to extract the morphological parameters, using the simple computational algorithms previously described (Table 5.1).

Parameter	Measured Value
Average Number of Pores per Slice	15
Average Porosity	38.5%
Void Space Ratio	37.4%
Pore Circularity	0.44 ± 0.24
Pore Roundness	0.55 ± 0.16
Pore Equivalent Diameter (mean)	16.3 µm
Pore Equivalent Diameter (mode)	3.8 µm
Pore Equivalent Diameter (min)	3.6 µm
Pore Equivalent Diameter (max)	77.8 µm

Table 5.1 Measured values of the shape descriptor parameters obtained by image processing of the CARS 3D reconstruction of the PHBHV-gel scaffold.

A collagen-based scaffold has been also characterized immersed in culture medium using SHG microscopy technique. A water immersion 40x objective (LUMPLFLN 40XW NA=0.8, W.D.=3.3mm, Olympus) was used to focus excitation sources on the sample, while a 20x objective (UPLSAPO 20x objective NA=0.75, W.D.=0.6 mm, Olympus) was used to collect SHG signal in forward detection. A scaffold volume of about 353x353x65 μm^3 (Fig. 5.21) was then analyzed to extract the morphological parameters (Table 5.2).

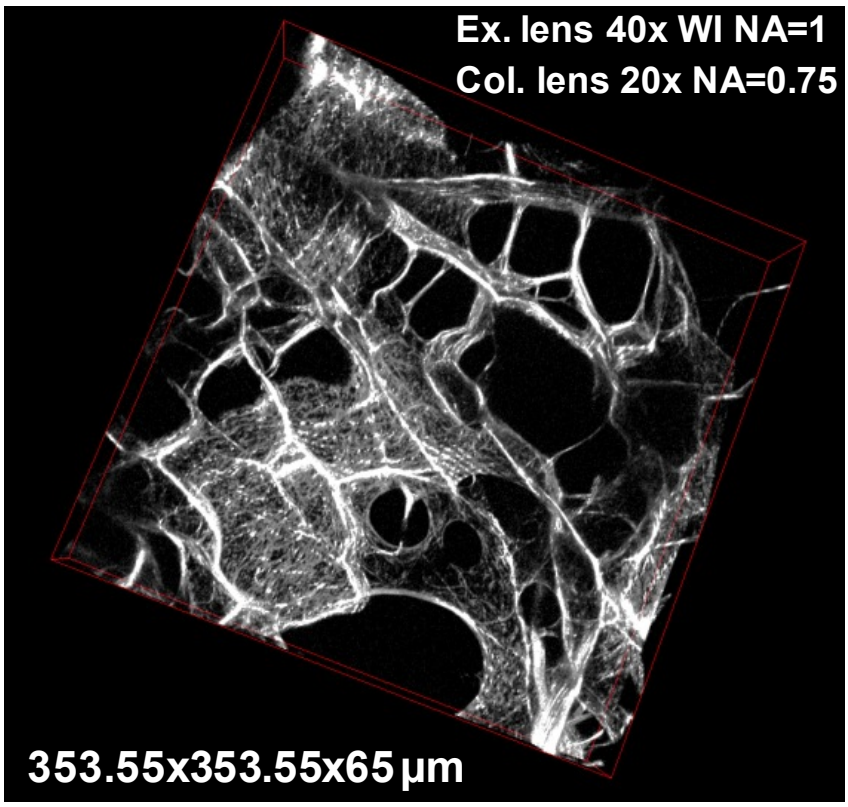

Fig. 5.21 Three-dimensional view of a collagen-based scaffold acquired using SHG microscopy.

Parameter	Measured Value
Average Number of Pores per Slice	11
Average Porosity	90.8%
Void Space Ratio	85.3%
Pore Circularity	0.38 ± 0.19
Pore Roundness	0.57 ± 0.19
Pore Equivalent Diameter (mean)	79.2 µm
Pore Equivalent Diameter (mode)	9.5 µm
Pore Equivalent Diameter (min)	6.3 µm
Pore Equivalent Diameter (max)	358.5 µm

Table 5.2 Measured values of the shape descriptor parameters obtained by image processing of the SHG 3D reconstruction of the collagen-based scaffold.

Optical attenuation and distortion phenomena in microscopy could lead to measurement artifacts that depend on the sample characteristics. A comparison between CARS and TPEF microscopy techniques has been done imaging the same scaffold simultaneously with the two techniques. The K-BC2000 scaffold was chosen for this experiment and it was stained with cyanine3 fluorophore (Cy3, Cyanine Technologies) in a way it could be imaged also using TPEF with the same wavelength of the pump signal used to excite CARS process. The scaffold was immersed in a solution of high concentrated cyanine3 (about 1 µmol/mol) and staining was obtained leaving the molecules to penetrate the scaffold and to deposit on its surface until the fluorophore solution was completely evaporated.

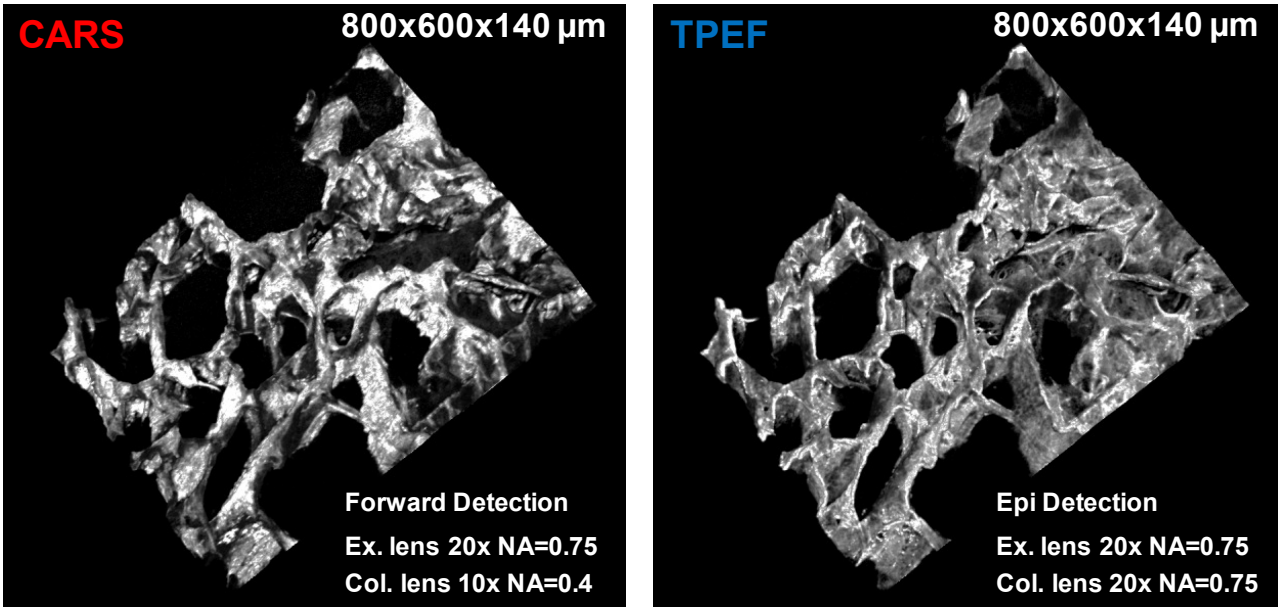

Fig. 5.22 Comparison between 3D reconstructions obtained using CARS (left image) and TPEF (right image) of a portion of KBC2000 scaffold stained with cyanine3.

A 3D imaging of dried stained K-BC2000 scaffold was performed acquiring a volume of about 800x600x140 μm^3 using a 20x objective lens (UPLSAPO 20x objective NA=0.75, W.D.=0.6 mm, Olympus) to focus the

excitation sources and the same objective lens was used to collect the back propagating TPEF signal while a 10x objective lens (UPLSAPO 10x objective NA=0.4, W.D.=3.1 mm, Olympus) was used to collect the forward propagating CARS signal. It is possible to see that the images of the scaffold acquired with the two techniques (Fig. 5.22) do overlap each other even if some slight differences can be easily recognized. A morphological analysis was conducted using the simple computational algorithms previously described to extract the morphological parameters for data collected with both techniques (Table 5.3).

Parameter	Measured Value from CARS Images	Measured Value from TPEF Images
Average Number of Pores per Slice	6	15
Average Porosity	86.6%	82.6%
Void Space Ratio	89.6%	87%
Pore Circularity	0.3 ± 0.14	0.17 ± 0.1
Pore Roundness	0.54 ± 0.19	0.57 ± 0.16
Pore Equivalent Diameter (mean)	140.8 µm	53.6 µm
Pore Equivalent Diameter (mode)	6.3 µm	6.3 µm
Pore Equivalent Diameter (min)	6.3 µm	6.3 µm
Pore Equivalent Diameter (max)	749.5 µm	739.5 µm

Table 5.3 Measured values of the shape descriptor parameters obtained by image processing of the CARS and TPEF 3D reconstructions of the fluorophore stained KBC2000 scaffold.

The shape descriptors extracted from the CARS and TPEF images show different values above all for such regards the average number of pores per acquired slice, the pore circularity and pore equivalent diameter. The differences between the images acquired with the two techniques could be related probably to the different collector lenses used (CARS signal was detected with a lower NA objective), the excitation and generated signals attenuation after passing through the material and the wavefront distortion that could affect the CARS phase matching [140,141]. These phenomena led to different segmentation profiles increasing the number of pores per acquired slice in the TPEF images with respect to the CARS ones and reducing the average equivalent diameter in the TPEF images with respect to the CARS ones.

To measure the volume occupied by the scaffold and the volume occupied by a culture medium using the chemical contrast of CARS microscopy, a thin slab of K-BC2000 scaffold immersed in culture medium was also imaged tuning the excitation sources to visualize the scaffold at about 2920 cm^{-1} and the culture medium at about 3175 cm^{-1}. A water immersion 60x objective (LUMPLFLN 60XW NA=1, W.D.=2mm, Olympus) was used to focus excitation sources on the sample, while a 20x objective (UPLSAPO 20x objective NA=0.75, W.D.=0.6 mm, Olympus) was used to collect CARS signal in forward detection. Two sets of Z stack images related to the scaffold and the water solution respectively were acquired in succession and then merged in a RGB Z stack using the red channel for the scaffold and the blue channel for the water solution.

A volume of about 235x235x175 μm^3 was reconstructed and visualized using the *ImageJ* plug-ins *3D viewer*. In Fig. 5.23 shows this 3D reconstruction and it demonstrates that regions of culture medium just under the scaffold are not displayed correctly. These artifacts could be due to attenuated transmission of the excitation source and/or to a wavefront distortion of the sources after going through the scaffold.

Although it has been shown that multimodal microscopy creates some artifacts that should be taken into account during scaffold characterization, this kind of microscope gives the chance to image without

damages living samples in time-lapse experiment. This exceptional feature could be used to investigate possible interferences between the scaffold morphology and the cellular behaviors, improving the understanding of the interactions between cells and scaffold.

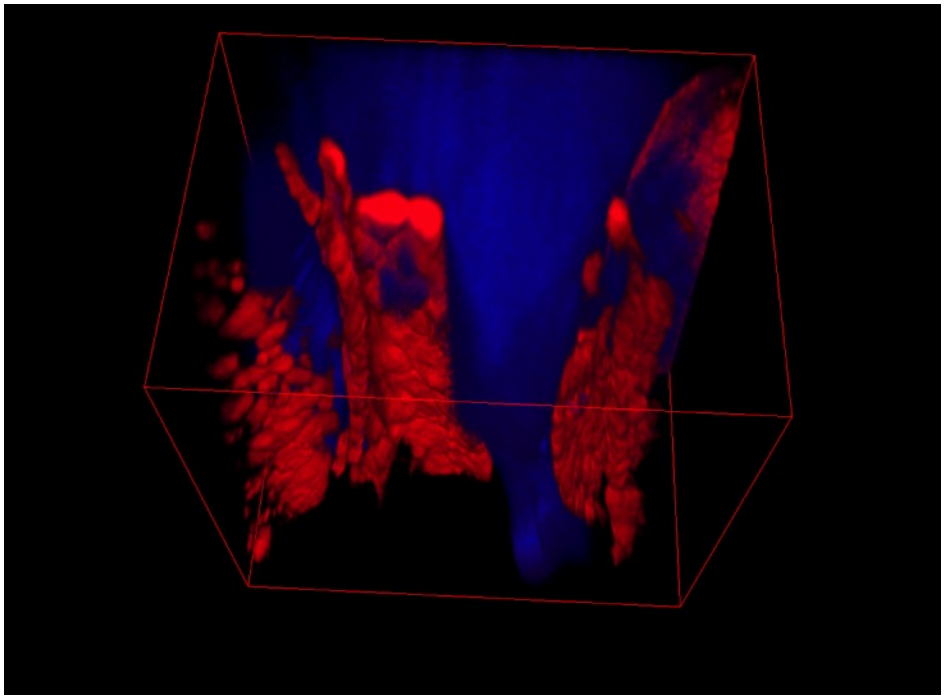

Fig. 5.23 Three-dimensional view of the KBC2000 scaffold (in red) immersed in culture medium (in blue) acquired using CARS microscopy.

To show the feasibility of this kind of experiments using multimodal CARS/SHG/TPEF microscopy, a time-lapse experiment during two days of culture of human mesenchymal stem cells (hMSCs) on a PHBHV-gel scaffold has been conducted. In collaboration with the department of clinical science of UNITO (group of C. Giachino), hMSCs have been seeded on a PHBHV-gel scaffold and stained with green calcein AM in our lab. After one day in culture, a 48 hours time-lapse experiment has been done measuring part of the scaffold, every 30 minutes during the first 12 hours, and every hour for the remaining time.
PHBHV-gel scaffold was imaged using CARS at 2936cm^{-1} in forward detection, while hMSCs were imaged simultaneously using TPEF in epi-detection. A water immersion 60x objective (LUMPLFLN 60XW NA=1, W.D.=2mm, Olympus) was used to focus excitation sources on the sample and collect the back propagating TPEF signal, while a 20x objective (UPLSAPO 20x objective NA=0.75, W.D.=0.6 mm, Olympus) was used to collect CARS signal in forward detection. The cells seeded on the scaffold were immersed in culture medium inside a Petri dish and placed in the custom-made incubator that kept the dish temperature at around 37° C. Culture medium together with calcein AM was added each 12 hours, to compensate evaporation and fluorophore bleaching estimated at about 100 µl per hour, using the intravenous cannula piercing the lattice. In each measurement a stack of 66 images of 512x512 pixels Kalman averaged 2 times with a Z step of 1 µm were collected in around six minutes. At the end of the time-lapse experiment all the Z stacks were converted in a 3D view using *3D viewer imageJ* plug-ins. All these 3D views related to different periods of the time-lapse experiment were converted in images and thus in a frames succession in order to have a global vision of the cells on the scaffold dynamic, part of them are displayed in Fig. 5.24.

It is possible to observe that hMSCs preferred to lie inside the channels of the micro-structured PHVHV-gel scaffold, although they still had a pronounced mobility. In this 48 hours experiments some cells proliferated while other disappeared probably due to apoptosis processes. It is clearly understandable the cell positions with respect to the scaffold can be clearly identified during the full time-lapse experiment and no cells seem to be under the scaffold.

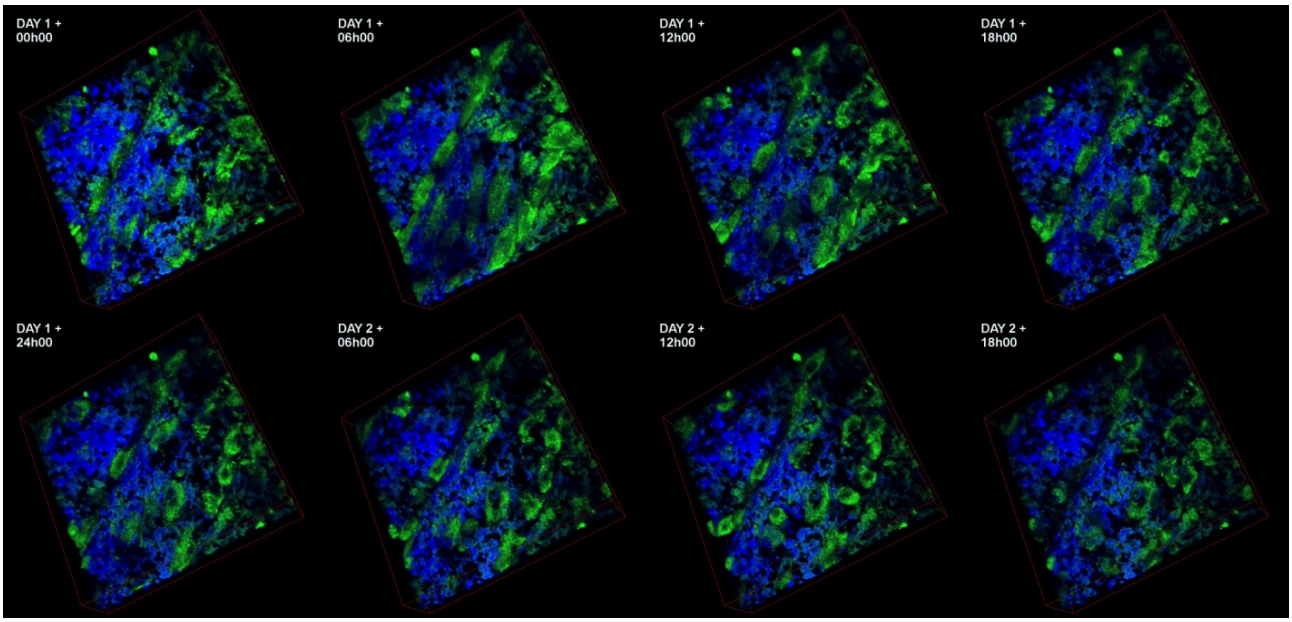

Fig. 5.24 Three-dimensional views representing the PHBHV-gel scaffold (in blue) and the hMSCs stained with calcein AM (in green) acquired using respectively CARS and TPEF microscopy techniques during a two days time-lapse experiment. In the figure the sample measurements are displayed with a time frame of six hours from each other.

Conclusion

Multimodal CARS microscopy has been demonstrated to be a useful tool for high 3D resolution scaffolds characterization. Multimodal microscopy offers observation of live cells behavior on the scaffold itself. This allows a better understanding of what scaffold properties can influence cells behavior and how they do influence the cells. This information can be fed back to improve the scaffold. However, measurement artifacts that can affect 3D scaffold imaging have been observed. Techniques to correct (all or part of) these artifacts are to be addressed and require further study.

This work has been submitted for publication and presented in the European conference on nonlinear optics spectroscopy ECONOS [148] and the microCARS2012 international workshop [149].

Acknowledgments

This work was partially supported by the Regione Piemonte under the programs CIPE 2007- converging technologies, grant 0126000010-METREGEN and POR-FESR I-I.1.3-I1.1 - ACTIVE.
Special acknowledgements to Prof. G. Ciardelli and his group (Polytechnic of Torino, Italy) and to Dr. C. Cristallini and her group (Institute for Composite and Biomedical Materials, National Research Council, Pisa, Italy) for providing the scaffold structures, to Prof. C. Giachino and her group (University of Torino, Italy) for in-situ cell cultures, to C. Divieto (INRIM) for helpful discussions and ideas in projecting the mini-incubator, to M. Franco (INRIM) for its mechanical realization, to Dr. M. Pisani for the use of the SEM microscope.

6. Conclusions

In this thesis has been developed and realized an advanced multimodal CARS-SHG-TPEF microscope able to perform 3D imaging of living samples with low invasiveness.

A detailed description of the experimental setup has been reported, analyzing the best configurations of the adopted optical filters according to the spectral region of interest during the measurements.

Part of the work has been dedicated to the software development in order to control the measurement tools like the optical spectrum analyzer and the OPO excitation wavelengths, to perform automatic spectroscopy and microscopy measurements and enabling also remote control of the microscope using remote desktop applications.

Moreover, some physical modifications of the microscope used have been designed and realized in order to achieve also forward imaging, realizing a PMT and filters holder that reduces stray light detection and an objective lens holder with all the degrees of freedom for precise optical alignment. A particular attention has been also paid for the realization of an external aluminum shield to cover part of the optical table and the microscopy stage, in order to minimize further the stray light from the ambient during the measurements.

A metrological characterization of the microscope is described with a theoretical study of the main sources of uncertainty of the measurements, separating the aspects related to the quantification of the amount of substance, from those related to the spatial localization and the microscope spatial resolution.

A specific algorithm has been developed to process images acquired using CARS microscopy and to extrapolate equivalent diameter of the reference beads analysed. This parameter has been compared with other geometrical parameters extrapolated using the Analyze Particles tool of ImageJ. Furthermore, in this thesis is presented a study on the influence of the threshold level choice during image processing, analyzing also the ImageJ autothresholding methods in function of beads diameter and images quality.

A fundamental part of this thesis has been the realization of novel biological applications using these microscopy techniques:

- To study the collagen production from fixed histological sections of human dermal fibroblasts cultured in fibrin gel scaffold using CARS and SHG techniques
- To study the collagen production by live human fibroblasts and mesenchymal stem cells cultured in fibrin gel scaffold using CARS and SHG techniques
- To characterize polymeric scaffolds in culture media with a label-free method using CARS and SHG techniques
- To study the colonization in a two days time-lapse experiment of a polymeric scaffold by human mesenchymal stem cells stained with calcein using CARS and TPEF techniques

Some of the accomplished results have also characteristic of originality namely:

- For the first time in Italy, it has been conducted biological sample imaging using CARS microscopy and combination of CARS technique with SHG or TPEF techniques.

- The proposed method for collagen produced by live stem cells in culture detection using SHG technique as a first signal of cellular differentiation in a non-invasive and non-destructive way.
- The proposed method to use CARS microscopy to characterize scaffolds in culture medium analysing also the cells migration and colonization in a time-lapse experiment using TPEF microscopy.
- The theoretical study of the main sources of uncertainty in the measurements with CARS, TPEF and SHG techniques.

Future Works

Further modifications in the experimental setup will allow also the implementation of additional NLO techniques complementary to the CARS-SHG-TPEF microscopy techniques already available. Interference CARS detection scheme together with the integration of Stimulated Raman Scattering (SRS) have been designed and as soon as there will be the availability of all the required components, they will be implemented to the existent setup, enriching the capabilities of this microscopy system.

7.Acknowledgements

I want to deeply acknowledge Maria Paola Sassi for the huge help given to me in the supervision of the thesis work.

I am grateful to the colleagues that I met during my participation in the international conferences and workshops, giving to me a helpful background and expertise for the realization of my thesis.

Special acknowledgements to Gary Morley from LGC (UK) for providing the collagen samples and the hDFs histological sections used in this work.

Special acknowledgements to Prof. Gianluca Ciardelli and his group (Polytechnic of Torino, Italy) and to Dr. Caterina Cristallini and her group (Institute for Composite and Biomedical Materials, National Research Council, Pisa, Italy) for providing the polymeric scaffold structures, to Prof. Claudia Giachino and her group (University of Torino, Italy) for in-situ cell cultures on polymeric scaffolds, to Dr. Sizzano of the Transplants Centre of the Regione Piemonte for providing the hCFs used in this work.

I want to acknowledge Mauro Franco for his great work in the realization of all the plastic and metallic parts that I designed in this work.

I want to thank Dr. Marco Pisani for his help and the chance to use the SEM microscope.

A big thanks to my colleagues of the INRIM group “Metrology for bioscience and trace substances” for their support in these years. In particular a special thanks to Massimo Artiglia for the great help during the realization of the first CARS experimental setups. A special thanks to Carla Divieto for providing the biological samples used during the realization of the microscopy experiments. I would like to thank Ettore Bernardi for the characterization of the 5 µm beads using the AFM, Gianni Intermite for the help given to me for the TPEF microscopy setup, Alessia Demichelis for the interesting scientific discussions and Chaiwat Prawettongsopon for providing the samples for hMSCs adhesion study on different substrates using TPEF microscopy.

I want to thank in a very special way my family and all my friends for their personal support.

8. References

[1] Marriott, J., O'Conner, G., Parkes, H. "Final Report: Study on Measurement Services and Comparison Needs for an International Measurement Infrastructure for the Biosciences and Biotechnology: Input for the BIPM Work Programme", Rapport BIPM-2011/02, Reprot Number: LGC/R2011/123 Number, BIPM. (2011).

[2] Sang-Ryoul Park, Jun-Hyuk Choi and Ji-Seon Jeong "Development of Metrology for Modern Biology, Modern Metrology Concerns", Dr. Luigi Cocco (Ed.), ISBN: 978-953-51-0584-8, InTech, DOI:10.5772/37367. (2012). Available from: http://www.intechopen.com/books/modern-metrology-concerns/development-of-metrology-for-modern-biology

[3] Y. Ozeki, F. Dake, S. I. Kajiyama, K. Fukui, and K. Itoh, *Opt. Express* 17, 3651–3658 (2009).

[4] P. Nandakumar, A. Kovalev, and A. Volkmer, *New J. Phys.* 11, 033026 (2009).

[5] C. W. Freudiger, W. Min, B. G. Saar, S. Lu, G. R. Holtom, C. He, J. C. Tsai, J. X. Kang, and X. S. Xie, *Science* 322, 1857–1861 (2008).

[6] Shen Y.R., "*The Principles of Nonlinear Optics"*, John Wiley & Sons, New York (1984).

[7] Cheng J.X., Volkmer A. and Xie X. S., *J. Opt. Soc. Am. B*, Vol. 19, No. 6, pp 1363-1375 (2002).

[8] Volkmer A., Cheng J.X. and Xie X.S., *Phys. Rev. Lett.,* Vol. 87, no. 2, art. 023901 (2001).

[9] W. Denk, J. Strickler, W. Webb, *Science*, New Series, Vol. 248, No.4951., pp. 73-76. (apr. 6, 1990).

[10] M. Göppert-Mayer, Ann. Phys. (Paris), Vol. 9, pp. 273-295. (1931).

[11] C. Xu, "Cross-sections of fluorescence molecules in multiphoton microscopy". In: "Confocal and two-photon microscopy: foundations, applications, and advances." A. Diaspro, editor. New York: WileyLiss (2002).

[12] M. Oheim et al., *Advanced Drug Delivery Reviews,* Vol. 58, pp. 788-808. (2006).

[13] P. E. Hänninen, M. Schrader, E. Soini and S. W. Hell, *Bioimaging,* Vol. 3, pp. 70-75. (1995).

[14] J. Bewersdorf and S.W. Hell, *J. Microsc.,* Vol.191 pp. 28-38. (1998).

[15] Y. Liu et al, *Biophys. J.,* Vol. 68, pp. 2137-2144. (1995).

[16] A. Schönle et al, *Opt. Lett.,* Vol. 23, pp. 325-327. (1998).

[17] P. A. Franken et al., *Phys. Rev. Lett.* 7 (4), 118 (1961).

[18] Kleinman, D. A., *Phys. Rev.* 126:1977–1979. (1962).

[19] Bloembergen, N., R. K. Chang, S. S. Jha, and C. H. Lee., *Phys. Rev.* 174:19813–19822. (1968).

[20] Hellwarth, R., and P. Christensen. *Optics Comm.* 12: 318 –322. (1974).

[21] I. Freund, M. Deutsch, and A. Sprecher, *Biophys. J.* 50, 693–712 (1986).

[22] P. J. Campagnola, A. C. Millard, M. Terasaki, P. E. Hoppe, C. J. Malone, W. A. Mohler, *Biophys. J.,* 81, 493–508. (2002)

[23] P. J. Campagnola, H. A. Clark, W. A. Mohler, A. Lewis, L. M. Loew, *J. Biomed. Opt.* 6(3), 277–286. (2001).

[24] M. D. Duncan, J. Reintjes, and T. J. Manuccia, *Opt. Lett.* 7, 350 (1982).

[25] A. Zumbusch, G. R. Holtom, and X. S. Xie, *Phys. Rev. Lett.* 82, 4142 (1999).

[26] R. A. Baumgartner and R. L. Byer, *IEEE J. Quantum Electron.* QE-15 (6), 432 (1979).

[27] J. X. Cheng, A. Volkmer, L. D. Book, X. S. Xie, *J. Phys. Chem. B* 105, 1278-1280 (2001)

[28] J. X. Cheng, Y. K. Jia, G. Zheng, X. S. Xie, *Biophysical J.* 83, 502-509 (2002).

[29] L. Lefort, K. Puech, S. D. Butterworth, Y. P. Svirko, and D. C. Hanna, *Opt. Lett.* 24 n. 1, 28-30 (1999).
[30] T. W. Kee, H. Zhao, and M. T. Cicerone, *Optics Express* 14 n. 8, 3631-3640 (2006).
[31] S. Maeda, T. Kamisuki, Y. Adachi, In "Advances in Nonlinear Spectroscopy"; R. J. H. Clark, R. E. Hester; Ed. John Wiley and Sons Ltd.: New York, (1988); p 253.
[32] J. X. Cheng, L. D. Book, X. S. Xie, *Opt. Lett.* 26, 1341 (2001).
[33] S. A. Akhmanov, A. F. Bunkin, S. G. Ivanov, and N. I. Koroteev, *JETP Lett.* 25, 416 (1977).
[34] A. Volkmer, L. D. Book, X. S. Xie, *Appl. Phys. Lett.* Vol 80 n. 9 1505-1507 (2002)
[35] E. Potma, C. L. Evans, X. S. Xie, *Opt. Lett.* vol. 31 n. 2, 241-243 (2006).
[36] M. Jurna, J.P. Korterik, C. Otto , J.L. Herek, and H.L. Offerhaus, *Optics Express* Vol. 16, No. 20, 15863-15869. (2008).
[37] M. Jurna, J.P. Korterik, C. Otto, and H.L. Offerhaus, *Optics Express*, Vol. 15, No. 23 15207-15213. (2007).
[38] J. X. Cheng, A. Volkmer, L. D. Book, X. S. Xie, *J. Phys. Chem. B* 106, 8493-8498 (2002).
[39] B. D. Boyan, C. H. Lohmann, D. D. Dean, V. L. Sylvia, D. L. Cochran, Z. Schwartz, *Ann Rev Mater Res* 31: 357-71. (2001).
[40] N. R. Washburn, K. M. Yamada, C. G. Jr Simon, S. B. Kennedy, E. J. Amis, *Biomaterials* 25(7-8): 1215-24. (2004).
[41] T. P. Kunzler, T. Drobek, M. Schuler, N. D. Spencer, *Biomaterials* 28(13): 2175-82. (2007).
[42] A. J. Engler, S. Sen, H. L. Sweeney, D. E. Discher, *Cell* 126(4): 677-89. (2006).
[43] H. L. Holtorf, N. Datta, J. A. Jansen, A. G. Mikos, *J Biomed Mater Res A* 74(2): 171-80. (2005).
[44] B.D. Boyan, L.F. Bonewald, E.P. Paschalis, C.H. Lohmann, J. Rosser, D.L. Cochran, D.D. Dean, Z. Schwartz, A.L. Boskey, *Calcif Tissue Int* 71(6): 519-29. (2002).
[45] F. Rehfeldt, A. J. Engler, A. Eckhardt, F. Ahmed, D. E. Discher, *Adv Drug Deliv Rev* 59(13): 1329-39 (2007).
[46] D.E. Discher, P. Janmey, Y.L. Wang, *Science* 310(5751): 1139-43. (2005).
[47] V. Karageorgiou, D. Kaplan, *Biomaterials* 26(27): 5474-91. (2005).
[48] M. C. Wake, C. W. Jr. Patrick, A. G. Mikos, *Cell Transplant* 3(4): 339-43. (1994).
[49] T. P. Kunzler, C. Huwiler, T. Drobek, J. Voros, N. D. Spencer, *Biomaterials* 28(33): 5000-6. (2007).
[50] J. Zeltinger, J. K. Sherwood, D. A. Graham, R. Mueller, L. G. Griffith, *Tissue Eng* 7(5): 557-72. (2001).
[51] M. L. Mather, S. P. Morgan, L. J. White, H. Tai, W. Kockenberger, S. M. Howdle, K. M. Shakesheff and J. A. Crowe, *Biomed. Mater.* 3 015011 (2008). doi:10.1088/1748-6041/3/1/015011.
[52] S. T. Hoa, D. W. Hutmacher, *Biomaterials* 27 1362-1376 (2006).
[53] B. Otsuki, M. Takemoto, S. Fujibayashi, M. Neo, T. Kokubo, T. Nakamura, *Biomaterials* 27 5892-5900 (2006).
[54] A. L. Darling, and W. Sun, *J. Biomed. Mater. Res.* 70B: 311-317. doi: 10.1002/jbm.b.30050. (2004).
[55] H. Yoshimotoa, Y.M. Shina, H. Teraia, J.P. Vacantia, *Biomaterials* 24 2077-2082 (2003).
[56] Y. Wang, L. Liu, S. Guo, *Polymer Degradation and Stability* 95 207-213 (2010).
[57] S. Sell, C. Barnes, D. Simpson, G. Bowlin, *J. Biomed. Mater. Res.* 85A: 115-126. (2008). doi: 10.1002/jbm.a.31556
[58] F. J. O'Brien, B. A. Harley, M.A. Waller, I.V. Yannas, L. J. Gibson, P. J. Prendergast, *Journal Technology and Health Care*, Volume 15, Number 1, 3-17 (2007).
[59] S. Li, J. R. de Wijn, J. Li, P. Layrolle, K. de Groot. *Tissue Engineering.* 9(3): 535-548. (2003). doi:10.1089/107632703322066714.

[60] Y. Wana, X. Qua, J. Lub, C. Zhub, L. Wanb, J. Yanga, J. Beia, S. Wanga, *Biomaterials* 25 4777-4783 (2004).
[61] D. M. Doroski, K. S. Brink, J. S. Temenoff, *Biomaterials* 28 187-202 (2007).
[62] C. Vitale-Brovarone, E. Verné, L. Robiglio, P. Appendino, F. Bassi, G. Martinasso, G. Muzio, R. Canuto, *Acta Biomaterialia*, 3(2): 199-208, (2007) doi:10.1016/j.actbio.2006.07.012.
[63] W. Kolch, *Nature Reviews Molecular Cell Biology*, 6 827-837, (2005). doi:10.1038/nrm1743
[64] W.-J. Li, K. G. Danielson, P. G. Alexander, R. S. Tuan, *J. Biomed. Mater. Res.*, 67A: 1105–1114 (2003). doi: 10.1002/jbm.a.10101
[65] B. A.C. Harley, H.-D. Kim, M. H. Zaman, I. V. Yannas, D. A. Lauffenburger, L. J. Gibson, *Biophysical Journal*, 15 95(8) 4013-4024 (2008). doi: 10.1529/biophysj.107.122598.
[66] A. Horii, X. Wang, F. Gelain, S. Zhang, *PLoS ONE*, 2(2): e190. (2007). doi:10.1371/journal.pone.0000190
[67] Warren R. Zipfel, Rebecca M. Williams, Richard Christie, Alexander Yu Nikitin, Bradley T. Hyman and Watt W. Webb, *PNAS* - - vol. 100 - no. 12 - 7075–7080 (2003).
[68] Li Li, Haifeng Wang, and Ji-Xin Cheng - *Biophysical Journal* Volume 89 3480–3490 (2005).
[69] Erik M. Vartiainen, Hilde A. Rinia, Michiel Müller, Mischa Bonn - *Optics Express* Vol. 14, No. 8 3622 (2006).
[70] Richard A. Meyer, *Applied Optics* Vol. 18, No. 5, 585-588 (1979).
[71] Sog Nke Johnsen And Edith A. Widder, *J. Theor. Biol.* 199, 181-198 (1999).
[72] Yifeng Zhou, Kenny K. H. Chan, Tom Lai, and Shuo Tang, *Biomedical Optics Express* 38, Vol. 4, No. 1 (2013).
[73] Huafeng Ding, Jun Q Lu, William A Wooden, Peter J Kragel and Xin-Hua Hu, *Phys. Med. Biol.* 51 1479–1489 (2006).
[74] N. Otsu, *IEEE Trans. Sys., Man., Cyber.* 9 (1): 62–66 (1979). doi:10.1109/TSMC.1979.4310076.
[75] W.H. Walton, *Nature* 162, 329-330 (1948).
[76] http://fiji.sc/wiki/index.php/Auto_Threshold
[77] http://www.kitware.com/source/home/post/54
[78] Huang, L-K & Wang, M-J J, *Pattern Recognition* 28(1): 41-51 (1995).
[79] Prewitt, JMS & Mendelsohn, ML, *Annals of the New York Academy of Sciences* 128: 1035-1053 (1966).
[80] Ridler, TW & Calvard, S, *Man and Cybernetics* 8: 630-632 (1978).
[81] Li, CH & Lee, CK, *Pattern Recognition* 26(4): 617-625, (1993).
[82] Li CH, Tam PKS, *Pattern Recognition Letters* 18(8): 771-776, (1998).
[83] Sezgin, M & Sankur, B, *Journal of Electronic Imaging* 13(1): 146-165 (2004).
[84] Kapur, JN; Sahoo, PK & Wong, ACK, *Graphical Models and Image Processing* 29(3): 273-285 (1985).
[85] Glasbey, CA, *CVGIP: Graphical Models and Image Processing* 55: 532-537 (1993).
[86] Kittler, J & Illingworth, J, *Pattern Recognition* 19: 41-47 (1986).
[87] Tsai, W, *Computer Vision, Graphics, and Image Processing* 29: 377-393 (1985).
[88] Doyle, W, *Journal of the Association for Computing Machinery,* 9259-267, (1962). doi:10.1145/321119.321123.
[89] Shanbhag, Abhijit G., *Graph. Models Image Process.* (Academic Press, Inc.) 56 (5): 414--419, (1994). ISSN 1049-9652
[90] Zack GW, Rogers WE, Latt SA, *J. Histochem. Cytochem.* 25 (7): 741–53, (1977). PMID 70454.
[91] Yen JC, Chang FJ, Chang S, *IEEE Trans. on Image Processing* 4 (3): 370-378, (1995). ISSN 1057-7149, doi:10.1109/83.366472.
[92] NIST Certificate of Analysis 1690 (1996).
[93] NIST Certificate of Analysis 1692 (1991).
[94] Xiaolin Nan, Ji-Xin Cheng, and X. Sunney Xie *Lipid Res.* 44: 2202–2208 (2003).
[95] J. F. Burke, I. V. Yannas, W. C. Quinby Jr, C. C. Bondoc, W. K. Jung, *Ann. Surg.* 194, 413–428 (1981).
[96] C. Edwards, R. Marks, *Clin. Dermatol.* 13, 375–380 (1995).

[97] H. M. Powell, S. T. Boyce, *Biomaterials* 27, 5821–5827 (2006).
[98] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, J. D. Watson, *"Molecular Biology of the Cell"*, Garland Science, New York, (1994).
[99] A.L. Mazlyzam, B.S. Aminuddin, N.H. Fuzina, M.M. Norhayati, O. Fauziah, M.R. Isa, L. Saim, B.H.I. Ruszymah, *Burns* 33, 355-363 (2007).
[100] A. I. Caplan, *J. Cell. Physiol.* 213: 341-347 (2007).
[101] R. G. M. Bruels, T. U. Jiya, T. H. Smit, *The Open Othopeadic Journal.* 2, 103 –109 (2008).
[102] H. Duong, B. Wu, B. Tawil, *Tissue Engineering: Part A.* 15 (7): 1865-1876 (2009).
[103] W. P. Daley, S. B. Peters, M. Larsen, *Journal of Cell Science.* 121: 255-264, (2007).
[104] M.F. Pittenger, A.M. Mackay, S.C. Beck, R.K. Jaiswal, R. Douglas, J.D. Mosca, M.A. Moorman, D.W. Simonetti, S. Craig, D.R. Marshak, *Science.* 284:143-147 (1999).
[105] D. Pelaez, C. C. Huang, H. S. Cheung, *Stem Cells Dev.* 18(1), 93–102 (2009).
[106] R. Hass, C. Kasper, S. Bohm, R. Jacobs, *Cell Communication and Signaling.* 9 ,12– 26 (2011).
[107] M. Jager, T. Feser, H. Denck, R. Krauspe, *Ann. Biomed. Eng.* 33, 1319 –1332 (2005).
[108] R. T. Franceschi, B. S. Iyer, *J. Bone Miner. Res.* 7 (2), 235– 246, (1992).
[109] L. A. Solchaga, K. J. Penik, J. F. Welter, *Methods Mol. Biol.* 698, 253 –278 (2011).
[110] Y. Zhou, X. Guan, Z. Zhu, S. Gao, C. Zhang, C. Li, K. Zhou, W. Hou, H. Yu, *Eur. Cell. Mater.* 22 , 12 – 25 (2011).
[111] J. George, Y. Kuboki, T. Miyata, *Biotechnology and Bioengineering.* 95 (3): 404-411 (2006).
[112] G. Cox, E. Kable, A. Jones, I. Fraser, F. Manconi, M. D. Gorrellc, *Journal of Structural Biology*. 141:53-62 (2003).
[113] P. A. Janmey, J. P. Winer, J. W. Weisel, *J. R. Soc. Interface.*; 6: 1-10 (2009).
[114] F. Djouad, B. Delorme, M. Maurice, C. Bony, F. Apparailly, P. Louis-Plence, F. Canovas, P. Charbord, D. Noël, C. Jorgensen, *Arthritis Research & Therapy.* 9 (2), (2007).
[115] G. Chamberlain, J. Fox, B. Ashton, J. Middleton, *Stem Cells*, 25, 2739–2749, (2007).
[116] I.Catelas, N. Sese, B.M. Wu, J.C.Y. Dunn, S. Helgerson, B.Tawil, *Tissue Engineering*. 12(8): 2385-2396 (2006).
[117] T. Nishida, A. Ueda, M. Fukuda, H. Mishina, K. Yasumoto, T. Otori, *In Vitro Cellular & Development Biology.* 24(10): 1009-1014 (1988).
[118] W.R. Zipfel, R.M. Williams, W.W. Webb, *Nature Biotechnology.* 21(11) (2003).
[119] A. Zoumi, A. Yeh, B. J. Tromberg, *PNAS*. 99(17): 11014-11019 (2002).
[120] P.Stoller, P.M. Celliers, K.M. Reiser, A.M. Rubenchik, *Applied Optics*. 42 (25): 5209-5219. (2003).
[121] J. Chen, A. Lee, J. Zhao, H. Wang, H. Lui, D. I. McLean, H. Zeng, *Skin Research and Technology*. 15: 418-426. (2009).
[122] T. Luo, J. X. Chen, S. M. Zhuo, K. C. Lu, X. S. Jiang, Q. G. Liu *Laser Physics.* 19 (3): 478-482. (2009).
[123] R. M. Williams, W. R. Zipfel, W.W. Webb *Biophysical Journal.* 88: 1377-1386 (2005).
[124] P. Su, W. Chen, T. Li, C. Chou, T. Chen, Y. Ho, C. Huang, S. Chang, Y. Huang, H. Lee, C. Dong, *Biomaterials.* 31(36):9415-21 (2010).
[125] S. V. Plotnikov, A. C. Millard, P. J. Campagnola, W. A. Mohler, *Biophys. J.* 90(2):693-703. (2006).
[126] C.P. Pfeffer, B. R. Olsen, F. Ganikhanov, F. Légaré, *J Struct Biol.* 164(1): 140-145 (2008).
[127] C.Brackmann, A. Bodin, M. Åkeson, P. Gatenholm, A. Enejder, *Biomacromolecules.* 11: 542–548. (2010).
[128] A. D. Slepkov, A. Ridsdale, A. F. Pegoraro, D. J. Moffatt, A.Stolow, *Biomedical Optics Express.* 1(5):1347-1357 (2010).
[129] R. S. Lim, A. Kratzer, N. P. Barry, S. Miyazaki-Anzai, M. Miyazaki, W. W. Mantulin, M.Levi, E. O. Potma, B. J. Tromberg, *J. Lipid Res.* 51: 1729-1737 (2010).
[130] A. C. T. Ko, A. Ridsdale, M.S. D. Smith, L. B. Mostaço-Guidolin, M. D. Hewko, A. F. Pegoraro, E. K. Kohlenberg, B. Schattka, M. Shiomi, A. Stolow, M. G. Sowa, *Journal of Biomedical Optics.* 15(2). (2010).
[131] F. Gelain, D. Bottai, A. Vescovi, S. Zhang, *PLoS ONE* 1(1): e119 (2006). doi:10.1371/journal.pone.0000119

[132] M. L. Mather, S. P. Morgan, L. J. White, H. Tai, W. Kockenberger, S. M. Howdle, K. M. Shakesheff and J. A. Crowe, *Biomed. Mater.* 3 015011 (2008). doi:10.1088/1748-6041/3/1/015011
[133] V. Chiono, E. Descrovi, S. Sartori, P. Gentile, M. Ballarini, F. Giorgis, G. Ciardelli, *NanoScience and Technology* Part 3, 645-689 (2011).
[134] C. Cristallini, M. Gagliardi, N. Barbani, D. Giannessi, G. Guerra, *Journal of Materials Science: Materials in Medicine* 23: 205-216 (2012).
[135] I. V. Yannas, E. Lee, D. P. Orgill, E. M. Skrabut, G. F. Murphy, *Proc. Natl Acad. Sci. USA* 86, 933–937 (1989). (doi:10.1073/pnas.86.3.933)
[136] A. S. Mistry, A. G. Mikos, *Adv. Biochem. Eng. Biotechnol.* 94, 1–22. (2005). doi:10.1007/b99997
[137] B. Kinner, R. M. Capito, M. Spector, *Adv. Biochem. Eng. Biotechnol.* 94, 91–123 (2005). doi:10.1007/b100001
[138] E. Rabkin-Aikawa, J. E. Mayer, F. J. Schoen, *Adv. Biochem. Eng. Biotechnol.* 94, 141–178 (2005). (doi:10.1007/b100003)
[139] N. Davidenko, J. J. Campbell, E. S. Thian, C. J. Watson, R. E. Cameron, *Acta Biomater.* 6(10):12 (2010).
[140] E. Chaigneau, A. J. Wright, S. P. Poland, J. M. Girkin, R. Angus Silver, *Optics Express* Vol. 19(23) 22755-22774 (2011). doi:10.1364/OE.19.022755
[141] J. Zeng, P. Mahou, M.-C. Schanne-Klein, E. Beaurepaire, D. Débarre , *Biomedical Optics Express* 3(8) 1898-1913 (2012). doi: 10.1364/BOE.3.001898
[142] L. Mortati, M. P. Sassi, "CARS microscopy of polystyrene materials", Book of Abstract of 466th WE-Heraus-Seminar Coherent Raman Scattering Microscopy microCARS2010, 18-20th October 2010, Bad Honnef, Germania
[143] M. P. Sassi, L. Mortati "Chemical selective in vitro bioassay for cells/EC matrix detection in 3D nonlabeled cell cultures", *Regenerative Medicine*, Nov 2011, Vol. 6, No. 6s2, Pages 37-181.
[144] L. Mortati, C. Divieto, M. P. Sassi, *J. Raman Spectrosc.*, 43: 675–680. (2012) doi: 10.1002/jrs.3171
[145] L. Mortati, C. Divieto, M. P. Sassi, "Label-free imaging of collagen extracellular matrix and fibroblast cells cultured on 3D scaffolds", Book of Abstract of ECONOS 2011, May 23-25th 2011, Twente, Paesi Bassi
[146] L. Mortati, C. Divieto, M. P. Sassi, "Detection of collagen produced by live human corneal fibroblasts and human mesenchymal stem cells cultured in 3D fibrin gel using label-free noninvasive imaging", *Regenerative Medicine*, Nov 2011, Vol. 6, No. 6s2, Pages 37-181.
[147] L. Mortati, C. Divieto, M. P. Sassi, "Early stage differentiation of living human mesenchymal stem cells followed by multimodal CARS/SHG microscopy", Book of Abstract of Workshop on Applications of Coherent Raman Scattering Microscopy, 23-24th April 2012, Exeter, United Kingdom
[148] L. Mortati, M. P. Sassi, "3D architectural characterization of scaffolds in a cell culture medium using CARS microscopy", Book of abstract of ECONOS 2012, 8-11 July 2012, Aberdeen, United Kingdom
[149] L. Mortati, M. P. Sassi, "NLO multimodal microscopy as a novel tool for scaffold characterization in a 3D cell culture" ", Book of abstract of MicroCARS 2012, 14-16 Ottobre 2012, Naurod, Germany

9. List of Publications

Book of abstract

1. A. Kummrow, M.-O. Baradez, M. Shaw, **L. Mortati**, M. Frankowski, D. Metcalf, S. Pavarelli, C. Weißbach, G. Intermite, P. Tomlins, M.P. Sassi, D. Marshall and R. Macdonald, **"Assessment of measurement of cell confluency level by image analysis"**, Regenerative Medicine, Nov 2011, Vol. 6, No. 6s2, Pages 37-181.
2. **L. Mortati**, C. Divieto, M. P. Sassi, **"Detection of collagen produced by live human corneal fibroblasts and human mesenchymal stem cells cultured in 3D fibrin gel using label-free noninvasive imaging"**, Regenerative Medicine, Nov 2011, Vol. 6, No. 6s2, Pages 37-181.
3. M. P. Sassi, **L. Mortati "Chemical selective in vitro bioassay for cells/EC matrix detection in 3D nonlabeled cell cultures"**, Regenerative Medicine, Nov 2011, Vol. 6, No. 6s2, Pages 37-181.
4. M. Artiglia, L. Mortati, H. Marchelli, M.P. Sassi, **"CARS micro-spectroscopy at INRIM"**, Book of Abstract of microCARS Spring Workshop, 9-11 Maggio 2010, Göteborg, Sweden
5. L. Mortati, M.P. Sassi, **"A Metrological Study for 3D CARS Imaging of Polystyrene Spheres"**, Book of Abstract of 466th WE-Heraus-Seminar Coherent Raman Scattering Microscopy microCARS2010, Ottobre 18-20 2010, Bad Honnef, Germany
6. L. Mortati, C. Divieto, M. P. Sassi, **"Label-free imaging of collagen extracellular matrix and fibroblast cells cultured on 3D scaffolds"**, Book of Abstract of ECONOS 2011, Maggio 23-25 2011, Twente, Netherlands
7. L. Mortati, C. Divieto, M. P. Sassi, **"Early stage differentiation of living human mesenchymal stem cells followed by multimodal CARS/SHG microscopy"**, Book of Abstract of Workshop on Applications of Coherent Raman Scattering Microscopy, 23-24 aprile 2012, Exeter, United Kingdom
8. L. Mortati, M. P. Sassi, **"3D architectural characterization of scaffolds in a cell culture medium using CARS microscopy"**, Book of abstract of ECONOS 2012, 8-11 luglio 2012, Aberdeen, United Kingdom
9. L. Mortati, M. P. Sassi, **"NLO multimodal microscopy as a novel tool for scaffold characterization in a 3D cell culture"** , Book of abstract of MicroCARS 2012, 14-16 Ottobre 2012, Naurod, Germany

ISI Paper

10. L. Mortati, C. Divieto, M. P. Sassi, **"CARS and SHG microscopy to follow collagen production in living human corneal fibroblasts and mesenchymal stem cells in fibrin hydrogel 3D cultures",** J. Raman Spectrosc., 43: 675–680. (2012) doi: 10.1002/jrs.3171

INRIM Technical Report

11. L. Mortati, **"Progetto preliminare per la realizzazione di un microscopio ottico non lineare di tipo CARS (Coherent Anti-Stokes Raman Scattering)"**, Technical I.N.Ri.M. Report n°158/2008, Torino (Italia)
12. M.P. Sassi, M, Artiglia, L. Mortati, **"Report on the CARS facility (Coherent Anti-Stokes**

Raman Scattering Non-linear Microscope) and initial results analysis", Technical I.N.Ri.M. Report n°23/2009, Torino (Italia)

13. M. Artiglia, L. Mortati, **"Experiments of Coherent Anti-Stokes Raman Scattering Spectroscopy on polystyrene samples: experimental set-up and preliminary results"**, Technical I.N.Ri.M. Report n°22/2010 Torino (Italy)
14. L. Mortati, G. Intermite, **"Realization of 2 Photons Excitation Microscope for Metrology on a Cellular Scale"**, Technical I.N.RI.M. Report n°32/2010, Torino (Italia)

10. APPENDIX

findDiameterAll_.java

```
import ij.*;
import ij.process.*;
import ij.measure.*;
import ij.gui.*;
import java.awt.*;
import java.math.*;
import java.io.*;
import java.lang.Number;
import java.lang.Byte;
import java.lang.Short;
import java.lang.Double;
import java.lang.Boolean;
import ij.plugin.*;
import ij.plugin.filter.*;
import ij.plugin.filter.GaussianBlur.*;
import ij.plugin.frame.*;
import ij.plugin.filter.PlugInFilter;
import ij.text.TextPanel;
import java.util.*;
import java.lang.Object.*;
import java.lang.reflect.Array;
import ij.plugin.filter.ParticleAnalyzer;
import ij.measure.Measurements;
import ij.plugin.filter.Analyzer;
import ij.measure.ResultsTable;


public class findDiameterAll_ implements PlugInFilter {

    protected ImageStack stack;

    public int setup(String arg, ImagePlus imp) {
        stack=imp.getStack();
        if (arg.equals("about"))
            { return DONE; }
        return DOES_8G+STACK_REQUIRED+SUPPORTS_MASKING;
    }

    public void run(ImageProcessor ipp) {

         GenericDialog gd = new GenericDialog("Find Diameter");

        double Zsc=0.3,XYsc=0.115089;
        int psz=100;


        gd.addNumericField("Particle Size Filter: ", psz, 0);

        gd.addNumericField("XY scale (pixel*um): ", XYsc, 6);
        gd.addNumericField("Z stack scale (stack*um): ", Zsc, 3);

        gd.showDialog();
        if (gd.wasCanceled()) return;
```

```
psz = (int)gd.getNextNumber();

XYsc = (double)gd.getNextNumber();
Zsc = (double)gd.getNextNumber();


ImagePlus ip =  new ImagePlus("Slices",stack);

ImageStack is= new ImageStack();

is = ip.getStack();

ArrayList <Double> med = new ArrayList<Double>();

ImageProcessor nip;
for(int i=1;i<=is.getSize();i++){
    nip = is.getProcessor(i);
    int width = nip.getWidth();
    int height = nip.getHeight();
    int c=0;
    Double m= new Double(0);

    for(int x=0;x<width;x++){
        for(int y=0;y<height;y++){
        m=m+nip.get(x,y);
        c++;
        }
    }
    m=m/c;
    med.add((Double)m);


}



double max=0;
int id=0;

ResultsTable rt2 = ResultsTable.getResultsTable();
rt2.reset();

rt2.incrementCounter();

ResultsTable rt = ResultsTable.getResultsTable();
rt.reset();

rt.incrementCounter();
 for(int i=0;i<med.size();i++){
    Double m=(Double) med.get(i);
    if(max<=m) {
        max=m;
        id=i+1;
    }



int th[];
```

```
        int ths=10;
        nip = is.getProcessor(id);
        nip.setAutoThreshold("Otsu");
        ths=nip.getAutoThreshold();



        nip = is.getProcessor(1);

        ImageStack nis = new ImageStack(nip.getWidth(),nip.getHeight());
        ImageStack nis2 = new ImageStack(nip.getWidth(),nip.getHeight());
        ImageStack nis3 = new ImageStack(nip.getWidth(),nip.getHeight());

        rt.setPrecision(6);
        rt.addValue("XY Pixel size",XYsc);

        rt.addValue("Z step size",Zsc);

        rt.incrementCounter();

        ArrayList <ArrayList> ar = new ArrayList<ArrayList>();
        ArrayList <ArrayList> arf = new ArrayList<ArrayList>();

        ImageProcessor nip3;
        nip3=nip.duplicate();
        int idx=0,f=0,c=0;


        for(i=1;i<=is.getSize();i++){


            nip = is.getProcessor(i).duplicate();

            double t;


            nip.threshold((int)ths);
            nip.invert();
            ImageProcessor ApIp;
            ApIp=nip.duplicate();
            ImagePlus ApIpp = new ImagePlus("Analyze",ApIp);

            ParticleAnalyzer pa = new
ParticleAnalyzer(ParticleAnalyzer.SHOW_RESULTS , Measurements.RECT +
Measurements.ELLIPSE + Measurements.FERET , rt2, 20, Double.POSITIVE_INFINITY);

            pa.analyze(ApIpp);

            nip.setPixels(ParticleSizeFilter(nip,50));
            rt.addValue("th",ths);


            ImageProcessor nip2;
            nip2=nip.duplicate();
            nip2.setPixels(Edge(nip2));
            nip.setPixels(Diff(nip,nip2));

            nip.setPixels(FillHoles(nip));
            nip.setPixels(FillHoles(nip));
            nip.setPixels(FillHoles(nip));
            nip.setPixels(ParticleSizeFilter(nip,psz));
```

```
            ArrayList <posdiam> pos = new ArrayList<posdiam>();
            ArrayList <particle> prt = new ArrayList<particle>();

            nip2.setPixels(Edge(nip));
            prt=FindParticle(nip2);

            nip.setPixels(FillEdge(nip2,prt));

            nip2.setPixels(Edge(nip));
            prt=FindParticle(nip2);

            pos=FindDiam3(prt);

            nip3=nip.duplicate();
            nip3.fill();
            nip3.invert();
            int x=0,y=0;
            double dmx=0;
            idx=pos.size();
            c=0;

             for(int k=0;k<pos.size();k++){
                posdiam ps = new posdiam();
                ps=pos.get(k);
                if((ps.x!=0)&&(ps.y!=0)){

                    nip3.drawOval((int)(ps.x-ps.d/2),(int) (ps.y-
ps.d/2),(int)ps.d,(int)ps.d);
                    nip3.drawPixel(ps.x,ps.y);
                    rt.addValue(c+" Average Diameter",ps.d*XYsc);
                    rt.addValue(c+" Average Diameter stdev",ps.std*XYsc);
                    c++;
                    if(ps.d!=0) f++;
                }
            }
            ar.add(pos);
            nis.addSlice("PSFilter_"+i,nip);
            nis2.addSlice("PSFilterEdge_"+i,nip2);
            nis3.addSlice("Diameters_"+i,nip3);
            rt.incrementCounter();
        }

        rt.addValue(c+" Z Axis size",(f-1)*Zsc);


        ArrayList <posdiam> pos_ = new ArrayList<posdiam>();
        pos_=ar.get(0);
        ArrayList <posdiam> pos0 = new ArrayList<posdiam>();
        for(int j=0;j<pos_.size();j++){
            posdiam ps = new posdiam();
            ps=pos_.get(j);
            pos0.add(ps);
        }

        ImagePlus nipp = new ImagePlus("PSFilter Stack",nis);

        nipp.show();

        ImagePlus nipp2 = new ImagePlus("PSFilterEdge Stack",nis2);

        nipp2.show();
```

```
        ImagePlus nipp3 = new ImagePlus("Diameters Stack",nis3);

        nipp3.show();




        rt.show("Results");
    }

}


    public class hist{
        double x,y;
        }



    public int[] getThresPkF(ImageProcessor iip){

        int[] th = new int[2];


        ImagePlus imp = new ImagePlus() ;

        ImageProcessor ip = iip.duplicate();                //e una variabile
ImageProcessor

        ArrayList hst,hmx,hmx2;
        hst = new ArrayList();
        hmx = new ArrayList();
        hmx2 = new ArrayList();


        double[] bkp,xp,db,db2,lgn,adb,adb2,gs;
        int[] histg = new int[256];
        histg=ip.getHistogram();



        int i,ct=0,ct2=0,ct3=0;
        double mode=0;
        double max=0;

        for(int j=0;j<256;j++){
            hist bk= new hist();
            bk.y=histg[j];
            bk.x=j;
            hst.add(bk);

        }


        hmx=getHistMax(hst,2);
        hmx2=getHistMax(hmx,2);

        for(i=0;i<hst.size();i++)  {
            hist bk= new hist();
```

```
            bk=(hist) hst.get(i);
        if (bk.y!=0) {

            ct++;
        }


    }
    for(i=0;i<hmx.size();i++)  {
        hist bk= new hist();
            bk=(hist) hmx.get(i);
        if (bk.y!=0) {

            ct2++;
        }



    }


    for(i=0;i<hmx2.size();i++)  {
        hist bk= new hist();
            bk=(hist) hmx2.get(i);
        if (bk.y!=0) {

            ct3++;
        }


    }

    double[] bmx = new double[ct2];
    double[] xmx = new double[ct2];
    double[] bmx2 = new double[ct3];
    double[] xmx2 = new double[ct3];
    int j;
    i=0;
    for(j=0;j<hmx.size();j++)  {
        hist bk= new hist();
            bk=(hist) hmx.get(j);
        if (bk.y!=0){
            bmx[i]=bk.y;
            xmx[i]=bk.x;
            i++;
        }
    }
    i=0;
    for(j=0;j<hmx2.size();j++)  {
        hist bk= new hist();
            bk=(hist) hmx2.get(j);
        if (bk.y!=0){
            bmx2[i]=bk.y;
            xmx2[i]=bk.x;
            i++;
        }
    }

    double mn=1,xf=0;
    int thp=1000,imn=0;
    bkp= new double[ct];
    xp= new double[ct];
```

```
db= new double[ct];
adb= new double[ct];
adb2= new double[ct];
db2= new double[ct];
lgn= new double[ct];
gs= new double[ct];

i=0;
for(j=0;j<hst.size();j++)  {
    hist bk= new hist();
        bk=(hist) hst.get(j);
    if (bk.y!=0){
        bkp[i]=bk.y;
        xp[i]=bk.x;
        if(i>0) {
            db[i]=bkp[i]-bkp[i-1];
            db2[i]=db[i]-db[i-1];
        }
        else db[i]=0;
        if(mn>=db[i]){
            mn=db[i];

            imn=i;
        }

        i++;
    }


}

max=0;
if(ct2>2){
    for(i=0;i<3;i++){
        if(max<=bmx[i]){
            max=bmx[i];
            mode=xmx[i];
        }
    }
}
else{
    for(i=0;i<ct2;i++){
        if(max<=bmx[i]){
            max=bmx[i];
            mode=xmx[i];
        }
    }

}


i=1;
int f=0;
while(i<ct&&f==0) {
    if(bkp[i-1]<=max/2&&bkp[i]>=max/2) f=1;
    i++;
}
xf=xp[i-1];
```

```
        double dxf=mode-xf;


        double s=Math.exp(-0.5*(Math.pow(-dxf,2)-Math.log(bkp[i-1]/max)));

        if(Math.round(mode)<=2) s=1;


        for(i=0;i<ct;i++)  {
            gs[i]=Math.exp(-Math.pow(xp[i]-
mode,2)/(2*Math.pow(s,2)))/(s*Math.sqrt(2*Math.PI));
        }

        double mxd=-20000000;
        int imxd=0;
        for(i=imn;i<ct-2;i++)  {

            if(mxd<=db2[i]){
            mxd=db2[i];
            //thp=(int) xp[i];
            imxd=i;
            }
        }
        i=imxd;


        th[0]=(int) Math.round(mode+2*s);
        th[1]=(int) Math.round(s);


        Plot pl1 = new Plot("Histogram1", "Level","Density", xp, bkp);
        pl1.show();

        Plot pl3 = new Plot("Histogram3", "Level","Density",xp, gs);
        pl3.show();

        return (int[]) th;

    }



    public float[] mkGaussianKernel(int dim, double sigma) {
        float[] kernel;
        kernel = new float[dim*dim];

        float[] hg;
        hg = new float[dim*dim];

        float sm=0;
        int k;
        for(int i=0;i<dim;i++) {

            for(int j=0;j<dim;j++) {
            Float f = new Float(Math.exp((i*i+j*j)/(sigma*sigma)));
             hg[i+dim*j]=f;
            sm=sm+hg[i+dim*j];
            }
        }

        for(int i=0;i<dim;i++) {
```

```
            for(int j=0;j<dim;j++) {
             kernel[i+dim*j]=hg[i+dim*j]/sm;
            }
        }

    return (float[]) kernel;
    }


    public ArrayList prtcPos(ImageProcessor ip, byte[] pixels,int x, int y){
        int c=0;
        int width = ip.getWidth();
        int height = ip.getHeight();

        ArrayList pos;
        pos= new ArrayList();

        if(pixels[y*width+x]==0){

                        for(int i=-1;i<=1;i++){
                            for(int j=-1;j<=1;j++){

if((x+i>=0)&&(y+j>=0)&&(x+i<width)&&(y+j<height)){
                                        if(pixels[(y+j)*width+x+i]==0){
                                            position ps = new position();
                                            ps.x=x+i;
                                            ps.y=y+j;
                                            pos.add((position) ps);

                                        }

                                    }

                                }
                            }


        }



    return (ArrayList) pos;
    }


    public ArrayList <position> addPrtcPos(ImageProcessor ip,byte[] pixels ,
int x, int y,ArrayList <position>prev){

        int width = ip.getWidth();
        int height = ip.getHeight();


                    if(pixels[y*width+x]==0){

                        for(int i=-1;i<=1;i++){
                            for(int j=-1;j<=1;j++){

if((x+i>=0)&&(y+j>=0)&&(x+i<width)&&(y+j<height)){

                                        if(pixels[(y+j)*width+x+i]==0){
                                            position ps = new position();
                                            ps.x=x+i;
```

```
                                    ps.y=y+j;
                                    prev.add((position) ps);
                                }

                            }

                        }
                    }


                }


return (ArrayList <position>) prev;
}


public ArrayList <position> prtcPos4n(ImageProcessor ip, int x, int y){
    int c=0;
    int width = ip.getWidth();
    int height = ip.getHeight();
    byte[] pixels = (byte[])ip.getPixels();
    //Array pos;
    //pos = newInstance(position, 0);
    ArrayList <position> pos;
    pos= new ArrayList<position>();


                if(pixels[y*width+x]==0){

                    for(int i=-1;i<=1;i++){

                            if((x+i>=0)&&(i!=0)&&(x+i<width)&&(y<height)){
                                if(pixels[y*width+x+i]==0){
                                    position ps = new position();
                                    ps.x=x+i;
                                    ps.y=y;
                                    pos.add((position) ps);

                                    c++;
                                }

                            }


                    }

                    for(int j=-1;j<=1;j++){
                            if((j!=0)&&(y+j>=0)&&(x<width)&&(y+j<height)){
                                if(pixels[(y+j)*width+x]==0){
                                    position ps = new position();
                                    ps.x=x;
                                    ps.y=y+j;
                                    pos.add((position) ps);

                                    c++;
                                }

                            }

                    }
```

```
                }


return (ArrayList <position>) pos;
}


public ArrayList <position> prtcPos4nb(ImageProcessor ip, int x, int y){
    int c=0;
    int width = ip.getWidth();
    int height = ip.getHeight();
    byte[] pixels = (byte[])ip.getPixels();
    //Array pos;
    //pos = newInstance(position, 0);
    ArrayList <position> pos;
    pos= new ArrayList<position>();


                if(pixels[y*width+x]==255){

                    for(int i=-1;i<=1;i++){

                            if((x+i>=0)&&(i!=0)&&(x+i<width)&&(y<height)){
                                if(pixels[y*width+x+i]==0){
                                    position ps = new position();
                                    ps.x=x+i;
                                    ps.y=y;
                                    pos.add((position) ps);

                                    c++;
                                }

                            }


                    }

                    for(int j=-1;j<=1;j++){
                            if((j!=0)&&(y+j>=0)&&(x<width)&&(y+j<height)){
                                if(pixels[(y+j)*width+x]==0){
                                    position ps = new position();
                                    ps.x=x;
                                    ps.y=y+j;
                                    pos.add((position) ps);

                                    c++;
                                }

                            }

                    }


                }
```

```
return (ArrayList <position>) pos;
}



public class position {
    int x,y;
}


public byte[] ParticleSizeFilter(ImageProcessor ip,int size){
int width = ip.getWidth();
int height = ip.getHeight();

    byte[] pixels = (byte[])ip.getPixels();
    ArrayList <position> pos = new ArrayList<position>();
ImageProcessor ip2=ip.duplicate();
    byte[] pix2 = (byte[])ip2.getPixels();

    int p1, p2, c=0, f=0,k=0;


    double pr,cnt=0;

        for(int x=0;x<width;x++){
            for(int y=0;y<height;y++){

                if(pixels[y*width+x]==0){

                    pos =   prtcPos(ip,pixels,x,y);

                    k=0;

                    while(k<pos.size()){
                        position ps = new position();
                        ps = (position) pos.get(k);
                        pos= addPrtcPos(ip,pixels,ps.x,ps.y,pos);
                        k++;

                        pixels[ps.y*width+ps.x]=(byte) 128;

                    }

                    if(pos.size()<=size&&pos.size()>0) {

                        for( k=0;k<pos.size();k++) {
                            position ps = new position();
                            ps = (position) pos.get(k);

                            pix2[ps.y*width+ps.x]=(byte) 255;

                        }
                    }


                }
                cnt++;

                pr=cnt/(width*height);
```

```
                }
            }


        return (byte[]) pix2;

    }


    public byte[] CleanParticle(ImageProcessor ip,int n,int type){
    int width = ip.getWidth();
    int height = ip.getHeight();
        byte[] pix = (byte[])ip.getPixels();

        ArrayList <position> pos = new ArrayList<position>();

        int p1, p2, c=0, f=0,k=0;


        IJ.showProgress(0.0);

        double pr,cnt=0;

            for(int x=0;x<width;x++){
                for(int y=0;y<height;y++){

                    if(type==0) pos =   prtcPos4n(ip,x,y);
                    else pos =   prtcPos(ip,pix,x,y);

                    if(n>=pos.size())   ip.set(x,y,255);



                    cnt++;

                    pr=cnt/(width*height*2);
                    IJ.showProgress(pr);

                }
            }



        byte[] pixels = (byte[])ip.getPixels();
        return (byte[]) pixels;

    }


    public byte[] Erosion(ImageProcessor ip){
    int width = ip.getWidth();
    int height = ip.getHeight();
    byte[] pix = (byte[])ip.getPixels();

        ArrayList <position> pos = new ArrayList<position>();

        int p1, p2, c=0, f=0,k=0;
```

```
        IJ.showProgress(0.0);

        double pr,cnt=0;

            for(int x=0;x<width;x++){
                for(int y=0;y<height;y++){

                    pos =   prtcPos(ip,pix,x,y);

                    if(5>=pos.size())   ip.set(x,y,0);



                    cnt++;

                    pr=cnt/(width*height);
                    IJ.showProgress(pr);

                }
            }



        byte[] pixels = (byte[])ip.getPixels();
        return (byte[]) pixels;

    }


    public byte[] Edge(ImageProcessor ip){

        ImageProcessor ip2;
        ip2=ip.duplicate();
        ip2.fill();
        ip2.invert();
        int width = ip.getWidth();
        int height = ip.getHeight();
        //byte[] pixels;
        int p1,p2,p3;

            for(int i=0;i<height;i++) {
                for(int j=1;j<width-1;j++) {

                    p1=ip.get(j-1,i);
                    p2=ip.get(j,i);
                    p3=ip.get(j+1,i);

                    if((p1 == 0)&&(p2 == 255)&&(p3 == 255)) ip2.set(j-1,i,0);
                    if((p1 == 255)&&(p2 == 0)&&(p3 == 255)) ip2.set(j,i,0);
                    if((p1 == 255)&&(p2 == 255)&&(p3 == 0)) ip2.set(j+1,i,0);
                    if((p1 == 0)&&(p2 == 255)&&(p3 == 0)){
                        ip2.set(j-1,i,0);
                        ip2.set(j+1,i,0);
                    }


                }
            }

            for(int i=1;i<height-1;i++) {
                for(int j=0;j<width;j++) {
```

```
                    p1=ip.get(j,i-1);
                    p2=ip.get(j,i);
                    p3=ip.get(j,i+1);

                    if((p1 == 0)&&(p2 == 255)&&(p3 == 255)) ip2.set(j,i-1,0);
                    if((p1 == 255)&&(p2 == 0)&&(p3 == 255)) ip2.set(j,i,0);
                    if((p1 == 255)&&(p2 == 255)&&(p3 == 0)) ip2.set(j,i+1,0);
                    if((p1 == 0)&&(p2 == 255)&&(p3 == 0)){
                        ip2.set(j,i-1,0);
                        ip2.set(j,i+1,0);
                    }
                }
            }

        byte[] pixels = (byte[])ip2.getPixels();
        return (byte[]) pixels;
    }

    public byte[] Diff(ImageProcessor ip1,ImageProcessor ip2){

        byte[] pixels = (byte[])ip1.duplicate().getPixels();

        int width = ip1.getWidth();
        int height = ip1.getHeight();

        int p1,p2;

            for(int i=0;i<height;i++) {
                for(int j=0;j<width;j++) {
                    p1=ip1.get(j,i);
                    p2=ip2.get(j,i);
                if(p2 == 0)     pixels[j+width*i]= (byte) 255;
                else    pixels[j+width*i]=(byte) p1;
                }
            }



        return (byte[]) pixels;
    }

    public byte[] FillHoles(ImageProcessor ip){
    int width = ip.getWidth();
    int height = ip.getHeight();

        ArrayList <position> pos = new ArrayList<position>();

        int p1, p2, c=0, f=0,k=0;


        IJ.showProgress(0.0);

        double pr,cnt=0;

            for(int x=0;x<width;x++){
                for(int y=0;y<height;y++){

                    pos =   prtcPos4nb(ip,x,y);

                    if(pos.size()>2)    ip.set(x,y,0);
```

```
                cnt++;

                pr=cnt/(width*height);
                IJ.showProgress(pr);

            }
        }



    byte[] pixels = (byte[])ip.getPixels();
    return (byte[]) pixels;

}



public byte[] FillEdge(ImageProcessor ip, ArrayList <particle> prt){
int width = ip.getWidth();
int height = ip.getHeight();

     ArrayList <position> pos = new ArrayList<position>();

ImageProcessor ip2=ip.duplicate();

    int p1, p2, f=0,k=0;


    IJ.showProgress(0.0);

    double prg,cnt=0;


        for(int x=0;x<width-1;x++){
            for(int y=0;y<height-1;y++){

                for(int i=0;i<prt.size();i++){

                    f=0;

                    particle pr = new particle();
                    pr= (particle) prt.get(i);
                    pos=pr.pos;


                    boolean a = (boolean) false;
                    boolean b = (boolean) false;
                    boolean c = (boolean) false;
                    boolean d = (boolean) false;

                    for( k=0;k<pos.size();k++) {
                        position ps2 = new position();
                        ps2 = (position) pos.get(k);

                        a=((x>=ps2.x)&&(x<=pr.xc));
                        b=((y>=ps2.y)&&(y<=pr.yc));
                        c=((x<=ps2.x)&&(x>=pr.xc));
                        d=((y<=ps2.y)&&(y>=pr.yc));
```

```
                        if((a&&b)||(a&&d)||(c&&b)||(c&&d)) f=1;

                    }

                    if(f==1)  ip2.putPixel(x,y,0);
                }

                cnt++;

                prg=cnt/(width*height*2);
                IJ.showProgress(prg);

            }
        }

    byte[] pixels = (byte[])ip2.getPixels();
    return (byte[]) pixels;

}


public class particle{
    ArrayList <position> pos;
    int xc, yc, n;
}

public ArrayList <particle> FindParticle(ImageProcessor ip){
int width = ip.getWidth();
int height = ip.getHeight();
    byte[] pix = (byte[])ip.getPixels();

    ArrayList <position> pos = new ArrayList<position>();
    ArrayList <particle> posc = new ArrayList<particle>();
ImageProcessor ip2=ip.duplicate();

    int p1, p2, c=0, f=0,k=0;


    IJ.showProgress(0.0);

    double pr,cnt=0;
    Double xc = new Double(0);
    Double yc = new Double(0);

        for(int x=1;x<width-1;x++){
            for(int y=1;y<height-1;y++){
                xc=(double) 0;
                yc=(double) 0;
                pos =   prtcPos(ip,pix,x,y);
                 k=0;

                while(k<pos.size()){
                    position ps = new position();
                    ps = (position) pos.get(k);
                    pos= addPrtcPos(ip,pix,ps.x,ps.y,pos);
                    k++;
                    pix[ps.y*width+ps.x]=(byte) 128;
                }


                    for( k=0;k<pos.size();k++) {
```

```
                    position ps = new position();
                    ps = (position) pos.get(k);
                    xc=xc+ps.x;
                    yc=yc+ps.y;
                    }
                    xc=xc/pos.size();
                    yc=yc/pos.size();
                    particle prt = new particle();
                    prt.xc=xc.intValue();
                    prt.yc=yc.intValue();
                    prt.pos=pos;
                    prt.n=posc.size();
                    k=0;
                    f=0;
                    while(k<posc.size()){
                        particle prt2 = new particle();
                        prt2=(particle) posc.get(k);
                        if((prt.xc==prt2.xc)&&(prt.yc==prt2.yc)) f=1;
                        k++;
                    }
                    if(f==0) posc.add(prt);

                cnt++;

                pr=cnt/(width*height);
                IJ.showProgress(pr);

            }
        }




    return (ArrayList <particle>) posc;

}

public ArrayList <position> FindCentre(ImageProcessor ip){
int width = ip.getWidth();
int height = ip.getHeight();
byte[] pix = (byte[])ip.getPixels();

    ArrayList <position> pos = new ArrayList<position>();
    ArrayList <position> posc = new ArrayList<position>();
ImageProcessor ip2=ip.duplicate();

    int p1, p2, c=0, f=0,k=0;

    IJ.showProgress(0.0);

    double pr,cnt=0;
    Double xc = new Double(0);
    Double yc = new Double(0);

        for(int x=1;x<width-1;x++){
            for(int y=1;y<height-1;y++){
                xc=(double) 0;
                yc=(double) 0;
                pos =   prtcPos(ip,pix,x,y);
                 k=0;
```

```
                while(k<pos.size()){
                    position ps = new position();
                    ps = (position) pos.get(k);
                    pos= addPrtcPos(ip,pix,ps.x,ps.y,pos);
                    k++;
                    pix[ps.y*width+ps.x]=(byte) 128;
                }


                    for( k=0;k<pos.size();k++) {
                    position ps = new position();
                    ps = (position) pos.get(k);
                    xc=xc+ps.x;
                    yc=yc+ps.y;
                    }
                    xc=xc/pos.size();
                    yc=yc/pos.size();
                    position ps = new position();
                    ps.x=xc.intValue();
                    ps.y=yc.intValue();

                    posc.add(ps);

                cnt++;

                pr=cnt/(width*height*2);
                IJ.showProgress(pr);

            }
        }




    return (ArrayList <position>) posc;

}

public ArrayList <posdiam> FindDiam3(ArrayList <particle>  prt){


    ArrayList <position> pos = new ArrayList<position>();
    ArrayList <posdiam> posc = new ArrayList<posdiam>();
    ArrayList <posdiam> poso = new ArrayList<posdiam>();

    int p1, p2, c=0, f=0,k=0,idx=0;

    IJ.showProgress(0.0);

    double prg,cnt=0,d=0,pr_xc,pr_yc,dmx=0;

    for(int i=0;i<prt.size();i++){
        d=0;
        particle pr = new particle();
        pr=(particle) prt.get(i);
        pos = pr.pos;
        posdiam psd = new posdiam();
        psd.x=pr.xc;
        psd.y=pr.yc;
        for( k=0;k<pos.size();k++) {
            position ps = new position();
```

```
                ps = (position) pos.get(k);
                d=d+2*Math.sqrt((pr.xc-ps.x)*(pr.xc-ps.x)+(pr.yc-ps.y)*(pr.yc-
ps.y));
            }

            d=d/pos.size();
            psd.d=d;
            if(dmx<=d){
                dmx=d;
                idx=i;
            }
            double std=0;
            for( k=0;k<pos.size();k++) {
                position ps = new position();
                ps = (position) pos.get(k);
                std=std+Math.pow(2*Math.sqrt((pr.xc-ps.x)*(pr.xc-ps.x)+(pr.yc-
ps.y)*(pr.yc-ps.y))-d,2);
            }

            std=Math.sqrt(std/(pos.size()-1));
            psd.std=std;
            posc.add(psd);
        }

        posdiam psd = new posdiam();
        psd=(posdiam) posc.get(idx);
        poso.add(psd);

    return (ArrayList <posdiam>) poso;

    }

    public ArrayList <posdiam> FindDiam2(ArrayList <particle>  prt){


        ArrayList <position> pos = new ArrayList<position>();
        ArrayList <posdiam> posc = new ArrayList<posdiam>();

        int p1, p2, c=0, f=0,k=0;

        IJ.showProgress(0.0);

        double prg,cnt=0,d=0,pr_xc,pr_yc;

        for(int i=0;i<prt.size();i++){
            d=0;
            particle pr = new particle();
            pr=(particle) prt.get(i);
            pos = pr.pos;
            posdiam psd = new posdiam();
            psd.x=pr.xc;
            psd.y=pr.yc;
            for( k=0;k<pos.size();k++) {
                position ps = new position();
                ps = (position) pos.get(k);
                d=d+2*Math.sqrt((pr.xc-ps.x)*(pr.xc-ps.x)+(pr.yc-ps.y)*(pr.yc-
ps.y));
            }

            d=d/pos.size();
            psd.d=d;
            double std=0;
```

```
            for( k=0;k<pos.size();k++) {
                position ps = new position();
                ps = (position) pos.get(k);
                std=std+Math.pow(2*Math.sqrt((pr.xc-ps.x)*(pr.xc-ps.x)+(pr.yc-
ps.y)*(pr.yc-ps.y))-d,2);
            }

            std=Math.sqrt(std/(pos.size()-1));
            psd.std=std;
            posc.add(psd);
        }
    return (ArrayList <posdiam>) posc;

    }


    public ArrayList <posdiam> FindDiam(ImageProcessor ip){
    int width = ip.getWidth();
    int height = ip.getHeight();

        byte[] pix = (byte[])ip.getPixels();
        ArrayList <position> pos = new ArrayList<position>();
        ArrayList <posdiam> posc = new ArrayList<posdiam>();
        ArrayList <posdiam> posc2 = new ArrayList<posdiam>();
    ImageProcessor ip2=ip.duplicate();

        int p1, p2, c=0, f=0,k=0;

        IJ.showProgress(0.0);

        double pr,cnt=0,d=0,pr_xc,pr_yc;
        Double xc = new Double(0);
        Double yc = new Double(0);

            for(int x=1;x<width-1;x++){
                for(int y=1;y<height-1;y++){


                    d=0;

                    pos =   prtcPos(ip,pix,x,y);

                    k=0;
                    while(k<pos.size()){
                        position ps = new position();
                        ps = (position) pos.get(k);
                        pos= addPrtcPos(ip,pix,ps.x,ps.y,pos);
                        k++;
                        pix[ps.y*width+ps.x]=(byte) 128;
                    }
                    xc=(double) 0;
                    yc=(double) 0;

                        for( k=0;k<pos.size();k++) {
                        position ps = new position();
                        ps = (position) pos.get(k);
                        xc=xc+ps.x;
                        yc=yc+ps.y;
                        }
                        if(pos.size()!=0){
                            xc=xc/pos.size();
                            yc=yc/pos.size();
```

```
                                f=0;
                                k=0;
                                while( k<posc.size()) {
                                    posdiam psd = new posdiam();
                                    psd = (posdiam) posc.get(k);

if((psd.x==xc.intValue())&&(psd.y==yc.intValue())) f=1;

                                    k++;
                                }


                                if(f==0){
                                    posdiam psd = new posdiam();
                                    psd.x=xc.intValue();
                                    psd.y=yc.intValue();

                                        for( k=0;k<pos.size();k++) {
                                            position ps = new position();
                                            ps = (position) pos.get(k);
                                            d=d+2*Math.sqrt((xc-ps.x)*(xc-ps.x)+(yc-
ps.y)*(yc-ps.y));

                                        }

                                    d=d/pos.size();
                                    psd.d=d;
                                    double std=0;

                                        for( k=0;k<pos.size();k++) {
                                            position ps = new position();
                                            ps = (position) pos.get(k);
                                            std=std+Math.pow(2*Math.sqrt((xc-
ps.x)*(xc-ps.x)+(yc-ps.y)*(yc-ps.y))-d,2);

                                        }

                                    std=Math.sqrt(std/(pos.size()-1));
                                    psd.std=std;
                                    posc.add(psd);
                                }
                            }

                        cnt++;

                        pr=cnt/(width*height);
                        IJ.showProgress(pr);

                    }
                }



            return (ArrayList <posdiam>) posc;

        }


        public class posdiam {
            int x,y,n;
```

```
    double d,std;
}


public ArrayList getHistMax(ArrayList <hist> hst,int stp){
    ArrayList max;
    max= new ArrayList();

    int i,ct=0,j;

    for(i=0;i<hst.size();i++)  {
        hist bk= new hist();
            bk=(hist) hst.get(i);
        if (bk.y!=0)  ct++;
    }
    double[] bkp,xp;
    bkp= new double[ct];
    xp= new double[ct];

    i=0;
    for(j=0;j<hst.size();j++)  {
        hist bk= new hist();
            bk=(hist) hst.get(j);
        if (bk.y!=0){
            bkp[i]=bk.y;
            xp[i]=bk.x;
            i++;
        }
    }


    double mx=0;
    int c=0;

    for(i=0;i<ct;i++){
        if(mx<=bkp[i]){
            mx=bkp[i];
            c=i;
        }
        if(i-c>=stp){
            hist bk= new hist();
            bk.x=xp[c];
            bk.y=bkp[c];
            c=0;
            mx=0;
            max.add(bk);
        }

    }




    return(ArrayList) max;
}


public ArrayList getHistogram(ImageProcessor ip){
    ArrayList hst;
```

```
        hst= new ArrayList();
        int width = ip.getWidth();
        int height = ip.getHeight();
        byte[] pixels = (byte[])ip.getPixels();
        double max=ip.getMax();

        int val=0;
        double c=0;


        int f=0;
        for(int j=0;j<256;j++){
            hist bk= new hist();
            bk.y=0;
            bk.x=j;
            hst.add(bk);

        }


        for(int i=0;i<width*height;i++){
                    val=pixels[i];

                hist bk= new hist();
                    bk=(hist)hst.get(val+1);

                bk.x=val;
                bk.y=bk.y+1;

                    hst.set(val,bk);
                    bk=(hist)hst.get(val);


        }


        return(ArrayList) hst;
    }

}
```

AnalyzeSingleAll_.java

```
import ij.*;
import ij.process.*;
import ij.measure.*;
import ij.gui.*;
import java.awt.*;
import java.math.*;
import java.io.*;
import java.lang.Number;
import java.lang.Byte;
import java.lang.Short;
import java.lang.Double;
import java.lang.Boolean;
import ij.plugin.*;
import ij.plugin.filter.*;
import ij.plugin.filter.GaussianBlur.*;
import ij.plugin.frame.*;
import ij.plugin.filter.PlugInFilter;
import ij.text.TextPanel;
import java.util.*;
import java.lang.Object.*;
import java.lang.reflect.Array;
import ij.plugin.filter.ParticleAnalyzer;
import ij.measure.Measurements;
import ij.plugin.filter.Analyzer;
import ij.measure.ResultsTable;


public class AnalyzeSingleAll_ implements PlugInFilter {

    protected ImageProcessor image;

    public int setup(String arg, ImagePlus imp) {
        image=imp.getProcessor();
        if (arg.equals("about"))
            { return DONE; }
        return DOES_8G+SUPPORTS_MASKING;
    }

    public void run(ImageProcessor ipp) {

         GenericDialog gd = new GenericDialog("Find Diameter");

        double XYsc=0.115089;
        int psz=600,ths2=10;


        gd.addNumericField("Particle Size Filter: ", psz, 0);
        gd.addNumericField("Manual Threshold: ", ths2, 0);
        gd.addNumericField("XY scale (pixel*um): ", XYsc, 6);


        gd.showDialog();
        if (gd.wasCanceled()) return;

        psz = (int)gd.getNextNumber();
        ths2 = (int)gd.getNextNumber();
        XYsc = (double)gd.getNextNumber();
```

```
        ImageProcessor nip;
        nip=image.duplicate();
        ImageStack nis = new ImageStack(nip.getWidth(),nip.getHeight());

        ResultsTable rt = ResultsTable.getResultsTable();
        rt.reset();
        rt.incrementCounter();
        rt.setPrecision(6);

        rt.addValue("XY Pixel size",XYsc);

        int idx=0,f=0,c=0,ths;
        double t;

        for (ths=5;ths<=250;ths=ths+5){


            ImageProcessor ApIp;
            ApIp=nip.duplicate();
            ApIp.threshold((int)ths);
            ApIp.invert();

            ImagePlus ApIpp = new ImagePlus("Analyze",ApIp);


            ParticleAnalyzer pa = new
ParticleAnalyzer(ParticleAnalyzer.SHOW_RESULTS , Measurements.AREA
+Measurements.RECT + Measurements.ELLIPSE + Measurements.FERET , rt, psz,
Double.POSITIVE_INFINITY);

            pa.analyze(ApIpp);
            rt.addValue("th",ths);
            nis.addSlice("Thresholded_"+c,ApIp);
            c++;

        }

        ImagePlus nipp = new ImagePlus("Thresholded Stack",nis);

        nipp.show();


            nip.threshold((int)ths2);
            nip.invert();
            nip.setPixels(ParticleSizeFilter(nip,50));

            rt.addValue("th",ths2);


            ImageProcessor nip2;
            nip2=nip.duplicate();
            nip2.setPixels(Edge(nip2));
            nip.setPixels(Diff(nip,nip2));

            nip.setPixels(FillHoles(nip));
            nip.setPixels(FillHoles(nip));
            nip.setPixels(FillHoles(nip));
            nip.setPixels(ParticleSizeFilter(nip,psz));

            ArrayList <posdiam> pos = new ArrayList<posdiam>();
            ArrayList <particle> prt = new ArrayList<particle>();
```

```
            nip2.setPixels(Edge(nip));
            prt=FindParticle(nip2);

            nip.setPixels(FillEdge(nip2,prt));

            nip2.setPixels(Edge(nip));
            prt=FindParticle(nip2);

            pos=FindDiam3(prt);

            ImageProcessor nip3;
            nip3=nip.duplicate();
            nip3.fill();
            nip3.invert();
            int x=0,y=0;
            double dmx=0;
            idx=pos.size();

            rt.incrementCounter();
             for(int k=0;k<pos.size();k++){
                posdiam ps = new posdiam();
                ps=pos.get(k);
                if((ps.x!=0)&&(ps.y!=0)){

                    nip3.drawOval((int)(ps.x-ps.d/2),(int) (ps.y-
ps.d/2),(int)ps.d,(int)ps.d);
                    nip3.drawPixel(ps.x,ps.y);
                    rt.addValue(" Average Diameter",ps.d*XYsc);
                    rt.addValue(" Average Diameter stdev",ps.std*XYsc);
                    rt.addValue("XY Pixel size",XYsc);

                    if(ps.d!=0) f++;
                }
            }


        rt.show("Results");


}


    public class hist{
        double x,y;
        }



    public float[] mkGaussianKernel(int dim, double sigma) {
        float[] kernel;
        kernel = new float[dim*dim];

        float[] hg;
        hg = new float[dim*dim];

        float sm=0;
        int k;
        for(int i=0;i<dim;i++) {

            for(int j=0;j<dim;j++) {
```

```
            Float f = new Float(Math.exp((i*i+j*j)/(sigma*sigma)));
             hg[i+dim*j]=f;
            sm=sm+hg[i+dim*j];
            }
        }

        for(int i=0;i<dim;i++) {
            for(int j=0;j<dim;j++) {
             kernel[i+dim*j]=hg[i+dim*j]/sm;
            }
        }

    return (float[]) kernel;
    }


    public ArrayList prtcPos(ImageProcessor ip, byte[] pixels,int x, int y){
        int c=0;
        int width = ip.getWidth();
        int height = ip.getHeight();

        ArrayList pos;
        pos= new ArrayList();

        if(pixels[y*width+x]==0){

                        for(int i=-1;i<=1;i++){
                            for(int j=-1;j<=1;j++){

if((x+i>=0)&&(y+j>=0)&&(x+i<width)&&(y+j<height)){
                                        if(pixels[(y+j)*width+x+i]==0){
                                            position ps = new position();
                                            ps.x=x+i;
                                            ps.y=y+j;
                                            pos.add((position) ps);

                                        }

                                    }

                                }
                            }


        }



    return (ArrayList) pos;
    }


    public ArrayList <position> addPrtcPos(ImageProcessor ip,byte[] pixels ,
int x, int y,ArrayList <position>prev){

        int width = ip.getWidth();
        int height = ip.getHeight();


                    if(pixels[y*width+x]==0){

                        for(int i=-1;i<=1;i++){
```

```
                            for(int j=-1;j<=1;j++){

if((x+i>=0)&&(y+j>=0)&&(x+i<width)&&(y+j<height)){

                                    if(pixels[(y+j)*width+x+i]==0){
                                        position ps = new position();
                                        ps.x=x+i;
                                        ps.y=y+j;
                                        prev.add((position) ps);
                                    }

                                }

                            }
                        }


                    }


    return (ArrayList <position>) prev;
    }


    public ArrayList <position> prtcPos4n(ImageProcessor ip, int x, int y){
        int c=0;
        int width = ip.getWidth();
        int height = ip.getHeight();
        byte[] pixels = (byte[])ip.getPixels();

        ArrayList <position> pos;
        pos= new ArrayList<position>();


                    if(pixels[y*width+x]==0){

                        for(int i=-1;i<=1;i++){

                                if((x+i>=0)&&(i!=0)&&(x+i<width)&&(y<height)){
                                    if(pixels[y*width+x+i]==0){
                                        position ps = new position();
                                        ps.x=x+i;
                                        ps.y=y;
                                        pos.add((position) ps);

                                        c++;
                                    }

                                }


                        }

                        for(int j=-1;j<=1;j++){
                                if((j!=0)&&(y+j>=0)&&(x<width)&&(y+j<height)){
                                    if(pixels[(y+j)*width+x]==0){
                                        position ps = new position();
                                        ps.x=x;
                                        ps.y=y+j;
                                        pos.add((position) ps);
```

```
                                    c++;
                                }

                            }

                        }



                }


return (ArrayList <position>) pos;
}


public ArrayList <position> prtcPos4nb(ImageProcessor ip, int x, int y){
    int c=0;
    int width = ip.getWidth();
    int height = ip.getHeight();
    byte[] pixels = (byte[])ip.getPixels();

    ArrayList <position> pos;
    pos= new ArrayList<position>();


                if(pixels[y*width+x]==255){

                    for(int i=-1;i<=1;i++){

                            if((x+i>=0)&&(i!=0)&&(x+i<width)&&(y<height)){
                                if(pixels[y*width+x+i]==0){
                                    position ps = new position();
                                    ps.x=x+i;
                                    ps.y=y;
                                    pos.add((position) ps);

                                    c++;
                                }

                            }


                    }

                    for(int j=-1;j<=1;j++){
                            if((j!=0)&&(y+j>=0)&&(x<width)&&(y+j<height)){
                                if(pixels[(y+j)*width+x]==0){
                                    position ps = new position();
                                    ps.x=x;
                                    ps.y=y+j;
                                    pos.add((position) ps);

                                    c++;
                                }

                            }

                    }
```

```
                }


return (ArrayList <position>) pos;
}




public class position {
    int x,y;
}


public byte[] ParticleSizeFilter(ImageProcessor ip,int size){
int width = ip.getWidth();
int height = ip.getHeight();

    byte[] pixels = (byte[])ip.getPixels();
    ArrayList <position> pos = new ArrayList<position>();
ImageProcessor ip2=ip.duplicate();
    byte[] pix2 = (byte[])ip2.getPixels();

    int p1, p2, c=0, f=0,k=0;


    double pr,cnt=0;

        for(int x=0;x<width;x++){
            for(int y=0;y<height;y++){

                if(pixels[y*width+x]==0){

                    pos =   prtcPos(ip,pixels,x,y);

                    k=0;

                    while(k<pos.size()){
                        position ps = new position();
                        ps = (position) pos.get(k);
                        pos= addPrtcPos(ip,pixels,ps.x,ps.y,pos);
                        k++;

                        pixels[ps.y*width+ps.x]=(byte) 128;

                    }

                    if(pos.size()<=size&&pos.size()>0) {

                        for( k=0;k<pos.size();k++) {
                            position ps = new position();
                            ps = (position) pos.get(k);

                            pix2[ps.y*width+ps.x]=(byte) 255;

                        }
                    }
```

```
                }
                cnt++;

                pr=cnt/(width*height);


            }
        }


    return (byte[]) pix2;

}


public byte[] CleanParticle(ImageProcessor ip,int n,int type){
int width = ip.getWidth();
int height = ip.getHeight();
    byte[] pix = (byte[])ip.getPixels();

    ArrayList <position> pos = new ArrayList<position>();

    int p1, p2, c=0, f=0,k=0;

    IJ.showProgress(0.0);

    double pr,cnt=0;

        for(int x=0;x<width;x++){
            for(int y=0;y<height;y++){

                if(type==0) pos =   prtcPos4n(ip,x,y);
                else pos =   prtcPos(ip,pix,x,y);

                if(n>=pos.size())   ip.set(x,y,255);



                cnt++;

                pr=cnt/(width*height*2);
                IJ.showProgress(pr);

            }
        }




    byte[] pixels = (byte[])ip.getPixels();
    return (byte[]) pixels;

}


public byte[] Erosion(ImageProcessor ip){
int width = ip.getWidth();
int height = ip.getHeight();
byte[] pix = (byte[])ip.getPixels();

    ArrayList <position> pos = new ArrayList<position>();
```

```
        int p1, p2, c=0, f=0,k=0;


        IJ.showProgress(0.0);

        double pr,cnt=0;

            for(int x=0;x<width;x++){
                for(int y=0;y<height;y++){

                    pos =   prtcPos(ip,pix,x,y);

                    if(5>=pos.size())   ip.set(x,y,0);



                    cnt++;

                    pr=cnt/(width*height);
                    IJ.showProgress(pr);

                }
            }



        byte[] pixels = (byte[])ip.getPixels();
        return (byte[]) pixels;

    }


    public byte[] Edge(ImageProcessor ip){

        ImageProcessor ip2;
        ip2=ip.duplicate();
        ip2.fill();
        ip2.invert();
        int width = ip.getWidth();
        int height = ip.getHeight();

        int p1,p2,p3;

            for(int i=0;i<height;i++) {
                for(int j=1;j<width-1;j++) {

                    p1=ip.get(j-1,i);
                    p2=ip.get(j,i);
                    p3=ip.get(j+1,i);

                    if((p1 == 0)&&(p2 == 255)&&(p3 == 255)) ip2.set(j-1,i,0);
                    if((p1 == 255)&&(p2 == 0)&&(p3 == 255)) ip2.set(j,i,0);
                    if((p1 == 255)&&(p2 == 255)&&(p3 == 0)) ip2.set(j+1,i,0);
                    if((p1 == 0)&&(p2 == 255)&&(p3 == 0)){
                        ip2.set(j-1,i,0);
                        ip2.set(j+1,i,0);
                    }


                }
```

```
            }

            for(int i=1;i<height-1;i++) {
                for(int j=0;j<width;j++) {
                    p1=ip.get(j,i-1);
                    p2=ip.get(j,i);
                    p3=ip.get(j,i+1);

                    if((p1 == 0)&&(p2 == 255)&&(p3 == 255)) ip2.set(j,i-1,0);
                    if((p1 == 255)&&(p2 == 0)&&(p3 == 255)) ip2.set(j,i,0);
                    if((p1 == 255)&&(p2 == 255)&&(p3 == 0)) ip2.set(j,i+1,0);
                    if((p1 == 0)&&(p2 == 255)&&(p3 == 0)){
                        ip2.set(j,i-1,0);
                        ip2.set(j,i+1,0);
                    }
                }
            }

        byte[] pixels = (byte[])ip2.getPixels();
        return (byte[]) pixels;
    }

    public byte[] Diff(ImageProcessor ip1,ImageProcessor ip2){

        byte[] pixels = (byte[])ip1.duplicate().getPixels();

        int width = ip1.getWidth();
        int height = ip1.getHeight();

        int p1,p2;

            for(int i=0;i<height;i++) {
                for(int j=0;j<width;j++) {
                    p1=ip1.get(j,i);
                    p2=ip2.get(j,i);
                if(p2 == 0)     pixels[j+width*i]= (byte) 255;
                else    pixels[j+width*i]=(byte) p1;
                }
            }




        return (byte[]) pixels;
    }

    public byte[] FillHoles(ImageProcessor ip){
    int width = ip.getWidth();
    int height = ip.getHeight();

        ArrayList <position> pos = new ArrayList<position>();

        int p1, p2, c=0, f=0,k=0;

        IJ.showProgress(0.0);

        double pr,cnt=0;

            for(int x=0;x<width;x++){
                for(int y=0;y<height;y++){

                    pos =   prtcPos4nb(ip,x,y);
```

```
                if(pos.size()>2)     ip.set(x,y,0);



                cnt++;

                pr=cnt/(width*height);
                IJ.showProgress(pr);

            }
        }



    byte[] pixels = (byte[])ip.getPixels();
    return (byte[]) pixels;

}



public byte[] FillEdge(ImageProcessor ip, ArrayList <particle> prt){
int width = ip.getWidth();
int height = ip.getHeight();

     ArrayList <position> pos = new ArrayList<position>();

ImageProcessor ip2=ip.duplicate();

    int p1, p2, f=0,k=0;

    IJ.showProgress(0.0);

    double prg,cnt=0;


        for(int x=0;x<width-1;x++){
            for(int y=0;y<height-1;y++){

                for(int i=0;i<prt.size();i++){

                    f=0;

                    particle pr = new particle();
                    pr= (particle) prt.get(i);
                    pos=pr.pos;


                    boolean a = (boolean) false;
                    boolean b = (boolean) false;
                    boolean c = (boolean) false;
                    boolean d = (boolean) false;

                    for( k=0;k<pos.size();k++) {
                        position ps2 = new position();
                        ps2 = (position) pos.get(k);

                        a=((x>=ps2.x)&&(x<=pr.xc));
                        b=((y>=ps2.y)&&(y<=pr.yc));
                        c=((x<=ps2.x)&&(x>=pr.xc));
```

```
                        d=((y<=ps2.y)&&(y>=pr.yc));

                        if((a&&b)||(a&&d)||(c&&b)||(c&&d)) f=1;

                    }

                    if(f==1)  ip2.putPixel(x,y,0);
                }

                cnt++;

                prg=cnt/(width*height*2);
                IJ.showProgress(prg);

            }
        }

    byte[] pixels = (byte[])ip2.getPixels();
    return (byte[]) pixels;

}


public class particle{
    ArrayList <position> pos;
    int xc, yc, n;
}

public ArrayList <particle> FindParticle(ImageProcessor ip){
int width = ip.getWidth();
int height = ip.getHeight();
    byte[] pix = (byte[])ip.getPixels();

    ArrayList <position> pos = new ArrayList<position>();
    ArrayList <particle> posc = new ArrayList<particle>();
ImageProcessor ip2=ip.duplicate();

    int p1, p2, c=0, f=0,k=0;


    IJ.showProgress(0.0);

    double pr,cnt=0;
    Double xc = new Double(0);
    Double yc = new Double(0);

        for(int x=1;x<width-1;x++){
            for(int y=1;y<height-1;y++){
                xc=(double) 0;
                yc=(double) 0;
                pos =   prtcPos(ip,pix,x,y);
                 k=0;

                while(k<pos.size()){
                    position ps = new position();
                    ps = (position) pos.get(k);
                    pos= addPrtcPos(ip,pix,ps.x,ps.y,pos);
                    k++;
                    pix[ps.y*width+ps.x]=(byte) 128;
                }
```

```
                    for( k=0;k<pos.size();k++) {
                    position ps = new position();
                    ps = (position) pos.get(k);
                    xc=xc+ps.x;
                    yc=yc+ps.y;
                    }
                    xc=xc/pos.size();
                    yc=yc/pos.size();
                    particle prt = new particle();
                    prt.xc=xc.intValue();
                    prt.yc=yc.intValue();
                    prt.pos=pos;
                    prt.n=posc.size();
                    k=0;
                    f=0;
                    while(k<posc.size()){
                        particle prt2 = new particle();
                        prt2=(particle) posc.get(k);
                        if((prt.xc==prt2.xc)&&(prt.yc==prt2.yc)) f=1;
                        k++;
                    }
                    if(f==0) posc.add(prt);

                cnt++;

                pr=cnt/(width*height);
                IJ.showProgress(pr);

            }
        }



    return (ArrayList <particle>) posc;

}

public ArrayList <position> FindCentre(ImageProcessor ip){
int width = ip.getWidth();
int height = ip.getHeight();
byte[] pix = (byte[])ip.getPixels();

    ArrayList <position> pos = new ArrayList<position>();
    ArrayList <position> posc = new ArrayList<position>();
ImageProcessor ip2=ip.duplicate();

    int p1, p2, c=0, f=0,k=0;


    IJ.showProgress(0.0);

    double pr,cnt=0;
    Double xc = new Double(0);
    Double yc = new Double(0);

        for(int x=1;x<width-1;x++){
            for(int y=1;y<height-1;y++){
                xc=(double) 0;
                yc=(double) 0;
                pos =   prtcPos(ip,pix,x,y);
```

```
                    k=0;

                while(k<pos.size()){
                    position ps = new position();
                    ps = (position) pos.get(k);
                    pos= addPrtcPos(ip,pix,ps.x,ps.y,pos);
                    k++;
                    pix[ps.y*width+ps.x]=(byte) 128;
                }


                    for( k=0;k<pos.size();k++) {
                    position ps = new position();
                    ps = (position) pos.get(k);
                    xc=xc+ps.x;
                    yc=yc+ps.y;
                    }
                    xc=xc/pos.size();
                    yc=yc/pos.size();
                    position ps = new position();
                    ps.x=xc.intValue();
                    ps.y=yc.intValue();

                    posc.add(ps);

                cnt++;

                pr=cnt/(width*height*2);
                IJ.showProgress(pr);

            }
        }




    return (ArrayList <position>) posc;

}

public ArrayList <posdiam> FindDiam3(ArrayList <particle>  prt){


    ArrayList <position> pos = new ArrayList<position>();
    ArrayList <posdiam> posc = new ArrayList<posdiam>();
    ArrayList <posdiam> poso = new ArrayList<posdiam>();

    int p1, p2, c=0, f=0,k=0,idx=0;

    IJ.showProgress(0.0);

    double prg,cnt=0,d=0,pr_xc,pr_yc,dmx=0;

    for(int i=0;i<prt.size();i++){
        d=0;
        particle pr = new particle();
        pr=(particle) prt.get(i);
        pos = pr.pos;
        posdiam psd = new posdiam();
        psd.x=pr.xc;
        psd.y=pr.yc;
```

```
            for( k=0;k<pos.size();k++) {
                position ps = new position();
                ps = (position) pos.get(k);
                d=d+2*Math.sqrt((pr.xc-ps.x)*(pr.xc-ps.x)+(pr.yc-ps.y)*(pr.yc-
ps.y));
            }

            d=d/pos.size();
            psd.d=d;
            if(dmx<=d){
                dmx=d;
                idx=i;
            }
            double std=0;
            for( k=0;k<pos.size();k++) {
                position ps = new position();
                ps = (position) pos.get(k);
                std=std+Math.pow(2*Math.sqrt((pr.xc-ps.x)*(pr.xc-ps.x)+(pr.yc-
ps.y)*(pr.yc-ps.y))-d,2);
            }

            std=Math.sqrt(std/(pos.size()-1));
            psd.std=std;
            posc.add(psd);
        }

        posdiam psd = new posdiam();
        psd=(posdiam) posc.get(idx);
        poso.add(psd);

    return (ArrayList <posdiam>) poso;

    }

    public ArrayList <posdiam> FindDiam2(ArrayList <particle>  prt){


        ArrayList <position> pos = new ArrayList<position>();
        ArrayList <posdiam> posc = new ArrayList<posdiam>();

        int p1, p2, c=0, f=0,k=0;

        IJ.showProgress(0.0);

        double prg,cnt=0,d=0,pr_xc,pr_yc;

        for(int i=0;i<prt.size();i++){
            d=0;
            particle pr = new particle();
            pr=(particle) prt.get(i);
            pos = pr.pos;
            posdiam psd = new posdiam();
            psd.x=pr.xc;
            psd.y=pr.yc;
            for( k=0;k<pos.size();k++) {
                position ps = new position();
                ps = (position) pos.get(k);
                d=d+2*Math.sqrt((pr.xc-ps.x)*(pr.xc-ps.x)+(pr.yc-ps.y)*(pr.yc-
ps.y));
            }

            d=d/pos.size();
```

```
            psd.d=d;
            double std=0;
            for( k=0;k<pos.size();k++) {
                position ps = new position();
                ps = (position) pos.get(k);
                std=std+Math.pow(2*Math.sqrt((pr.xc-ps.x)*(pr.xc-ps.x)+(pr.yc-
ps.y)*(pr.yc-ps.y))-d,2);
            }

            std=Math.sqrt(std/(pos.size()-1));
            psd.std=std;
            posc.add(psd);
        }
    return (ArrayList <posdiam>) posc;

    }


    public ArrayList <posdiam> FindDiam(ImageProcessor ip){
    int width = ip.getWidth();
    int height = ip.getHeight();

        byte[] pix = (byte[])ip.getPixels();
        ArrayList <position> pos = new ArrayList<position>();
        ArrayList <posdiam> posc = new ArrayList<posdiam>();
        ArrayList <posdiam> posc2 = new ArrayList<posdiam>();
    ImageProcessor ip2=ip.duplicate();

        int p1, p2, c=0, f=0,k=0;


        IJ.showProgress(0.0);

        double pr,cnt=0,d=0,pr_xc,pr_yc;
        Double xc = new Double(0);
        Double yc = new Double(0);

            for(int x=1;x<width-1;x++){
                for(int y=1;y<height-1;y++){


                    d=0;

                    pos =   prtcPos(ip,pix,x,y);

                    k=0;
                    while(k<pos.size()){
                        position ps = new position();
                        ps = (position) pos.get(k);
                        pos= addPrtcPos(ip,pix,ps.x,ps.y,pos);
                        k++;
                        pix[ps.y*width+ps.x]=(byte) 128;
                    }
                    xc=(double) 0;
                    yc=(double) 0;

                        for( k=0;k<pos.size();k++) {
                        position ps = new position();
                        ps = (position) pos.get(k);
                        xc=xc+ps.x;
                        yc=yc+ps.y;
                        }
```

```
                        if(pos.size()!=0){
                            xc=xc/pos.size();
                            yc=yc/pos.size();
                            f=0;
                            k=0;
                            while( k<posc.size()) {
                                posdiam psd = new posdiam();
                                psd = (posdiam) posc.get(k);

if((psd.x==xc.intValue())&&(psd.y==yc.intValue())) f=1;

                                k++;
                            }


                            if(f==0){
                                posdiam psd = new posdiam();
                                psd.x=xc.intValue();
                                psd.y=yc.intValue();

                                    for( k=0;k<pos.size();k++) {
                                        position ps = new position();
                                        ps = (position) pos.get(k);
                                        d=d+2*Math.sqrt((xc-ps.x)*(xc-ps.x)+(yc-
ps.y)*(yc-ps.y));

                                    }

                                d=d/pos.size();
                                psd.d=d;
                                double std=0;

                                    for( k=0;k<pos.size();k++) {
                                        position ps = new position();
                                        ps = (position) pos.get(k);
                                        std=std+Math.pow(2*Math.sqrt((xc-
ps.x)*(xc-ps.x)+(yc-ps.y)*(yc-ps.y))-d,2);

                                    }

                                std=Math.sqrt(std/(pos.size()-1));
                                psd.std=std;
                                posc.add(psd);
                            }
                        }

                    cnt++;

                    pr=cnt/(width*height);
                    IJ.showProgress(pr);

                }
            }



        return (ArrayList <posdiam>) posc;

    }
```

```
public class posdiam {
    int x,y,n;
    double d,std;
}



public ArrayList getHistMax(ArrayList <hist> hst,int stp){
    ArrayList max;
    max= new ArrayList();

    int i,ct=0,j;

    for(i=0;i<hst.size();i++)  {
        hist bk= new hist();
            bk=(hist) hst.get(i);
        if (bk.y!=0)  ct++;
    }
    double[] bkp,xp;
    bkp= new double[ct];
    xp= new double[ct];

    i=0;
    for(j=0;j<hst.size();j++)  {
        hist bk= new hist();
            bk=(hist) hst.get(j);
        if (bk.y!=0){
            bkp[i]=bk.y;
            xp[i]=bk.x;
            i++;
        }
    }


    double mx=0;
    int c=0;

    for(i=0;i<ct;i++){
        if(mx<=bkp[i]){
            mx=bkp[i];
            c=i;
        }
        if(i-c>=stp){
            hist bk= new hist();
            bk.x=xp[c];
            bk.y=bkp[c];
            c=0;
            mx=0;
            max.add(bk);
        }

    }




    return(ArrayList) max;
}
```

```
    public ArrayList getHistogram(ImageProcessor ip){
        ArrayList hst;
        hst= new ArrayList();
        int width = ip.getWidth();
        int height = ip.getHeight();
        byte[] pixels = (byte[])ip.getPixels();
        double max=ip.getMax();
        //double binsz=max/255;
        int val=0;
        double c=0;


        int f=0;
        for(int j=0;j<256;j++){
            hist bk= new hist();
            bk.y=0;
            bk.x=j;
            hst.add(bk);

        }


        for(int i=0;i<width*height;i++){
                    val=pixels[i];

                hist bk= new hist();
                    bk=(hist)hst.get(val+1);

                bk.x=val;
                bk.y=bk.y+1;

                    hst.set(val,bk);
                    bk=(hist)hst.get(val);


        }


        return(ArrayList) hst;
    }

}
```

findDiameterSingleAllthresMethods_.java

```
import ij.*;
import ij.process.*;
import ij.measure.*;
import ij.gui.*;
import java.awt.*;
import java.math.*;
import java.io.*;
import java.lang.Number;
import java.lang.Byte;
import java.lang.Short;
import java.lang.Double;
import java.lang.Boolean;
import ij.plugin.*;
import ij.plugin.filter.*;
import ij.plugin.filter.GaussianBlur.*;
import ij.plugin.frame.*;
import ij.plugin.filter.PlugInFilter;
import ij.text.TextPanel;
import java.util.*;
import java.lang.Object.*;
import java.lang.reflect.Array;
import ij.plugin.filter.ParticleAnalyzer;
import ij.measure.Measurements;
import ij.plugin.filter.Analyzer;
import ij.measure.ResultsTable;
import ij.process.AutoThresholder;


public class findDiameterSingleAllthresMethods_ implements PlugInFilter {


    protected ImageProcessor image;

    public int setup(String arg, ImagePlus imp) {
        image=imp.getProcessor();
        if (arg.equals("about"))
            { return DONE; }
        return DOES_8G+SUPPORTS_MASKING;
    }

    public void run(ImageProcessor ipp) {

         GenericDialog gd = new GenericDialog("Find Diameter");

        double XYsc=0.115089;
        int psz=100,k=2;


        gd.addNumericField("Particle Size Filter: ", psz, 0);

        gd.addNumericField("XY scale (pixel*um): ", XYsc, 6);


        gd.showDialog();
        if (gd.wasCanceled()) return;

        psz = (int)gd.getNextNumber();
```

```
XYsc = (double)gd.getNextNumber();

ImageProcessor ApIp;
ApIp = image.duplicate();



ResultsTable rt = ResultsTable.getResultsTable();
rt.reset();
rt.incrementCounter();
rt.setPrecision(6);



int idx=0,f=0,c=0,ths;
double t;
AutoThresholder at = new AutoThresholder();
String[] mtd;
mtd=at.getMethods();


for (int i=0;i<mtd.length;i++){

    ImageProcessor nip;
    nip=image.duplicate();







    ths=at.getThreshold(mtd[i],nip.getHistogram());
    rt.addValue("th",ths);
    nip.threshold((int)ths);
    nip.invert();
    nip.setPixels(ParticleSizeFilter(nip,50));



    ImageProcessor nip2;
    nip2=nip.duplicate();
    nip2.setPixels(Edge(nip2));
    nip.setPixels(Diff(nip,nip2));

    nip.setPixels(FillHoles(nip));
    nip.setPixels(FillHoles(nip));
    nip.setPixels(FillHoles(nip));
    nip.setPixels(ParticleSizeFilter(nip,psz));

    ArrayList <posdiam> pos = new ArrayList<posdiam>();
    ArrayList <particle> prt = new ArrayList<particle>();

    nip2.setPixels(Edge(nip));
    prt=FindParticle(nip2);

    nip.setPixels(FillEdge(nip2,prt));

    nip2.setPixels(Edge(nip));
    prt=FindParticle(nip2);
```

```
            pos=FindDiam3(prt);


            int x=0,y=0;
            double dmx=0;



             for(k=0;k<pos.size();k++){
                posdiam ps = new posdiam();
                ps=pos.get(k);
                if((ps.x!=0)&&(ps.y!=0)){


                    rt.addValue(" Average Diameter",ps.d*XYsc);
                    rt.addValue(" Average Diameter stdev",ps.std*XYsc);
                    rt.addValue("XY Pixel size",XYsc);

                    if(ps.d!=0) f++;
                }
            }
            rt.addValue(mtd[i],i);
            rt.incrementCounter();

        }
        rt.show("Results");


}


    public class hist{
        double x,y;
        }



    public float[] mkGaussianKernel(int dim, double sigma) {
        float[] kernel;
        kernel = new float[dim*dim];

        float[] hg;
        hg = new float[dim*dim];

        float sm=0;
        int k;
        for(int i=0;i<dim;i++) {

            for(int j=0;j<dim;j++) {
            Float f = new Float(Math.exp((i*i+j*j)/(sigma*sigma)));
             hg[i+dim*j]=f;
            sm=sm+hg[i+dim*j];
            }
        }

        for(int i=0;i<dim;i++) {
            for(int j=0;j<dim;j++) {
             kernel[i+dim*j]=hg[i+dim*j]/sm;
            }
        }

    return (float[]) kernel;
```

```
    }


    public ArrayList prtcPos(ImageProcessor ip, byte[] pixels,int x, int y){
        int c=0;
        int width = ip.getWidth();
        int height = ip.getHeight();

        ArrayList pos;
        pos= new ArrayList();

        if(pixels[y*width+x]==0){

                        for(int i=-1;i<=1;i++){
                            for(int j=-1;j<=1;j++){

if((x+i>=0)&&(y+j>=0)&&(x+i<width)&&(y+j<height)){
                                        if(pixels[(y+j)*width+x+i]==0){
                                            position ps = new position();
                                            ps.x=x+i;
                                            ps.y=y+j;
                                            pos.add((position) ps);

                                        }

                                    }

                            }
                        }


        }



    return (ArrayList) pos;
    }


public ArrayList <position> addPrtcPos(ImageProcessor ip,byte[] pixels ,  int x,
int y,ArrayList <position>prev){

        int width = ip.getWidth();
        int height = ip.getHeight();


                    if(pixels[y*width+x]==0){

                        for(int i=-1;i<=1;i++){
                            for(int j=-1;j<=1;j++){

if((x+i>=0)&&(y+j>=0)&&(x+i<width)&&(y+j<height)){

                                        if(pixels[(y+j)*width+x+i]==0){
                                            position ps = new position();
                                            ps.x=x+i;
                                            ps.y=y+j;
                                            prev.add((position) ps);
                                        }

                                    }
```

```
                        }
                    }


                }


return (ArrayList <position>) prev;
}


public ArrayList <position> prtcPos4n(ImageProcessor ip, int x, int y){
    int c=0;
    int width = ip.getWidth();
    int height = ip.getHeight();
    byte[] pixels = (byte[])ip.getPixels();

    ArrayList <position> pos;
    pos= new ArrayList<position>();


                if(pixels[y*width+x]==0){

                    for(int i=-1;i<=1;i++){

                            if((x+i>=0)&&(i!=0)&&(x+i<width)&&(y<height)){
                                if(pixels[y*width+x+i]==0){
                                    position ps = new position();
                                    ps.x=x+i;
                                    ps.y=y;
                                    pos.add((position) ps);

                                    c++;
                                }

                            }


                    }

                    for(int j=-1;j<=1;j++){
                            if((j!=0)&&(y+j>=0)&&(x<width)&&(y+j<height)){
                                if(pixels[(y+j)*width+x]==0){
                                    position ps = new position();
                                    ps.x=x;
                                    ps.y=y+j;
                                    pos.add((position) ps);

                                    c++;
                                }

                            }

                    }



                }
```

```
return (ArrayList <position>) pos;
}


public ArrayList <position> prtcPos4nb(ImageProcessor ip, int x, int y){
    int c=0;
    int width = ip.getWidth();
    int height = ip.getHeight();
    byte[] pixels = (byte[])ip.getPixels();

    ArrayList <position> pos;
    pos= new ArrayList<position>();


                if(pixels[y*width+x]==255){

                    for(int i=-1;i<=1;i++){

                            if((x+i>=0)&&(i!=0)&&(x+i<width)&&(y<height)){
                                if(pixels[y*width+x+i]==0){
                                    position ps = new position();
                                    ps.x=x+i;
                                    ps.y=y;
                                    pos.add((position) ps);

                                    c++;
                                }

                            }


                    }

                    for(int j=-1;j<=1;j++){
                            if((j!=0)&&(y+j>=0)&&(x<width)&&(y+j<height)){
                                if(pixels[(y+j)*width+x]==0){
                                    position ps = new position();
                                    ps.x=x;
                                    ps.y=y+j;
                                    pos.add((position) ps);

                                    c++;
                                }

                            }

                    }


                }


return (ArrayList <position>) pos;
}




public class position {
```

```
    int x,y;
}


public byte[] ParticleSizeFilter(ImageProcessor ip,int size){
int width = ip.getWidth();
int height = ip.getHeight();

    byte[] pixels = (byte[])ip.getPixels();
    ArrayList <position> pos = new ArrayList<position>();
ImageProcessor ip2=ip.duplicate();
    byte[] pix2 = (byte[])ip2.getPixels();

    int p1, p2, c=0, f=0,k=0;



    double pr,cnt=0;

        for(int x=0;x<width;x++){
            for(int y=0;y<height;y++){

                if(pixels[y*width+x]==0){

                    pos =   prtcPos(ip,pixels,x,y);

                    k=0;

                    while(k<pos.size()){
                        position ps = new position();
                        ps = (position) pos.get(k);
                        pos= addPrtcPos(ip,pixels,ps.x,ps.y,pos);
                        k++;

                        pixels[ps.y*width+ps.x]=(byte) 128;

                    }

                    if(pos.size()<=size&&pos.size()>0) {

                        for( k=0;k<pos.size();k++) {
                            position ps = new position();
                            ps = (position) pos.get(k);

                            pix2[ps.y*width+ps.x]=(byte) 255;

                        }
                    }


                }
                cnt++;

                pr=cnt/(width*height);


            }
        }


    return (byte[]) pix2;
```

```
}


public byte[] CleanParticle(ImageProcessor ip,int n,int type){
int width = ip.getWidth();
int height = ip.getHeight();
    byte[] pix = (byte[])ip.getPixels();

    ArrayList <position> pos = new ArrayList<position>();

    int p1, p2, c=0, f=0,k=0;


    IJ.showProgress(0.0);

    double pr,cnt=0;

        for(int x=0;x<width;x++){
            for(int y=0;y<height;y++){

                if(type==0) pos =   prtcPos4n(ip,x,y);
                else pos =   prtcPos(ip,pix,x,y);

                if(n>=pos.size())   ip.set(x,y,255);



                cnt++;

                pr=cnt/(width*height*2);
                IJ.showProgress(pr);

            }
        }



    byte[] pixels = (byte[])ip.getPixels();
    return (byte[]) pixels;

}


public byte[] Erosion(ImageProcessor ip){
int width = ip.getWidth();
int height = ip.getHeight();
byte[] pix = (byte[])ip.getPixels();

    ArrayList <position> pos = new ArrayList<position>();

    int p1, p2, c=0, f=0,k=0;

    IJ.showProgress(0.0);

    double pr,cnt=0;

        for(int x=0;x<width;x++){
            for(int y=0;y<height;y++){
```

```
                    pos =   prtcPos(ip,pix,x,y);

                    if(5>=pos.size())   ip.set(x,y,0);



                    cnt++;

                    pr=cnt/(width*height);
                    IJ.showProgress(pr);

                }
            }



        byte[] pixels = (byte[])ip.getPixels();
        return (byte[]) pixels;

    }


    public byte[] Edge(ImageProcessor ip){

        ImageProcessor ip2;
        ip2=ip.duplicate();
        ip2.fill();
        ip2.invert();
        int width = ip.getWidth();
        int height = ip.getHeight();

        int p1,p2,p3;

            for(int i=0;i<height;i++) {
                for(int j=1;j<width-1;j++) {

                    p1=ip.get(j-1,i);
                    p2=ip.get(j,i);
                    p3=ip.get(j+1,i);

                    if((p1 == 0)&&(p2 == 255)&&(p3 == 255)) ip2.set(j-1,i,0);
                    if((p1 == 255)&&(p2 == 0)&&(p3 == 255)) ip2.set(j,i,0);
                    if((p1 == 255)&&(p2 == 255)&&(p3 == 0)) ip2.set(j+1,i,0);
                    if((p1 == 0)&&(p2 == 255)&&(p3 == 0)){
                        ip2.set(j-1,i,0);
                        ip2.set(j+1,i,0);
                    }


                }
            }

            for(int i=1;i<height-1;i++) {
                for(int j=0;j<width;j++) {
                    p1=ip.get(j,i-1);
                    p2=ip.get(j,i);
                    p3=ip.get(j,i+1);

                    if((p1 == 0)&&(p2 == 255)&&(p3 == 255)) ip2.set(j,i-1,0);
                    if((p1 == 255)&&(p2 == 0)&&(p3 == 255)) ip2.set(j,i,0);
                    if((p1 == 255)&&(p2 == 255)&&(p3 == 0)) ip2.set(j,i+1,0);
```

```
                if((p1 == 0)&&(p2 == 255)&&(p3 == 0)){
                    ip2.set(j,i-1,0);
                    ip2.set(j,i+1,0);
                }
            }
        }

    byte[] pixels = (byte[])ip2.getPixels();
    return (byte[]) pixels;
}

public byte[] Diff(ImageProcessor ip1,ImageProcessor ip2){

    byte[] pixels = (byte[])ip1.duplicate().getPixels();

    int width = ip1.getWidth();
    int height = ip1.getHeight();
    //byte[] pixels;
    int p1,p2;

        for(int i=0;i<height;i++) {
            for(int j=0;j<width;j++) {
                p1=ip1.get(j,i);
                p2=ip2.get(j,i);
            if(p2 == 0)     pixels[j+width*i]= (byte) 255;
            else    pixels[j+width*i]=(byte) p1;
            }
        }



    return (byte[]) pixels;
}

public byte[] FillHoles(ImageProcessor ip){
int width = ip.getWidth();
int height = ip.getHeight();

    ArrayList <position> pos = new ArrayList<position>();

    int p1, p2, c=0, f=0,k=0;

    IJ.showProgress(0.0);

    double pr,cnt=0;

        for(int x=0;x<width;x++){
            for(int y=0;y<height;y++){

                pos =   prtcPos4nb(ip,x,y);

                if(pos.size()>2)    ip.set(x,y,0);



                cnt++;

                pr=cnt/(width*height);
                IJ.showProgress(pr);

            }
```

```
        }


    byte[] pixels = (byte[])ip.getPixels();
    return (byte[]) pixels;

}


public byte[] FillEdge(ImageProcessor ip, ArrayList <particle> prt){
int width = ip.getWidth();
int height = ip.getHeight();

     ArrayList <position> pos = new ArrayList<position>();

ImageProcessor ip2=ip.duplicate();

    int p1, p2, f=0,k=0;

    IJ.showProgress(0.0);

    double prg,cnt=0;


        for(int x=0;x<width-1;x++){
            for(int y=0;y<height-1;y++){

                for(int i=0;i<prt.size();i++){

                    f=0;

                    particle pr = new particle();
                    pr= (particle) prt.get(i);
                    pos=pr.pos;


                    boolean a = (boolean) false;
                    boolean b = (boolean) false;
                    boolean c = (boolean) false;
                    boolean d = (boolean) false;

                    for( k=0;k<pos.size();k++) {
                        position ps2 = new position();
                        ps2 = (position) pos.get(k);

                        a=((x>=ps2.x)&&(x<=pr.xc));
                        b=((y>=ps2.y)&&(y<=pr.yc));
                        c=((x<=ps2.x)&&(x>=pr.xc));
                        d=((y<=ps2.y)&&(y>=pr.yc));

                        if((a&&b)||(a&&d)||(c&&b)||(c&&d)) f=1;

                    }

                    if(f==1)  ip2.putPixel(x,y,0);
                }

                cnt++;
```

```
                prg=cnt/(width*height*2);
                IJ.showProgress(prg);

            }
        }

    byte[] pixels = (byte[])ip2.getPixels();
    return (byte[]) pixels;

}


public class particle{
    ArrayList <position> pos;
    int xc, yc, n;
}

public ArrayList <particle> FindParticle(ImageProcessor ip){
int width = ip.getWidth();
int height = ip.getHeight();
    byte[] pix = (byte[])ip.getPixels();

    ArrayList <position> pos = new ArrayList<position>();
    ArrayList <particle> posc = new ArrayList<particle>();
ImageProcessor ip2=ip.duplicate();

    int p1, p2, c=0, f=0,k=0;

    IJ.showProgress(0.0);

    double pr,cnt=0;
    Double xc = new Double(0);
    Double yc = new Double(0);

        for(int x=1;x<width-1;x++){
            for(int y=1;y<height-1;y++){
                xc=(double) 0;
                yc=(double) 0;
                pos =   prtcPos(ip,pix,x,y);
                 k=0;

                while(k<pos.size()){
                    position ps = new position();
                    ps = (position) pos.get(k);
                    pos= addPrtcPos(ip,pix,ps.x,ps.y,pos);
                    k++;
                    pix[ps.y*width+ps.x]=(byte) 128;
                }


                    for( k=0;k<pos.size();k++) {
                    position ps = new position();
                    ps = (position) pos.get(k);
                    xc=xc+ps.x;
                    yc=yc+ps.y;
                    }
                    xc=xc/pos.size();
                    yc=yc/pos.size();
                    particle prt = new particle();
                    prt.xc=xc.intValue();
                    prt.yc=yc.intValue();
                    prt.pos=pos;
```

```
                    prt.n=posc.size();
                    k=0;
                    f=0;
                    while(k<posc.size()){
                        particle prt2 = new particle();
                        prt2=(particle) posc.get(k);
                        if((prt.xc==prt2.xc)&&(prt.yc==prt2.yc)) f=1;
                        k++;
                    }
                    if(f==0) posc.add(prt);

                cnt++;

                pr=cnt/(width*height);
                IJ.showProgress(pr);

            }
        }




    return (ArrayList <particle>) posc;

}

public ArrayList <position> FindCentre(ImageProcessor ip){
int width = ip.getWidth();
int height = ip.getHeight();
byte[] pix = (byte[])ip.getPixels();

    ArrayList <position> pos = new ArrayList<position>();
    ArrayList <position> posc = new ArrayList<position>();
ImageProcessor ip2=ip.duplicate();

    int p1, p2, c=0, f=0,k=0;

    IJ.showProgress(0.0);

    double pr,cnt=0;
    Double xc = new Double(0);
    Double yc = new Double(0);

        for(int x=1;x<width-1;x++){
            for(int y=1;y<height-1;y++){
                xc=(double) 0;
                yc=(double) 0;
                pos =   prtcPos(ip,pix,x,y);
                 k=0;

                while(k<pos.size()){
                    position ps = new position();
                    ps = (position) pos.get(k);
                    pos= addPrtcPos(ip,pix,ps.x,ps.y,pos);
                    k++;
                    pix[ps.y*width+ps.x]=(byte) 128;
                }


                    for( k=0;k<pos.size();k++) {
                    position ps = new position();
```

```
                        ps = (position) pos.get(k);
                        xc=xc+ps.x;
                        yc=yc+ps.y;
                        }
                        xc=xc/pos.size();
                        yc=yc/pos.size();
                        position ps = new position();
                        ps.x=xc.intValue();
                        ps.y=yc.intValue();

                        posc.add(ps);

                    cnt++;

                    pr=cnt/(width*height*2);
                    IJ.showProgress(pr);

                }
            }




        return (ArrayList <position>) posc;

    }

    public ArrayList <posdiam> FindDiam3(ArrayList <particle>  prt){


        ArrayList <position> pos = new ArrayList<position>();
        ArrayList <posdiam> posc = new ArrayList<posdiam>();
        ArrayList <posdiam> poso = new ArrayList<posdiam>();

        int p1, p2, c=0, f=0,k=0,idx=0;

        IJ.showProgress(0.0);

        double prg,cnt=0,d=0,pr_xc,pr_yc,dmx=0;

        for(int i=0;i<prt.size();i++){
            d=0;
            particle pr = new particle();
            pr=(particle) prt.get(i);
            pos = pr.pos;
            posdiam psd = new posdiam();
            psd.x=pr.xc;
            psd.y=pr.yc;
            for( k=0;k<pos.size();k++) {
                position ps = new position();
                ps = (position) pos.get(k);
                d=d+2*Math.sqrt((pr.xc-ps.x)*(pr.xc-ps.x)+(pr.yc-ps.y)*(pr.yc-
ps.y));
            }

            d=d/pos.size();
            psd.d=d;
            if(dmx<=d){
                dmx=d;
                idx=i;
            }
```

```
            double std=0;
            for( k=0;k<pos.size();k++) {
                position ps = new position();
                ps = (position) pos.get(k);
                std=std+Math.pow(2*Math.sqrt((pr.xc-ps.x)*(pr.xc-ps.x)+(pr.yc-
ps.y)*(pr.yc-ps.y))-d,2);
            }

            std=Math.sqrt(std/(pos.size()-1));
            psd.std=std;
            posc.add(psd);
        }

        posdiam psd = new posdiam();
        psd=(posdiam) posc.get(idx);
        poso.add(psd);

    return (ArrayList <posdiam>) poso;

    }

    public ArrayList <posdiam> FindDiam2(ArrayList <particle>  prt){


        ArrayList <position> pos = new ArrayList<position>();
        ArrayList <posdiam> posc = new ArrayList<posdiam>();

        int p1, p2, c=0, f=0,k=0;

        IJ.showProgress(0.0);

        double prg,cnt=0,d=0,pr_xc,pr_yc;

        for(int i=0;i<prt.size();i++){
            d=0;
            particle pr = new particle();
            pr=(particle) prt.get(i);
            pos = pr.pos;
            posdiam psd = new posdiam();
            psd.x=pr.xc;
            psd.y=pr.yc;
            for( k=0;k<pos.size();k++) {
                position ps = new position();
                ps = (position) pos.get(k);
                d=d+2*Math.sqrt((pr.xc-ps.x)*(pr.xc-ps.x)+(pr.yc-ps.y)*(pr.yc-
ps.y));
            }

            d=d/pos.size();
            psd.d=d;
            double std=0;
            for( k=0;k<pos.size();k++) {
                position ps = new position();
                ps = (position) pos.get(k);
                std=std+Math.pow(2*Math.sqrt((pr.xc-ps.x)*(pr.xc-ps.x)+(pr.yc-
ps.y)*(pr.yc-ps.y))-d,2);
            }

            std=Math.sqrt(std/(pos.size()-1));
            psd.std=std;
            posc.add(psd);
        }
```

```
    return (ArrayList <posdiam>) posc;

    }


    public ArrayList <posdiam> FindDiam(ImageProcessor ip){
    int width = ip.getWidth();
    int height = ip.getHeight();

        byte[] pix = (byte[])ip.getPixels();
        ArrayList <position> pos = new ArrayList<position>();
        ArrayList <posdiam> posc = new ArrayList<posdiam>();
        ArrayList <posdiam> posc2 = new ArrayList<posdiam>();
    ImageProcessor ip2=ip.duplicate();

        int p1, p2, c=0, f=0,k=0;


        IJ.showProgress(0.0);

        double pr,cnt=0,d=0,pr_xc,pr_yc;
        Double xc = new Double(0);
        Double yc = new Double(0);

            for(int x=1;x<width-1;x++){
                for(int y=1;y<height-1;y++){


                    d=0;

                    pos =   prtcPos(ip,pix,x,y);

                    k=0;
                    while(k<pos.size()){
                        position ps = new position();
                        ps = (position) pos.get(k);
                        pos= addPrtcPos(ip,pix,ps.x,ps.y,pos);
                        k++;
                        pix[ps.y*width+ps.x]=(byte) 128;
                    }
                    xc=(double) 0;
                    yc=(double) 0;

                        for( k=0;k<pos.size();k++) {
                        position ps = new position();
                        ps = (position) pos.get(k);
                        xc=xc+ps.x;
                        yc=yc+ps.y;
                        }
                        if(pos.size()!=0){
                            xc=xc/pos.size();
                            yc=yc/pos.size();
                            f=0;
                            k=0;
                            while( k<posc.size()) {
                                posdiam psd = new posdiam();
                                psd = (posdiam) posc.get(k);

if((psd.x==xc.intValue())&&(psd.y==yc.intValue())) f=1;

                                k++;
                            }
```

```
                                if(f==0){
                                    posdiam psd = new posdiam();
                                    psd.x=xc.intValue();
                                    psd.y=yc.intValue();

                                        for( k=0;k<pos.size();k++) {
                                            position ps = new position();
                                            ps = (position) pos.get(k);
                                            d=d+2*Math.sqrt((xc-ps.x)*(xc-ps.x)+(yc-
ps.y)*(yc-ps.y));

                                        }

                                    d=d/pos.size();
                                    psd.d=d;
                                    double std=0;

                                        for( k=0;k<pos.size();k++) {
                                            position ps = new position();
                                            ps = (position) pos.get(k);
                                            std=std+Math.pow(2*Math.sqrt((xc-
ps.x)*(xc-ps.x)+(yc-ps.y)*(yc-ps.y))-d,2);

                                        }

                                    std=Math.sqrt(std/(pos.size()-1));
                                    psd.std=std;
                                    posc.add(psd);
                                }
                            }

                        cnt++;

                        pr=cnt/(width*height);
                        IJ.showProgress(pr);

                    }
                }




        return (ArrayList <posdiam>) posc;

    }


    public class posdiam {
        int x,y,n;
        double d,std;
    }



    public ArrayList getHistMax(ArrayList <hist> hst,int stp){
        ArrayList max;
        max= new ArrayList();
```

```
        int i,ct=0,j;

        for(i=0;i<hst.size();i++)  {
            hist bk= new hist();
                bk=(hist) hst.get(i);
            if (bk.y!=0)  ct++;
        }
        double[] bkp,xp;
        bkp= new double[ct];
        xp= new double[ct];

        i=0;
        for(j=0;j<hst.size();j++)  {
            hist bk= new hist();
                bk=(hist) hst.get(j);
            if (bk.y!=0){
                bkp[i]=bk.y;
                xp[i]=bk.x;
                i++;
            }
        }


        double mx=0;
        int c=0;

        for(i=0;i<ct;i++){
            if(mx<=bkp[i]){
                mx=bkp[i];
                c=i;
            }
            if(i-c>=stp){
                hist bk= new hist();
                bk.x=xp[c];
                bk.y=bkp[c];
                c=0;
                mx=0;
                max.add(bk);
            }

        }




        return(ArrayList) max;
    }



    public ArrayList getHistogram(ImageProcessor ip){
        ArrayList hst;
        hst= new ArrayList();
        int width = ip.getWidth();
        int height = ip.getHeight();
        byte[] pixels = (byte[])ip.getPixels();
        double max=ip.getMax();

        int val=0;
        double c=0;
```

```
        int f=0;
        for(int j=0;j<256;j++){
            hist bk= new hist();
            bk.y=0;
            bk.x=j;
            hst.add(bk);

        }


        for(int i=0;i<width*height;i++){
                    val=pixels[i];

                hist bk= new hist();
                    bk=(hist)hst.get(val+1);

                bk.x=val;
                bk.y=bk.y+1;

                    hst.set(val,bk);
                    bk=(hist)hst.get(val);


        }

        return(ArrayList) hst;
    }

}
```