Journal of
Synchrotron
Radiation

ISSN 1600-5775

Received 26 September 2013
Accepted 5 January 2014

Image stack alignment in full-field X-ray absorption spectroscopy using SIFT_PyOCL

Pierre Paleo, Emeline Pouyet and Jérôme Kieffer*

European Synchrotron Radiation Facility, Grenoble, France. *E-mail: jerome.kieffer@esrf.fr

Full-field X-ray absorption spectroscopy experiments allow the acquisition of millions of spectra within minutes. However, the construction of the hyperspectral image requires an image alignment procedure with sub-pixel precision. While the image correlation algorithm has originally been used for image re-alignment using translations, the Scale Invariant Feature Transform (SIFT) algorithm (which is by design robust *versus* rotation, illumination change, translation and scaling) presents an additional advantage: the alignment can be limited to a region of interest of any arbitrary shape. In this context, a Python module, named SIFT_PyOCL, has been developed. It implements a parallel version of the SIFT algorithm in OpenCL, providing high-speed image registration and alignment both on processors and graphics cards. The performance of the algorithm allows online processing of large datasets.

Keywords: hyperspectral imaging; image alignment; full-field; XAS; SIFT OpenCL; GPU.

1. Introduction

Image alignment is required by many synchrotron beamlines and for various techniques such as speckle image reconstruction in the field of coherent X-ray diffraction imaging (CXDI), with module-based pixel detectors or image stack alignment for full-field X-ray absorption spectroscopy (FFXAS), or simply for re-positioning a sample on the experimental stage using visible light.

After a short presentation of the FFXAS experimental set-up (based on the design of ESRF-ID21), image registration algorithms (based on keypoint extraction) will be compared with the correlation algorithms through application to an example using FFXAS data collected from historical paintings.

Finally, SIFT_PyOCL, the parallel version of the SIFT algorithm we developed, will be presented. In the same section, some implementation details will be described along with a short tutorial and some benchmark figures in the scope of the data pre-processing of a FFXAS experiment.

2. Full-field X-ray absorption spectroscopy

At the European Synchrotron Radiation Facility (ESRF), beamline ID21 recently developed a full-field method for X-ray absorption near-edge spectroscopy (XANES) (De Andrade *et al.*, 2011; Fayard *et al.*, 2013). In this experiment (Fig. 1), for each energy point across a given absorption edge, a couple of magnified two-dimensional transmission images, with and without the sample, are acquired by a camera coupled with an X-ray scintillator and optics magnifying visible light. For each sample transmission image, a 'flat-field' image, recorded without the sample, is used for normalization of the spatial variation of the beam intensity. This flat-field image needs to be acquired at the same energy (to cope with the scintillator response), before and after the sample exposure. Since the sample moves in and out of the beam for every energy step, a realignment procedure has to be carried out for every frame.

A 3D-XANES stack consists of a series of normalized images (radiographs) that characterize the sample absorption across the absorption edge of the element of interest. After taking the negative logarithm of all transmitted intensities, each pixel of the stack contains a full XANES spectrum.

The energy resolution of the XANES spectrum depends on the number of frames in the stack and is therefore only limited by the energy resolution of the monochromator.

3. Image alignment algorithm

3.1. The limits of phase correlation

Image and phase correlation algorithms, obtained in Fourier space [see equations (1) and (2)], have extensively been used during the development of FFXAS for image stack alignment (the latter being an optimization of the former). However, they can only measure translation movement and turn out to be very sensitive to artifacts, such as intensity differences on the image border, defects in the scintillator or on the camera[1] *etc.*

Offsets (translations) are obtained by the image and phase correlations algorithms

$$\text{offset}_{\text{img cor}} = \text{argmax}_{x,y}\left\{\mathcal{F}^{-1}[\mathcal{F}(\text{img1}) \times \mathcal{F}(\text{img2})^*]\right\}, \quad (1)$$

$$\text{offset}_{\text{phase cor}} = \text{argmax}_{x,y}\left\{\mathcal{F}^{-1}\left[\frac{\mathcal{F}(\text{img1})}{|\mathcal{F}(\text{img1})|} \times \frac{\mathcal{F}(\text{img2})^*}{|\mathcal{F}(\text{img2})|}\right]\right\}, \quad (2)$$

where $\mathcal{F}$ is the two-dimensional Fourier transform and $\times$ denotes the element-wise product.

[1] While those artifacts are weakened by the flat-field normalization, they can still remain and disturb the alignment.

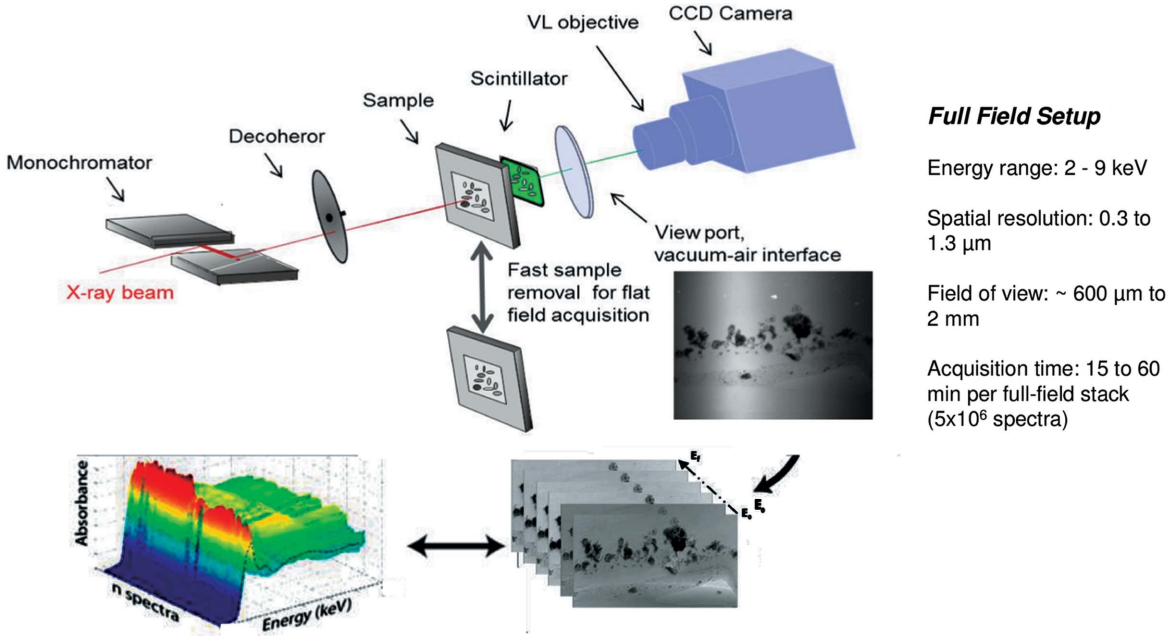

Figure 1
Experimental set-up of a full-field X-ray absorption spectroscopy experiment at ID21, as taken from Fayard *et al.* (2013).

These defects could be corrected by some sample-specific pre-processing such as border cropping and apodization. However, in order to make this procedure automatic and suppress human intervention, image alignment based on keypoints extraction has been considered.

3.2. Feature-based image registration algorithm

Feature-based registration methods establish a correspondence between a number of distinctive points in images. These keypoints are not only defined by their spatial position on the image (x and y) but are also associated with the *scale* of the feature and its orientation (or *angle*), making keypoints naturally robust *versus* translation, rotation and change of scale. In addition, each keypoint is associated with a descriptor specific to the neighbourhood of the keypoint which is used during the matching procedure. Position, scale and orientation can be further used for filtering-out outliers, for example by using a RANSAC-like algorithm (Moisan *et al.*, 2012).

The SIFT algorithm (Lowe, 1999, 2004), widely used for panoramic image stitching, has been adapted to FFXAS from the IPOL implementation (Yu & Morel, 2011). Another registration algorithm, SURF [for speeded up robust feature (Bay *et al.*, 2008; Oyallon & Rabin, 2013)], has been evaluated; it produces fewer keypoints with a keypoint descriptor twice as small as that generated by SIFT (64 bytes instead of 128). While being faster than SIFT, this algorithm was not retained due to some coarse approximation in the blurring procedure (box-filtering) and its inadequate descriptor size, making matching less reliable.

Initially, the image offset between a given frame and a reference frame (*i.e.* the translation to be applied for shifting the frame) was obtained from the median value of the difference in keypoint positions for all matched keypoints. The median value turned out to be more reliable than the mean value when false matches are present. Knowing the correspondence between a set of keypoints in images, more sophisticated transformation patterns like rotation, affine or projective transformations can be envisaged after a least-squares fit of the transformation parameters on the matched keypoints. Stack alignment of FFXAS data obtained by SIFT keypoints extraction and match using a simple translation is comparable with image correlation in terms of quality while exhibiting very good robustness against artifacts.

Moreover, the possibility to easily remove keypoints in unwanted regions of the image is a major asset. Regions containing non-characteristic features of the sample, *e.g.* scintillator-specific, may be removed. Alternatively, keypoints located within the region of interest (with an arbitrary shape) may be selected.

This possibility is illustrated in Fig. 2 which shows a typical frame extracted from an XANES stack. The image represents the normalized X-ray transmission of the sample at 2.46 keV. The image stack, composed of 346 frames, was collected at the K-edge of sulfur (2.45–2.66 keV) from a resin-based thin section (15 µm) of Matisse's *Flowers in a Pitcher* (1906) fragment, from the Barnes foundation (accession number 205), and analyzed in the context of investigations on the yellow cadmium pigment degradation in historical paintings.

In Fig. 2, the brighter top right corner corresponds to a place where no resin is present. Very few scintillator artifacts remain visible after flat-field normalization. Each arrow on the figure represents a keypoint, as extracted by the SIFT algorithm. Red arrows are inside the region of interest (*i.e.* on the sample) while the blue ones are outside it and correspond either to features of the resin (matrix) or features present on the scintillator (which absolutely need to be removed as they do not move with the sample). The size of the arrow is proportional to the scale of the feature detected and its direction depends on its orientation, as given by the SIFT algorithm.

The XANES stack alignment using image correlation has failed for this sample, possibly due to the strong change in intensity on the resin borders, forcing an alignment on the related line. Evidence of the poor alignment (on the pixel scale) is given in Fig. 3 which is a vertical section of the 3D XANES map; the sample is cut vertically at $x = 500$ and for $y \in [490, 580]$ (picture coordinates of Fig. 2), and the frame number[2] is used as the horizontal axis. To check the alignment, the

[2] The horizontal axis is not given as energy because energy steps were not constant in this scan.

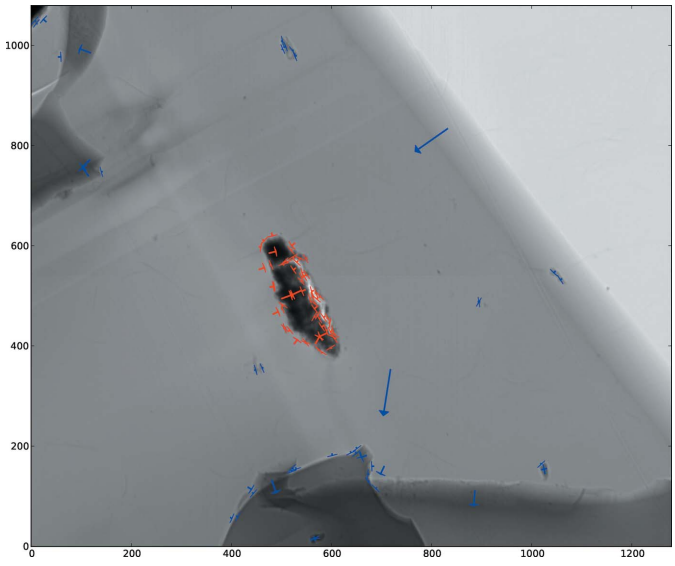

Figure 2
X-ray radiograph of a thin section of the painting taken at ID21 at the sulfur *K*-edge (2.46 keV). Arrows represent the position of SIFT keypoints, those corresponding to the sample in red and those from artifacts and from the matrix in blue.

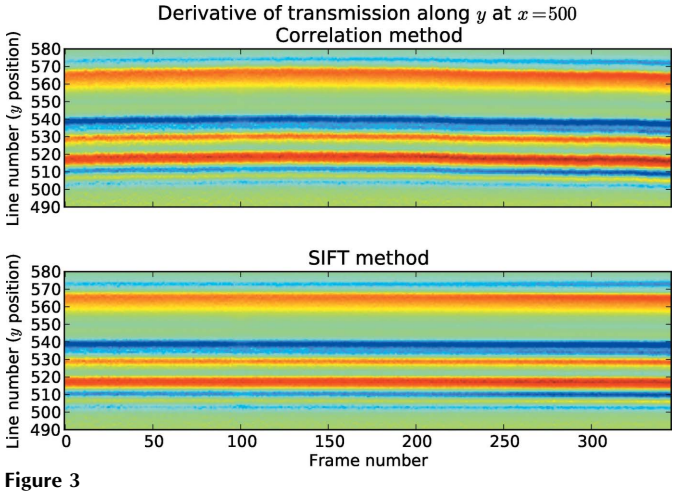

Figure 3
Comparison of the alignment quality obtained from the correlation method (upper frame) with the SIFT method (lower frame). All frames are aligned on frame 11. The figure is a vertical slice through the sample in Fig. 2 at x = 500 through all frames. To highlight the sample boundaries, the y-derivative of the transmission function is represented in false colours.

transmission signal has been derived along the vertical axis to highlight the sample boundaries. When the alignment is perfect, maxima (red lines) and minima (blue lines) of the derivative become horizontal lines. In this example, it should be noted that the stack aligned using image correlation presents clear bent sections. Fig. 4 shows the measured offsets with both methods. Offsets in the horizontal direction (δx) agree within an error lower than 0.5 pixels which is acceptable for experimental purposes, but offsets in the vertical direction (δy) differ sometimes by up to two pixels. While showing the same trend, the correlation algorithm apparently underestimates the displacement.

3.3. SIFT algorithm overview

The keypoints are detected in several steps according to Lowe's original paper (Lowe, 1999). The scale variation is simulated by

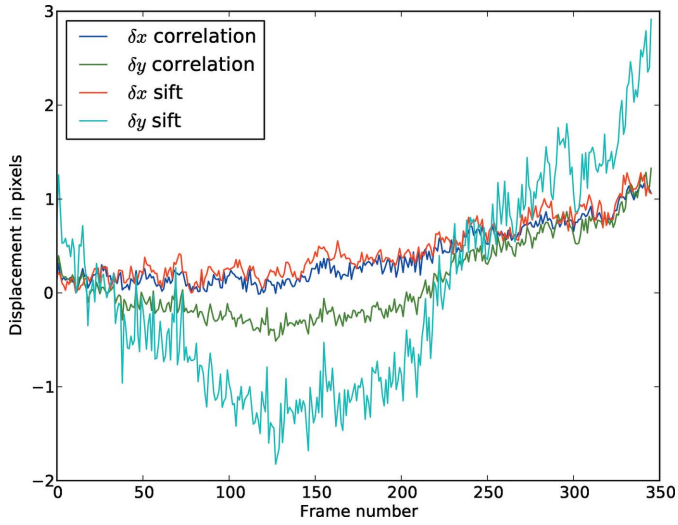

Figure 4
Offset along the x and y axes measured using image correlation and the SIFT procedure, respectively. All frames were aligned on frame 11.

blurring the image: a very blurred image represents a scene seen from a longer distance, where small-scale details are not visible. Features vanishing on a given scale are local maxima as obtained when subtracting the next blurred image from the current one. Finally, the keypoint orientation and descriptor are obtained from gradient orientation histograms.

3.3.1. Keypoints detection. The image is increasingly blurred by convolution with Gaussian kernels, in order to imitate the scale variations. Consecutive blurs are subtracted to obtain the so-called differences of Gaussian (DoG) images where pixels are defined in a three-dimensional scale-space, composed of their *spatial* positions (x, y) and their *scale* positions s (related to the blur factor). Keypoints are local maxima in this scale-space (x, y, s).

3.3.2. Keypoints refinement. At this stage, still many keypoints are not reliable: low-contrast keypoints are discarded using a fixed threshold, and many keypoints are located in regions of strong intensity gradient called edges. As keypoints situated on such edges can slide along the edge (due to noise for example), they are rejected in favour of those more constrained located on corners (based on the curvature of the edge). To improve the keypoints' accuracy, the coordinates are interpolated by a second-order Taylor expansion.

3.3.3. Orientation assignment. In order to make SIFT descriptors invariant to rotation, they are calculated according to the keypoint orientation, obtained using the following procedure. For each blurred version of the image, the gradient magnitude and orientation are computed. From the neighbourhood of a keypoint, a histogram of orientations is built (36 bins, 1 bin per 10°).

The maximum value of this histogram is the so-called dominant orientation, which represents the characteristic orientation of the keypoint. Additionally, every peak greater than 80% of the maximum peak generates a new keypoint with a different orientation.

3.3.4. Descriptor computation. Histograms of gradient orientations are generated around every keypoint to define its descriptor. The neighbourhood of a keypoint is divided into four regions of four sub-regions of 4 × 4 pixels. In every sub-region, an 8-bin histogram is computed; then all histograms are concatenated into a 128-value descriptor. The histogram is weighted by the gradient magnitudes and the current scale factor in order for the descriptor to be robust with respect to rotation, illumination, translation and scaling.

4. Implementation

Correlation algorithms have been easily ported on graphics cards (GPU) thanks to PyCUDA (Klöckner *et al.*, 2012) and cuFFT (Nvidia, 2007–2013), which makes them very fast and well suited for online data pre-processing. Conversely, the SIFT implementation used at the ESRF beamline ID21, based on Yu & Morel (2011), takes about 8 s per 4 Mpixel frame (a stack can contain up to 500 frames) using a single core. Although the process is distributed using the EDNA framework (Incardona *et al.*, 2009) on a 16-core computer, the performance limits are heavily constraining.

The SIFT algorithm (much more complicated than correlation algorithms) needed to be parallelized in order to benefit also from modern largely parallel hardware like GPU and other accelerator devices to achieve the required data rate.

SIFT_PyOCL is a Python library implementing this algorithm for GPU and other massively parallel computing devices. Like other synchrotron-centric tools (Mirone *et al.*, 2013; Favre-Nicolin *et al.*, 2011; Kieffer & Karkoulis, 2013), SIFT_PyOCL benefited from both a large scientific ecosystem based on NumPy (Oliphant, 2007) and the high-performance computing GPU capabilities.

4.1. Parallelization of the algorithm

Besides the Pythonic interface, most of the SIFT_PyOCL code is divided into dozens of functions designed to be executed in parallel and called kernels. These kernels are written in Open Computing Language (OpenCL) (Stone *et al.*, 2010) and can be run on various devices like GPU, multi-core CPU and accelerators. They are launched subsequently from a Python module using PyOpenCL (Klöckner *et al.*, 2012), which provides both execution speed and code readability.

Once the above-mentioned descriptors of two images are computed and matched, a least-squares method is used to determine the transformation which will convert keypoint positions of one image into the keypoint positions of the other. As yet, only affine transformations between images have been implemented in OpenCL using bi-linear interpolation.

Unlike existing parallel versions of SIFT (Lu, 2013; Rister *et al.*, 2013; Vasilyev *et al.*, 2011), the entire process is performed on the compute-device to avoid time-consuming transfers between central memory and the device's memory. This leads to several subtle tasks such as the use of *atomic* instructions or writing different versions of the same kernel to adapt it to various platforms (CPU or GPU) and devices (*e.g. compute capabilities* for Nvidia GPUs).

The implementation of the first steps of the algorithm (keypoints detection and refinement) did not raise any particular difficulty, since device parallelism allows handling every pixel by a single thread. Besides, convolution was implemented in the direct space (without Fourier transform), currently up to 100 times faster than the convolution performed in the C++ reference implementation of IPOL (Yu & Morel, 2011) (and even faster, depending on the CPU/GPU pair). A pyramid is used to represent the image in the scale space (Lowe, 2004).

The parallel implementation of the further steps (orientation assignment and descriptors computation) was more complex. For a given kernel, the performances strongly depend on the image complexity and on the device the code is executed on. Consequently, different versions have been written for a given kernel, each tailored to different platforms and devices, the optimal version being selected at run-time by the Python module based on the *compute capabilities* of the selected device.

4.2. Installation and usage

SIFT_PyOCL, as any Python module, can be installed from its sources, available on GitHub (Paleo & Kieffer, 2013). Whilst SIFT_PyOCL is open source and licensed under a very permissive MIT license, the SIFT algorithm itself is patented by the University of British Columbia (Lowe, 2003). This patent apparently applies only in the USA (and was not extended to other countries) and even there its use is tolerated for academic research purposes.[3] For any other use, the reader is referred to the University–Industry Liaison Office of the University of British Columbia.

Besides Python (version 2.6 or 2.7) and NumPy, SIFT_PyOCL needs PyOpenCL. All of them are available on Windows, Linux and MacOSX. It was tested on the OpenCL drivers from Nvidia on a large variety of their GPUs (Tesla, Fermi and Kepler generations), on accelerator cards such as the Intel Xeon Phi, and on multi-core processors with drivers from Intel, AMD and Apple. The full installation procedure is simply (including testing)

```
$ python setup.py build test install
```

Every single OpenCL kernel has been tested *versus* the reference implementation and can be run without installing the library by executing `test/test_all.py`.

4.3. Examples

In this section we have collected some basic examples of how SIFT_PyOCL can be used in IPython (Pérez & Granger, 2007) with Pylab (Hunter, 2007) mode: a Python interactive interface with scientific visualization capabilities.

4.3.1. Extract keypoints.

```
In [1]: import fabio
In [2]: img1 = fabio.open('image1.edf').data
In [3]: import sift
In [4]: siftplan = sift.SiftPlan(template=img1,
        devicetype="GPU")
In [5]: kp1 = siftplan.keypoints(img1)
```

After having imported the FabIO (Knudsen *et al.*, 2013) module in [1], a first absorption image is read in [2]. The SIFT_PyOCL library is loaded in [3] and the GPU is initialized in [4] with the whole required memory allocated on the device (depending on the image size). In [4], the keypoint extraction takes 60 ms for a 4 Mpixel image and returns a 261 keypoints vector as a numpy array named kp1 (depending on the image).

4.3.2. Match keypoints between images.

```
In [6]: img2 = fabio.open('image2.edf').data
In [7]: kp2 = siftplan.keypoints(img2)
In [8]: matchplan = sift.MatchPlan(devicetype="GPU")
In [9]: m = matchplan.match(kp1, kp2)
```

A second image is read [6] and its keypoints are extracted [7]. In [8], a matching object is created, targeted to run on a graphics card. Keypoint association is performed in [9], returning a numpy array of 66 lines and 2 columns of matched keypoints (execution time: 2.7 ms), each keypoint having attributes *x*, *y*, *scale* and *angle* in addition to the 128-byte descriptor.

[3] Personal communication from the inventor.

Table 1
Execution times for typical FFXAS images (2560 × 2160) using different implementations on a dual Intel E5-2667 (12 cores, 2.90 GHz) with an Nvidia Tesla K20m.

Implementation	Driver	Keypoint extraction	Keypoint matching	Image alignment
ASIFT	C++	3610 ms	95 ms	–
SIFT_PyOCL	AMD	522 ms	21 ms	578 ms
SIFT_PyOCL	Intel	741 ms	13 ms	843 ms
SIFT_PyOCL	Nvidia	75 ms	5.8 ms	90 ms

4.3.3. Align an image on a reference.

```
In [10]: aligner = sift.LinearAlign(img1,
         devicetype="GPU")
In [11]: img2_cor = aligner.align(img2)
```

It is also possible to align directly an image on a reference frame using an affine transformation: combination of translation, rotation, homothety, reflection, *etc.* An *aligner* object is defined in [10] by instantiating the *LinearAlign* class from a reference image and the device type. The *aligner* contains keypoints of the reference, a *SiftPlan* and a *MatchPlan* object plus the least-squares refinement and a bi-linear interpolation kernel. The *align* method performs keypoint extraction, matching, least-square optimization of the affine transformation coefficients and eventually applies the correction to the frame [11] by returning the corrected frame (execution time: 75 ms, all timings were measured on a Nvidia GeForce Titan GPU).

4.4. Performances

The SIFT_PyOCL implementation has been compared with the SIFT implementation from ASIFT (Yu & Morel, 2011) found on the IPOL server (reference implementation, single threaded) on a dual Intel E5-2667 (12 cores, 2.90 GHz) computer with an Nvidia Tesla K20m GPU. Execution speeds are summarized in Table 1. The acceleration measured on large images (more than 3 Mpixels) is between 30 and 100 times (on a GPU), depending on the image complexity, and from four to ten times when running on a multicore CPU. The most considerable speed-ups have been obtained on large images thanks to the massive acceleration of the Gaussian blurring.

4.5. Limitations

While all calculations are performed in single precision floating point (which is compatible even with the oldest graphic cards supported by OpenCL), the memory consumption has been traded off for performance: to prevent memory allocation on the device at run time (which is very costly), all buffers are allocated when the size of the image is given (at the plan creation). If the requested device does not have enough memory, the library tries to find another device, and is likely to pick the CPU which is slower. Moreover, the SIFT algorithm has been developed for linear colour scales and has some fixed built-in thresholds, making it unsuitable for high dynamic range images like diffraction patterns or even some back-light images with little contrast in dark areas.

5. Future prospects

SIFT_PyOCL has been included in the 4.7 release of the PyMca imaging tool (Solé *et al.*, 2007) as a plugin for image stack alignment, offering a user-friendly graphical interface which makes the image stack alignment both intuitive and easy to use. Registration of three-dimensional objects has a huge application potential, especially in the field of tomography and medical images. This field of research is very active and is fostered by the rapid progress in applied mathematics and computer vision.

6. Conclusion

The SIFT algorithm is currently used at the ID21 beamline of the ESRF for full-field X-ray absorption spectroscopy image alignment due to its robustness against scale, rotation and illumination changes. The SIFT_PyOCL Python module implements a parallel version of the SIFT algorithm running both on graphics cards and on multi-core processors, with appealing speed-ups. Its programming interface tries to be simple to use and pythonic while supporting high-performance computing 'under the hood'. We believe that it can be adopted by other software developers as a general purpose image alignment tool; this is why the code was made open-source and free for re-distribution.

The authors would like to thank the ID21 team, especially the beamline responsible, Marine Cotte, Murielle Salomé and Barbara Fayard, who are in charge of developing the full-field XAS technique. In the instrumentation division (ISDD) we would like to thank Claudio Ferrero, head of the data analysis unit, and Andy Götz, head of the software group, for supporting the parallelization work on the presented algorithm. V. Armando Solé, the author of PyMca, is also acknowledged for providing a graphical interface to the SIFT_PyOCL library and for all the debugging work he performed, testing it on all operating systems. The Barnes foundation and Jennifer Mass are also acknowledged for providing the painting fragment which was used as an example in this work and was originally studied to understand the degradation mechanisms of cadmium-based pigments.

References

Bay, H., Ess, A., Tuytelaars, T. & Van Gool, L. (2008). *Comput. Vis. Image Underst.* **110**, 346–359.
De Andrade, V., Susini, J., Salomé, M., Beraldin, O., Rigault, C., Heymes, T., Lewin, E. & Vidal, O. (2011). *Anal. Chem.* **83**, 4220–4227.
Favre-Nicolin, V., Coraux, J., Richard, M.-I. & Renevier, H. (2011). *J. Appl. Cryst.* **44**, 635–640.
Fayard, B., Pouyet, E., Berruyer, G., Bugnazet, D., Cornu, C., Cotte, M., Andrade, V. D., Chiaro, F. D., Hignette, O., Kieffer, J., Martin, T., Papillon, E., Salomé, M. & Solé, V. A. (2013). *J. Phys. Conf. Ser.* **425**, 192001.
Hunter, J. D. (2007). *Comput. Sci. Eng.* **9**, 90–95.
Incardona, M.-F., Bourenkov, G. P., Levik, K., Pieritz, R. A., Popov, A. N. & Svensson, O. (2009). *J. Synchrotron Rad.* **16**, 872–879.
Kieffer, J. & Karkoulis, D. (2013). *J. Phys. Conf. Ser.* **425**, 202012.
Klöckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P. & Fasih, A. (2012). *Parallel Comput.* **38**, 157–174.
Knudsen, E. B., Sørensen, H. O., Wright, J. P., Goret, G. & Kieffer, J. (2013). *J. Appl. Cryst.* **46**, 537–539.
Lowe, D. G. (1999). *Proc. Intl Conf. Comput. Vis.* **2**, 1150–1157.
Lowe, D. G. (2003). US Patent 6711293 B1.
Lowe, D. G. (2004). *Intl J. Comput. Vis.* **60**, 91–110.
Lu, M. (2013). *Appl. Math.* **7**, 717–722.
Mirone, A., Gouillart, E., Brun, E., Tafforeau, P. & Kieffer, J. (2013). *Nucl. Instrum. Methods Phys. Res. B*, http://arxiv.org/abs/1306.1392.
Moisan, L., Moulon, P. & Monasse, P. (2012). *Image Processing On Line*, http://dx.doi.Org/10.5201/ipol.2012.mmm-oh.
Nvidia (2007–2013). *CUDA CUFFT Library*, http://docs.nvidia.com/cuda/cufft/.
Oliphant, T. E. (2007). *Comput. Sci. Eng.* **9**, 10–20.
Oyallon, E. & Rabin, J. (2013). *Image Processing On Line*, http://www.ipol.im/pub/pre/69.
Paleo, P. & Kieffer, J. (2013). *An implementation of sift on gpu with opencl*, https://github.com/kif/sift_pyocl/archive/master.zip.

Pérez, F. & Granger, B. E. (2007). *Comput. Sci. Eng.* **9**, 21–29.
Rister, B., Wang, G., Wu, M. & Cavallaro, J. R. (2013). *IEEE International Conference on Acoustics, Speech and Signal Processing*, http://hgpu.org/?p=8983.
Solé, V., Papillon, E., Cotte, M., Walter, P. & Susini, J. (2007). *Spectrochim. Acta B*, **62**, 63–68.
Stone, J. E., Gohara, D. & Shi, G. (2010). *Comput. Sci. Eng.* **12**, 66–73.
Vasilyev, A. I., Boguslavskiy, A. A. & Sokolov, S. M. (2011). *21st International Conference on Computer Graphics and Vision (GraphiCon)*, 26–30 September 2011, Moscow, Russia.
Yu, G. & Morel, J.-M. (2011). *Image Processing On Line*, http://dx.doi.org/10.5201/ipol.2011.my-asift.