The number of images which can be loaded depends on available machine memory, typically 8 bytes per pixel.

Typical running time
Execution times vary upon desired image processing tasks and size of the image. A linescan requires less than 1 s for computation. Convolution of a 1000 × 1000 pixel image requires about 1 s on a typical PC with a 266 MHz Pentium-II processor.

Unusual features of the program
The program is self-contained and portable. The image analysis features are incorporated whenever possible in a library of independent, standalone subroutines, so they may be re-used in other programs and applications.

LONG WRITE-UP

1. Introduction

Image processing and analysis, or the mathematical and logical manipulation of large two-dimensional arrays of data, is a requirement with ever increasing importance for many areas in the physical sciences. For example, experimental techniques such as X-ray and neutron diffraction, formerly dependent on single-channel detectors yielding data in simple x-y plots (count rate vs. detector position, for example) are being increasingly supplanted with two-dimensional detection systems such as CCD detectors, in which the output is a real-time sequence of hundreds or thousands of two-dimensional data sets. These data are, quite literally, movies which describe a real-time process, such as the oxidation of metal surfaces [1]. In many new techniques such as scanning probe microscopy, two-dimensional data sets are obtained directly but the measured data are contaminated by various artifacts and noise, and image processing is in all cases essential before these data can be interpreted [2].

There are a number of commercial solutions to this problem. Software products such as MathCad, PV-WAVE, and IDL bundle a wealth of image processing techniques with convenient user interfaces, and these packages generally provide for user-written “macros”, or internal programs, to enable new or repetitive tasks [3]. There are significant drawbacks to using such software, however. The mathematical routines are generally “black-box” by design; that is, the source code is proprietary and not available for user inspection or modification. The programs are costly, and they may not be available on all desired platforms. To perform custom tasks, the user is forced to learn a specific macro language, which hinders recycling of numerical code and prevents easy leveraging of existing libraries of scientific analysis routines. Finally, depending on the program and the analysis needs of the user, important image tasks simply may not be possible.

To circumvent these difficulties in our own research, we have developed our own image analysis software (DataScan). The design criteria were these: (1) the program should be open-source software, so that the source code can be freely inspected and modified; (2) the program should be easy to modify, that is, the source code should clearly separate the language-specific graphical user interface (GUI) elements from the analysis routines, so that new analysis features are easy to incorporate; (3) the GUI should be powerful enough, with sufficient features, that users can focus on incorporating new data analysis features, rather than on technical details involving graphics/windows/mouse/etc.; (4) the program should be platform-independent, that is, it should produce the same results on all computers and operating systems. A final criterium included the incorporation of image analysis tasks relevant to our research program, which focuses on the reduction of two-dimensional data collected in X-ray and neutron diffraction experiments, and on image analysis for scanning probe microscopy.

This report describes the internal structure and features of DataScan. It is organized as follows. First, we briefly describe the "look-and-feel" of DataScan, and we provide a list of the image analysis capabilities it incorporates. We next discuss the internal structure of the program, in sufficient detail to enable a new user, knowledgeable in scientific programming practices, to modify the program. Finally, we briefly discuss our experiences with DataScan implemented on a variety of platforms, from personal computers to high-end workstations.

2. Features for image analysis

2.1. Overview

DataScan is a program to display and perform numerical analysis tasks on large two-dimensional arrays (typically 500×500 or 1000×1000) of double-precision numbers. These numbers may represent, for example, the output from a measuring device (such as a two-dimensional X-ray detector, scanning probe microscope, etc.), or they may be pixel intensities from digitally scanned images. Although the requirement for double-precision accuracy is infrequently realized in the raw data acquired from experiments, its use implies that subsequent mathematical operations can be performed on the dataset with the maximum numerical precision allowed by the Java programming language.

2.2. "Look and feel"

DataScan GUI is based on the so-called multiple document interface (MDI) model, in which there is a parent frame which contains menu options and toolbars; this parent frame is then populated with various "child" frames, such as images, graphs, or panels which contain image analysis options. A typical screenshot of DataScan is shown in Fig. 1. All of the image analysis and file handling features are accessible from a menu bar; additionally, the most-used features are accessible from buttons affixed to a movable toolbar.

The child frames may be selected with the mouse, moved, resized, iconized, and closed. There are five types of child frames:

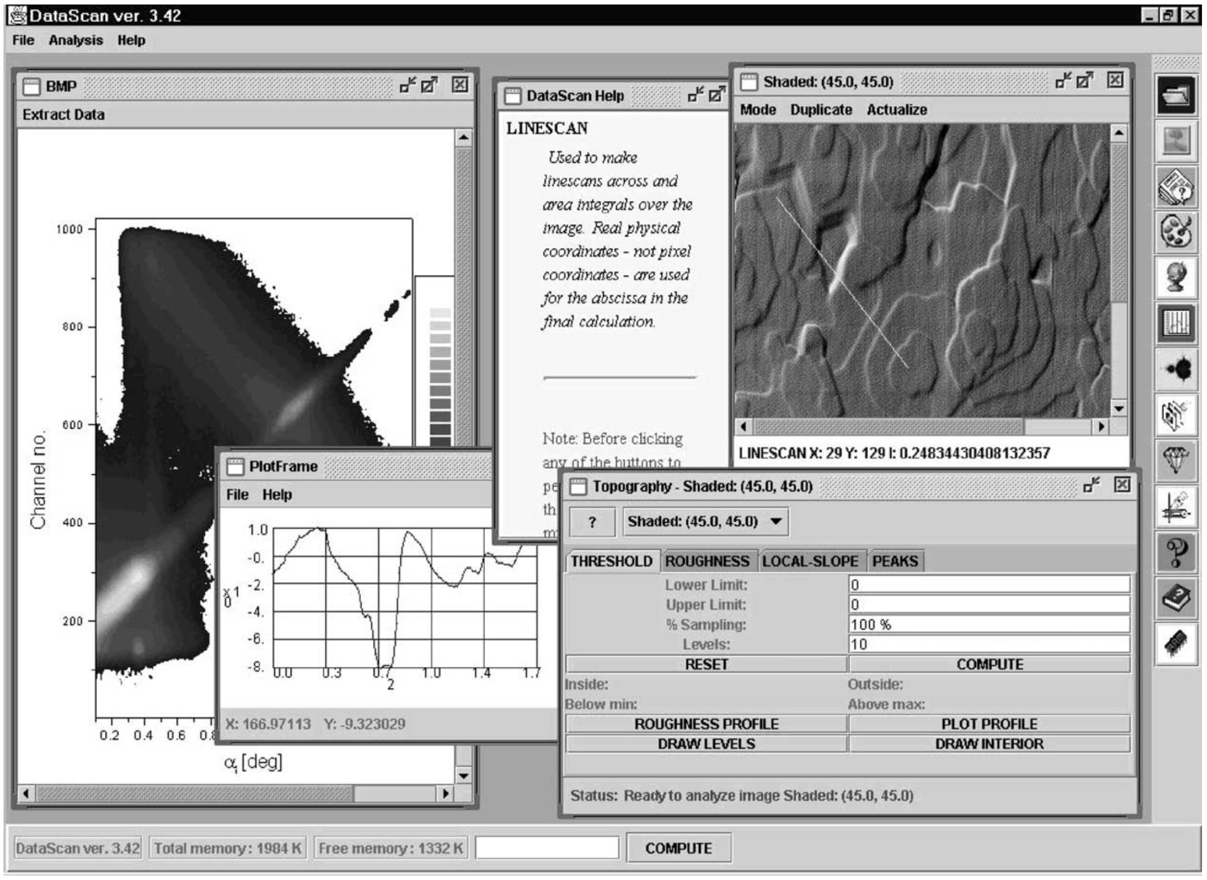
(1) *ImageDataFrame*, the principal frame which is used for displaying two-dimensional arrays of double-precision numbers. Although internally stored in double-precision format, the data arrays are graphically displayed as pixels with grayscale intensity values between 0 and 255; color "look-up tables" can be used to display the image in false color format, or to enable contrast adjustment. The mouse can be used to select regions-of-interest (such as linescans, area scans, point selection, etc.) as well as to display local image information (x and y pixel coordinates, and intensity). Scroll-bars appear when the image size in pixels exceeds the size of the child frame.

(2) *ImageFrame*, used for displaying simple color images obtained from binary data files in standard formats (GIF, JPG, BMP, etc.). Although the color data represented in an ImageFrame cannot be analyzed with the other DataScan analysis options, the ImageFrame contains menu items for conversion of the image to internal double-precision data arrays, for example, by separation of the color channels (RGB, CMY, etc.), by extraction of hue, saturation, and brightness levels, etc. These channels can then be displayed in ImageDataFrames.

(3) *PlotFrame*, used for displaying simple x-y plots. The mouse can be used to selected coordinates from the graph, and menu options are provided for saving the displayed data to ASCII files.

(4) *HelpFrame*, used for displaying context sensitive help information. The help files are written in hypertext markup language (HTML) format. The appropriate help files are displayed by clicking on a button located in each analysis frame (see below).

(5) *AnalysisFrames*, used for providing the user with a choice of image analysis features. There are over 10 separate analysis frames, each providing a set of related image analysis options, discussed below. The analysis options are applied to a user-selected dataset, which is mouse-selectable from box in the upper-left-hand corner of the AnalysisFrame



LINESCAN

Used to make linescans across and area integrals over the image. Real physical coordinates - not pixel coordinates - are used for the abscissa in the final calculation.

Note: Before clicking any of the buttons to

LINESCAN X: 29 Y: 129 I: 0.24834430408132357

X: 166.97113 Y: -9.323029

Lower Limit: 0
Upper Limit: 0
% Sampling: 100 %
Levels: 10

Status: Ready to analyze image Shaded: (45.0, 45.0)

DataScan ver. 3.42 | Total memory: 1984 K | Free memory: 1332 K

Fig. 1. A typical “screen shot” of the DataScan MDI, showing an example of each type of available frame. Clockwise from upper left: ImageFrame, HelpFrame, ImageDataFrame, AnalysisFrame, and PlotFrame.

(see Fig. 1). A help button displays a context-sensitive HelpFrame.

2.3. *List of image analysis options*

A complete list of the AnalysisFrames appears in Table 1, and we briefly comment on several of the most important analysis options herewith. Many of these features have widespread application to general image analysis problems (such as filtering and convolution), but a large number of features have been incorporated to assist in data analysis for X-ray/neutron diffraction experiments (such as batch processing and integration options) and for scanning probe microscopy (background subtraction and fractal surface analysis).

2.3.1. *EditFrame*

This frame contains a set of options to change the numerical values of the pixels in data array. Some basic operations (assigned to “buttons”) include the application of basic mathematical operations, such as absolute value, logarithm, deviation from the image mean, etc. Additionally, DataScan includes a recursive-descent mathematical equation parser [4], so that user-input strings (such as “$\sin(x)$” or “PI-$\tan(\log(x))$” can be applied to the elements of the array. Arithmetical operations between pairs of images are possible, such as addition or subtraction of two images.

DataScan incorporates various features for image background subtraction, which is of special impor-

Table 1
The classes which comprise DataScan, and the principal tasks they accomplish. Multiple instances of each class are allowed, but the Utils classes (UtilsFile, UtilsImage, UtilsParser) serve as libraries for static class methods and no instantiation is normally required

Class	Function
HelpFrame	display context-sensitive HTML help files
ImageFrame	display common image formats (GIF, JPG, etc.)
ImageDataFrame	display stored 2D double-precision arrays
PlotFrame	display (x, y) scientific graphs
DigitizationFrame	digitize points from displayed images
EditFrame	mathematical/logical manipulation of data arrays, background subtraction options
FileFrame	read and write various file formats
FilterFrame	perform filtering operations (discrete and Fourier space methods)
FractalFrame	computation of fractal properties of surfaces
InformationFrame	display stored dataset information, contrast adjustment
LinescanFrame	perform various linescan and sectioning operations
ProcessFrame	enable sequential batch-mode processing of large numbers of images
TopographyFrame	roughness characterization, thresholding operations
ImageClass	singleton class which makes image/dataset parameters available globally
UtilsFile	independent subroutines for reading and writing various file formats
UtilsImage	independent subroutines for mathematical manipulation of images
UtilsMisc	independent subroutines for assorted mathematical tasks (e.g., least squares fitting)
UtilsParser	a recursive-descent mathematical equation parser

tance for the analysis of scanning probe microscopy data [2]. In this experimental technique, a piezoelectrically-controlled probe is rastered line by line across a rectangular region of a surface. Interactions between the probe and the surface (e.g., electrical current, magnetic field gradients, or force) are measured to produce images of the surface. The measured data contain various artifacts which must be subtracted prior to data analysis: due to sample misalignment in the instrument, the data contain a linear background contribution; due to the line-by-line rastering, data points within a line may be smooth but successive scan lines may be linearly offset; and due to nonlinearities in the piezoelectric scanners, the data may be superimposed on a quadratic or higher-order polynomial background. The EditFrame includes features to perform these artifact removal tasks.

2.3.2. TopographyFrame

This frame contains features to perform topographical image analysis. These assume that each pixel in the two-dimensional dataset represents the height of a surface. Of special relevance for scanning probe microscopy applications, scalar quantities such as roughness (R_A, R_{RMS}, etc.) can be computed, and there are thresholding operations to compute the fraction of the surface area which falls between user-specified heights.

Several interesting histogram options are provided, including local-slope histograms [5]. As shown in Fig. 3, a local-slope histogram is a two-dimensional statistical distribution of facet angles on a surface; it is an invaluable tool in scanning probe microscopy applications for the characterization of facetted surfaces.

Extensive features for peak and valley determination are also included. The dataset (or mouse-

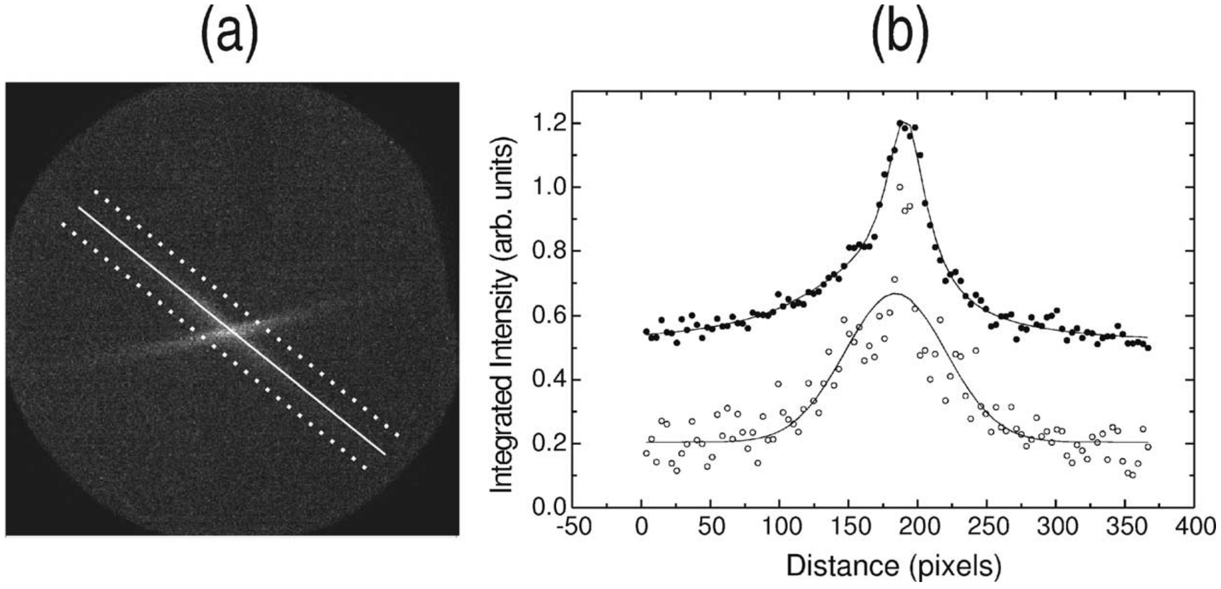


Fig. 2. A real-time X-ray diffraction experiment to investigate the real-time growth of a CoGa oxide was performed at Beamline ID32 at the European Synchrotron Radiation Facility [1]. A collimated beam of monoenergetic X-rays was diffracted from structural periodicities at the surface the sample, and the diffracted radiation was then detected using a two-dimensional detector at an approximate rate of one image per second during the oxidation process. (a) A typical image acquired during the experiment. Data analysis in these experiments involves careful integration of the specular and diffuse features, repeated for each sequence of hundreds of images, using linescan and integration options provided by the LinescanFrame and ProcessingFrame in DataScan. (b, open circles) A conventional linescan (solid line in (a)) misses a portion of the diffusively-scattered intensity; (b, solid circles) a linescan which integrates between dotted lines in (a), showing increased signal height above the background intensity. The solid curves in (b) are guides to the eye.

selectable subregions) can be searched for an arbitrary number of peaks and valleys. A peak/valley search may also be performed on the dataset row-by-row or column-by-column; this information can be used in combination with other image analysis tasks, as discussed in Section 2.3.8.

2.3.3. *DigitizationFrame*

This analysis frame is used with the mouse to select a list of points (x and y coordinates, and intensity) from a dataset. One important non-scientific application is the “stealing” of data from published scientific graphs: Scientific plots from journals and magazines may be scanned or otherwise obtained in digital format (GIF, JPG, etc.); these files are then read by DataScan, converted to a dataset, and coordinates of the plotted points can be selected and saved to a file. Additionally, this frame provides the capability for user calibration of the physical size of the image: the user selects points on the image, inputs the known separation of these points in physical coordinates, and the overall physical coordinates of the image are appropriately recalibrated.

2.3.4. *LinescanFrame*

The features in this analysis frame are used with the mouse to produce (x, y) graphs corresponding to linear slices through the data array. In general a user-selected line will not pass directly through the data points; a variety of interpolation features to handle this problem are possible. In addition to simple linescan features, two-dimensional “integrated” linescans are also possible. These are of importance to data analysis in X-ray and neutron diffraction experiments, as shown in Fig. 2. A two-dimensional detector collects the intensity of X-ray or neutron radiation diffracted from a sample. The “specular” intensity which appears along a linear portion of the detector derives from radiation which diffracts from perfectly-ideal periodicities within the sample; this must be compared with the

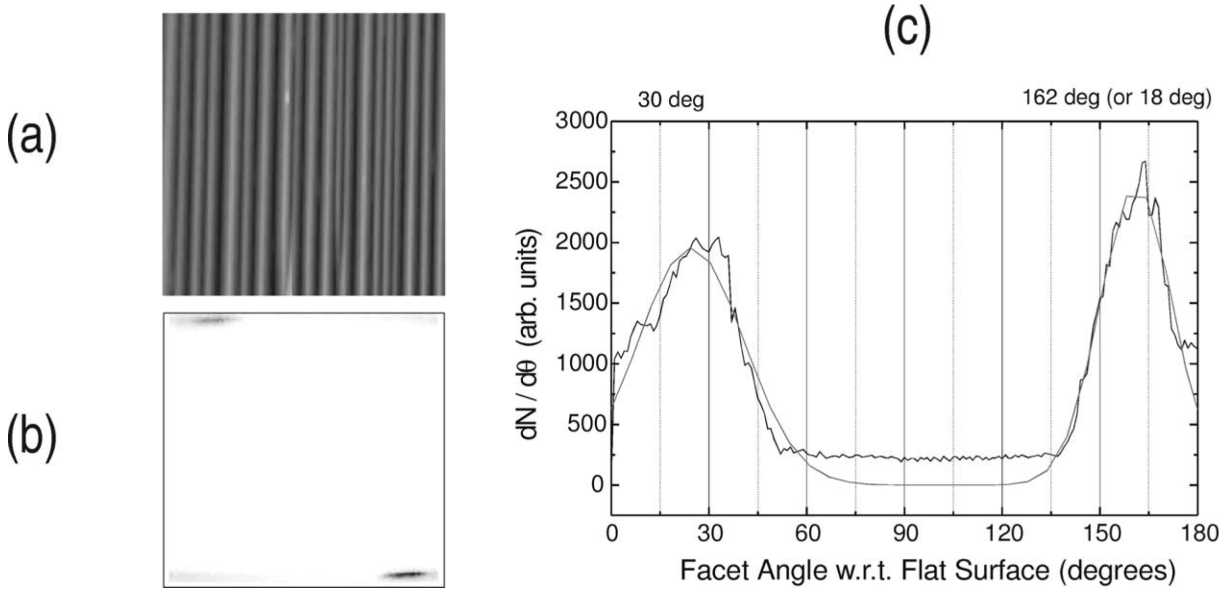


Fig. 3. (a) An atomic force microscopy image (1500 nm × 1500 nm) of a facetted sapphire surface. (b) A two-dimensional local slope histogram of this surface, where the upper right-hand coordinate represents (0°, 0°) and the lower-left-hand coordinate represents (180°, 180°). (c) The integration of the columns of the local slope histogram, in order to quantitatively show the distribution of facet angles on the surface. The fit is to show the deviation from a Gaussian distribution. It is apparent from this analysis that the facet surface with the shallowest angle (approximately 18°) has a higher probability of occurance, as expected from obvious geometrical arguments.

"diffuse" intensity in the region adjacent to the specular region, which derives from radiation scattered by roughness or inhomogeneities in the sample.

2.3.5. *ProcessingFrame*

This frame provides the capability to apply selected linescan and region-of-interest operations sequentially to a large number of datasets. The list of datasets to sequentially analyze is read from an ASCII file; in turn, each dataset contained in this list is opened and analyzed. In this way, a large collection of dozens or hundreds of datasets may be sequentially analyzed. This is of particular importance in recent X-ray and neutron scattering applications, in which hundreds or thousands of images are collected during the measurement of a real-time process, such as the oxidation of a sample [1] or during a structural phase transition [6]. Two types of ASCII datafiles are produced: *ROI datafiles*, in which each row in the file contains information (maximum and minimum intensity of the region, etc.) about a selected region-of-interest for each successive image in the series; and *linescan datafiles*, in which each column in the file contains a linescan (many types of linescans are possible) for each successive image in the series.

2.3.6. *FractalFrame*

This frame provides a several methods to calculate the fractal and self-similarity properties of surfaces. As discussed in Ref. [7], a precise mathematical definition of these terms is complex, but they can be qualitatively understood as the degree to which a surface "spreads out" into three dimensions. This analysis frame includes a calculation of the fractal (Hausdorf) dimension using Minkowski cover and RMS vs. area algorithms [7].

Additionally, several calculations of the height-height distribution functions are also included. Unlike scalar measurements of surface roughness (such as the RMS roughness and other quantities calculated by the TopographyFrame), these functions provide information about the length scales over which the surface roughness approaches its saturation value [8].

The radial height-difference g and height-height C distribution functions are defined as

$$g(R - R') = \langle (h(R) - h(R'))^2 \rangle \tag{1}$$

and

$$C(R - R') = \langle h(R)h(R') \rangle, \tag{2}$$

where $h(R)$, $h(R')$ denote the surface height (image intensity) at two-points on the surface R, R', and $\langle \cdots \rangle$ denotes an average over the entire surface. The calculation of these quantities are performed using a Monte Carlo algorithm, to permit user-control over the accuracy and computation time [9]. Additionally, a routine to compute the two-dimensional height–height correlation function f

$$f(x - x', y - y') = \langle h(x, y)h(x', y') \rangle \tag{3}$$

is also available [10]. This routine uses all data points in the image; for an $M \times N$ image, $M \times N \times (M - 1) \times (N - 1)$ evaluations are necessary and the calculation may require several minutes.

2.3.7. *FilterFrame*

This frame provides extensive capabilities for image filtering, convolution, and Fourier transformation. This section briefly lists the filtering capabilities of DataScan; detailed information about filtering can be found, for example, in Ref. [11].

Discrete filtering is a powerful image analysis technique based on discrete convolution of the image with a so-called filter kernel. The convolution h of two continuous functions f and k is defined by

$$h(x) = f(x) * k(x) = \int_{-\infty}^{+\infty} f(t)k(x - t)\,\mathrm{d}t, \tag{4}$$

where k is known as the filter kernel, and the integral need only be performed where $k(x - t)$ is non-zero. For a discrete two-dimensional function, such as an image, Eq. (4) can be extended to

$$H[x][y] = \sum_{j=0}^{\text{height}-1} \sum_{i=0}^{\text{width}-1} F[x + i][y + j]K[i][j]. \tag{5}$$

Depending on the choice of the filter kernel, extensive capabilities for manipulation of the image are possible. DataScan includes over 20 separate 3×3 and 5×5 kernels, including kernels for line detection (all directions), gradient detection (embossing), smoothing, blurring, as well as high-pass and low-pass filters. Soebel edge detection, based on successive application of horizontal and vertical edge detection kernels, is also included. Additionally, DataScan incorporates a convenient array-like GUI for kernel selection and modification, to make it easy for the user to inspect existing kernels or else construct new kernels.

DataScan also includes two-dimensional Fourier transform capabilities. A wide variety of image analysis techniques, including frequency-space filtering, autocorrelation, and shape determination are based Fourier transformation techniques. Given a discrete two-dimensional function f defined over the two-dimensional grid $0 \leqslant k_1 \leqslant N_1 - 1, 0 \leqslant k_2 \leqslant N_2$, the Fourier transform H of f can be written

$$\begin{aligned} H(n_1, n_2) = \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} & \exp(2\pi \mathrm{i} k_2 n_2 / N_2) \\ & \times \exp(2\pi \mathrm{i} k_1 n_1 / N_1) f(k_1, k_2). \end{aligned} \tag{6}$$

DataScan uses a Java modification of the well-established Danielson–Lanczos algorithm to efficiently compute the real and complex parts of the discrete Fourier transform of an image [12].

2.3.8. *Cross-frame capabilities*

Because the output of many of the image analysis frames are available globally within DataScan, more exotic image analysis tasks may frequently be accomplished by a employing a combination or succession of options from various analysis frames. We provide three simple examples. *Island size analysis*. Island size analysis is an important task in scanning probe microscopy applications; by using a combination of TopographyFrame, EditFrame and DigitizationFrame features, thresholding features may be used to cleanly isolate surface islands, which then may be quantitatively analyzed. *Peak tracking*. This example concerns the analysis of data collected by position-sensitive detectors (PSD) in X-ray diffraction experiments: in this case, each row in a two-dimensional image may contain the detection channels from a PSD, where successive rows are measured at different times [13]. The row-wise or column-wise peak finding capabilities provided by the TopographyFrame can be used to track the motion of diffraction peaks in the spatial- or time-coordinates, and the intensity information may subsequently be accessed, plotted or exported using features provided by the DigitizationFrame. *Local slope histograms*. The statistical analysis of surface facets is an important task for scanning probe microscopy experiments. A typical example appears in Fig. 3(a), which shows an atomic force microscopy (AFM) image of an epitaxially-polished $(10\overline{1}0)$ α-Al_2O_3 sapphire surface. This surface is unstable, and upon heating in air to over 1400°C, the surface spontaneously develops $\{10\overline{1}1\}$ and $\{10\overline{1}2\}$ facets. The statistical analysis of the facet angles can be accomplished by means of a local slope histogram, a feature implemented on the TopographyFrame which calculates a two-dimensional histogram of the local slopes across the surface (Fig. 3(b)). For quantitative analysis the local slopes in the separate x- and y-directions, it is useful to integrate the LSH image along the rows and columns (Fig. 3(c)), using features provided by the LinescanFrame.

3. Internal program structure

3.1. The Java programming language

The aforementioned design criteria were met in every case by using the programming language Java. Developed by the Sun Computer Corp. in 1995, Java is a modern, object-oriented programming language with a syntax nearly identical to that of C/C++. It contains extensive (literally, hundreds) of features for GUI development. Platform independence is achieved by program interpretation, rather than compilation. Java source code is compiled to produce files of "byte-codes", or machine-independent instructions for the Java Virtual Machine (JVM), a platform-dependent pseudo-compiler which translates the byte-codes to machine-language instructions during run-time. Since JVMs are standardized and available for a wide variety of computers and operating systems, the same source code can be guaranteed to run the same way under all JVMs, without regard to platform or environment.

An additional advantage of Java is the inclusion of tools (Javadoc) to automatically generate user-friendly, standardized, HTML-based internal program documentation. This is of enormous importance for users who wish to extend the capabilities of DataScan; in every case, these features have been employed to copiously document DataScan, to simplify the program modification process.

DataScan has been developed using the Forte-For-Java interactive development environment (IDE), written and distributed free-of-charge by the Sun Computer Corp. Its features greatly facilitate the construction of GUIs in Java. However, this IDE is not a requirement for modifying DataScan. The DataScan code can be modified with any text editor, and it can be compiled with any compiler that supports the Java 2 (SDK 1.3) specification.

3.2. Program structure

To facilitate rapid and easy modification of the program to accomplish new tasks, DataScan has been internally divided into an object-oriented GUI section, which handles non-scientific programming tasks, such image display and mouse handling (for linescans, sub-area selection, etc.); and a procedurally-oriented mathematical section, in which the image data is easily accessed and numerical operations can be performed. The following sections describe the internal program structure in greater detail.

3.2.1. GUI section

The user-interface section makes use of the extensive object-oriented capabilities of Java for GUI development. The Java Swing classes are used to construct the multiple document interface (MDI). The main program module implements the JDesktop class, and this can be populated by the five aforementioned types of classes, each of which extends the JInternalFrame class. These classes are labeled with the word Frame by convention: ImageDataFrame, ImageFrame, PlotFrame, HelpFrame, and AnalysisFrames.

The graphical display of the images is achieved within each ImageDataFrame by means of the BufferedImage class. While this entails increased memory cost (i.e., double precision arrays are used to store the datasets, while parallel to this a scaled (0–255) representation of the data is stored in the raster which belongs to each instance of a BufferedImage), the need for fast graphical updating capabilities requires the use of BufferedImages, for example, for quickly updating the image, selection of linescans and regions-of-interest with the mouse, etc.

3.2.2. Numerical analysis section

The numerically-intensive section is procedurally structured rather than object-oriented. Several classes (labeled with the word Utils by convention: UtilsImage, UtilsFile, UtilsMisc, UtilsParser) serve as libraries for independent subroutines (or static class methods) for performing specific image analysis tasks. Examples of these tasks include linear least squares fitting for background subtraction, fast Fourier transformation, discrete convolution with a variety of kernels, resizing the image via interpolation, etc. These routines are invoked from specific AnalysisFrames; additionally, they are well-documented and available for use in user extensions or modifications to DataScan (see Section 3.3).

3.2.3. Global variables

Java does not support global variables, which is somewhat contrary to the standard procedural paradigm (e.g., Fortran) generally employed for scientific programming. Although this offers advantages (such as "data-hiding" to prevent misuse of variables in complicated programs) it can lead to unnecessary complication, particularly for scientific applications in many variables must be shared widely throughout the program. DataScan implements an efficient, object-oriented solution to this problem, known as the Singleton model for global variable storage [14]. In this model, the variables which have global scope are stored as statically-declared class variables in a single class (ImageClass), which is accessible by all other classes within the program. Use of the "static" key-

```
1    private void testSubroutine() {
     // Access the global variables
2         ImageClass MyImage = new ImageClass();
     // Test that a dataset has been loaded
3         if (MyImage.getNumberOfImages==-1) return;
     // Get the size of the image in pixel coordinates
4         int iX = MyImage.getImageXSize(); int iY = MyImage.getImageYSize();
5         double[][] dA = new double[iX]][iY];
     // Get the size of the image in physical (real world) coordinates
6         double xSize=(MyImage1.getXStopCoord()-MyImage1.getXStartCoord());
     // Fetch a copy of the data array
7         MyImage1.getImage( dA );
     // Perform the image/dataset manipulations here . . .
7    // . . .
     // Now store the new dataset
8         if (MyImage.take(dA,iX,iY,"comment","Image Name")==-1) return;
     // Optional:  store physical information about the new dataset
9         MyImage.takePhysicalSize(200,300,500,600);
     // Display the new image from the dataset
10        MyImage.display(); // Displays the new image on the desktop
11   return; }
```

Fig. 4. An example of a typical user-written subroutine, in order to incorporate a new image analysis feature in DataScan. The specific instructions are discussed in the text.

word ensures that only one instance of each variable is created. This offers advantages for rapid expansion or modification of DataScan to accomplish new tasks.

The datasets are stored as two-dimensional arrays of double-precision numbers, which in Java corresponds to a 64-bit implementation of the IEEE-754 standard. These arrays are allocated dynamically as needed during execution, and the size of each array is individually specified, to conserve required memory. In contrast to programming languages such as C++, in which memory allocation/deallocation is controlled directly by the programmer, Java allocates storage space when needed and handles memory deallocation by "garbage collection", a process running in parallel to the program, at low-priority, which deallocates memory belonging to variables with no active references. DataScan provides options for the user to start a garbage collection thread, as well as to display available and occupied memory.

3.3. Extending/modifying DataScan

To incorporate additional calculations into DataScan, some knowledge of Java programming is obviously required. But because the image handling and GUI aspects of the program are well-separated from the numerical analysis section, knowledge of C/C++ syntax is usually sufficient. An example of a user-written subroutine appears in Fig. 4. The first instruction (line 2) declares and creates and instance of the global Singleton image class (ImageClass), which makes information about all the images available within the subroutine. The first method invocation (line 3) tests whether any images are currently loaded. If one or more images are loaded, then the last image which was "activated" (either by virtue of most recent construction or else by mouse-clicking on the image) is the default image, upon which all methods in the image class will operate. Subsequent lines (lines 4 and 5) show useful methods which return information about the image, as well as a method which returns the double precision data array itself (line 6). There are additional methods (not shown) which return information about linescans, regions-of-interest, or digitized points the user has selected on the image. The user is free to perform whatever image operations are necessary (line 8). Finally, the user may store the new dataset (line 9) and display the image on the desktop (line 12). Additional methods can be invoked, for example, to store parameters such as image size in physical units (line 10) or to specify the displayed interleave factor (line 11), useful for images otherwise too large to fit the display screen. A complete and well-documented list of publicly-declared static variables and methods is available within the on-line documentation.

4. Performance and summary

DataScan has been tested on a number of machines and operating systems, including PCs running the Microsoft Windows95/98/NT operating systems, and a Compac DS20E Alphaserver running True64 Unix (ver. 4.0F). It has also been run remotely across X-Windows-type interfaces with Linux systems and Sun Solaris-based workstations. Despite the Java Virtual Machine performing program interpretation rather than highly-optimized compilation, in all cases execution speed was more than adequate for performing the desired image processing tasks. Additionally, we verified the platform-independent "look-and-feel" promised by Java; although there were several small differences, such as the design of the window borders, in no cases was the GUI functionality of DataScan hindered or impaired.

DataScan has been used in collaboration with R. Streitel and A. Stierle to analyze X-ray diffraction data collected during a metal oxidation experiment at Beamline ID32 of the European Synchrotron Radiation Facility (ESRF) in Grenoble, France [1]. In this experiment, clean, oxide-free surfaces of a high-quality CoGa sample were allowed to oxidize under carefully controlled conditions, at temperatures up to 500°C. X-ray diffraction measurements were performed in situ during the oxidation process, in order to study the crystal structure of the growing oxide surface. Diffraction data were collected using a Siemens two-dimensional wire detector, which typically generated sequences of 100–200 images, each with 1024×1024 pixels, 16 bits per pixel, one image per second. The data analysis for this experiment involves using the DataScan ProcessFrame to perform an established set of topographical operations (linescans, region-of-interest analysis, peak position, etc.) in exactly the same way for each image. Using a PC with a Pentium II/400 processor with 512 Mb of memory and the WindowsNT (v. 4.0) operating system, a total time of less than two minutes is typically required for DataScan to read each of over 100 images, perform the necessary analysis, and write the results to an output file.

In summary, we have used Java to develop a platform-independent package (DataScan) with an intuitive GUI to display and perform numerical operations on large two-dimensional arrays of double-precision numbers. It includes a suite of analysis routines to perform common image analysis tasks, routines to perform specific tasks relevant to problems in X-ray/neutron diffraction and scanning probe microscopy, and routines to enable batch processing for automating the processing of large numbers of images. To simplify user modifications to DataScan, the internal program structure has been divided into a object-oriented section which implements the GUI, and a procedural-oriented section which implements the numerical calculations. DataScan has been tested and found to give good performance on a wide variety of computers and operating systems.

Acknowledgements

We thank B. Gliss for useful discussions regarding Java. We thank D.G. Cahill, N. Funk, M. Huth, J. O'Kane and D. Zhong for important contributions to the calculations, and V. Kruppa, W. Donner, V. Kruppa, A. Rühm, F. Schreiber, A. Stierle and R. Streitel for useful discussions.

References

[1] A. Stierle, R. Streitel, private communication.
[2] S. Magonov, M.-H. Whangbo, Surface Analysis with STM and AFM, VCH, Weinheim, 1996.
[3] PV-WAVE is a product from Visual Numerics, Inc., 1300 W. Sam Houston Pkwy S., Suite 150, Houston, Texas 77042, USA; MathCad is a product from MathSoft, Inc., 101 Main Street, Cambridge, MA 02142, USA; IDL is a product from Research Systems, Inc., 4990 Pearl East Circle, Boulder, CO 80301, USA.
[4] N. Funk, private communication.
[5] J.E. van Nostrand, D.G. Cahill, I. Petrov, J.E. Green, J. App. Phys. 83 (1998) 1096–1102.
[6] M. Linde, J. Trenkler, V. Bugaev, Y. Sikula, K. Du, F. Phillipp, H. Dosch, Science (submitted).
[7] J.C. Russ, Fractal Surfaces, Plenum, New York, 1994.
[8] H.-N. Yang, Y.-P. Zhao, A. Chan, T.-M. Lu, G.-C. Wang, Phys. Rev. B 56 (1997) 4224–4232.
[9] J.E. van Nostrand, D.G. Cahill, I. Petrov, J.E. Green, J. App. Phys. 83 (1998) 1096–1102.
[10] R.C. Munoz, G. Vidal, M. Mulsow, J.G. Lisoni, C. Arenas, A. Concha, F. Mora, R. Espejo, G. Kremer, L. Moraga, R. Esparza, P. Haberle, Phys. Rev. B 62 (2000) 4686–4697.
[11] J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes, Computer Graphics: Principles and Practice, Addison-Wesley, Reading, MA, 1990.
[12] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C, Cambridge Univ. Press, Cambridge, 1992.
[13] H. Dosch, Critical phenomena at surfaces and interfaces, Springer Tracts in Modern Phys. 126 (1992).
[14] E. Gamma, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1997.