REVIEW OF SCIENTIFIC INSTRUMENTS VOLUME 75, NUMBER 9 SEPTEMBER 2004

Digital imaging processing for biophysical applications

Jörg Schilling, Erich Sackmann, and Andreas R. Bausch[a]
Lehrstuhl für Biophysik E22, Technische Universität München, Garching, Germany

(Received 28 January 2004; accepted 17 May 2004; published 2 September 2004)

Many biological and biophysical experimental setups rely on digital imaging processing. The introduction of a new generation of digital cameras enables new experiments where time dependent processes can be detected with a high time resolution and high signal-to-noise ratio. However, there are no software tools available with which the full potential of the digital cameras can be explored. Although the data streams of up to 24 MB/s are readily processed by the available hardware, they present an immense challenge to the current software packages. We present a software concept based on the object oriented paradigm, with which digital cameras can be controlled and full images at full rate are captured, processed, and displayed simultaneously over extended time periods, just limited by the capacity of the hard disk space. By implementing wavelet based compression algorithms the obstacle of archiving the immense amount of data is overcome. We present examples in which original data files are compressed to 10% of its original size without loss of information. The modular character of the object based program enables the implementation of a wide range of different applications into the program. © *2004 American Institute of Physics.*
[DOI: 10.1063/1.1783598]

I. INTRODUCTION

Most biophysical experiments from the single molecule level to the collective phenomena rely on digital image processing of video sequences.[1–3] Despite the fact that in recent years immense progress in the design of cameras has been achieved, the full potential of the new generation of digital cameras is seldom used due to limitations of available software. However, in biology and biophysics a highly sensitive, fast and low noise level detection over long periods of time is a prerequisite for most experiments. Until now the standard approach for recording long image sequences is to use analog cameras, where the interlaced images are stored on video tapes with standard frequency. Time lapses are digitalized afterwards, where the total time of digitalization is limited by the available RAM. Thus for full frame digitalization of 720×480 pixels and 30 images/s only a sequence of 40 s can be obtained (assuming a typical RAM of 512 MB). A major disadvantage of using video tapes for storage is the limited quality of video-tape recordings. This limitation of image quality of video tapes and the limitation of the fixed image size, the fixed bit depth, and fixed frame rate of the analog interface can be overcome by using digital interfaces. Therefore, modern cameras are most times only equipped with a digital interface which cannot be used together with a video-tape recorder.

The limitations for storing video data from cameras with digital or analog interfaces directly to hard disk are due to the technical limitations of older computer standards where the transfer rates to hard disk were lower than 10 MB/s. New possibilities opened when the capacity and speed of hard disk increased considerably over the last years. Presently, data streams of 100 MB/s for writing data to hard disk are possible. Most available software packages do not use these capabilities. Once the data streams of digital cameras are usable the archiving and processing becomes an important issue. Already a 10 min experimental observation at typical frame rates of digital cameras produces 14 GB of data, which is not handable any more. In order to proper analyze and archive this amount of data, the introduction of compression algorithms is mandatory.

Image capturing and processing is a complex process which would result in unmanageable complex codes once written in the traditional procedural programming techniques. This general limitation of the procedural programming techniques was already recognized in the 1960's and therefore Dahl and Nygaard[4] introduced the object oriented programming technique. In the beginning this concept was confronted with problems of lacking ability for fast execution. However, now powerful development systems like VC++ (Microsoft) exist, which support object oriented design.

Here we demonstrate that the limitations for the application of digital cameras can be overcome by developing a software tool, which fully explores the hardware capabilities of current computer technique. The presented concept was realized in our lab as an application with the name "OpenBox." This software is based on the object oriented paradigm and thus allows the easy extension for future requirements. The now commercially available digital cameras produce data streams of about 24 MB/s (e.g., Orca ER; Hamamatsu, Herrsching, Germany). With the presented new software tool this data stream can be saved to hard disk continuously in order to process long time series with highest frame rates. At the same time the images are simultaneously displayed,

[a]Author to whom correspondence should be addressed; electronic mail: abausch@ph.tum.de

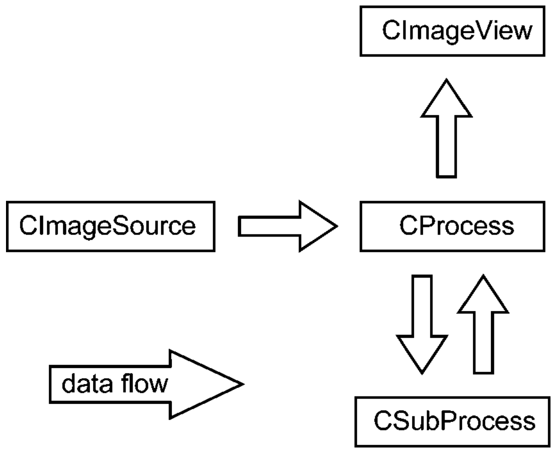

FIG. 1. Basic class diagram consists of CImageSource for data delivery, CImageView for image display, CSubprocess for processing images and the CProcess for the dynamic control. The arrows show the flow of image data between the classes.

which is mandatory for most experiments. For data analysis and manageable archival storage the huge amounts of data have to be compressed. This is solved by a wavelet based compression algorithm.

The object oriented software leads to a clearly arranged and expandable code, without concessions to execution speed. After introducing the assembly of the frame work, we illustrate the static and especially the dynamic interactions of the introduced classes. The resulting application is able to proceed with arbitrary processes like saving to hard disk or object tracking simultaneously and supports different sources like cameras and files. The object oriented approach makes it simple to expand this concept for new requirements.

II. METHODS

A. Object oriented design

The first step in programming an object oriented application is to identify and separate the problem into well defined tasks and relate them to distinct classes.[5] For the realization of the software tool for digital imaging processing we identified three major steps:

(i) Reading of images from different sources like camera or data files;
(ii) processing images; and
(iii) displaying of images.

For these steps three classes are defined: CImageSource for reading the images, CSubprocess for processing the images and CImageView for displaying them. For the dynamic organization of the interactions of these three classes a fourth class is necessary, which is called CProcess. These classes and the data stream between them are illustrated in Fig. 1.

The operating system Windows offers support for multi-document-interface-applications (MDI-Applications), which enables the simultaneous data handling and processing of multiple data sources at the same time in one instance of a program. The development system VC++ encapsulates this support into the Microsoft foundation classes (MFC) by offering defined classes.[6] For developing MDI Applications

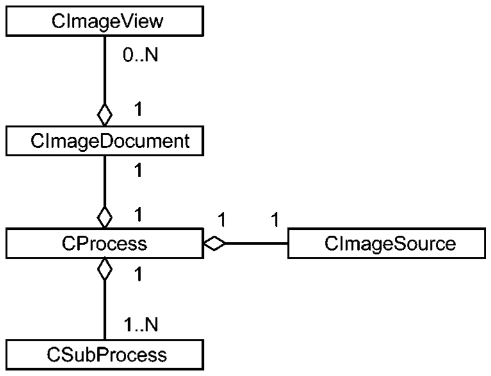

FIG. 2. Class diagram expanded to the necessities of the document/view model of the MFC. The new classes are implemented for easy access to the Win32-Api. The class CImageView is now derived from the MFC-class the Cview and a new class CImageDocument which is derived from the MFC-class CDocument is added. The relation between the different classes is illustrated. For each image source one object of the classes CImageSource, CImageDocument and Cprocess is created (1:1:1). One or more objects (N) for the display of the images (CImageView) or imaging processing (CSubProcess) can created. The open diamonds at the end of the lines show the “has-a” relation.

with the help of the MFC the two main classes CDocument and CView have to be implemented into the class scheme. In general, the class CDocument can be used to handle the data and offers the possibility of accessing, processing and saving different sets of data. Whereas, the class CView puts the data into graphs.

We implement this document/view model into the program. The class scheme of Fig. 1 has to be expanded by the classes CDocument and CView of the MFC (Fig. 2). The class CImageView is directly derived from the MFC-class CView which is also responsible for the user interaction. The interfacing with the MDI Application needs also the implementation of CDocument into the program. This is realized by introducing the additional class CImageDocument which is derived from CDocument. Handling and management of the data is thus achieved by the interplay of the three classes CImageDocument, CImageSource, and CProcess.

As these three classes define only one image source at a time only one object from each class is needed. In principal this would allow the implementation of these three classes as a single class. However, for reusability and clarity we decided to keep the three different tasks separate.

While CImageDocument is responsible for the communication with the Operating System, CProcess controls the dynamic interaction and CImageSource the data delivery.

The classes CImageSource and CSubProcess are virtual base classes. No objects are made and only the relation for the derived classes is defined. In Fig. 3 a part of the expanded class scheme is shown. A more detailed scheme of the frame work is presented elsewhere.[7] From the class CImageSource the classes CImageSourceAvi and CImageSourceTif are derived. These classes enable the access to Avi or TIF data files. The Class CImageSourceCamera allows the control of a camera From the class CSubProcessSaveAvi the class CSubProcessAvi is derived, which allows saving image stacks into files with Avi format. The derived class CSubProcessShowlmage is needed for

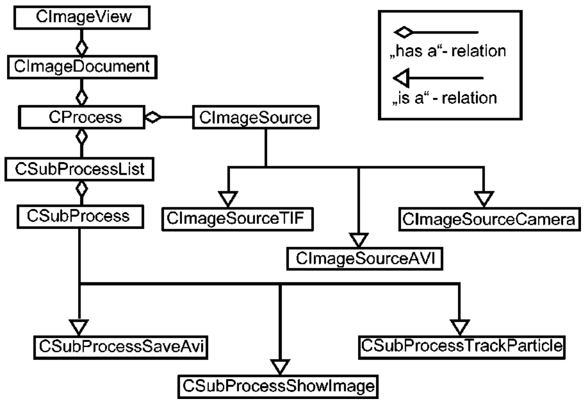

FIG. 3. Part of the framework build from the class diagram from Fig. 2. Three different image sources are derived from the virtual base class CImageSource and three different Subprocesses are derived from the virtual class CSubProcess. The arrows ("is a") and the diamonds ("has a") show the relation between the classes.

preparing the data for the display of the images. The class CSubProcessTrackParticle is an example of a derived class for the imaging processing where the position of single colloidal particles is determined.

The class CProcess organizes the interplay of the different subprocesses and thus needs to save the existence of subprocess into a list CSubProcessList, which is based on the template CTypedPtrList of the MFC. Until now only the static relation between the classes has been described. For the online imaging processing however, the dynamic organization is crucial.

B. Dynamical interaction of the classes

For the dynamic organization of the different classes it is important to distinguish two main kinds of image sources: real time sources like cameras and non-real time sources like data files. The non-real time sources, like data files, do not have any critical time steps and thus reading, processing, and displaying the images do not need any special coordination. This is different for the case that the images are read directly from the camera. Here, it is of outstanding importance, that no images are lost during the processing and displaying. This limits the time, in which images can be processed and displayed to the time difference between two consecutive frames. Consequently, the acquisition, processing, and displaying of the images must be executed with different priorities. As the processing time can differ from image to image, it is important that a dynamic compensation is implemented: if necessary the images are shifted into a buffer (see Fig. 5).

The different priorities are realized by introducing different threads, which are sorted according to their priorities. For the real time sources the first thread, capturing the images from the camera, has the highest priority. The second thread is the processing of the images, which is more important than the third thread, where the images are displayed. Capturing the images is absolutely time critical, as the images have to be read out directly after acquisition. The images do not have to be processed immediately after capturing, as the images can be shifted into the buffer. Displaying the images is obviously the least time critical thread. For

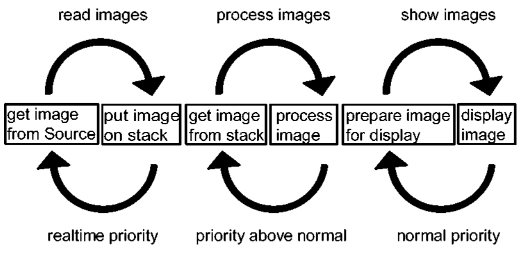

FIG. 4. Program flow consists of three threads. Here the tasks and priority are shown. Capturing the images from the real time source is time critical and has thus the highest priority. The processing of the image has a higher priority as the displaying of the image; both processes are executed in the free processor time between the acquisition of the images from the camera. The arrows indicate the circular program flow of the single threads.

example a typical image frequencies of 100 Hz are not resolvable by eye, and thus for visualization displaying every second image is most times sufficient. The dynamical process flow of the different threads with the priorities is illustrated in Fig. 4.

As a consequence of this prioritization of the different threads the image capturing thread builds a fixed framework to which all other threads have to subordinate. As schematically shown in Fig. 5, the quasiparallel execution of the different threads is achieved by adapting the execution of the non-time critical processes to this framework. Here, display-

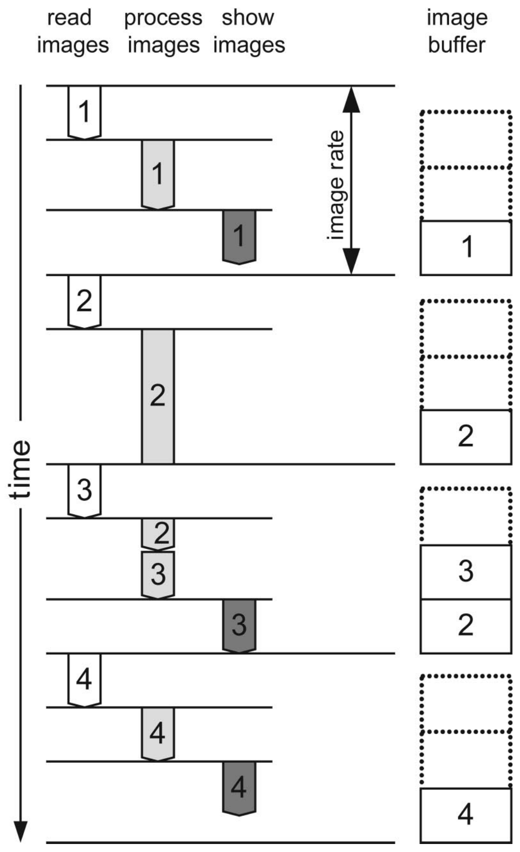

FIG. 5. Time scheme of quasiparallel execution of program. The bars show the calculation needed computer time for every thread. The label shows which image is processed. The rectangles on the right side represent the images in the image buffer. A scenario is shown, where the time needed for processing image number 2 is longer than the provided time. This is compensated by using the buffer and skipping the display of the image 2.

ing the images fills only into the remaining free time intervals.

It is important to note that the division of the program into threads is not in contradiction to the object oriented paradigm. This concept only defines with which priority the class coordinating class CProcess calls the methods of the classes CView, CImageSource, and CSubprocess.

C. Compression

Typical digital cameras with 1344×1024 pixels and frame rates of 117 ms produce a data stream of 24 MB/s which corresponds to 1.4 GB/min. Consequently, an experimental observation over 10 min using full frame and full frame rate results in a file of the size of 14 GB. For archiving this video sequences 20 CD or 3 DVD would be necessary. Clearly, this is not manageable with standard approaches and thus image compression has to be implemented in order to make digital cameras useable. For a compression to 10% of the original size, it would be possible to store image sequences of 32 min on a single DVD (assuming 24 MB/s).

Typical compression methods use the fact that a single image is represented by a 2*d* Matrix. The first step of compression is relying on a transformation of this matrix to a basis in which the data are represented. The basis is chosen to minimize the complexity of the matrix. In a second step the matrix elements are rounded and small values are set to 0. In the third and final step an entropy encoder is applied, in which similar coefficients are pooled. Most widely used entropy encoders are Hufmann or Runlength Coding.[8] In the consumer market the well known jpeg and mpeg1/2 compressions are used. These methods use in the first step a discrete cosine transformation (DCT). As the trigonometrical function is periodically this basis is well suited for periodical patterns but are failing for small irregular limited details. Consequently, transforming of an irregular image results in an even more complex matrix, which in turn can not be compressed further. However, by dividing the image into 8×8 blocks, irregularities are reduced and a meaningful transformation can be achieved. These blocks are than transformed one by one, which evidently results in block artefacts as boundary conditions are neglected. These artefacts are visualized in Fig. 6, where an image from reflection interference contrast microscopy (RICM) of a deflated vesicle lying on a substrate is shown. In this image the height information of the membrane is shown as interference pattern. This image has been compressed to 10% of its original size by the jpeg compression method. Although the artefacts are already visible in the real space image, the drastic loss of information is more evident once a derivative of the image is made. It is evident, that for contour analysis or particle tracking methods this image is of no use anymore. However, for consumer purposes where mostly time sequences are visualized, the artefacts are small for the eye and are often not noticeable.

The limitations of the trigonometrical basis can be overcome by introducing wavelet functions as basis. These functions are per definition suited to transform objects which are localized as well as scalable. Thus this basis allows the transformation of a complete image without dividing into blocks. Also in the consumer market the need for higher quality compression has been realized and a wavelet based compression algorithm has been introduced by the jpeg2000 format.[9]

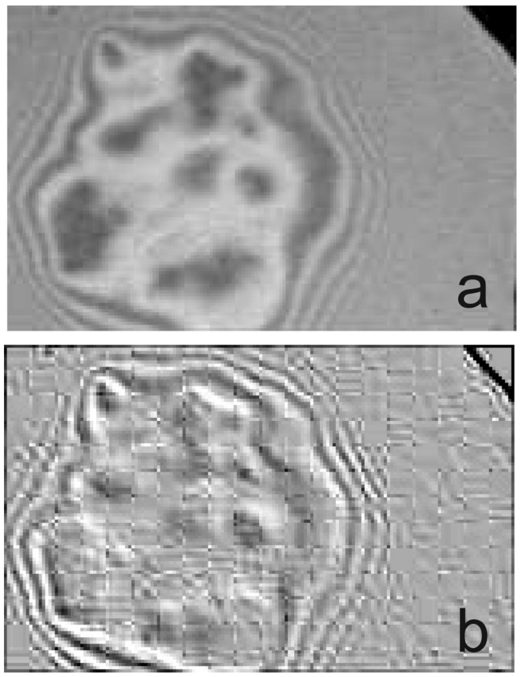

FIG. 6. (a) Interference pattern of a RICM-Image of a vesicle. The image has been compressed to 10% of his original with jpeg-compression. The block artefacts due to blockwise compression are weakly recognizable; (b) deviation of image shown in (a) makes the block artefacts clearly visible.

Here we use the wavelet compression library from Mevis (Bremen, Germany) which supports 16-bit greyscale images. This library allows also the compression of 24- and 48-bit RGB images.

Two distinct principals of compression can be applied: either every single image is compressed or only the changes between the images are considered. The latter would result in a higher compression rate maintaining the same image quality. However, the standard image to image compression algorithms, which are optimized onto the physiology of the human eye, result in smearing out of moving objects. As for most biophysical experiments the dynamics of the objects are of interest we decided to apply the single image compression method. To study and quantify the effect of compression onto the image analysis we present two typical applications: single particle tracking algorithm and the reconstruction of microinterferometric data.

D. Effects of lossy compression on quantitative results

The library of Mevis gives the opportunity for lossless compression of video microscopy images with a pixel depth of 12 bit at rates smaller than 1:3. This is not sufficient to solve the problem of archiving. Higher compression rates act as a low pass filter which removes the high frequency noise from the images. At very high compression rates the edge blurring effects are observed. For biophysical applications it is important to confirm that the quantitative data analysis of the video sequences is not affected from the compression.

Thus the optimal compression rate depends on the respective analysis.

The experimental studies have been done using a Computer with a 700 MHZ Pentium III processor and 256 MB RAM. The camera used is the Hamamatsu C4880–80 (Hamamatsu, Herrsching, Germany).

In a first series of experiments the effect of the compression rate on the analysis on periodic movements of single particles is determined. These movements are typical for the magnetic tweezers microrheology experiments.[10] Beads with a diameter of 4.5 μm are observed with phase contrast microscopy. With the typical optical magnification and the camera used the scale was 123 nm/pixel. The sinusoidal oscillation was obtained by applying an oscillating magnetic field. The single particle algorithm used, fits a 2D Gaussfunction to the beads' image, with which typically an accuracy of a few nanometers can be achieved.[11] The traces obtained from the original and the compressed data files are compared in terms of absolute bead position, amplitude and phase of the oscillating particle. The results are shown in Fig. 7. As expected the error is increasing with higher compression rates, reaching an error in the determination of the absolute position of a single particle of 50 nm at a compression rate which reduces the amount of data to 0.5%. However, the determination of the absolute position can be achieved with an error of 2 nm at a compression rate of 1:10. This is still in the range of the achievable total experimental accuracy. For the relative error in the determination of the phase and the amplitude an error of 0.02% was determined for this compression rate.

In a second series of experiments we studied the effect of compression on the reconstruction of the contour of a vesicle in contact with a passivated glass substrate from microinterferometric data. The height fluctuation of the contact area can be determined by the Reflection Interference Contrast Microscopy (RICM).[12] With this technique the height profile of a vesicle above a glass surface is imaged as an interference pattern. From the intensity at any position of the image the height above the substrate can be calculated. To determine the effect of compression on the calculated height, we analyzed video sequences of 2000 images at a frame rate of 83 Hz. The intensity of a single pixel of the fluctuating contact zone is determined and transformed into a height. The height of single points of the contact zone is fluctuating between 10 and 110 nm. Compression rates of 1:2–1:200 are applied. Figure 8 shows the different resulting images. It becomes evident that from a compression rate of 1:50 severe blurring effects are observable. The resulting relative error of the determined height is shown in Fig. 9. At a compression rate of 1:10 a relative error of 2% has been determined, which is in the range of the resolution limit of the experimental setup. We were able to show that for single particle tracking and in the reconstruction of microinterferometric images a compression rate of 1:10 results in relative errors which are comparable or below the experimental accuracy. It is important to note that this finding can not be generalized for all applications. Even for single particle tracking, the optimal compression rate depends evidently on the contrast and the number of pixels per particle. However, most applications tested in our lab confirmed that a compression of 1:10 is yielding consistent results.

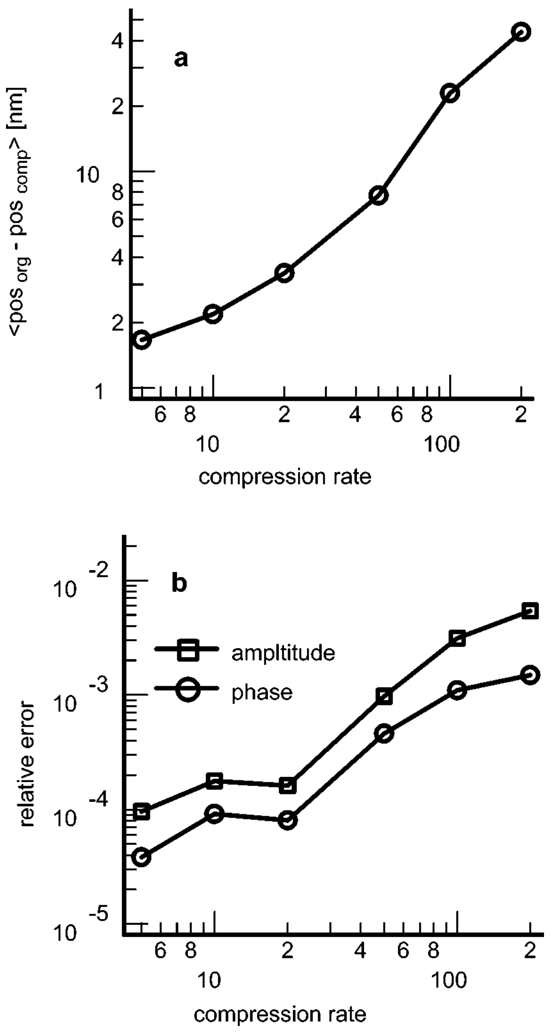

FIG. 7. Results of a magnetic tweezers experiment, where a 4.5 μm paramagnetic oscillating bead is observed. The data were obtained from images on which a compression with different compression rates were performed. The image sequence consist 1238 images with a size of 164×123 pixel. The frame rate was 100 Hz, the oscillation was excited with a sinus of 4 Hz. The deviation was calculated for compression rates from 1:5 to 1:200. Image (a) shows the average deviation of the position between the compressed and uncompressed images. Image (b) shows the relative error in the determination of the amplitude and the phase of the sinusoidal motion.

The object oriented approach enables the extension of the program to many different experimental setups. For example we realized an automated magnetic tweezers setup, which enables us to determine frequency dependent viscoelastic moduli over a wide range of frequencies. Only the implementation of digital cameras enabled us to extend the frequency range up to 40 Hz.[10] The control of the magnetic coils is achieved by the computer in such a way, that fully automated series of experiments, where the particle positions are determined online, can be achieved. One main advantage is that this approach enables us to study the temporal changes of biological systems. In a further application we have implemented a new kind of tracking algorithm for RICM patterns of beads into our software. With this new algorithm and an enhanced microinterferometric microscopy technique we enabled us the measuring of absolute interfa-

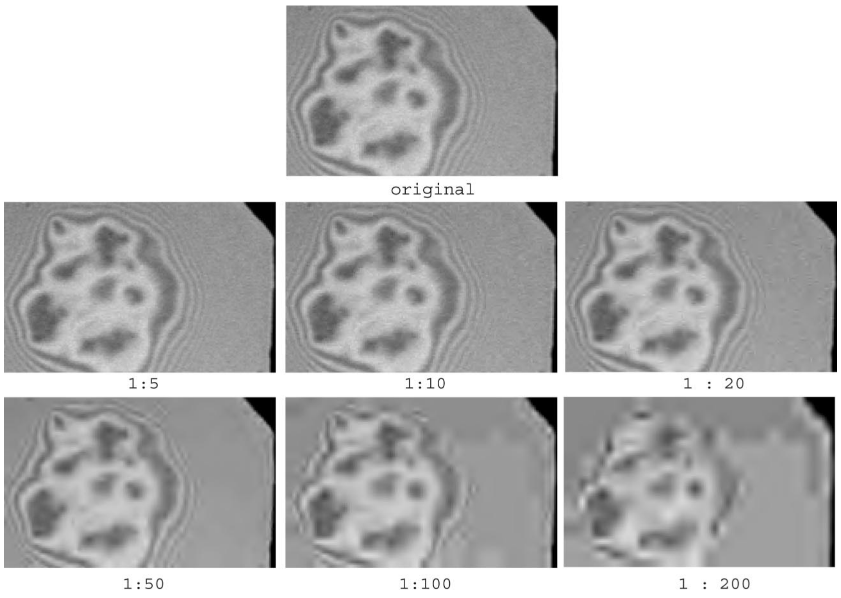

FIG. 8. Wavelet based compression performed to a RICM Image of a vesicle. The compression acts like low-pass-filter which removes first the noise from the image and effects at very high compression rates a smearing out of edges. Note that unlike the jpg compression no block artefacts occur.

cial distances in the micrometer range.[13] The modular extension of the software enabled to study different aspects and systems ranging from polymer networks to living cells.[14–18] With the most recent computer setup[19] the acquisition and storage of data streams of 24 MB/s onto hard disk results in a CPU load of under 20%, which shows that there are still many possibilities for the implementation of online imaging processing procedures. This real time ability is relying on the implemented prioritization of the different threads: an image is directly read out from the grabber card after the image has been transported from the camera to the frame grabber card. If this action would be delayed, the image in the grabber card would be overwritten, which has to be impeded. We showed that with the use of the IC-PCI-Card (Stemmer, München, Germany) and a System with 700 MHz Pentium III CPU the response time of the system is less than 0.5 ms, which was measured during an acquisition with 127.7 Hz. The images were permanently shown and the system was simultaneously loaded by a Bead Tracking Algorithm.

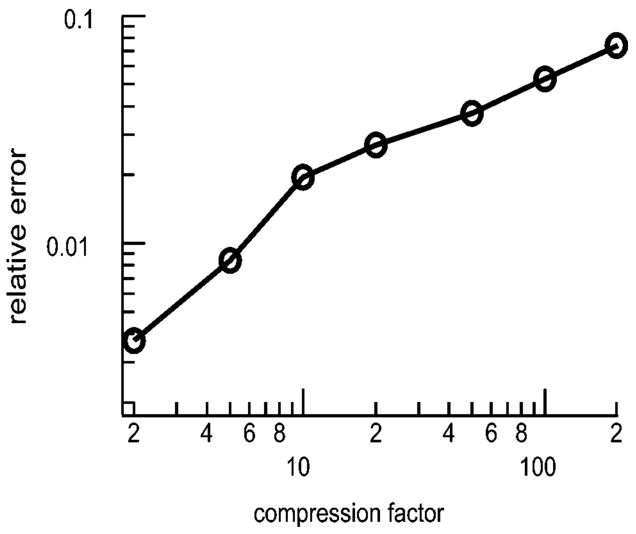

FIG. 9. Effect of the compression rate of the height calculation from the interference pattern of the RICM-Image of Fig. 4. The height was calculated from a single pixel. The average relative error of a sequence of 2000 images in dependence of the compression factor is shown.

ACKNOWLEDGMENTS

The authors are most grateful to the company Mevis for providing the MT-Wice Library. The support of the SFB 563 B14 and of the "Fonds der Chemie" is gratefully acknowledged. We also thank Rudi Merkel and Norbert Kirchgeβner for carefully reading the manuscript.

[1] B. P. Helmke, A. B. Rosen, and P. F. Davies, Biophys. J. **84**, 2691 (2003).
[2] M. P. Koonce, J. Köhler, R. Neujahr, J. Schwartz, I. Tikhonenko, and G. Gerisch, EMBO J. **18**, 6786 (1999).
[3] K. Kawakami, H. Tatsumi, and M. Sokabe, J. Cell. Sci. **114**, 3125 (2001).
[4] O.-J. Dahl, B. Myhrhaug, and K. Nygaard, Technical Report No. 22, Norwegian Computer Centre, Oslo, Norwegen (1970).
[5] G. Booch, *Object-Oriented Analysis and Design* (Benjamin/Cummings, Menlo Park, CA, 1994).
[6] D. Kruglinsky, *Inside Visual C++*, 3rd ed. (Microsoft Redmond, WA, 1996).
[7] J. Schilling, Dissertation thesis, Technische Universität München, 2003.
[8] G. K. Wallace, Commun. ACM **14**, 31 (1991).
[9] D. S. Taubman and M. W. Marcellin, *Jpeg 2000: Image Compression Fundamentals, Standards, and Practice* (Kluwer International, Dordrecht, 2001).
[10] M. Keller, J. Schilling, and E. Sackmann, Rev. Sci. Instrum. **9**, 3626 (2001).
[11] M. K. Cheezum, W. F. Walker, and W. H. Guilford, Biophys. J. **81**, 2378 (2001).
[12] S. Marx, J. Schilling, E. Sackmann, and R. Bruinsma, Phys. Rev. Lett. **88**, 138102 (2002).
[13] J. Schilling, K. Sengupta, S. Gönnenwein, A. R. Bausch, and E. Sackmann, Phys. Rev. E **69**, 021901 (2003).
[14] A. R. Bausch and D. A. Weitz, J. Nanopart. Res. **4**, 477 (2002).
[15] L. Limozin and E. Sackmann, Phys. Rev. Lett. **89**, 168103 (2002).
[16] L. Vonna, A. Wiedemann, M. Aepfelbacher, and E. Sackmann, J. Cell. Sci. **116**, 785 (2002).
[17] S. Goennenwein, M. Tanaka, B. Hu, L. Moroder, and E. Sackmann, Biophys. J. **85**, 846 (2003).
[18] W. Feneberg, M. Aepfelbacher, and E. Sackmann, Biophys. J. **87**, 1338 (2004).
[19] Configuration, Camera C4742-95-12ER (Hamamatsu, Herrsching, Germany), PC-Dig-frame grabber card (Stemmer, München, Germany), 2 GHz Intel Pentium IV-System with two additional hard disks for data acquisition in raid 0 mode (specially assembled for our requirements by Com:3000, München, Germany).