QuickSOFI: High-Performance Superresolution Optical Fluctuation Imaging (SOFI) Plugin for ImageJ Using GPU Computing

Nils Gustafsson

16 January 2014

Abstract: The field of superresolution microscopy holds the potential to truly visualise biological samples at the molecular scale – at resolution as low as 10nm, 20-fold greater than conventional microscopes. However these methods are still hindered by the need of large-data sampling and constrains such as a low densities of actively emitting fluorophores. Here we present an adaptation of a super resolution algorithm that tackles this limitation in the form of an easy to use plugin for ImageJ. This works holds the potential to simplify and greatly accelerate the acquisition of super-resolution data, potentially allowing live-cell superresolution to be achieved. This method is based on superresolution optical fluctuation imaging (SOFI) and accomplishes a 3D resolution enhancement by calculating the temporal or spatio-temporal cross cumulants of independent scholastically blinking fluorophores. SOFI analysis can be performed without extensive optimisation of fluorophore blinking characteristics, on conventional microscope setups and due to its simplicity will be available to a wide range of biological research applications.

Contents

Introduction

Since the first descriptions of the cell by Hooke in 1665 light microscopy has established itself as a main driver of cell biology research. In recent decades the development of techniques in fluorescence microscopy have become increasingly applicable (for a review see Lichtman & Conchello, 2005) with the search term “fluorescence” returning over 19000 articles in PubMed for 2013 alone. The prevalence of this technique is owing to a number of factors chief among them the ability to specifically label and differentiate biological molecules via targeted and typically non-invasive labelling with fluorescent dies or genetically encoded proteins. The versatility of these methods is evidenced in the Molecular Probes Handbook, 11th edition available online (http://probes.invitrogen.com-/handbook/). This handbook contains over 3000 bio-molecular labelling reagents for studying cell structure and function. The versatility extends to sample preparation and microscope modalities. Reconstruction of detailed three-dimensional architectures can be achieved both in live (with sub second accuracy) ad fixed cells at high resolution through methods such as confocal, two-photon, Total Internal Reflection Fluorescence (TIRF) and widefield microscopy. The importance of fluorescence microscopy shows little sign of diminishing and with a fast developing collection of superresolution techniques the past decade has seen the principal limitation to fluorescence microscopy, the resolution limit imposed by light diffraction, overcome.

In 1873 Ernest Abbe first described the diffraction limit (Abbe, 1873) for an imaging system as proportional to the wavelength of light and inversely proportional to the numerical aperture of the

objective lens. For practical reasons this relationship presents a barrier to the improvement in resolution much beyond 200nm using conventional far field microscopy. With modern optics and the use of refractive index matching between the sample and objective, by water or oil emersion, a numerical aperture of approximately 1.4 can be achieved without introducing significant aberrations. The use of wavelengths beyond the near ultraviolet results in undesirable effects ranging from gradual build-up of photo cross-linked proteins to cell death. Molecules and viruses however fall well below this threshold resulting in a reliance on methods such as electron microscopy and scanning probe microscopy which require expensive equipment and extensive sample preparation including sectioning and staining with metals.

The perception of the diffraction limit as being absolute was held until 1994 when Hell and Wichmann published a theoretical paper proposing the use of STimulated Emission Depletion (STED) to selectively "turn off" fluorophores effectively reducing the size of the otherwise diffraction limited, illuminating spot, of a scanning fluorescence microscope (Hell & Wichmann, 1994). This method was then subsequently implemented in 1999/2000 by Klar and Hell (Klar & Hell, 1999). Following this first proof of concept, methods such as REversible Saturable OpticaL Fluorescence Transitions (RESOLFT) have led to the rise of several superresolution techniques taking advantage of the stochastic switching between actively fluorescent and non-fluorescent states of fluorescent molecules.

Methods such as PALM (Betzig et al., 2006), STROM (Rust, Bates, & Zhuang, 2006), GSDIM (Fölling et al., 2008) and dSTORM (Heilemann et al., 2008) employ procedures where a large population of fluorophores are kept in a transiently non-fluorescent state, allowing only a minute and random subpopulation transition into a fluorescent state, until switching off again. The active fluorophores can then be individually identified and localised due to their low concentration, by fitting approximations to the point spread function (PSF) of the microscope, therefore estimating the position of the fluorophore with a high precision. A superresolution image is generated by plotting all the localised fluorophores in space. Generally a large number of frames, typically tens of thousands) need to be collected to recover the position of a sufficient population of fluorophores.

In principal the resolution is unlimited using these methods and 6nm resolutions have been demonstrated by Rittweger in 2006 (Rittweger, Han, Irvine, Eggeling, & Hell, 2009) using STED, 1nm with fluorophore fitting approaches (Yildiz et al., 2003) and previously unknown biology has already been elucidated using PALM, for example see Kanchanawong et al., 2010.

In direct comparison to conventional fluorescence techniques, superresolution microscopy still has many disadvantages. High incident light intensities, often exceeding acceptable levels for living systems, are required in STED to achieve complete depletion of fluorophores in the periphery of the PSF. Long acquisition times are required in single molecule location methods, such as PALM and STORM, limiting temporal resolution. Expensive equipment is often required to work with single molecules and aberrations are a significant difficulty which specific expertise is required to overcome. Expertise is also required to work with and exploit specific fluorophore properties and to understand and interpret data. The main strength of conventional techniques is the ability to non-invasively study live cells and superresolution techniques have not reliably proven to be capable despite a number of promising advances (Huang et al., 2013; Jones, Shim, He, & Zhuang, 2011; Klein et al., 2011).

First published in 2009 Superresolution Optical Fluctuation Imaging (SOFI) (T Dertinger, Colyer, Iyer, Weiss, & Enderlein, 2009) makes use of higher order, temporal statistical analysis of an image stack to discriminate between independently fluctuating fluorophores and improve resolution. This method has been shown to work at high fluorophore densities, using nonspecific photo-switching kinetics and supress noise and enhance contrast even at low signal to noise ratio (SNR) (Geissbuehler,

Dellagiacoma, & Lasser, 2011) with experimental resolution enhancements of up to five fold demonstrated (Geissbuehler et al., 2012).

The widespread adoption of this relatively simple superresolution technique has been limited because software used to implement SOFI has in general not been made freely available. Furthermore the computational complexity is quadratic in the order n of the statistical analysis implemented with the maximum resolution improvement achieved being n-fold meaning that real time reconstruction is limited to low resolution gains. To date only one freely available software implementing SOFI is available which makes use of $2nd$ order SOFI and conveys a resolution enhancement of only $\sqrt{2}$ (Dedecker & Neely, 2012).

Here I present QuickSOFI, an implementation of SOFI written in the Java programming language (http://www.oracle.com/us/technologies/java/overview/index.html) and making use of the Aparapi API (https://code.google.com/p/aparapi/) which, where possible, converts Java byte code to graphical processor unit (GPU) compatible OpenCL (http://www.khronos.org/opencl/). OpenCL is a high-performance instruction set to run data analysis massively parallelised on the GPU of computers, OpenCL generally improves data analysis speed by a factor of 100x or more. This code is ready to be packaged as a freely available and open-source plugin for the widely used ImageJ (http://rsb.info.nih.gov/ij/) software allowing real time reconstruction up to $6th$ order SOFI. I will also discuss the potential to produce maps of blinking statistics, molecular brightness and molecular density using the balanced SOFI (bSOFI) method (Geissbuehler et al., 2012) and the potential to acquire up to 30Hz superresolution images of live cells following sCMOS camera-specific modifications to the algorithm (Huang et al., 2013).

Superresolution Optical Fluctuation Imaging

For fluorescent labelling densities outside of the Förster resonance energy transfer regimes (>10nm separation) the emitters can be considered as non-interacting and fluctuating their intensity stochastically and independently of each other. This observation is exploited in SOFI to separate the emission contributions from individual fluorophores (for a recent review see Thomas Dertinger et al., 2013). Considering the correlation function at a point in an image it is intuitive that given non-interacting conditions a single fluctuating fluorophore will yield a highly correlated signal whereas a superposition of signals from two or more fluorophores will be less correlated.

For some position $\boldsymbol{r}$ in the image volume of a far field microscope the intensity recorded will be the convolution of the emitter distribution with the microscope's characteristic PSF U. This can be written as the sum over all contributions from k stationary emitters of brightness ε_k and normalized time dependent fluctuating intensity $s_k(t)$ giving the time dependent signal

$$I(\boldsymbol{r},t) = \sum_{k=1}^{N} U(\boldsymbol{r}-\boldsymbol{r}_k)\varepsilon_k s_k(t)$$

This intensity can be expressed as a zero mean fluctuation

$$\delta I(\boldsymbol{r},t) = I(\boldsymbol{r},t) - \langle I(\boldsymbol{r},t)\rangle_t$$

$$= \sum_{k=1}^{N} U(\boldsymbol{r}-\boldsymbol{r}_k)\varepsilon_k \delta s_k(t)$$

where $\langle \dots \rangle_t$ is an average over time. The second order auto-correlation function is then given by

$$G_2(\boldsymbol{r},\tau) = \langle \delta I(\boldsymbol{r},t+\tau)\cdot \delta I(\boldsymbol{r},t)\rangle_t$$

$$= \sum_{jk}^{N} U(\boldsymbol{r} - \boldsymbol{r}_j)U(\boldsymbol{r} - \boldsymbol{r}_k)\varepsilon_j\varepsilon_k\langle\delta s_j(t+\tau)\cdot\delta s_k(t)\rangle_t$$

For uncorrelated emitters cross-correlation terms $\langle\delta s_j(t+\tau)\cdot\delta s_k(t)\rangle_t$ for $j \neq k$ can be considered zero and the auto correlation appears as the sum of the PSF squared weighted by the squared brightness of the individual emitters and their autocorrelation function.

$$G_2(\boldsymbol{r},\tau) = \sum_{k}^{N} U^2(\boldsymbol{r} - \boldsymbol{r}_k)\varepsilon_k^2\langle\delta s_k(t+\tau)\cdot\delta s_k(t)\rangle_t$$

This value defines the second order SOFI image which provides a visualization of the fluorescence brightness and its degree of molecular correlation rather than the more conventional visualizations of intensity at a point in the image plane. Since the PSF is replaced with the square of the original, assuming an approximately Gaussian PSF, the resolution is improved by a factor of $\sqrt{2}$ in all three dimensions. Careful interpretation of this image is required as the molecular brightness is also raised to the power of 2 leading to a very large dynamic range and the possibility of masking dim emitters in close proximity to bright emitters. Also, a very bright emitter which does not fluctuate over time will not yield any correlation and will consequently not appear in the SOFI image. These effects are much more pronounced in higher order images but techniques now exist to correct the resulting image by combining information from several different order SOFI images (Geissbuehler et al., 2012).

Higher powers of the PSF and therefore better resolution can be achieved by considering higher order correlations. The nth order correlation function is given by

$$G_n(\boldsymbol{r},\tau_1,\dots,\tau_{n-1}) = \langle\delta I(\boldsymbol{r},t)\cdot\delta I(\boldsymbol{r},t+\tau_1)\cdots\delta I(\boldsymbol{r},t+\tau_{n-1})\rangle_t$$

This value however contains lower order correlation contributions containing terms where the PSF is not raised to the power n which would conceal the nth power contribution which arises from the fluctuations of a single emitter. The nth order auto cumulant, AC_n, however does not contain these cross terms and will ensure superresolution. It can be shown that

$$AC_n(\boldsymbol{r},\tau_1,\dots,\tau_{n-1}) = \sum_{k}^{N} U^n(\boldsymbol{r} - \boldsymbol{r}_k)\varepsilon_k^n w_k(\tau_1,\dots,\tau_{n-1})$$

where $w_k(\tau_1,\dots,\tau_{n-1})$ is the correlation based weighting function (T Dertinger et al., 2009). In principal the calculation of higher and higher auto cumulants will return an increasing reduction in PSF. To continue in this vein however requires oversampling of the PSF using higher magnification objectives. Consider a 150nm per pixel magnification and an original PSF spanning approximately 3 pixels. At some order n the PSF would span only a single pixel and further resolution gains would be pooled into that pixel negating any improvements. Over sampling can be deleterious however as the number of photons per pixel is reduced and SNR adversely effected. Furthermore it is computationally less expensive to use a zero time lag, $\tau_1 = \tau_2 = \cdots = \tau_n = 0$, however this will amplify camera shot noise. The use of spatio-temporal cross cumulants has been exploited (Thomas Dertinger, Colyer, Vogel, Enderlein, & Weiss, 2010) to overcome these problems by generating a finer sampling grid.

The second order cross cumulant, XC_n, is given by

$$XC_2(\boldsymbol{r_1},\boldsymbol{r_2},\tau_1,\tau_2) = \sum_{k=1}^{N} U(\boldsymbol{r}_1 - \boldsymbol{r}_k)U(\boldsymbol{r}_2 - \boldsymbol{r}_k)\varepsilon_k^2\langle\delta s_k(t+\tau_1)\cdot\delta s_k(t+\tau_2)\rangle_t$$

$$= U^2\left(\frac{\boldsymbol{r}_1 - \boldsymbol{r}_2}{2}\right)\cdot\sum_{k=1}^{N} U^2\left(\frac{\boldsymbol{r}_1 + \boldsymbol{r}_2}{2} - \boldsymbol{r}_k\right)\varepsilon_k^2\langle\delta s_k(t+\tau_1)\cdot\delta s_k(t+\tau_2)\rangle_t$$

Which implies that the location of the cross correlation pixel is the center of mass of the two pixels used in the cross correlation and that it is weighted by a distance factor, a PSF shaped weighting factor $U^2\left(\frac{r_1-r_2}{2}\right)$, dependent on the distance between the two pixels used. This distance factor can be corrected for by multiplying each pixel by the inverse of its distance factor. For a second order SOFI image every other pixel in x and y will have the same distance factor. This applies also to the diagonal pixels. The distance factor can be estimated by minimization of a cost function such as a 2D Laplacian. Equivalently, it can be found by taking the mean of all pixels which have the same distance factor and computing the factor required to equalise to the mean of the pixels with a different distance factor. Correction of this distance factor gives an estimate of the PSF which can be used in deconvolution analysis to convey a further resolution enhancement.

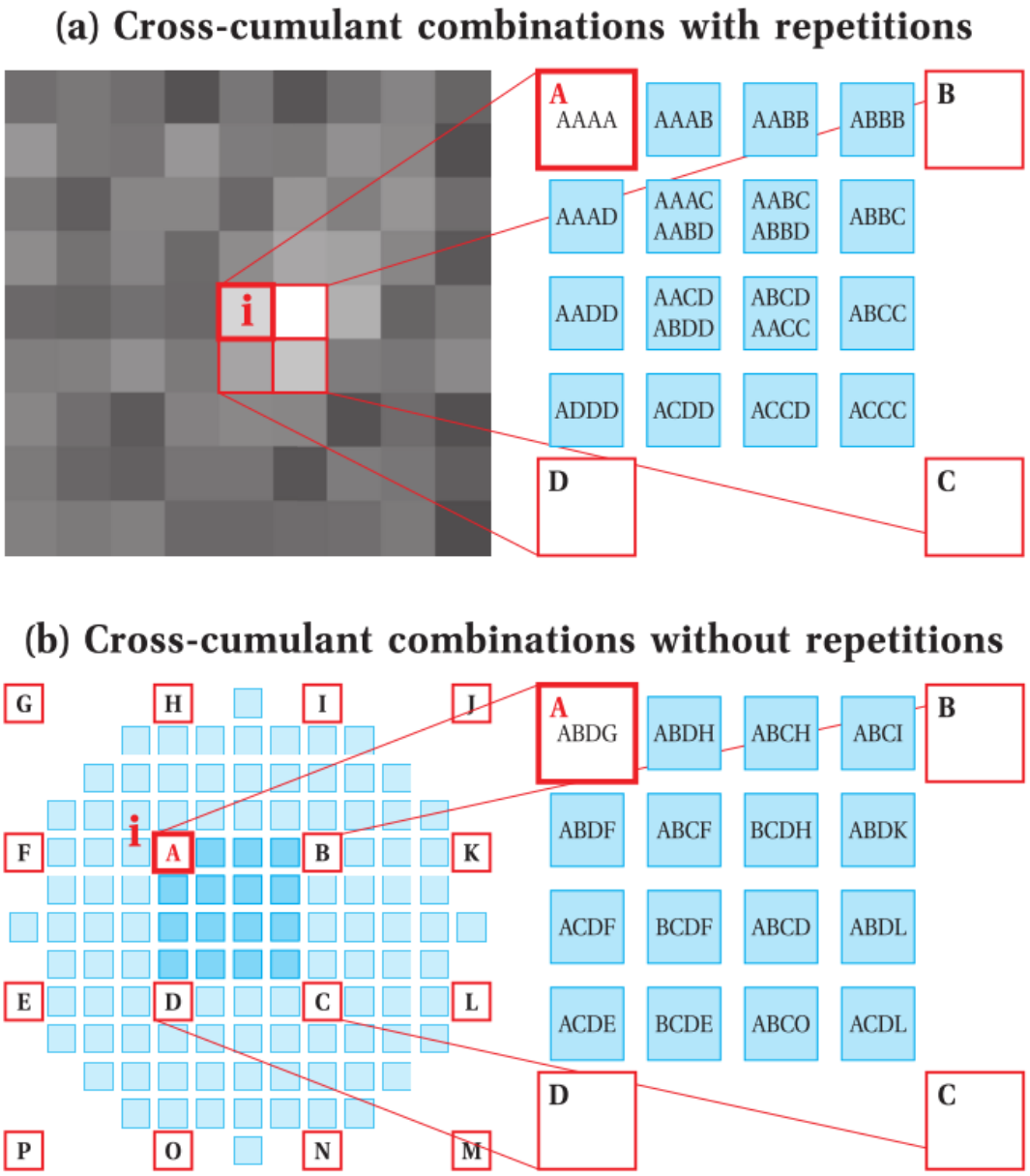


Figure 1: Different Combinations of a 4x4 pixel neighbourhood used to create the 15 virtual interpixels of the fourth order cross cumulant using a) repetitions and b) no repetitions. Taken from Geissbuehler et. al. 2011.

The cross cumulant calculations can be extended to higher orders with the general formula given by

$$XC_n(\boldsymbol{r_1},\dots,\boldsymbol{r_n},\tau_1,\dots,\tau_n)=\left[\prod_{j<l}^{n}U\left(\frac{\boldsymbol{r}_j-\boldsymbol{r}_l}{\sqrt{n}}\right)\right]\cdot\sum_{k=1}^{N}U^n\left(\frac{\sum_i^n\boldsymbol{r}_i}{n}-\boldsymbol{r}_k\right)\varepsilon_k^n w_k(\tau_1,\dots,\tau_{n-1})$$

An n^2 finer sampling grid can be achieved using this method either with repetitions in a 2x2 neighborhood matrix or without repetition in a 4x4 neighborhood ( ). This is possible up to 10th order before a larger neighborhood is required. A further advantage of this approach is that combinations which do not include repetition ( ) and thereby no auto-cumulant terms can use a zero time lag, $\tau=0$, without amplification of noise. Multiple combinations of the 4x4 neighborhood can be calculated and summed for each intra pixel further reducing the effect of noise although this is not required except when there is a very poor SNR (Geissbuehler et al., 2011). A recursive formula for the calculation of multivariate higher order cumulants and their relationship to higher order moments is given by (Mendel, 1991). In the general case the nth order cumulant (A.Shiryaev, 1959) is given by

$$C_n\left(\boldsymbol{r}=\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{r}_i;\tau\right)=\sum_P(-1)^{|P|-1}(|P|-1)!\prod_{p\in P}\langle\prod_{i\in p}I(\boldsymbol{r}_i,t-\tau_i)\rangle_t$$

For the pixel set $\mathbb{P}=\{\boldsymbol{r}_1,\boldsymbol{r}_2,\dots,\boldsymbol{r}_n\}$ with time lags $\tau_1,\tau_2,\dots,\tau_n$. P runs over all partitions of the set $\mathbb{S}=\{1,2,\dots,n\}$, which is the set of all possible non-overlapping, non-empty subsets of all elements of $\mathbb{S}$. $|P|$ is the number of parts in the partition P and p enumerates these parts.

For the sixth order cross cumulant, without repetitions, for each pixel in the original image there are 36 virtual pixels with positions $\boldsymbol{r}$ for which is a set $\mathbb{P}$ of 6 pixels selected from the neighborhood matrix. Since there are 203 partitions of the set $\mathbb{S}$ for $n=6$, in comparison with 52 for $n=5$ and 877 for $n=7$ this computation could quickly become unmanageable. However since $I(\boldsymbol{r}_i,t-\tau_i)$ has been mean subtracted, any part $p\in P$ which is only one element in length will have a mean of zero and its product with all $p\in P$ will also be zero. For $n=6$ this leaves only 41 partitions P of $\mathbb{S}$ which must be included in this sum. This however still requires a significant number of calculations to be made per pixel, per frame. Since each new pixel requires only knowledge of the original mean subtracted pixel neighborhood time series and only simple multiplicative operations are required this problem is easily parallelized such that the calculations for each original pixel are done concurrently.

Methods for further resolution enhancement beyond the $\sqrt{n}$ fold improvement conveyed by the raw cumulants up to a maximum of n fold improvements have been published. Thomas Dertinger et al., 2010 employs a reweighting scheme in the Fourier domain equivalent to a Wiener filter deconvolution and reconvolution with $U(n\boldsymbol{r})$. (Geissbuehler et al., 2012) explicitly splits this into a two-step process by using a Lucy-Richardson algorithm (Lucy, 1974; Richardson, 1972) which is an iterative deconvolution without regularization using a known PSF assuming Poisson distributed noise. This is followed by a reconvolution with $U(n\boldsymbol{r})$. The reason for separating these steps becomes clear on inspection of the deconvolved cumulant. Assuming perfect deconvolution this is given by

$$\check{C}_n(\boldsymbol{r})\approx\sum_k^N\delta(\boldsymbol{r}-\boldsymbol{r}_k)\varepsilon_k^n w_k(\tau_1,\dots,\tau_{n-1})$$

which can be linearised with respect to the brightness response ε_k^n by taking the nth root without canceling out the resolution improvement. This results in a final SOFI image with a n fold resolution improvement.

QuickSOFI

The aim in writing the program QuickSOFI (QSOFI) is to make available the advantages of superresolution imaging to a wide range of researchers with differing backgrounds and no prior expertise in superresolution. To this end the program has been written in the Java programming language to make use of the ImageJ API such that it can be packaged as a plugin to the widely used image processing programs ImageJ and FIJI Is Just ImageJ (FIJI). Additionally QSOFI is deigned to analyse large data sets quickly not only for the benefit of time but so that the output can be used in real time data acquisition control. As discussed it is possible to parallelise the SOFI algorithm on a pixel by pixel basis and this can be achieved using an appropriate GPU instruction set such as OpenCL. This however limits the availability of the algorithm to users with compatible hardware and the technical expertise to configure and compile source code. To overcome this limitation the Aparapi API has been used which extends the "Write Once Run Anywhere" philosophy of Java to GPU devices. Aparapi does this by first attempting to convert Java byte code into OpenCL at runtime and executing this code on the GPU. If no GPU is available the OpenCL code will be executed on the computer processing unit (CPU). If, for any reason, this is also not possible Aparapi makes use of the Java Virtual Machine's (JVM) internal thread pool. These options allow the program to be run on any platform with any hardware that has ImageJ/Fiji installed and a Java virtual machine (also required for ImageJ/FIJI) with the expense of speed the only drawback if a compatible GPU is not available.

QSOFI is currently able to perform AC-SOFI analysis with zero time lag up to $6th$ order, AC-SOFI with one frame time lag up to $4th$ order and XC-SOFI with no repetitions and zero time lag up to $4th$ order. The pixel neighborhood combinations without repetition and the formulas for XC calculation for $5th$ and $6th$ orders have been calculated for future implementation in QSOFI (Appendix: Supplementary Information).

QuickSOFI Structure

The structure of QSOFI is shown in the program flow diagram in Figure 2. The image data is received as an ImageJ ImagePlus object. Values for three options must be supplied. First the method, XC, AC, AC with time lag. Second the order, up to 6 for AC and up to 4 otherwise. Finally, the size, in frames, of the raw data which is to be used in each SOFI image. A separate input is provided automatically which tests for the presence of a GPU and assess its maximum memory capacity. Based on this memory capacity and the user input number of frames a block size is calculated which maximises the amount of

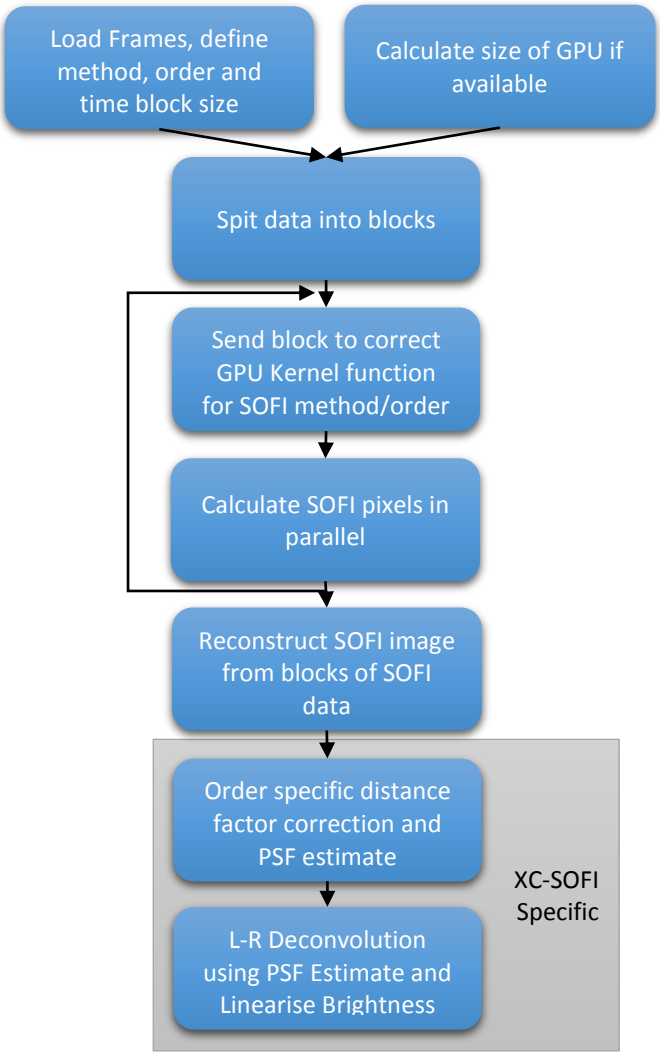


Figure 2: QuickSOFI program flow diagram

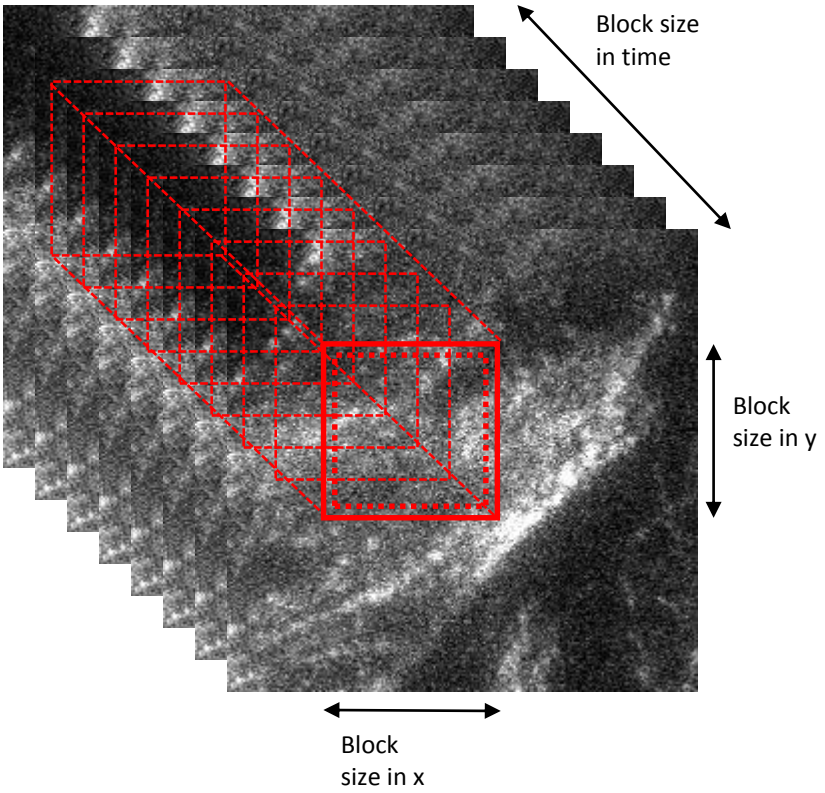


Figure 3: Blocking of image data to prevent overloading the memory of the GPU

data that is passed to the GPU to be analysed. This is done by splitting the data into blocks (Figure 3) with variable x and y sizes and constant frame number equal to the user input block size. For XC methods where a 4x4 neighborhood is required a two pixel wide margin is also included in this block. The edges where it is not possible to define the 4x4 pixel neighborhood as per Figure 1 are set to zero in the XC SOFI image output.

Each block of image data is passed to a GPU Kernel specific to the method and order chosen (for an example see Kernel_XCSOFI4.java, Appendix: Supplementary Code). The blocks of analysed SOFI data are recombined, without margins, to form a single SOFI image per time block in the original data. As previously discussed, for XC SOFI images the inter pixels generated by the analysis need to be reweighted by the inverse of the distance factor. This is done in QSOFI by passing the processed SOFI data to an order specific correction function which calculates the correction factors by equalising the mean values of differently weighted pixels. This method proved robust only for large data sets which contain variations across a large number of pixels. The alternative method of using a 2D Laplacian as a cost function and minimising iteratively has been implemented but not tested due to time constraints and no results using this method will be included here. Furthermore this correction can be used to define the PSF of the SOFI image thereby allowing a Lucy Richardson deconvolution to be used both to convey a further resolution enhancement and to allow linearisation of the brightness response. Similarly this has been implemented but not tested.

All components of QSOFI are unit tested and QSOFI is saved in a private online repository hosted by BitBucket (www.bitbucket.org) which will be made publicly available upon completion of the program and following extensive validation. Validation of QSOFI in its current state has been done using simulated data and in experiments on fixed HeLa cells with imunofluorescently labeled microtubules.

Results

Simulated data

The fluorescence signal from two independent emitters was simulated to test the SOFI algorithm. The two emitters were defined as 2D Gaussians with a sigma of 235nm and were separated by a distance of 500nm in both the horizontal and vertical directions giving a final separation of approximately 700nm. The image was constructed with 55nm pixel size such that 10 pixels covered the full width half maximum (FWHM) of the Gaussian. Both fluorophores fluctuated randomly between an emitting and a dark state at a rate 40% that of the frame rate. Their intensity in the emitting state was also modulated randomly with a variance of 1% that of the mean intensity. 200 frames of this simulation were used to calculate the average and the $2nd, 4th$ and $6th$ order AC-SOFI images (Figure: 4 a-d). The FWHM of the averaged image can be calculated from the sigma value to be approximately 555nm. Cross sections of the SOFI images (Figure 4 e) have been used to fit 1D Gaussians and a FWHM of approximately 400nm, 280nm, and 231nm for the $2nd$, $4th$ and $6th$ orders was found respectively. This indicates a resolution improvement of a factor of $1.38 \approx \sqrt{2}, 1.98 \approx \sqrt{4}$ and $2.40 \approx \sqrt{6}$ respectively as predicted by the theory. Similar results were observed up to $4th$ order using the XC-SOFI algorithm.

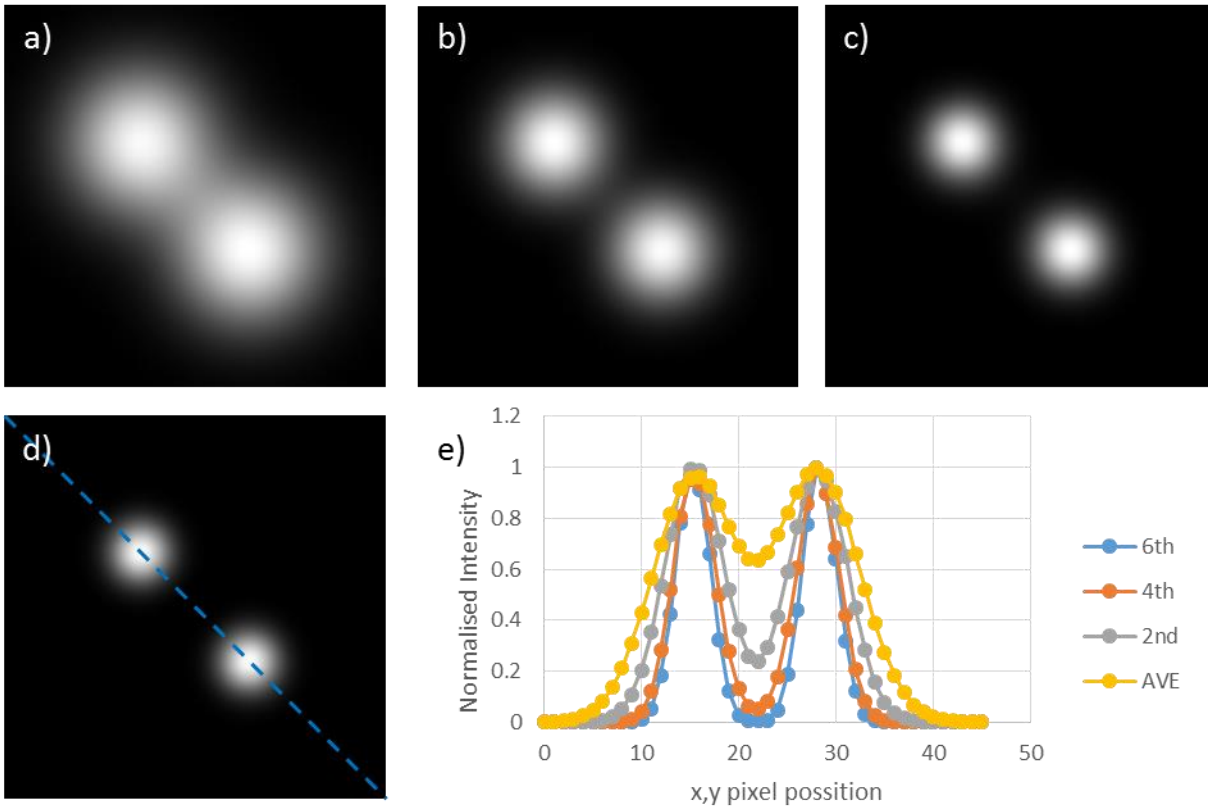


Figure 4: Noise and background free emitter pair simulations. a) Averaged intensity over 200 frames. b) Second order AC SOFI from 200 frames c) Fourth order AC SOFI from 200 frames d) Sicth order AC SOFI from 200 frames e) Normalised cross sections through the dashed line in d for each image.

Microscope Data

It was found using the data from Hela cells with imunofluorescent staining of the microtubules with Alexa 647 that a large number of frames were required to achieve full coverage of the image volume in the analysed SOFI image. Figure 5 a and b shows the average of a 1500 frame data set in red over laid with the $2nd$ and the $4th$ order XC-SOFI images respectively in green. Figure 5 c and d provides a

comparison between an average of another 2000 images and the $4th$ order XC-SOFI image. Visual inspection indicates there is some resolution improvement however not as good as theory predicts and this inspection is impaired by nonlinear brightness response.

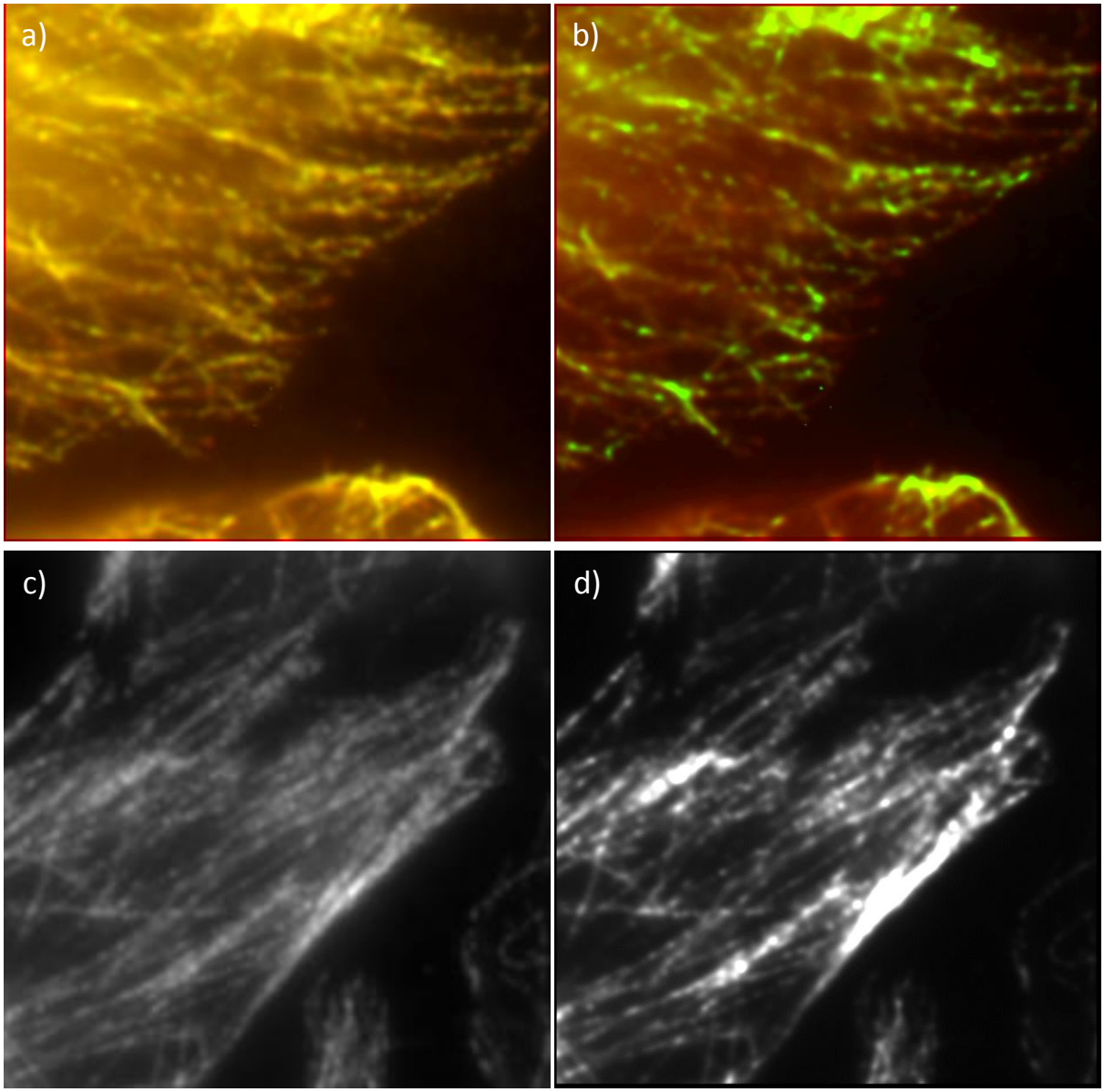


Figure 5: Alexa 647 stained microtubules in Hela Cells. a) Overlay of an average of 1500 images in red and the second order XC SOFI image from 1500 images in green. b) Overlay of an average of 1500 images in red and the fourth order XC SOFI image from 1500 images in green. c) Average of 2000 images for comparison with d) fourth order XC SOFI for 2000 images. Note that d) contains 4 times the number of pixels present in c). Interpolation was used to scale up original images in a) and b) to the size of the SOFI images.

Conclusions

The simulated data proves that QSOFI works as expected and this has been repeated for the XC-SOFI algorithm, results not shown. Only one sample could be imaged on the microscope in the time available. This was taken on a microscope optimised for STORM experiments which require low density fluctuations. Since the resolution gains of SOFI are dependent on the fluctuations this setup this requires a large number of images to be used in order to acquire full coverage of the image with fluorophores. This minimum requirement of full coverage is sufficient for single molecule localisation techniques such as STORM. For SOFI however, it is more appropriate to have fluctuations between emitting and non-emitting states at or close to the frame rate such that individual emitters switch

states a number of times in the image sequence. These results serve to show however the robustness of the algorithm and the ability to use it to produce small resolution gains in imaging conditions very far from the optimal set up. With further development of the code and optimisation of imaging conditions, keeping in mind sample preparation techniques and microscope hardware that is already widely employed outside the superresolution field, the simplicity of the implementation of QSOFI has the possibility to make it applicable to a wide range of cell biology research.

Future of QuickSOFI

A number of improvements to the current state of QSOFI have already been partially implemented or implemented but not tested. This includes the 5^{th} and 6^{th} order XC-SOFI algorithms and their associated distance factor correction algorithms. It also includes the Lucy-Richardson algorithm which allows linearisation of the brightness response and removes the difficulty of interpreting images with overly large dynamic ranges typical of higher order SOFI images (Figure 5).

A future plan for QSOFI should be to immediately extend it to include the calculations of ratios between different cumulant orders. As described in Geissbuehler et al., 2012 this allows the calculation of maps of molecular brightness, emitter on-time ratios and molecular densities. Specifically the map of the emitter on-time ratios allows the identification of structural gaps in higher order cumulants which can be filled in with data from lower orders to smooth gaps in the SOFI images with real data (bSOFI).

The most promising future for QSOFI is the possibility for simple, superresolution, live cell imaging at over 30Hz. Huang et al., 2013 have published methods for calculating masks that can be applied to data taken on scientific complementary metal-oxide semiconductor (sCMOS) cameras which correct for the intrinsic pixel dependent readout noise. Combined with the ability of SOFI under optimal conditions to achieve superresolution with possibly as little as 100 frames (T Dertinger et al., 2009). The use of an sCMOS camera at 1024 frames per second in conjunction with QSOFI could provide up to 100 frames per second superresolution movies.

Acknowledgements

I would like to thank P. Almada[1] for the preparation of samples, Dr A. Lowe[2] for the use of his microscope and Dr R. Enriques[3] for the provision of array to image stack and linear array index conversion source code. I would also like to thank Dr A. Lowe and Dr R. Enriques for their supervision of this project.

[1]BBSRC London Interdisciplinary Biosciences PhD Consortium, UCL.

[2]London Centre for Nanotechnology, UCL.

[3]Medical Research Council Laboratory of Molecular Cell Biology, UCL.

References

A.Shiryaev, V. L. and. (1959). On a Method of Calculation of Semi-Invariants. *Theory Probability App*, *4*(3), 319–329.

Abbe, E. (1873). Beiträge zur Theorie des Mikroskops und der mikroskopischen Wahrnehmung. *Archiv für mikroskopische Anatomie*, *9*(1), 413–418.

Betzig, E., Patterson, G. H., Sougrat, R., Lindwasser, O. W., Olenych, S., Bonifacino, J. S., … Hess, H. F. (2006). Imaging intracellular fluorescent proteins at nanometer resolution. *Science (New York, N.Y.)*, *313*(5793), 1642–5. doi:10.1126/science.1127344

Dedecker, P., & Neely, R. K. (2012). Localizer: fast, accurate, open-source, and modular software package for superresolution microscopy Peter. *Journal of Biomedical Optics 17(12), 126008*. doi:10.1117/1.JBO.17.12

Dertinger, T, Colyer, R., Iyer, G., Weiss, S., & Enderlein, J. (2009). Fast, background-free, 3D super-resolution optical fluctuation imaging (SOFI). *PNAS*, *106*(52), 22287–22292.

Dertinger, Thomas, Colyer, R., Vogel, R., Enderlein, J., & Weiss, S. (2010). Achieving increased resolution and more pixels with Superresolution Optical Fluctuation Imaging (SOFI). *Optics express*, *18*(18), 18875–85. Retrieved from http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3072111&tool=pmcentrez&rendertype=abstract

Dertinger, Thomas, Pallaoro, A., Braun, G., Ly, S., Laurence, T. a, & Weiss, S. (2013). Advances in superresolution optical fluctuation imaging (SOFI). *Quarterly reviews of biophysics*, *46*(2), 210–21. doi:10.1017/S0033583513000036

Fölling, J., Bossi, M., Bock, H., Medda, R., Wurm, C. A., Hein, B., … Hell, S. W. (2008). Fluorescence nanoscopy by ground-state depletion and single-molecule return, *5*(11), 943–945. doi:10.1038/NMETH.1257

Geissbuehler, S., Bocchio, N. L., Dellagiacoma, C., Berclaz, C., Leutenegger, M., & Lasser, T. (2012). Mapping molecular statistics with balanced super-resolution optical fluctuation imaging (bSOFI). *Optical Nanoscopy*, *1*(1), 4. doi:10.1186/2192-2853-1-4

Geissbuehler, S., Dellagiacoma, C., & Lasser, T. (2011). Comparison between SOFI and STORM. *Biomedical optics express*, *2*(3), 408–20. doi:10.1364/BOE.2.000408

Heilemann, M., van de Linde, S., Schüttpelz, M., Kasper, R., Seefeldt, B., Mukherjee, A., … Sauer, M. (2008). Subdiffraction-resolution fluorescence imaging with conventional fluorescent probes. *Angewandte Chemie (International ed. in English)*, *47*(33), 6172–6. doi:10.1002/anie.200802376

Hell, S. W., & Wichmann, J. (1994). Breaking the diffraction resolution limit by stimulated emission : stimulated-emission-depletion fluorescence microscopy, *19*(11), 780–782.

Huang, F., Hartwich, T. M. P., Rivera-Molina, F. E., Lin, Y., Duim, W. C., Long, J. J., … Bewersdorf, J. (2013). Video-rate nanoscopy using sCMOS camera-specific single-molecule localization algorithms. *Nature methods*, *10*(7), 653–8. doi:10.1038/nmeth.2488

Jones, S. a, Shim, S.-H., He, J., & Zhuang, X. (2011). Fast, three-dimensional super-resolution imaging of live cells. *Nature methods*, *8*(6), 499–508. doi:10.1038/nmeth.1605

Kanchanawong, P., Shtengel, G., Pasapera, A. M., Ramko, E. B., Davidson, M. W., Hess, H. F., & Waterman, C. M. (2010). Nanoscale architecture of integrin-based cell adhesions. *Nature*, *468*(7323), 580–4. doi:10.1038/nature09621

Klar, T. A., & Hell, S. W. (1999). Subdiffraction resolution in far-field fluorescence microscopy, *24*(14), 954–956.

Klein, T., Löschberger, A., Proppert, S., Wolter, S., van de Linde, S., & Sauer, M. (2011). Live-cell dSTORM with SNAP-tag fusion proteins. *Nature methods*, *8*(1), 7–9. doi:10.1038/nmeth0111-7b

Lichtman, J. W., & Conchello, J. (2005). Fluorescence microscopy, *2*(12). doi:10.1038/NMETH817

Lucy, L. B. (1974). An iterative technique for the rectification of observed distributions. *The Astronomical Journal*. doi:10.1086/111605

Mendel, J. M. (1991). Tutorial on Higher-Order Statistics (Spectra) in Signal Processing and System Theory: Theoretical Results and Some Applications. *Proceedings of the IEEE*, *79*(3).

Osborn, M., Webster, R. E., Weber, K., & Planck, M. (1978). Cells and Antibodies, (14).

Richardson, W. (1972). Bayesian-based iterative method of image restoration. *Journal of the Optical Society of America*, *62*(1), 55–59. doi:10.1364/JOSA.62.000055

Rittweger, E., Han, K. Y., Irvine, S. E., Eggeling, C., & Hell, S. W. (2009). STED microscopy reveals crystal colour centres with nanometric resolution, *3*(March), 1–4. doi:10.1038/NPHOTON.2009.2

Rust, M. J., Bates, M., & Zhuang, X. (2006). imaging by stochastic optical reconstruction microscopy (STORM), *3*(10), 793–795. doi:10.1038/NMETH929

Yildiz, A., Forkey, J. N., McKinney, S. a, Ha, T., Goldman, Y. E., & Selvin, P. R. (2003). Myosin V walks hand-over-hand: single fluorophore imaging with 1.5-nm localization. *Science (New York, N.Y.)*, *300*(5628), 2061–5. doi:10.1126/science.1084398

Appendix

Supplementary Information

Methods

Sample Preperation

HeLa cells were cultured in DMEM media complemented with Fetal Bovine Serum (FBS) and penicillin and streptomycin. Cell fixation was achieved as previously described (Osborn, Webster, Weber, & Planck, 1978). Briefly, cells were bathed for 10 minutes in ice cold methanol to fix and, washed in 1x PBS (pH 7.2). Cells were incubated for 30 minutes in 10% FBS blocking solution followed by a 60 minute incubation with a mouse raised anti-tubulin antibody (Sigma, T9026). Cells were washed 3x for 5 minutes in PBS before incubation with goat raised anti-mouse antibody Alexa 647 (Life Technologies). Cells were washed 3x with PBS before finally being bathed for 5 min in DAPI diluted in PBS to stain the nucleus.

Imaging

Imaging was performed on a custom built microscope, using an Olympus IX81 body as a base. A single laser was used for illumination (100 mW 640 nm Coherent Cube). The beam was expanded and half-wave plates were used to adjust the polarisation before passing the beam through an Acousto-Optical Tunable Filter (AOTF, AA Optoelectronics, France) to quickly modulate laser power. The beam was

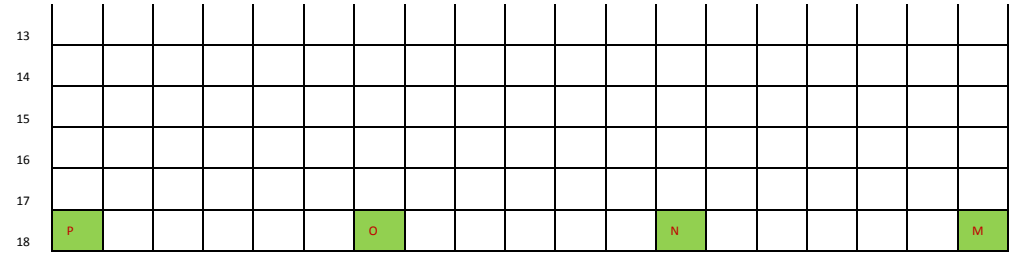
again expanded, spatially filtered and passed through a quarter-wave plate to circularly polarise. The free beam then passed through the TIRF lenses and was focused directly onto the back focal plane of the objective (Olympus 60x or 100x 1.49N.A. TIRF apochromatic objective) via a multi-edge dichroic filter (Semrock). An actively cooled EMCCD camera (iXon+ or iXon Ultra) was coupled to the camera port of the microscope via an additional 2x magnifying relay. Laser shutter, AOTF and camera firing were synchronised using a Data Translation DT9834 data acquisition module. Sample positioning was controlled via a micrometer stage with a XYZ-Nanopositioning stage (Physik Instrumente). Camera acquisition was at 30Hz. All software to control the microscope was written in C++ and Python.

Higher Order Pixels and cumulants

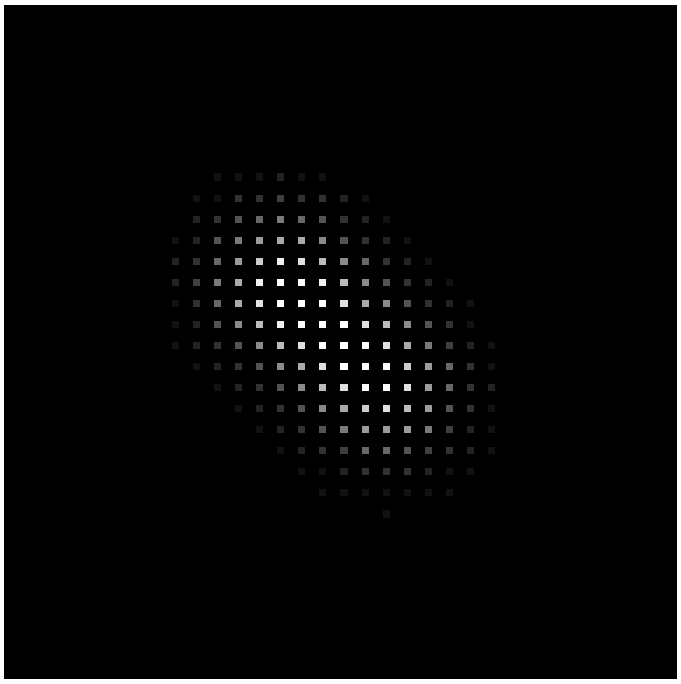
The 4x4 neighbourhood can be used in combinations of 5 to create 25 inter pixels for 5th order XCSOFI and in combinations of 6 to create 36 inter pixels for 6th order XCSOFI. Possible combinations (without repetitions) are shown in below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	G					H					I					J
1																
2																
3																
4																
5	F					FHC IE	JOG FB	KCF GB	BAF LI	JND HI	B					K
6						GH ND A	IPB GC	KOH EI	GHK DM	ILJE D						
7						ANC GF	PKO GI	PBJ NG	ACJ BP	BLD CH						
8						HOF AN	FH MC E	LOF DI	FCJ ND	BPJ OK						
9						BDH PO	FDP CK	AD OH M	PNJ FL	AKB NO						
10	E					D					C					L
11																
12																
13																
14																
15	P					O					N					M

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	G						H						I						J
1																			
2																			
3																			
4																			
5																			
6	F						GC FB HD	GHI AE N	GC HD AB	AB HDI C	JDI CA B	AIF LBK	B						K
7							FG CA OI	AF MIE H	HJN EFB	OL BAI G	FDL BK H	KOJ FIC							
8							GN DIH P	IEH MF D	EH AO KB	GP OJK B	CJG OIN	CJH OL A							
9							EH NF PJ	PH MG CA	EOI MB G	AC BD GM	BF NK GM	ON JAH L							
10							AP DNI F	EDL IPA	NA PJE B	NP GLJ D	DM KH OA	ON MIF J							
11							ECB HO P	MA BO EF	DG LM AP	DCF AL N	JCN FLP	ON LGI M							
12	E						D						C						L



The necessity for a correction of the differently weighted XC pixels is demonstrated by the uncorrected third order XC-SOFI image of two simulated emitters (Supplementary Figure 1).



Supplementary Figure 1: 3rd order XC-SOFI without distance factor correction

Supplementary Code

Kernel_XCSOFI4.java

```
1    package qs1java.CumulantCalc;
2
3    import com.amd.aparapi.Kernel;
4    import com.amd.aparapi.Range;
5
6    /**
7     * Created with IntelliJ IDEA.
8     * User: nilsgustafsson
9     * Date: 08/01/2014
10    * Time: 20:18
11    * To change this template use File | Settings | File Templates.
12    */
13   public class Kernel_XCSOFI4 extends Kernel {
14       float subSOFIData[];
15       float subData[];
16       int trueFullBlockSizeX, trueFullBlockSizeY, trueFullBlockSizeXY,
17               trueBlockSizeT, trueMarginStartSizeX, trueMarginStartSizeY,
18               trueMarginEndSizeX, trueMarginEndSizeY,
19               trueInteriorBlockSizeX, trueInteriorBlockSizeY,
20               trueInteriorBlockSizeXY;
21
22       public float[] SOFIAlgorithm(float [] subData, int trueFullBlockSizeX,
23                                    int trueFullBlockSizeY, int trueBlockSizeT,
24                                    int trueMarginStartSizeX, int trueMarginStartSizeY,
```

```
                                    int trueMarginEndSizeX, int trueMarginEndSizeY,
                                    int trueInteriorBlockSizeX, int trueInteriorBlockSizeY){
            this.subData = subData;
            this.trueFullBlockSizeX = trueFullBlockSizeX;
            this.trueFullBlockSizeY = trueFullBlockSizeY;
            this.trueFullBlockSizeXY = this.trueFullBlockSizeX*this.trueFullBlockSizeY;
            this.trueMarginStartSizeX = trueMarginStartSizeX;
            this.trueMarginStartSizeY = trueMarginStartSizeY;
            this.trueMarginEndSizeX = trueMarginEndSizeX;
            this.trueMarginEndSizeY = trueMarginEndSizeY;
            this.trueBlockSizeT = trueBlockSizeT;
            this.trueInteriorBlockSizeX=trueInteriorBlockSizeX;
            this.trueInteriorBlockSizeY=trueInteriorBlockSizeY;
                                                    this.trueInteriorBlockSizeXY     =
this.trueInteriorBlockSizeX*this.trueInteriorBlockSizeY;
            //4th order XCSOFI produces 4x4=16 pixels per original pixel
            this.subSOFIData = new float [this.trueInteriorBlockSizeXY*16];
            //upload data to GPU
            setExplicit(true);
            put(this.subData);
            put(this.subSOFIData);
            //launch one thread for every original pixel in the block
            execute(Range.create(this.trueFullBlockSizeXY));
            //download data from GPU and return to call
            get(this.subSOFIData);
            return this.subSOFIData;
        }

        @Override public void run(){
            //p enumerates the thread i.e. original pixel
            //x,y are positions in original image of pixel p
            //xout, yout are pixel positions in the up sampled output image
            //pout enumerates these pixels
            int p, x, y, xout, yout, pout;
            p=getGlobalId();
            y=p/this.trueFullBlockSizeX;
            x=p-(this.trueFullBlockSizeX*y);
            xout=(x-this.trueMarginStartSizeX)*4;
            yout=(y-this.trueMarginStartSizeY)*4;
            pout=xout+yout*this.trueInteriorBlockSizeX*4;
            //calculate mean subtracted data
            float ave = 0;
            for(int t=0; t<this.trueBlockSizeT;t++){
                ave = ave + this.subData[p+t*this.trueFullBlockSizeXY];
            }
            ave = ave / ((float) this.trueBlockSizeT);
            this.subData[p] = this.subData[p]-ave;
            //all threads must wait here for concurancy
            localBarrier();
            //calculate XC for each 16 pixels in up sampled image
            float ABDG = 0;
            float ABDH = 0;
            float ABCH = 0;
            float ABCI = 0;
            float ABDF = 0;
            float ABCF = 0;
            float BCDH = 0;
            float ABDK = 0;
            float ACDF = 0;
            float BCDF = 0;
            float ABCD = 0;
            float ABDL = 0;
            float ACDE = 0;
            float BCDE = 0;
            float ABCO = 0;
            float ACDL = 0;

            for(int t=0; t<this.trueBlockSizeT;t++){
                //do not calculate if the 4x4 neighbourhood does not exist
                //False for margins and for original image edge pixels
                if(x>0&&y>0&&x<trueFullBlockSizeX-2&&y<trueFullBlockSizeY-2)  {
                    float A = this.subData[p+t*this.trueFullBlockSizeXY];
                    float B = this.subData[p+1+t*this.trueFullBlockSizeXY];
                                                                        float    C    =
this.subData[p+1+this.trueFullBlockSizeX+t*this.trueFullBlockSizeXY];
                 float D = this.subData[p+this.trueFullBlockSizeX+t*this.trueFullBlockSizeXY];
```

```
99                                                                  float E = this.subData[p-1+this.trueFullBlockSizeX+t*this.trueFullBlockSizeXY];
100                 float F = this.subData[p-1+t*this.trueFullBlockSizeXY];
101                                                                 float G = this.subData[p-1-this.trueFullBlockSizeX+t*this.trueFullBlockSizeXY];
102                 float K = this.subData[p+2+t*this.trueFullBlockSizeXY];
103                                                                     float L = this.subData[p+2+this.trueFullBlockSizeX+t*this.trueFullBlockSizeXY];
104             float H = this.subData[p-this.trueFullBlockSizeX+t*this.trueFullBlockSizeXY];
105                                                                 float I = this.subData[p+1-this.trueFullBlockSizeX+t*this.trueFullBlockSizeXY];
106                                                                     float O = this.subData[p+2*this.trueFullBlockSizeX+t*this.trueFullBlockSizeXY];
107                 ABDK=ABDK+(A*B*D*K)-((A*B)*(D*K))-((A*D)*(B*K))-((A*K)*(D*B));
108                 ABDL=ABDL+(A*B*D*L)-((A*B)*(D*L))-((A*D)*(B*L))-((A*L)*(D*B));
109                 ACDL=ACDL+(A*C*D*L)-((A*C)*(D*L))-((A*D)*(C*L))-((A*L)*(D*C));
110                 ABCO=ABCO+(A*B*C*O)-((A*B)*(C*O))-((A*C)*(B*O))-((A*O)*(C*B));
111                 ABDF=ABDF+(A*B*D*F)-((A*B)*(D*F))-((A*D)*(B*F))-((A*F)*(D*B));
112                 ABCF=ABCF+(A*B*C*F)-((A*B)*(C*F))-((A*C)*(B*F))-((A*F)*(C*B));
113                 ACDF=ACDF+(A*C*D*F)-((A*C)*(D*F))-((A*D)*(C*F))-((A*F)*(D*C));
114                 BCDF=BCDF+(B*C*D*F)-((B*C)*(D*F))-((B*D)*(C*F))-((B*F)*(D*C));
115                 ACDE=ACDE+(A*C*D*E)-((A*C)*(D*E))-((A*D)*(C*E))-((A*E)*(D*C));
116                 BCDE=BCDE+(B*C*D*E)-((B*C)*(D*E))-((B*D)*(C*E))-((B*E)*(D*C));
117                 ABDH=ABDH+(A*B*D*H)-((A*B)*(D*H))-((A*D)*(B*H))-((A*H)*(D*B));
118                 ABCH=ABCH+(A*B*C*H)-((A*B)*(C*H))-((A*C)*(B*H))-((A*H)*(C*B));
119                 ABCI=ABCI+(A*B*C*I)-((A*B)*(C*I))-((A*C)*(B*I))-((A*I)*(C*B));
120                 BCDH=BCDH+(B*C*D*H)-((B*C)*(D*H))-((B*D)*(C*H))-((B*H)*(D*C));
121                 ABDG=ABDG+(A*B*D*G)-((A*B)*(D*G))-((A*D)*(B*G))-((A*G)*(D*B));
122                 ABCD=ABCD+(A*B*C*D)-((A*B)*(C*D))-((A*C)*(B*D))-((A*D)*(B*C));
123             }
124         }
125         ABDG = ABDG / ((float) this.trueBlockSizeT);
126         ABDH = ABDH / ((float) this.trueBlockSizeT);
127         ABCH = ABCH / ((float) this.trueBlockSizeT);
128         ABCI = ABCI / ((float) this.trueBlockSizeT);
129         ABDF = ABDF / ((float) this.trueBlockSizeT);
130         ABCF = ABCF / ((float) this.trueBlockSizeT);
131         BCDH = BCDH / ((float) this.trueBlockSizeT);
132         ABDK = ABDK / ((float) this.trueBlockSizeT);
133         ACDF = ACDF / ((float) this.trueBlockSizeT);
134         BCDF = BCDF / ((float) this.trueBlockSizeT);
135         ABCD = ABCD / ((float) this.trueBlockSizeT);
136         ABDL = ABDL / ((float) this.trueBlockSizeT);
137         ACDE = ACDE / ((float) this.trueBlockSizeT);
138         BCDE = BCDE / ((float) this.trueBlockSizeT);
139         ABCO = ABCO / ((float) this.trueBlockSizeT);
140         ACDL = ACDL / ((float) this.trueBlockSizeT);
141
142         //only asign a value to the output data array that comes from the interior block
143                             if(x>=this.trueMarginStartSizeX&&x<(this.trueFullBlockSizeX-this.trueMarginEndSizeX)
144                              &&y>=this.trueMarginStartSizeY&&y<(this.trueFullBlockSizeY-this.trueMarginEndSizeY)){
145             this.subSOFIData[pout]=abs(ABDG);
146             this.subSOFIData[pout+1]=abs(ABDH);
147             this.subSOFIData[pout+2]=abs(ABCH);
148             this.subSOFIData[pout+3]=abs(ABCI);
149             this.subSOFIData[pout + this.trueInteriorBlockSizeX*4]=abs(ABDF);
150             this.subSOFIData[pout + 1 + this.trueInteriorBlockSizeX*4]=abs(ABCF);
151             this.subSOFIData[pout + 2 + this.trueInteriorBlockSizeX*4]=abs(BCDH);
152             this.subSOFIData[pout + 3 + this.trueInteriorBlockSizeX*4]=abs(ABDK);
153             this.subSOFIData[pout + this.trueInteriorBlockSizeX*8]=abs(ACDF);
154             this.subSOFIData[pout + 1 + this.trueInteriorBlockSizeX*8]=abs(BCDF);
155             this.subSOFIData[pout + 2 + this.trueInteriorBlockSizeX*8]=abs(ABCD);
156             this.subSOFIData[pout + 3 + this.trueInteriorBlockSizeX*8]=abs(ABDL);
157             this.subSOFIData[pout + this.trueInteriorBlockSizeX*12]=abs(ACDE);
158             this.subSOFIData[pout + 1 + this.trueInteriorBlockSizeX*12]=abs(BCDE);
159             this.subSOFIData[pout + 2 + this.trueInteriorBlockSizeX*12]=abs(ABCO);
160             this.subSOFIData[pout + 3 + this.trueInteriorBlockSizeX*12]=abs(ACDL);
161         }
162     }
163 }
```