

CLIJ: GPU-accelerated image processing for everyone

Robert Haase

CASUS, Feb 4th 2021

Acknowledgements



Akanksha Jain
(Treutlein lab)
@jain_akanksha_



Dani Vorkel
(Myers lab)
@happifocus



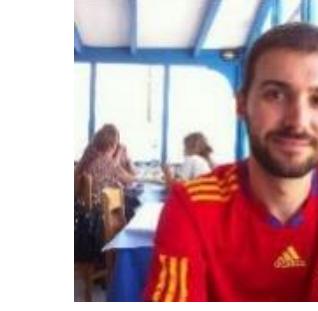
Theresa Suckert
(Krause lab,
Oncoray)



Pradeep Rajasekhar
(Poole lab, Monash U)



Talley J. Lambert
(HMS)
@pr4deepr



Juan Nunez-Iglesias
(Monash)
@jnuneziglesias



Pavel Tomancak
(CSBD / MPI-CBG)
@PavelTomancak



Gene Myers
(CSBD / MPI-CBG)
@TheGeneMyers



Stéphane Rigaud
(Institut Pasteur)
@StRigaud



Jean-Yves Tinevez
(Institut Pasteur)
@jytinevez



Florian Jug
(CSBD / MPI-CBG)
@florianjug



Szabolcs Horvát
(Modes lab)
@szhorvat



Johannes Girstmair
(Tomancak lab)
@jogirstmair



Brian Northon
@truenorth_ia



Matthias Arzt
(Myers/Jug lab)

Alex Herbert (University of Sussex)

Alexandr Dibrov (MPI CBG)

Bertrand Vernay (IGBMC, Strasbourg)

Bert Nitzsche (PoL TU Dresden)

Bradley Lowekamp (NIAID Washington)

Bram van den Broek (Netherlands

Cancer Institute)

Brenton Cavanagh (RCSI)

Bruno C. Vellutini (MPI CBG)

Carl D. Modes (CSBD/MPI-CBG)

Curtis Rueden (UW-Madison LOCI)

Damir Krunic (DKFZ)

Daniel J. White (GE)

Deborah Schmidt (CSBD/MPI CBG)

Douglas P. Shepard (Univ Arizona)

Eduardo Conde-Sousa (Univ. of Porto)

Erick Ratamero (The Jackson Laboratory)

Gaby G. Martins (IGC)

Gayathri Nadar (MPI CBG)

Guillaume Witz (Bern University)

Giovanni Cardone (MPI Biochem)

Irene Seijo Barandiaran (MPI CBG

Dresden)

Jan Brocher (Biovoxxel)

Juergen Gluch (Fraunhofer IKTS)

Kisha Sivanathan (Harvard Medical

School)

Kota Miura

Laurent Thomas (Acquifer)

Lior Pytowski (University of Oxford)

Loic A. Royer (CZ Biohub)

Marion Louveaux (Institut Pasteur Paris)

Martin Weigert (EPFL Lausanne)

Matthew Foley (University of Sydney)

Nico Stuurman (UCSF)

Nicola Maghelli (MPI CBG)

Nicolas Brouilly (IBDM Marseille)

Nik Cordes (Los Alamos National

Laboratory)

Noreen Walker (MPI CBG Dresden)

Ofra Golani (Weizmann Institute of

Science)

Patrick Dummer

Peter Haub

Pete Bankhead (University of

Edinburgh)

Peter Steinbach (HZDR Dresden)

Pit Kludig

Rita Fernandes (University of Porto)

Romain Guiet (BIOP-EPFL)

Ruth Whelan-Jeans

Sebastian Munck (VIB Leuven)

Siân Culley (MRC LMCB)

Stein Rørvik

Stéphane Dallongeville (Pasteur)

Tanner Fader (UNC-Chapel Hill)

Thomas Irmer (Zeiss)

Tobias Pietzsch (MPI-CBG)

Uwe Schmidt (MPI CBG)

Wilson Adams (VU Biophotonics)

MPI CBG Core Facilities

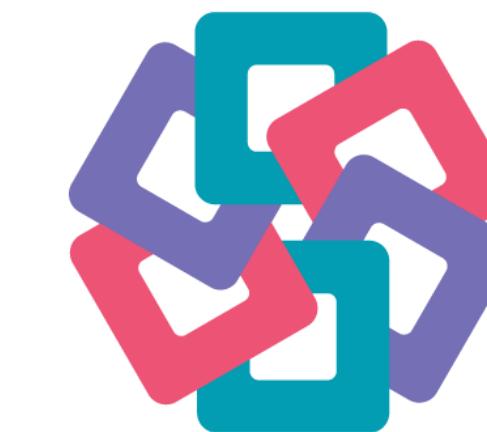
- Advanced Imaging Facility
- Light Microscopy Facility
- Scientific Computing
- IT Department
- Fly lab



<https://fiji.sc>



<https://napari.dev>



<https://image.sc>



<http://eubias.org/>
NEUBIAS/

Funding



R.H. was supported by the German Federal Ministry of Research and Education (BMBF) under the code 031L0044 (Sysbio II) and by the Deutsche Forschungsgemeinschaft under Germany's Excellence Strategy - EXC-2068 Cluster of Excellence Physics of Life of TU Dresden.

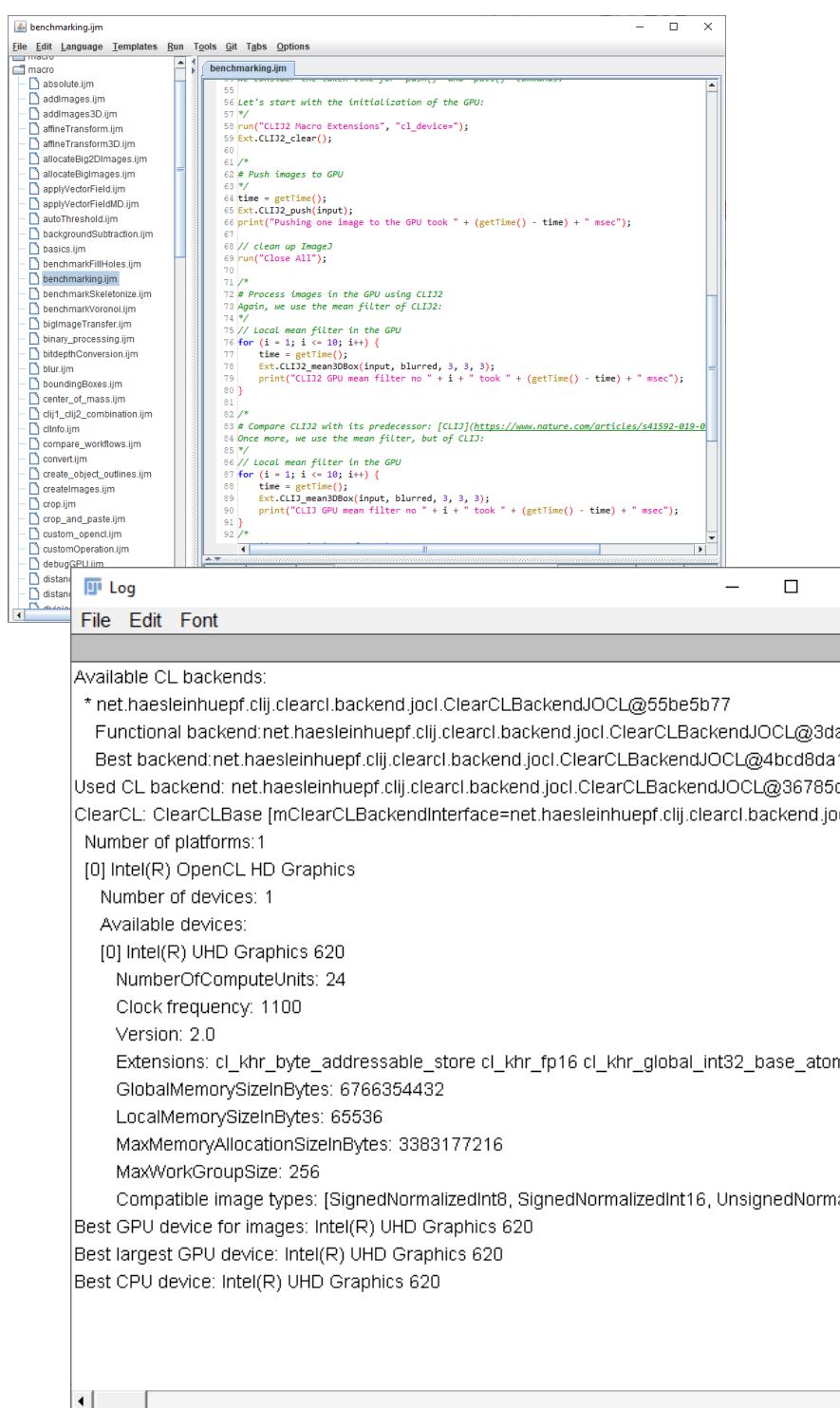


DRESDEN
concept

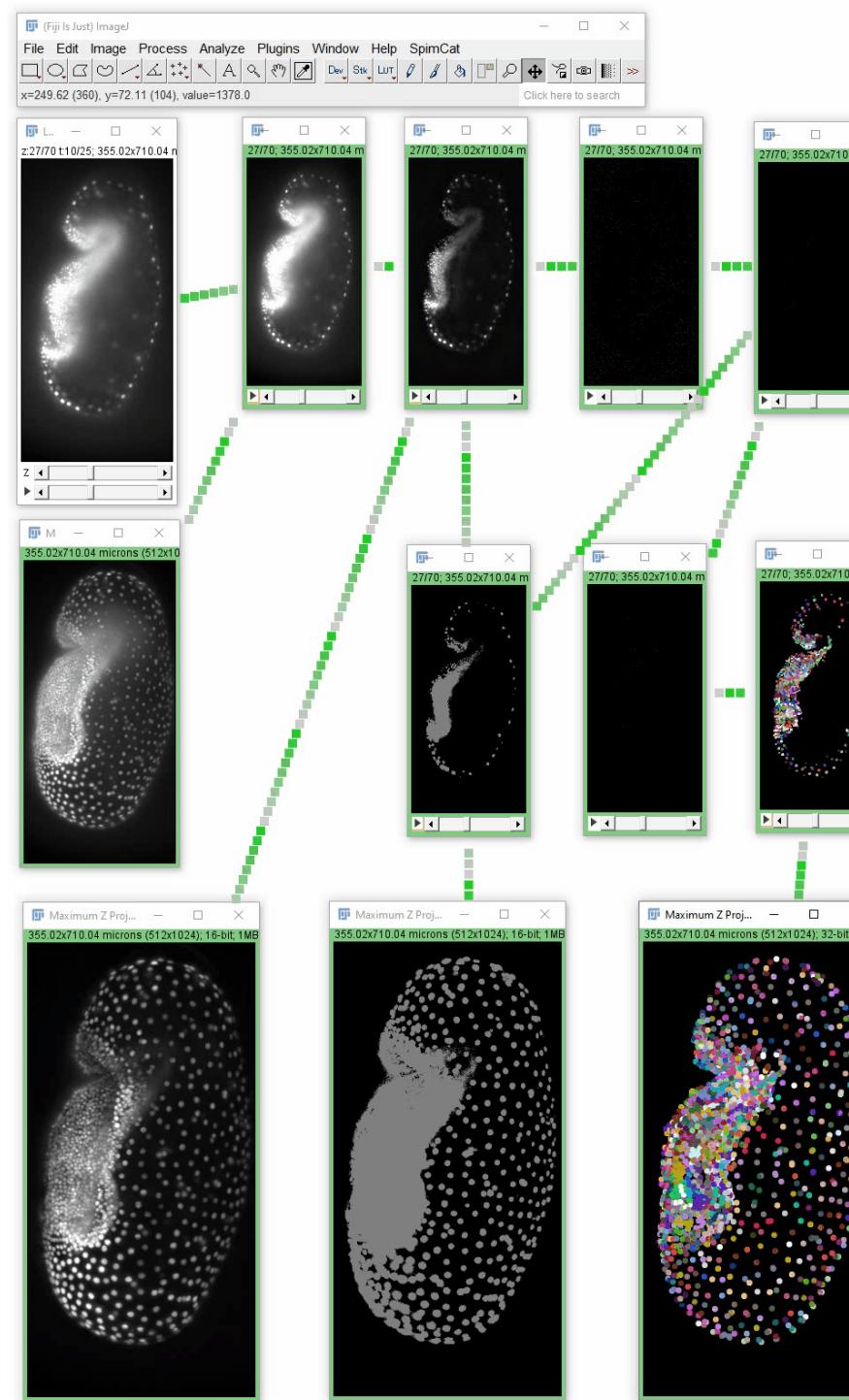


Overview

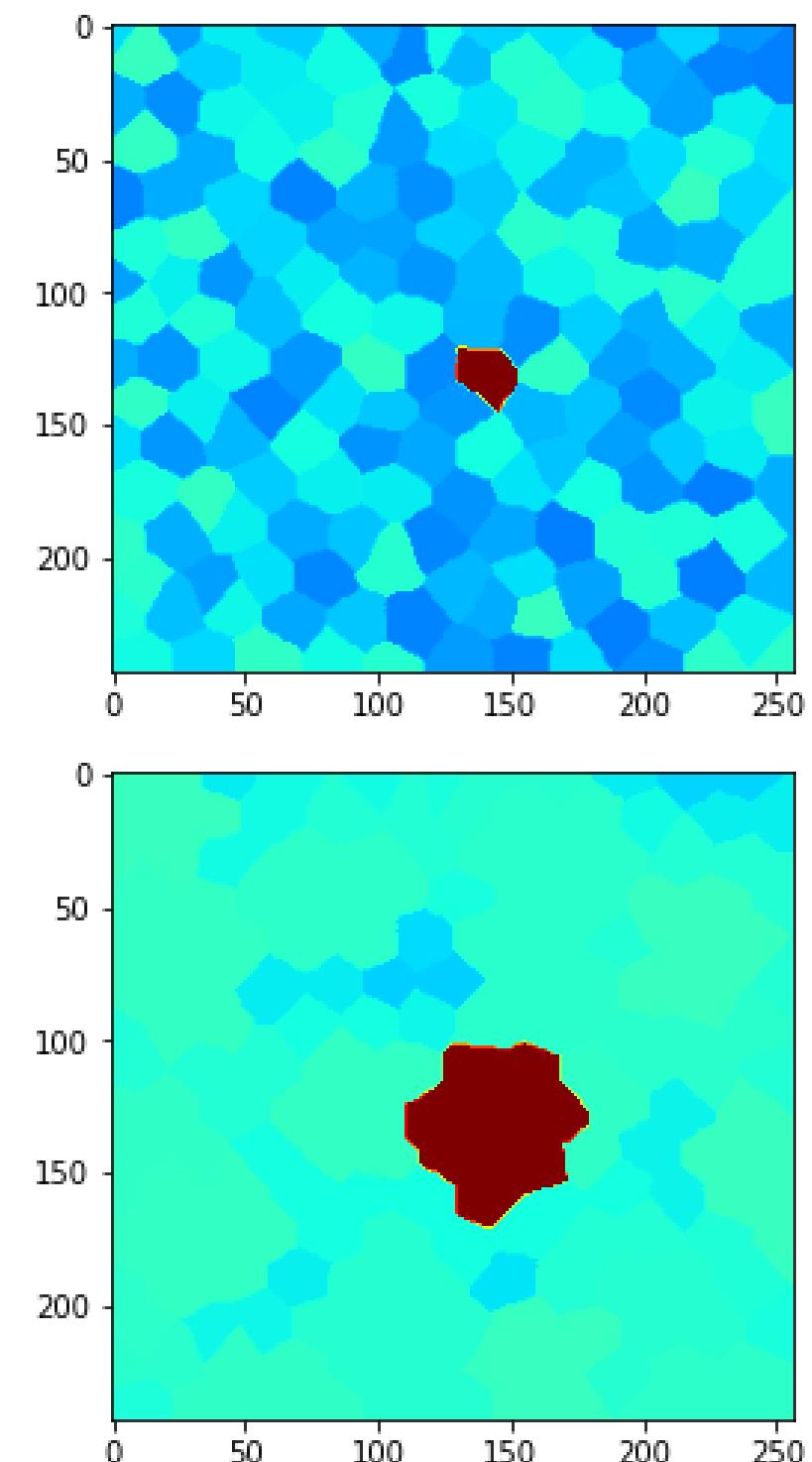
Part I: Basics



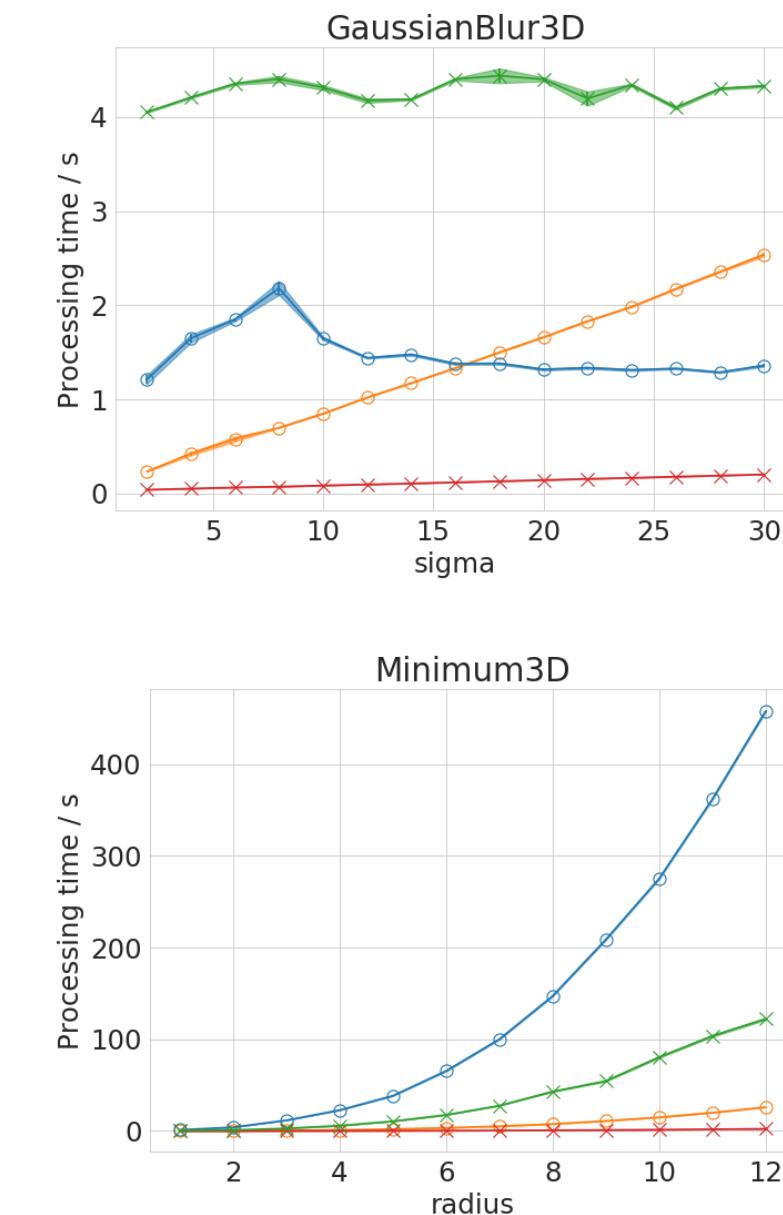
Part II: Interactive workflow design



Part III: Life-sciences specific image processing



Part IV: Benchmarking

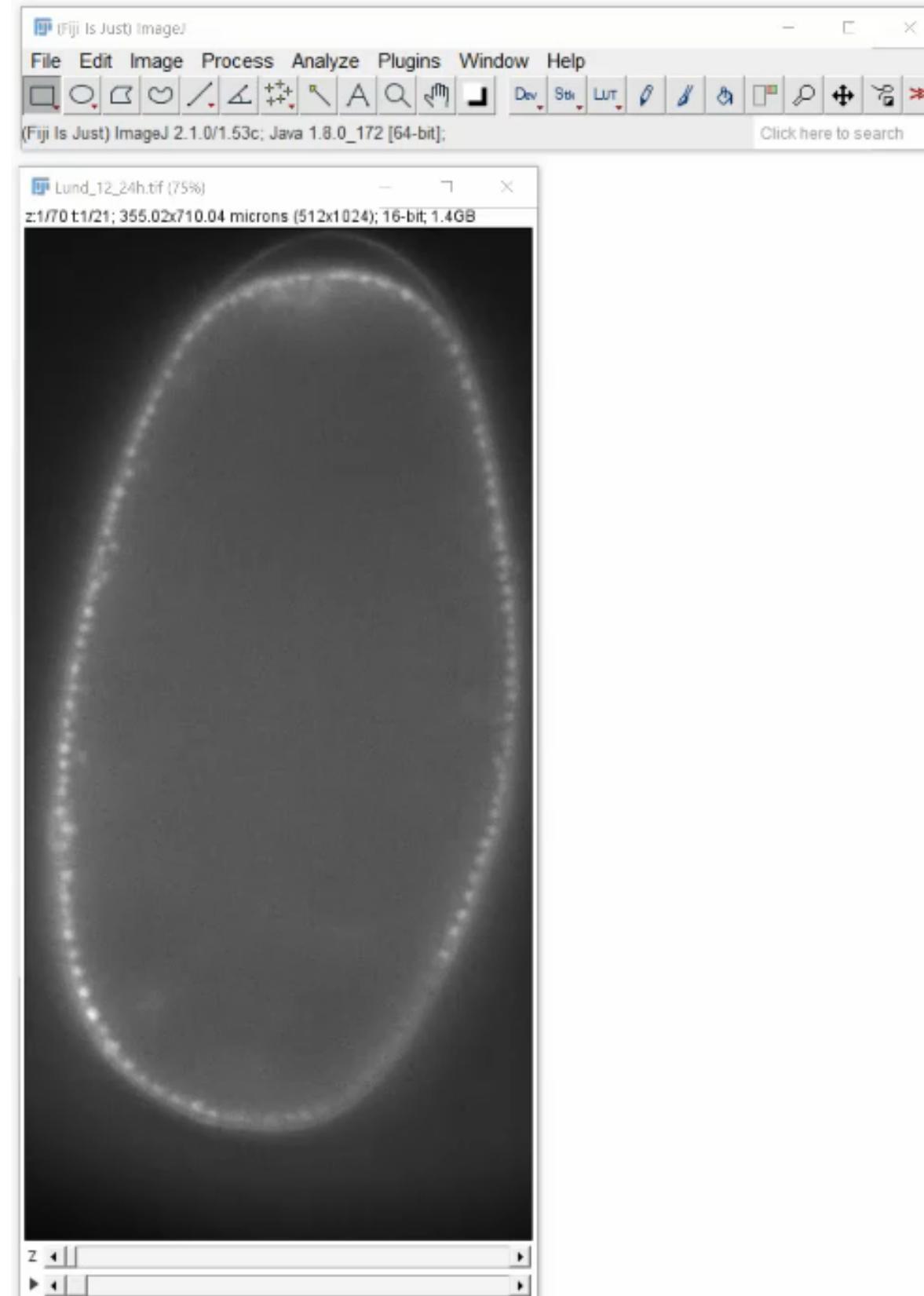


Part V: Accessibility

Part I: Basics

Image processing in life-sciences

- State-of-the-art software for more than 20 years: ImageJ / Fiji



ImageJ / Fiji integration for OpenCL



How image processing is supposed to be

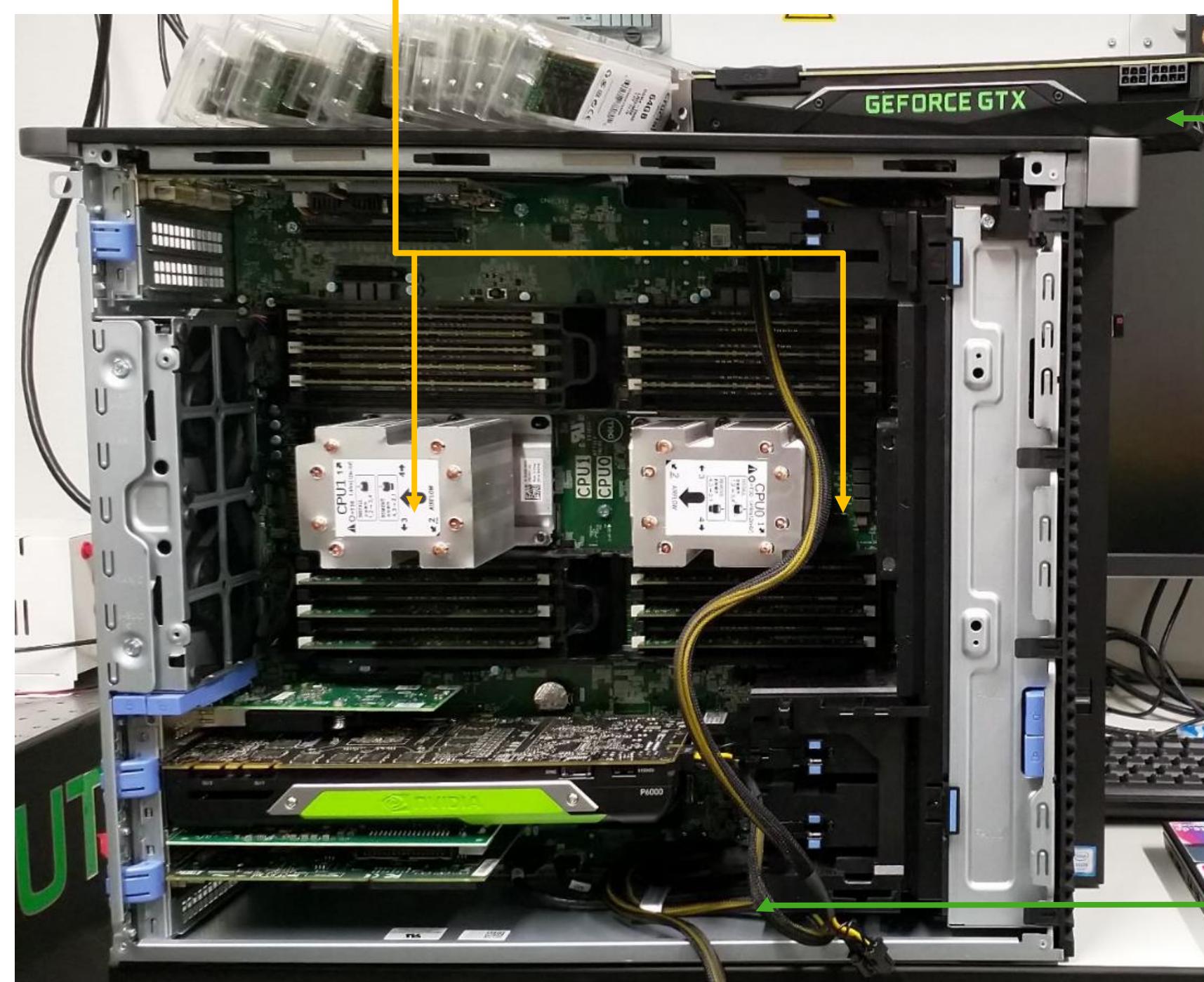
(... in my honest opinion)



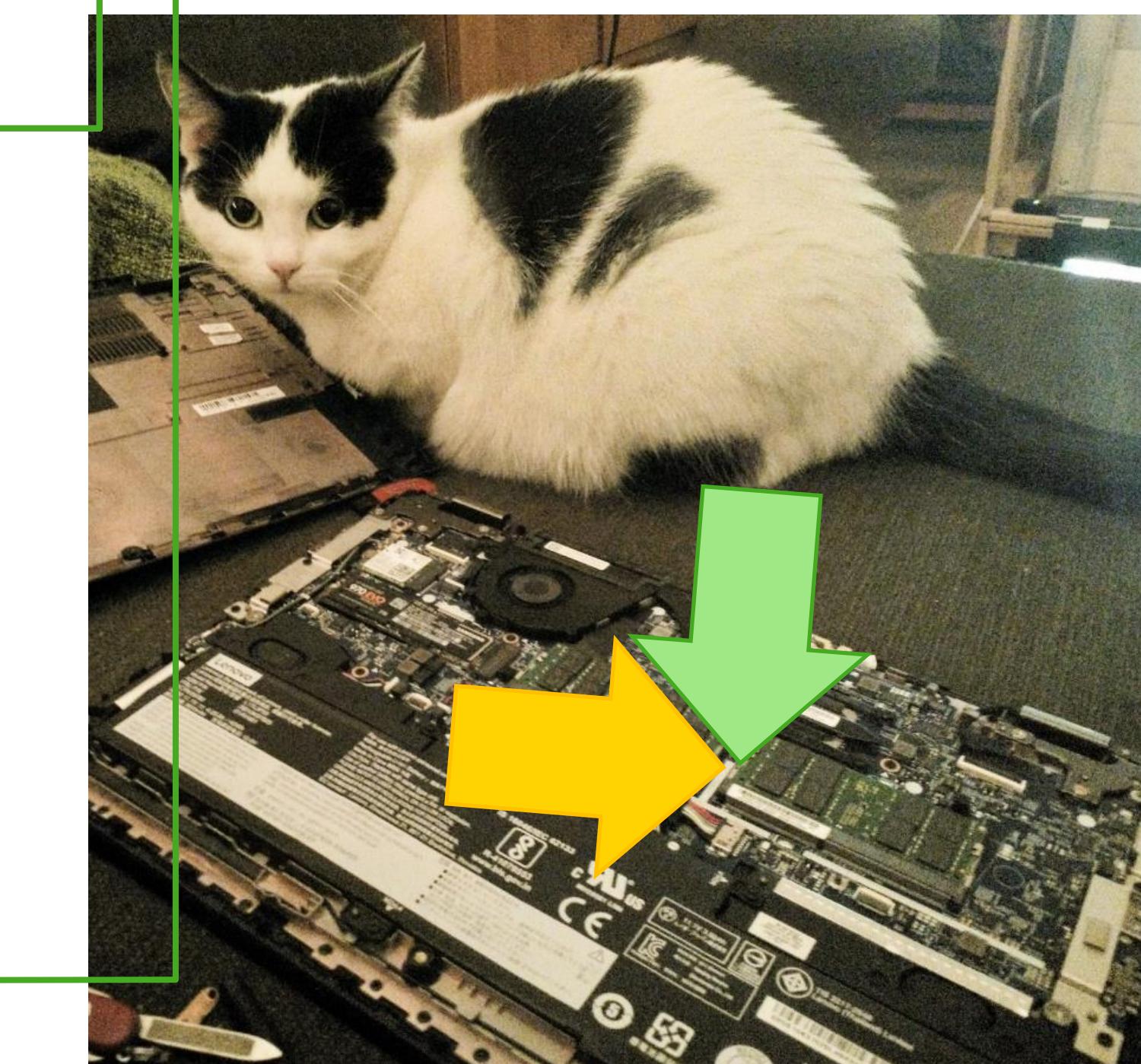
Graphics Processing Units (GPUs)

Every computer that has a screen also has a GPU.

Central Processing Unit
(CPU)

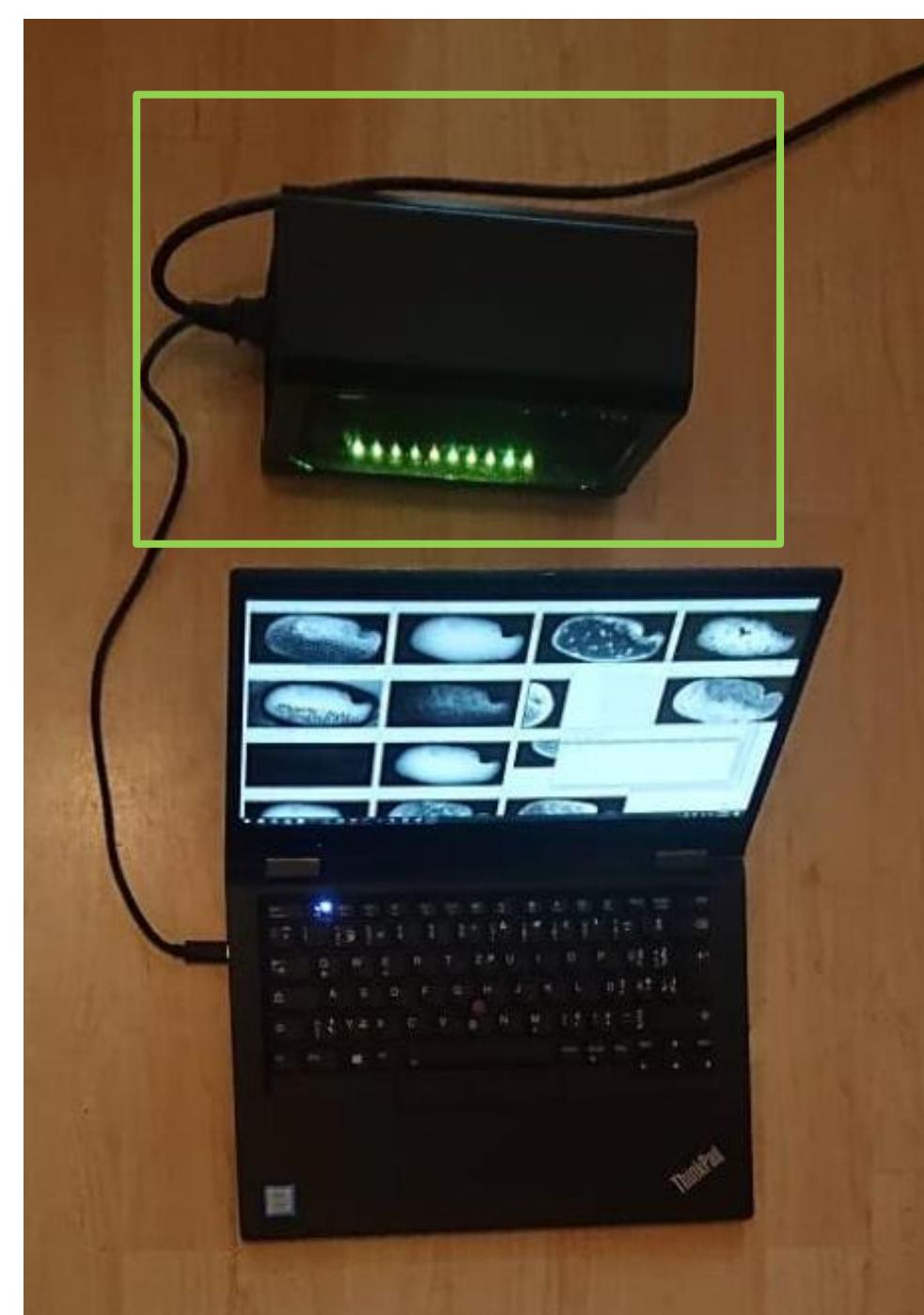


Graphics Processing Unit
(GPU)



dedicated GPUs

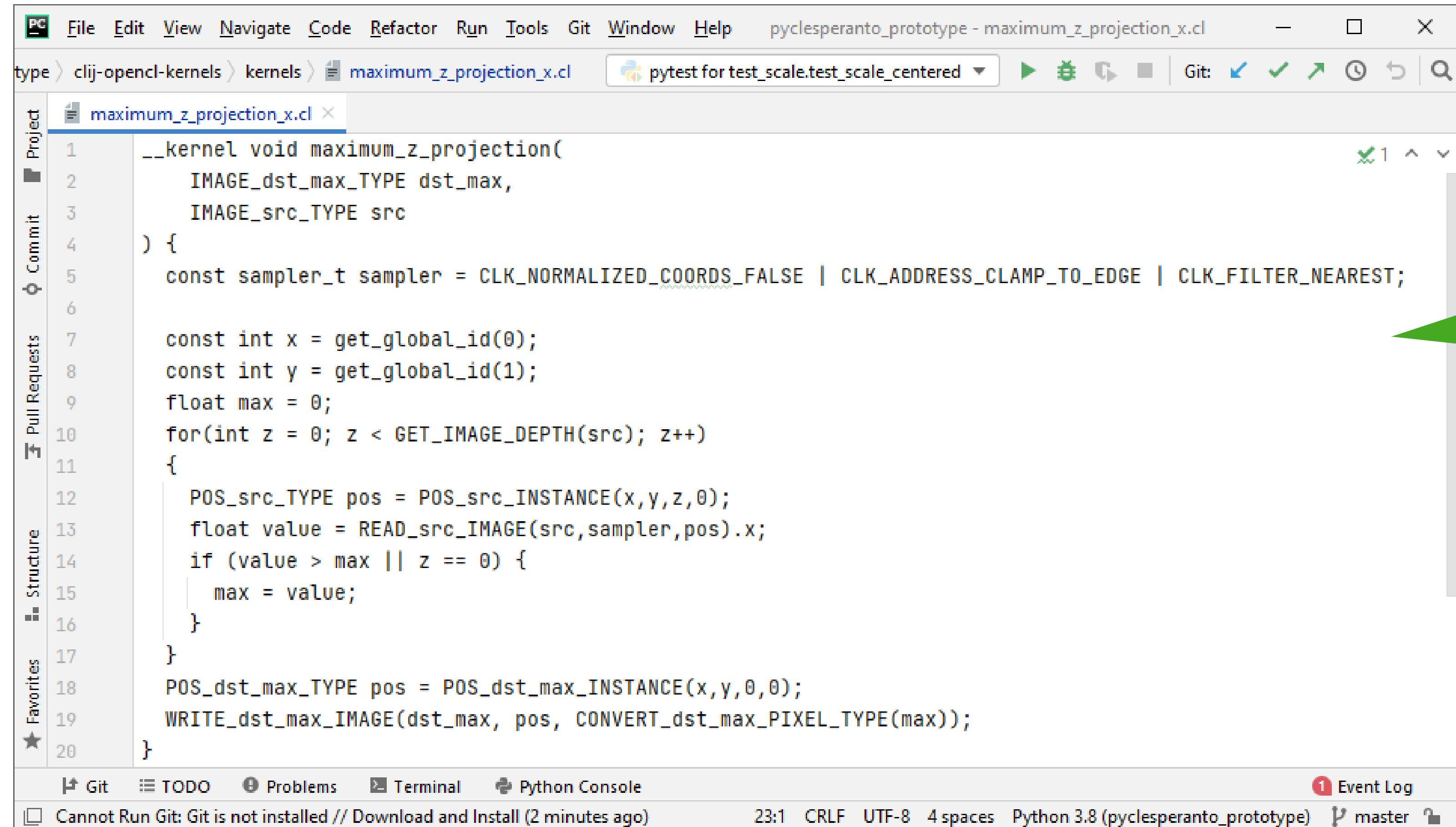
integrated GPUs



external "eGPUs"

OpenCL-based GPU-acceleration

GPU-acceleration? Learn the Open Computing Language (OpenCL)!



The screenshot shows a code editor window with the following details:

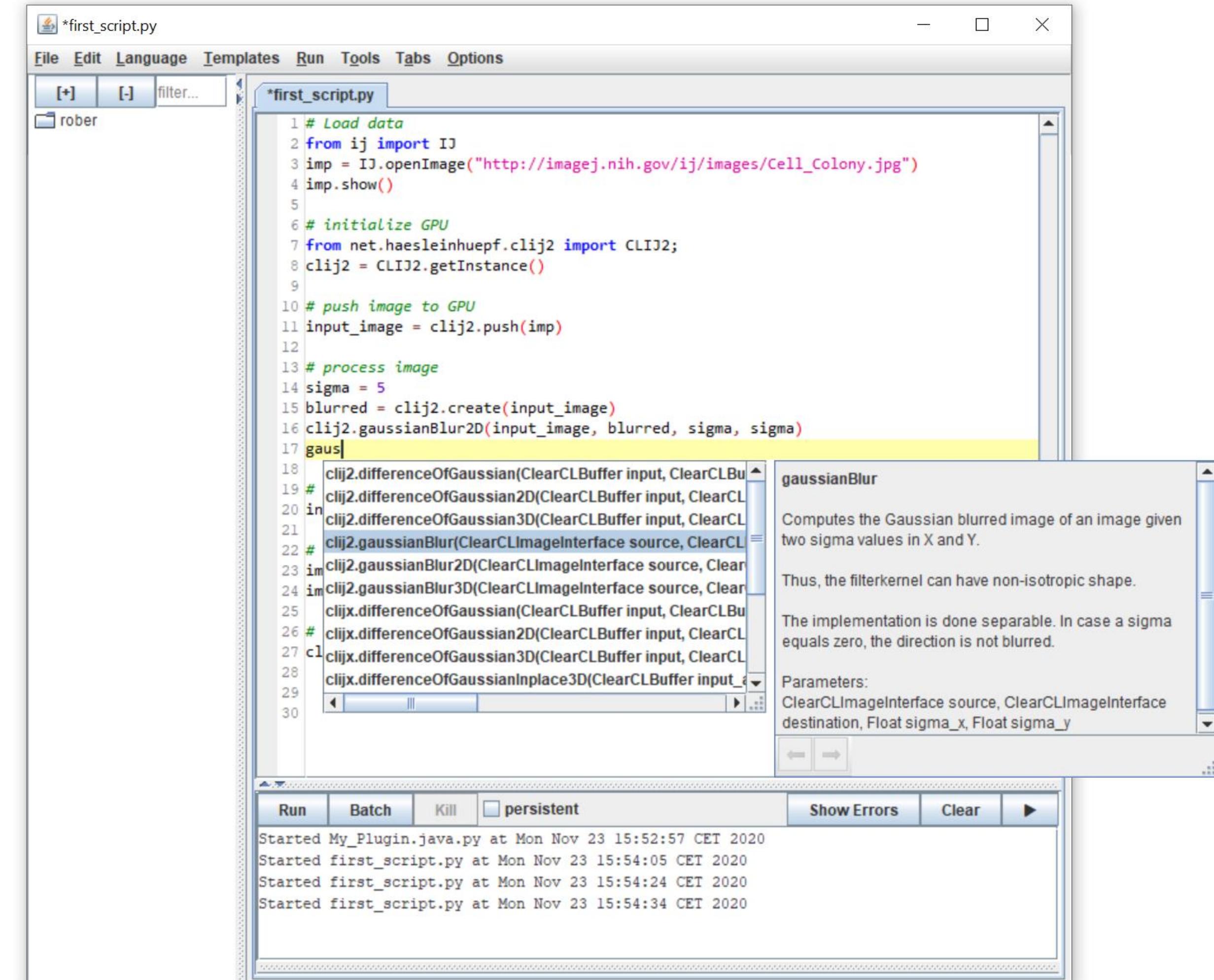
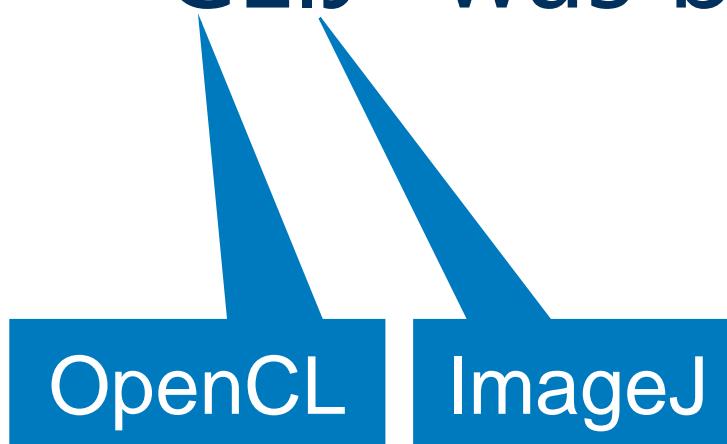
- Title Bar:** pyclesperanto_prototype - maximum_z_projection_x.cl
- Toolbar:** File, Edit, View, Navigate, Code, Refactor, Run, Tools, Git, Window, Help.
- Status Bar:** Cannot Run Git: Git is not installed // Download and Install (2 minutes ago), 23:1 CRLF, UTF-8, 4 spaces, Python 3.8 (pyclesperanto_prototype), master.
- Code Area:** The file maximum_z_projection_x.cl contains the following OpenCL kernel code:

```
1  __kernel void maximum_z_projection(
2      IMAGE_dst_max_TYPE dst_max,
3      IMAGE_src_TYPE src
4  ) {
5      const sampler_t sampler = CLK_NORMALIZED_COORDS_FALSE | CLK_ADDRESS_CLAMP_TO_EDGE | CLK_FILTER_NEAREST;
6
7      const int x = get_global_id(0);
8      const int y = get_global_id(1);
9      float max = 0;
10     for(int z = 0; z < GET_IMAGE_DEPTH(src); z++)
11     {
12         POS_src_TYPE pos = POS_src_INSTANCE(x,y,z,0);
13         float value = READ_src_IMAGE(src,sampler,pos).x;
14         if (value > max || z == 0) {
15             max = value;
16         }
17     }
18     POS_dst_max_TYPE pos = POS_dst_max_INSTANCE(x,y,0,0);
19     WRITE_dst_max_IMAGE(dst_max, pos, CONVERT_dst_max_PIXEL_TYPE(max));
20 }
```
- Right Panel:** A green callout box with the text "Maximum intensity projection along Z" points to the code area.

ImageJ / Fiji integration

- We integrated GPU-acceleration into ImageJ / Fiji using ImageJ Macro "instead" of OpenCL.

"CLIJ" was born.



The screenshot shows the ImageJ macro editor interface. The main window displays a Python script named "first_script.py". The script imports the ImageJ module and loads a sample image. It then initializes a GPU using the CLIJ2 library and performs a Gaussian blur operation on the image. A tooltip for the "gaussianBlur" function is open, providing documentation: "Computes the Gaussian blurred image of an image given two sigma values in X and Y. Thus, the filterkernel can have non-isotropic shape. The implementation is done separable. In case a sigma equals zero, the direction is not blurred." The parameters listed are "ClearCLImageInterface source, ClearCLImageInterface destination, Float sigma_x, Float sigma_y". The status bar at the bottom shows the execution history of the script.

```
*first_script.py
File Edit Language Templates Run Tools Tabs Options
[+] [-] filter...
*first_script.py
1 # Load data
2 from ij import IJ
3 imp = IJ.openImage("http://imagej.nih.gov/ij/images/Cell_Colony.jpg")
4 imp.show()
5
6 # initialize GPU
7 from net.haesleinhuepf.clij2 import CLIJ2;
8 clij2 = CLIJ2.getInstance()
9
10 # push image to GPU
11 input_image = clij2.push(imp)
12
13 # process image
14 sigma = 5
15 blurred = clij2.create(input_image)
16 clij2.gaussianBlur2D(input_image, blurred, sigma, sigma)
17 gaus|
18 clij2.differenceOfGaussian(ClearCLBuffer input, ClearCLBu|
19 # clij2.differenceOfGaussian2D(ClearCLBuffer input, ClearCL|
20 in clij2.differenceOfGaussian3D(ClearCLBuffer input, ClearCL|
21 clij2.gaussianBlur(ClearCLImageInterface source, ClearCL|
22 # clij2.gaussianBlur2D(ClearCLImageInterface source, ClearCL|
23 im clij2.gaussianBlur3D(ClearCLImageInterface source, ClearCL|
24 im clij2.gaussianBlur(ClearCLImageInterface source, ClearCL|
25 clijx.differenceOfGaussian(ClearCLBuffer input, ClearCLBu|
26 # clijx.differenceOfGaussian2D(ClearCLBuffer input, ClearCL|
27 clijx.differenceOfGaussian3D(ClearCLBuffer input, ClearCL|
28 clijx.differenceOfGaussianInplace3D(ClearCLBuffer input, C|
29
30
gaussianBlur
Computes the Gaussian blurred image of an image given two sigma values in X and Y. Thus, the filterkernel can have non-isotropic shape. The implementation is done separable. In case a sigma equals zero, the direction is not blurred.
Parameters:
ClearCLImageInterface source, ClearCLImageInterface destination, Float sigma_x, Float sigma_y
Run Batch Kill persistent Show Errors Clear >
```

Started My_Plugin.java.py at Mon Nov 23 15:52:57 CET 2020
Started first_script.py at Mon Nov 23 15:54:05 CET 2020
Started first_script.py at Mon Nov 23 15:54:24 CET 2020
Started first_script.py at Mon Nov 23 15:54:34 CET 2020

CLIJ2: What every ImageJ Macro script must have

Load data

```
1 // Load data
2 run("Cell Colony (31K)");
3
4 // initialize GPU
5 run("CLIJ2 Macro Extensions", "cl_device=");
6 Ext.CLIJ2_clear();
7
8 // push image to GPU
9 input_image = getTitle();
10 Ext.CLIJ2_push(input_image);
11
12 // process image
13 sigma = 5;
14 Ext.CLIJ2_gaussianBlur2D(input_image, result_image, sigma, sigma);
15
16 // optional: release input data
17 Ext.CLIJ2_release(input_image);
18
19 // pull result back from GPU
20 Ext.CLIJ2_pull(result_image);
21
22 // clean up by the end
23 Ext.CLIJ2_clear();
```

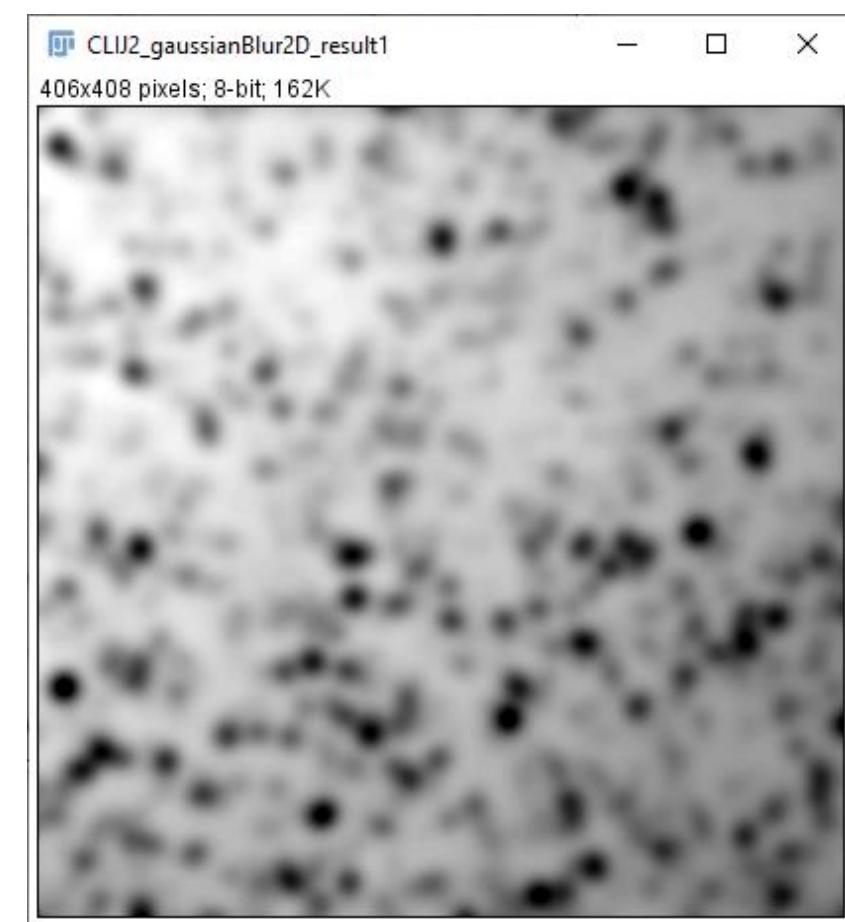
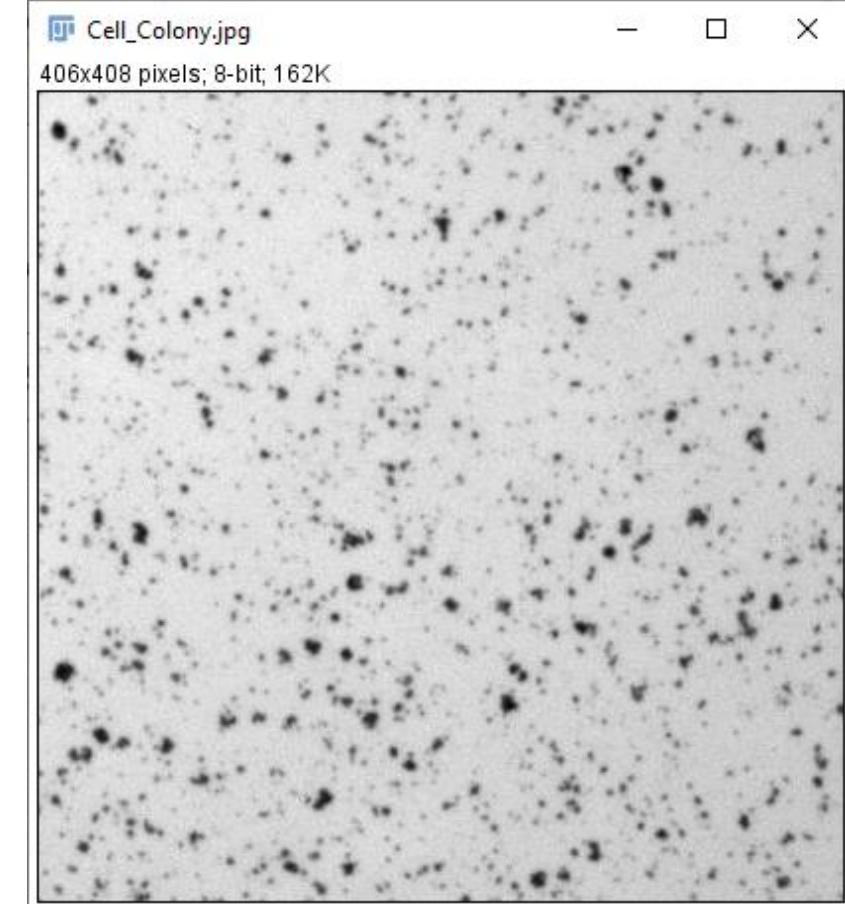
Initialize GPU

Push

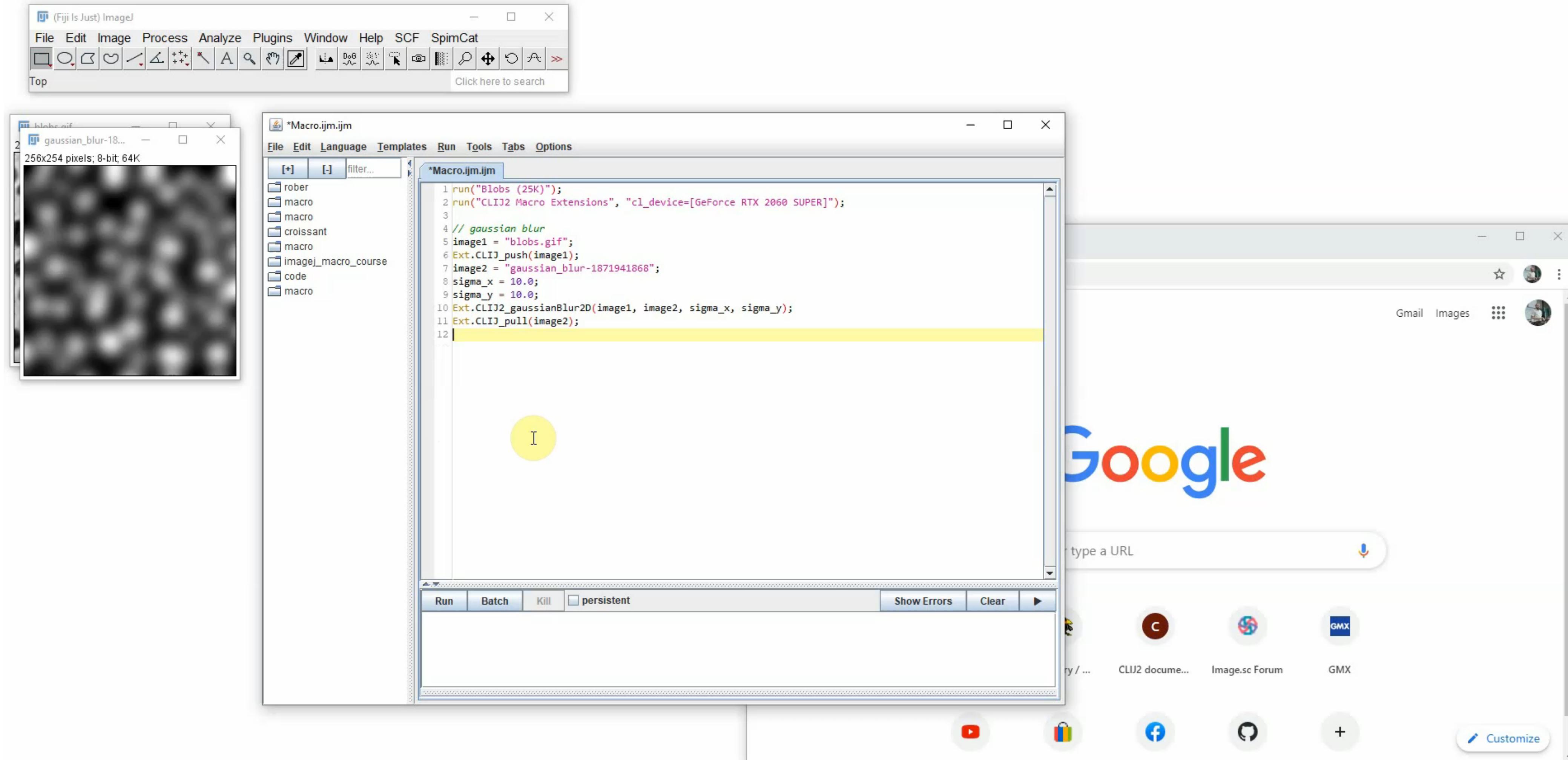
Process images

Pull

Cleanup



CLIJ2: Macro editing

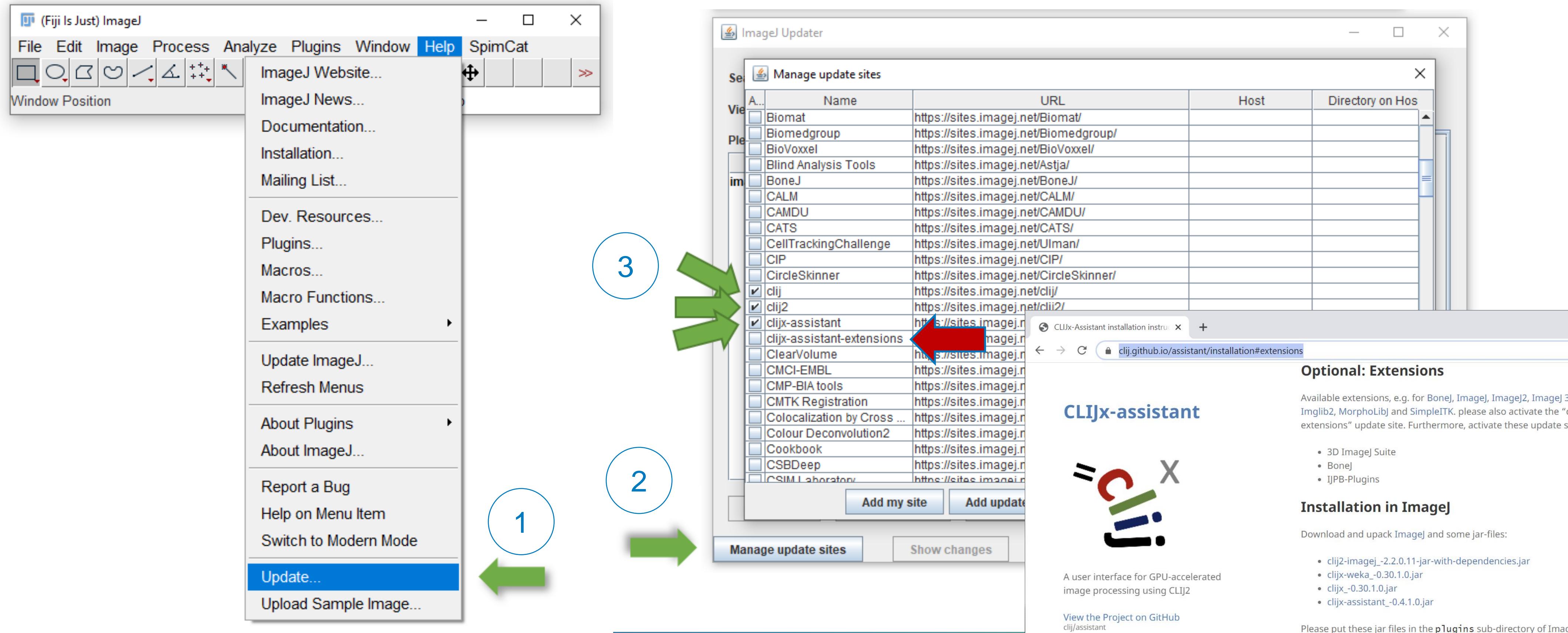


Exercises

Exercise 0: Installation

Just activate the CLIJ update sites, in menu Help > Update...

Linux users: you may need to install OpenCL, e.g. apt-get install ocl-icd-devel

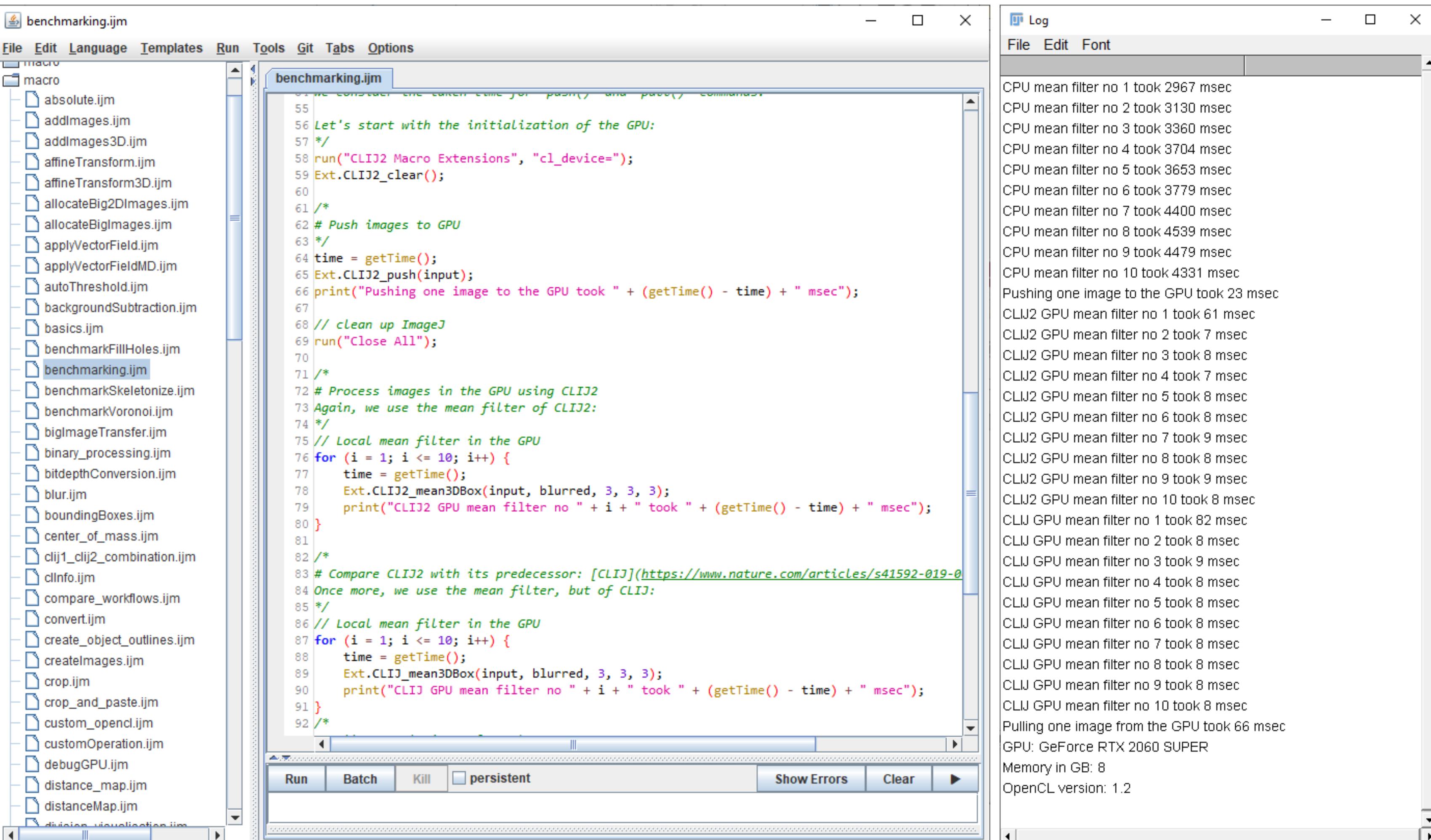


<http://clij.github.io/assistant/installation>

Exercise 1: Testing if CLIJ2 works

Get an example macro from
<http://clij.github.io/clij2-docs/>

- > Code examples
 - > ImageJ Macro
 - > benchmarking.ijm



The screenshot shows the ImageJ macro editor with the 'benchmarking.ijm' script open. The script performs several GPU operations, including initializing the GPU, pushing images to the GPU, processing them with mean filters, and comparing CLIJ2 with its predecessor CLIJ. The 'Log' window to the right displays the execution time for each operation, such as 'CPU mean filter no 1 took 2967 msec' and 'GPU mean filter no 1 took 61 msec'. The log also includes system information like 'GPU: GeForce RTX 2060 SUPER' and memory usage.

```
benchmarking.ijm
File Edit Language Templates Run Tools Git Tabs Options
macro
macro
absolute.ijm
addlImages.ijm
addlImages3D.ijm
affineTransform.ijm
affineTransform3D.ijm
allocateBig2DImages.ijm
allocateBigImages.ijm
applyVectorField.ijm
applyVectorFieldMD.ijm
autoThreshold.ijm
backgroundSubtraction.ijm
basics.ijm
benchmarkFillHoles.ijm
benchmarking.ijm
benchmarkSkeletonize.ijm
benchmarkVoronoi.ijm
bigImageTransfer.ijm
binary_processing.ijm
bitdepthConversion.ijm
blur.ijm
boundingBoxes.ijm
center_of_mass.ijm
clij1_clij2_combination.ijm
clInfo.ijm
compare_workflows.ijm
convert.ijm
create_object_outlines.ijm
createlImages.ijm
crop.ijm
crop_and_paste.ijm
custom_opengl.ijm
customOperation.ijm
debugGPU.ijm
distance_map.ijm
distanceMap.ijm
division_visualization.ijm

benchmarking.ijm
55
56 Let's start with the initialization of the GPU:
57 /*
58 run("CLIJ2 Macro Extensions", "cl_device=");
59 Ext.CLIJ2_clear();
60
61 /*
62 # Push images to GPU
63 /*
64 time = getTime();
65 Ext.CLIJ2_push(input);
66 print("Pushing one image to the GPU took " + (getTime() - time) + " msec");
67
68 // clean up ImageJ
69 run("Close All");
70
71 /*
72 # Process images in the GPU using CLIJ2
73 Again, we use the mean filter of CLIJ2:
74 /*
75 // Local mean filter in the GPU
76 for (i = 1; i <= 10; i++) {
77   time = getTime();
78   Ext.CLIJ2_mean3DBox(input, blurred, 3, 3, 3);
79   print("CLIJ2 GPU mean filter no " + i + " took " + (getTime() - time) + " msec");
80 }
81
82 /*
83 # Compare CLIJ2 with its predecessor: [CLIJ](https://www.nature.com/articles/s41592-019-0
84 Once more, we use the mean filter, but of CLIJ:
85 /*
86 // Local mean filter in the GPU
87 for (i = 1; i <= 10; i++) {
88   time = getTime();
89   Ext.CLIJ_mean3DBox(input, blurred, 3, 3, 3);
90   print("CLIJ GPU mean filter no " + i + " took " + (getTime() - time) + " msec");
91 }
92 */

Run Batch Kill persistent Show Errors Clear >
```

Log

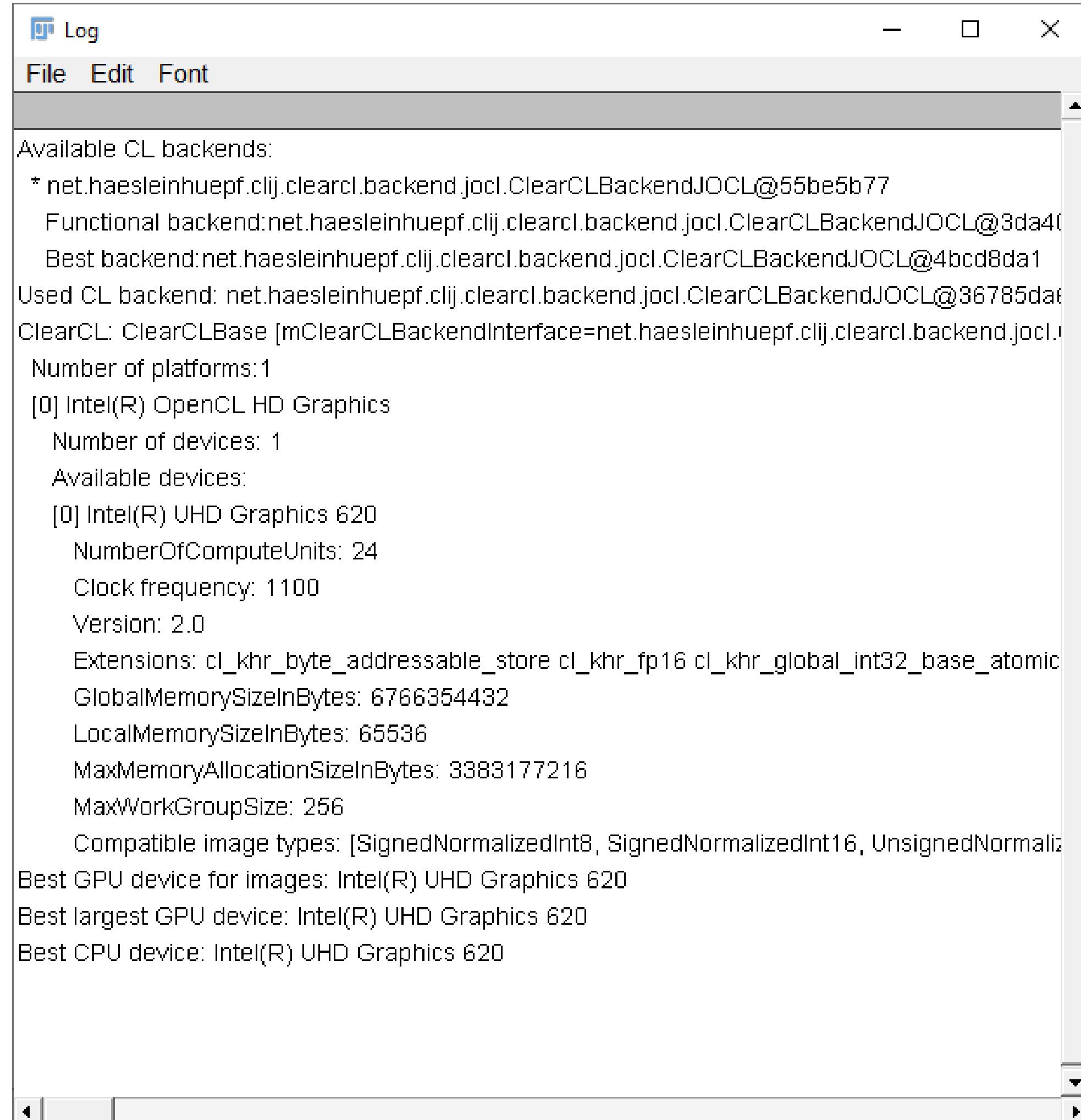
CPU mean filter no 1 took 2967 msec
CPU mean filter no 2 took 3130 msec
CPU mean filter no 3 took 3360 msec
CPU mean filter no 4 took 3704 msec
CPU mean filter no 5 took 3653 msec
CPU mean filter no 6 took 3779 msec
CPU mean filter no 7 took 4400 msec
CPU mean filter no 8 took 4539 msec
CPU mean filter no 9 took 4479 msec
CPU mean filter no 10 took 4331 msec
Pushing one image to the GPU took 23 msec
CLIJ2 GPU mean filter no 1 took 61 msec
CLIJ2 GPU mean filter no 2 took 7 msec
CLIJ2 GPU mean filter no 3 took 8 msec
CLIJ2 GPU mean filter no 4 took 7 msec
CLIJ2 GPU mean filter no 5 took 8 msec
CLIJ2 GPU mean filter no 6 took 8 msec
CLIJ2 GPU mean filter no 7 took 9 msec
CLIJ2 GPU mean filter no 8 took 8 msec
CLIJ2 GPU mean filter no 9 took 9 msec
CLIJ2 GPU mean filter no 10 took 8 msec
CLIJ GPU mean filter no 1 took 82 msec
CLIJ GPU mean filter no 2 took 8 msec
CLIJ GPU mean filter no 3 took 9 msec
CLIJ GPU mean filter no 4 took 8 msec
CLIJ GPU mean filter no 5 took 8 msec
CLIJ GPU mean filter no 6 took 8 msec
CLIJ GPU mean filter no 7 took 8 msec
CLIJ GPU mean filter no 8 took 8 msec
CLIJ GPU mean filter no 9 took 8 msec
CLIJ GPU mean filter no 10 took 8 msec
Pulling one image from the GPU took 66 msec
GPU: GeForce RTX 2060 SUPER
Memory in GB: 8
OpenCL version: 1.2

Exercise 2: Study your hardware properties

Execute this script:

<https://github.com/clij/clij2-docs/blob/master/src/main/macro/clInfo.ijm>

- What GPU is used per default?
- How much memory does it have available?
- How large can a single image be when being processed in the GPU?
- Assume your image stack is 1024x1024 pixels large and of type 32-bit float, how many planes can it have so that it still fits in GPU memory?



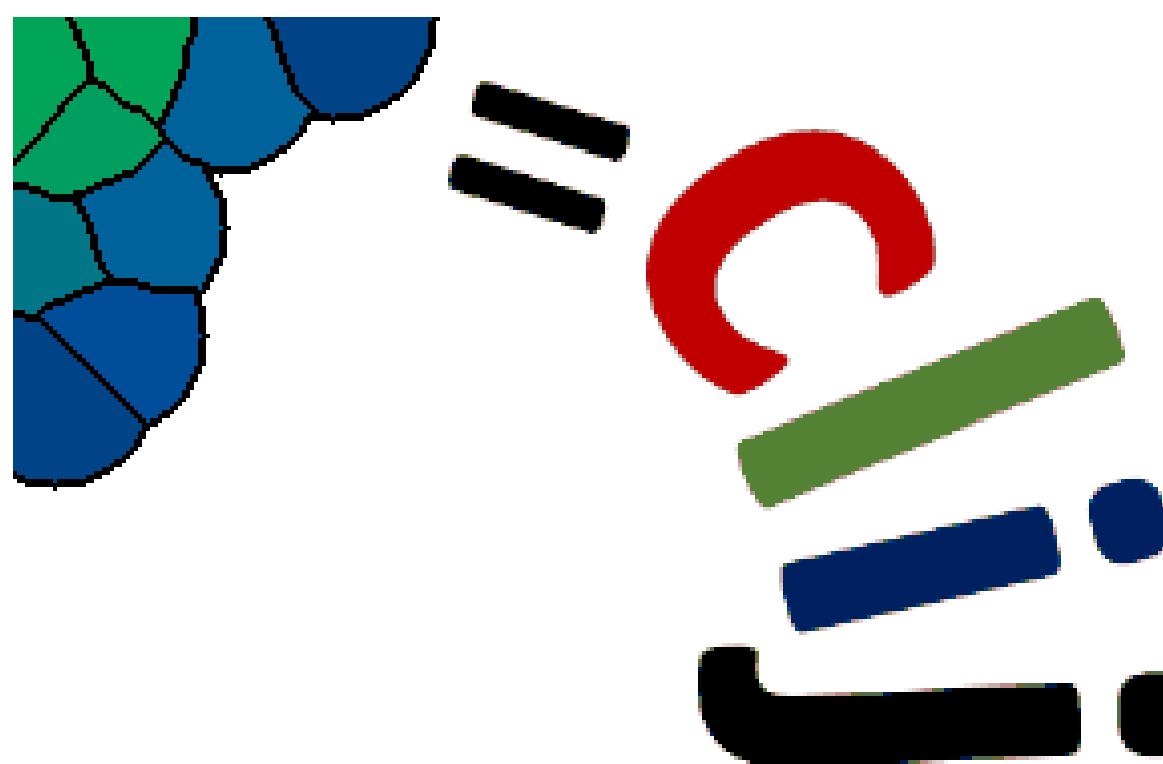
The screenshot shows a window titled "Log" with a menu bar containing "File", "Edit", and "Font". The main area displays the output of a script. It lists available CL backends, the used CL backend, clearCL base information, number of platforms, Intel(R) OpenCL HD Graphics details, available devices, Intel(R) UHD Graphics 620 details, and various memory and compute specifications. It also lists the best GPU device for images, the best largest GPU device, and the best CPU device.

```
Available CL backends:  
* net.haesleinhuepf.clj.clearcl.backend.jocl.ClearCLBackendJOCL@55be5b77  
  Functional backend: net.haesleinhuepf.clj.clearcl.backend.jocl.ClearCLBackendJOCL@3da40  
  Best backend: net.haesleinhuepf.clj.clearcl.backend.jocl.ClearCLBackendJOCL@4bcd8da1  
Used CL backend: net.haesleinhuepf.clj.clearcl.backend.jocl.ClearCLBackendJOCL@36785da  
ClearCL: ClearCLBase [mClearCLBackendInterface=net.haesleinhuepf.clj.clearcl.backend.jocl.JOCLBackend]  
Number of platforms: 1  
[0] Intel(R) OpenCL HD Graphics  
  Number of devices: 1  
  Available devices:  
  [0] Intel(R) UHD Graphics 620  
    NumberOfComputeUnits: 24  
    Clock frequency: 1100  
    Version: 2.0  
    Extensions: cl_khr_byte_addressable_store cl_khr_fp16 cl_khr_global_int32_base_atomic  
    GlobalMemorySizeInBytes: 6766354432  
    LocalMemorySizeInBytes: 65536  
    MaxMemoryAllocationSizeInBytes: 3383177216  
    MaxWorkGroupSize: 256  
    Compatible image types: [SignedNormalizedInt8, SignedNormalizedInt16, UnsignedNormalizedInt16]  
Best GPU device for images: Intel(R) UHD Graphics 620  
Best largest GPU device: Intel(R) UHD Graphics 620  
Best CPU device: Intel(R) UHD Graphics 620
```

Part II: Interactive workflow design

CLIJ2: More functions, wider applicability

- Functions for working with neighborhood-relationships (between cells)



CLIJ2: More functions, wider applicability

- Functions for working with neighborhood-relationships (between cells)



```
*help.ijm
File Edit Language Templates Run Tools Git Tabs Options
*help.ijm
1 // CLIJ example macro: help.ijm
2 //
3 // This macro shows how to get help on CLIJ methods
4 //
5 // Author: Robert Haase
6 // December 2018
7 //
8
9
10 run("CLIJ2 Macro Extensions", "cl_device=");
11 Ext.CLIJ2_help("clij_");

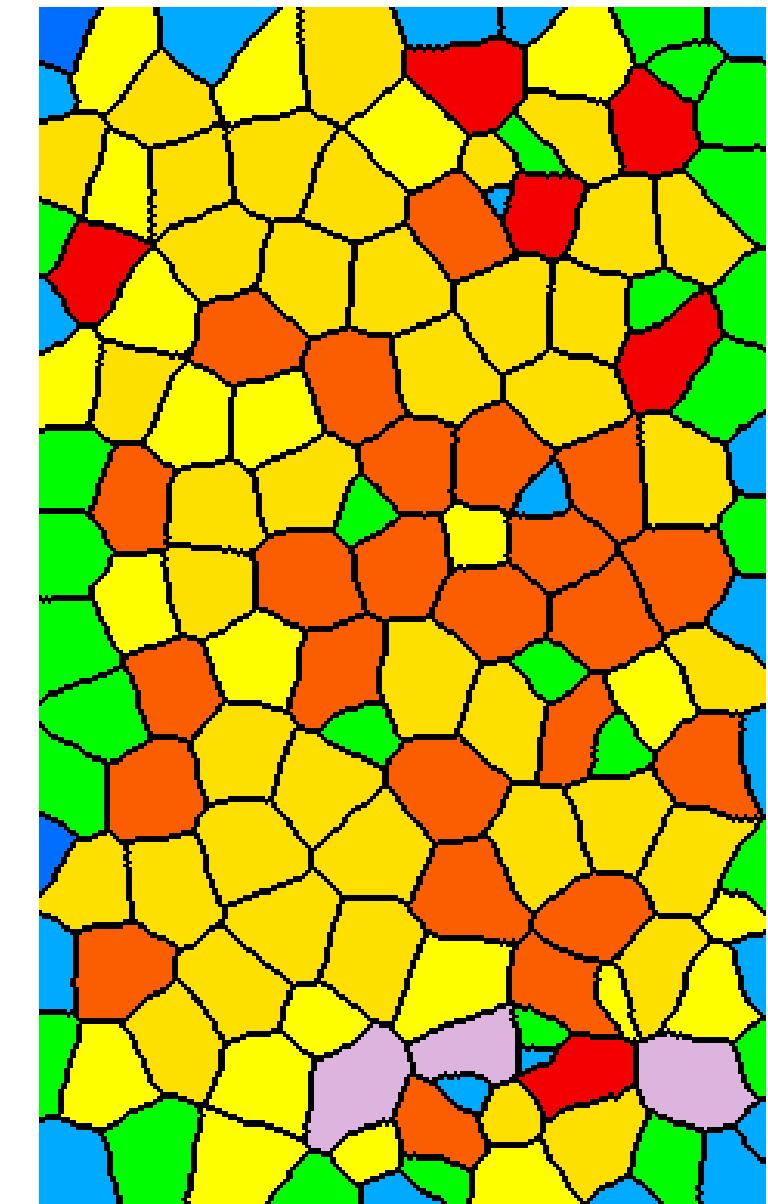
Run Batch Kill persistent Show Errors Clear ▶
Started help.ijm at Wed May 06 10:43:15 CEST 2020
Started help.ijm at Wed May 06 10:43:22 CEST 2020
```

```
*help.ijm (Running)
File Edit Language Templates Run Tools Git Tabs Options
*help.ijm (Running)
1 // CLIJ example macro: help.ijm
2 //
3 // This macro shows how to get help on CLIJ methods
4 //
5 // Author: Robert Haase
6 // December 2018
7 //
8
9
10 run("CLIJ2 Macro Extensions", "cl_device=");
11 Ext.CLIJ2_help("clij2_");

Run Batch Kill persistent Show Errors Clear ▶
Started help.ijm at Wed May 06 10:43:22 CEST 2020
Started help.ijm at Wed May 06 10:50:24 CEST 2020
```

```
Log
File Edit Font
Found 124 method(s) containing the pattern "clij_":
Ext.CLIJ_absolute(Image source, Image destination);
Ext.CLIJ_addImageAndScalar(Image source, Image destination);
Ext.CLIJ_addImages(Image summand1, Image summand2, Image destination);
Ext.CLIJ_addImagesWeighted(Image summand1, Image summand2, Image destination, double weight);
Ext.CLIJ_affineTransform(Image source, Image destination);
Ext.CLIJ_affineTransform2D(Image source, Image destination);
Ext.CLIJ_affineTransform3D(Image source, Image destination);
```

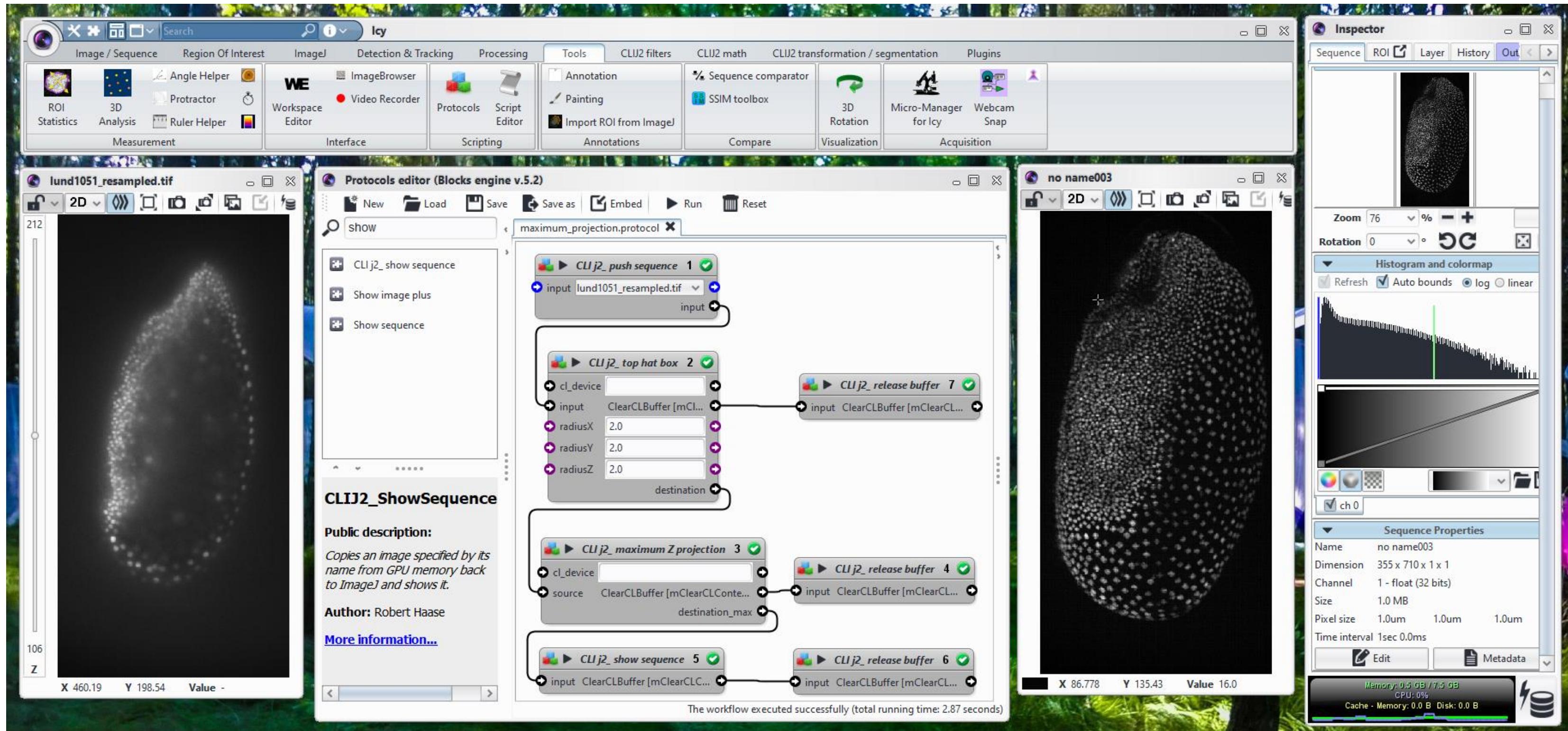
```
Log
File Edit Font
Found 324 method(s) containing the pattern "clij2_":
Ext.CLIJ2_GPUProperties();
Ext.CLIJ2_absolute(Image source, ByRef Image destination);
Ext.CLIJ2_addImageAndScalar(Image source, ByRef Image destination);
Ext.CLIJ2_addImages(Image summand1, Image summand2, Image destination);
Ext.CLIJ2_addImagesWeighted(Image summand1, Image summand2, Image destination, double weight);
Ext.CLIJ2_adjacencyMatrixToTouchMatrix(Image adjacencyMatrix, Image touchMatrix);
Ext.CLIJ2_affineTransform(Image source, ByRef Image destination);
```



Number of neighbors

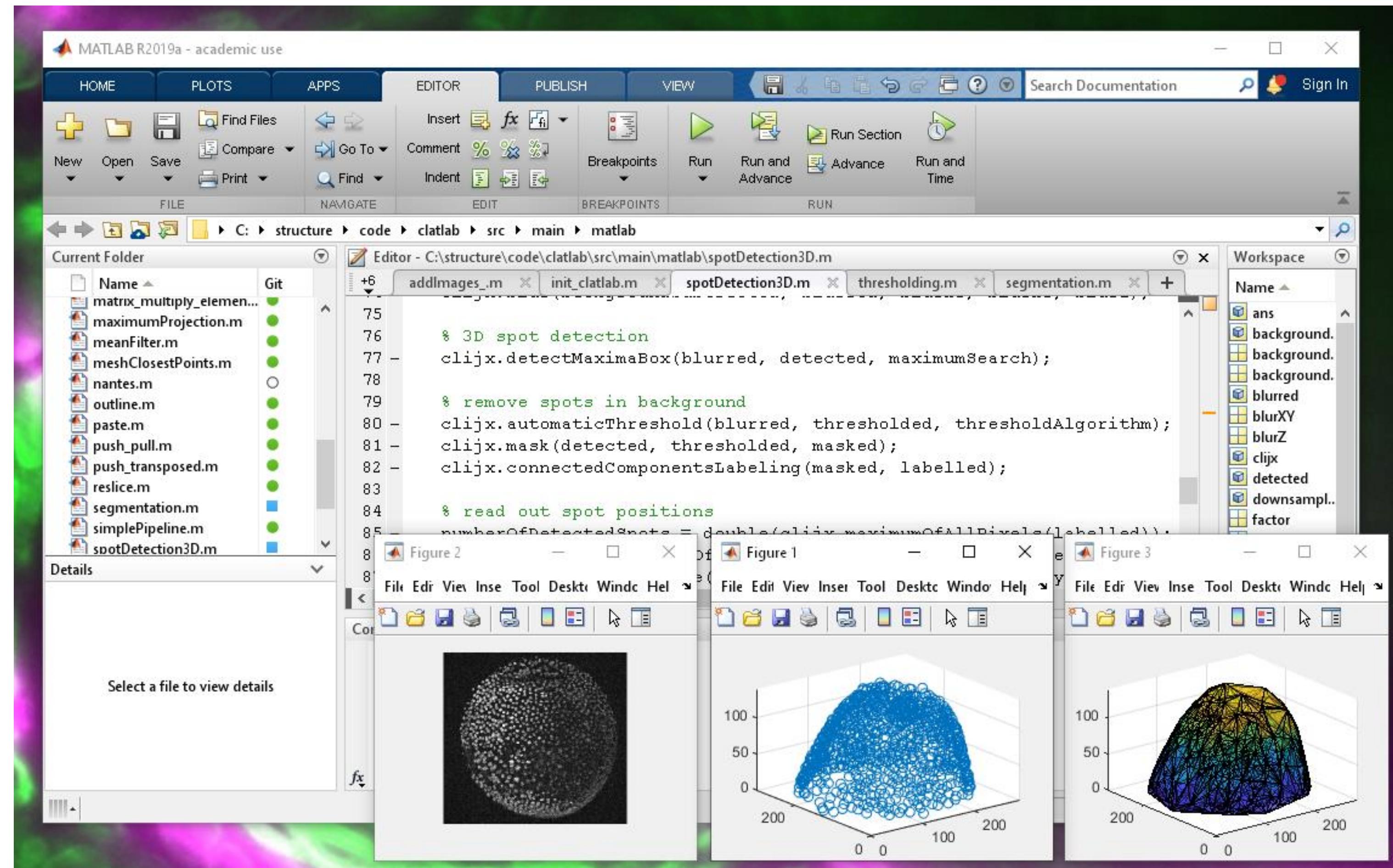
CLIJ2: More functions, wider applicability

- Functions for working with neighborhood-relationships (between cells)
- Integration for
 - Icy



CLIJ2: More functions, wider applicability

- Functions for working with neighborhood-relationships (between cells)
- Integration for
 - Icy
 - Matlab



<https://clij.github.io/clatlab/>

CLIJ2: What every Matlab script must have

Load data

```
1 % load data
2 image = imread("http://imagej.nih.gov/ij/images/Cell_Colony.jpg");
3 subplot(1, 2, 1), imshow(image);
```

Initialize GPU

```
5 % initialize GPU
6 clij2 = init_clatlab();
7
8 % Push Crop_ResSin1CGFP_016-1.tif to GPU memory
9 input_image = clij2.pushMat(image);
```

Push

```
11 % process image
12 sigma = 5;
13 blurred = clij2.create(input_image);
14 clij2.gaussianBlur2D(input_image, blurred, sigma, sigma)
```

Process images

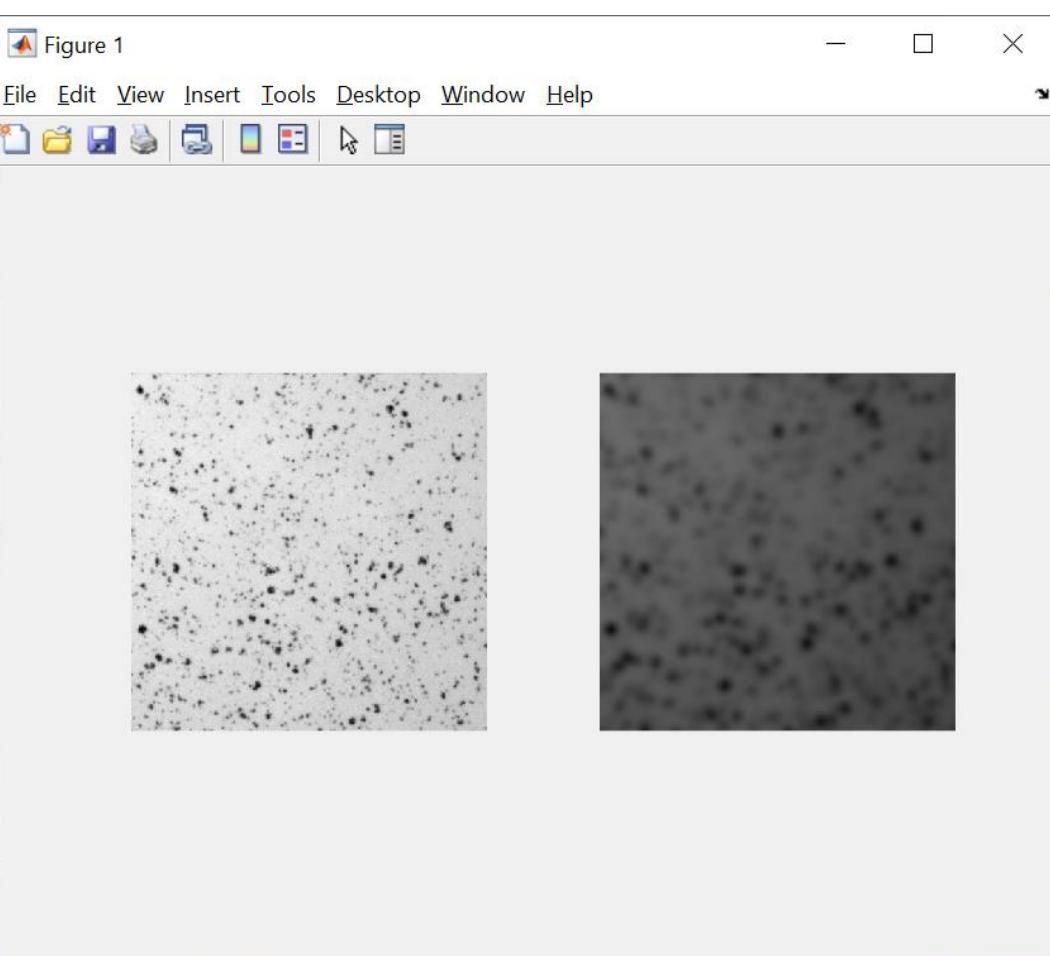
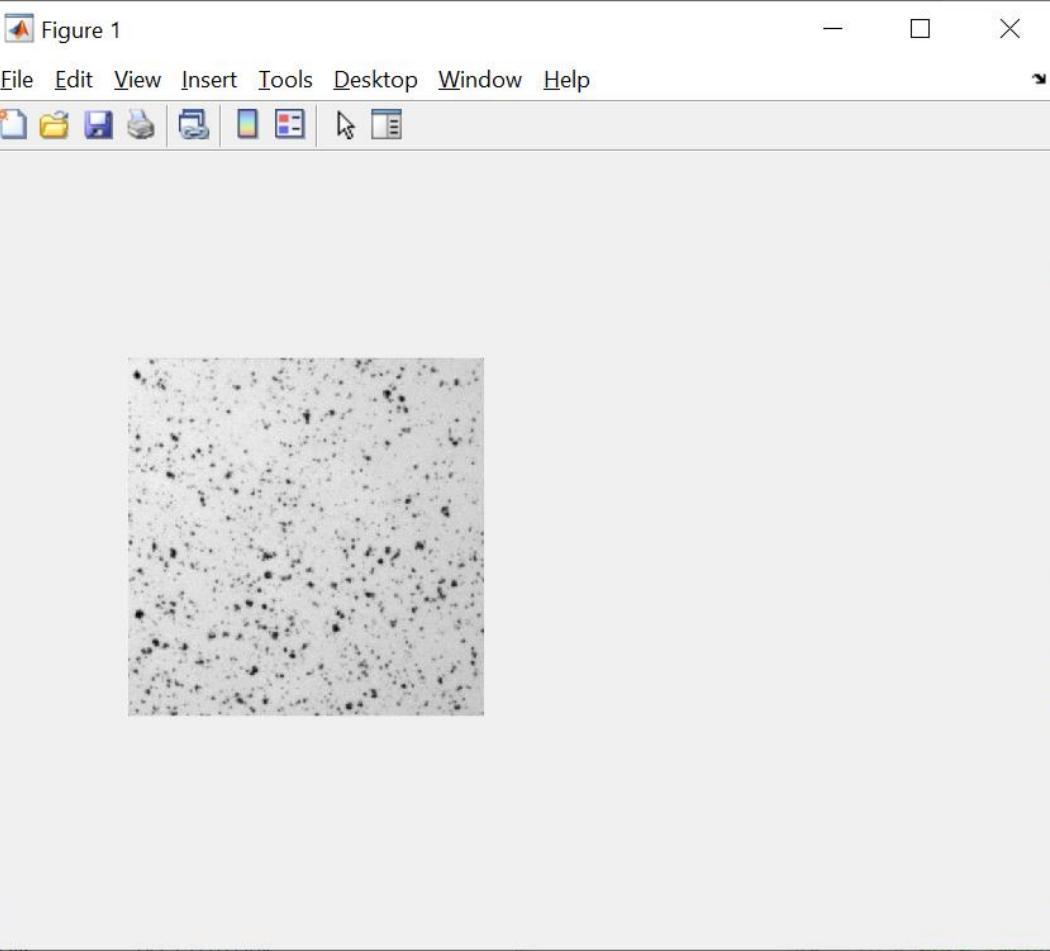
```
16 % optional: release input data
17 input_image.close()
```

Pull

```
19 % pull result back from GPU
20 result = clij2.pullMat(blurred);
21 %subplot(1, 2, 2), imshow(result);
```

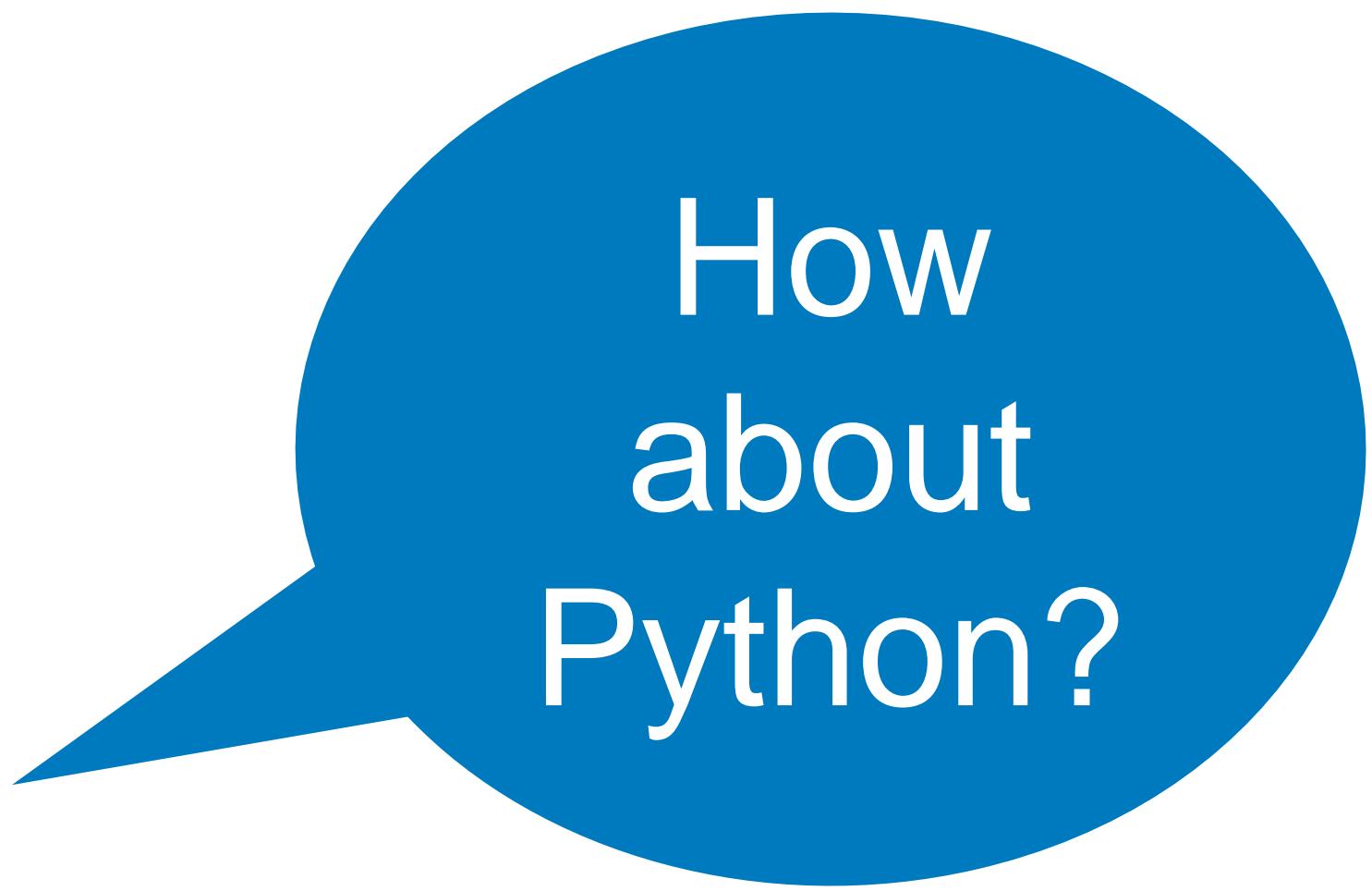
Cleanup

```
23 % clean up by the end
24 clij2.clear()
```

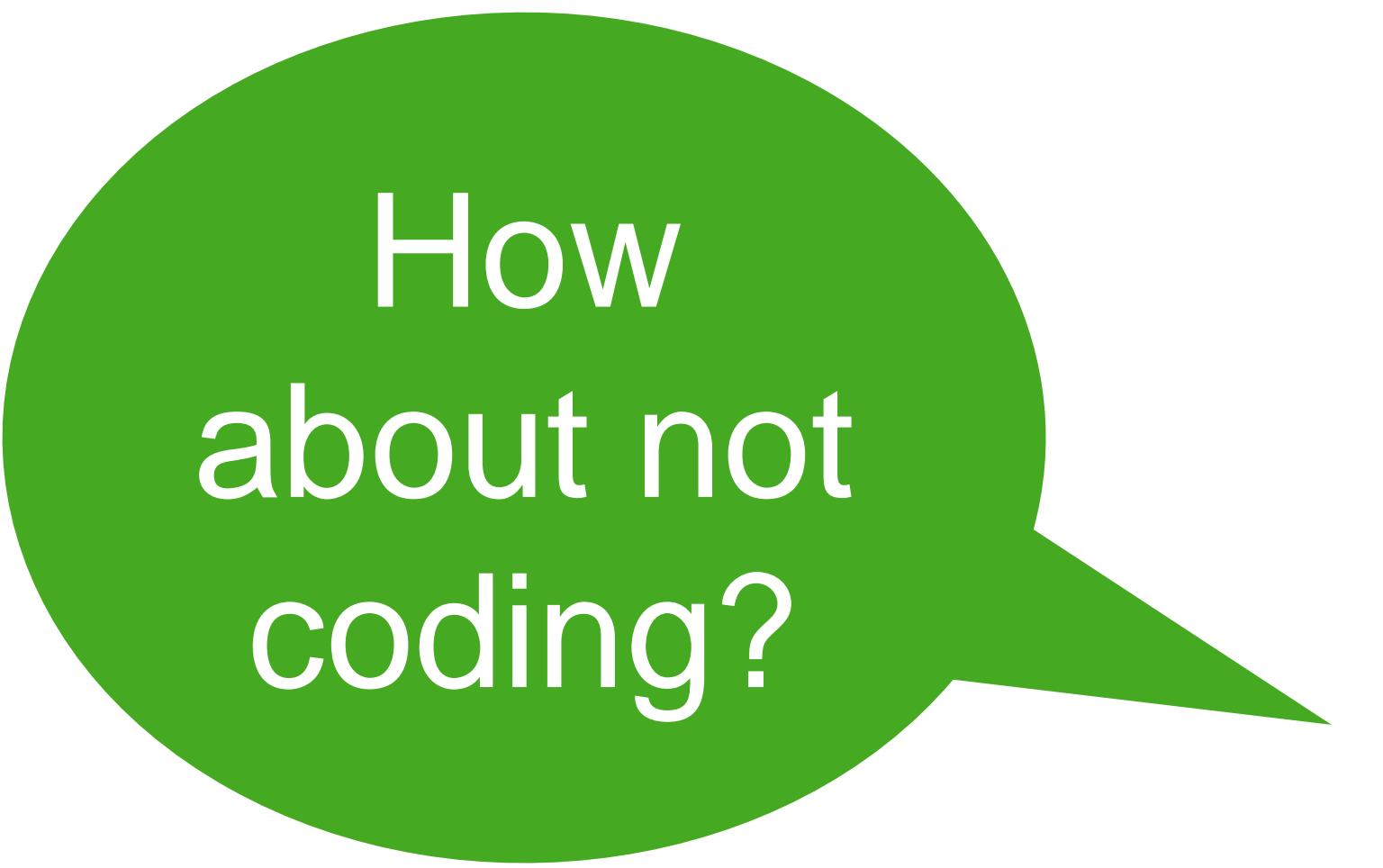


CLIJ3?

- Planned release: Summer 2021



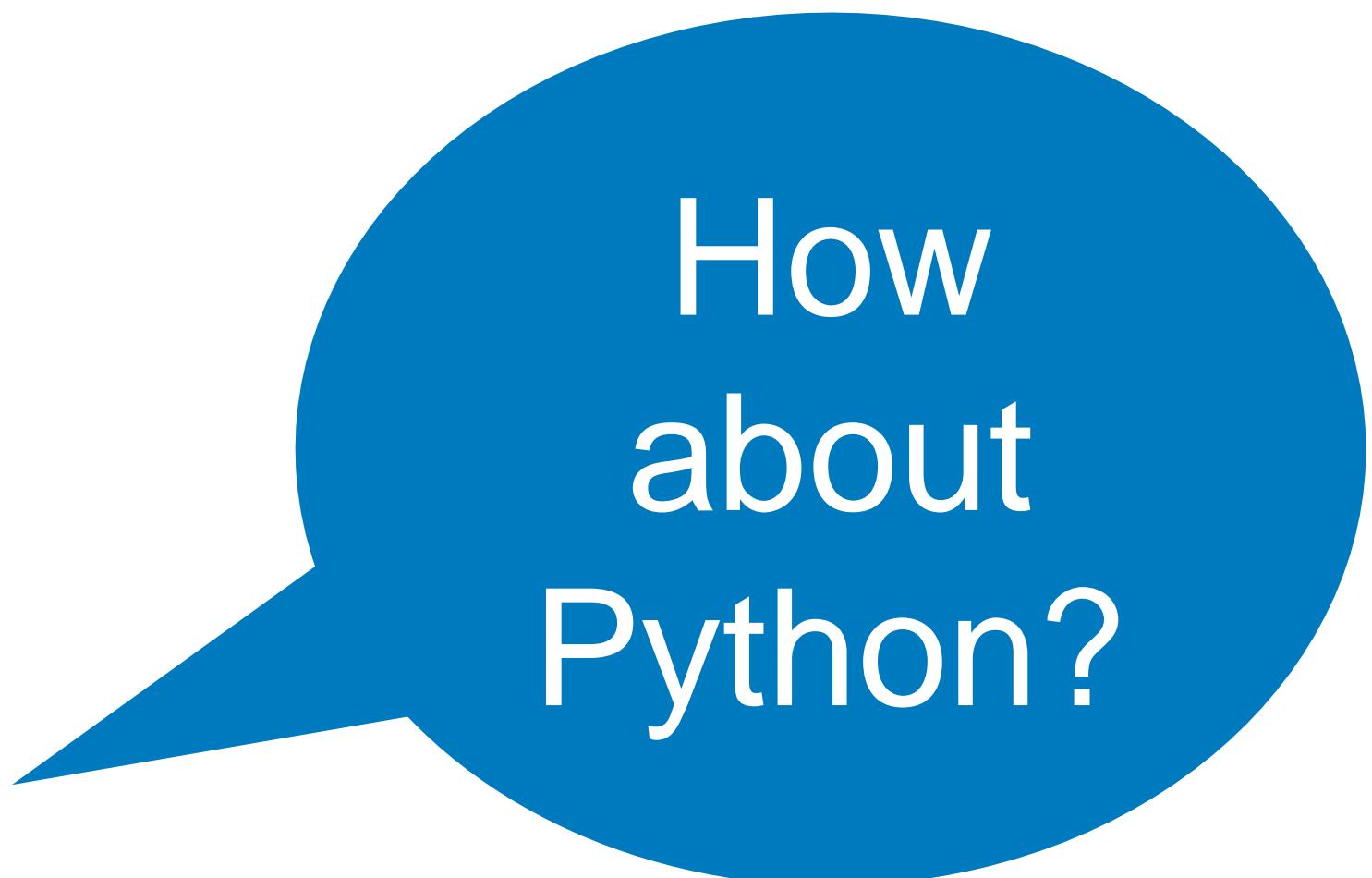
How
about
Python?



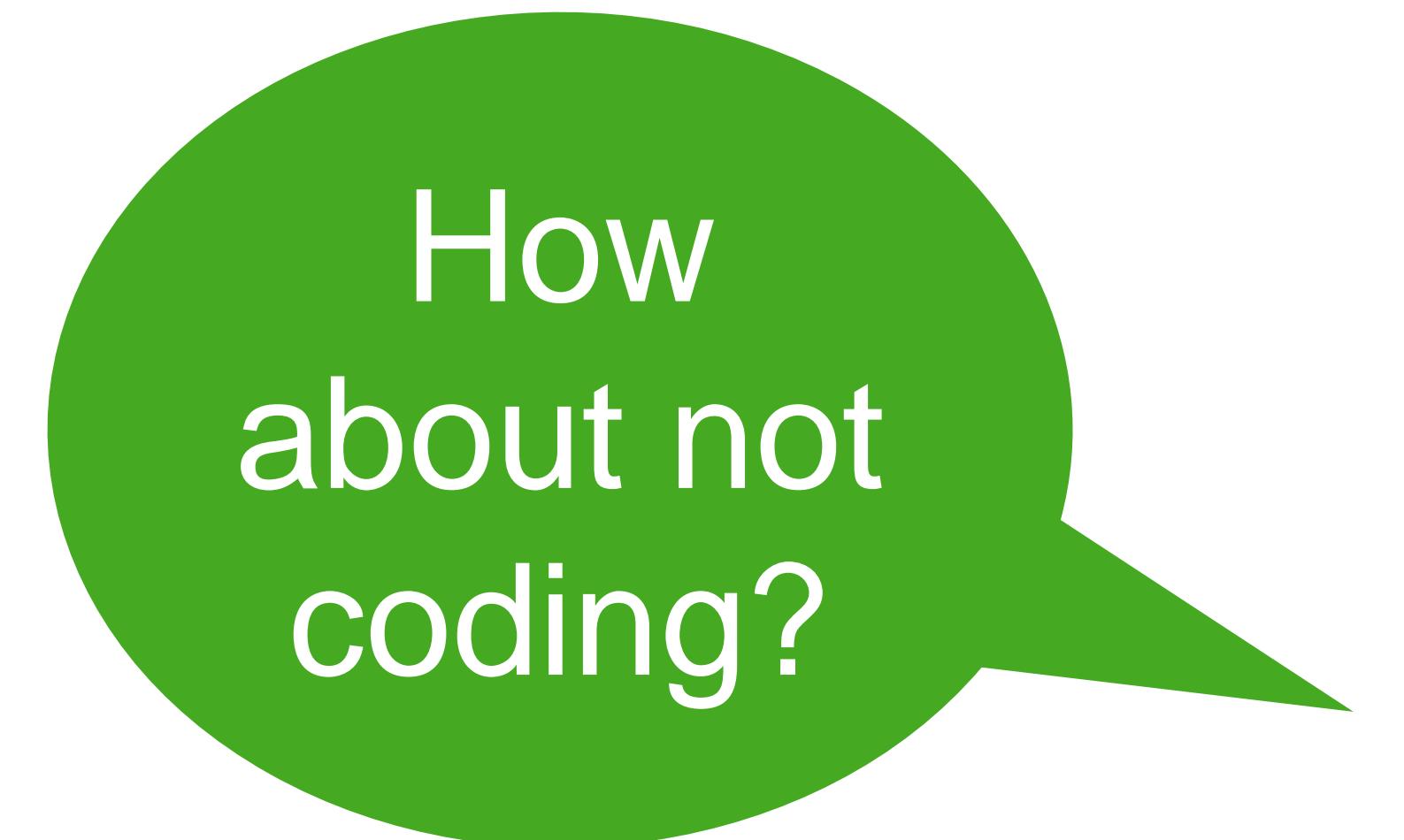
How
about not
coding?

CLIJ3? -> cEsperanto!

- Planned release: Summer 2021



How
about
Python?

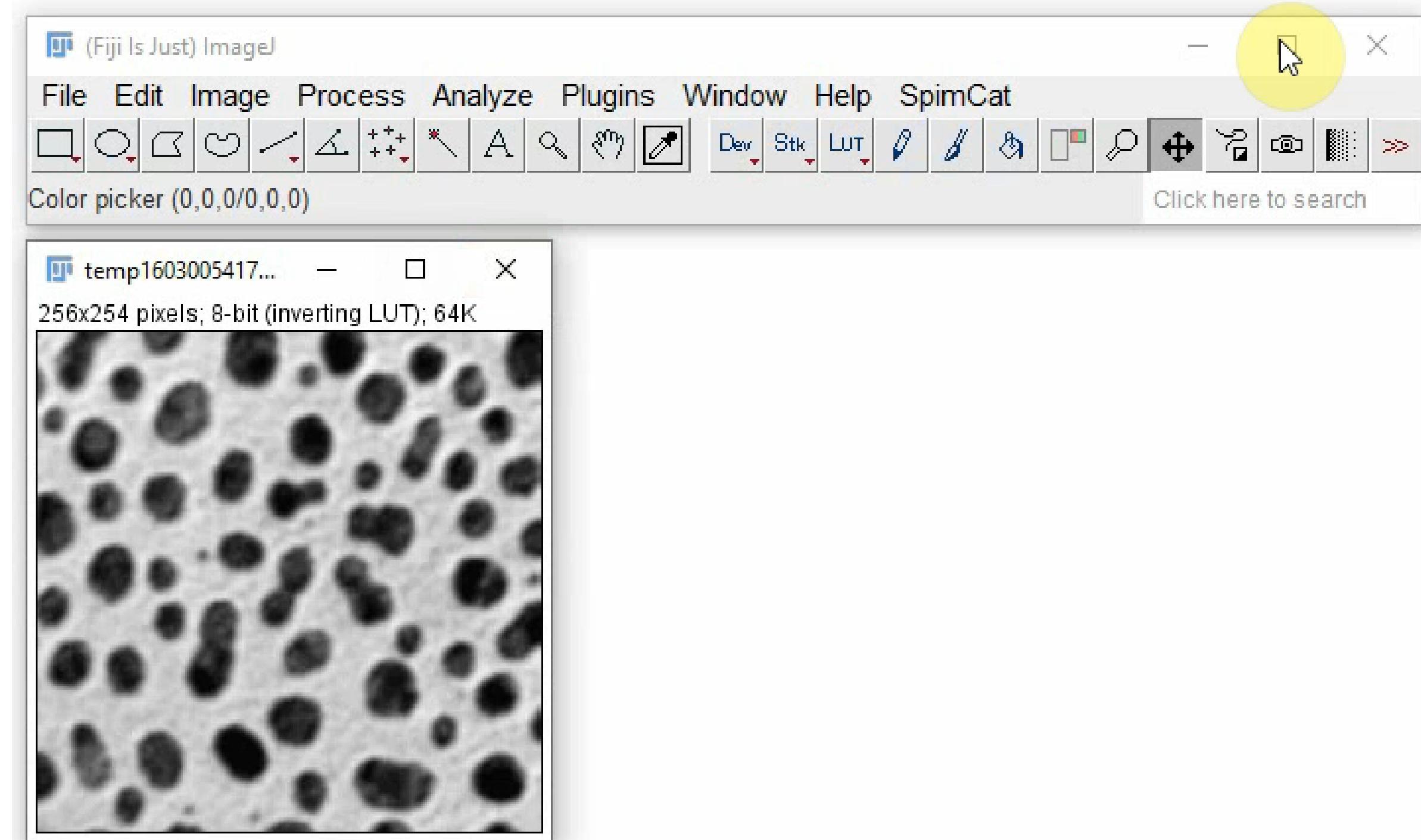


How
about not
coding?

c1e .

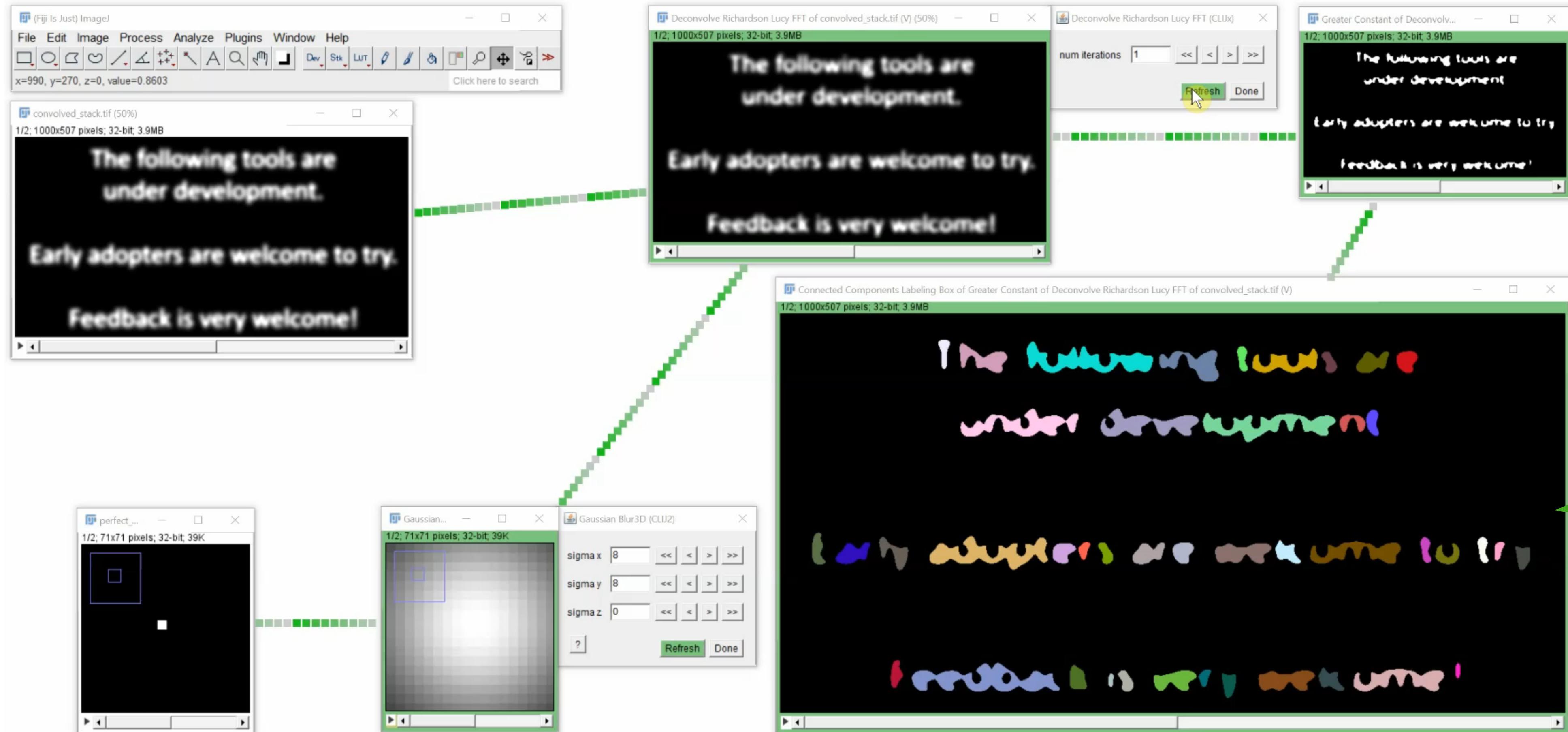
<https://clesperanto.net>

GPU acceleration – without coding



GPU acceleration – without coding

- “Image Data Flow Graphs” allow better understanding of connections between operations



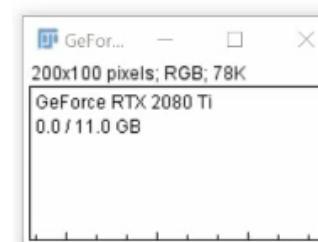
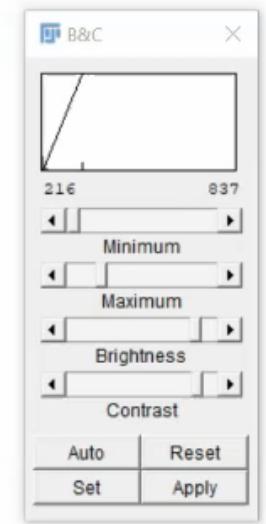
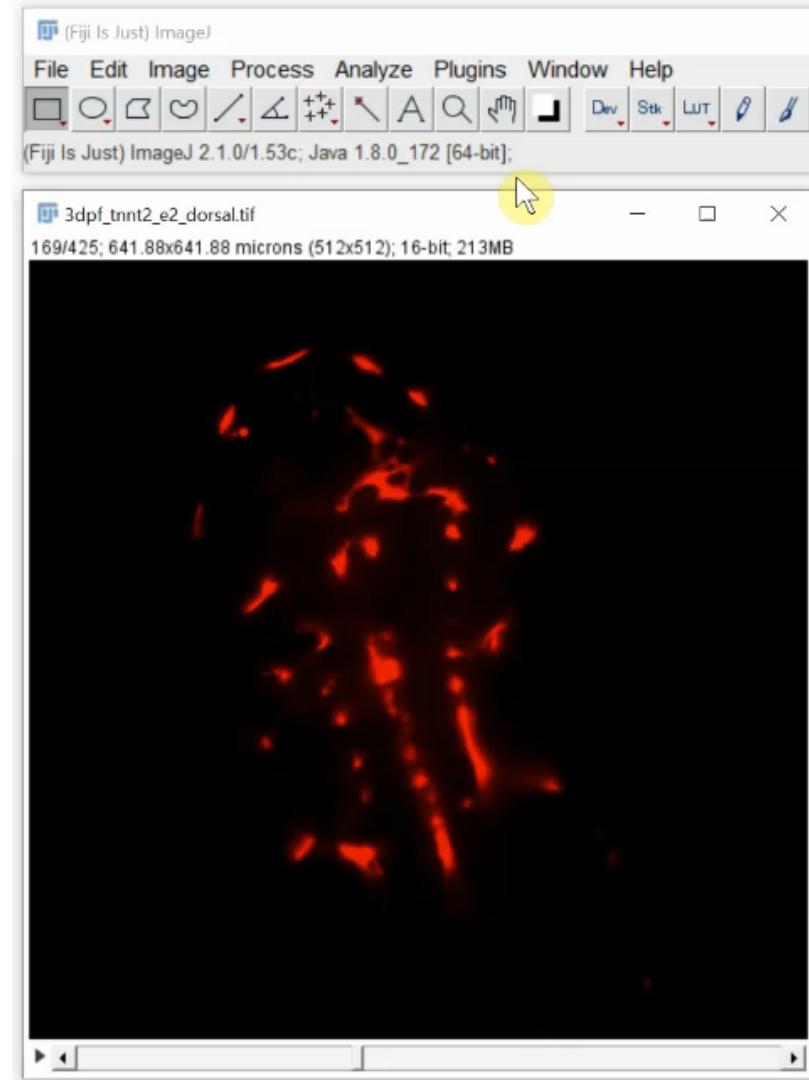
Special thanks
to Brian
Northon!



Brian Northon
@truenorth_ia

GPU acceleration + code generation

- After setting up the workflow, generate code!



Special
thanks to
Elisabeth
Kugler!



Elisabeth Kugler
@KuglerElisabeth

Image data source: Elisabeth Kugler; labs of Tim Chico and Paul Armitage, The TECHNISCHE UNIVERSITÄT DRESDEN, University of Sheffield (UK) " <https://zenodo.org/record/4204839#.X8DCRG7Q2w>

Image Data Flow Graphs & cI Esperanto

- Study relationships between operations and compare different strategies

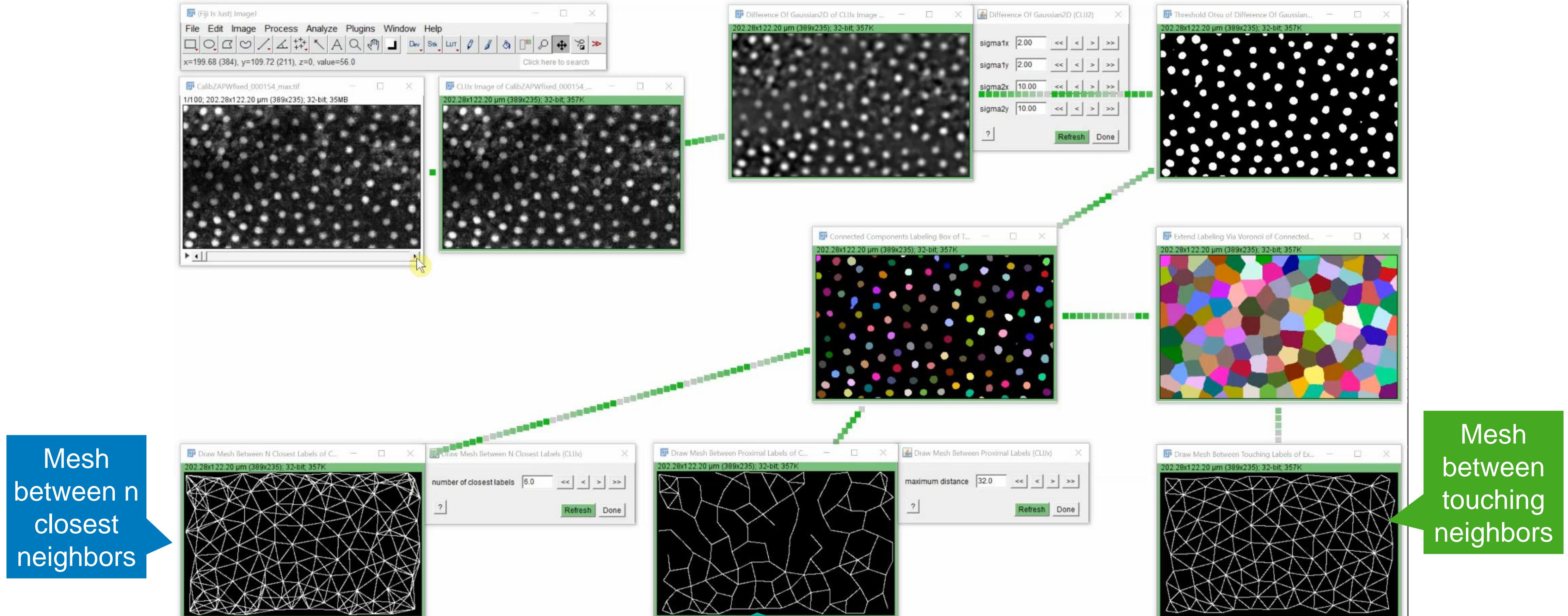
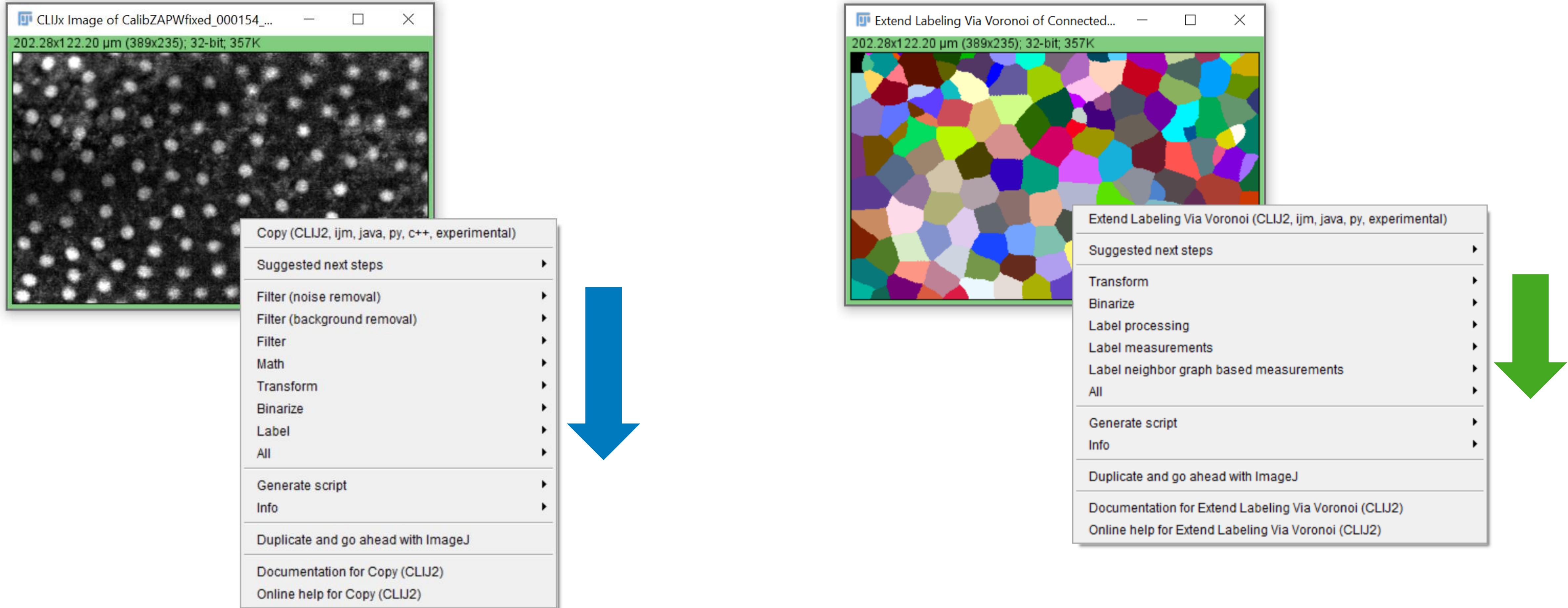


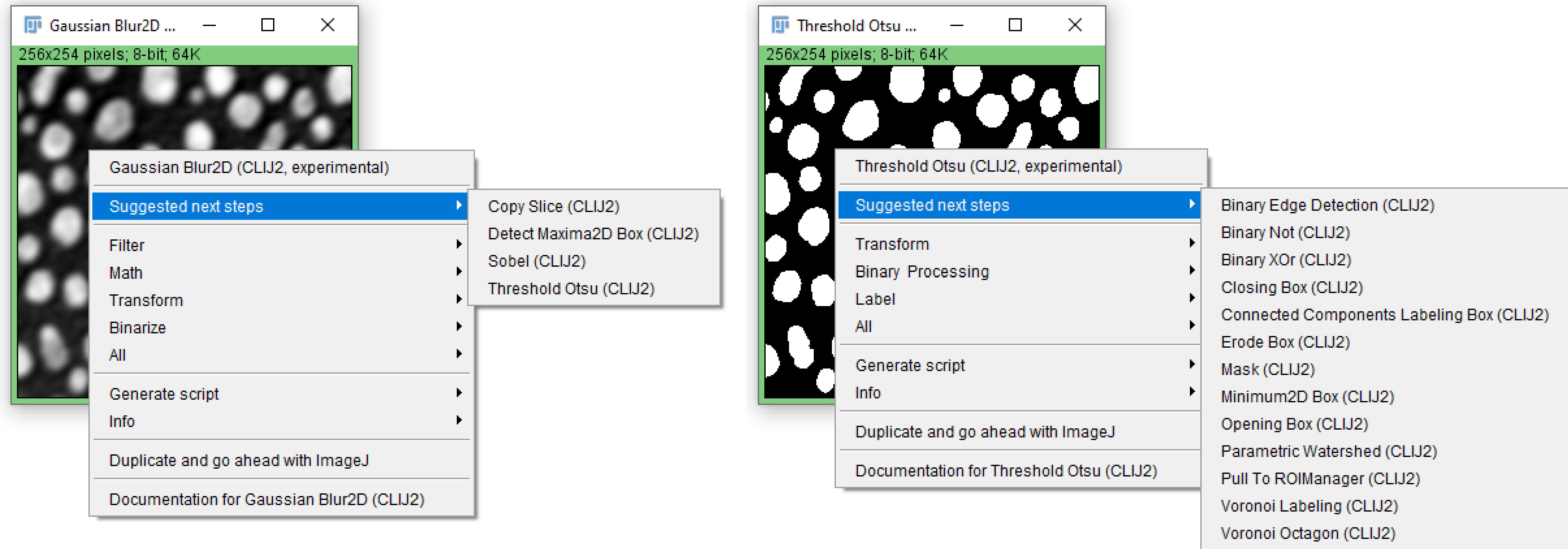
Image Data Flow Graph design

The menu order is intentional: From preprocessing to analysis



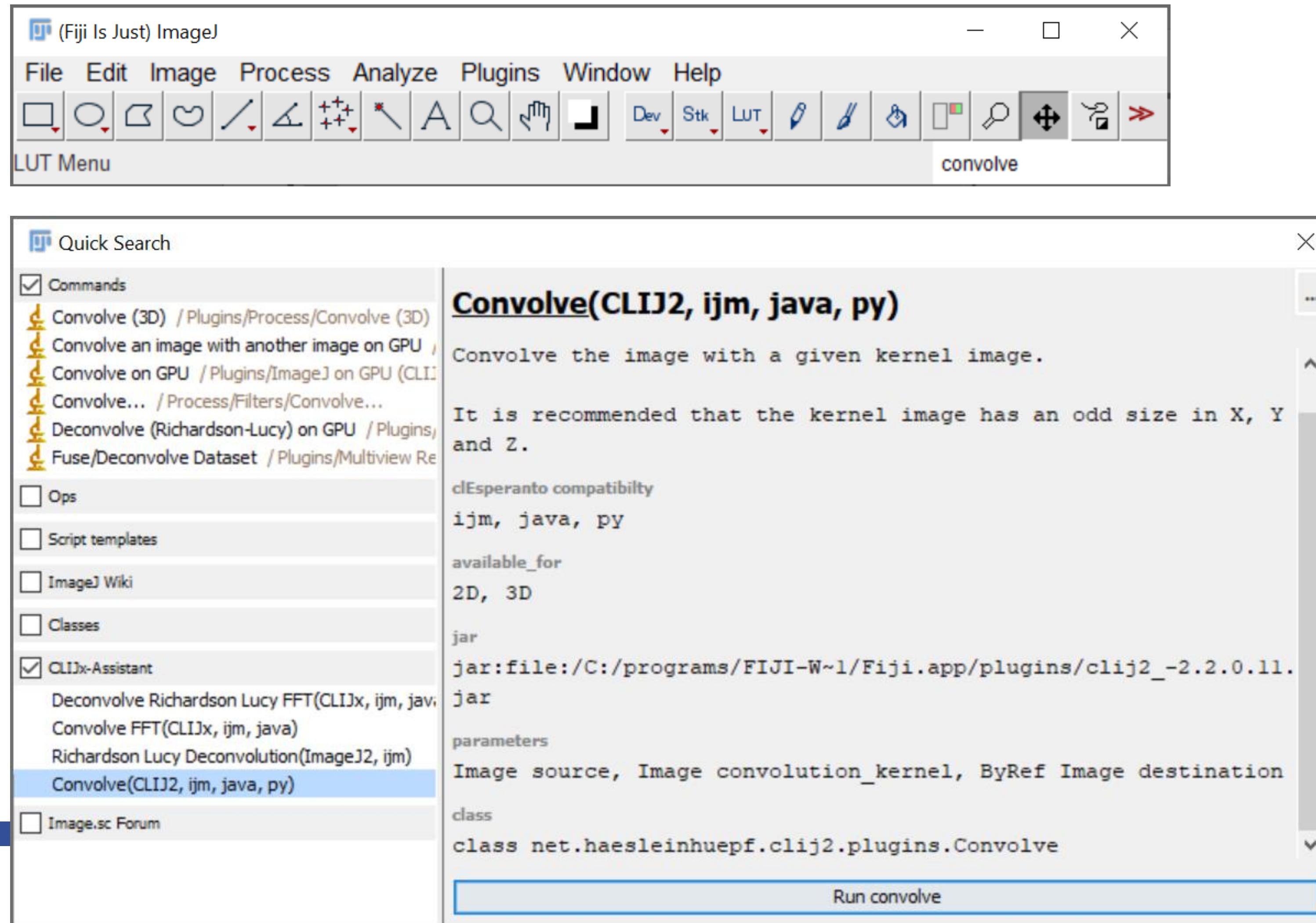
Expert system: Suggestions

Explore suggestions!



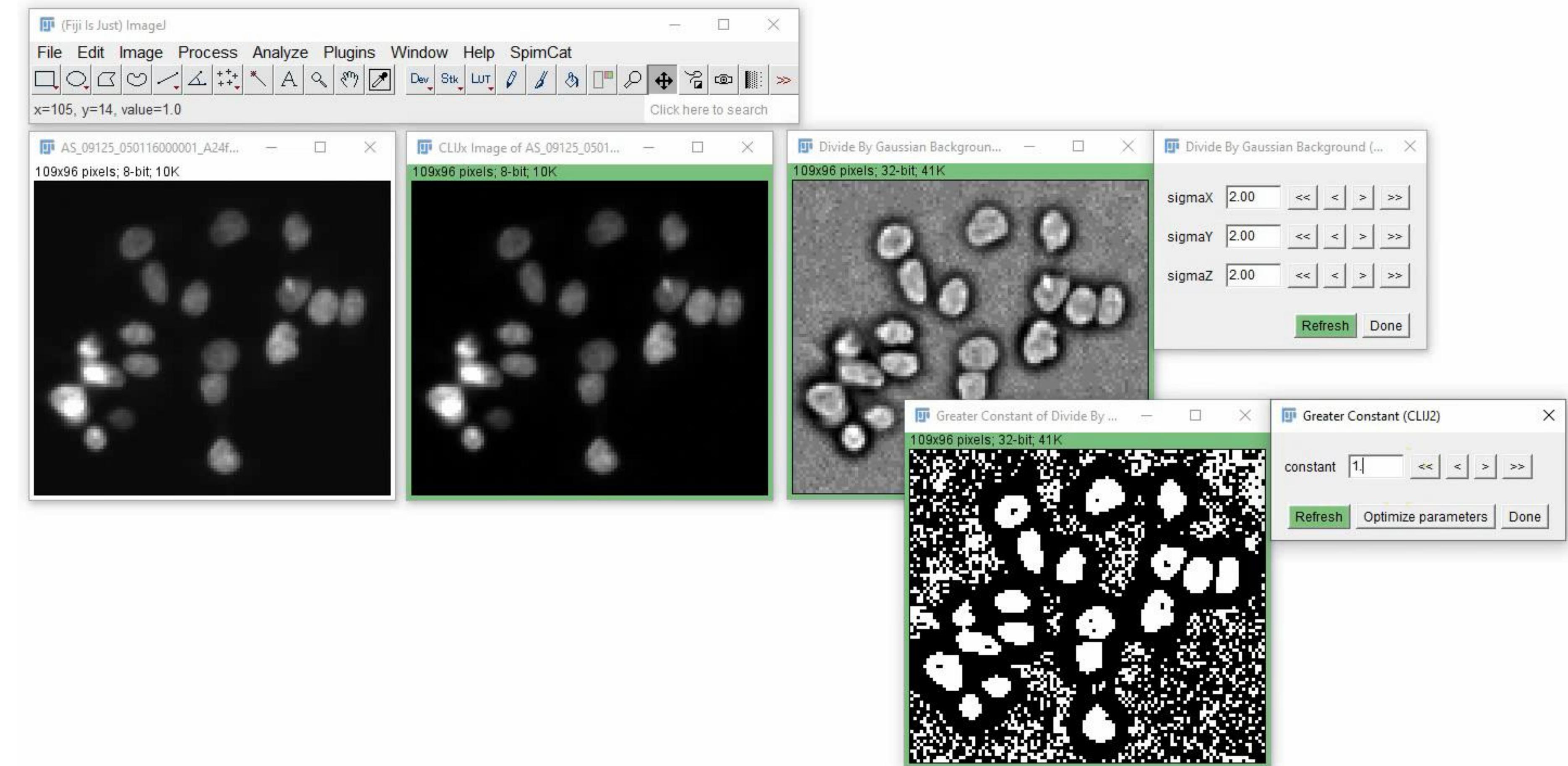
Fiji search bar

In Fijis search bar result, there is a new category: CLIJx-assistant which offers IDFG operations



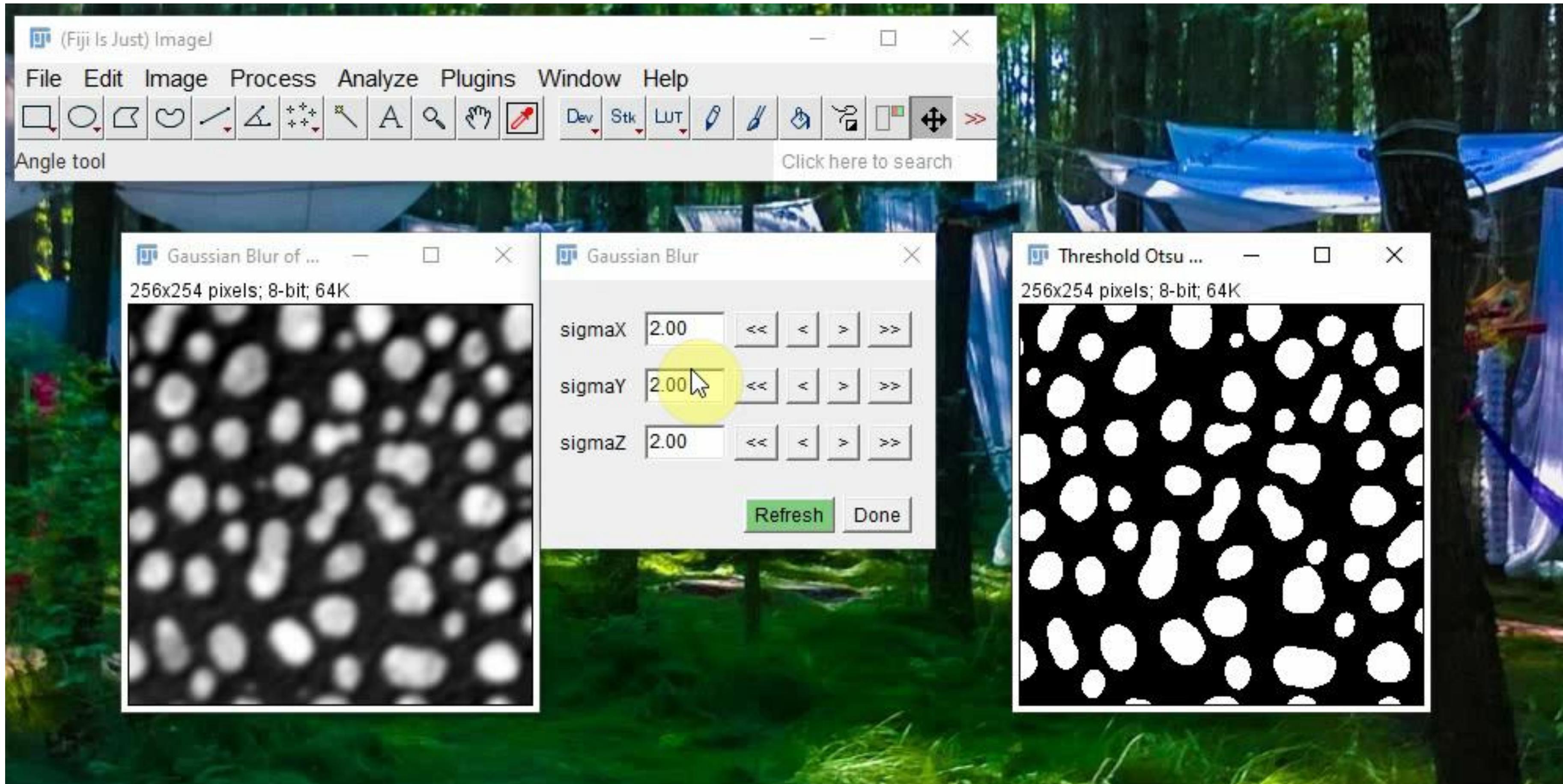
Parameter optimization

If your image data flow graph ends in a binary image, you can optimize numeric parameters automatically.



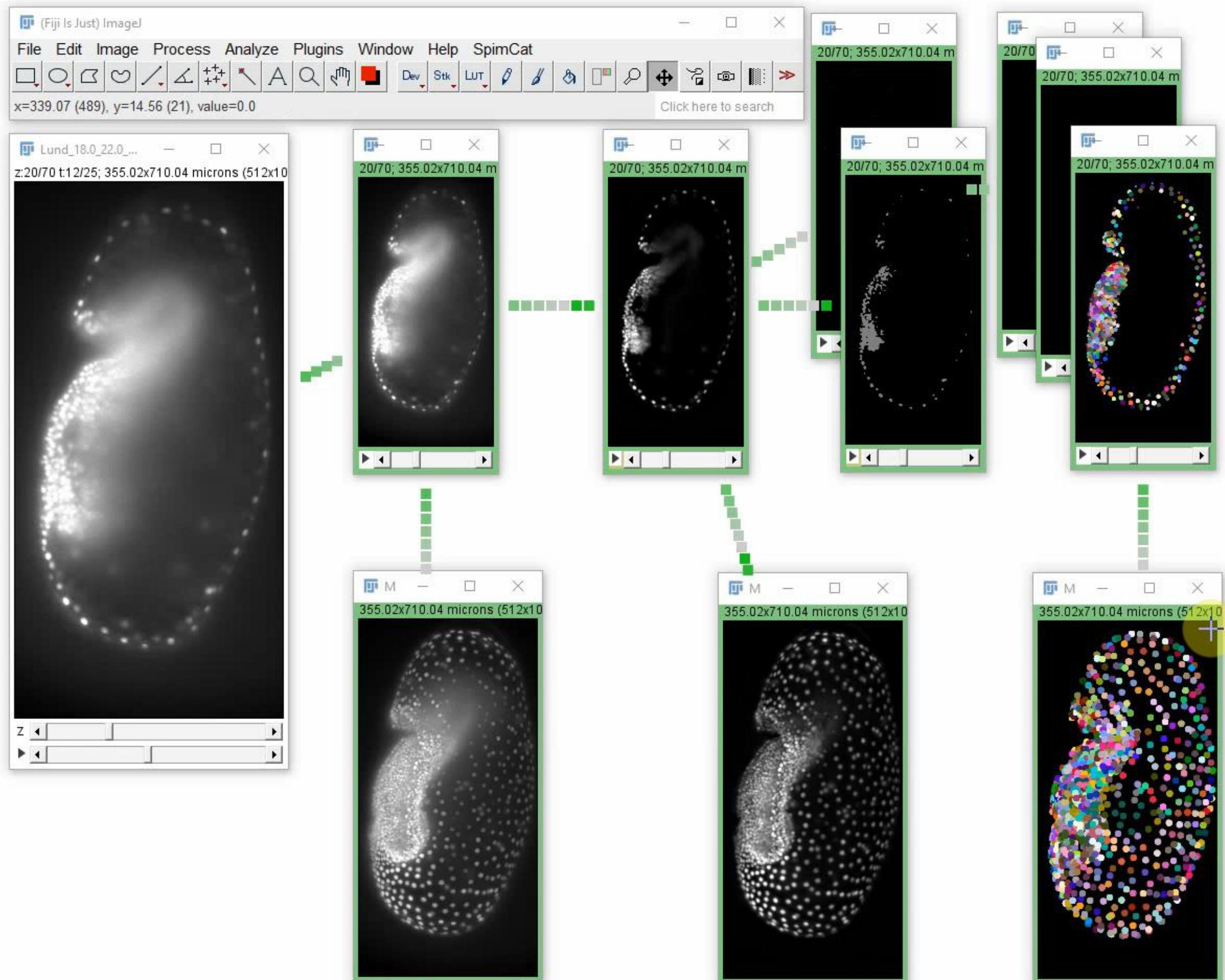
Reset parameters

If you or the optimizer screwed up parameter settings, you can get them back from the history!



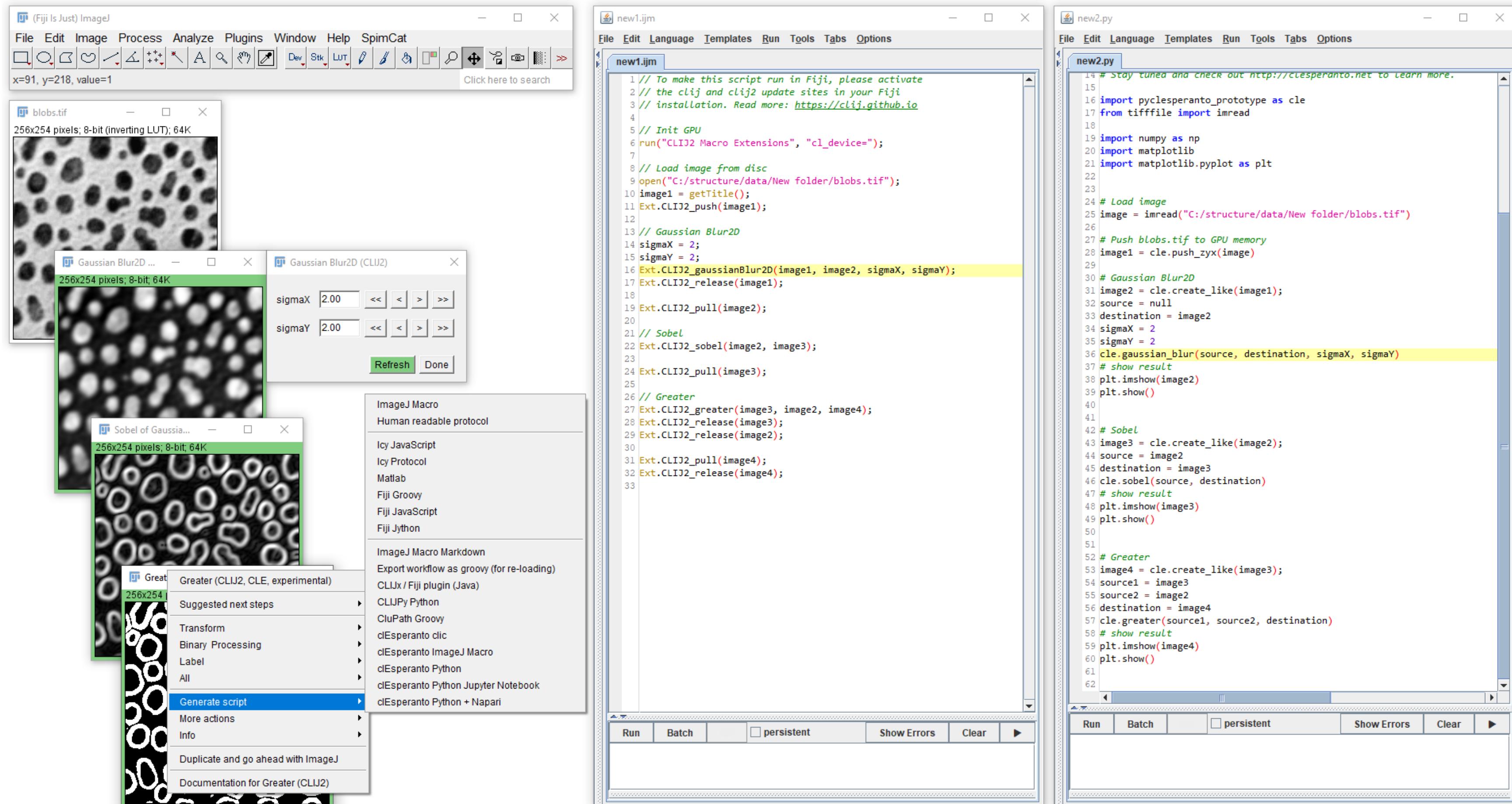
Export to scripts

Export to multiple programming languages



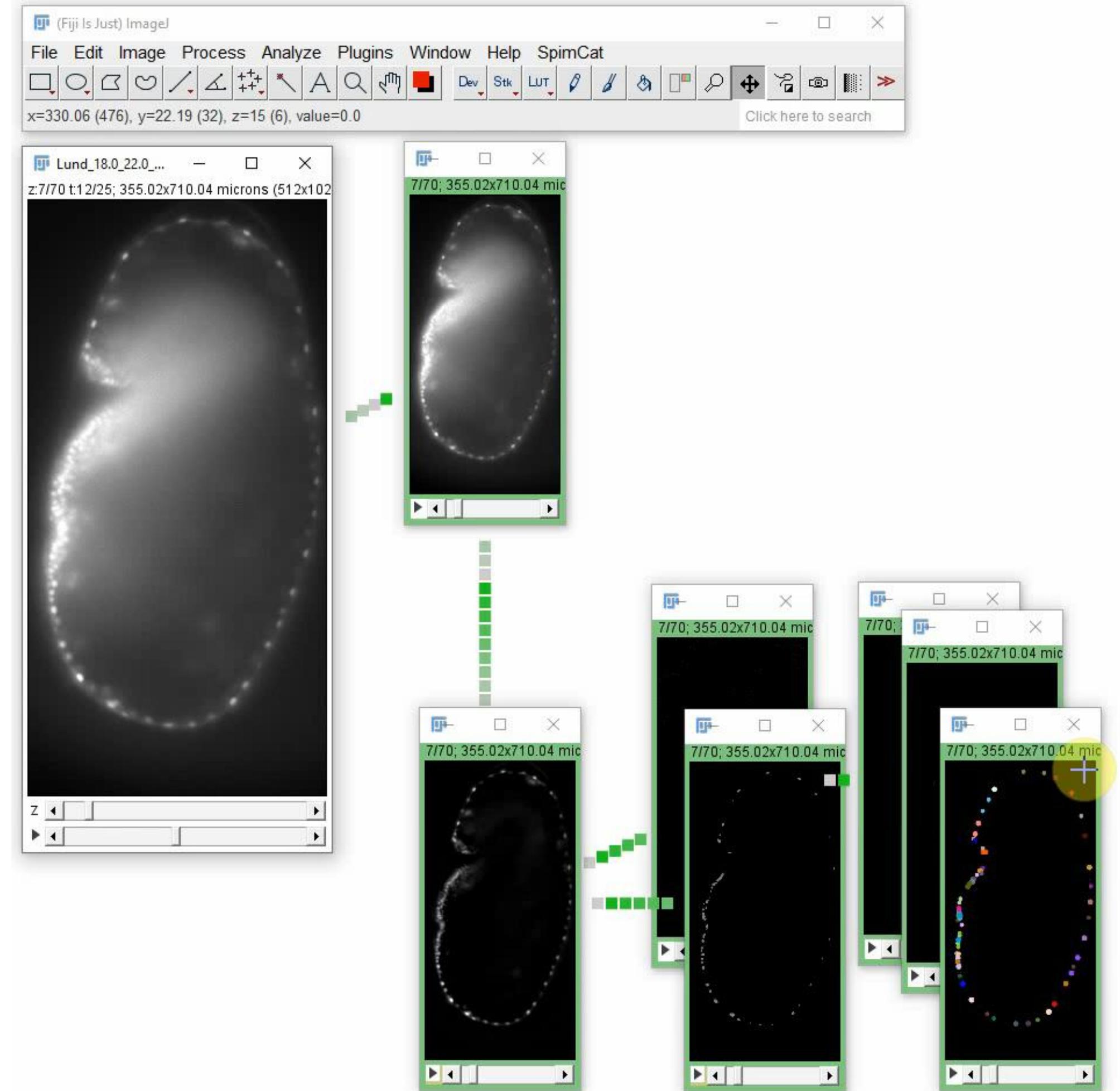
clEsperanto script language comparison

Generate multiple scripts in multiple languages from a given Image Data Flow Graph



Export from Fiji to napari

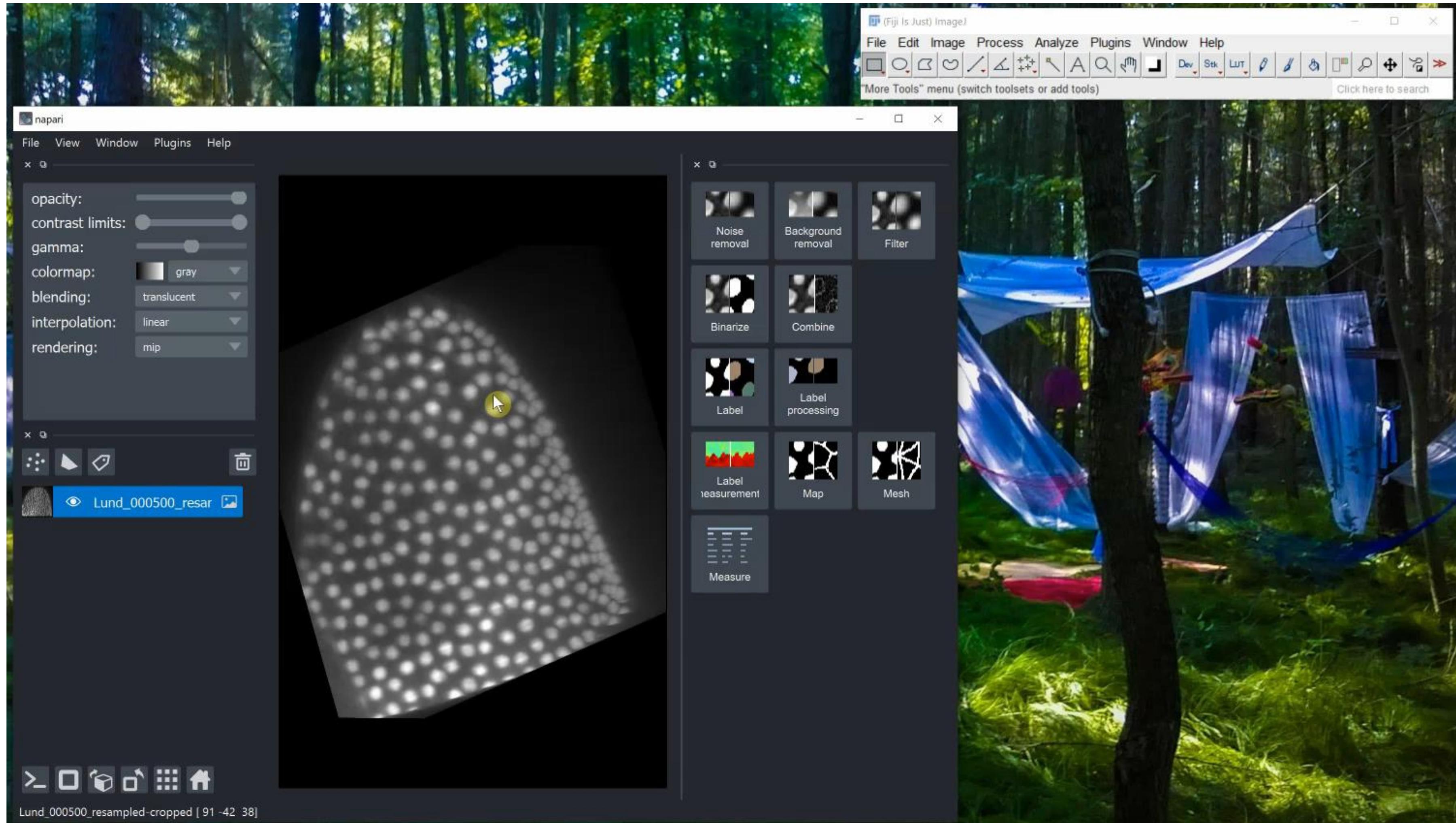
napari is a
multidimensional
image viewer in
the Python
ecosystem



https://clij.github.io/assistant/te_oki_export

Export from napari to Fiji

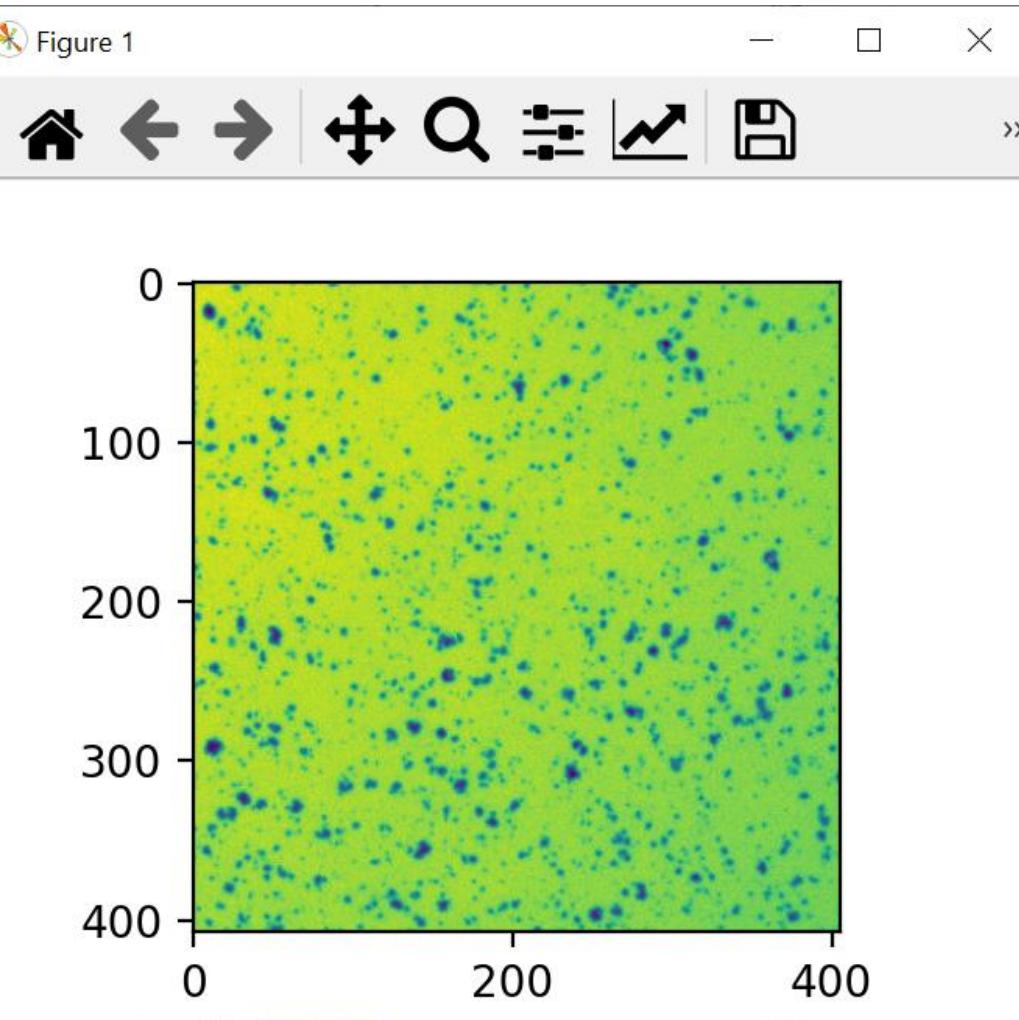
Using the napari-
pyclesperanto-assistant



py-clEsperanto: What every Python script must have

Load data

```
1 from skimage.io import imread
2 import matplotlib.pyplot as plt
3
4 # Load image
5 image = imread("https://imagej.nih.gov/ij/images/Cell_Colony.jpg")
6 plt.imshow(image)
7 plt.show()
```



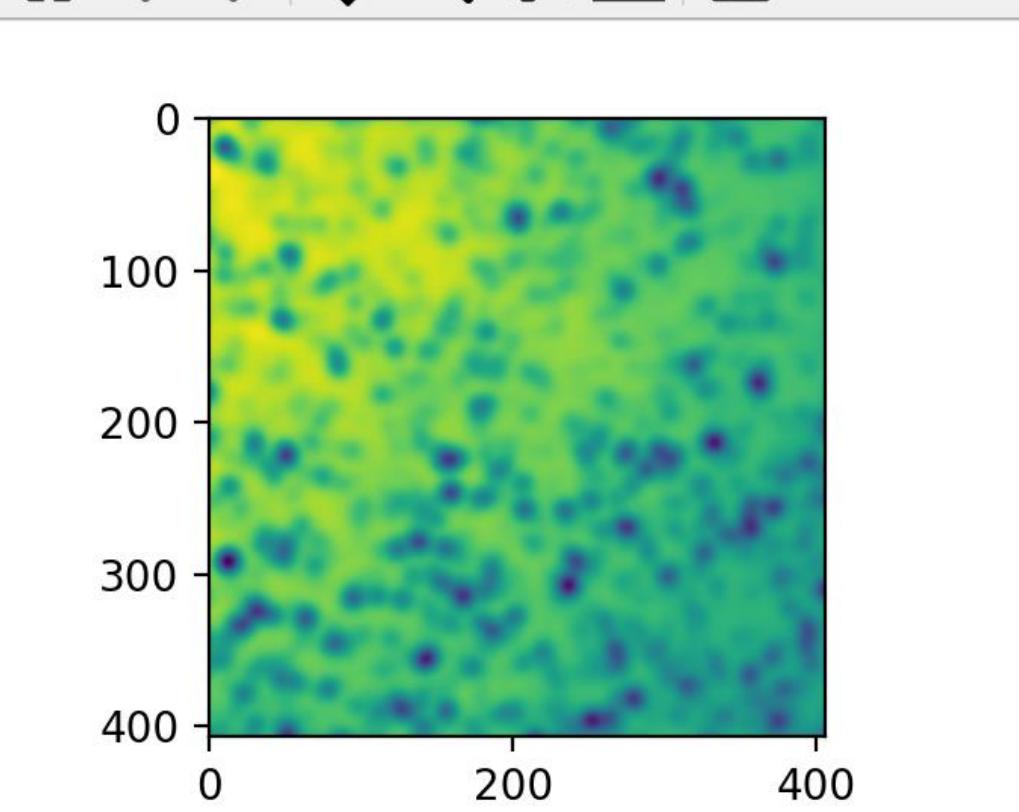
Initialize GPU

```
9 # initialize GPU
10 import pyclesperanto_prototype as cle
11 # optional:
12 cle.select_device("GPU name")
13
```



Push

```
14 # Push blobs.tif to GPU memory
15 image1 = cle.push_zyx(image)
16
17 # Threshold Otsu
18 sigma = 5
19 blurred = cle.create_like(image1)
20 cle.gaussian_blur(image1, blurred, sigma, sigma)
21
22 # pull result back from GPU
23 result = cle.pull_zyx(blurred)
24 plt.imshow(result)
25 plt.show()
```



Process images

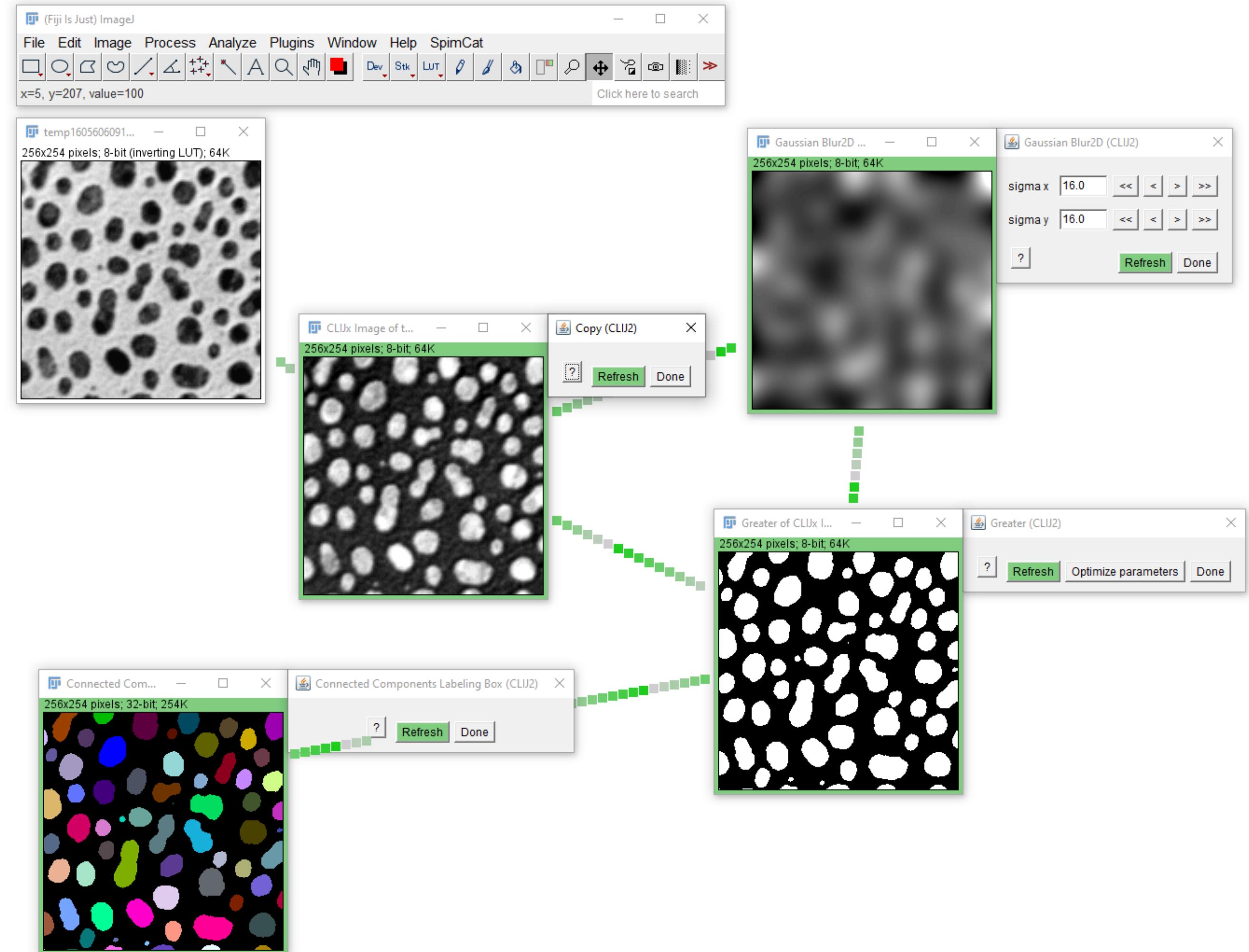
Pull

Exercises

Exercise 3: Segment blobs.gif

Design a workflow for segmenting blobs.gif (File > Open Samples...)

- Export the workflow as ImageJ Macro script for and Fiji Jython.
- Optional: Export the workflow as Icy Javascript, as Icy protocol and for QuPath as groovy script. Feel free to generate a Fiji plugin.
- Furthermore, if you only used operations, which are "py" compatible, export a cIEsperanto-based Jupyter notebook and a Python script that uses Napari.



Exercise 4: Install & test pyclesperanto

Install pyclesperanto

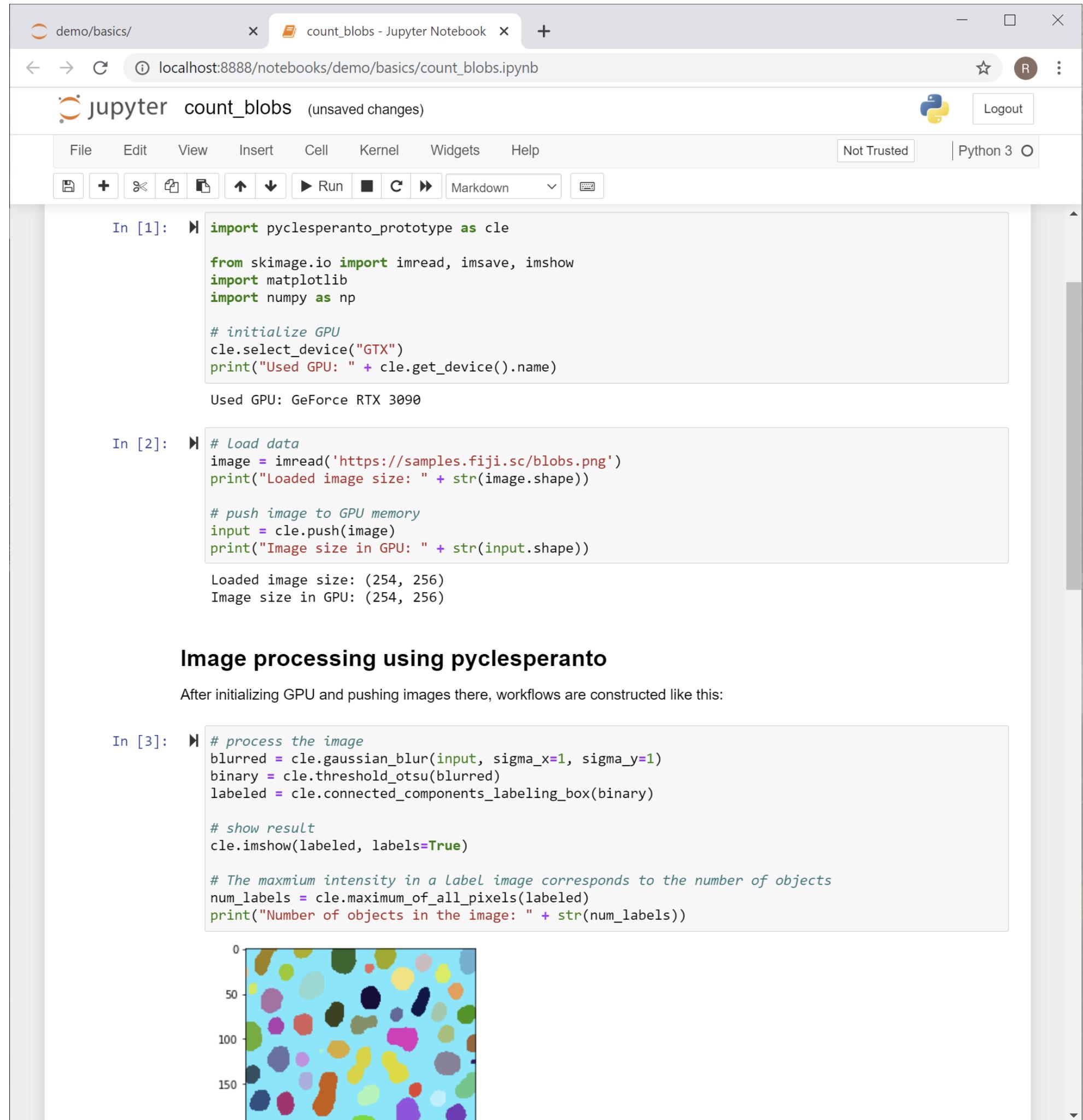
```
pip install pyclesperanto_prototype
```

And download & execute this notebook:

https://github.com/cI Esperanto/pyclesperanto_prototype/blob/master/demo/basics/count_blobs.ipynb

Linux users: you need to install OpenCL, e.g.

```
apt-get install ocl-icd-devel
```



```
In [1]: import pyclesperanto_prototype as cle
from skimage.io import imread, imsave, imshow
import matplotlib
import numpy as np

# initialize GPU
cle.select_device("GTX")
print("Used GPU: " + cle.get_device().name)

Used GPU: GeForce RTX 3090

In [2]: # Load data
image = imread('https://samples.fiji.sc/blobs.png')
print("Loaded image size: " + str(image.shape))

# push image to GPU memory
input = cle.push(image)
print("Image size in GPU: " + str(input.shape))

Loaded image size: (254, 256)
Image size in GPU: (254, 256)

In [3]: # process the image
blurred = cle.gaussian_blur(input, sigma_x=1, sigma_y=1)
binary = cle.threshold_otsu(blurred)
labeled = cle.connected_components_labeling_box(binary)

# show result
cle.imshow(labeled, labels=True)

# The maximum intensity in a Label image corresponds to the number of objects
num_labels = cle.maximum_of_all_pixels(labeled)
print("Number of objects in the image: " + str(num_labels))

0 50 100 150
```

Image processing using pyclesperanto

After initializing GPU and pushing images there, workflows are constructed like this:

```
# process the image
blurred = cle.gaussian_blur(input, sigma_x=1, sigma_y=1)
binary = cle.threshold_otsu(blurred)
labeled = cle.connected_components_labeling_box(binary)

# show result
cle.imshow(labeled, labels=True)

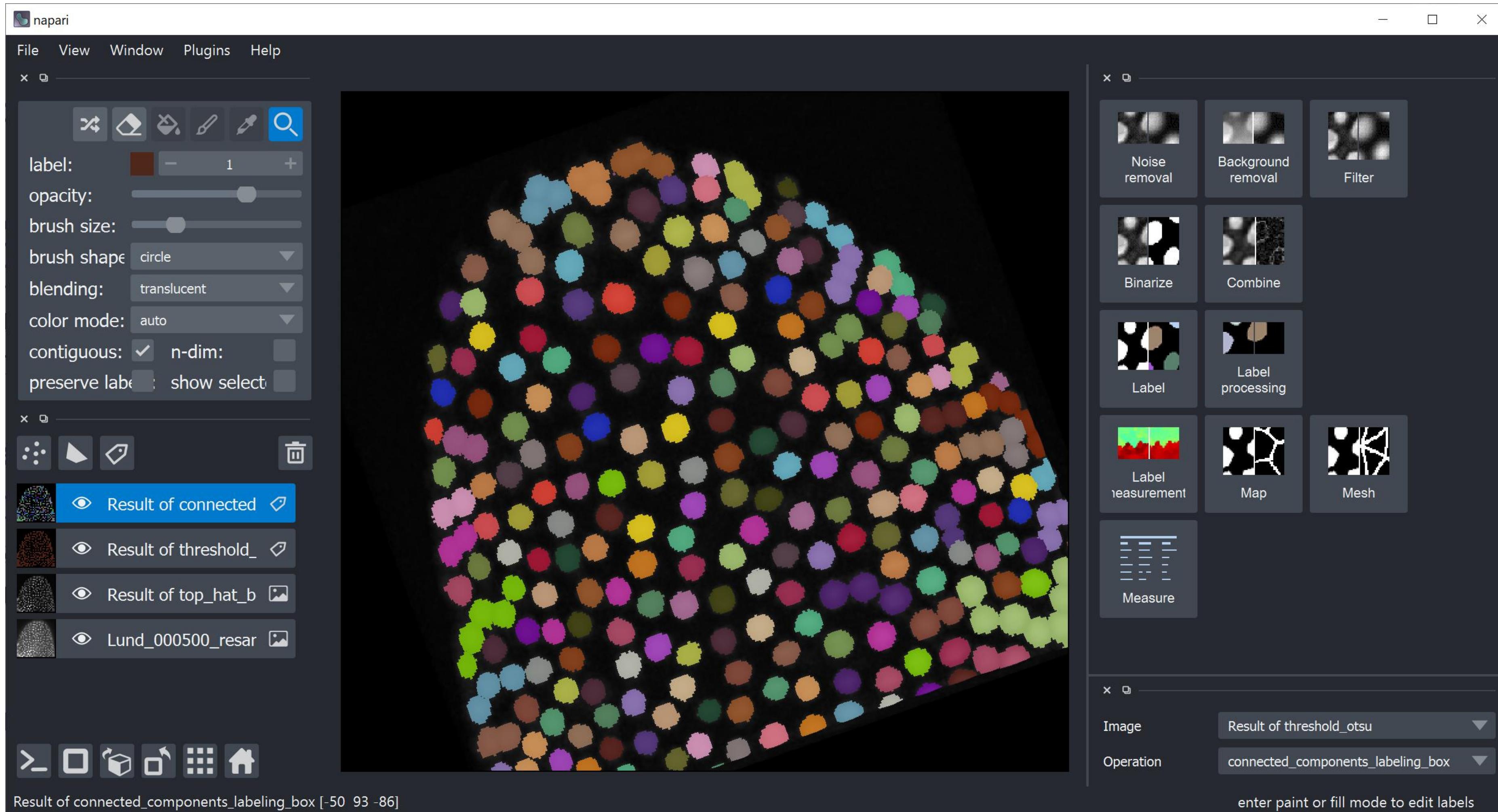
# The maximum intensity in a Label image corresponds to the number of objects
num_labels = cle.maximum_of_all_pixels(labeled)
print("Number of objects in the image: " + str(num_labels))
```



Exercise 5: Napari-pyclesperanto-assistant

Install and try the assistant for napari

https://clesperanto.git-hub.io/napari_pyclesperanto_assistant/



Exercise 6: Benchmarking: reuse memory

Compare reusing memory and re-allocating memory. Execute some operations in a loop and measure processing time.

Reuse memory

```
for i in range(0, num_iterations):
    cle.maximum_sphere(flip, flop, 10, 10, 0)
    cle.minimum_sphere(flop, flip, 10, 10, 0)
```

Re-allocate memory

```
for i in range(0, num_iterations):
    flop = cle.maximum_sphere(flip, radius_x=10, radius_y=10, radius_z=0)
    flip = cle.minimum_sphere(flop, radius_x=10, radius_y=10, radius_z=0)
```

```
c:\ Command Prompt
(beetlesafari) C:\structure\code\i2k2020_tutor
Used GPU: GeForce RTX 2080 Ti
Image size: (50, 1024, 1024)
Flip-flop took 0.3108978271484375s
Flip-flop took 0.21841955184936523s
Flip-flop took 0.21937274932861328s
Flip-flop took 0.21941328048706055s
Flip-flop took 0.22044825553894043s
Flip-flop took 0.220412015914917s
Flip-flop took 0.21941876411437988s
Flip-flop took 0.2214047908782959s
Flip-flop took 0.22040820121765137s
Flip-flop took 0.22040891647338867s
Re-alloc took 0.3904910087585449s
Re-alloc took 0.3560006618499756s
Re-alloc took 0.36129093170166016s
Re-alloc took 0.36267709732055664s
Re-alloc took 0.3624119758605957s
Re-alloc took 0.36183905601501465s
Re-alloc took 0.3612239360809326s
Re-alloc took 0.363983154296875s
Re-alloc took 0.36783385276794434s
Re-alloc took 0.35245370864868164s

(beetlesafari) C:\structure\code\i2k2020_tutor
```

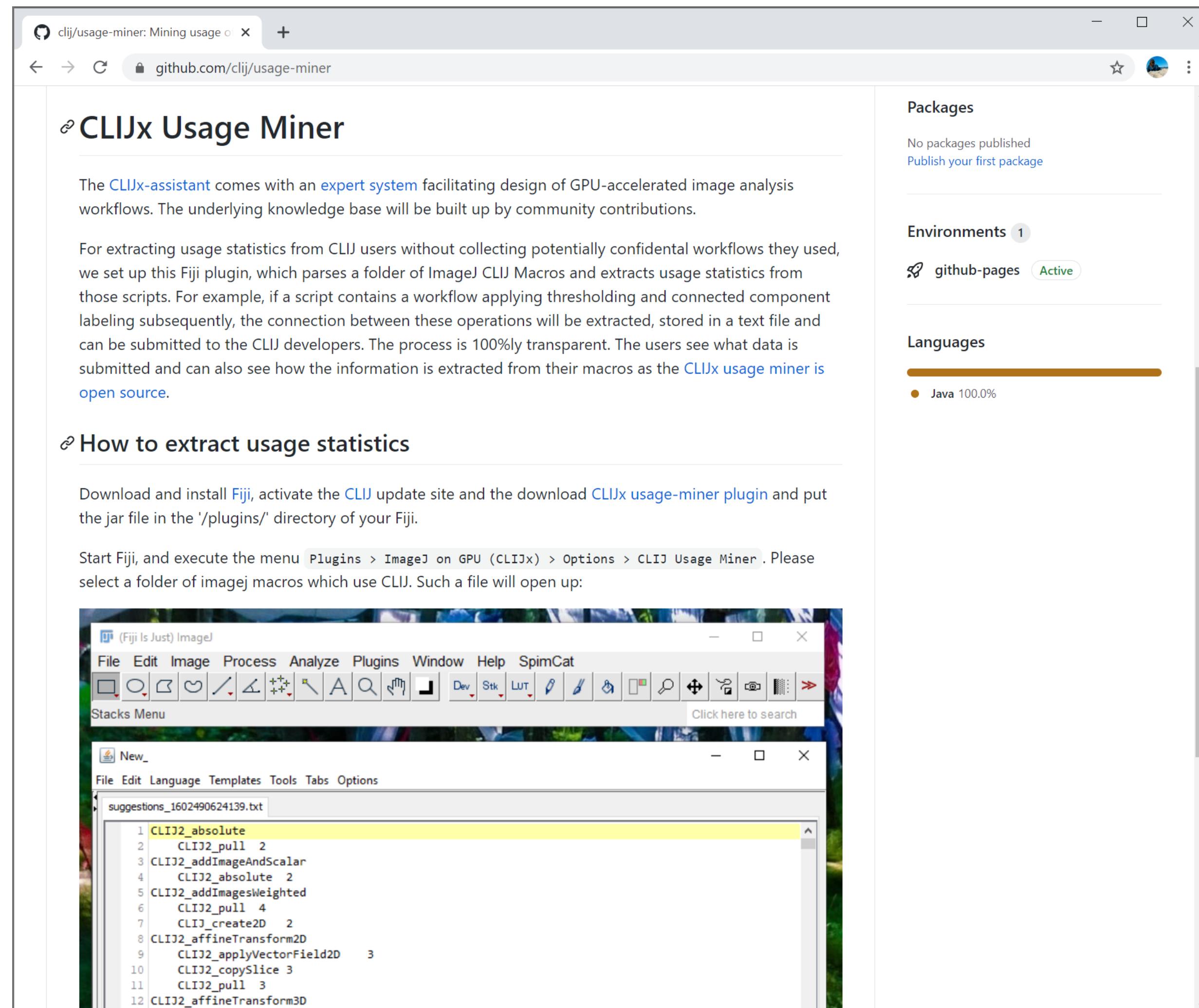
Exercise 7: Usage miner

Homework: After you implemented some ImageJ Macros at home, let us analyse your command usage statistics. Visit the usage-miner website to learn how.

<https://github.com/clij/usage-miner>

With your contribution, we can make suggestions better!

Thanks 😊



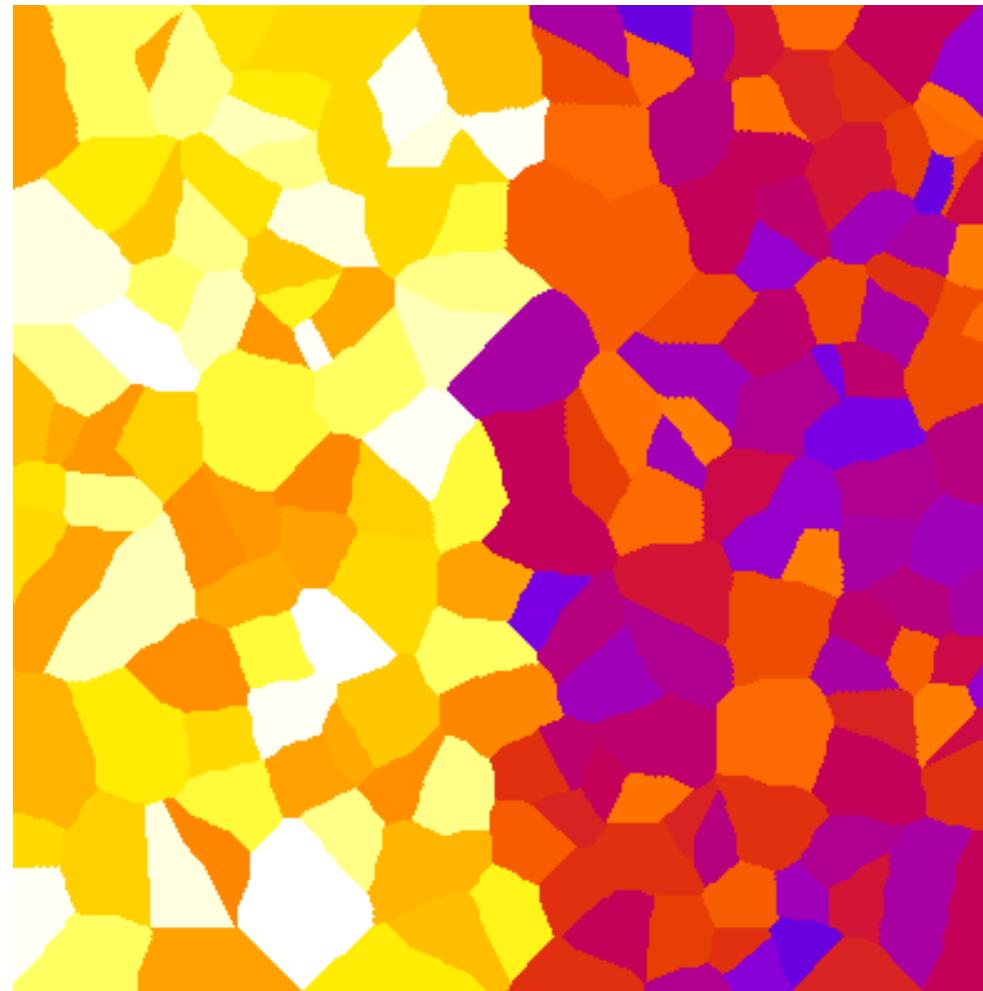
The screenshot shows the GitHub page for the CLIJx Usage Miner. The page title is "CLIJx Usage Miner". It explains that the CLIJx-assistant comes with an expert system facilitating design of GPU-accelerated image analysis workflows. It mentions that the plugin parses ImageJ CLIJ Macros and extracts usage statistics. A note states that the process is 100% transparent and the code is open source. Below this, there's a section titled "How to extract usage statistics" which provides instructions for using the plugin in Fiji. The Fiji application window is shown in the background, displaying a stack of images and a text editor window titled "suggestions_1602490624139.txt" containing a list of CLIJ2 command names and their counts.

```
suggestions_1602490624139.txt
1 CLIJ2_absolute
2 CLIJ2_pull 2
3 CLIJ2_addImageAndScalar
4 CLIJ2_absolute 2
5 CLIJ2_addImagesWeighted
6 CLIJ2_pull 4
7 CLIJ_create2D 2
8 CLIJ2_affineTransform2D
9 CLIJ2_applyVectorField2D 3
10 CLIJ2_copySlice 3
11 CLIJ2_pull 3
12 CLIJ2_affineTransform3D
13 CLIJ2_gaussianBlur 2
```

Part III: Life-sciences specific image processing

Operations on adjacency graphs

- “Pixels aren’t necessarily squares. They can have any shape.”



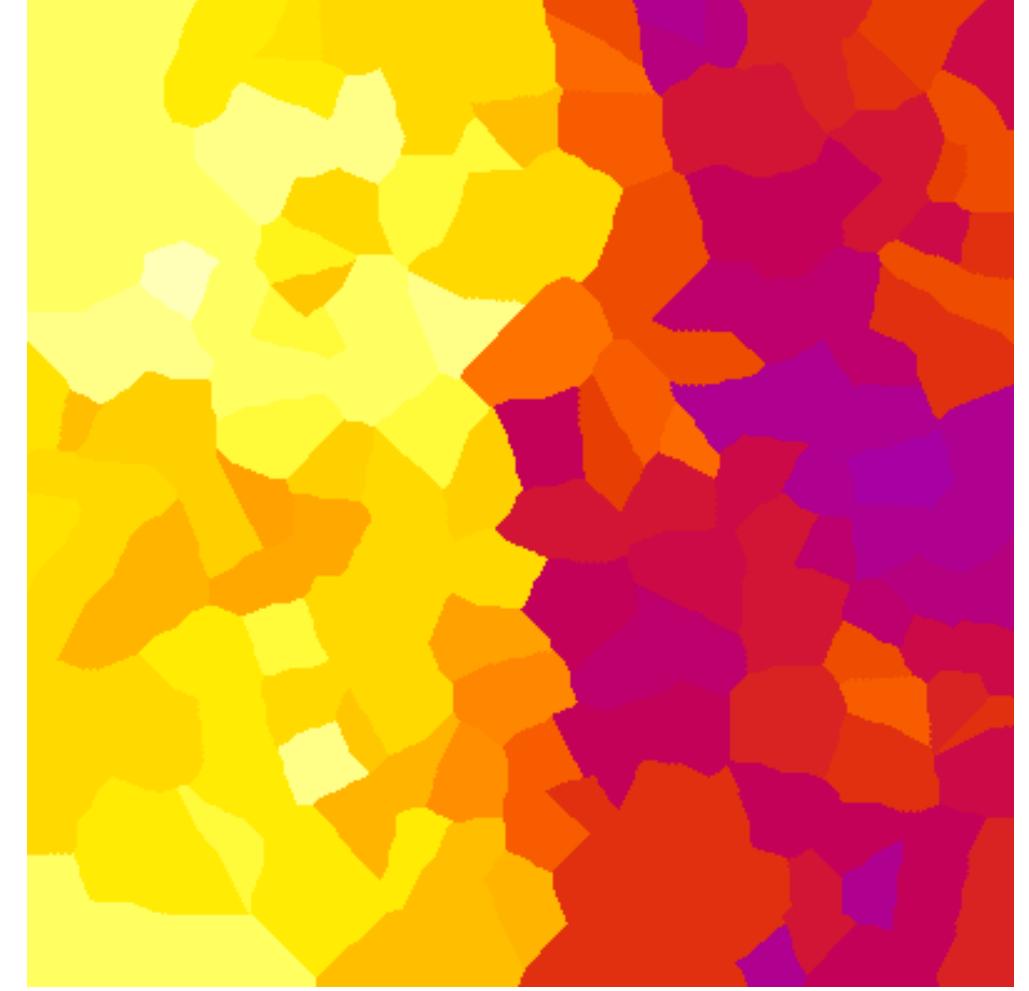
Original



Mean filtered



Mean filtered
(wider radius)



Median filtered

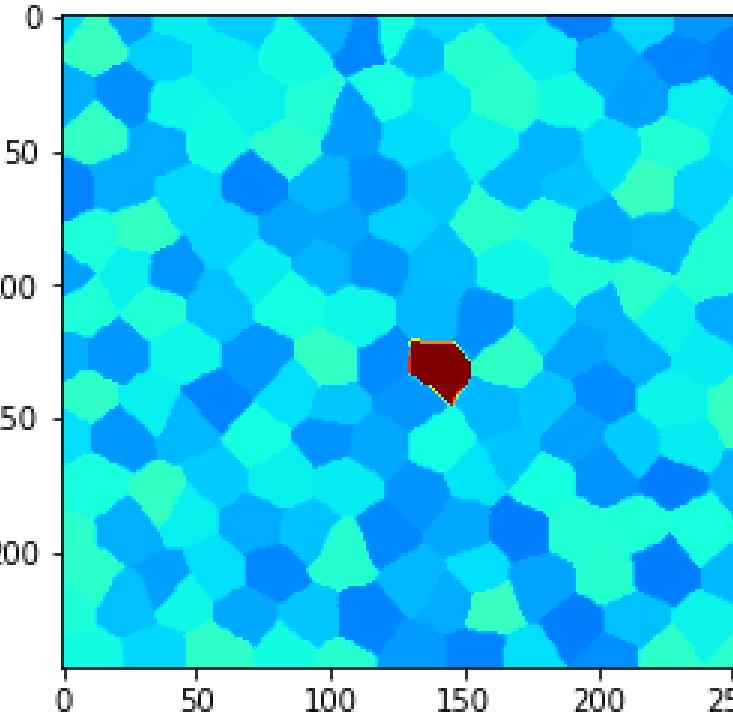


I wasn’t aware of any software where you can just do these operations.
Thus, with CLIJ2 you can.

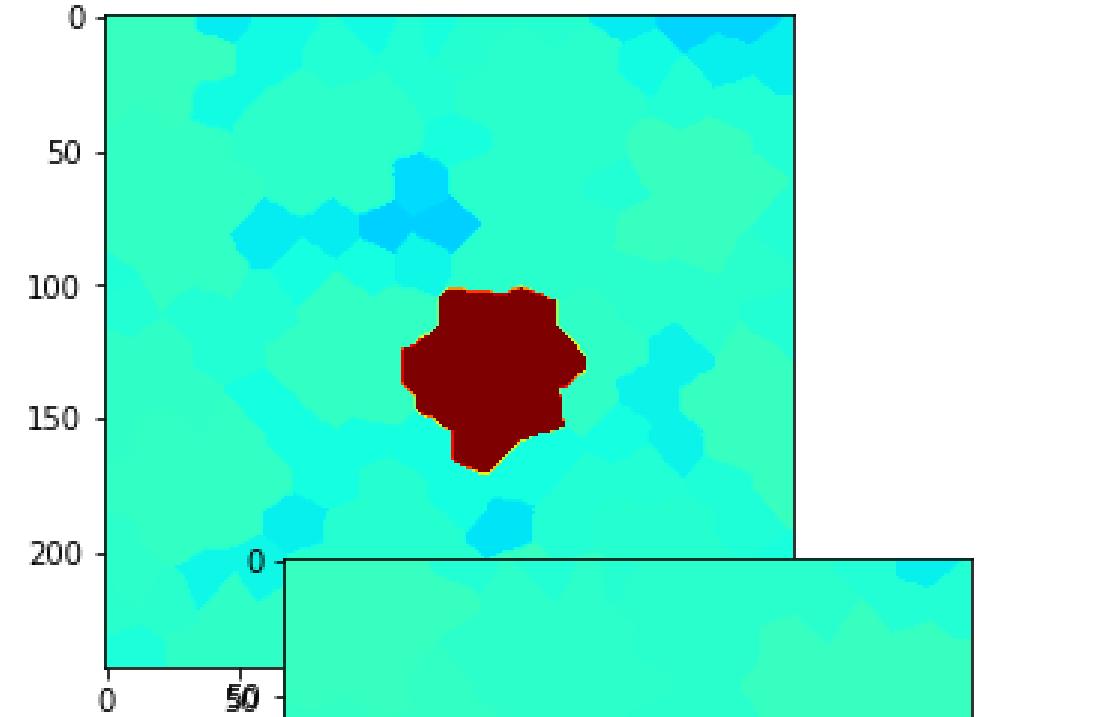
https://clij.github.io/clij2-docs/md/neighbors_of_neighbors/

Neighborhood definitions

“Cells”



Touching neighbors

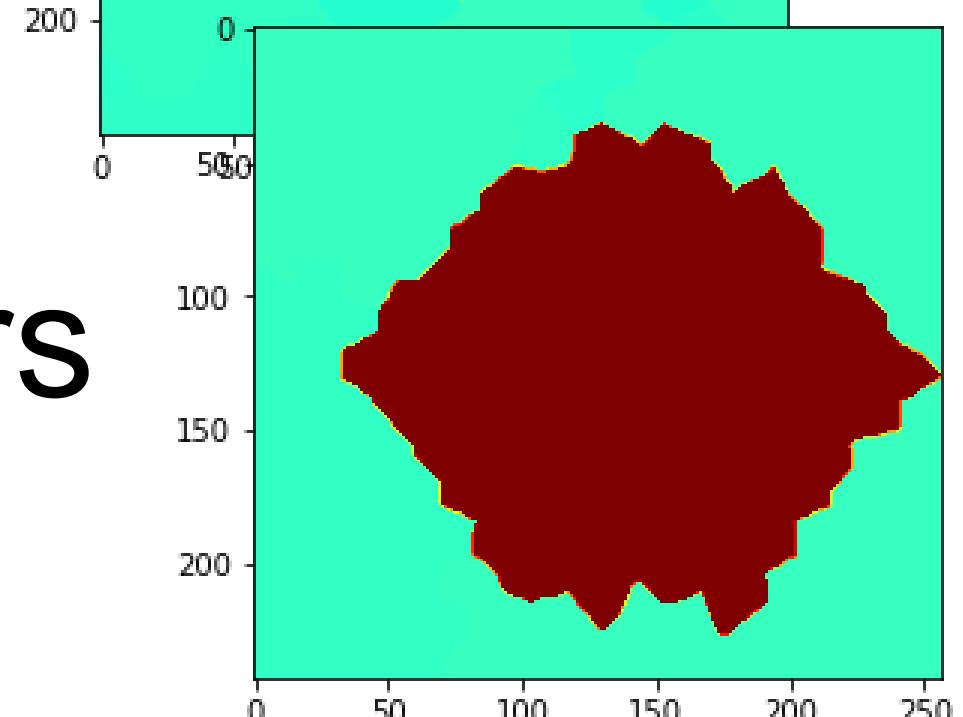


n=2

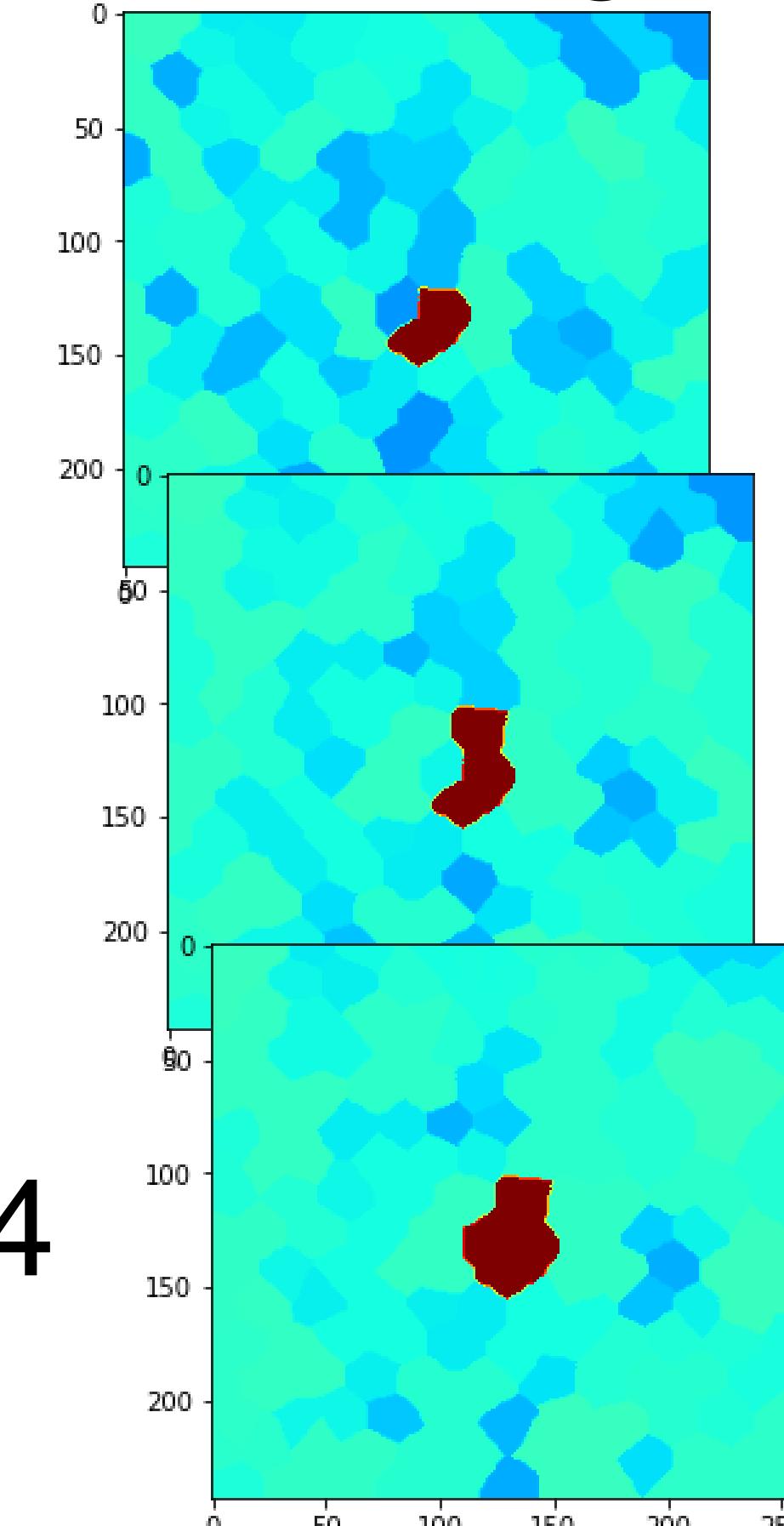
n=3

n=4

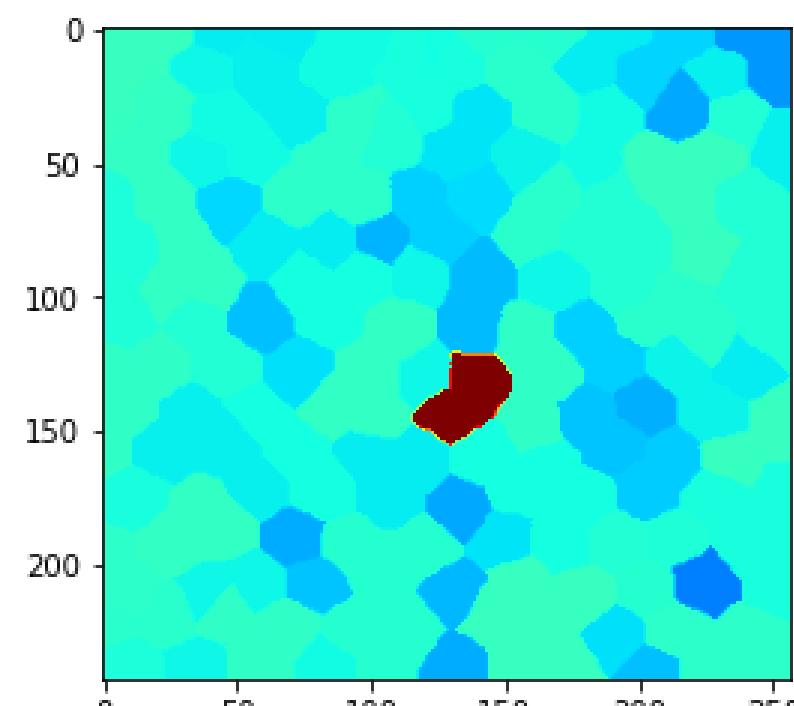
Neighbors of
neighbors
of neighbors



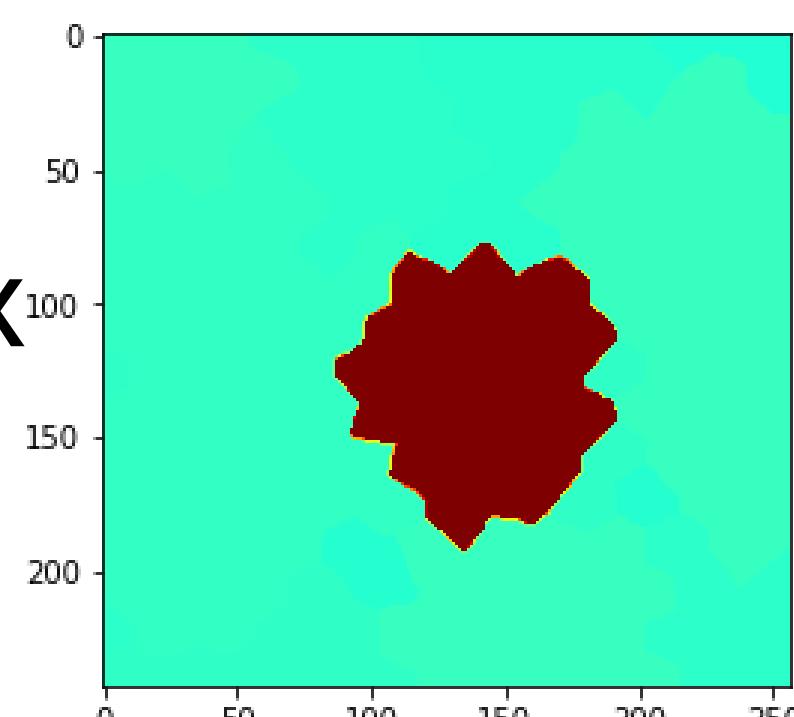
N nearest neighbors



d < 20 px



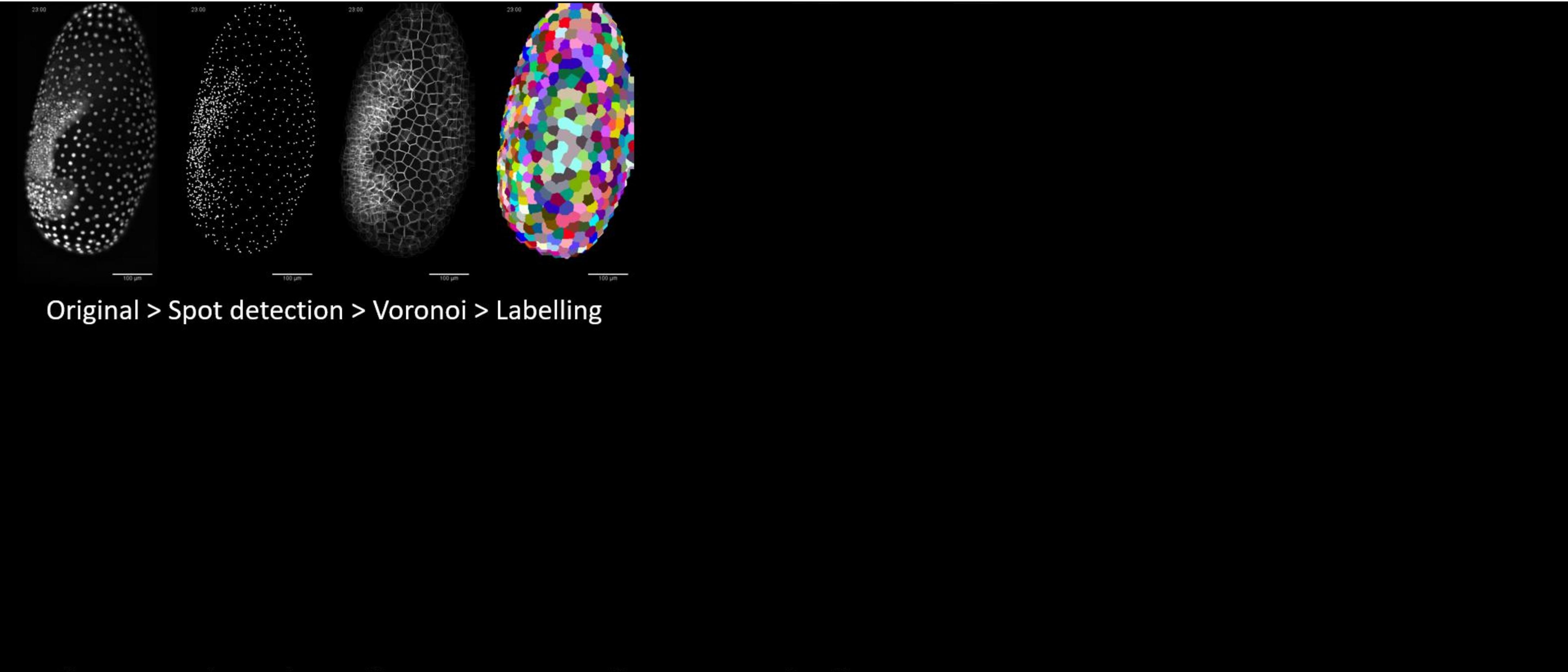
d < 120 px



Proximal
neighbors

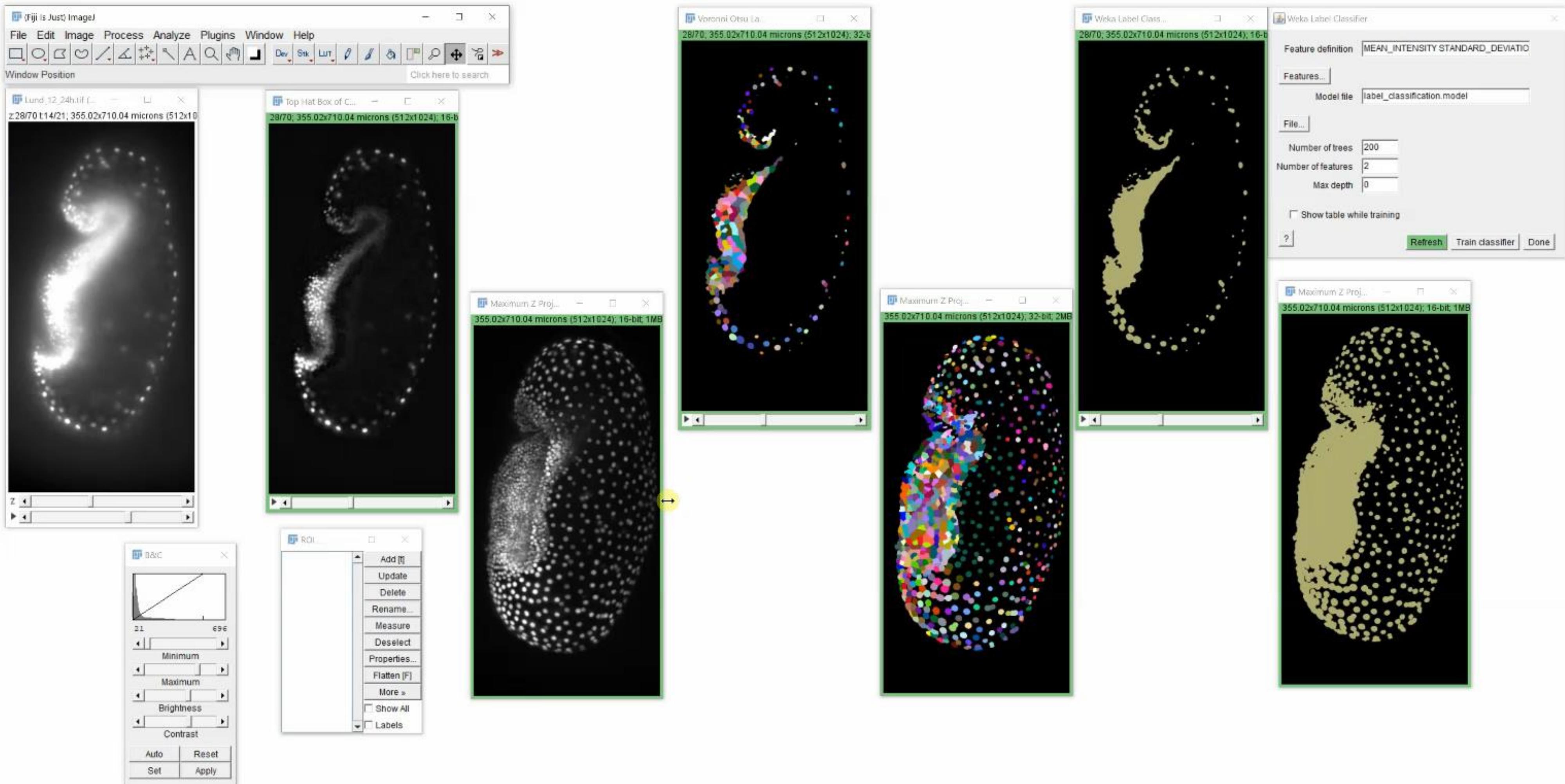
Operations on adjacency graphs

Convolution and machine learning for cell classification



Original > Spot detection > Voronoi > Labelling

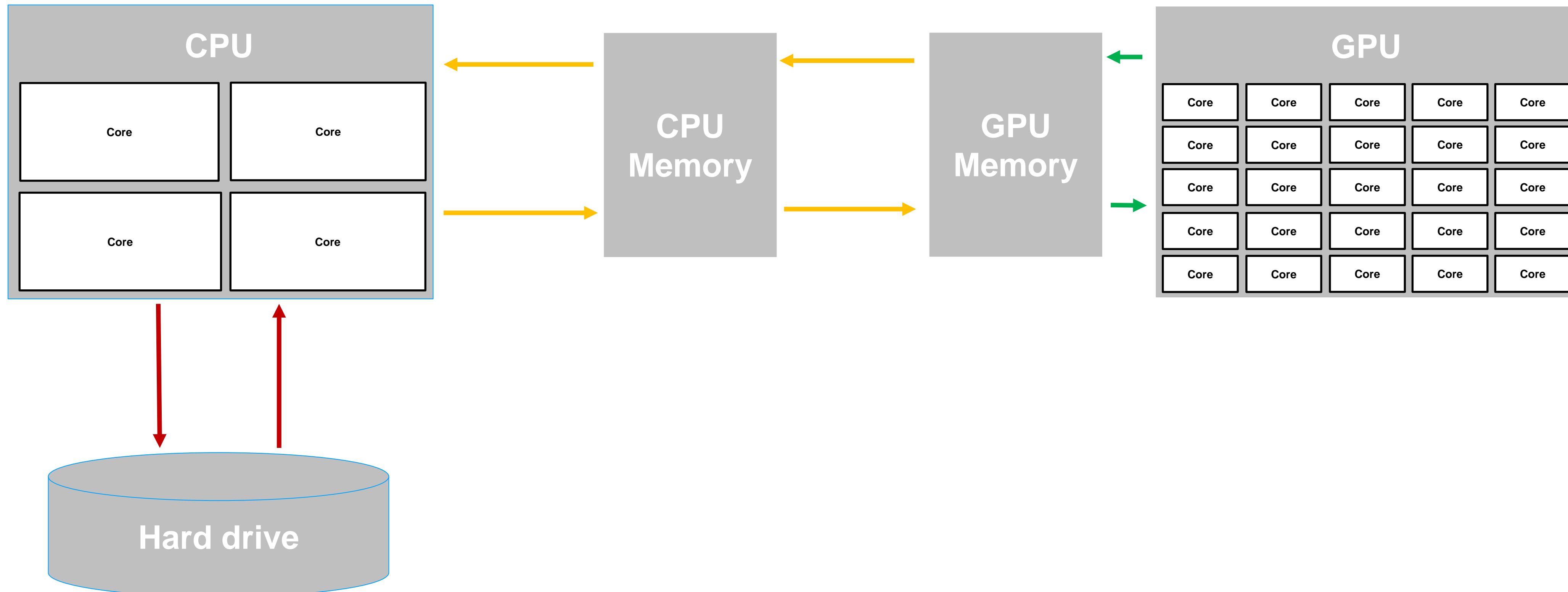
Machine learning for cell & tissue classification



Part IV: Benchmarking

GPUs allow real-time image processing

GPUs are specialised in processing, super fast thanks to many cores and fast memory access

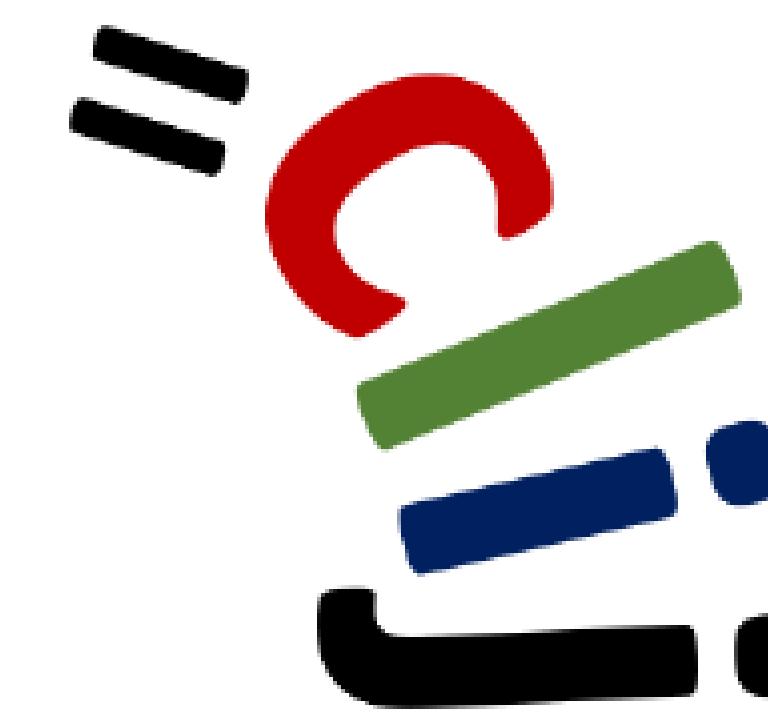


GPU-accelerated image processing

... depends on operation, image size, parameters, hardware,



vs.

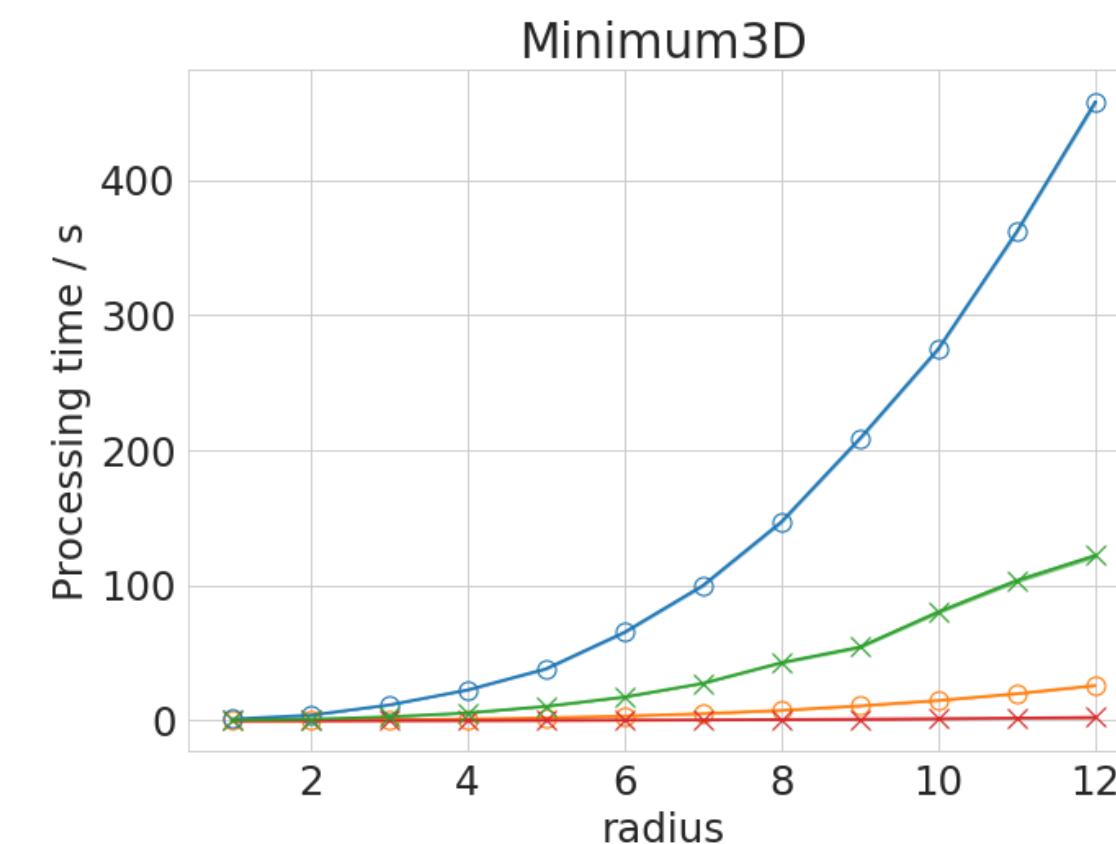
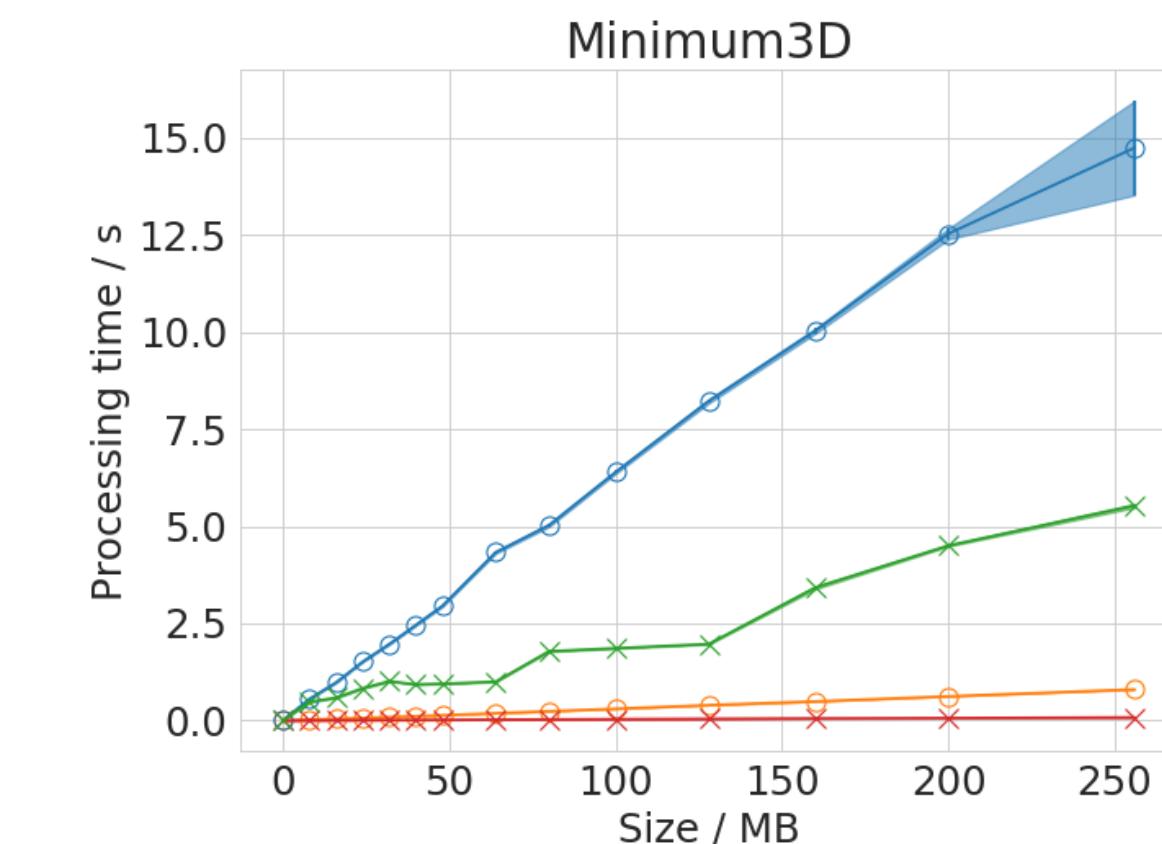
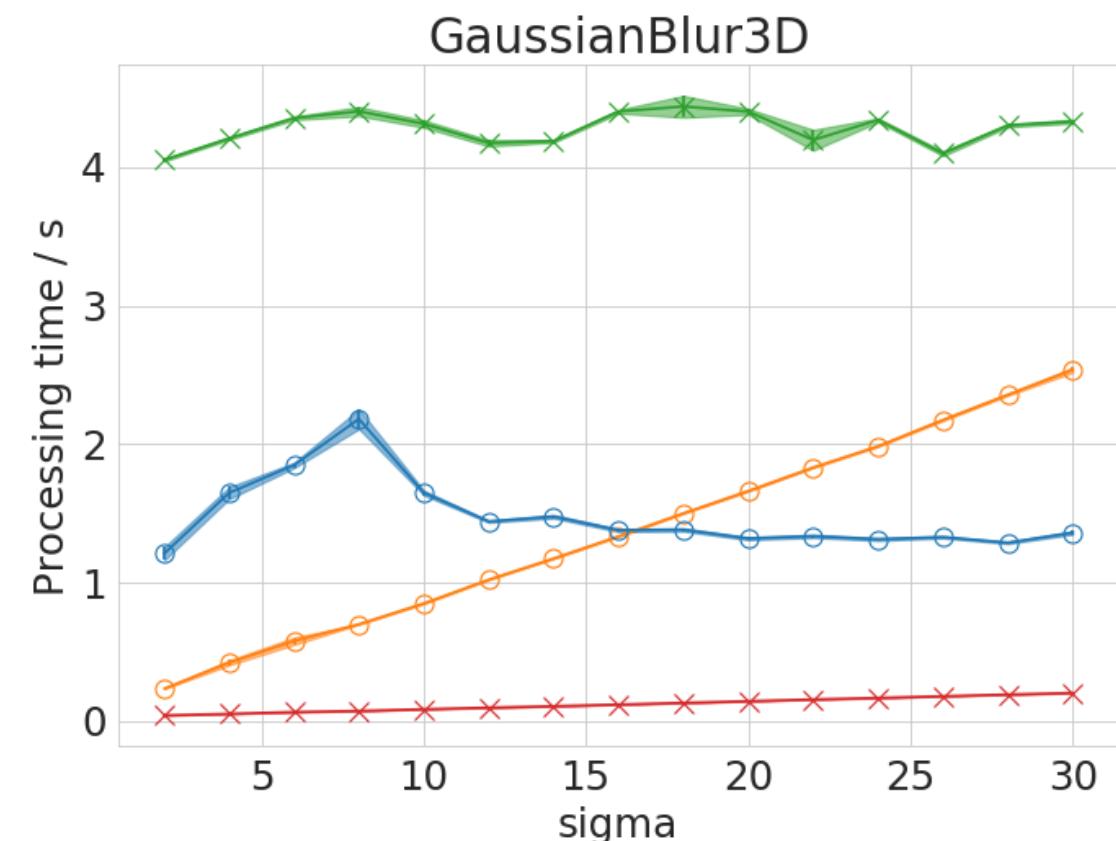
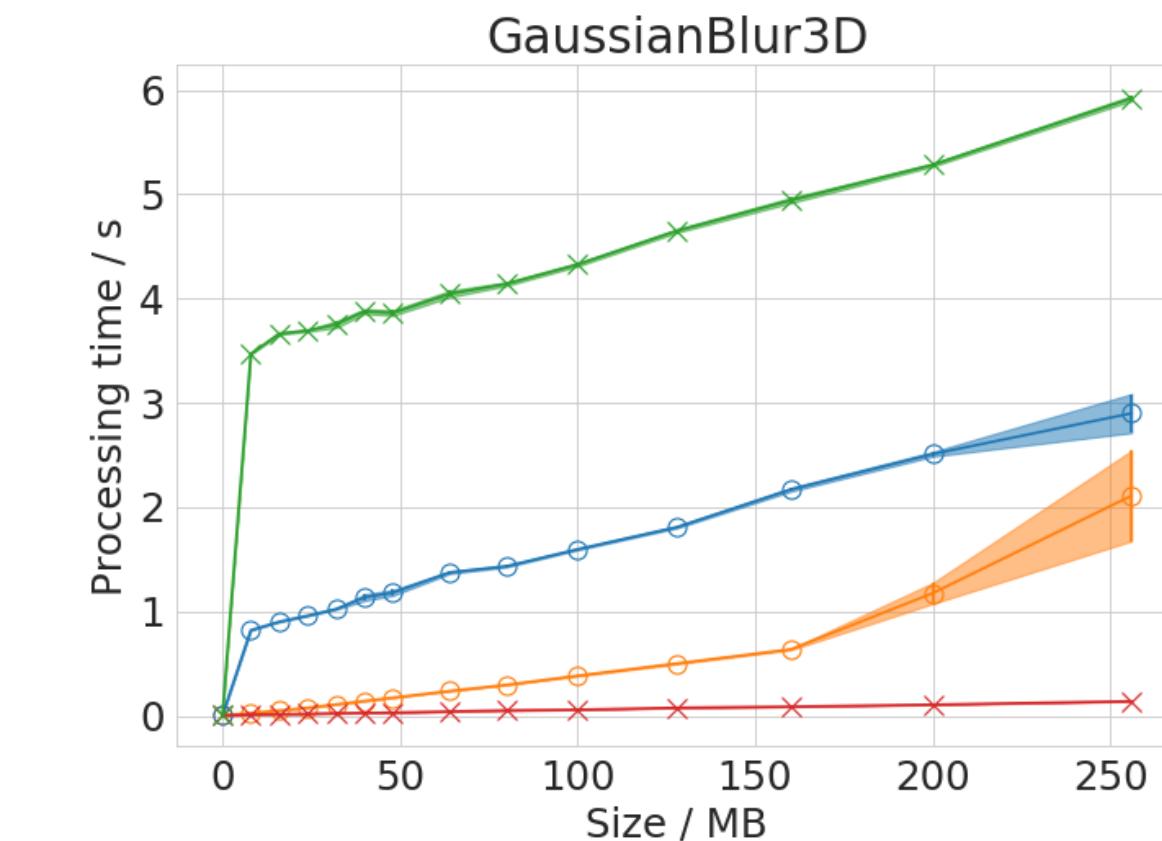


 Workstation CPU
2x Intel Xeon Silver 4110

 Laptop CPU
Intel Core i7-8650U

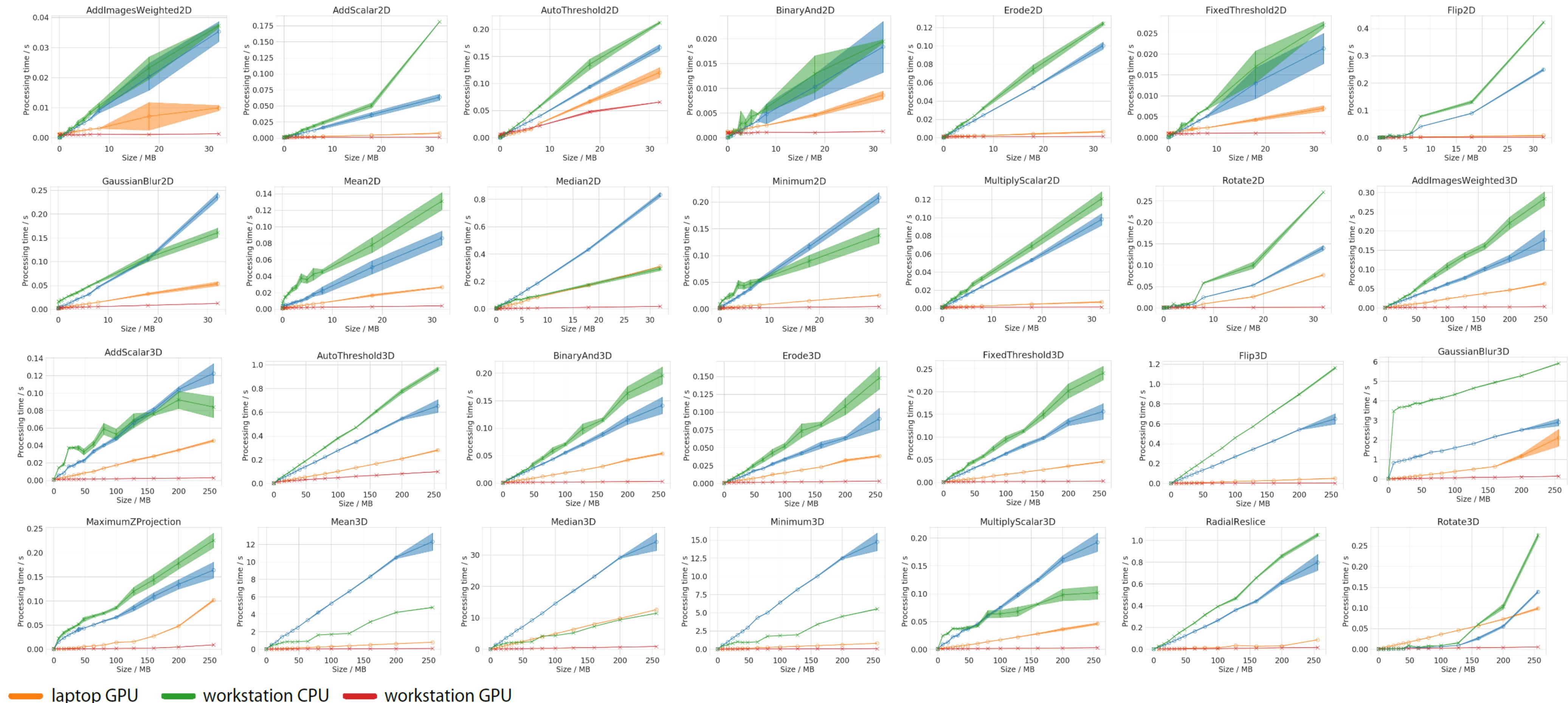
 Workstation GPU
Nvidia Quadro P6000

 Laptop GPU
Intel UHD 620 GPU

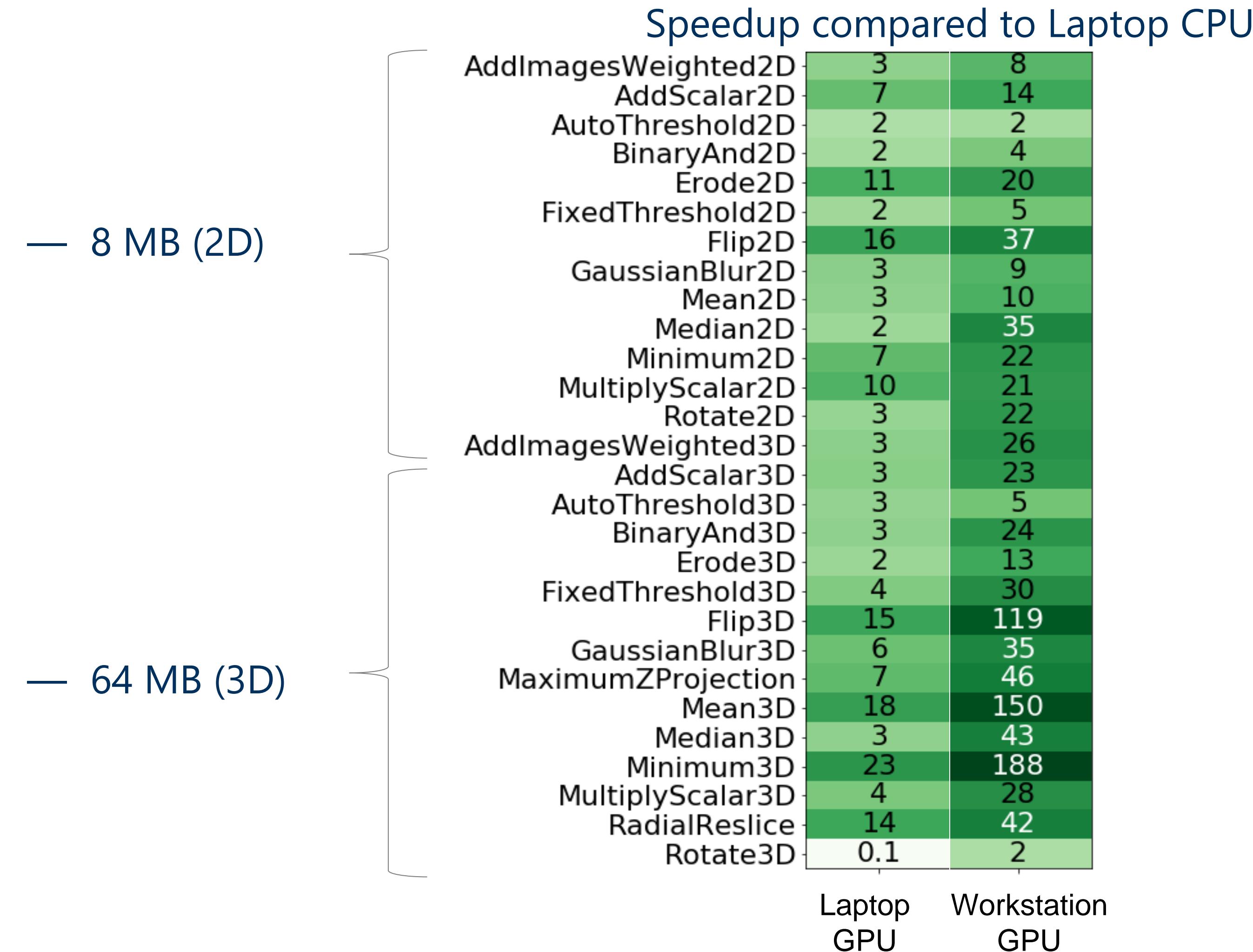


GPU-accelerated image processing

... depends on operation, image size, parameters, hardware,

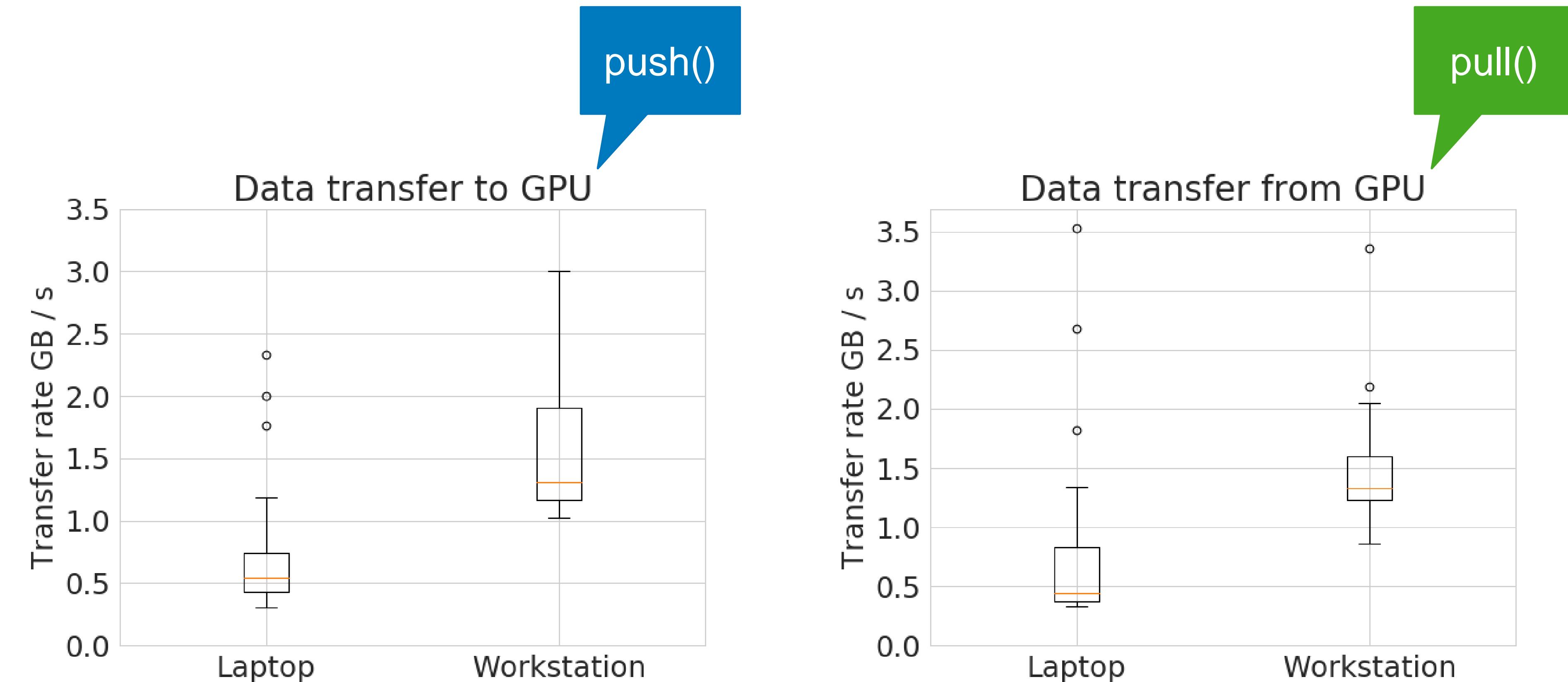


GPU-accelerated image processing



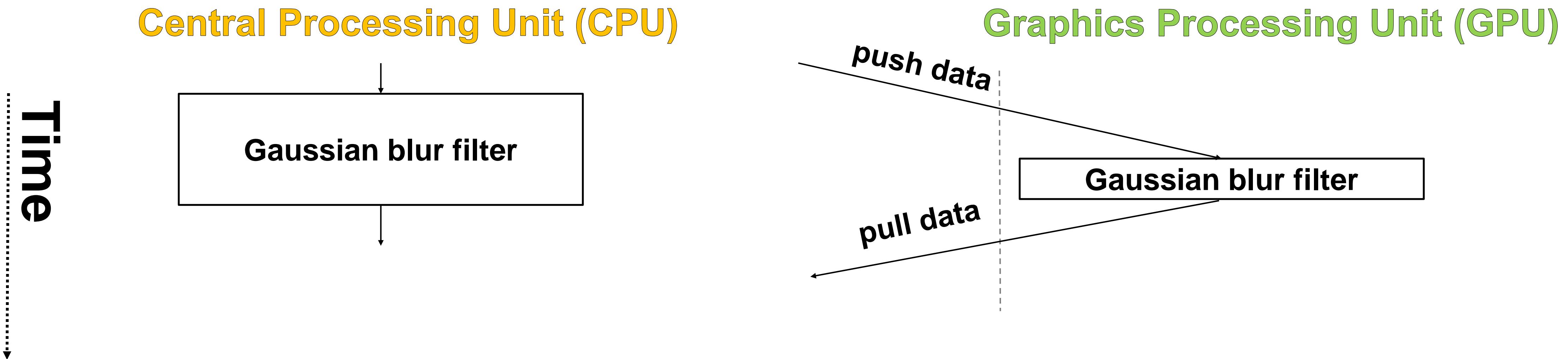
Data transfer takes time

BUT: Push() and Pull() take additional time!



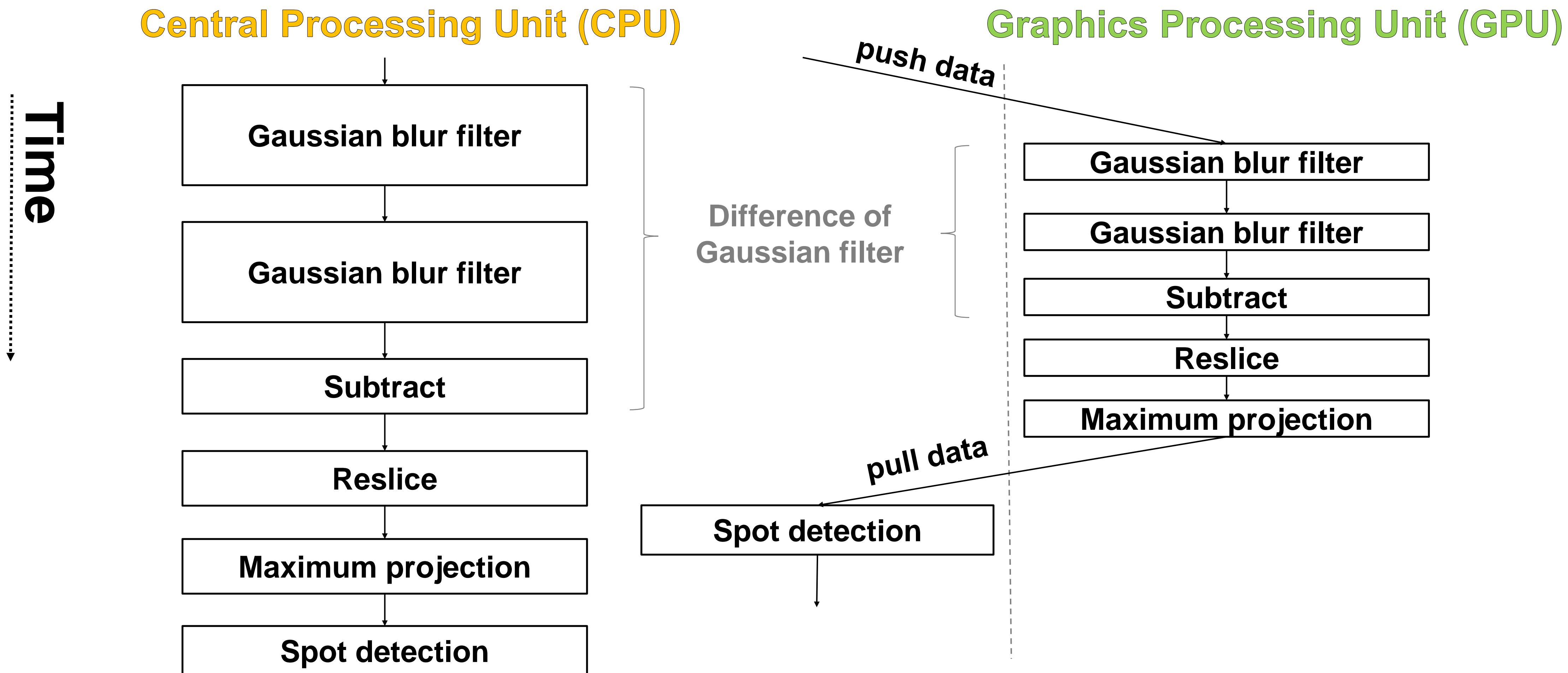
Build workflows consisting of many operations

GPU acceleration may suffer from data transfer between CPU and GPU



Build workflows consisting of many operations

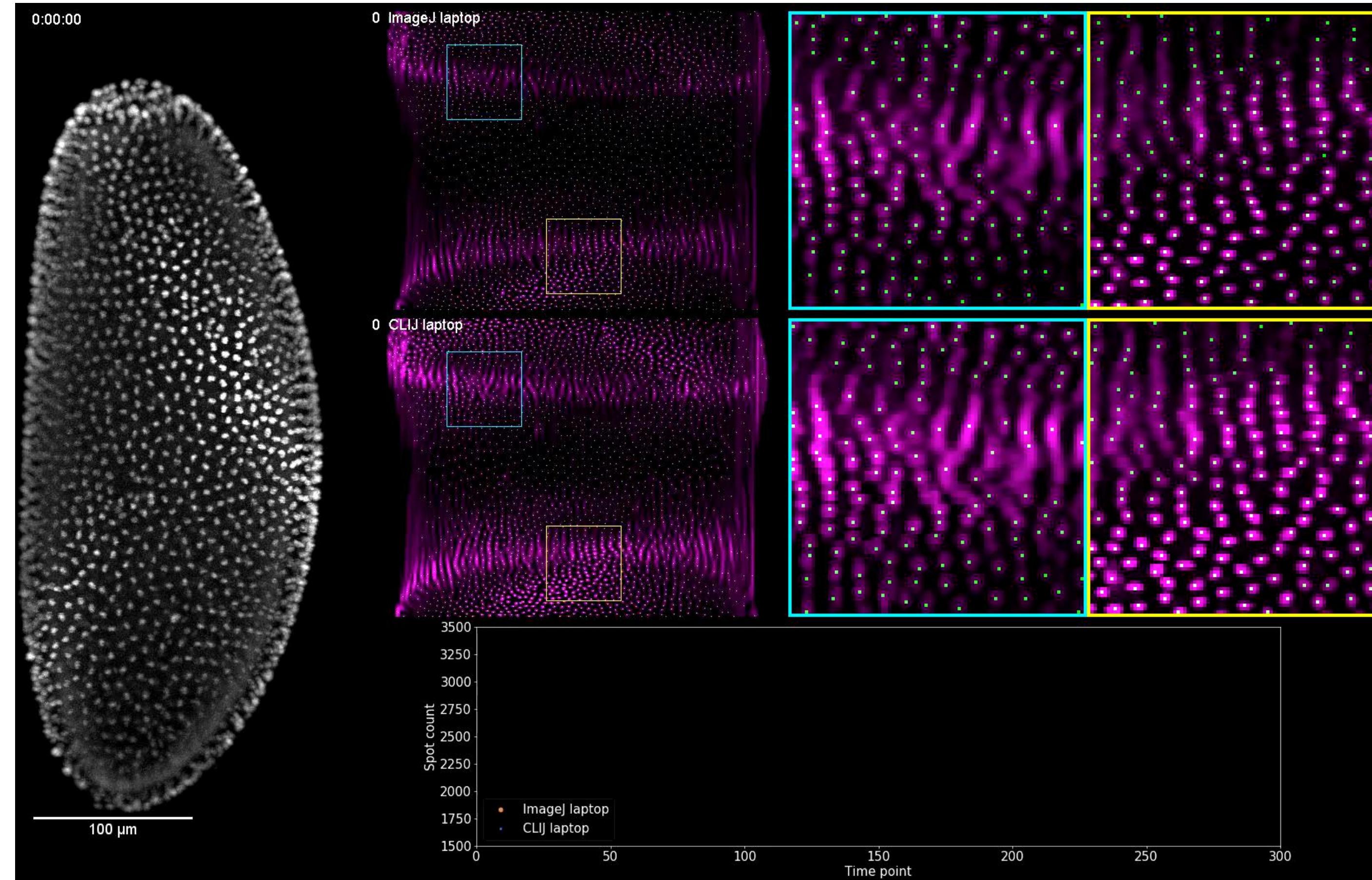
GPU acceleration may suffer from data transfer between CPU and GPU



Practical use-case: Cell counting

Counting spots in 300 frames of light sheet data (including I/O)

Drosophila melanogaster, histone-RFP



ImageJ on CPU (laptop)
33 seconds per frame

2:44 h (timelapse)



ImageJ using the GPU (laptop)
2.2 seconds per frame

11 min (timelapse)

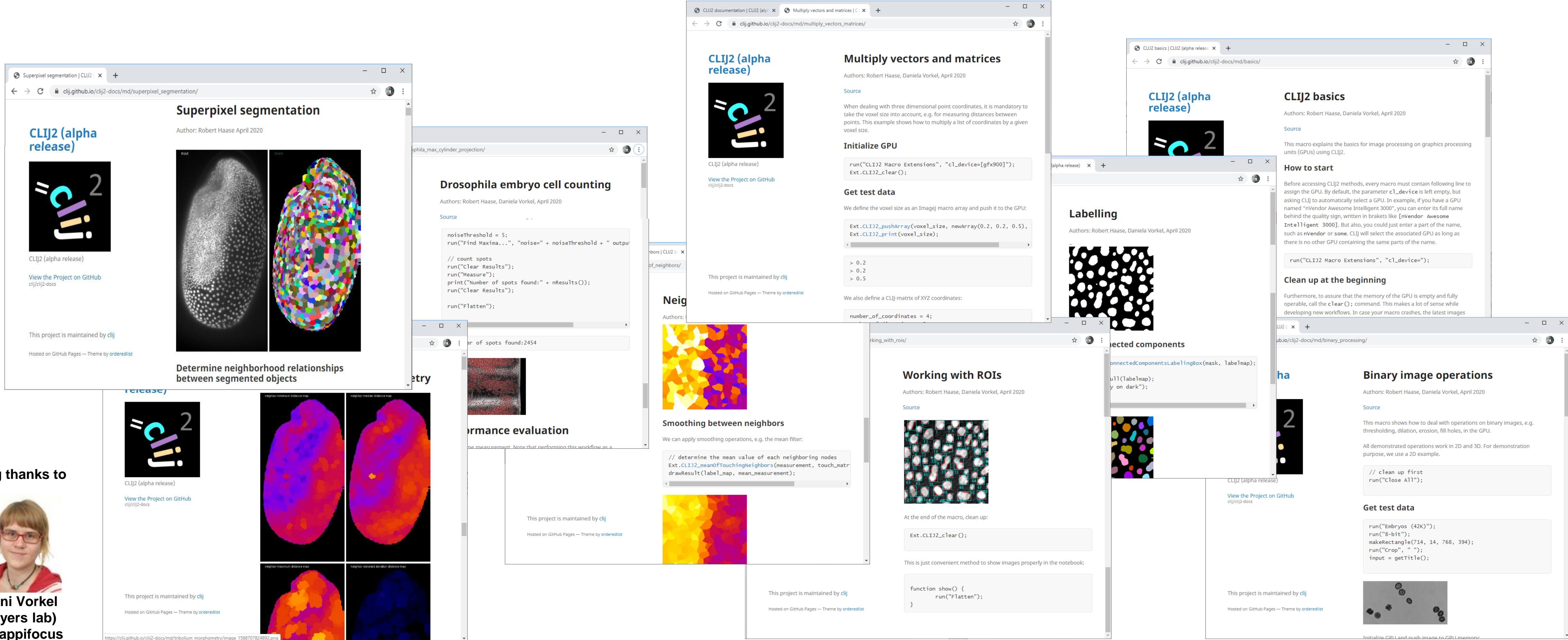


ImageJ using a dedicated
GPU (workstation)
1 second per frame

5 min (timelapse)

Part V: Accessibility

Online documentation: tutorials



The screenshot displays a grid of browser windows and tabs, each showing a different CLIJ2 tutorial or example. The titles include:

- Superpixel segmentation
- Drosophila embryo cell counting
- Multiply vectors and matrices
- Labelling
- Working with ROIs
- Binary image operations
- Big thanks to Dani Vorkel (Myers lab) @happifocus
- Performance evaluation
- Smoothing between neighbors
- Connected components
- Get test data

Each window contains a brief description, source code snippets, and visual results. The overall theme is image processing and analysis using GPU-accelerated Java macros.

Online documentation: Cheat sheets

Cheat sheets show the most important methods with input and output parameters visually.

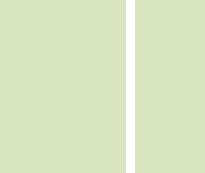
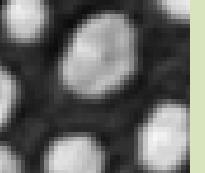
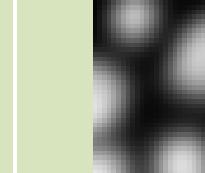
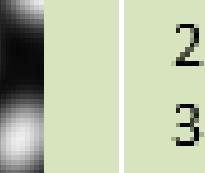


CLIJ2 cheat sheet: ImageJ macro II
GPU-accelerated image processing in Fiji

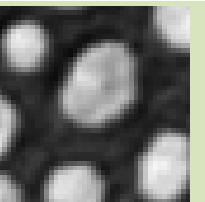
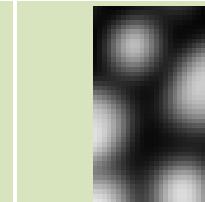
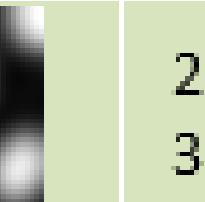
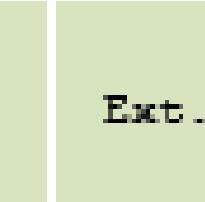
Basics / Wrangling

Operation	Parameters	Result	Dim	Example
Initialize CLIJ	I, HD, GFX or CPU			Ext.CLIJ2_initialize(I, HD, GFX or CPU);
Push			2D / 3D	Ext.CLIJ2_push(result, pointlist);
Pull			2D / 3D	Ext.CLIJ2_pull(result, pointlist);
Create	1024, 1024, 8		2D / 3D	Ext.CLIJ2_create(result, width, height, depth);
Convert			2D / 3D	Ext.CLIJ2_convert(result, type);
Copy			2D / 3D	Ext.CLIJ2_copy(result, source);
Copy slice	50, 50		2D / 3D	Ext.CLIJ2_copySlice(result, source, x, y, width, height);
Crop	20, 20		2D / 3D	Ext.CLIJ2_crop(result, x, y, width, height);
Paste	9, 9		2D / 3D	Ext.CLIJ2_paste(result, source, x, y, width, height);
Release			2D / 3D	Ext.CLIJ2_release(result);
Clear				Ext.CLIJ2_clear();

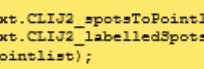
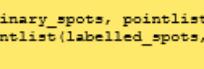
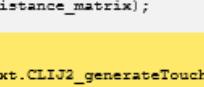
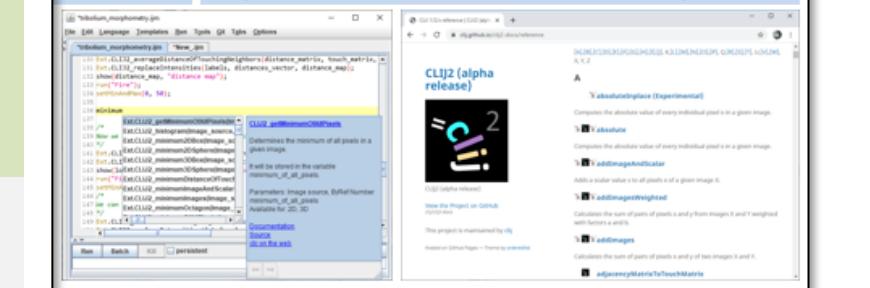
Spatial Transforms

Operation	Parameters	Result	Dim	Example
Rotate by 90 degrees			2D / 3D	Ext.CLIJ2_rotate(result, angle);
Rotate	, 45, true		2D / 3D	Ext.CLIJ2_rotate(result, angle, flip);
Flip	, true, false		2D / 3D	Ext.CLIJ2_flip(result, flip);
Translate	, 20, 20		2D / 3D	Ext.CLIJ2_translate(result, x, y);
Affine transform	"center rotate=45 -center"		2D / 3D	Ext.CLIJ2_affine(result, center, rotate, scale, shear, translate);
Deform / warp			2D / 3D	Ext.CLIJ2_deformWarp(result, source, parameters);
Projections			3D	Ext.CLIJ2_project(result, source, axis, angle, distance);

Filters

Operation	Parameters	Result	Dim	Examples
Gaussian blur	, 10, 10		2D / 3D	Ext.CLIJ2_gaussianBlur2D(result, sigmaX, sigmaY);
Difference of Gaussian	, 2, 2, 20, 20		2D / 3D	Ext.CLIJ2_differenceOfGaussian2D(result, sigma1x, sigma1y, sigma2x, sigma2y);
Invert			2D / 3D	Ext.CLIJ2_invert(result);
Laplace			2D / 3D	Ext.CLIJ2_laplaceBox(result);
Mean	, 5, 5		2D / 3D	Ext.CLIJ2_mean2DBox(result, radiusX, radiusY);

Result | Dim | Examples

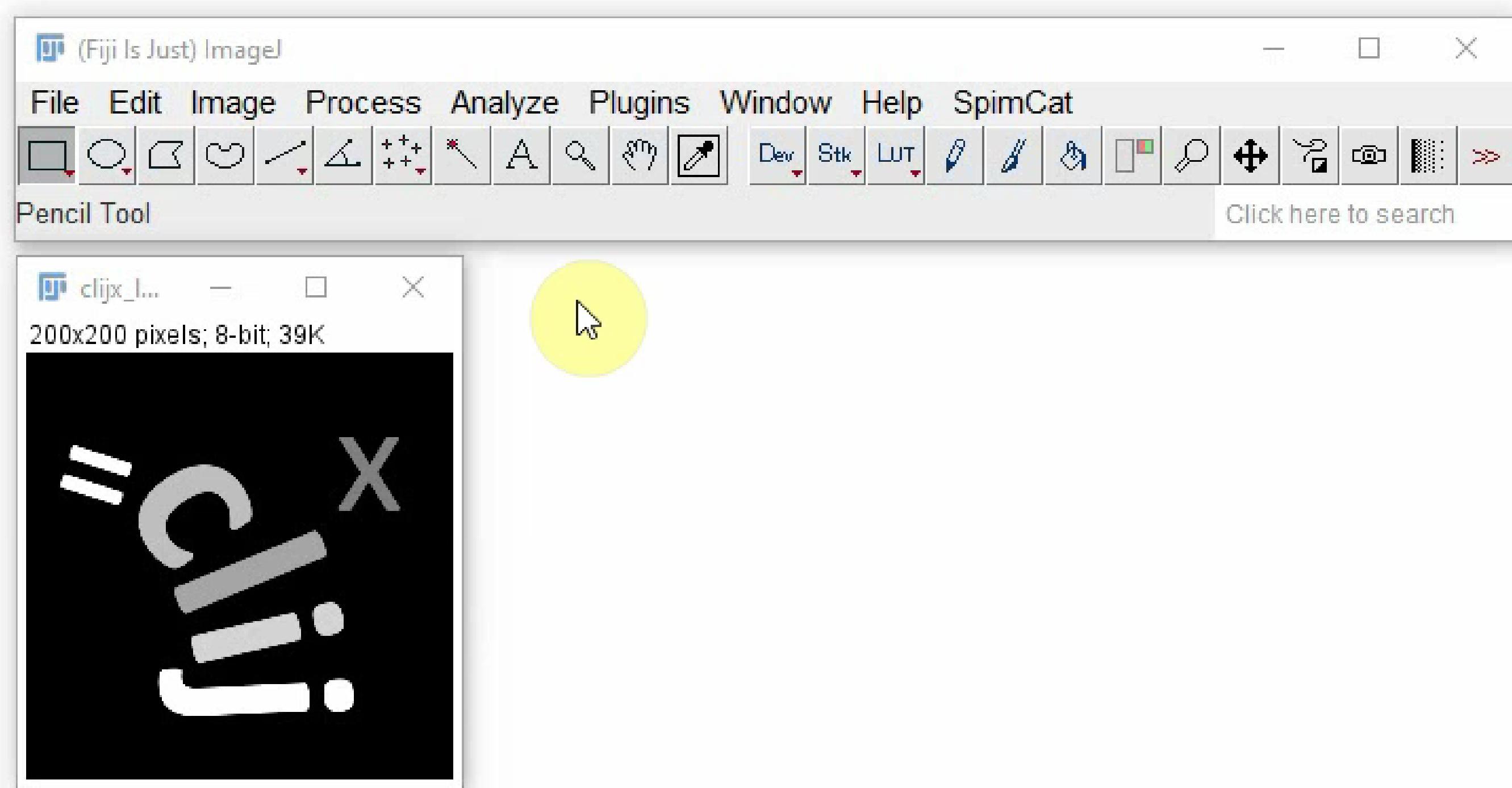
Result	Dim	Examples
	2D / 3D	Ext.CLIJ2_spotsToPointlist(binary_spots, pointlist); Ext.CLIJ2_labelledSpotsToPointlist(labelled_spots, pointlist);
	2D / 3D	Ext.CLIJ2_generateDistanceMatrix(pointlist1, pointlist2, distance_matrix);
	2D / 3D	Ext.CLIJ2_generateTouchMatrix(label_map, touch_matrix);
	2D / 3D	Ext.CLIJ2_touchMatrixToMesh(pointlist, touch_matrix, mesh);
	2D / 3D	Ext.CLIJ2_distanceMatrixToMesh(pointlist, distance_matrix, mesh, max_distance);
	2D / 3D	Ext.CLIJ2_meanOfTouchingNeighbors(values, touch_matrix, mean_values);
	2D / 3D	Ext.CLIJ2_countTouchingNeighbors(touch_matrix, count_vector);
	2D / 3D	Ext.CLIJ2_statisticsOfBackgroundAndLabelledPixels(image, labelmap); Ext.CLIJ2_statisticsOfLabelledPixels(input, labelmap);
	2D / 3D	Ext.CLIJ2_pushResultsTable(image_name);
	2D / 3D	Ext.CLIJ2_pushResultsTableColumn(image_name, column_name);
	2D / 3D	Ext.CLIJ2_pullToResultsTable(image_name);
	2D / 3D	Ext.CLIJ2_pushArray(image_name, array, width, height, depth);
Installation instructions		
On GPU (CLU2)		• Install CLU2 by activating the "clij" and "clij2" update sites in Fiji. • Commands listed as "CLU2" are experimental should be handled with care. They may change or disappear at any point. To build reliable, reproducible workflows use CLU or CLIJ2 commands only.
		

<https://clij.github.io/clij2-docs>  @haesleinhuepf

docs  @haesleinhuepf #clij cheat sheet 2020-04-20

Video tutorials

For example: The Fiji plugin generator



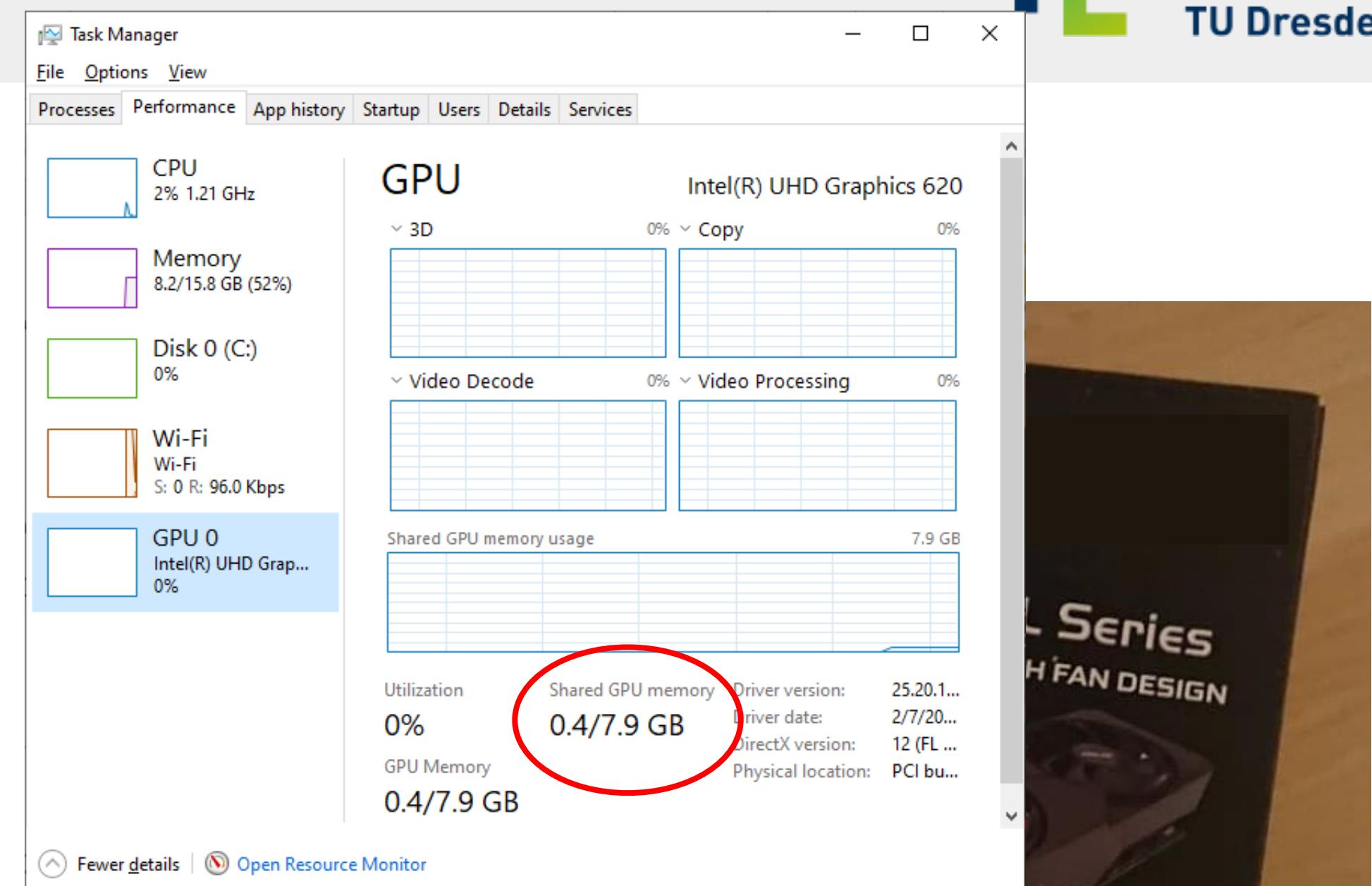
Checklist

When does GPU-accelerated image processing make sense?

In order to accelerate your image analysis workflow

- you need an image analysis workflow in the first place,
- it should be slow; ideally:
 $(\text{image processing time} / \text{image loading time}) > 10 \text{ to } 100$,
- a single processing step (rule of thumb: image size in GB x4) should **fit in your graphics card memory**,
- you need to re-write your workflow at least partly and
- you should have experience with ImageJ macro.

If you really want to get the most out of it, you should own a graphics card with **GDDR6** memory (memory bandwidth $> 400 \text{ Gb/s}$).



Last but not least: citability

- If you work with CLIJ and friends, please cite the paper(s). It'll be hard to apply for grants otherwise.

nature > nature methods > correspondence > article

nature methods

Cornell University

Correspondence | Published: 18 November 2020

CLIJ: GPU-accelerated ImageJ Macro imaging for everyone

Robert Haase , Loic A. Royer , Peter Stein , Dibrov, Uwe Schmidt, Martin Weigert, Nicolai Eugen W. Myers

Nature Methods 17, 5–6(2020) | Cite this article

arXiv.org > cs > arXiv:2008.11799

Computer Science > Mathematical Software

[Submitted on 26 Aug 2020]

GPU-accelerating ImageJ Macro imaging for everyone

Daniela Vorkel, Robert Haase

This chapter introduces GPU-accelerated image processing in ImageJ. Core concepts such as variables, for-loops, and functions are explained. We present in a step-by-step tutorial how to translate workflows. We present in a step-by-step tutorial how to translate workflows. We present in a step-by-step tutorial how to translate workflows.

Subjects: Mathematical Software (cs.MS); Distributed, Parallel, and Cluster Computing (cs.DC)

Cite as: arXiv:2008.11799 [cs.MS] (or arXiv:2008.11799v1 [cs.MS] for this version)

Submission history

From: Robert Haase [view email]
 [v1] Wed, 26 Aug 2020 20:38:31 UTC (2,079 KB)

Cold Spring Harbor Laboratory

bioRxiv
 THE PREPRINT SERVER FOR BIOLOGY

New Results

Comments (1)

Interactive design of GPU-accelerated Image Data Flow Graphs and cross-platform deployment using multi-lingual code generation

Robert Haase, Akanksha Jain, Stéphane Rigaud, Daniela Vorkel, Pradeep Rajashekhar, Theresa Suckert, Talley J. Lambert, Juan Nunez-Iglesias, Daniel P. Poole, Pavel Tomancak, Eugene W. Myers

doi: <https://doi.org/10.1101/2020.11.19.386565>

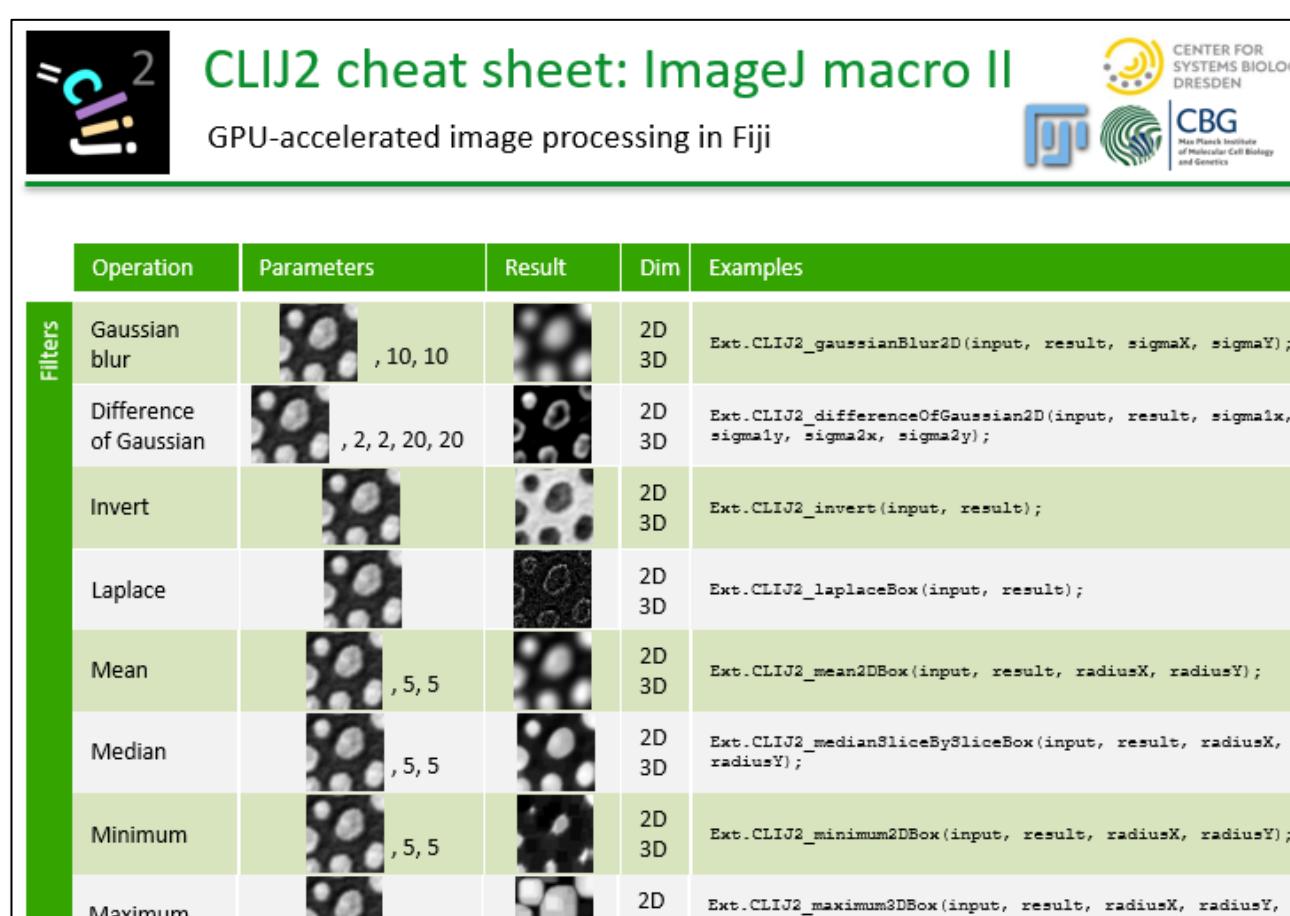
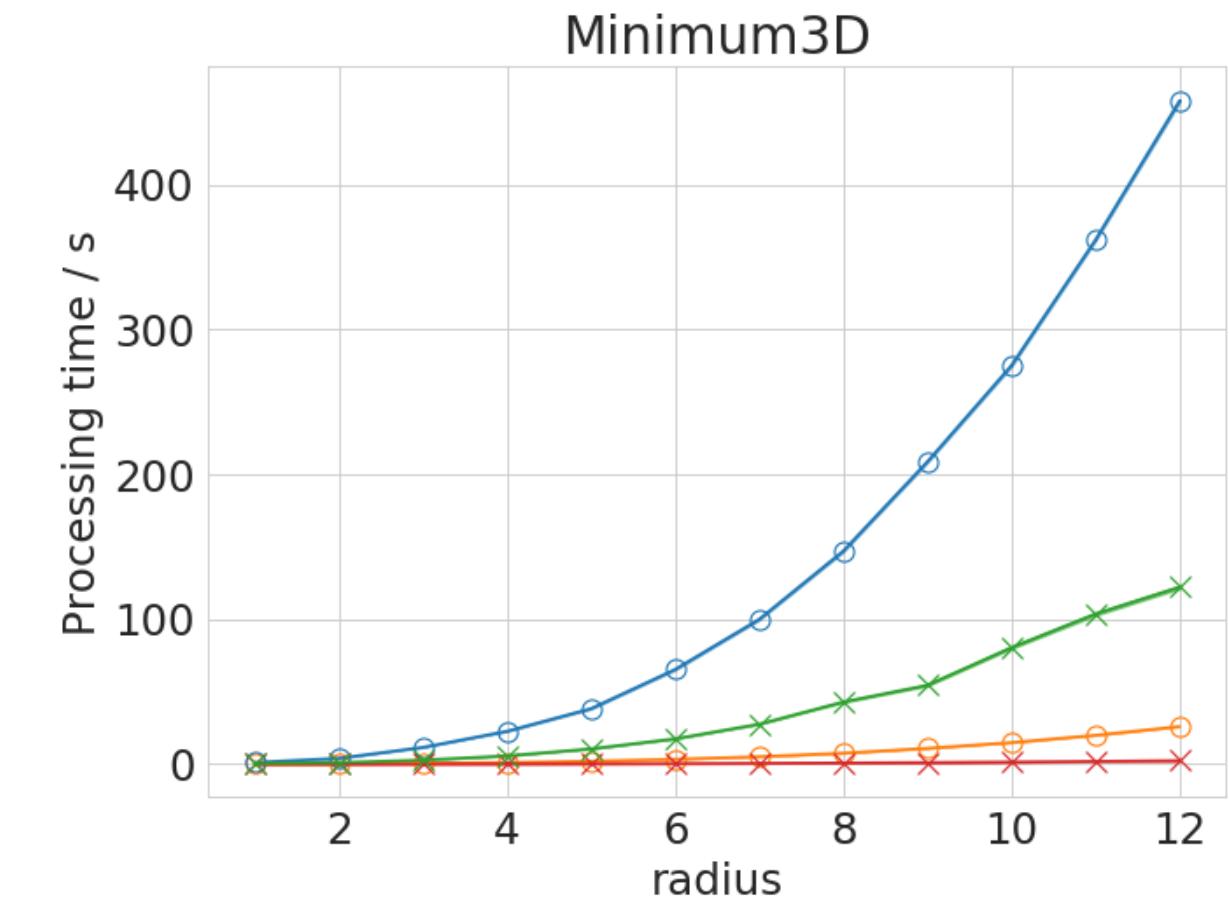
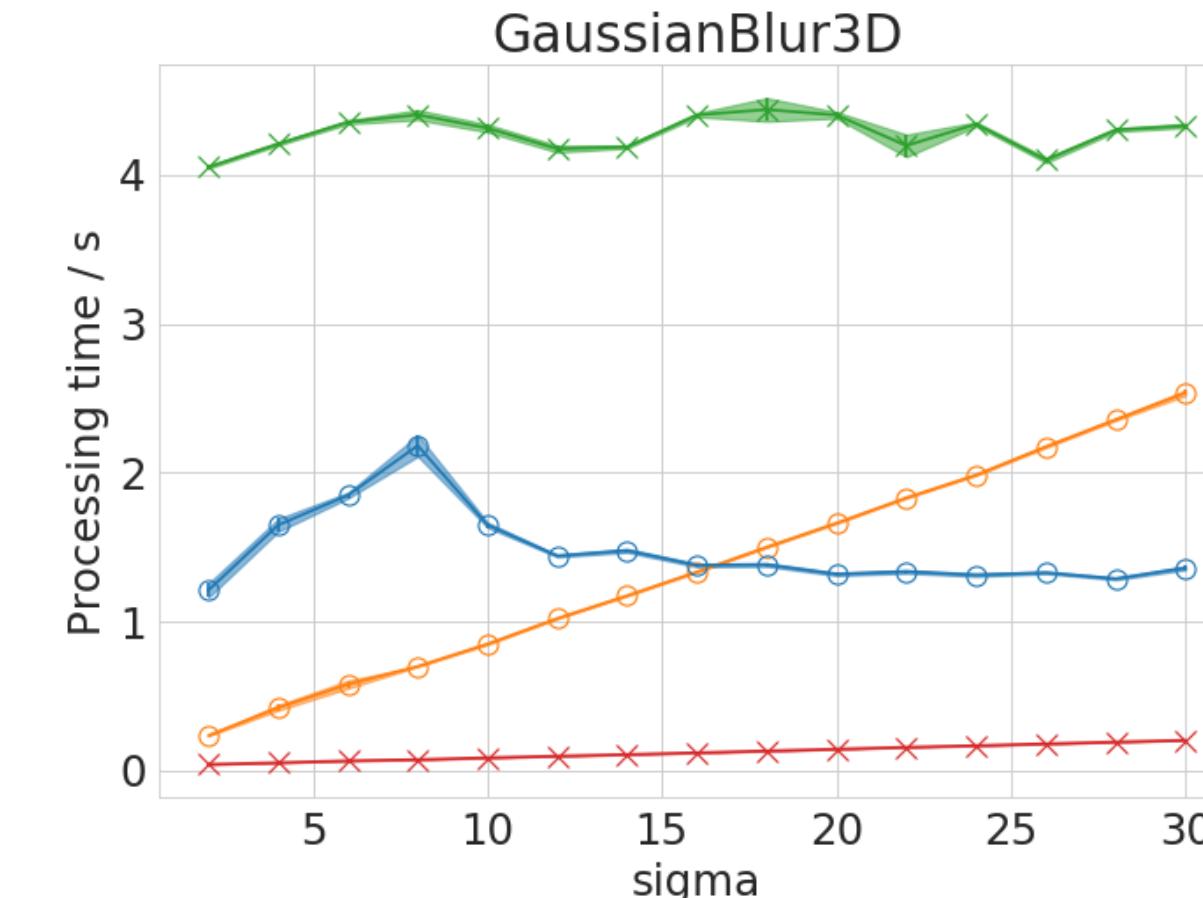
This article is a preprint and has not been certified by peer review [what does this mean?].

Abstract Full Text Info/History Metrics Preview PDF

Summary

- GPU-accelerated image processing works best with
 - workflows with many steps between push and pull
 - processing many images,
 - on dedicated GPUs.

- Human users like most
- documentation by example,
- integrated documentation and
- interactive user interfaces.



The screenshot shows the ImageJ Morphology Editor interface. The title bar says "tribolium_morphometry.ijm". The menu bar includes File, Edit, Language, Templates, Run, Tools, Git, Tabs, and Options. A tabs bar at the top has two tabs: "*tribolium_morphometry.ijm" and "*New_.ijm". The main area contains a script:

```
121 run("Green Fire Blue");
122 setMinAndMax(0, 50);
123 /*
124 ## Quantitative analysis of distances between neighbors
125 We next determine the average distance between a node and all of its neighbors
126 a vector with as many entries as nodes in the graph. We use this vector to
127 label map of the cell segmentation. This means, we replace label 1 with the
128 node 1 and label 2 with the average distance to node 2.
129 */
130 Ext.CLIJ2_averageDistanceOfTouchingNeighbors(distance_matrix, touch_matrix);
131 Ext.CLIJ2_replaceIntensities(labels, distances_vector, distance_map);
132 show(distance_map, "distance map");
133 run("Fire");
134 setMinAndMax(0, 50);
135
136 aver|
```

A tooltip for the "aver" command is displayed, showing the function signature and a brief description:

CLIJ2_averageDistanceOfNClosestPoints

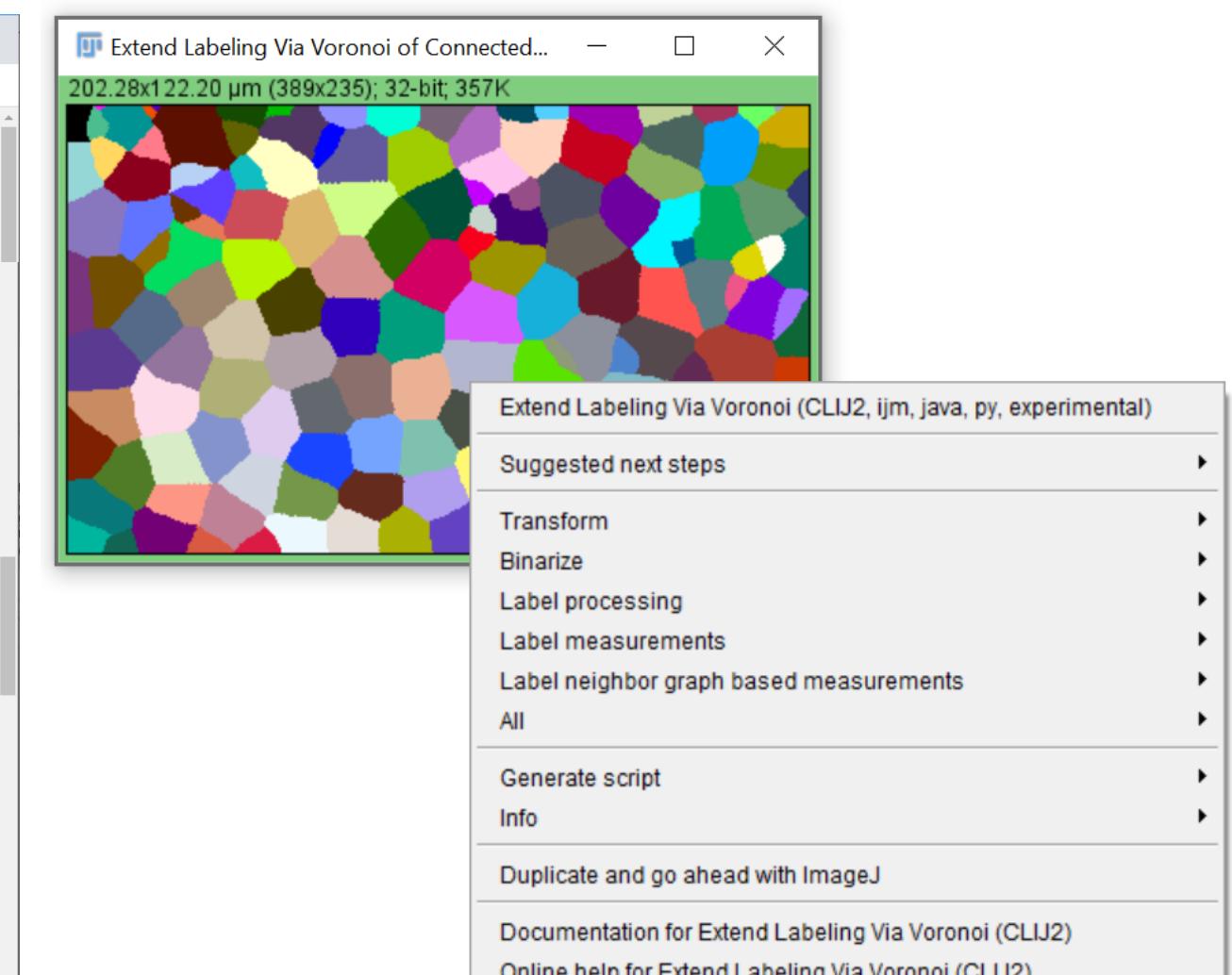
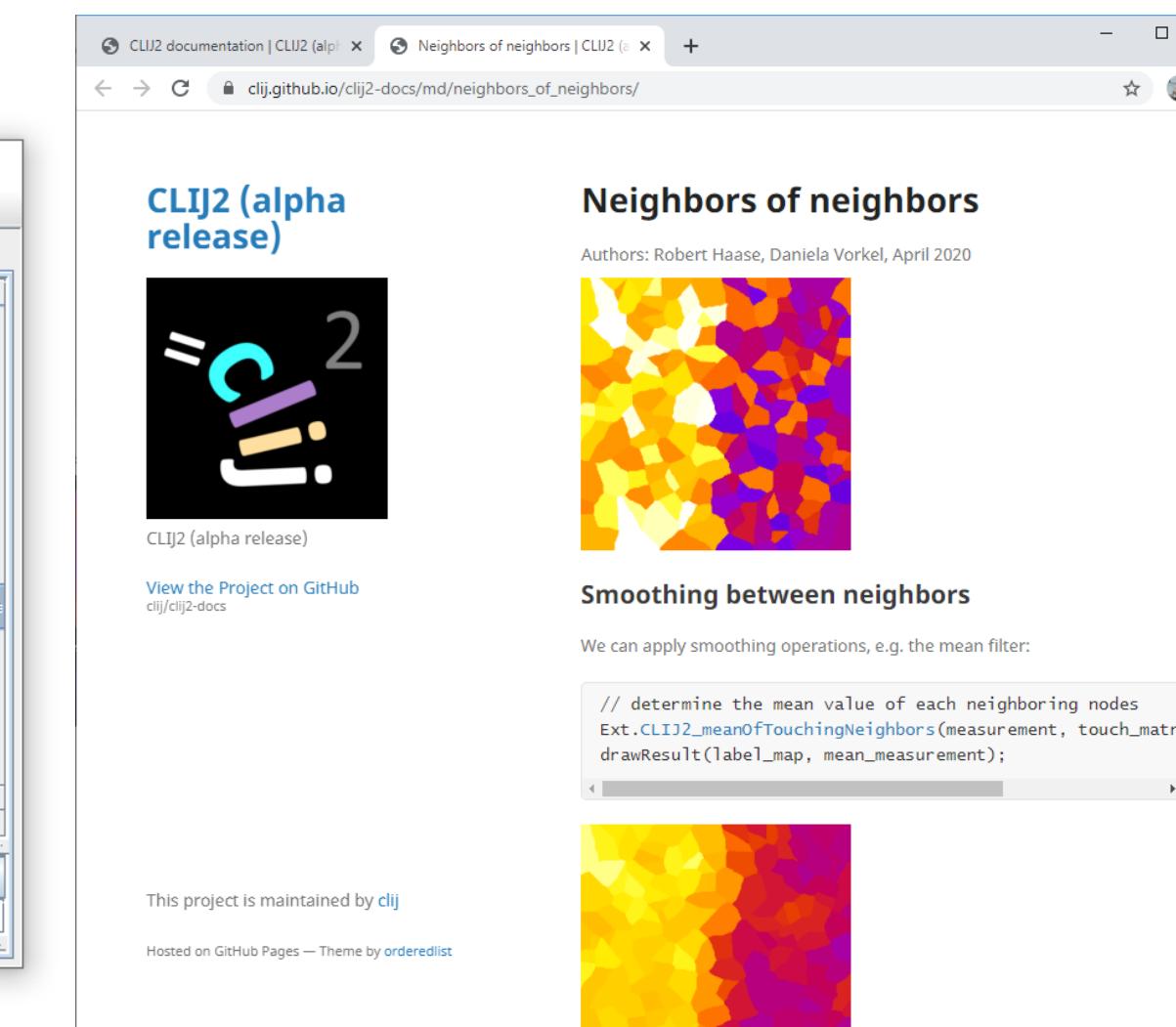
Determines the average of the n closest points for every point in a distance matrix.

This corresponds to the average of the n minimum values (rows) for each column of the distance matrix.

Parameters: Image distance_matrix, ByRef Image indexlist_destination, Number nClosestPointsTofind

Available for: 2D

At the bottom left, there is a toolbar with a "Run" button highlighted in blue.



Acknowledgements



Akanksha Jain
(Treutlein lab)
@jain_akanksha_



Dani Vorkel
(Myers lab)
@happifocus



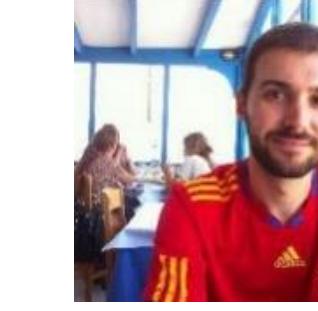
Theresa Suckert
(Krause lab,
Oncoray)



Pradeep Rajasekhar
(Poole lab, Monash U)



Talley J. Lambert
(HMS)
@pr4deepr



Juan Nunez-Iglesias
(Monash)
@jnuneziglesias



Pavel Tomancak
(CSBD / MPI-CBG)
@PavelTomancak



Gene Myers
(CSBD / MPI-CBG)
@TheGeneMyers



Stéphane Rigaud
(Institut Pasteur)
@StRigaud



Jean-Yves Tinevez
(Institut Pasteur)
@jytinevez



Florian Jug
(CSBD / MPI-CBG)
@florianjug



Szabolcs Horvát
(Modes lab)
@szhorvat



Johannes Girstmair
(Tomancak lab)
@jogirstmair



Brian Northon
@truenorth_ia



Matthias Arzt
(Myers/Jug lab)

Alex Herbert (University of Sussex)

Alexandr Dibrov (MPI CBG)

Bertrand Vernay (IGBMC, Strasbourg)

Bert Nitzsche (PoL TU Dresden)

Bradley Lowekamp (NIAID Washington)

Bram van den Broek (Netherlands

Cancer Institute)

Brenton Cavanagh (RCSI)

Bruno C. Vellutini (MPI CBG)

Carl D. Modes (CSBD/MPI-CBG)

Curtis Rueden (UW-Madison LOCI)

Damir Krunic (DKFZ)

Daniel J. White (GE)

Deborah Schmidt (CSBD/MPI CBG)

Douglas P. Shepard (Univ Arizona)

Eduardo Conde-Sousa (Univ. of Porto)

Erick Ratamero (The Jackson
Laboratory)

Gaby G. Martins (IGC)

Gayathri Nadar (MPI CBG)

Guillaume Witz (Bern University)

Giovanni Cardone (MPI Biochem)

Irene Seijo Barandiaran (MPI CBG

Dresden)

Jan Brocher (Biovoxxel)

Juergen Gluch (Fraunhofer IKTS)

Kisha Sivanathan (Harvard Medical

School)

Kota Miura

Laurent Thomas (Acquifer)

Lior Pytowski (University of Oxford)

Loic A. Royer (CZ Biohub)

Marion Louveaux (Institut Pasteur
Paris)

Martin Weigert (EPFL Lausanne)

Matthew Foley (University of Sydney)

Nico Stuurman (UCSF)

Nicola Maghelli (MPI CBG)

Nicolas Brouilly (IBDM Marseille)

Nik Cordes (Los Alamos National

Laboratory)

Noreen Walker (MPI CBG Dresden)

Ofra Golani (Weizmann Institute of

Science)

Patrick Dummer

Peter Haub

Pete Bankhead (University of

Edinburgh)

Peter Steinbach (HZDR Dresden)

Pit Kludig

Rita Fernandes (University of Porto)

Romain Guiet (BIOP-EPFL)

Ruth Whelan-Jeans

Sebastian Munck (VIB Leuven)

Siân Culley (MRC LMCB)

Stein Rørvik

Stéphane Dallongeville (Pasteur)

Tanner Fader (UNC-Chapel Hill)

Thomas Irmer (Zeiss)

Tobias Pietzsch (MPI-CBG)

Uwe Schmidt (MPI CBG)

Wilson Adams (VU Biophotonics)

MPI CBG Core Facilities

- Advanced Imaging Facility
- Light Microscopy Facility
- Scientific Computing
- IT Department
- Fly lab



<https://fiji.sc>



<https://napari.dev>



<https://image.sc>



<http://eubias.org/>
NEUBIAS/

Funding



R.H. was supported by the German Federal Ministry of Research and Education (BMBF) under the code 031L0044 (Sysbio II) and by the Deutsche Forschungsgemeinschaft under Germany's Excellence Strategy - EXC-2068 Cluster of Excellence Physics of Life of TU Dresden.



DRESDEN
concept

