Journal of open research software

Hughes, A 2014 pyparty: Blob Detection, Drawing and Manipulation in Python. *Journal of Open Research Software,* 2: e26, DOI: http://dx.doi.org/10.5334/jors.bh

SOFTWARE METAPAPER

pyparty: Blob Detection, Drawing and Manipulation in Python

Adam Hughes[1]

[1] Department of Physics, The George Washington University, USA

pyparty complements image processing workflows by providing utilities for drawing, manipulating and quantifying blob features in 2d-images. pyparty is built around the scientific Python stack, designed especially for compatibility with scikit-image. After a brief introduction, characterization and construction of faux nanoparticle images are provided. The source, documentation and more examples are available at https://github.com/hugadams/pyparty. pyparty is released under an open BSD 3-Clause License and support is available through pyparty@googlegroups.com

Keywords: image processing; images; nanoparticles; python; particle analysis; scipy; scikit-image
Funding Statement: Tupported in part by the George Gamow Research and Luther Rice Collaborative Research fellowship programs, as well as the George Washington University (GWU) Knox fellowship.

(1) Overview

Introduction

Open-source image processing tools are rapidly spreading across programming languages and scientific domains. The widely-used ImageJ[1] platform, along with its distributions such as Fiji[2], offer an excellent Java-based suite. In the last few years, image processing and computer vision in Python has rapidly matured[1] with the emergence of libraries like scikit-image[3], Mahotas[4], and SimpleCV[5]. scikit-image and Mahotas utilize 2d NumPy[6] arrays for their image data structures, thus providing a familiar API to users of NumPy and the other core SciPy libraries, especially Matplotlib[7], IPython[8].

The motivation for pyparty arose from difficulties encountered when quantifying nanoparticle distributions on silica substrates. We had found that after acquiring an image, preprocessing, segmentation and labelling techniques were easily applied; however, separating and measuring the various particle species in post-processing was difficult. This is primarily because the objects of interest were stored as boolean or integer-labeled arrays, which become unwieldy under heavy manipulation.

pyparty emerged as a means to abstract the concept of particles (i.e. image blobs) into custom data structures for intuitive manipulation and characterization whilst preserving the image API. In addition to integrating new particle constructs with the existing array and image processing functions, pyparty extends scikit-image's rasterization toolbox with new particle types, patterning, and an interface to Matplotlib patch objects for vectorized particle renderings. Thus, pyparty leverages the conventional imaging pipeline at both ends; it provides a tool set for artificial image composition, and a framework for particle post-processing.

Implementation and architecture

Rasterization, as well as particle labels and descriptors (e.g. area, eccentricity etc...), are all deferred to algorithms implemented in scikit-image[14-16]. pyparty adds a convenience layer to extend some of this functionality. For example, pyparty's implements a ***Grid*** object for patterning, and provides a simple framework for drawing multi-particles, for example circle dimers and trimers.

The ***Canvas*** is pyparty's primary datastructure. The Canvas is comprised of a background image, a ***Grid*** and ***ParticleManager***, which is a container for particle storage and manipulation. The Canvas fully decouples particle information such as position, color, and pixel from the background image. Since particle information is stored in python containers, slicing, arithmetic, boolean indexing and other common operations are very easy to perform. For example, an operation like dilate all circular particles in an image that are over a certain size and mean brightness is a simple task in pyparty. In addition to the Canvas, the ***MultiCanvas*** is provided for images with multiple particle types.

Finally, pyparty simplifies some common image processing tasks involving thresholding, artificial noise generation, color assignment, image type conversions, plotting and blob filtering, albeit not to the extent of IJBlob[9].

Examples of Use

We will first highlight pyparty's drawing capabilities to create pseudo electron microscope images. A more involved version is used to compare segmentation algorithms in a

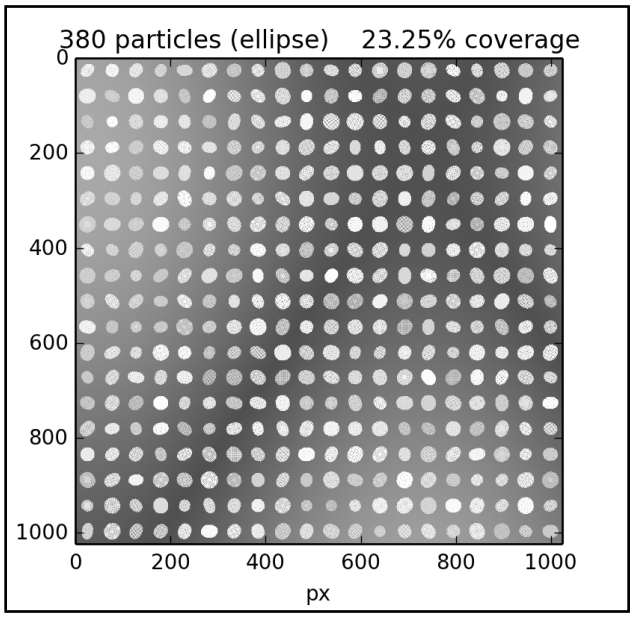

Figure 1: Randomly sized ellipses drawn using pyparty's Canvas and gridding.

corollary study[10]. In the example code snippets, Canvas and MultiCanvas methods are bolded for clarity. To begin, let us define the parameters corresponding to ranges for particle size and intensity, the noise fill-fraction and the image resolution. For brevity, we load a previously generated pyparty background image (see documentation) of a trigonometrically-varying contrast gradient.

```
from pyparty import Canvas, MultiCanvas

BG   =   'https://raw.github.com/hugadams...'
REZ =   (1024, 1024)     #Image resolution
RAD =   (12, 18)         #Radius range (px)
COLOR = (200, 255)       #Color range
NOISE = 0.10             #Percent noise
```

The Canvas crops the image to our desired resolution upon initialization. Next, randomly-sized ellipses are added, and the Canvas grid is set to 20 pixels per division; this controls the inter-particle spacing.

```
from random import randint as R_int

cnvs = Canvas(rez=REZ, background=BG)
cnvs.grid.div = 20 #20x20 grid

for (cx, cy) in cnvs.gpairs('centers'):
    cnvs.add('ellipse',
          center = (cx,cy),
          xradius = R_int(*RAD),
          yradius = R_int(*RAD),
          phi = R_int(0, 360),
          color = R_int(*COLOR) )

cnvs.show(annotate=True)
```

This results in 380 particles covering 23% of the image surface as shown in **Figure 1**. Next, we apply a Gaussian filter through scikit-image (σ = 3px) to smooth the particle boundaries. This negligibly affects the background,

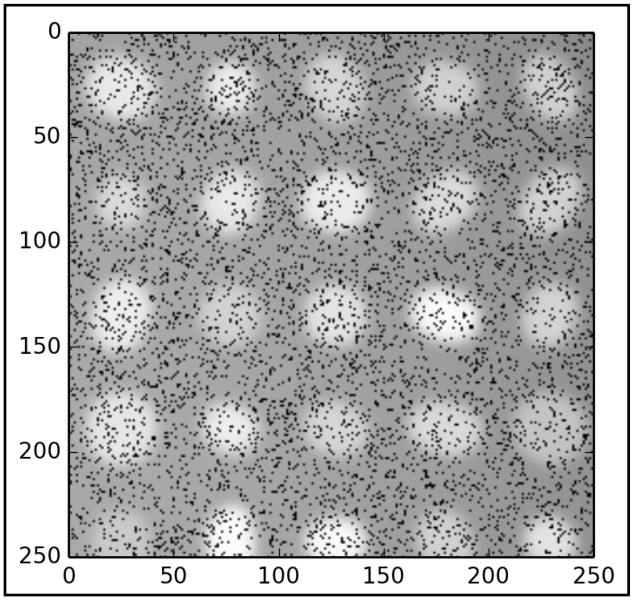

Figure 2: Zoomed view of particles after smoothing and adding black noise.

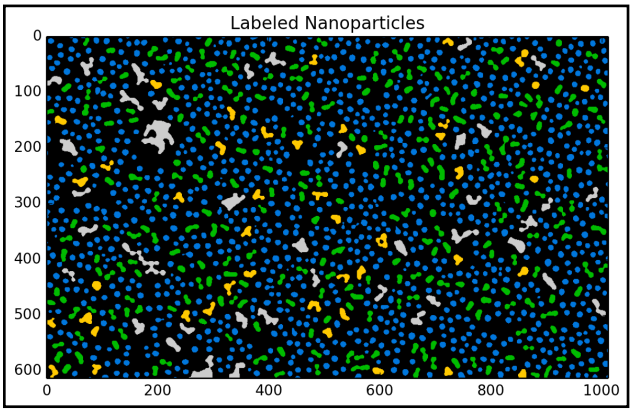

Figure 3: Bundled image of gold nanoparticles on a glass surface, grouped into four color-coded categories: singles, dimers, trimers and clusters.

which varies over a much larger scale. In pyparty, spectral noise can be assigned according to any NumPy distribution function (Gaussian, gamma etc..), but for clearer visualization, we'll use black pixels, also known as "pepper" noise (see **Figure 2**).

```
from skimage.filter import gaussian_filter
from pyparty.noise import pepper
from pyparty.plots import zoom

smooth = gaussian_filter(cnvs.image, 3)
noisy = pepper(smooth, NOISE)
zoom(noisy, (0, 0, 250, 250))
```

As a final exercise, we employ the MultiCanvas to characterize nanoparticle species in an electron microscope image. The image, which is packaged with pyparty, has been segmented using Ilastik's[11] object classification workflow into groups labelled as singles, dimers, trimers and clusters as shown in **Figure 3**. pyparty has an API for shape filtering, but it is not explored here. The MultiCanvas

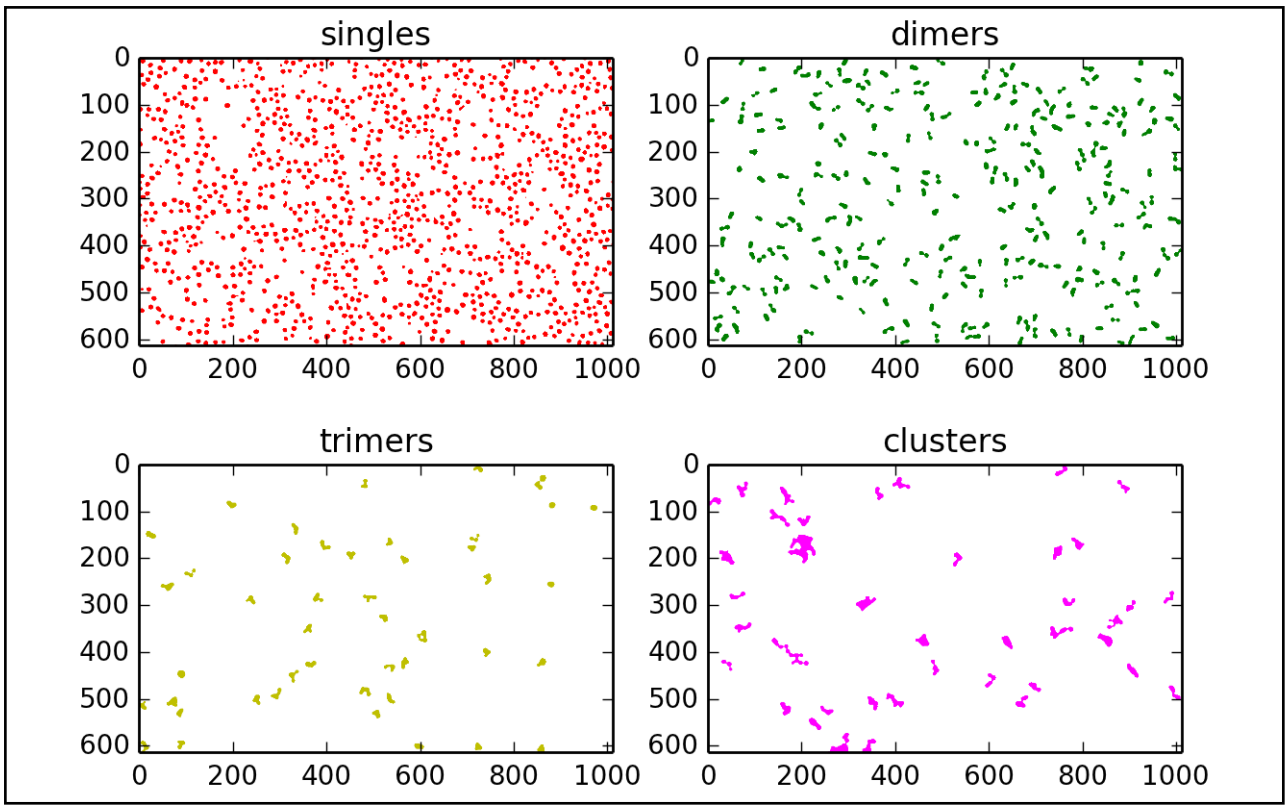

Figure 4: Separation of particle families based on color.

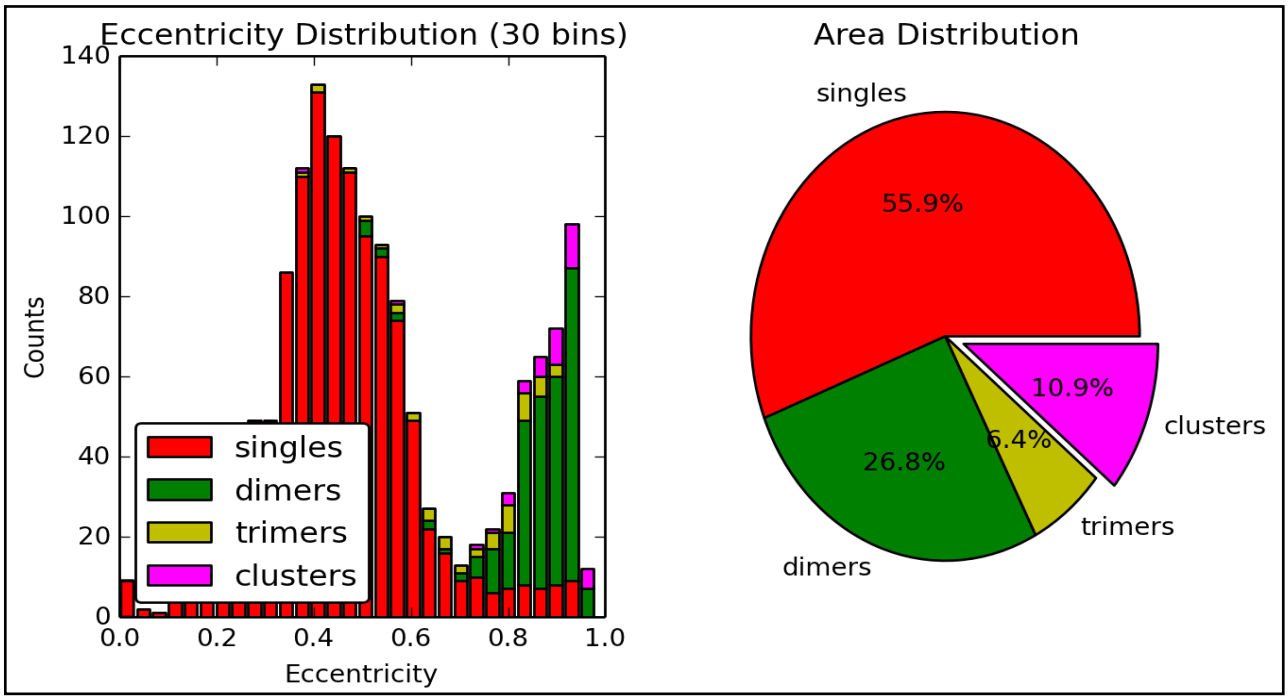

Figure 5: Eccentricity and area distribution of families of nanoparticles.

is built directly from the image via the from_labels() constructor. Persistent names and colors are assigned to the labels for easier identification between plots. **Figures 4** and **Figure 5** shows pyparty's decomposition of the image into these colored particle groups.

```
from pyparty.plots import showim, splot
from pyparty.data import nanolabels

NANOLABELS = nanolabels()
NAMES = ('singles', 'dimers', 'trimers',
'clusters')
showim(NANOLABELS, 'spectral', title='Labeled
Nanoparticles')

mc = MultiCanvas.from_labeled(nanolabels(),
*NAMES)
mc.set_colors('r', 'g', 'y', 'magenta')
mc.show(names = True, ncols=2, figsize=(7,5))

ax1, ax2 = splot(1,2)
mc.hist(ax1, attr='eccentricity', bins=30)
mc.pie(ax2, attr='area', explode=(0,0,0, 0.1))
```

Quality control
Development and testing is performed on Mac, Linux and Windows 7 systems. Tutorials and case studies are provided as IPython notebooks, which are also integrated into a quasi regression test suite.

(2) Availability
Operating system
pyparty runs on operating systems that support Python and the numpy stack. It has been tested on Ubuntu 10.04, 12.04, 13.10; Mac OS X 10.7, 10.8, and Windows 7.

Programming language
Python 2.7; some features of pyparty may be inaccessible on older versions. pyparty has not been tested for Python 3 compatibility.

Additional system requirements
At least 4GB RAM is suggested for working with multiple high-resolution images.

Dependencies
scikit-image, Matplotlib, Traits. IPython[2] is required to run notebooks; static versions are provided as well. We recommend Enthought Canopy for a powerful scientific computing environment with intrinsic notebook support.

List of contributors
Adam Hughes, Zhaowen Liu

Archive
Name
pyparty

Persistent identifier
DOI: 10.5281/zenodo.11194

License
BSD 3-Clause License

Publisher
Zenodo

Date published
08/05/2014 (v 0.3.1)

Code Repository
Name
Github

Identifier
https://github.com/hugadams/pyparty

License
Revised BSD

Date published
Fall 2013; some legacy code hosted since 2011.

(3) Reuse potential
Although developed in the context of nanotechnology, pyparty's post-processing and drawing capabilities are general to 2D imaging workflows, especially those found in microscopy[2]. pyparty is currently used for in-house research pursuits[13] and continues to develop. New applications like Ilastik integrate semi-supervised learning and object classification tools with unprecedented accessibility. This trend suggests that the need for general particle analysis tools will increase in coming years. As image analysis in Python continually expands to meet the needs of researchers, we believe pyparty will emerge as a small but important component in particle analysis workflows.

Acknowledgements
Thank you Annie Matsko for excellent revision suggestions. I'd also like to thank the scikit-image team for several helpful discussions that ultimately inspired this project, and for their exhaustive efforts in creating a superb library.

Notes
[1] SciPy's ndimage collection pre-dates scikit-image for example.

[2] Cell profiler[12] is a great example of a specialized microscopy application in Python.

References
1. **Schneider, C A, Rasband, W S** and **Eliceiri, K** 2012 NIH Image to ImageJ: 25 Years of Image Analysis. *Nature Methods: Focus on Bioimage Informatics*, 9(7), 671–675. DOI: http://dx.doi.org/10.1038/nmeth.2089
2. **Schindelin, J, Arganda-Carreras, I, Frise, E, Kaynig, V, Longair, M, Pietzsch, T, Preibisch, S, Rueden, C, Saalfeld, S, Schmid, B, Yinevez, J Y, James, W D, Hartenstein, V, Eliceiri, K, Pavel, T** and **Cardona, A** 2012 Fiji: An Open-source Platform for Biological-Image Analysis. *Nature Methods*, 9(7), 676–682. DOI: http://dx.doi.org/10.1038/nmeth.2019
3. **van der Walt S, Schönberger J L, Nunez-Iglesias J, Boulogne F** and **Warner J D** 2014 scikit-image: Image Processing in Python. *PeerJ PrePrints*, 2: e336v2. DOI: http://dx.doi.org/10.7287/peerj.preprints.336v2.
4. **Coelho, L P** 2013 Mahotas : Open Source Software for Scriptable Computer Vision. *Journal of Open Research Software*, 1, 1-7.
5. **Demaagd, K, Oliver, A, Oostendorp, N** and **Scott, K** 2012 Practical Computer Vision with SimpleCV: The Simple Way to Make Technology. *O'Reily Media, Inc.*
6. **Oliphant, T E** 2007 Python for Scientific Computing. *Computing in Science & Engineering*, 9(90). Available at http://numpy.org.
7. **Hunter, J D** 2007 Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(90). Available at http://matplotlib.org.
8. **Perez, F** and **Granger, B** 2007 IPython: a System for Interactive Scientific Computing. *Computing in Science*

and Engineering, 9(3), 21-29. Available at http://ipython.org. DOI: http://dx.doi.org/10.1109/MCSE.2007.53

9. **Wagner, T** and **Lipinski, H** 2013 IJBlob : An ImageJ Library for Connected Component Analysis and Shape Analysis. *Journal of Open Research Software*, 1(6), 6-8. DOI: http://dx.doi.org/10.5334/jors.ae.
10. **Hughes, A** and **Liu, Z** 2014 A Guide to Imaging Gold Nanoparticles. *Manuscript Submitted for Publication*.
11. **Sommer, C, Straehle, C, Köthe, U** and **Hamprecht, F** 2011 ilastik: Interactive Learning and Segmentation Toolkit. In: *8th IEEE International Symposium on Biomedial Imaging (ISBI)*.
12. **Jones, T, Kang, I H, Wheeler, D, Lindquist, R, Papallo, A, Sabatini, D, Golland, P** and **Carpenter, A** 2008 CellProfiler Analyst: Data Exploration and Analysis Software for Complex Image-based Screens. *IBMC Bioinformatics*, 9(482).
13. **Reeves Lab** 2014 SEM Imaging of Nanoparticles: A case study. Available at https://github.com/hugadams/imgproc_supplemental.
14. **Burger, W** and **Buege, M** 2009 *Principles of Digital Image Processing: Core Algorithms*. London: Springer-Verlag.
15. **Jähne, B** 2005 *Digital Image Processing*. 6th ed. Berlin-Heidelberg: Springer-Verlag.
16. **Reiss, T H** 1993 Recognizing Planar Objects Using Invariant Image Features. In: *Lecture Notes in Computer Science*. Berlin: Springer.
17. **Hughes, A** 2014 Github Example. Available at http://nbviewer.ipython.org/github/hugadams/pyparty/blob/master/examples/Notebooks/JORS_data.ipynb?create=1 [Last accessed 5 August 2014].

How to cite this article: Hughes, A 2014 pyparty: Blob Detection, Drawing and Manipulation in Python. *Journal of Open Research Software* 2:e26, DOI: http://dx.doi.org/10.5334/jors.bh

Published: 23 September 2014

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press

OPEN ACCESS