

Designing GPU-accelerated Image Data Flow Graphs for CLIJ2 and clEsperanto and deployment to ImageJ, Fiji, Icy, Matlab, QuPath, Python, napari and C++

Robert Haase DFG Cluster of Excellence Physics of Life, TU Dresden

Stéphane Rigaud Image Analysis Hub, C2RT, Institut Pasteur Paris

Introduction



Stéphane Rigaud

Image Analysis Hub, C2RT,
Institut Pasteur Paris



Robert Haase

Bio-image Analysis Technology Development group
DFG Cluster of Excellence “Physics of Life”, TU Dresden

Tutorial material download: https://github.com/clesperanto/i2k2020_tutorial_clij_clesperanto/

Acknowledgements



Akanksha Jain
(Treutlein lab)
@jain_akanksha



Dani Vorkel
(Myers lab)
@happifocus



Theresa Suckert
(Krause lab,
Oncoray)



Pradeep Rajasekhar
(Poole lab, Monash U)
@pr4deepr



Talley J. Lambert
(HMS)
@TalleyJLambert



Juan Nunez-Iglesias
(Monash)
@jnuneziglesias



Pavel Tomancak
(CSBD / MPI-CBG)
@PavelTomancak



Gene Myers
(CSBD / MPI-CBG)
@TheGeneMyers



Jean-Yves Tinevez
(Institut Pasteur)
@jytinevez



Florian Jug
(CSBD / MPI-CBG)
@florianjug



Szabolcs Horvát
(Modes lab)
@schorvat



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 (University 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)

Jean-Yves Tinevez (Institut Pasteur)

Johannes Girstmair (MPI CBG)

Juergen Gluch (Fraunhofer IKTS)

Kisha Sivanathan (Harvard Medical School)

Kota Miura

Laurent Thomas (Acquierer)

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 (Institut Pasteur)

Tanner Fader (UNC-Chapel Hill)

Thomas Irmer (Zeiss)

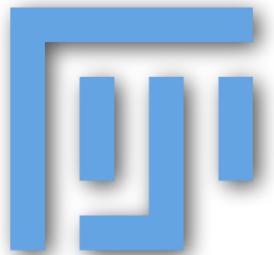
Tobias Pietzsch (MPI-CBG)

Uwe Schmidt (MPI CBG)

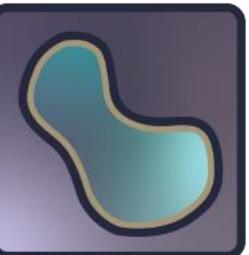
Wilson Adams (VU Biophotonics)

MPI CBG Core Facilities

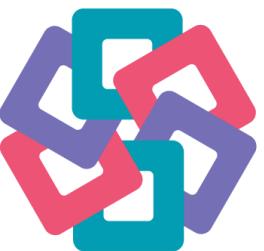
- Advanced Imaging Facility
- Light Microscopy Facility
- Scientific Computing
- IT Department



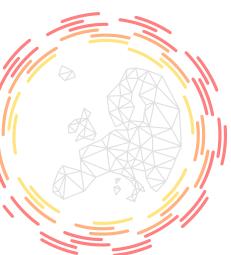
<https://fiji.sc>



<https://napari.dev>



<https://image.sc>



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

Funding:

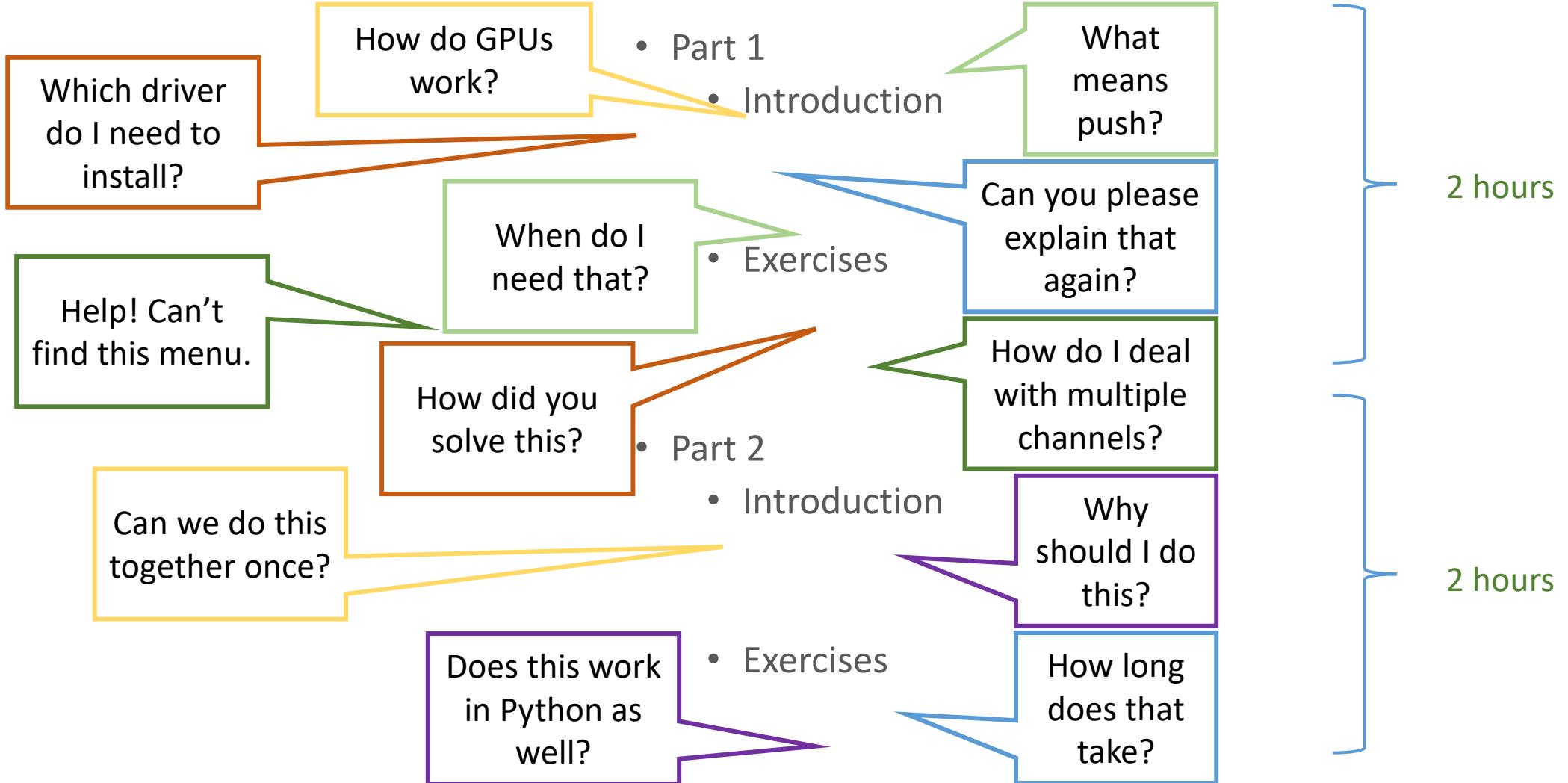


R.H. was supported by the German Federal Ministry of Research and Education (BMBF) under the code 031L0044 (Sysbio II) and by the Deutsche Forschungsgemeinschaft under Germany's Excellence Strategy - EXC2068 - Cluster of Excellence Physics of Life of TU Dresden.

This tutorial



Please introduce yourself before asking a question!



Part I: Programming workflows



- Basics
 - Push / pull: Data transfer
 - Image processing operations
 - Memory management
- Code recording
 - Icy Bioimaging
 - ImageJ/Fiji
- Auto-completion
- Languages
 - ImageJ Macro
 - Fiji Jython
 - Matlab
 - Icy JavaScript
- Exercises

The screenshot shows the Icy software interface. At the top, there's a toolbar with icons for various functions like Image/S, Region Of, Image, Detection, Process, Tools, CLU2 fil, CLU2, and Plugin. Below the toolbar, there are two main panels: 'Transformation' and 'Binary'. The 'Transformation' panel contains nodes for AffineTransform2D, AffineTransform3D, and ApplyVectorField2D. The 'Binary' panel contains nodes for BinaryAnd, BinaryOr, and BinaryXOr. In the center, there's a 'Script Editor - new 11' window with a tab bar for File, Edit, Templates, Options, Untitled, and new 11*. The code in the editor is as follows:

```
1 importClass(net.haesleinhuepf.clicy.CLIY);
2 importClass(Packages.icy.main.Icy);
3 clij2 = CLIY.getInstance();
4
5 // CLIJ2_GaussianBlur2DBlock
6 sequence1 = getSequence();
7 buffer2 = clij2.pushSequence(sequence1);
8 buffer3 = clij2.create([256, 254], clij2.Float);
9 clij2.gaussianBlur(buffer2, buffer3, 5.0, 5.0, 0.0);
10 sequence4 = clij2.pullSequence(buffer3);
11 Icy.addSequence(sequence4);
12
13 // CLIJ2_ThresholdTriangleBlock
14 buffer5 = buffer3;
15 buffer6 = clij2.create([256, 254], clij2.Float);
16 clij2.thresholdTriangle(buffer5, buffer6);
17 clij2.conn
sequence7
Icy.addSeq
connectedComponentsLabeling(ClearCLImageInterface a
connectedComponentsLabelingBox(ClearCLImageInterface a
// CLIJ2_C
buffer8 =
buffer9 =
clij2.conn
sequence10
Icy.addSeq
//Clean up memory
clij2.clear();
```

To the right of the script editor, there's a 'Fiji Is Just ImageJ' window showing various image processing tools like Crop, Paste, Rotate, Translate, and Transform. Below these windows, there are two tabs: '*tribolium_morphometry.ijm' and '*New_ijm'. The '*tribolium_morphometry.ijm' tab shows a large block of IJ macro code. The '*New_ijm' tab shows a smaller portion of the same code with some lines highlighted in yellow. A tooltip for the 'aver' command is visible at the bottom right.

CLIJ2 cheat sheet: ImageJ macro I
GPU-accelerated image processing in Fiji
CBERG SYSTEMS BIOLOGY

Operation **Parameters** **Result** **Dim** **Examples**

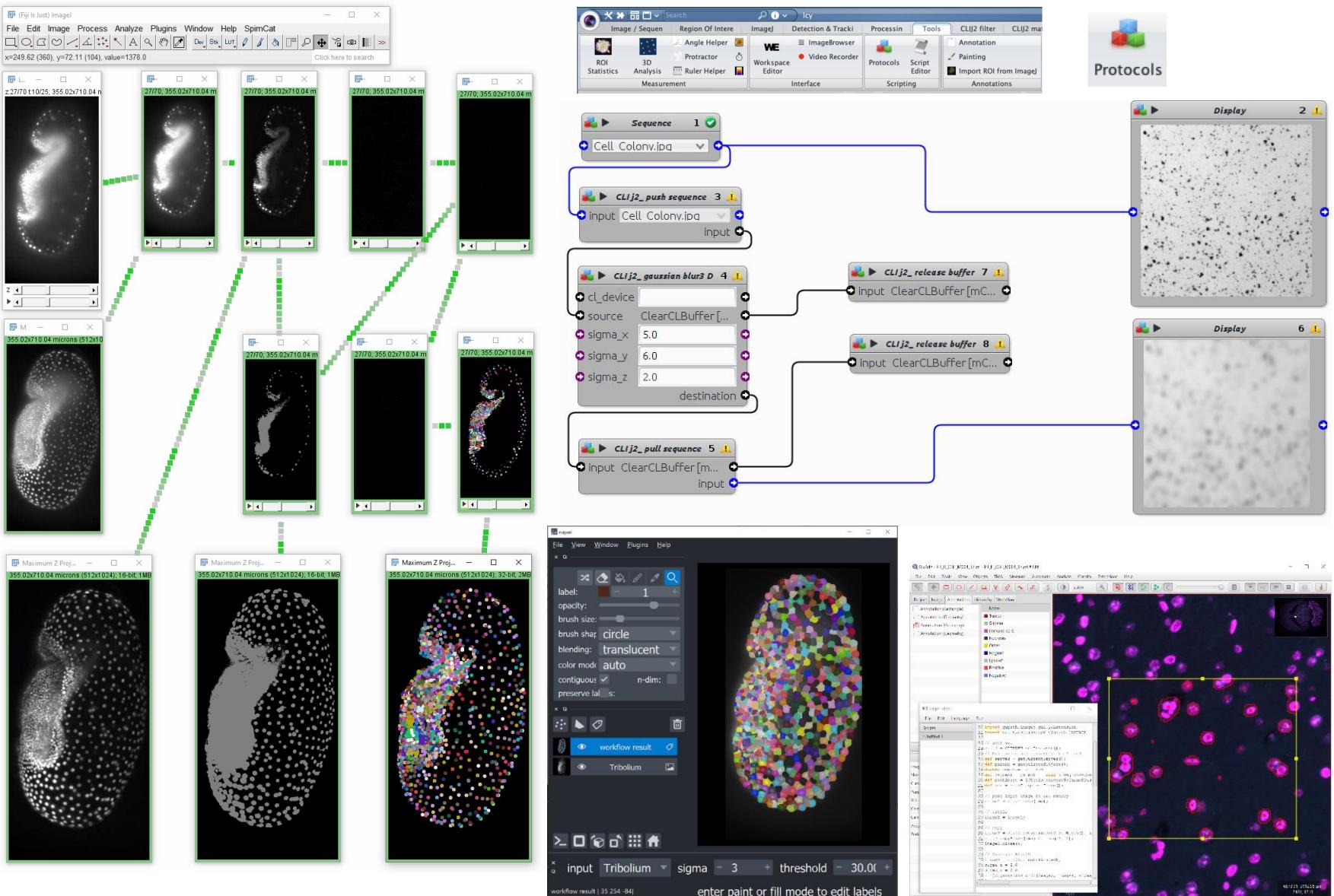
Initiate CLU	[I] HD, GFX or CPU	run("CLU2 Macro: Dimensions", "ui_device=HDI");		
Push	[I]	2D	input = getStack(); Ext.CLIJ2_push(input);	
Pull	[O]	2D	// set stack from GPU back	
Create	1024, 1024, 8	2D	Ext.CLIJ2_create2D("new2D", w, h, b16Depth);	
Convert		2D	Ext.CLIJ2_convertFromImage("result", input);	
Copy		2D	Ext.CLIJ2_copy(input, result);	// duplicate
Copy slice	50	3D	Ext.CLIJ2_copySlice(stack, slice, sliceIndex);	// put a slice into a stack
Crop	20, 20	2D	Ext.CLIJ2_crop(input, "cropped", w, y, width, height);	// copy a slice out of a stack
Paste	, 9, 9	2D	Ext.CLIJ2_paste("cropped", "target", w, y);	// paste image
Release		2D	Ext.CLIJ2_release("image name");	
Clear			Ext.CLIJ2_clear(); // empty GPU memory	
Rotate by 90 degrees		2D	Ext.CLIJ2_rotate90(input, result);	
Rotate	45, true	2D	Ext.CLIJ2_rotate(input, result, angle, rotateStack);	
Translate	20, 20	2D	Ext.CLIJ2_translate(input, result, flipX, flipY);	
Affine transform	zscale=43	2D	transf = affineTransform(zscale); result = transf * input;	
Deform / warp		2D	Ext.CLIJ2_warp(input, result, vstack, vstackID);	
Projections		2D	Ext.CLIJ2_projections(result, arg);	

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

Part II: Interactive workflow design



- Designing Image data flow graphs
- Expert system
- Code-generation
 - ImageJ Macro
 - Python
 - C++
 - QuPath Groovy
 - Matlab
 - Java/Maven
- Exercises
- Work on your own data



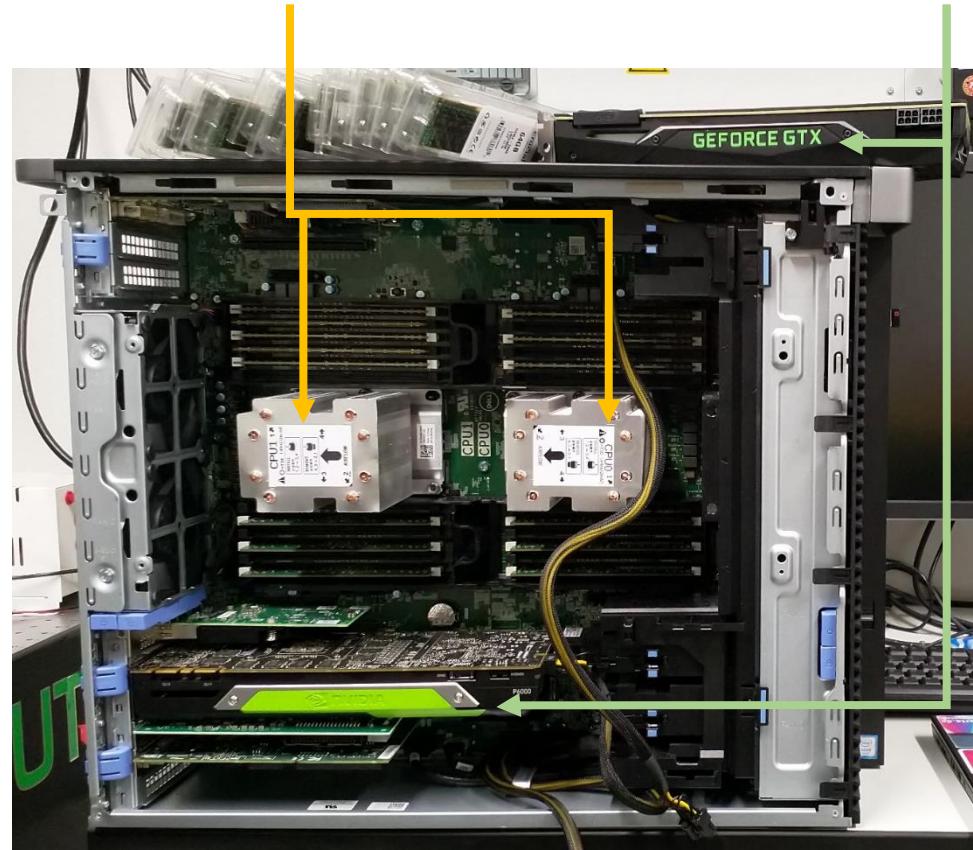
Introduction: [Image] processing on Graphics Processing Units

GPU-accelerated image processing

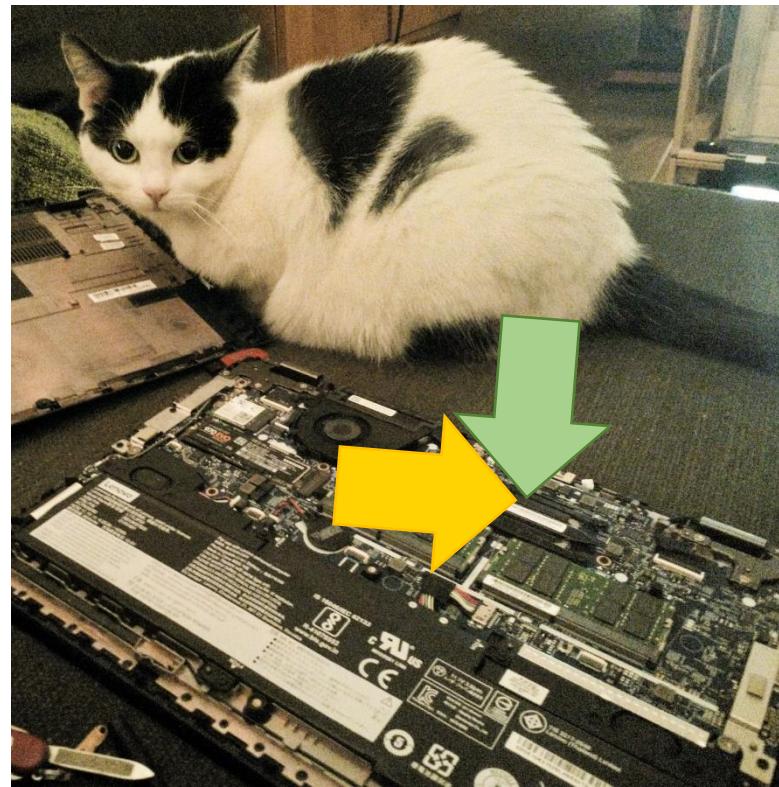


- Typical computers contain Graphics Processing Units

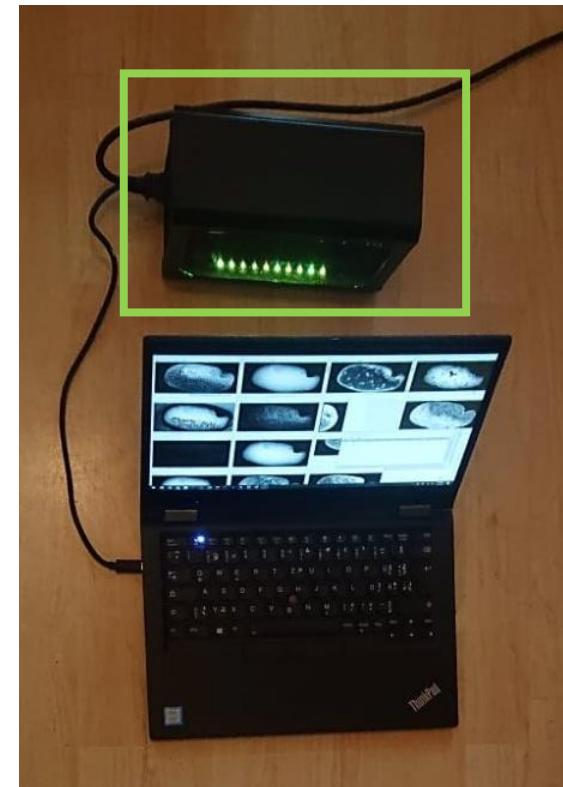
Central Processing Unit (CPU)



Graphics Processing Unit (GPU)



Most laptops contain *integrated* GPUs

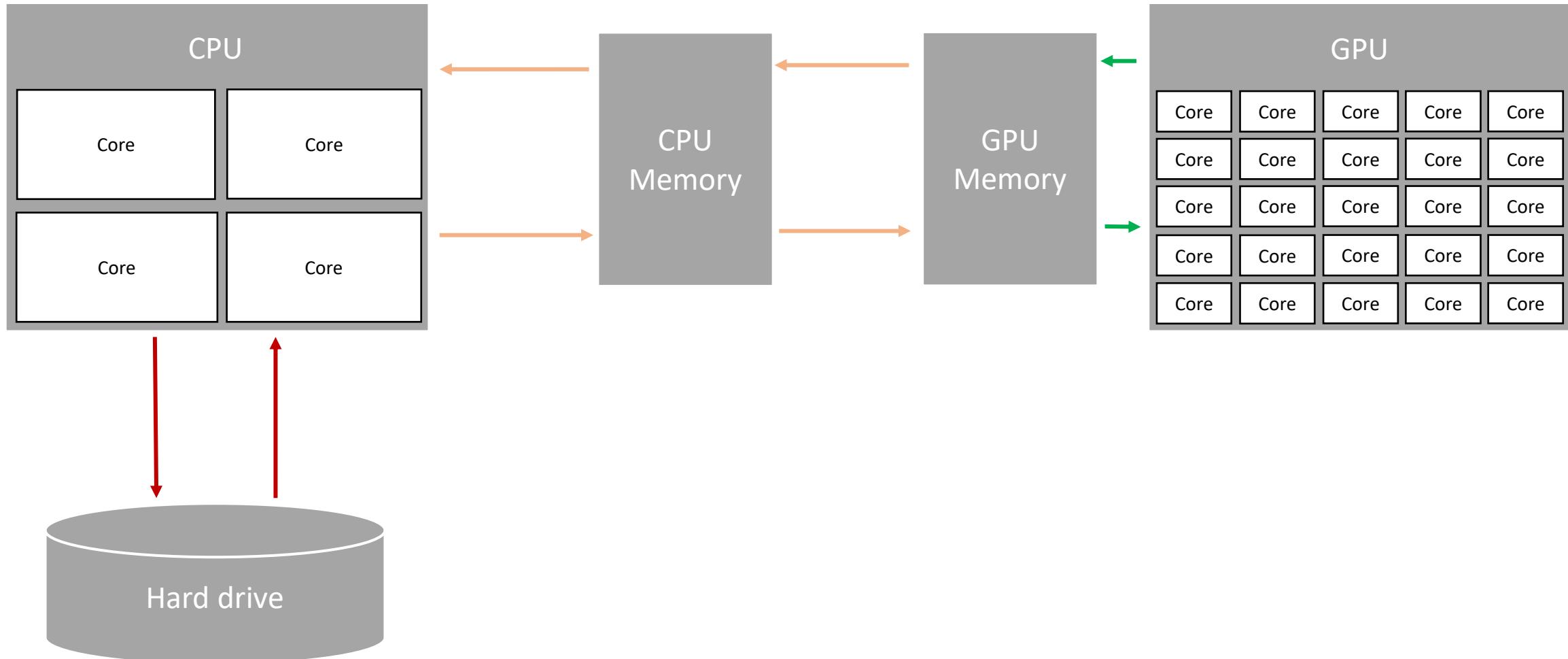


Alternative: *external* GPUs

GPUs allow real-time image processing



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



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 Gb/s).

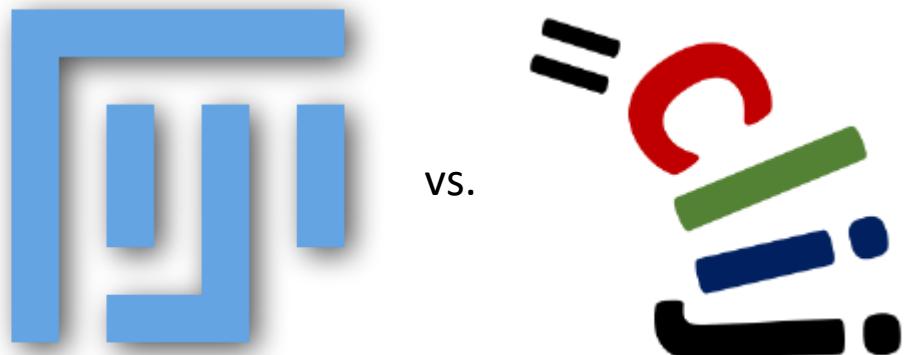
Disclosure: I don't receive any money or anything from any GPU vendor.

What can we gain from GPU-accelerated image processing?

GPU-accelerated image processing



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



Workstation CPU

2x Intel Xeon Silver 4110

Workstation GPU

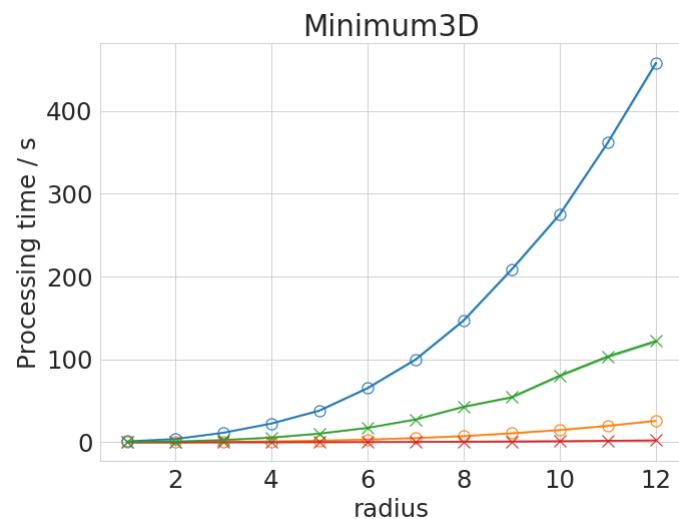
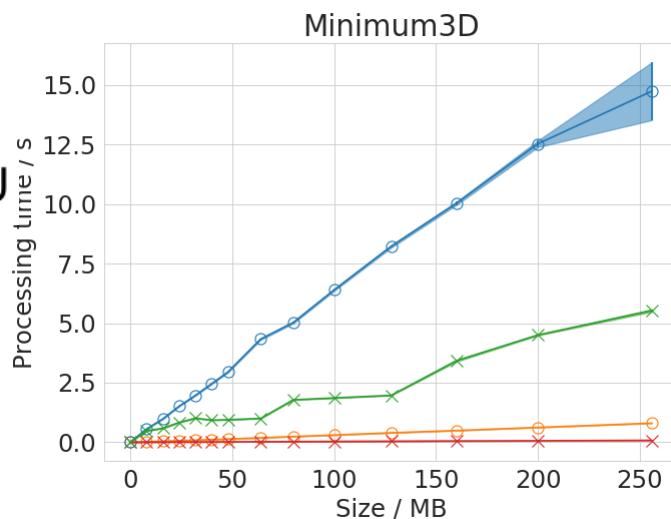
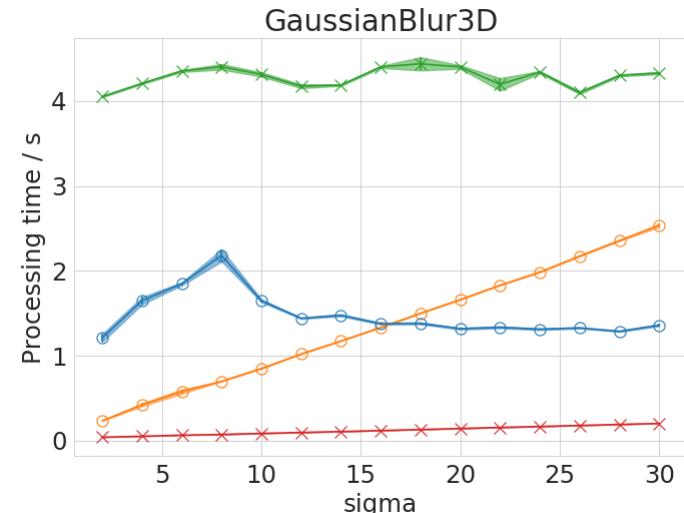
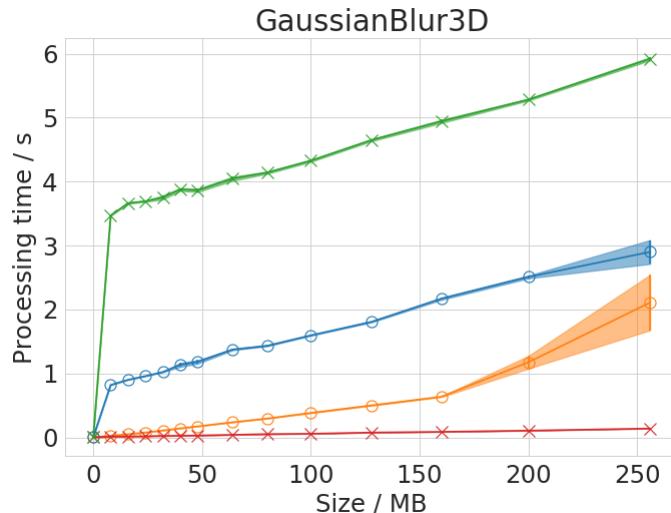
Nvidia Quadro P6000

Laptop CPU

Intel Core i7-8650U

Laptop GPU

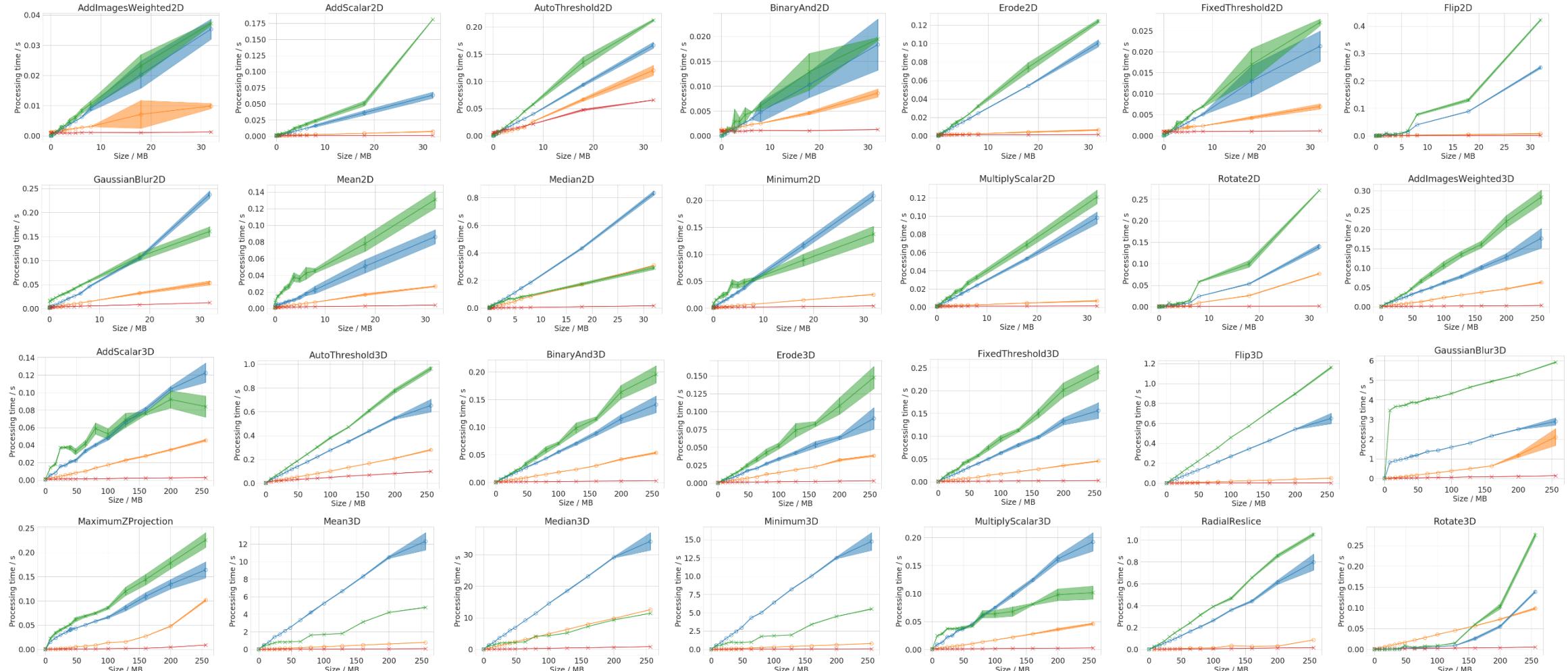
Intel UHD 620 GPU



GPU-accelerated image processing



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



laptop CPU laptop GPU workstation CPU workstation GPU

- 8 MB (2D)



- 64 MB (3D)



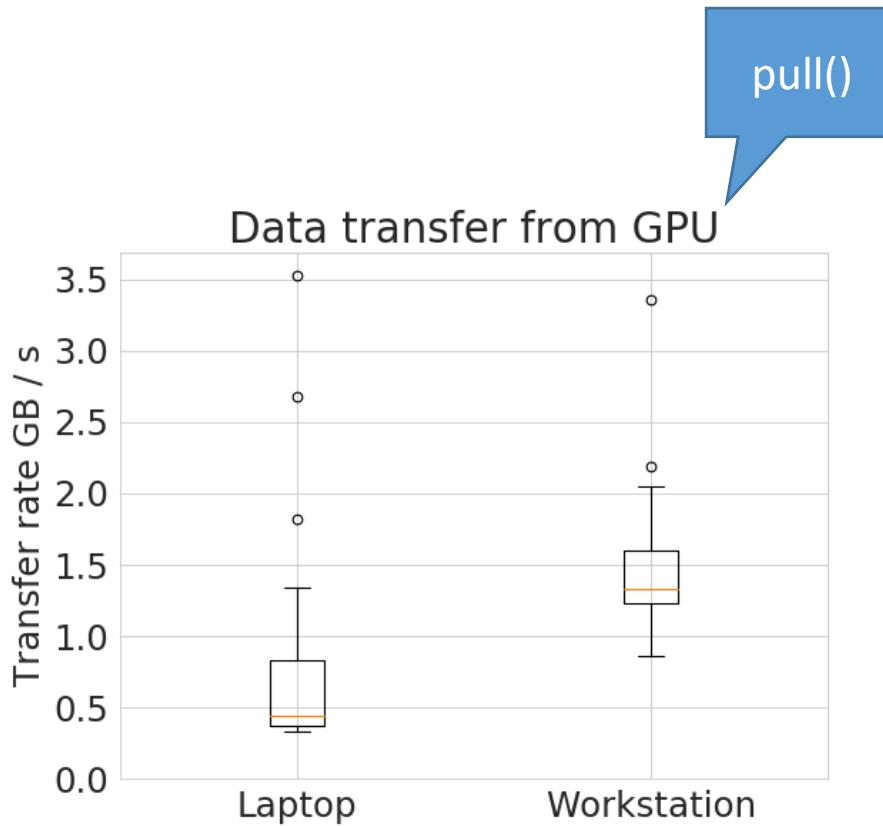
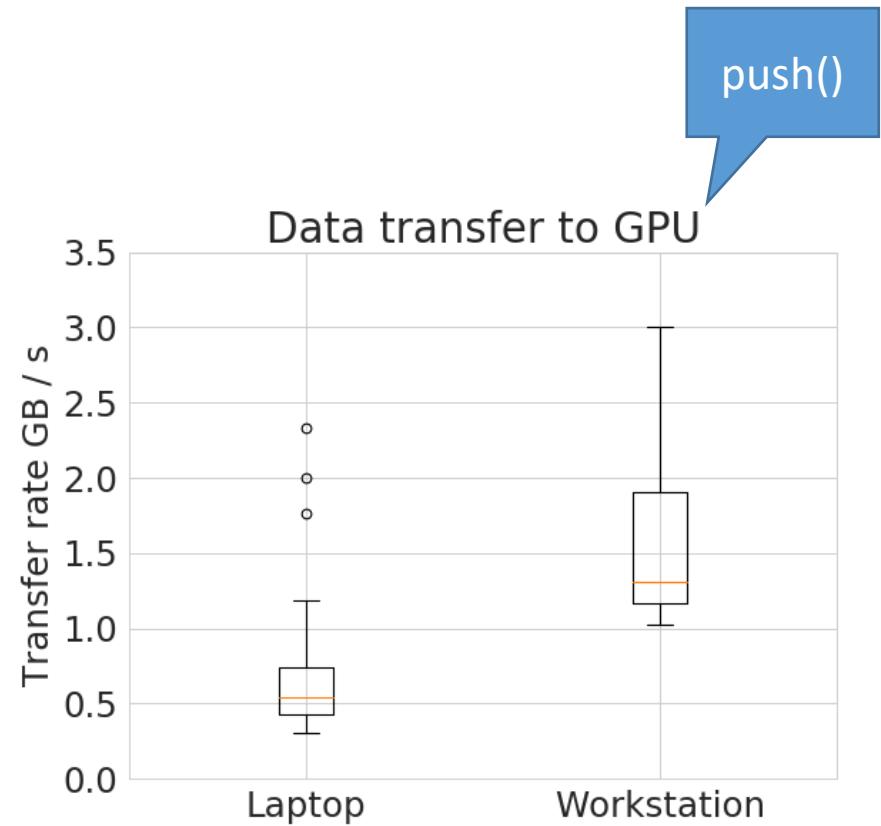
Speedup compared to Laptop CPU

	Laptop GPU	Workstation GPU
AddImagesWeighted2D	3	8
AddScalar2D	7	14
AutoThreshold2D	2	2
BinaryAnd2D	2	4
Erode2D	11	20
FixedThreshold2D	2	5
Flip2D	16	37
GaussianBlur2D	3	9
Mean2D	3	10
Median2D	2	35
Minimum2D	7	22
MultiplyScalar2D	10	21
Rotate2D	3	22
AddImagesWeighted3D	3	26
AddScalar3D	3	23
AutoThreshold3D	3	5
BinaryAnd3D	3	24
Erode3D	2	13
FixedThreshold3D	4	30
Flip3D	15	119
GaussianBlur3D	6	35
MaximumZProjection	7	46
Mean3D	18	150
Median3D	3	43
Minimum3D	23	188
MultiplyScalar3D	4	28
RadialReslice	14	42
Rotate3D	0.1	2

Data transfer takes time



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

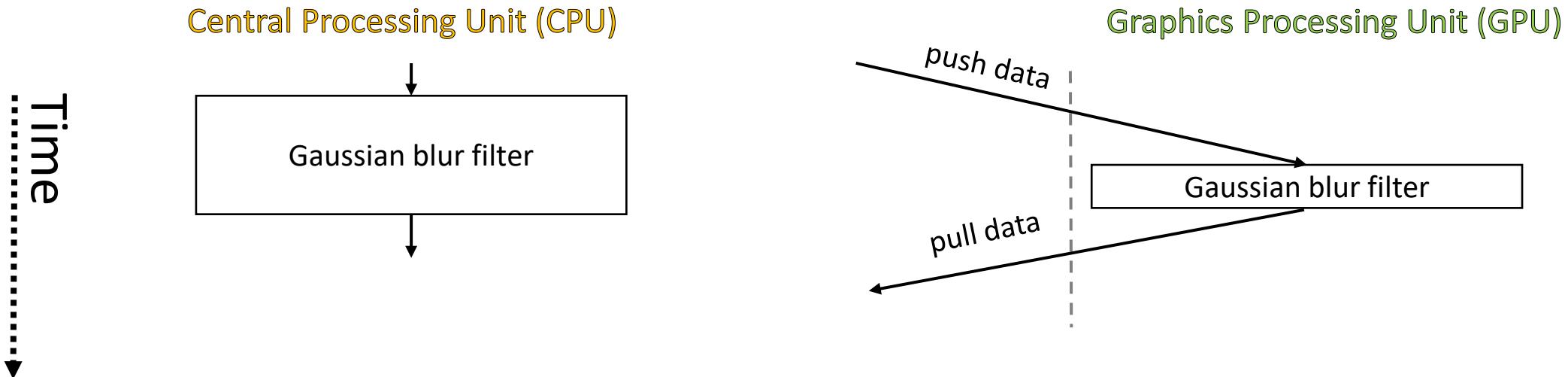


- Rule of thumb: One gigabyte takes one second.

Build workflows consisting of many operations



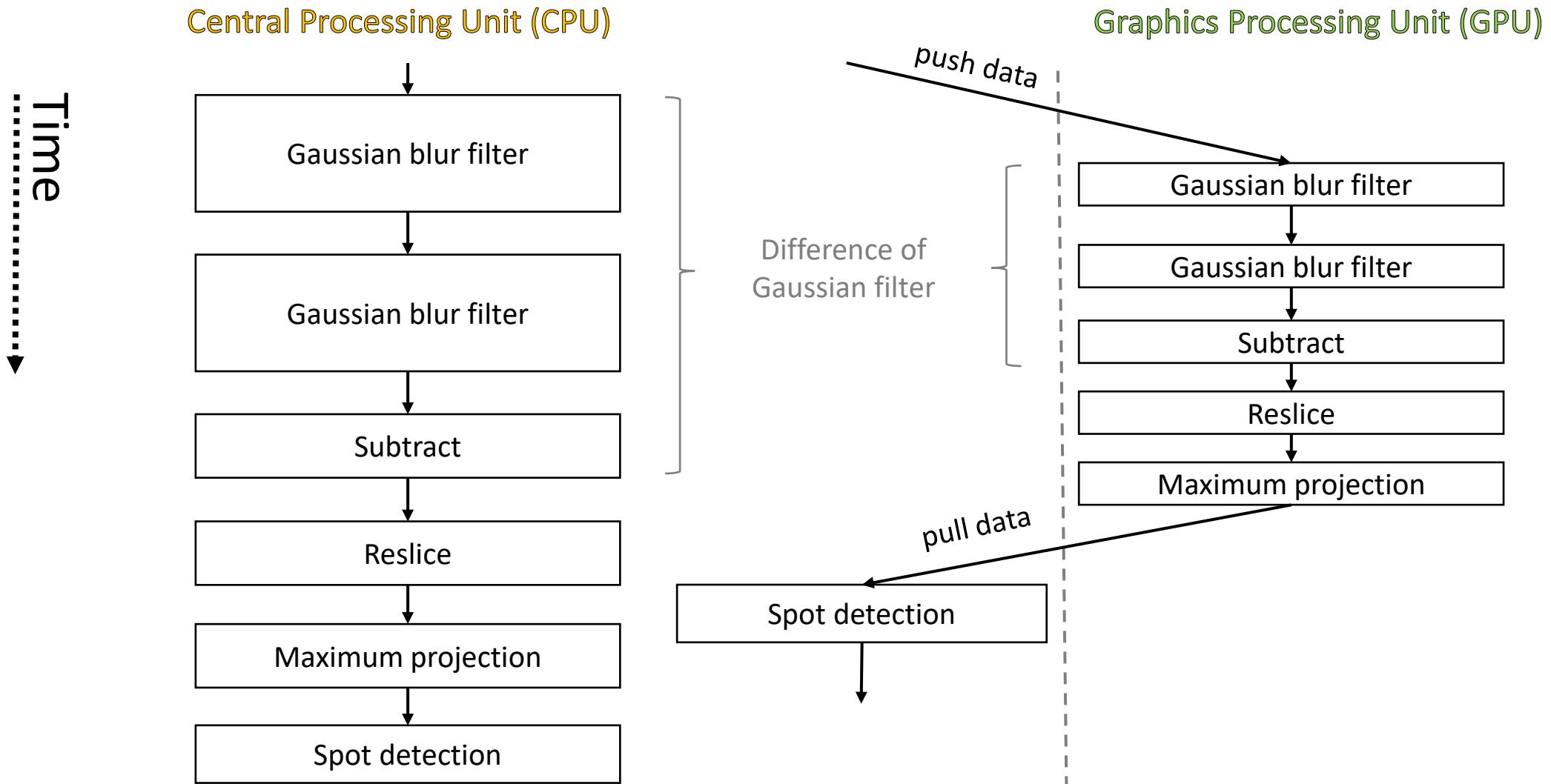
- GPU acceleration may suffer from data transfer between CPU and GPU



Build workflows consisting of many operations



- GPU acceleration may suffer from data transfer between CPU and GPU



Optimal performance through smart memory management



- Example workflow

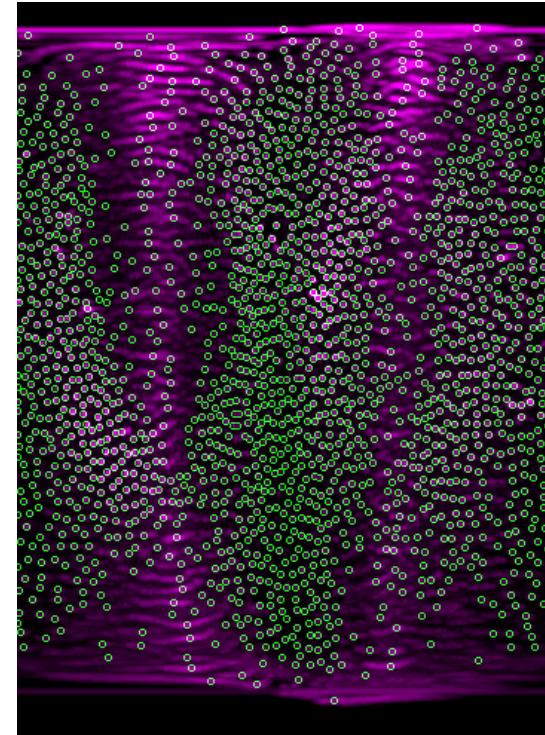
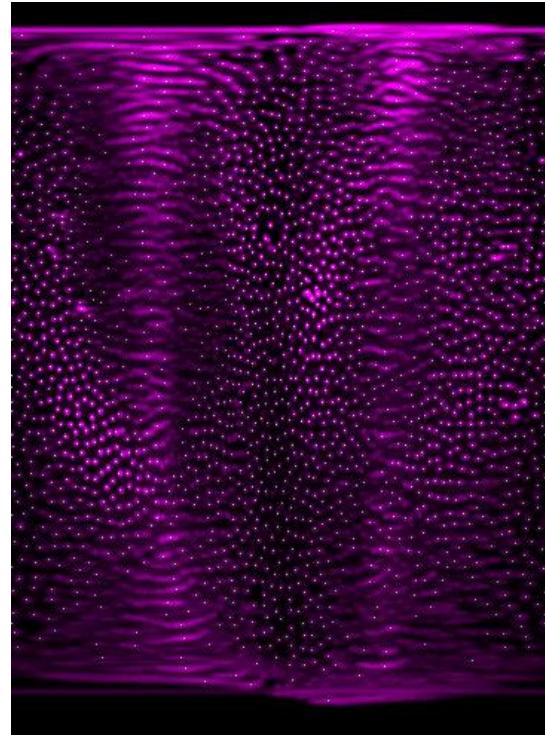
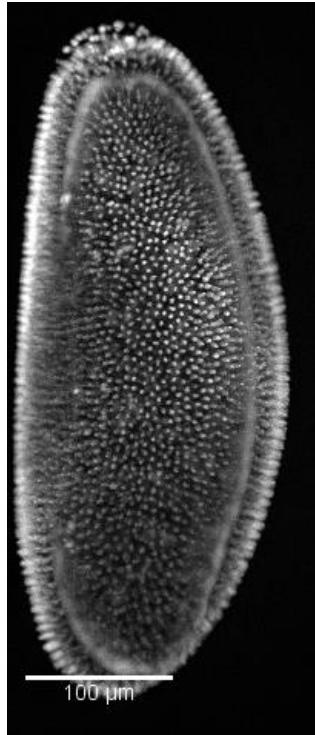
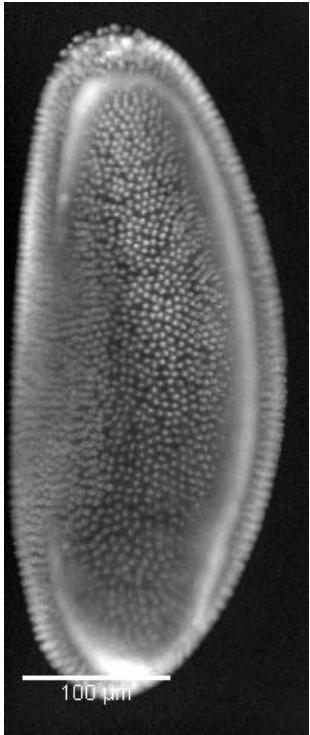
Load data

Preprocessing

Transformation

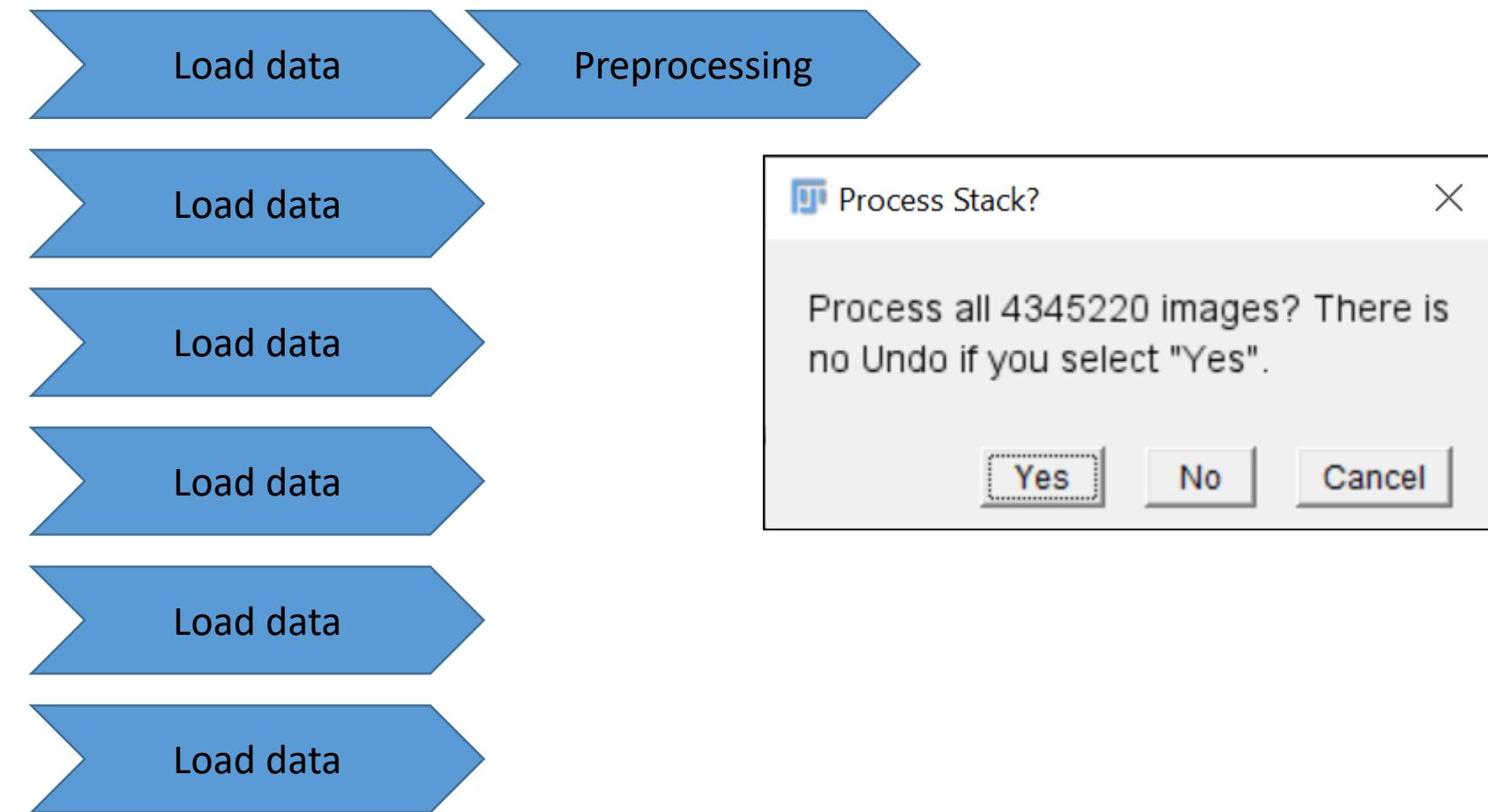
Segmentation

Save data

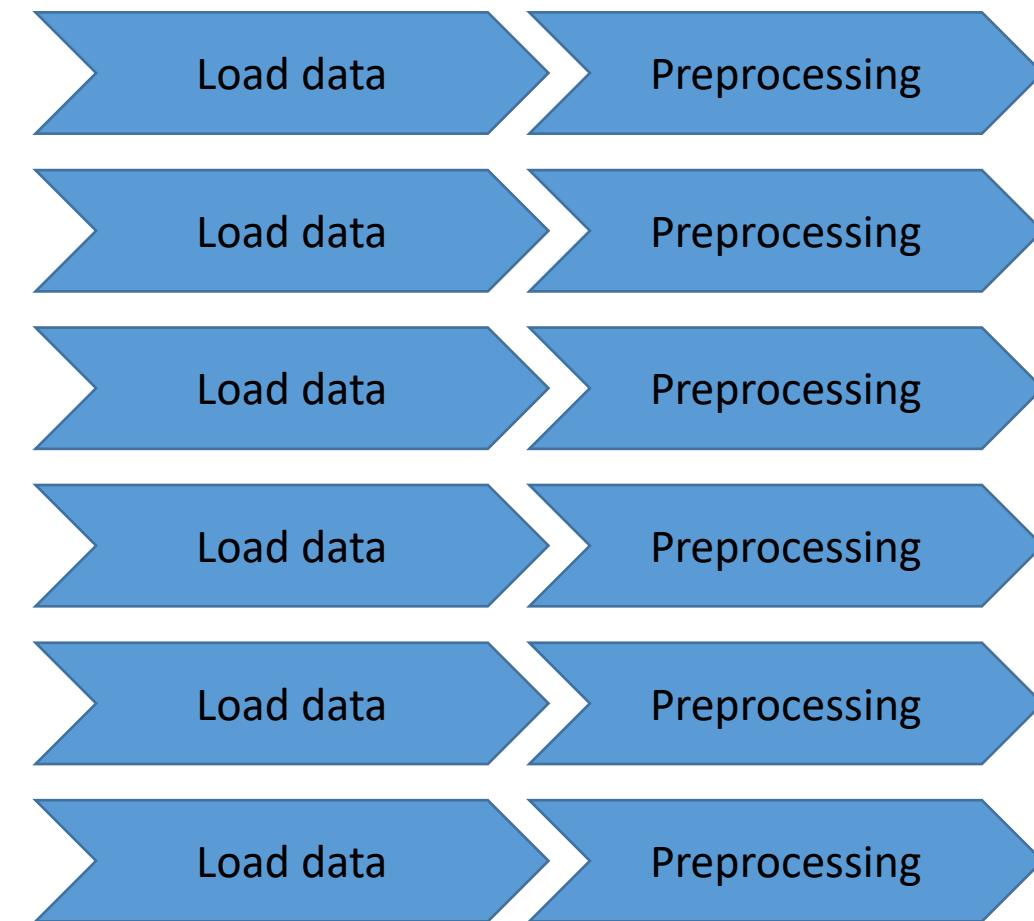




- The classical way of dealing with large image stacks...



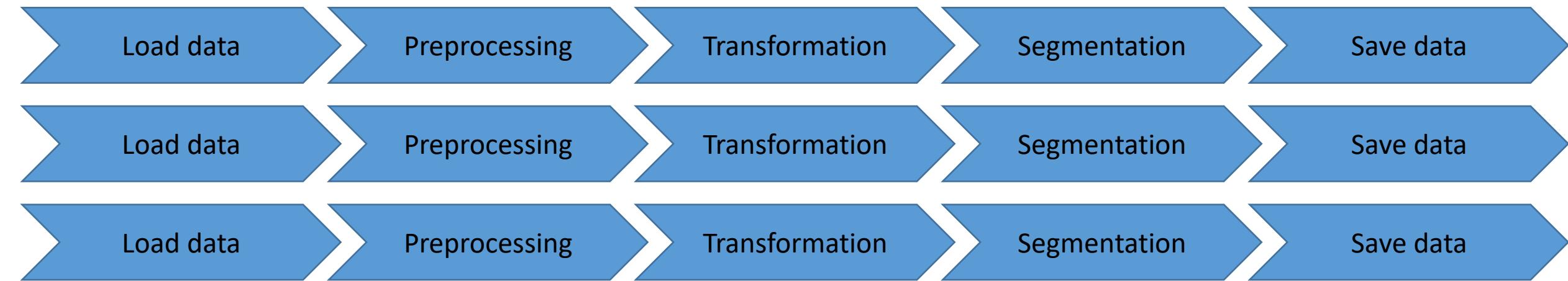
- The classical way of dealing with large image stacks... is suboptimal for memory consumption and performance.



This strategy does not just take long; it also costs a lot of memory!



- Processing time-point by time-point is more efficient!



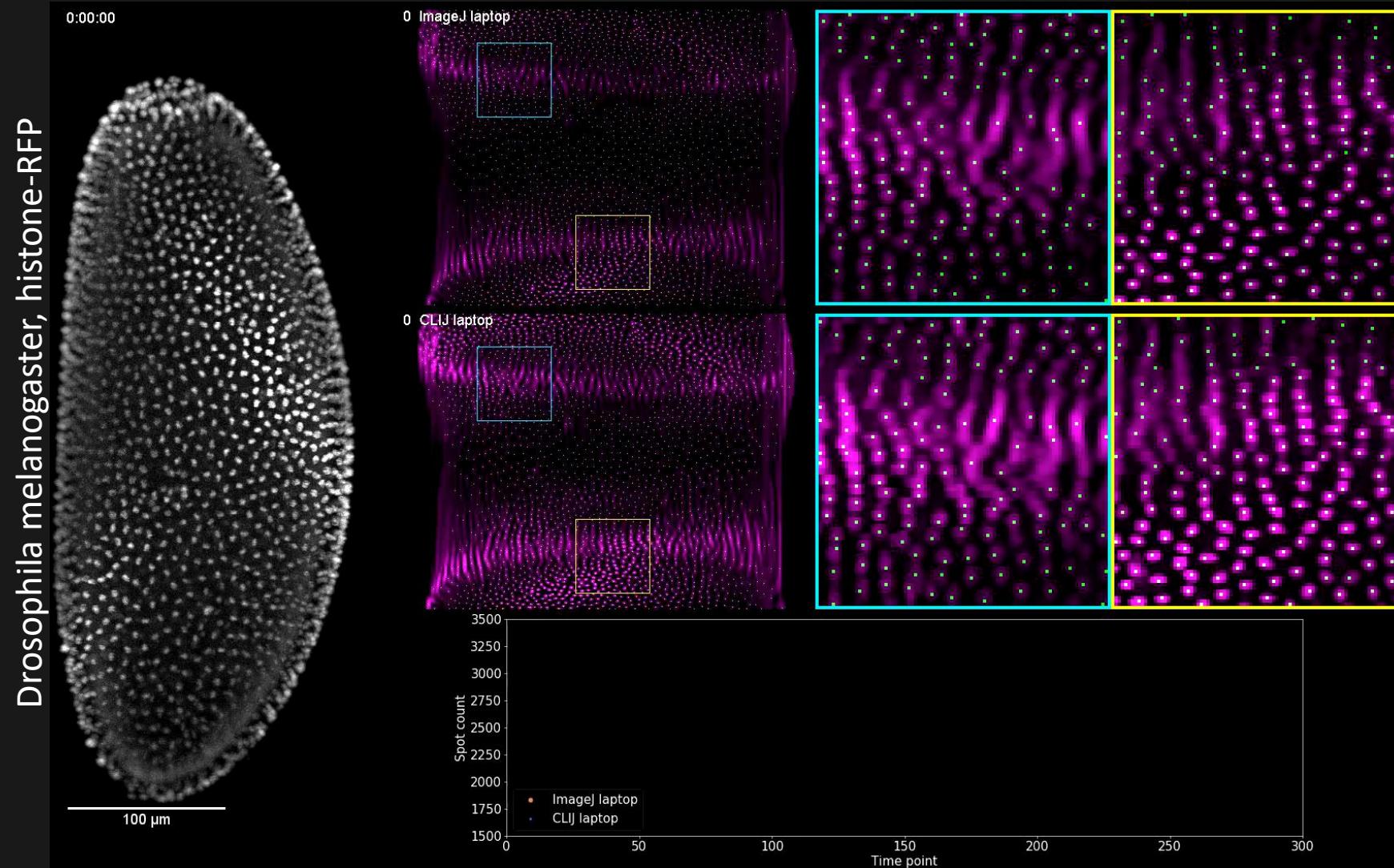
This strategy also works tile-by-tile on large 3D stacks!

Reuse memory!
See exercise 11

Optimal performance through smart memory management



- Counting spots in 300 frames of light sheet data takes:



ImageJ on CPU (laptop)

2:44 h
(31 seconds per frame)

ImageJ using the GPU (laptop)

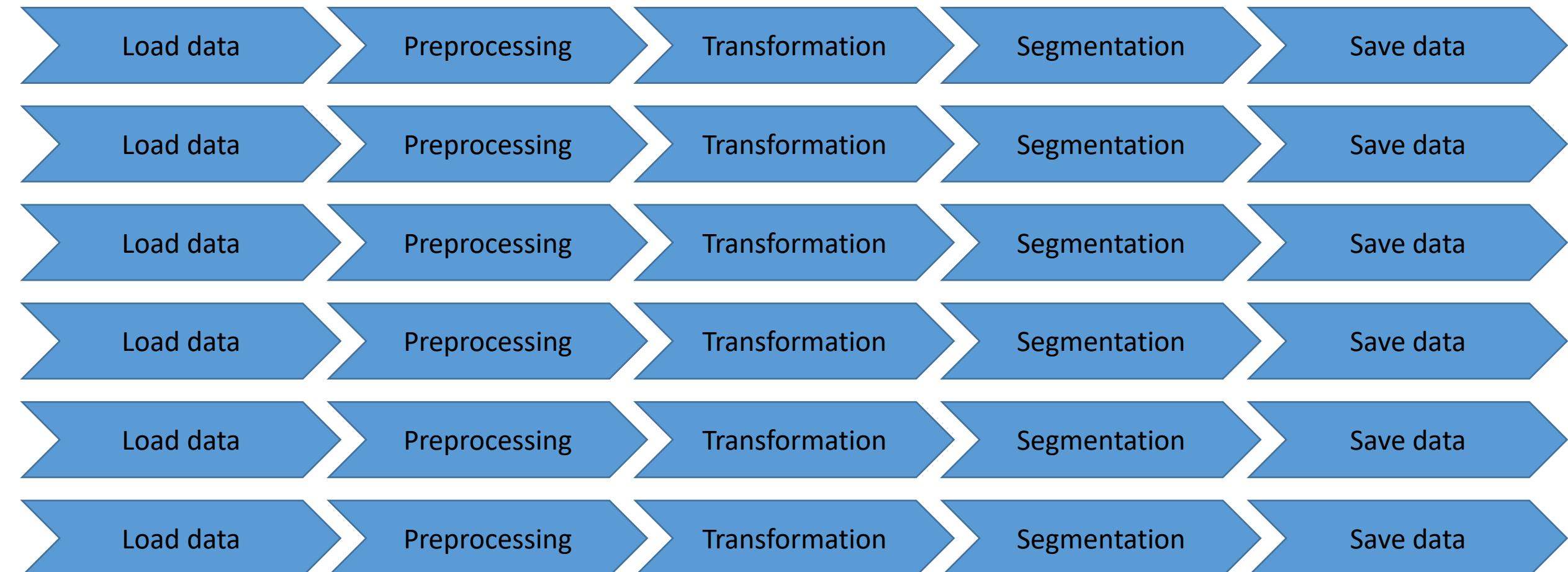
11 min
(1.2 seconds per frame)

ImageJ using a dedicated Nvidia GPU (workstation)

5 min
(0.4 seconds per frame)

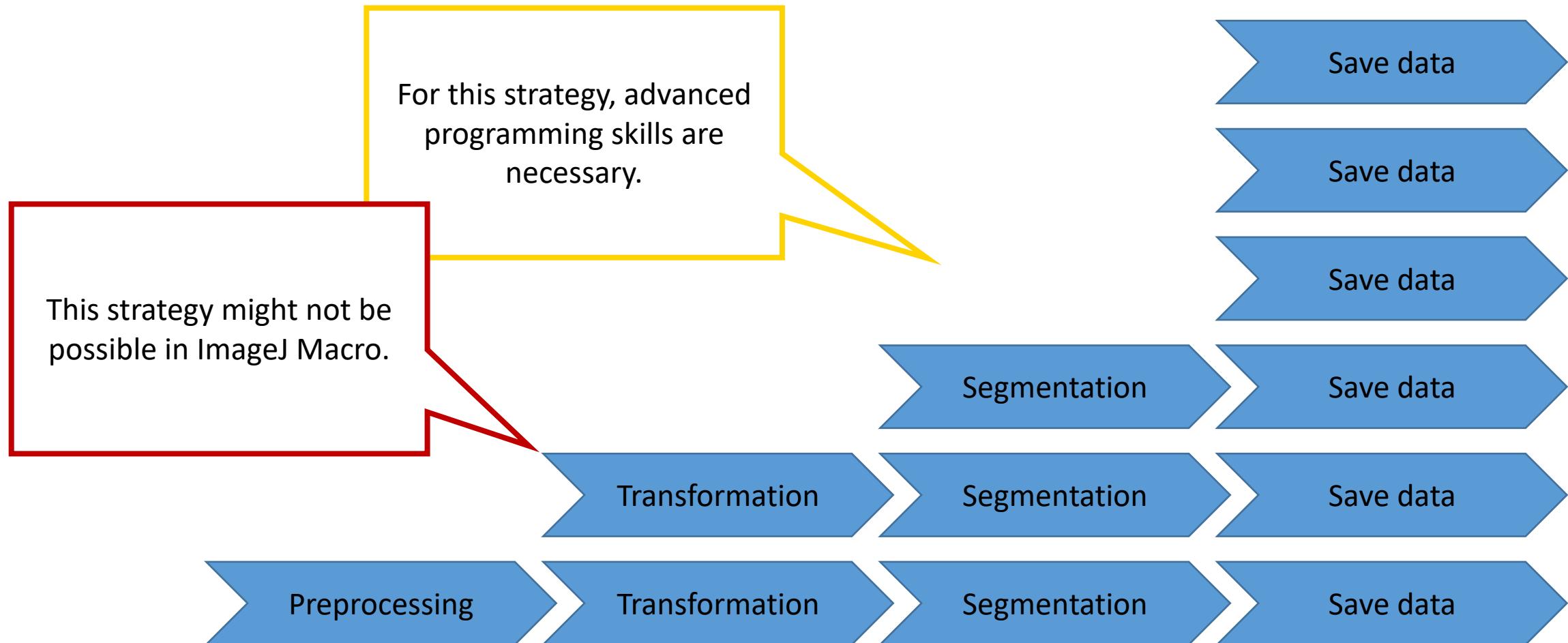


- Even better: Distribute tasks between parallelized computation systems





- Even better: Distribute tasks between parallelized computation systems



Scripting CLIJ2

CLIJ2: What every ImageJ Macro script must have



- Load data

```
1 // Load data  
2 run("Cell Colony (31K)");  
3
```

- Initialize GPU

```
4 // initialize GPU  
5 run("CLIJ2 Macro Extensions", "cl_device=");  
6 Ext.CLIJ2_clear();  
7
```

- Push

```
8 // push image to GPU  
9 input_image = getTitle();  
10 Ext.CLIJ2_push(input_image);  
11
```

- Process images

```
12 // process image  
13 sigma = 5;  
14 Ext.CLIJ2_gaussianBlur2D(input_image, result_image, sigma, sigma);  
15
```

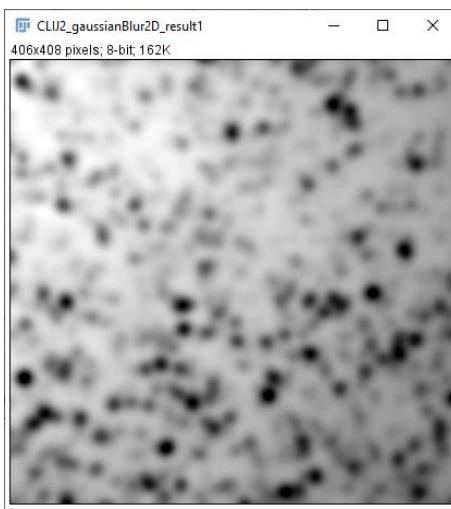
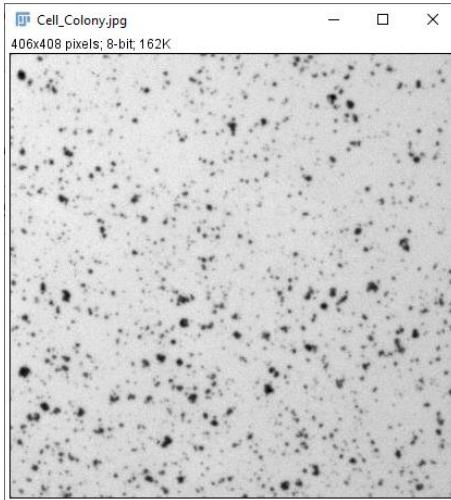
- Pull

```
16 // optional: release input data  
17 Ext.CLIJ2_release(input_image);  
18
```

- Cleanup

```
19 // pull result back from GPU  
20 Ext.CLIJ2_pull(result_image);  
21
```

```
22 // clean up by the end  
23 Ext.CLIJ2_clear();
```



CLIJ2: What every ImageJ Jython script must have



- Load data

```
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
```

- Initialize GPU

```
6 # initialize GPU
7 from net.haesleinhuepf.clij2 import CLIJ2;
8 clij2 = CLIJ2.getInstance()
9
```

- Push

```
10 # push image to GPU
11 input_image = clij2.push(imp)
12
```

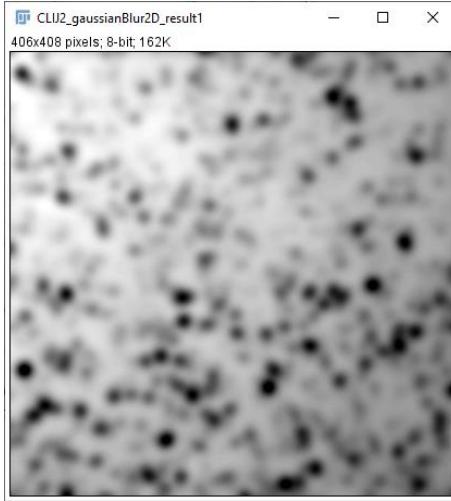
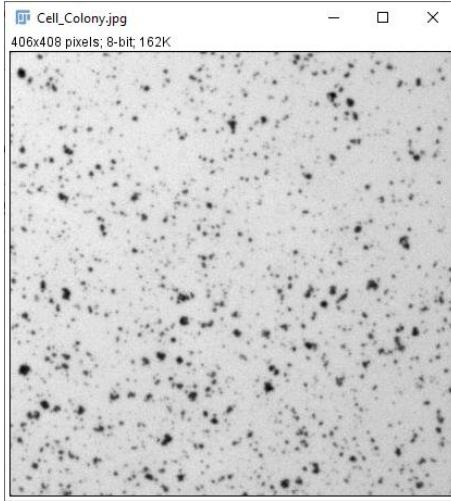
- Process images

```
13 # process image
14 sigma = 5
15 blurred = clij2.create(input_image)
16 clij2.gaussianBlur2D(input_image, blurred, sigma, sigma)
17
18 # optional: release input data
19 input_image.close()
20
```

- Pull

```
21 # pull result back from GPU
22 imp = clij2.pull(blurred)
23 imp.show()
24
25 # clean up by the end
26 clij2.clear()
```

- Cleanup



CLIJ2: What every Matlab script must have



- Load data

```
1 % load data
2 - image = imread("http://imagej.nih.gov/ij/images/Cell_Colony.tif");
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

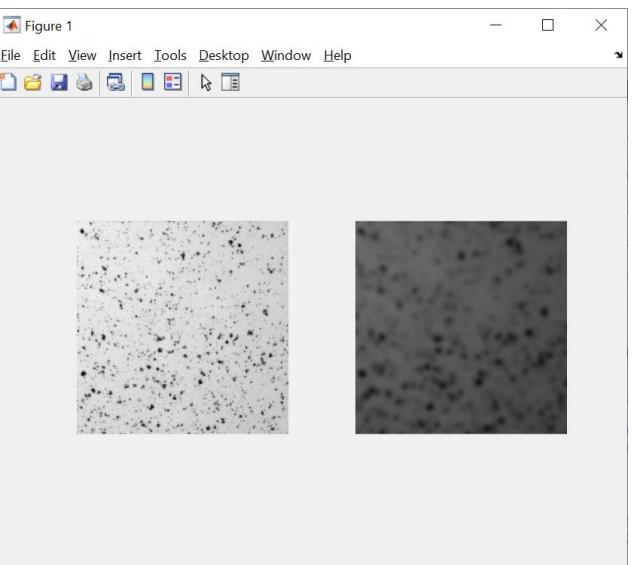
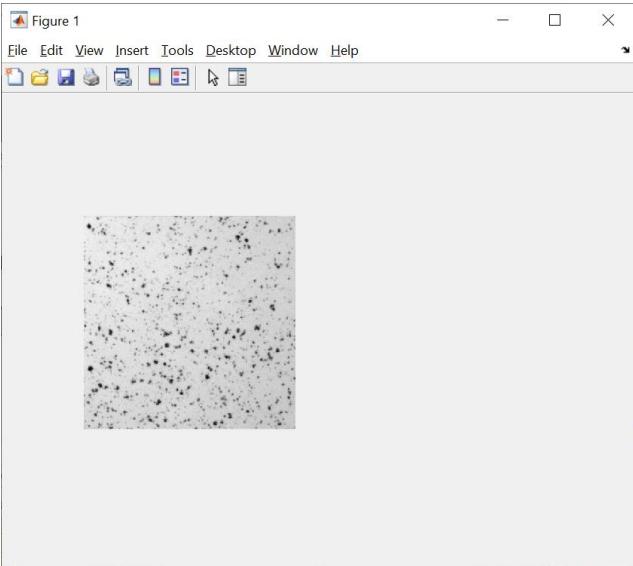
```
16 % optional: release input data
17 - input_image.close()
```

- 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()
```



Icy Bioimaging

CLIJ2: What every ICY javascript must have



- Load data

```
1 // load data
2 importClass(Packages.icy.main.Icy);
3 importClass(Packages.icy.file.Loader);
4 seq = Loader.loadSequence("Cell_Colony.jpg", 0, true);
5
```

- Initialize GPU

```
6 // initialise GPU
7 importClass(net.haesleinhuepf.clicy.CLICY);
8 clij2 = CLICY.getInstance();
9
```

- Push

```
10 // push data to GPU
11 sequence1 = getSequence();
12 buffer2 = clij2.pushSequence(sequence1);
13
```

- Process images

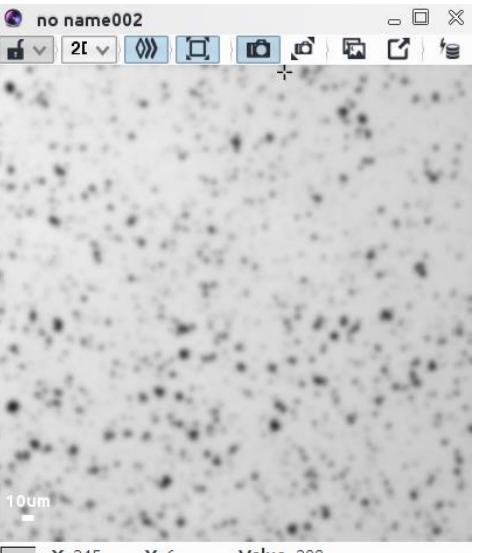
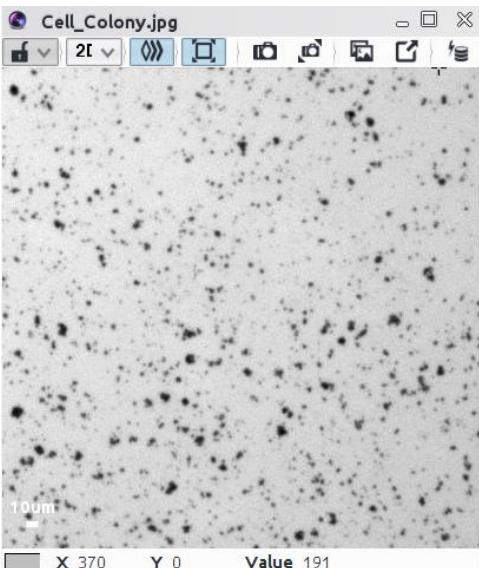
```
14 // process data
15 buffer3 = clij2.create([406, 408], clij2.Float);
16 clij2.gaussianBlur2D(buffer2, buffer3, 2.0, 2.0);
17
```

- Pull

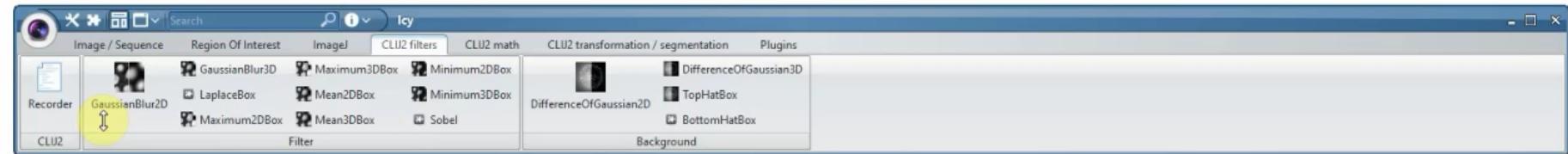
```
18 // pull data from GPU
19 sequence4 = clij2.pullSequence(buffer3);
20 Icy.addSequence(sequence4);
21
```

- Cleanup

```
22 // clean up memory
23 clij2.clear();
```

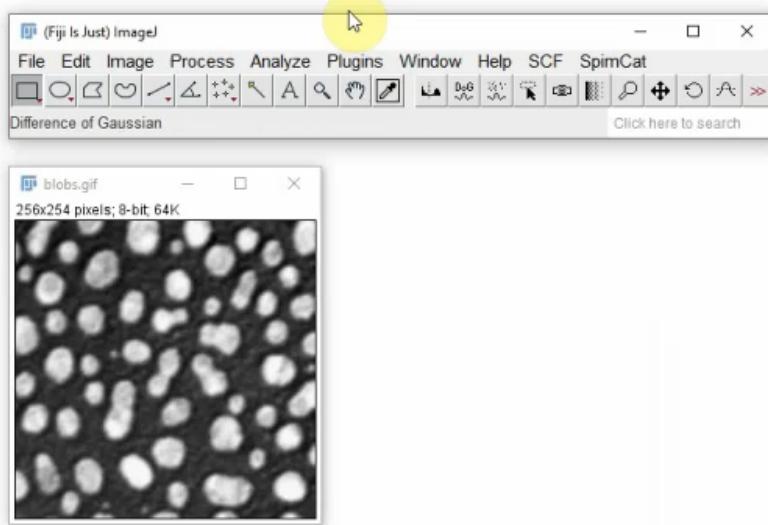


CLIJ2: use ICY javascript recorder!



ImageJ / Fiji

CLIJ2: Macro recording



CLIJ2: Macro recording



The screenshot shows the Fiji macro editor window titled '*Macro.ijm.ijm'. The script content is as follows:

```
0 sigma_x = 2.0;
1 sigma_y = 2.0;
2 Ext.CLIJ2_gaussianBlur2D(image1, image2, sigma_x, sigma_y);
3 Ext.CLIJ_pull(image2);

4 // threshold otsu
5 image3 = "threshold_otsu1213781482";
6 Ext.CLIJ2_thresholdOtsu(image2, image3);
7 Ext.CLIJ_pull(image3);

8 // connected components labeling
9 image4 = "connected_components_labeling117022162";
10 Ext.CLIJ2_connectedComponentsLabelingBox(image3, image4);
11 Ext.CLIJ_pull(image4);
12 run("glasbey on dark");

13 // exclude labels on edges
14 image5 = "exclude_labels_on_edges-2086602814";
15 Ext.CLIJ2_excludeLabelsOnEdges(image4, image5);
16 Ext.CLIJ_pull(image5);
17 run("glasbey on dark");
18
19
```

The bottom panel shows the macro status and history:

Run	Batch	Kill	<input type="checkbox"/> persistent	Show Errors	Clear	▶
Started Macro.ijm.ijm at Mon Apr 27 14:30:05 CEST 2020						
Started Macro.ijm.ijm at Mon Apr 27 14:30:12 CEST 2020						

CLIJ2: Macro editing



The screenshot shows the Fiji interface with several windows open:

- A top-level window titled "Fiji (Fiji Is Just) ImageJ" with a toolbar and menu bar.
- A preview window showing a grayscale image named "blobs.gif" with the dimensions "256x254 pixels; 8-bit; 64K".
- A macro editor window titled "*Macro.jm.jm" containing the following code:

```
1 run("Blobs (25K)");
2 run("CLIJ2 Macro Extensions", "cl_device=[GeForce RTX 2060 SUPER]");
3
4 // gaussian blur
5 image1 = "blobs.gif";
6 Ext.CLIJ_push(image1);
7 image2 = "gaussian_blur-1871941868";
8 sigma_x = 10.0;
9 sigma_y = 10.0;
10 Ext.CLIJ2_gaussianBlur2D(image1, image2, sigma_x, sigma_y);
11 Ext.CLIJ_pull(image2);
12
```

The code uses the CLIJ2 Macro Extensions to perform a Gaussian blur on the "blobs.gif" image, saving it as "gaussian_blur-1871941868".

Below the code editor is a toolbar with buttons for "Run", "Batch", "Kill", "persistent", "Show Errors", "Clear", and a play/pause button.

To the right of the macro editor, a portion of a web browser window is visible, showing the Google homepage and some search results.

Macro editing



- Auto-completion also works for Jython & friends.
- Start typing what you search e.g. “gauss” and not “clij”.
- Make use of code snippets

The screenshot shows a code editor window titled "first_script.py" containing Python code. The code uses the CLIJ2 API to load an image, initialize a GPU, push it to the GPU, process it with a Gaussian blur, and then compute the difference of Gaussian. The word "gauss" is being typed at the end of line 17, and the code completion dropdown shows "gaussianBlur" with its 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." Below the code editor is a terminal window showing the execution of the script multiple times.

```
# Load data
from ij import IJ
imp = IJ.openImage("http://imagej.nih.gov/ij/images/Cell_Colony.jpg")
imp.show()

# initialize GPU
from net.haesleinhuepf.clij2 import CLIJ2;
clij2 = CLIJ2.getInstance()

# push image to GPU
input_image = clij2.push(imp)

# process image
sigma = 5
blurred = clij2.create(input_image)
clij2.gaussianBlur2D(input_image, blurred, sigma, sigma)
gaus|clij2.differenceOfGaussian(ClearCLBuffer input, ClearCLBu
# clij2.differenceOfGaussian2D(ClearCLBuffer input, ClearCL
in|clij2.differenceOfGaussian3D(ClearCLBuffer input, ClearCL
21|clij2.gaussianBlur(ClearCLImageInterface source, ClearCL
22|# clij2.gaussianBlur2D(ClearCLImageInterface source, ClearCL
im|clij2.gaussianBlur2D(ClearCLImageInterface source, ClearCL
24|im|clij2.gaussianBlur3D(ClearCLImageInterface source, ClearCL
25|im|clijx.differenceOfGaussian(ClearCLBuffer input, ClearCLBu
26|# clijx.differenceOfGaussian2D(ClearCLBuffer input, ClearCL
27|clijx.differenceOfGaussian3D(ClearCLBuffer input, ClearCL
28|clijx.differenceOfGaussianInplace3D(ClearCLBuffer input, C
29|clijx.differenceOfGaussianInplace3D(ClearCLBuffer input, C
30|clijx.differenceOfGaussianInplace3D(ClearCLBuffer input, C

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

Programming language specific examples



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

The screenshot shows a web browser displaying the CLIJ2 documentation for the Gaussian Blur operation. The page is organized into several sections:

- Example notebooks:** Includes links for "basics", "blur", and "tribolium_morphometry".
- Example scripts:** Includes links for various IJM files: "backgroundSubtraction.ijm", "basics.ijm", "blur.ijm", "mean_squared_error.ijm", "meshTouchingNeighbors.ijm", "push_pull_selections.ijm", "push_pull_slices.ijm", "spot_distance_measurement.ijm", "tribolium_morphometry.ijm", "blurImage.m", "simplePipeline.m", "blur_frame_by_frame.js", "simplePipeline.js", and "backgroundSubtraction.py".
- Reference:** A sidebar on the right side of the browser window displays code examples for different languages:

 - Java:**

```
// init CLIJ and GPU
import net.haesleinhuepf.cluj2.CLIJ2;
import net.haesleinhuepf.cluj.clearCLbuffer;
CLIJ2 clij2 = CLIJ2.getInstance();
```
 - Matlab:**

```
% init CLIJ and GPU
clij2 = init_clatlab();
```
 - Icy JavaScript:**

```
// init CLIJ and GPU
importClass(net.haesleinhuepf.clicy.CLICY);
importClass(Packages.icy.main.Icy);

clij2 = CLICY.getInstance();

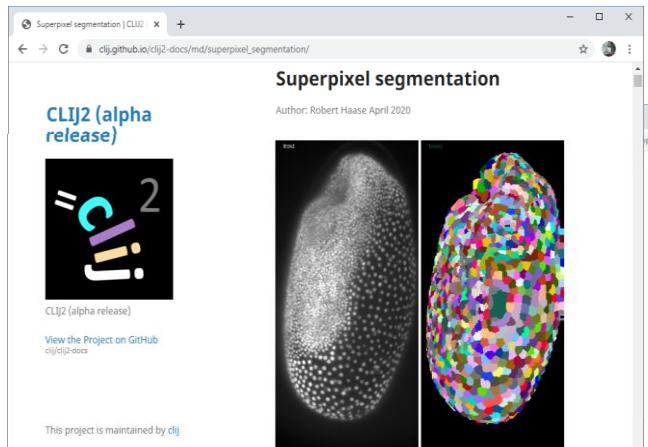
// get input parameters
source_sequence = getSequence();
source = clij2.pushSequence(source_sequence);
destination = clij2.create(source);
sigma_x = 1.0;
sigma_y = 2.0;

// Execute operation on GPU
clij2.gaussianBlur(source, destination, sigma_x, sigma_y)

// show result
destination_sequence = clij2.pullSequence(destination)
Icy.addSequence(destination_sequence);
```

Check out the tutorials

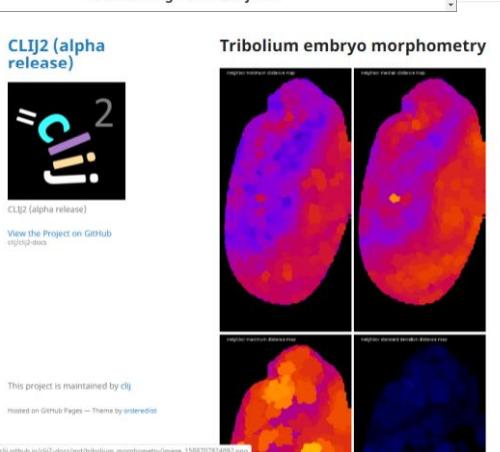
<https://clij.github.io/>



Superpixel segmentation
Author: Robert Haase April 2020
CLIJ2 (alpha release)
View the Project on GitHub
clij/clij2-docs

This project is maintained by clij
Hosted on GitHub Pages — Theme by orderedlist

Determine neighborhood relationships between segmented objects



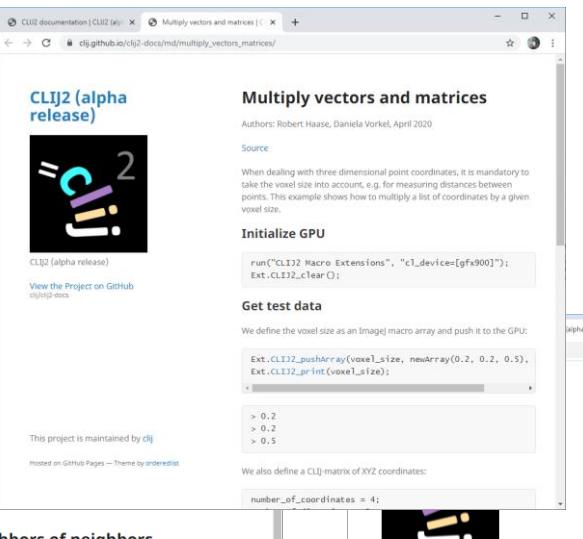
CLIJ2 (alpha release)
View the Project on GitHub
clij/clij2-docs

This project is maintained by clij
Hosted on GitHub Pages — Theme by orderedlist

https://github.com/CLIJ/clij2-docs/blob/main/tribolium_morphometry/image_1588707924902.png



Dani Vorkel
(Myers lab)
@happifocus



CLIJ2 (alpha release)
View the Project on GitHub
clij/clij2-docs

Multiply vectors and matrices
Authors: Robert Haase, Daniela Vorkel, April 2020
Source

When dealing with three dimensional point coordinates, it is mandatory to take the voxel size into account, e.g. for measuring distances between points. This example shows how to multiply a list of coordinates by a given voxel size.

Initialize GPU

```
run("CLIJ2 Macro Extensions", "cl_device=[gfx900]");
Ext.CLIJ2_clear();
```

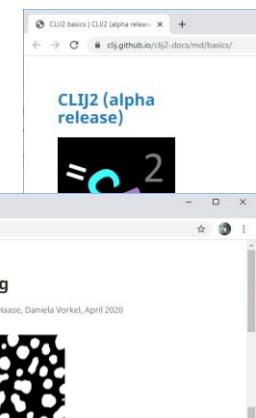
Get test data

We define the voxel size as an image macro array and push it to the GPU:

```
Ext.CLIJ2_pushArray(voxel_size, newArray(0.2, 0.2, 0.5),
Ext.CLIJ2_imIn(voxel_size);
```

> 0.2
> 0.2
> 0.5

This project is maintained by clij
Hosted on GitHub Pages — Theme by orderedlist



CLIJ2 basics
Authors: Robert Haase, Daniela Vorkel, April 2020
Source

This macro explains the basics for image processing on graphics processing units (GPUs) using CLIJ.

How to start

Before accessing CLIJ2 methods, every macro must contain following line to assign the GPU. By default, the parameter `cl_device` is left empty, but asking CLIJ to automatically select a GPU. In example, if you have a GPU named "Vendor Awesome Intelligent 3000", you can enter its full name behind the quality sign, written in brackets like `[Vendor Awesome Intelligent 3000]`. But also, you could just enter a part of the name, such as `wendor` or `some`. CLIJ will select the associated GPU as long as there is no other GPU containing the same parts of the name.

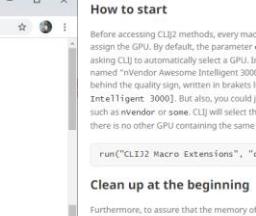
```
run("CLIJ2 Macro Extensions", "cl_device");
```



Labeling
Authors: Robert Haase, Daniela Vorkel, April 2020
Source

We also define a CLIJ-matrix of XYZ coordinates:

```
number_of_coordinates = 4;
```



Label connected components
Authors: Robert Haase, Daniela Vorkel, April 2020
Source

We can apply smoothing operations, e.g. the mean filter:

```
// determine the mean value of each neighboring nodes
Ext.CLIJ2_meanOfTouchingNeighbors(measurement, touch_matrix,
drawResult(label_map, mean_measurement);
```


Working with ROIs
Authors: Robert Haase, Daniela Vorkel, April 2020
Source

At the end of the macro, clean up:

```
Ext.CLIJ2_clear();
```

This is just convenient method to show images properly in the notebook:

```
function show() {
    run("Flatten");
}
```


Binary image operations
Authors: Robert Haase, Daniela Vorkel, April 2020
Source

This macro shows how to deal with operations on binary images, e.g. thresholding, dilation, erosion, fill holes, in the GPU.

All demonstrated operations work in 2D and 3D. For demonstration purpose, we use a 2D example.

```
// clean up first
run("Close All");
```


Get test data
Authors: Robert Haase, Daniela Vorkel, April 2020
Source

This project is maintained by clij
Hosted on GitHub Pages — Theme by orderedlist

```
run("Embryos (42k)");
run("8-bit");
makeRectangle(714, 14, 768, 394);
run("Crop", "=");
input = getTitle();
```


Trilateration (CLIJ and pushPoints to CLIJ interface)

Cheat sheets



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

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

Operation	Parameters	Result	Dim	Examples
Initialize CLIJ	[], HD, GFX or CPU			Ext.CLIJ2_init([], "CPU");
Push			2D 3D	Ext.CLIJ2_push(result, input);
Pull			2D 3D	Ext.CLIJ2_pull(result, input);
Create	1024, 1024, 8		2D 3D	Ext.CLIJ2_create(result, width, height, depth);
Convert			2D 3D	Ext.CLIJ2_convert(result, input, type);
Copy			2D 3D	Ext.CLIJ2_copy(result, input);
Copy slice	, 50		2D 3D	Ext.CLIJ2_copySlice(result, input, slice);
Crop	, 20, 20		2D 3D	Ext.CLIJ2_crop(result, input, x, y, width, height);
Paste			2D 3D	Ext.CLIJ2_paste(result, input, x, y);
Release				Ext.CLIJ2_release(result);
Clear				Ext.CLIJ2_clear();

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

Operation	Parameters	Result	Dim	Examples
Gaussian blur			2D 3D	Ext.CLIJ2_gaussianBlur2D(input, result, sigmaX, sigmaY);
Difference of Gaussian			2D 3D	Ext.CLIJ2_differenceOfGaussian2D(input, result, sigmaX, sigmaY, sigmaX2, sigmaY2);
Invert			2D 3D	Ext.CLIJ2_invert(input, result);
Laplace			2D 3D	Ext.CLIJ2_laplaceBox(input, result);
Mean			2D 3D	Ext.CLIJ2_mean2DBox(input, result, radiusX, radiusY);
Median			2D 3D	Ext.CLIJ2_medianSliceBySliceBox(input, result, radiusX, radiusY);
Minimum			2D	Ext.CLIJ2_minimum2DBox(input, result, radiusX, radiusY);

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

Operation	Parameters	Result	Dim	Examples
Ext.CLIJ2_meanOfTouchingNeighbors			2D 3D	Ext.CLIJ2_meanOfTouchingNeighbors(result, touchMatrix, meanValue);
Ext.CLIJ2_touchMatrixToMesh			2D 3D	Ext.CLIJ2_touchMatrixToMesh(pointList, touchMatrix, mesh);
Ext.CLIJ2_distanceMatrixToMesh			2D 3D	Ext.CLIJ2_distanceMatrixToMesh(pointList, distanceMatrix, mesh);
Ext.CLIJ2_meanValue			2D 3D	Ext.CLIJ2_meanValue(result, meanValue);
Ext.CLIJ2_countTouchingNeighbors			2D 3D	Ext.CLIJ2_countTouchingNeighbors(touchMatrix, countVector);
Ext.CLIJ2_statisticsOfBackgroundAndLabelledPixels			2D 3D	Ext.CLIJ2_statisticsOfBackgroundAndLabelledPixels(image, labelMap); Ext.CLIJ2_statisticsOfLabelledPixels(input, labelMap);
Ext.CLIJ2_pushResultsTable			2D 3D	Ext.CLIJ2_pushResultsTable(imageName);
Ext.CLIJ2_pushResultsColumn			2D 3D	Ext.CLIJ2_pushResultsTableColumn(imageName, columnName);
Ext.CLIJ2_pullToResultsTable			2D 3D	Ext.CLIJ2_pullToResultsTable(imageName);
Ext.CLIJ2_pushFromArray			2D 3D	Ext.CLIJ2_pushFromArray(imageName, array, width, height, depth);

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

Result	Dim	Examples
	2D 3D	Ext.CLIJ2_epicToPointList(binaryImage, pointList); Ext.CLIJ2_labelledPointToPointList(labelledImage, pointList);
	2D 3D	Ext.CLIJ2_generateDistanceMatrix(pointList, pointList);
	2D 3D	Ext.CLIJ2_generateTouchMatrix(labelMap, touchMatrix);
	2D 3D	Ext.CLIJ2_touchMatrixToMesh(pointList, touchMatrix, mesh);
	2D 3D	Ext.CLIJ2_meanOfTouchingNeighbors(result, meanValue);
	2D 3D	Ext.CLIJ2_meanValue(result, meanValue);
	2D 3D	Ext.CLIJ2_countTouchingNeighbors(touchMatrix, countVector);
	2D 3D	Ext.CLIJ2_statisticsOfBackgroundAndLabelledPixels(image, labelMap); Ext.CLIJ2_statisticsOfLabelledPixels(input, labelMap);
	2D 3D	Ext.CLIJ2_pushResultsTable(imageName);
	2D 3D	Ext.CLIJ2_pushResultsTableColumn(imageName, columnName);
	2D 3D	Ext.CLIJ2_pullToResultsTable(imageName);
	2D 3D	Ext.CLIJ2_pushFromArray(imageName, array, width, height, depth);

<https://clij.github.io/clij2-docs> [@haesleinhuepf](https://twitter.com/haesleinhuepf)

https://clij.github.io/clij2-docs/CLIJ2-cheatsheet_V3.pdf

39

CLIJ2: Out of memory?



- Two options:
 - Buy more memory. This means, buy a more expensive graphics card.
 - Change your workflow (release memory! Downsample!)

2010:

2020:

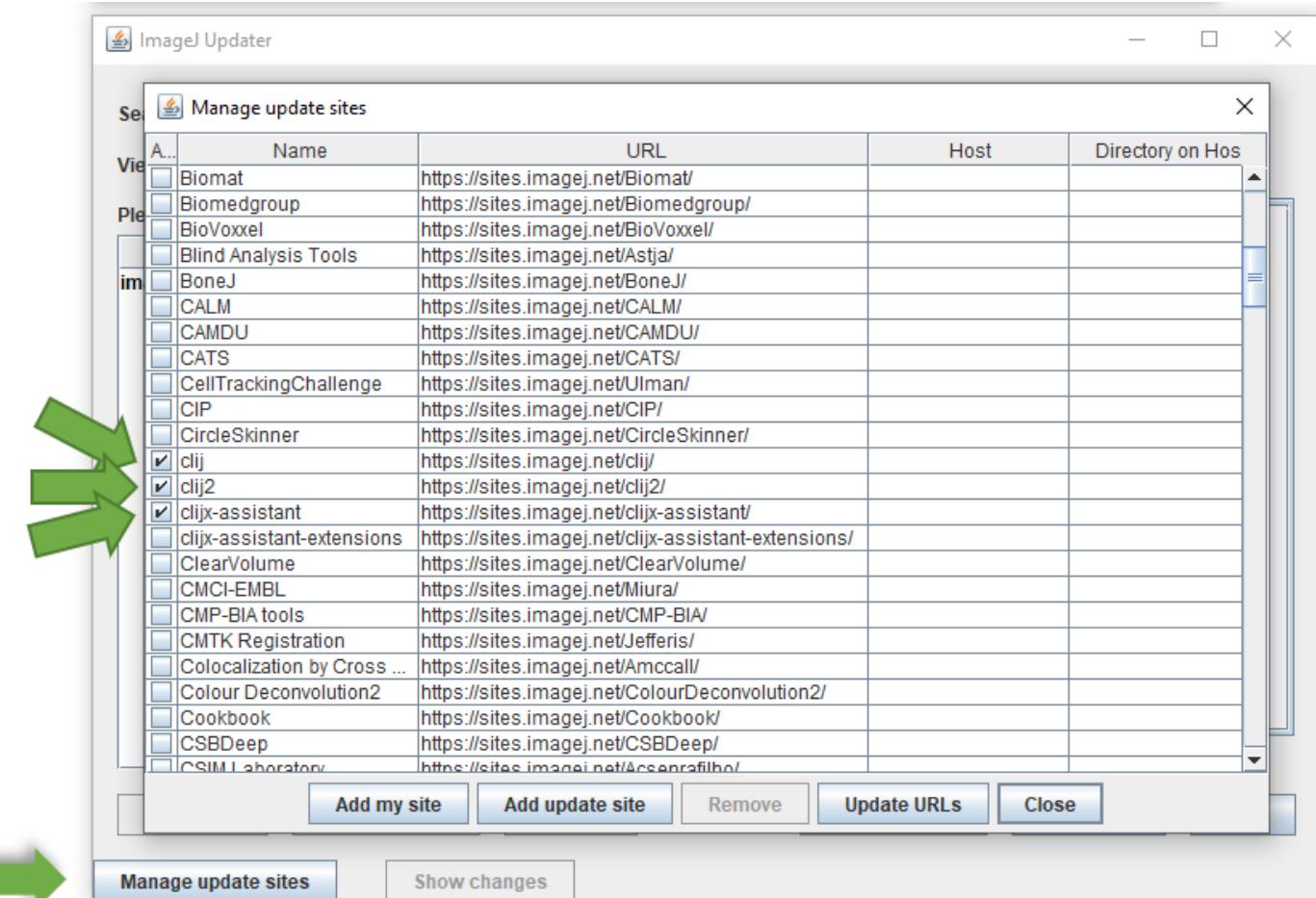
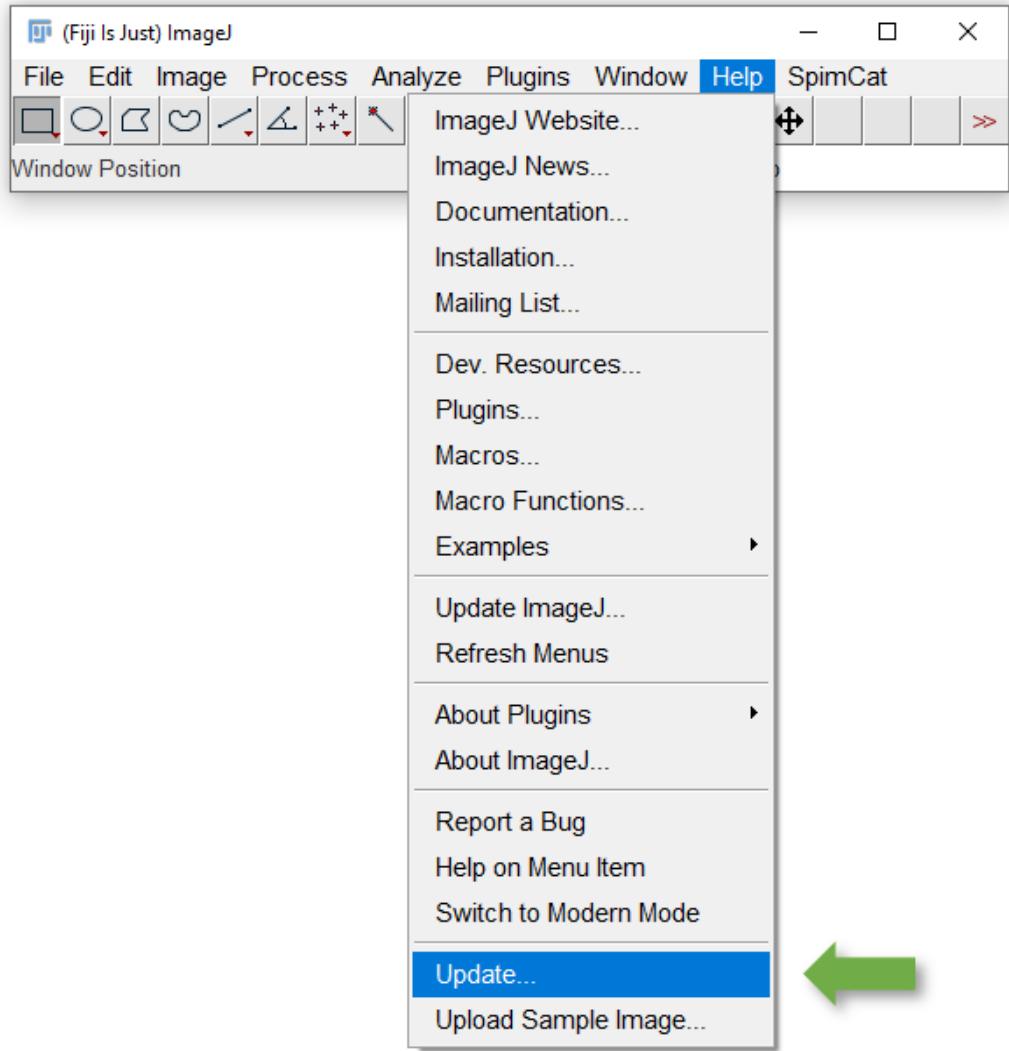
```
Clearing
net.haesleinhuepf.clij.clearcl.exceptions.OpenCLEException: OpenCL error: -4 -> CL_MEM_OBJECT_ALLOCATION_FAILURE
    at net.haesleinhuepf.clij.clearcl.backend.BackendUtils.checkOpenCLError(BackendUtils.java:346)
    at net.haesleinhuepf.clij.clearcl.backend.jocl.ClearCLBackendJOCL.lambda$enqueueKernelExecution$23(ClearCLBackendJOCL.java:171)
    at net.haesleinhuepf.clij.clearcl.backend.BackendUtils.checkExceptions(BackendUtils.java:171)
    at net.haesleinhuepf.clij.clearcl.backend.jocl.ClearCLBackendJOCL.enqueueKernelExecution(ClearCLBackendJOCL.java:171)
    at net.haesleinhuepf.clij.clearcl.ClearCLKernel.lambda$run$0(ClearCLKernel.java:489)
    at net.haesleinhuepf.clij.clearcl.util.ElapsedTime.measure(ElapsedTime.java:97)
    at net.haesleinhuepf.clij.clearcl.util.ElapsedTime.measure(ElapsedTime.java:64)
    at net.haesleinhuepf.clij.clearcl.ClearCLKernel.run(ClearCLKernel.java:477)
    at net.haesleinhuepf.clij.clearcl.ClearCLKernel.run(ClearCLKernel.java:459)
    at net.haesleinhuepf.clij.clearcl.util.CLKernelExecutor.lambda$enqueue$1(CLKernelExecutor.java:267)
    at net.haesleinhuepf.clij.clearcl.util.ElapsedTime.measure(ElapsedTime.java:97)
    at net.haesleinhuepf.clij.clearcl.util.ElapsedTime.measure(ElapsedTime.java:28)
    at net.haesleinhuepf.clij.clearcl.util.CLKernelExecutor.enqueue(CLKernelExecutor.java:266)
    at net.haesleinhuepf.clij2.CLIJ2.lambda$executeSubsequently$0(CLIJ2.java:406)
    at net.haesleinhuepf.clij.util.ElapsedTime.measure(ElapsedTime.java:97)
    at net.haesleinhuepf.clij.util.ElapsedTime.measure(ElapsedTime.java:28)
    at net.haesleinhuepf.clij2.CLIJ2.executeSubsequently(CLIJ2.java:397)
    at net.haesleinhuepf.clij2.CLIJ2.executeSubsequently(CLIJ2.java:384)
    at net.haesleinhuepf.clij2.CLIJ2.executeSubsequently(CLIJ2.java:379)
    at net.haesleinhuepf.clij2.CLIJ2.execute(CLIJ2.java:366)
    at net.haesleinhuepf.clij2.utilities.CLIJUtilities.tableKernel(CLIJUtilities.java:98)
    at net.haesleinhuepf.clij2.plugins.GaussianBlur3D.(GaussianBlur3D.java:62)
    at net.haesleinhuepf.clij2.plugins.GaussianBlur3D.gaussianBlur3D(GaussianBlur3D.java:58)
```

Exercises

Exercise 0: Installing clij, clij2 and the clijx-assistant



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



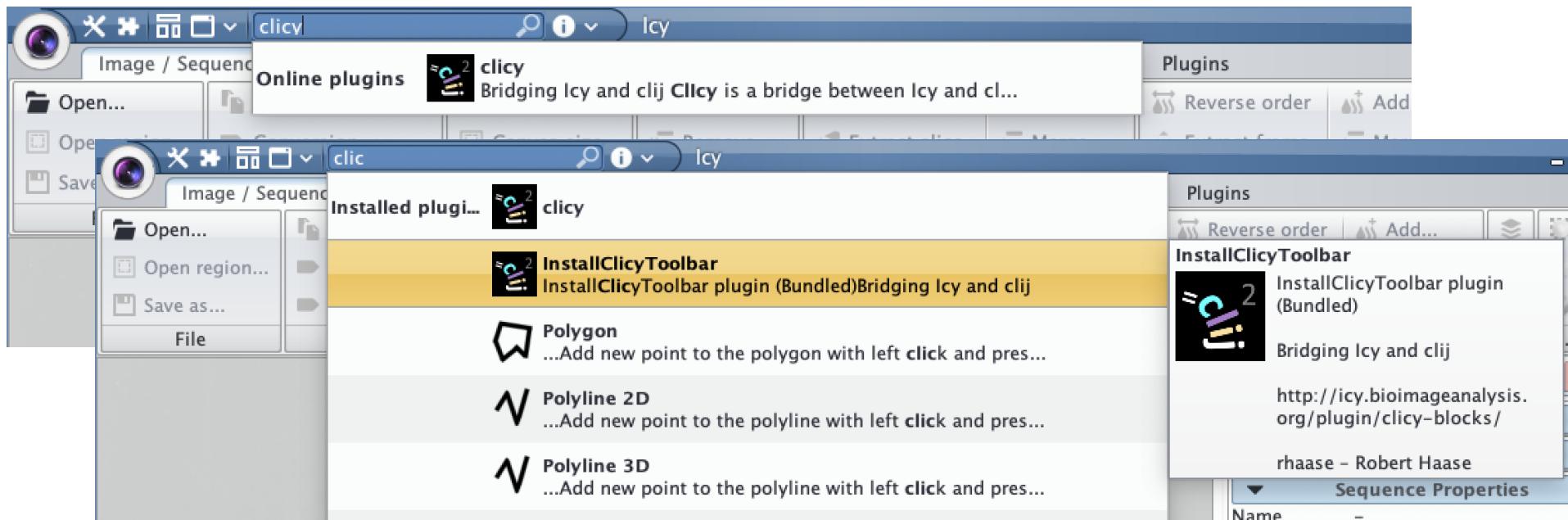
Exercise 0: Installing ICY / CLIcy



Download and unzip ICY from : <http://icy.bioimageanalysis.org> (Java required)

Search and select “CLIcy” in the search bar to download and Install it

Finally run CLIcy Toolbar Installer (hit: use the search again!)



Restart ICY and have fun with the new possibilities!

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 file `benchmarking.ijm` open. The left sidebar lists other macros in the `macro` folder. The main editor area contains the following code:

```
54 // We construct the taken time for push() and pace() commands.
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 */
```

The bottom of the editor has buttons for **Run**, **Batch**, **Kill**, **persistent**, **Show Errors**, and **Clear**.

To the right is a `Log` window showing the output of the macro execution:

```
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?

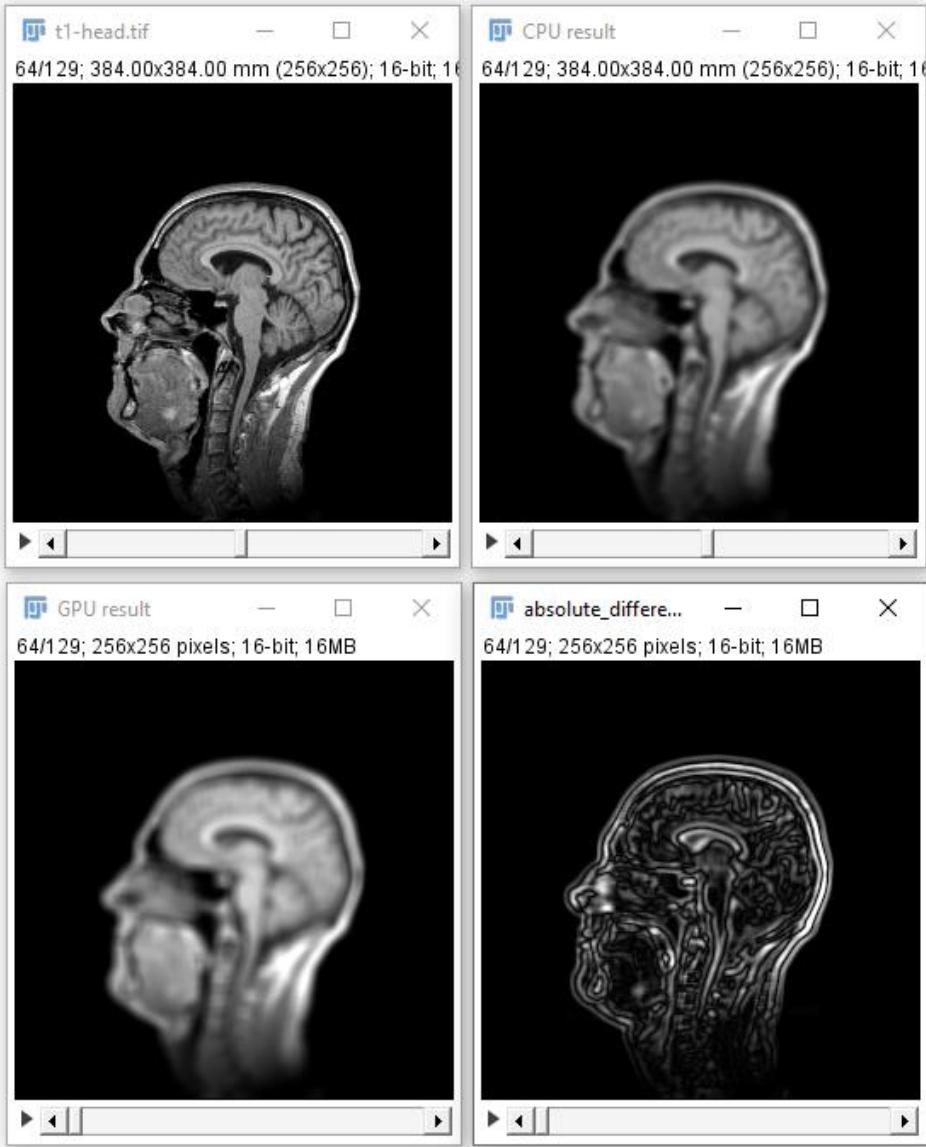
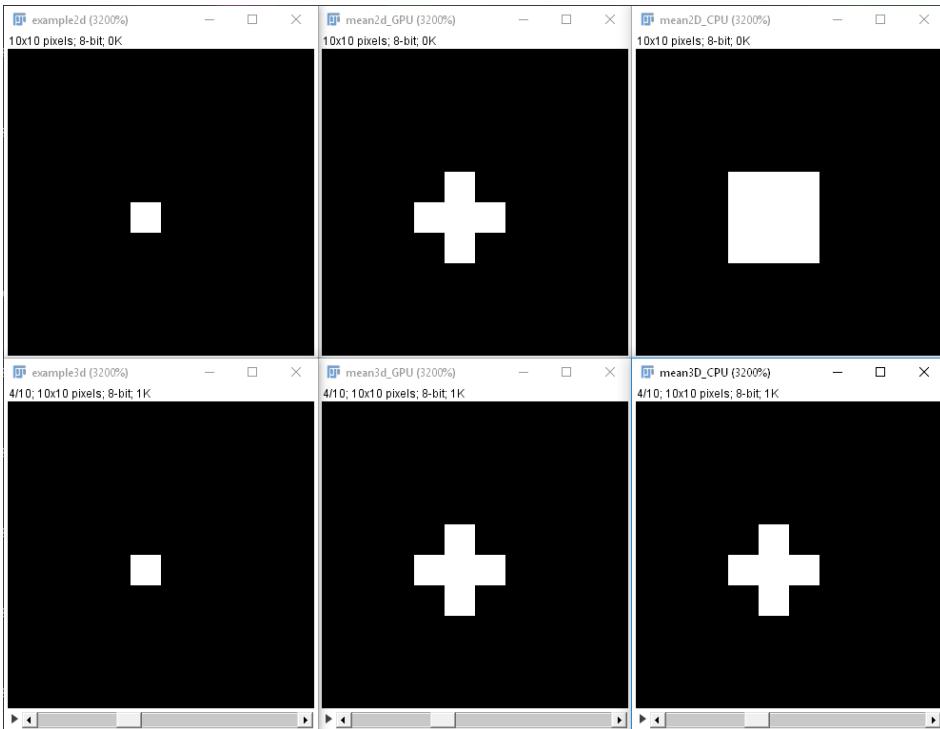
The screenshot shows a window titled "Log" with a menu bar for File, Edit, and Font. The main content area displays the output of the clInfo.ijm script. The output includes information about available CL backends, the best backend, used CL backend, ClearCL base, number of platforms, devices, and extensions. 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@36785da1  
ClearCL: ClearCLBase [mClearCLBackendInterface=net.haesleinhuepf.clj.clearcl.backend.jocl.  
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
```

Exercise 3: Quantitative comparison



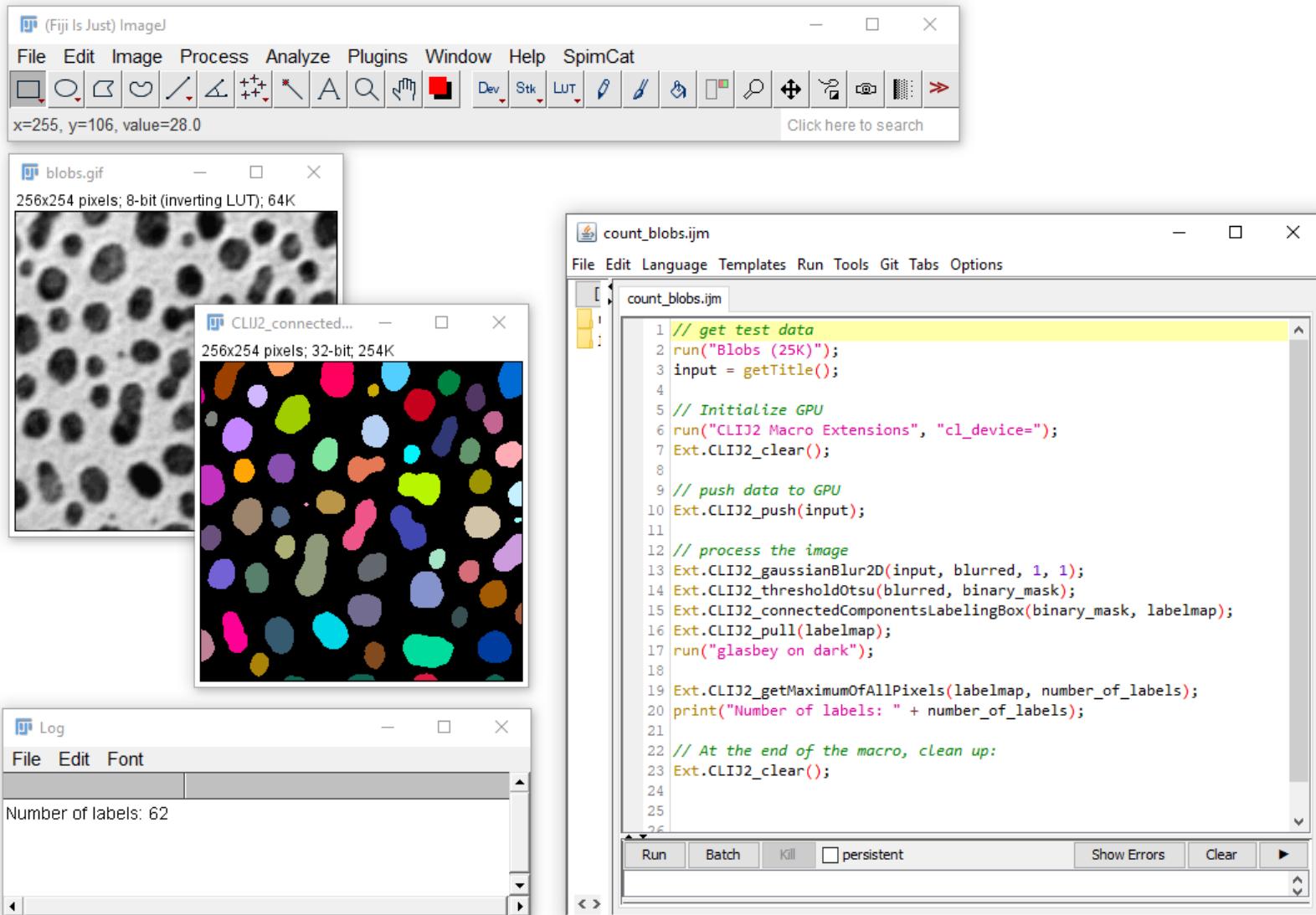
- Compare the mean filters in ImageJ and CLIJ when processing the t1-head example dataset. Compute an image showing the difference or the absolute difference between both results.
- Try mean3DBox and mean3DSphere. Which one is closer to ImageJ's 'Mean 3D' filter?
- Also run [mean detailed comparison IJ CLIJ.ijm](#). What can you learn from it?



Exercise 4: Segmentation

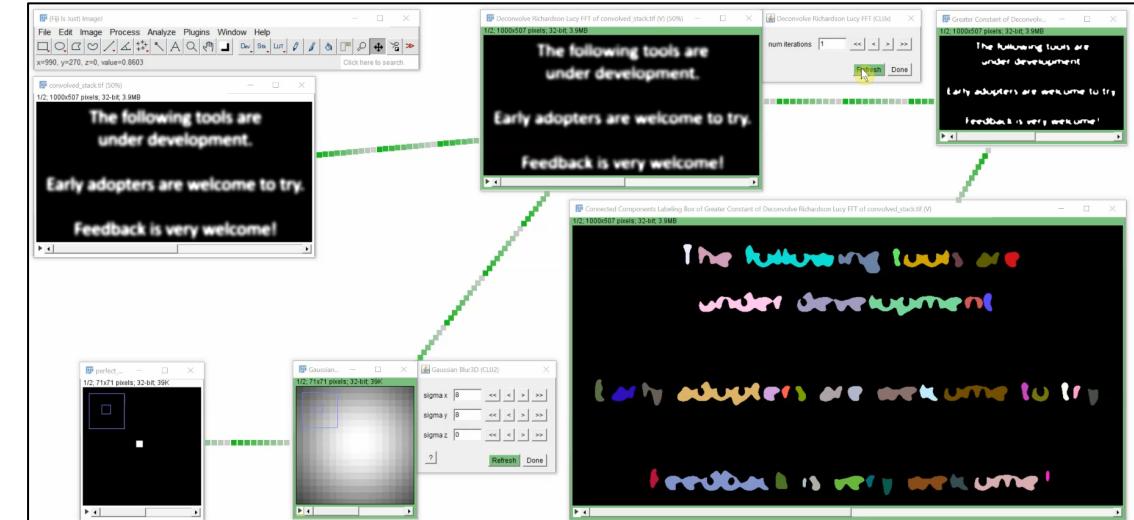


- Setup a workflow for counting blobs in ImageJ's blobs.gif example image. Use these functions:
 - Gaussian Blur
 - Otsu's threshold method
 - Connected Component Labeling
 - Measure maximum intensity in the resulting label image
- In order to achieve this, it is recommended to use the Macro recorder or auto-completion in the script editor.
- Optional: Write this script in Fijis Jython or Icy JavaScript. If you use Icy, turn on CLIJs recorder and execute the operations from the menu.



Pause 15 min

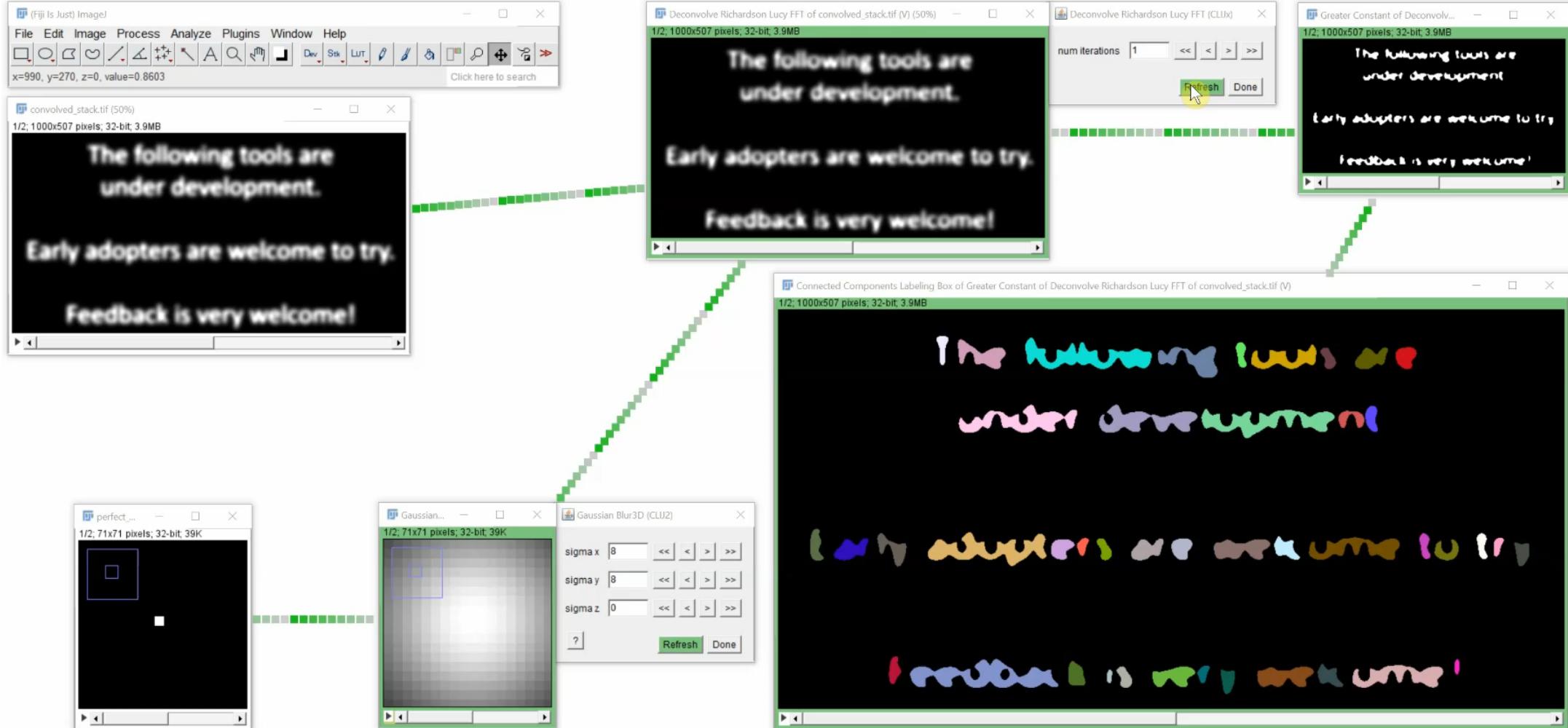
Coming up next:
Workflow design



Part II

Interactive Image Data Flow Graph design and code-generation

Image Data Flow Graphs & cI Esperanto



Special thanks to Brian Northon! @truenorth_ia

https://en.wikipedia.org/wiki/Release_early,_release_often

Image Data Flow Graphs & cI Esperanto



- Use the magic tool button and the magic mouse.

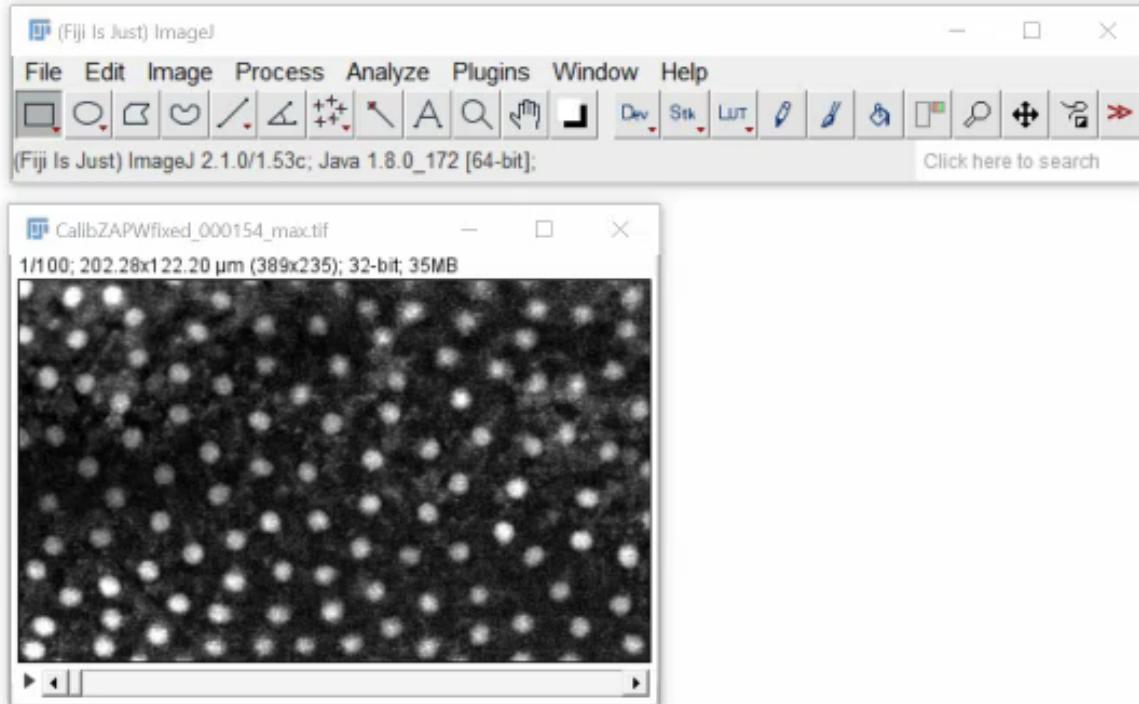
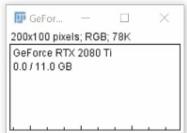
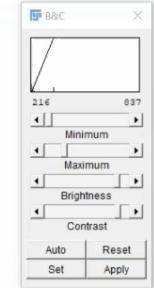
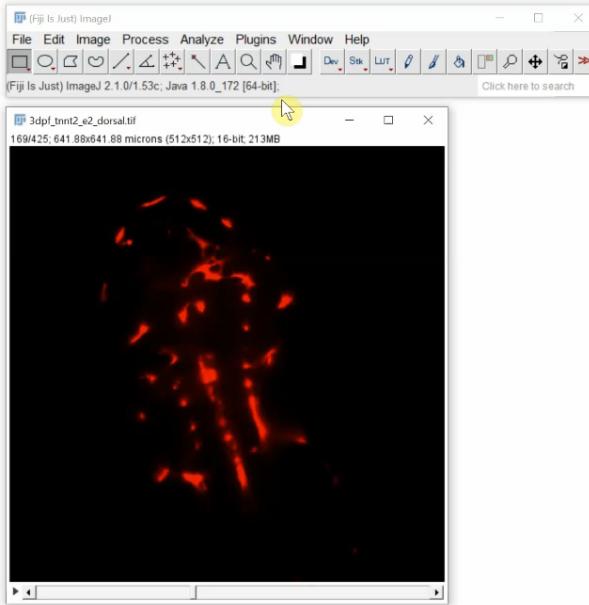


Image Data Flow Graphs & cEsperanto



- Windows are connected and image data flows from the input image downstream.



Special thanks to Elisabeth Kugler! @KuglerElisabeth

@haesleinhuepf

@StRigaud

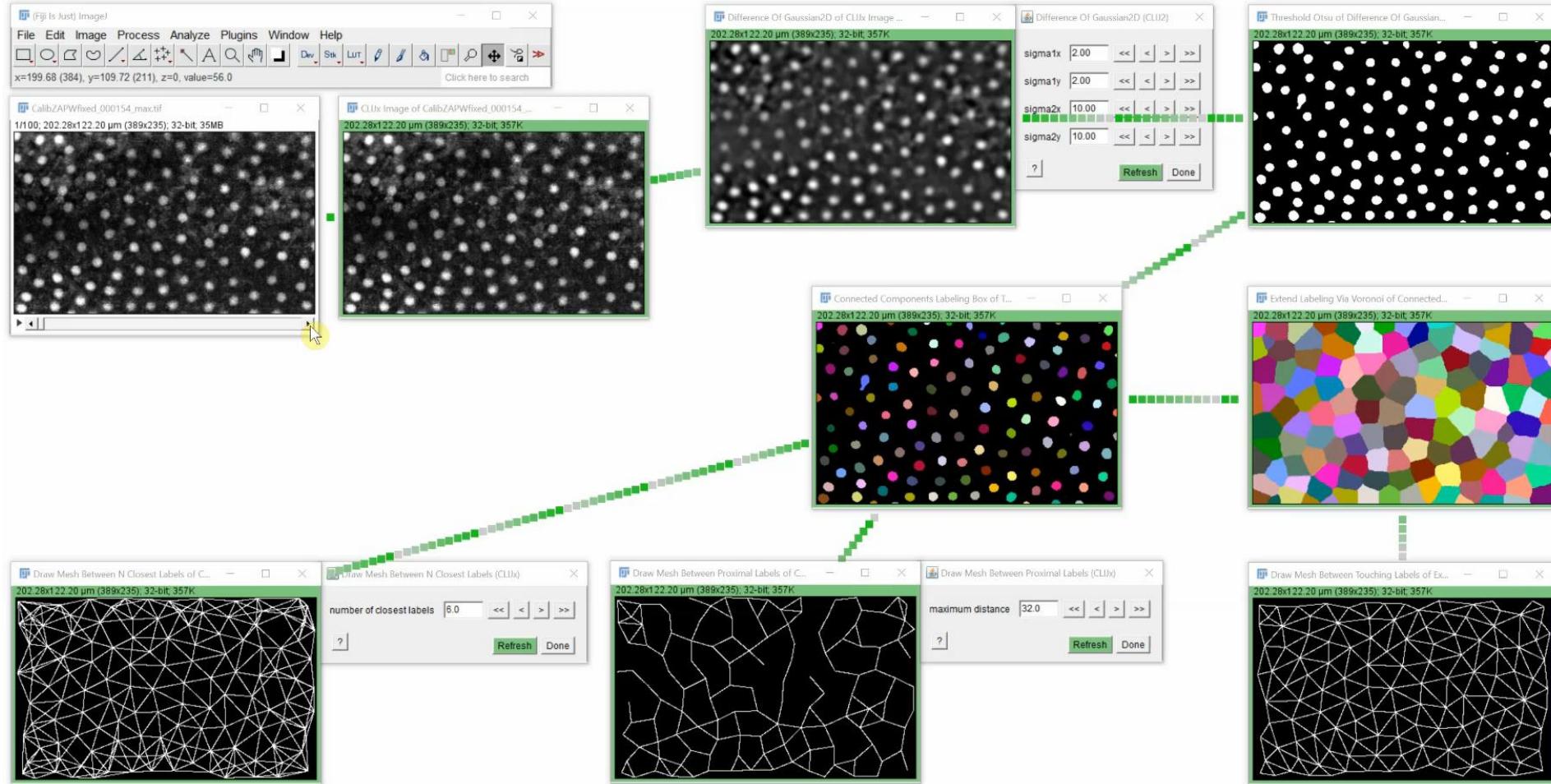
<https://clij.github.io/>

Image data source: Elisabeth Kugler; labs of Tim Chico and Paul Armitage, The University of Sheffield (UK) " <https://zenodo.org/record/4204839#.X8DCRGj7Q2w>

Image Data Flow Graphs & cIJ Esperanto



- Study relationships between operations and compare different strategies



Draw mesh between N closest
neighbors

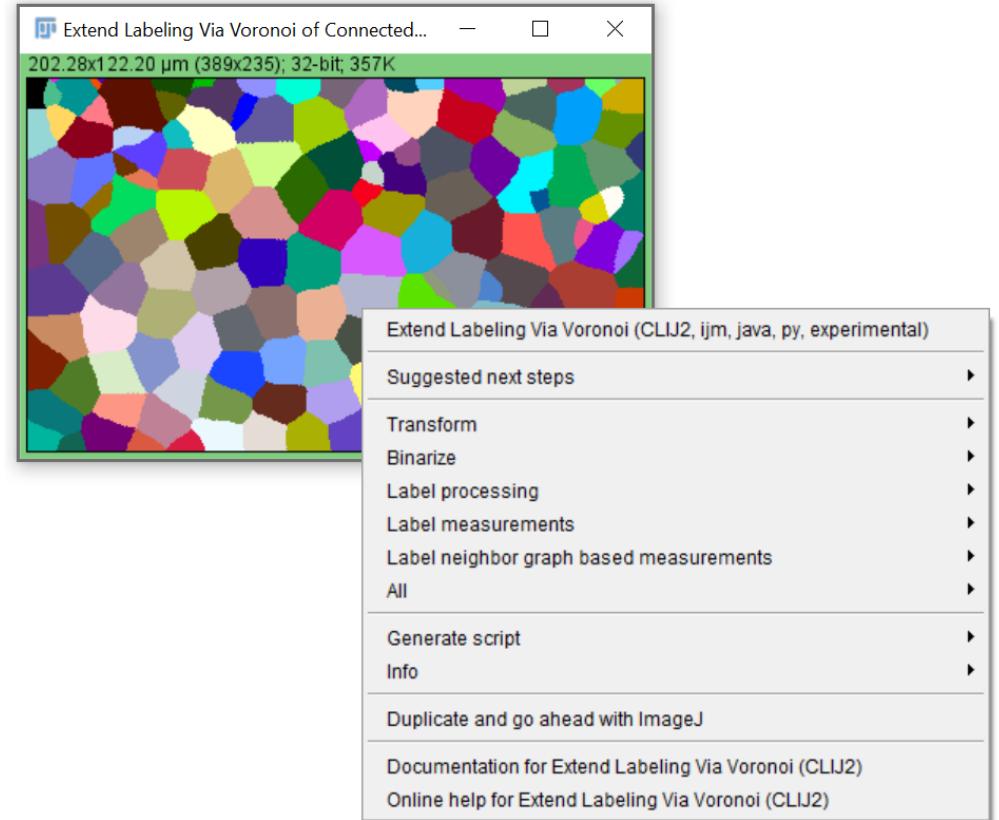
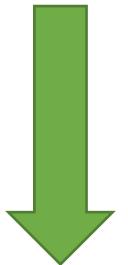
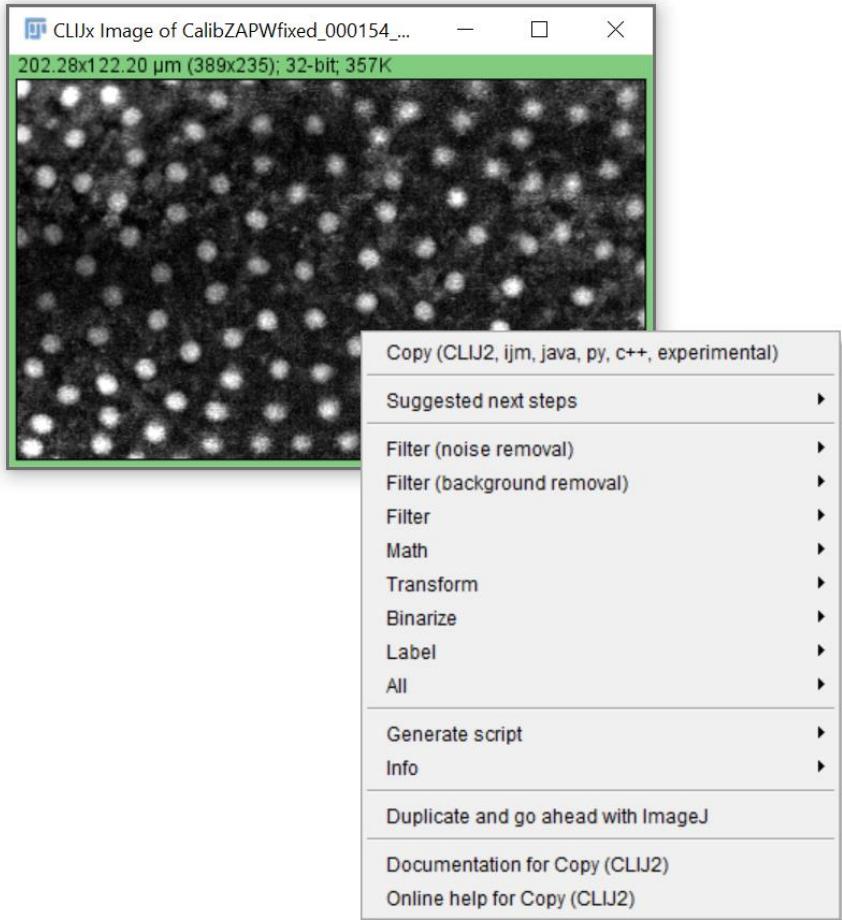
Draw mesh between neighbors
closer than d pixels

Draw mesh between touching
neighbors

Image Data Flow Graph design



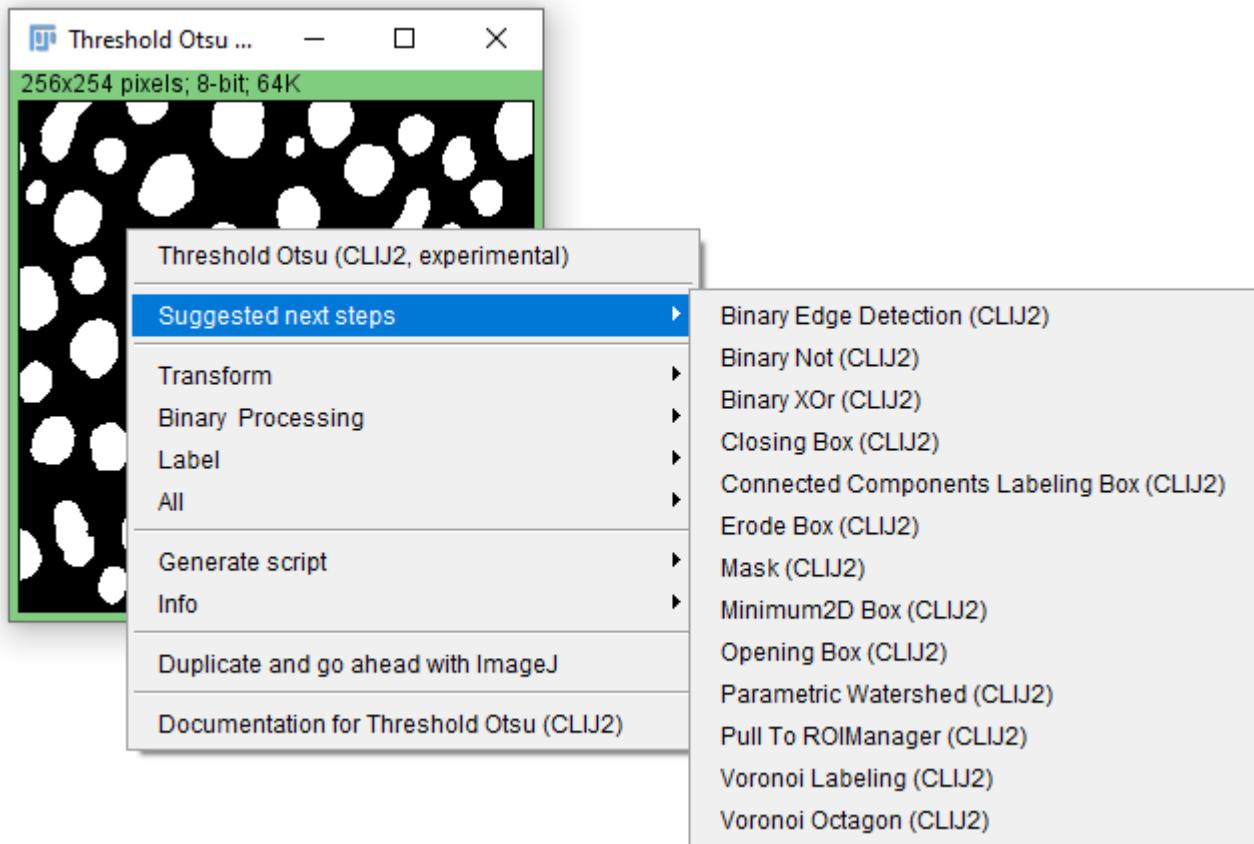
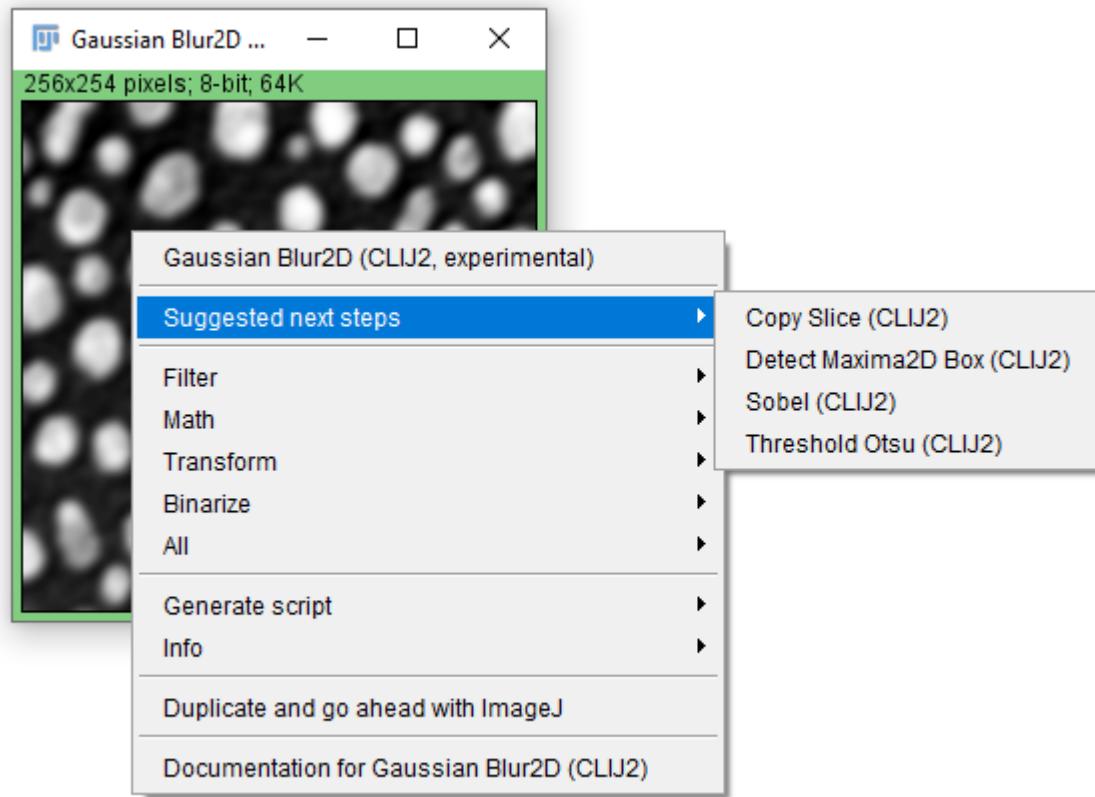
- The menu order is intentional: From preprocessing to analysis



Expert system: Suggestions



- Explore suggestions!

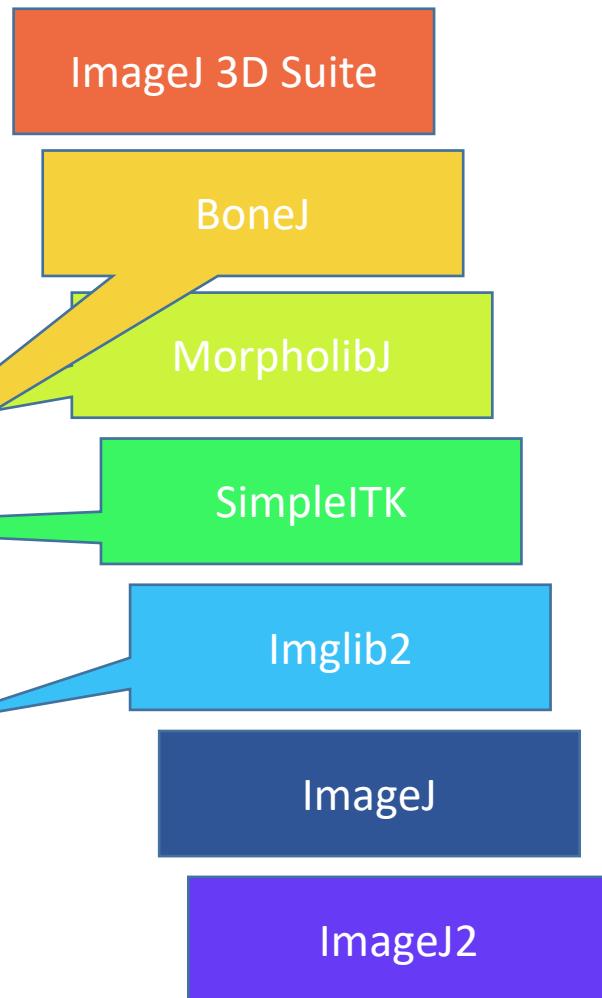
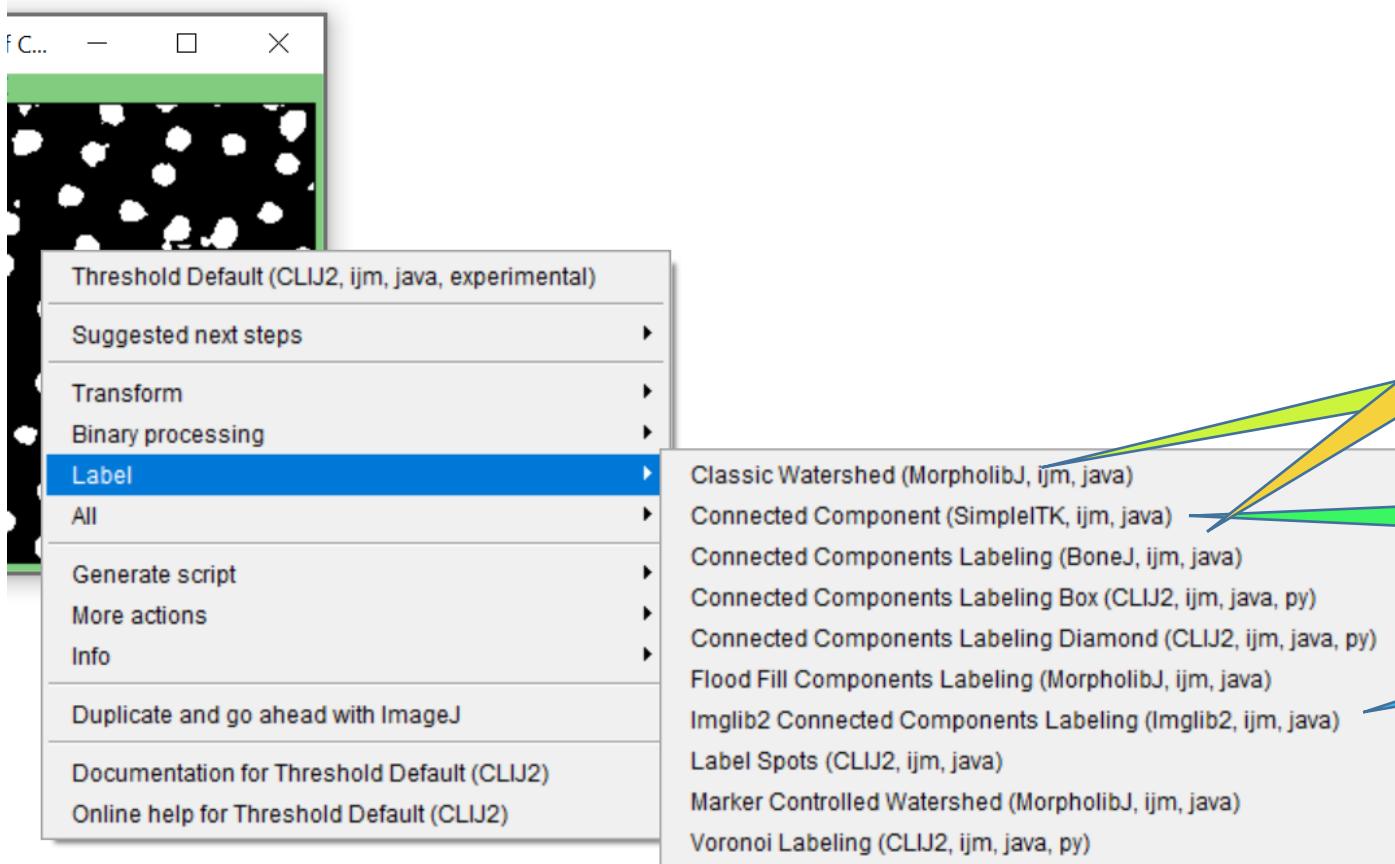


clEsperanto: Extensibility



- Watch out for other image processing libraries.

- Install: <https://clij.github.io/assistant/installation#extensions>



Fiji search bar



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

The screenshot shows the Fiji application window with the title '(Fiji Is Just) ImageJ'. The menu bar includes File, Edit, Image, Process, Analyze, Plugins, Window, and Help. Below the menu is a toolbar with various icons. A search bar at the bottom contains the text 'convolve'. A large green arrow points from the left towards a search results window.

Quick Search

Commands

- Convolve (3D) / Plugins/Process/Convolve (3D)
- Convolve an image with another image on GPU / Plugins/Process/Convolve on GPU
- Convolve on GPU / Plugins/ImageJ on GPU (CLIJ)
- Convolve... / Process/Filters/Convolve...
- Deconvolve (Richardson-Lucy) on GPU / Plugins/Fuse/Deconvolve Dataset / Plugins/Multiview Re

Ops

Script templates

ImageJ Wiki

Classes

CLIJx-Assistant

- Deconvolve Richardson Lucy FFT(CLIJx, ijm, jav)
- Convolve FFT(CLIJx, ijm, java)
- Richardson Lucy Deconvolution(ImageJ2, ijm)
- Convolve(CLIJ2, ijm, java, py)**

Image.sc Forum

Convolve(CLIJ2, ijm, java, py)

Convolve the image with a given kernel image.

It is recommended that the kernel image has an odd size in X, Y and Z.

clEsperanto compatibility

ijm, java, py

available_for

2D, 3D

jar

jar:file:/C:/programs/FIJI-W~1/Fiji.app/plugins/clij2_-2.2.0.11.

jar

parameters

Image source, Image convolution_kernel, ByRef Image destination

class

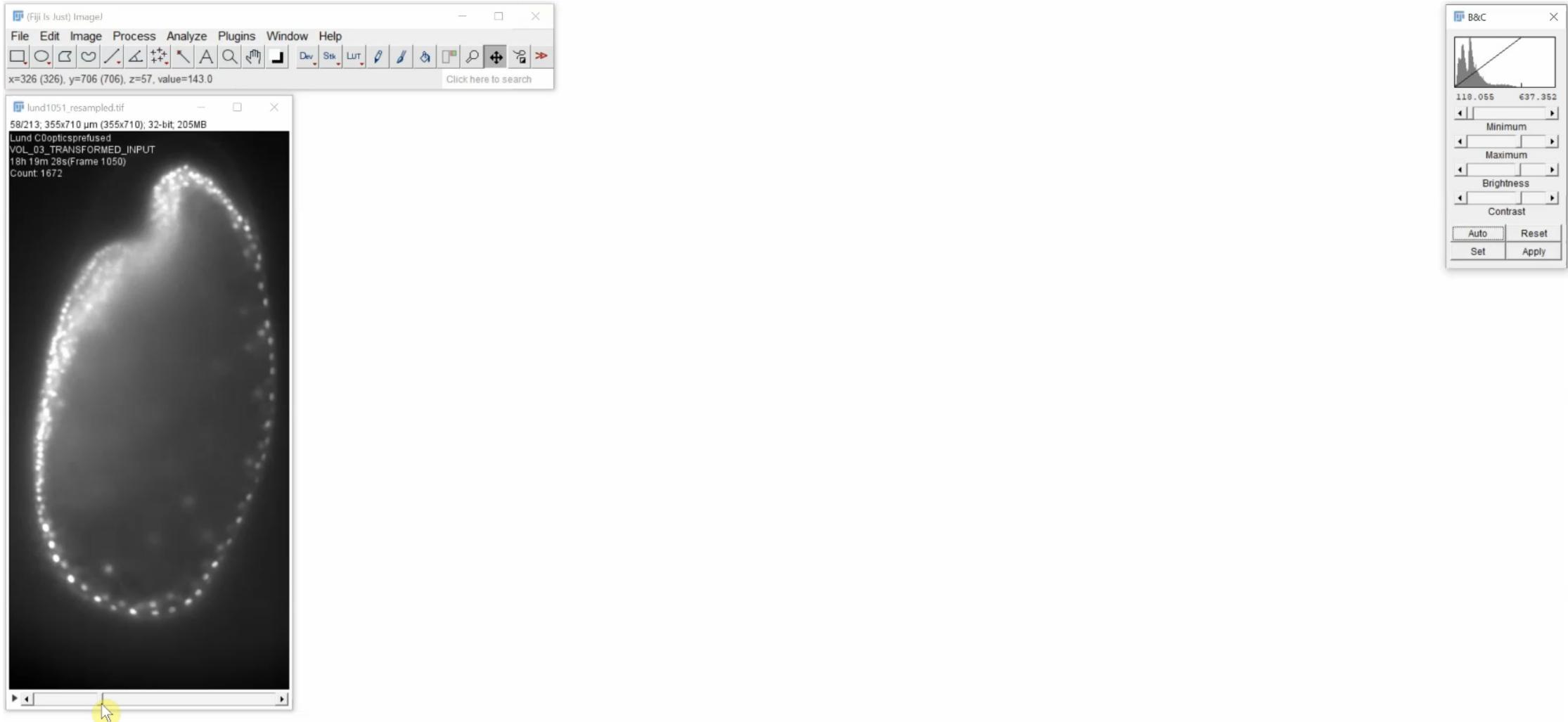
class net.haesleinhuepf.clij2.plugins.Convolve

Run convolve

Image Data Flow Graphs



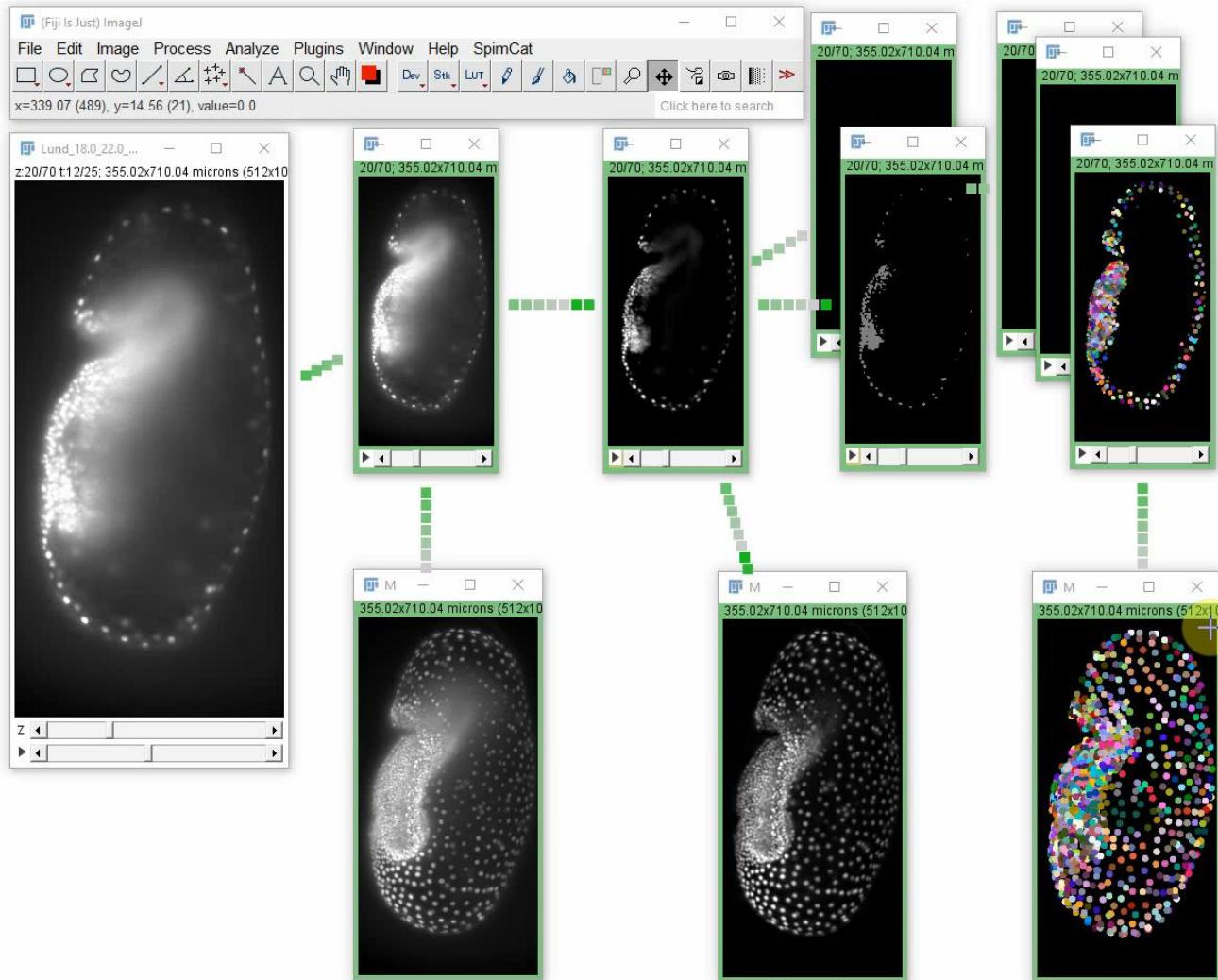
- Don't let the windows rule your desktop. Rule the windows!



Export to scripts



- Export to multiple programming languages*



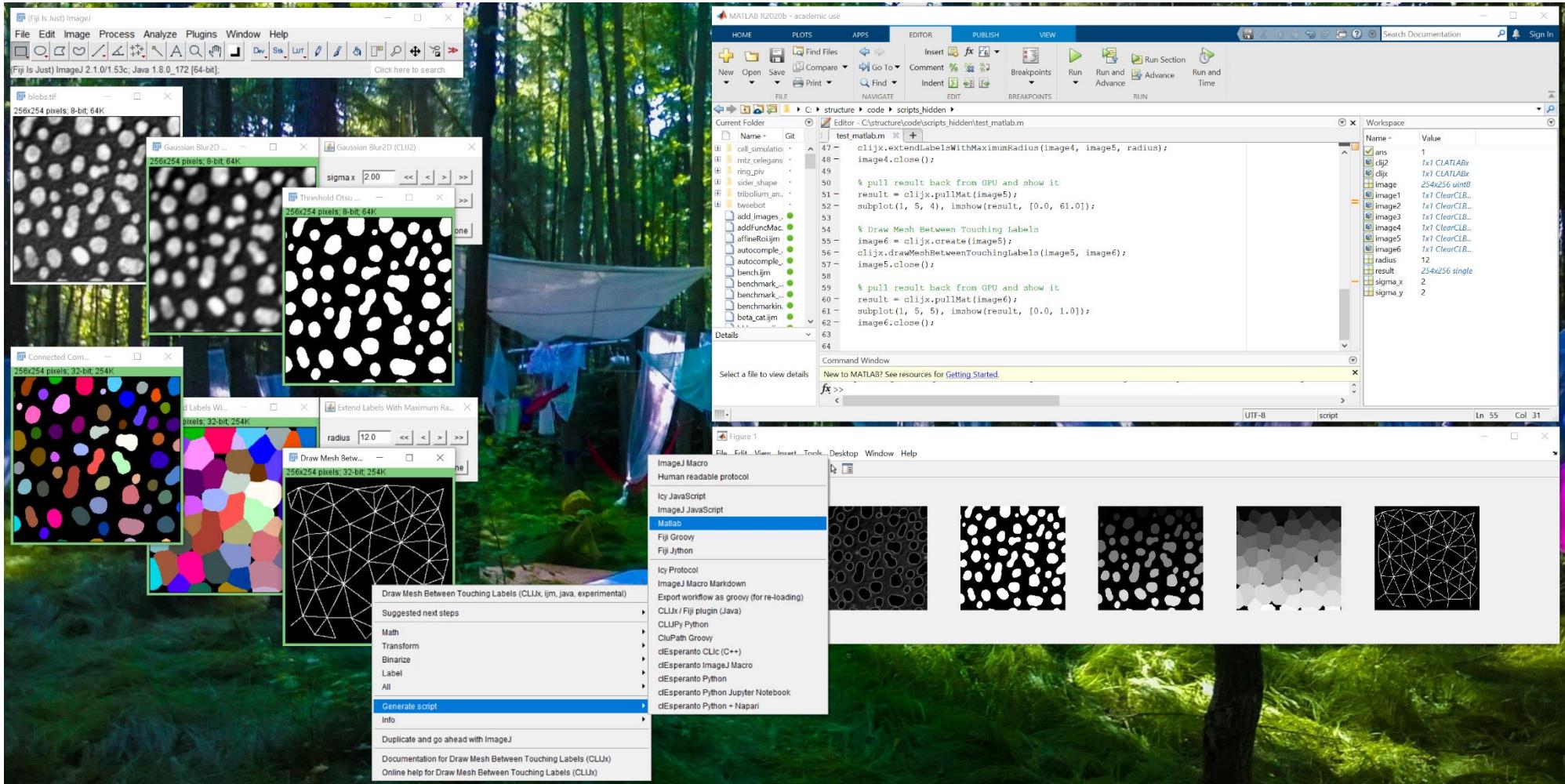
* Icy Protocols, QuPath Groovy, Python, C++ ready for testing / early adopters

clEsperanto: clatlab

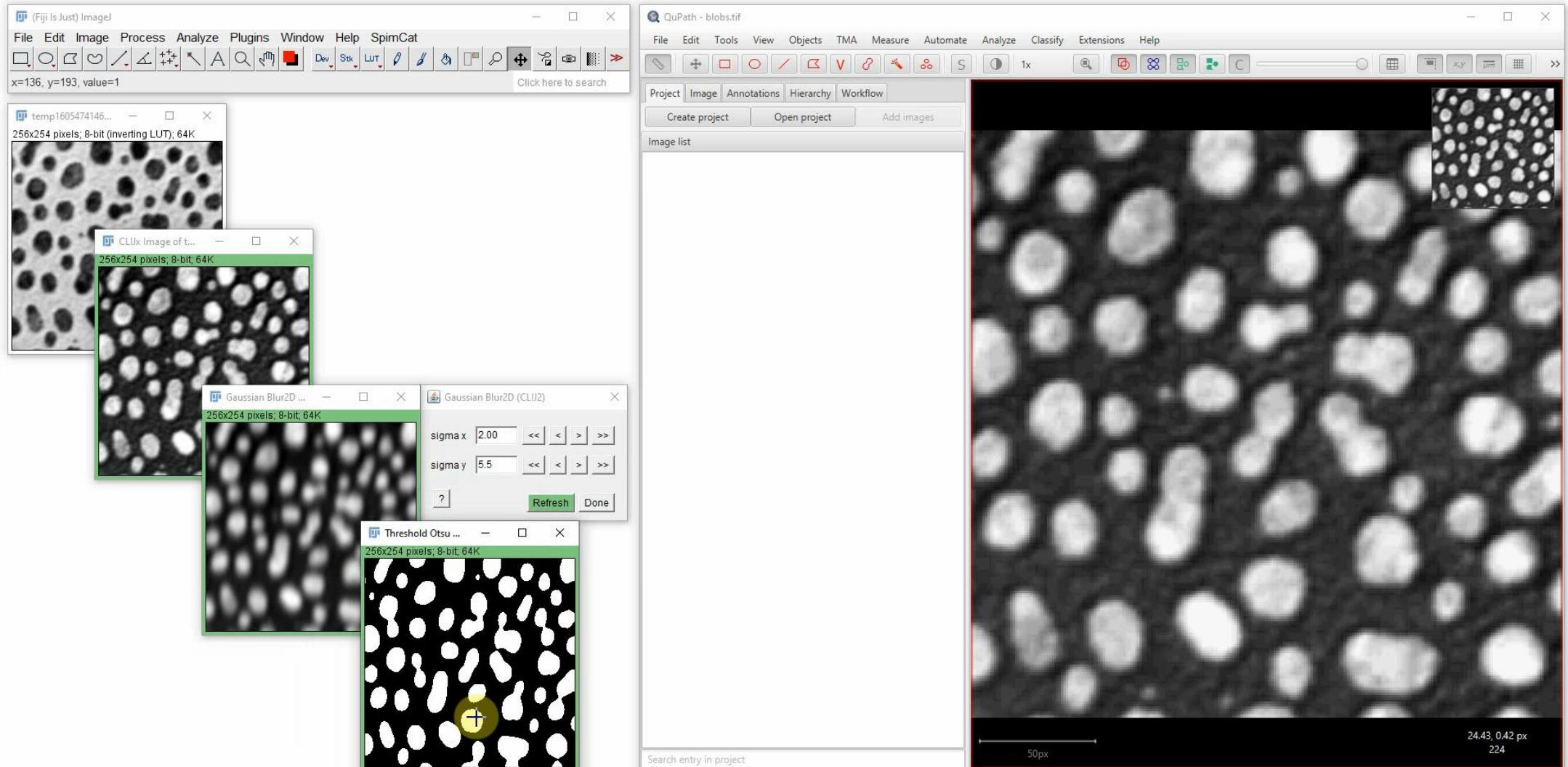


- Export scripts to Matlab

- Install <https://github.com/clij/clatlab> or <https://github.com/clij/clatlabx>



- If your workflow ends in a binary image, you can export a Groovy script for QuPath



clEsperanto script language comparison



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

The screenshot illustrates the workflow for generating scripts from an image data flow graph. On the left, the Fiji interface is shown with several windows open:

- A main window titled "blobs.tif" showing a grayscale image of blobs.
- A "Gaussian Blur2D ..." dialog with "sigmaX" and "sigmaY" set to 2.00.
- A "Sobel of Gaussian..." dialog showing the result of the blur operation.
- A context menu for the Sobel result window, with "Generate script" highlighted.

To the right, two code editors show the generated scripts:

- new1.ijm**: An ImageJ Macro (Human readable protocol) script:

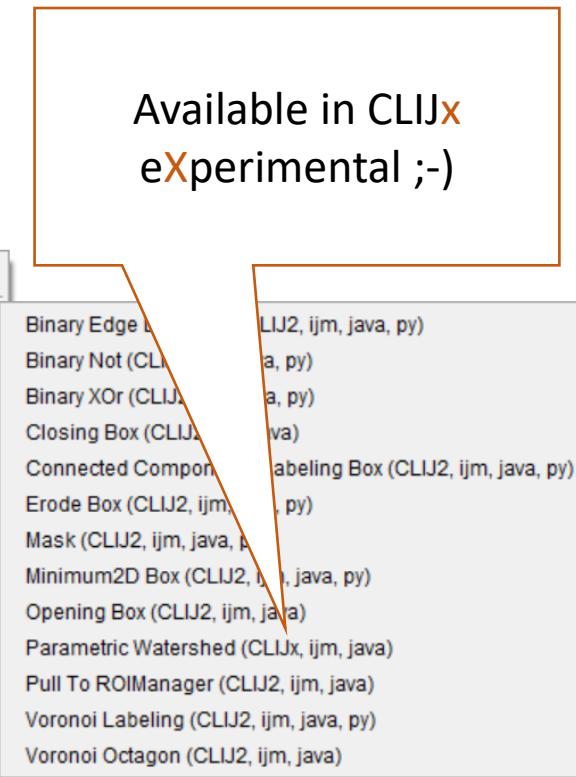
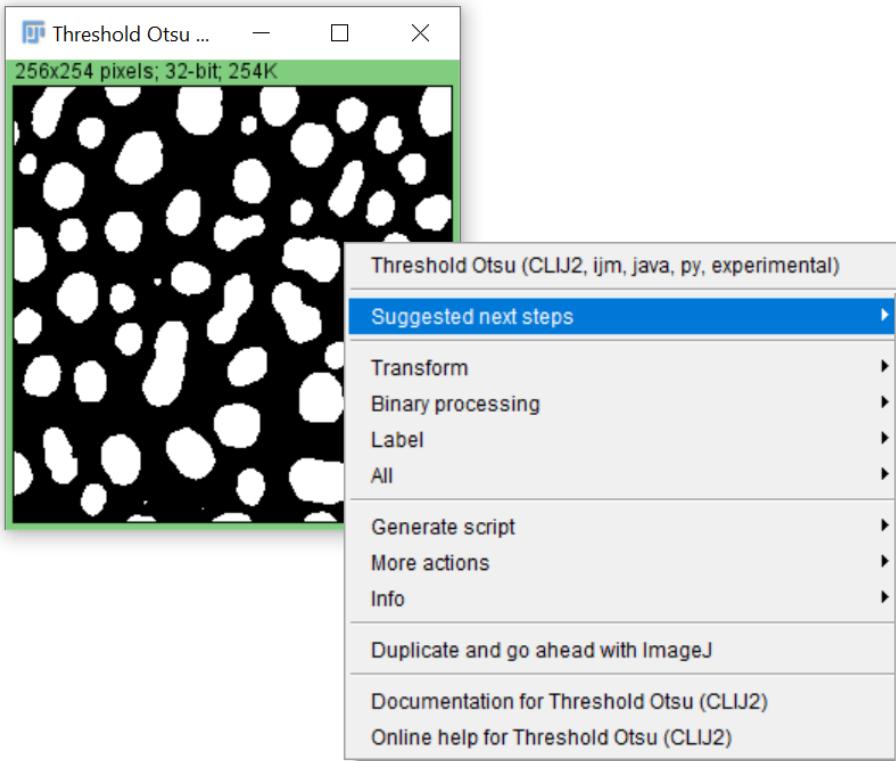
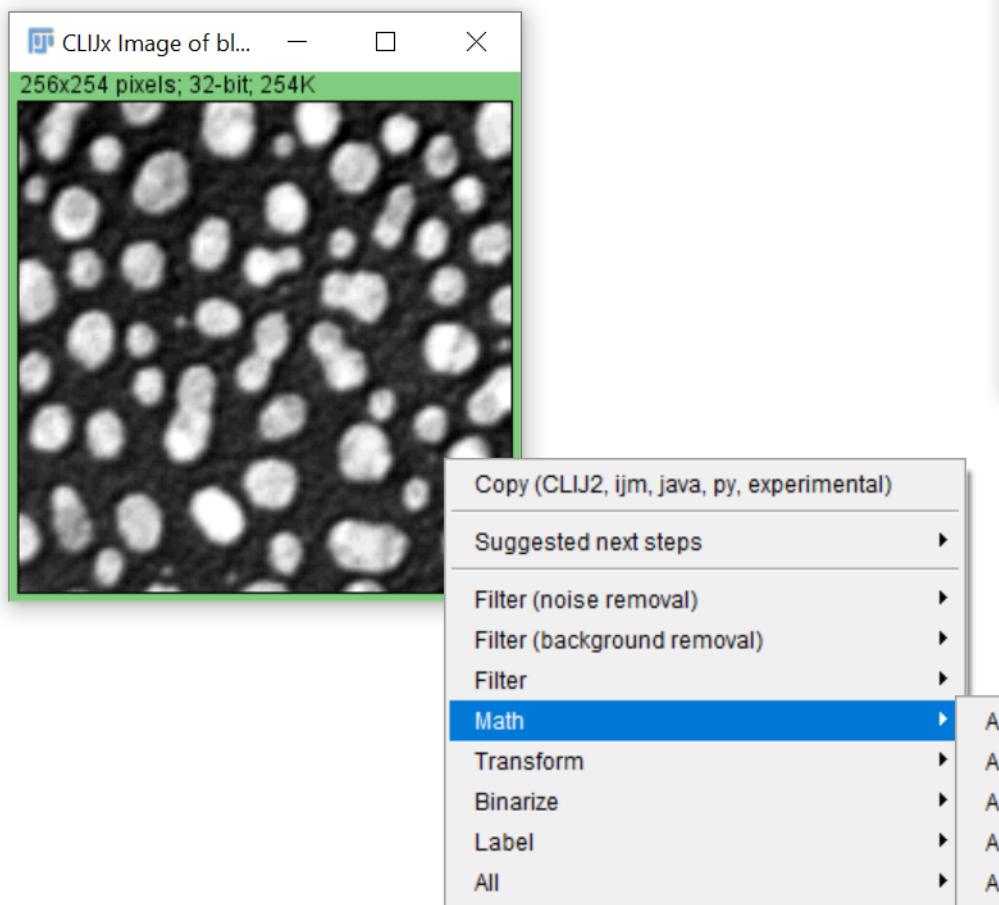
```
1 // To make this script run in Fiji, please activate
2 // the clij and clij2 update sites in your Fiji
3 // installation. Read more: https://clij.github.io
4
5 // Init GPU
6 run("CLIJ2 Macro Extensions", "cl_device=");
7
8 // Load image from disc
9 open("C:/structure/data/New folder/blobs.tif");
10 image1 = getTitle();
11 Ext.CLIJ2_push(image1);
12
13 // Gaussian Blur2D
14 sigmaX = 2;
15 sigmaY = 2;
16 Ext.CLIJ2_gaussianBlur2D(image1, image2, sigmaX, sigmaY);
17 Ext.CLIJ2_release(image1);
18
19 Ext.CLIJ2_pull(image2);
20
21 // Sobel
22 Ext.CLIJ2_sobel(image2, image3);
23
24 Ext.CLIJ2_pull(image3);
25
26 // Greater
27 Ext.CLIJ2_greater(image3, image2, image4);
28 Ext.CLIJ2_release(image3);
29 Ext.CLIJ2_release(image2);
30
31 Ext.CLIJ2_pull(image4);
32 Ext.CLIJ2_release(image4);
33
```
- new2.py**: A Python script using pyclesperanto_prototype:

```
14 # stay tuned and check out http://clesperanto.net to learn more.
15
16 import pyclesperanto_prototype as cle
17 from tifffile import imread
18
19 import numpy as np
20 import matplotlib
21 import matplotlib.pyplot as plt
22
23
24 # Load image
25 image = imread("C:/structure/data/New folder/blobs.tif")
26
27 # Push blobs.tif to GPU memory
28 image1 = cle.push_zyx(image)
29
30 # Gaussian Blur2D
31 image2 = cle.create_like(image1);
32 source = null
33 destination = image2
34 sigmaX = 2
35 sigmaY = 2
36 cle.gaussian_blur(source, destination, sigmaX, sigmaY)
37 # show result
38 plt.imshow(image2)
39 plt.show()
40
41
42 # Sobel
43 image3 = cle.create_like(image2);
44 source = image2
45 destination = image3
46 cle.sobel(source, destination)
47 # show result
48 plt.imshow(image3)
49 plt.show()
50
51
52 # Greater
53 image4 = cle.create_like(image3);
54 source1 = image3
55 source2 = image2
56 destination = image4
57 cle.greater(source1, source2, destination)
58 # show result
59 plt.imshow(image4)
60 plt.show()
61
62
```

clEsperanto: Compatibility



- Some operations are not available in all environments



Available in CLIJ2
Accessible from Macro,
Java (Matlab, Icy, Fiji),
Python and C++

pyclesperanto: 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()
8
9 # initialize GPU
10 import pyclesperanto_prototype as cle
11 # optional:
12 cle.select_device("GPU name")
13
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()
```

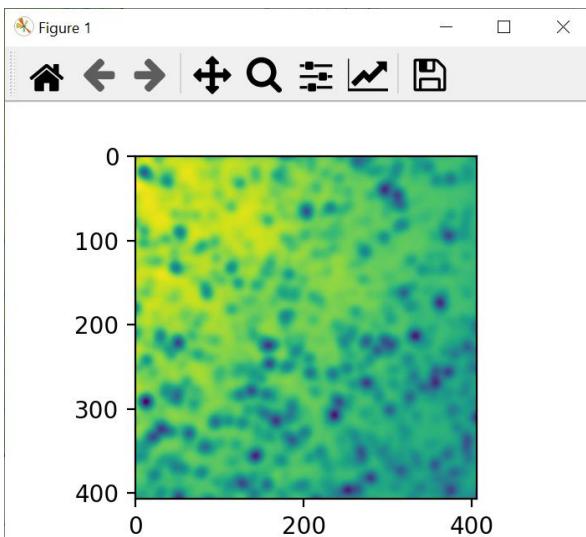
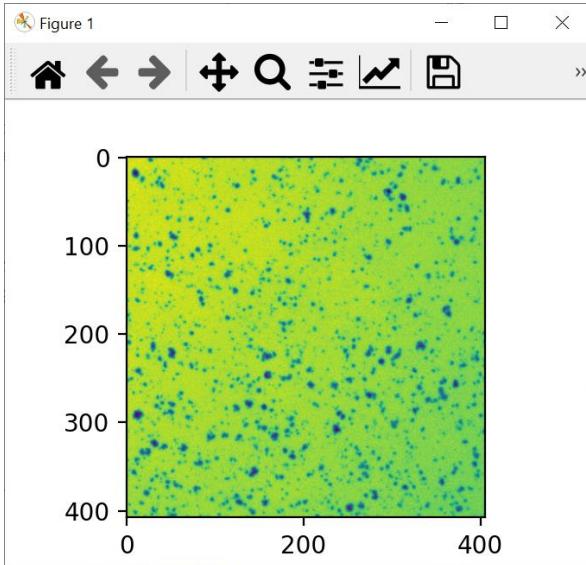
- Initialize GPU

- Push

- Process images

- Pull

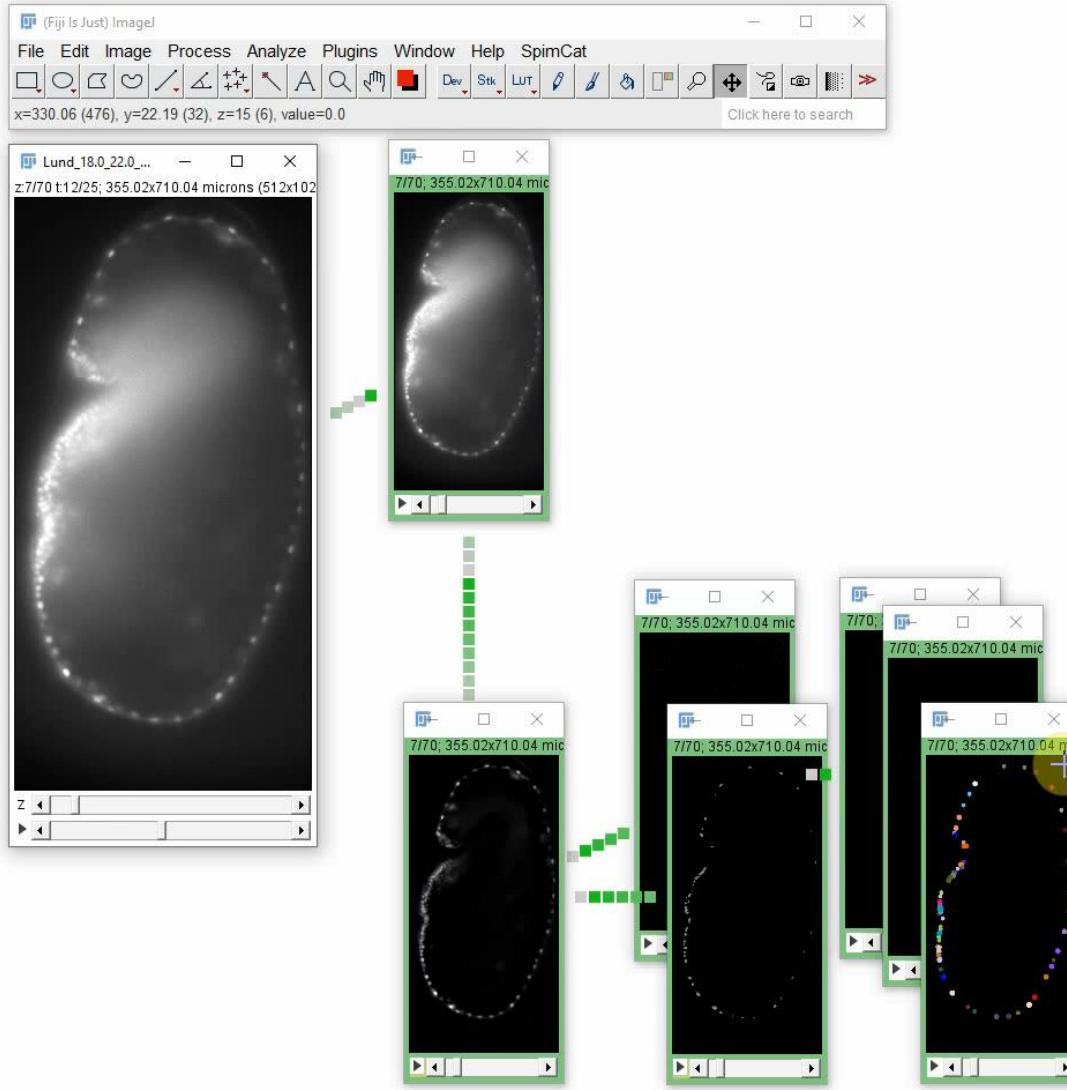
- Cleanup



pyclesperanto + napari script export

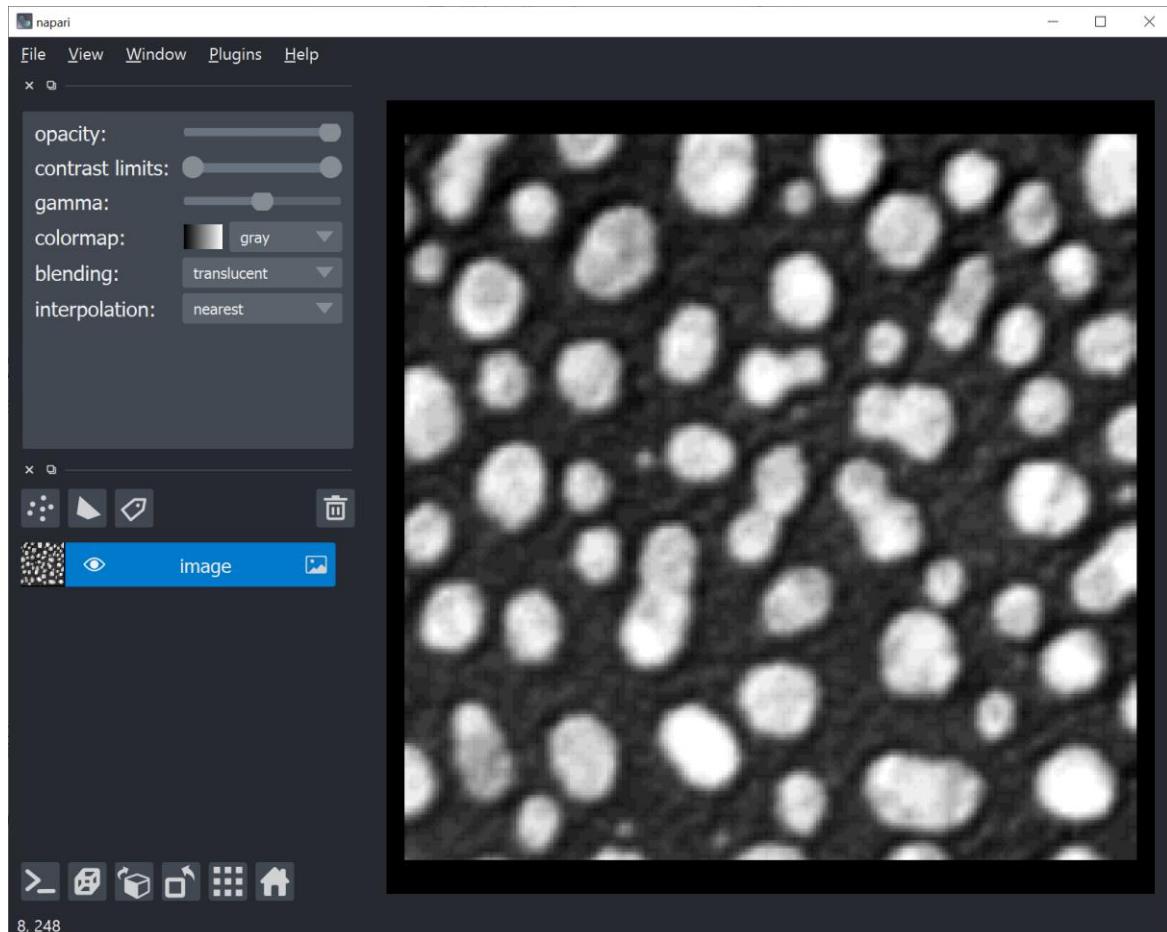


- napari is a multidimensional image viewer in the Python ecosystem



- Barebone minimal napari script

```
1 # Load image
2 from tifffile import imread
3 image = imread("C:/structure/data/blobs.tif")
4
5 # Start napari viewer
6 import napari
7 with napari.gui_qt():
8     viewer = napari.Viewer()
9
10    # show image
11    viewer.add_image(image)
```



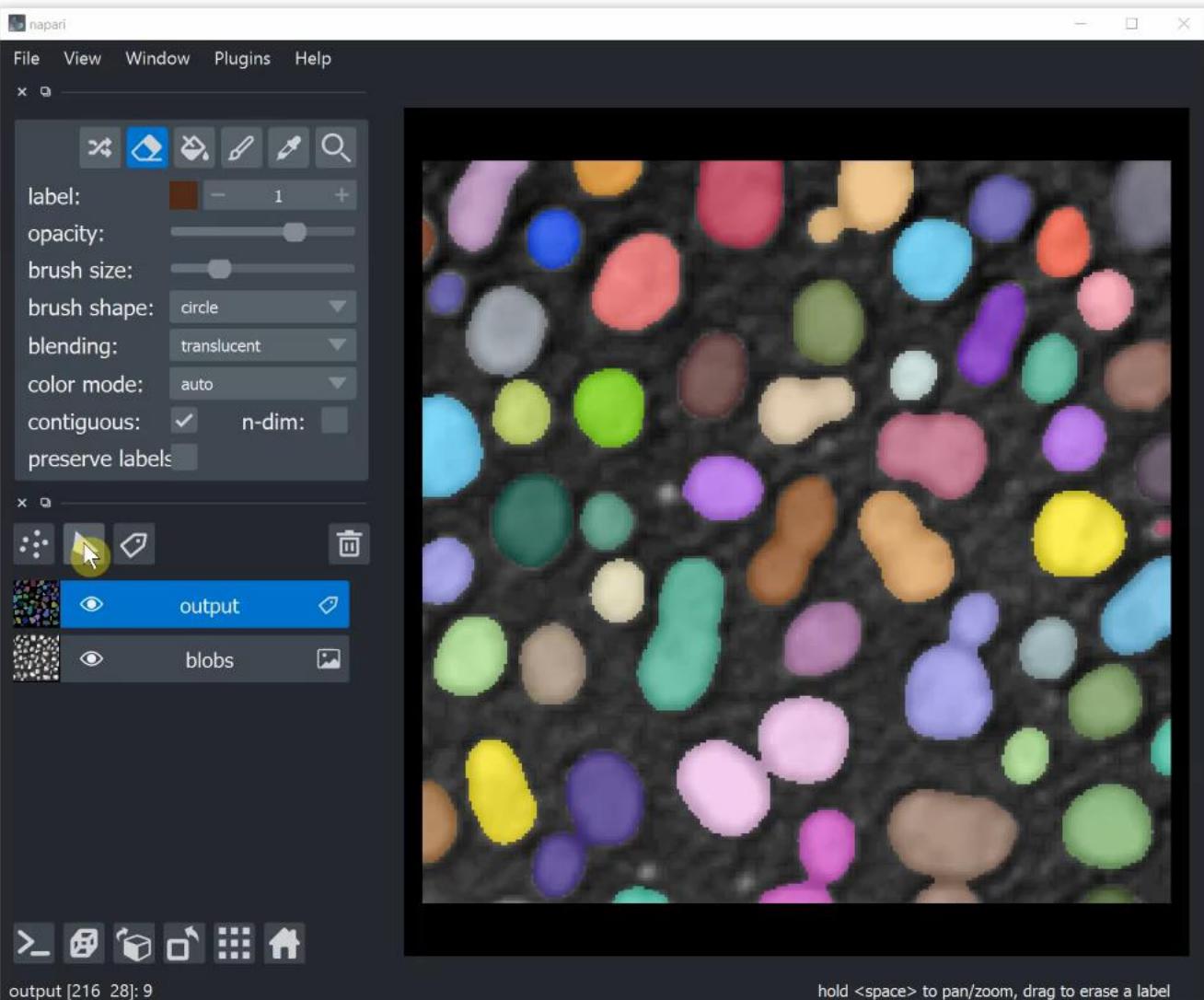
- Call it with
ipython --gui=qt napari_script.py

pyclesperanto + napari



- Minimal napari + clesperanto example

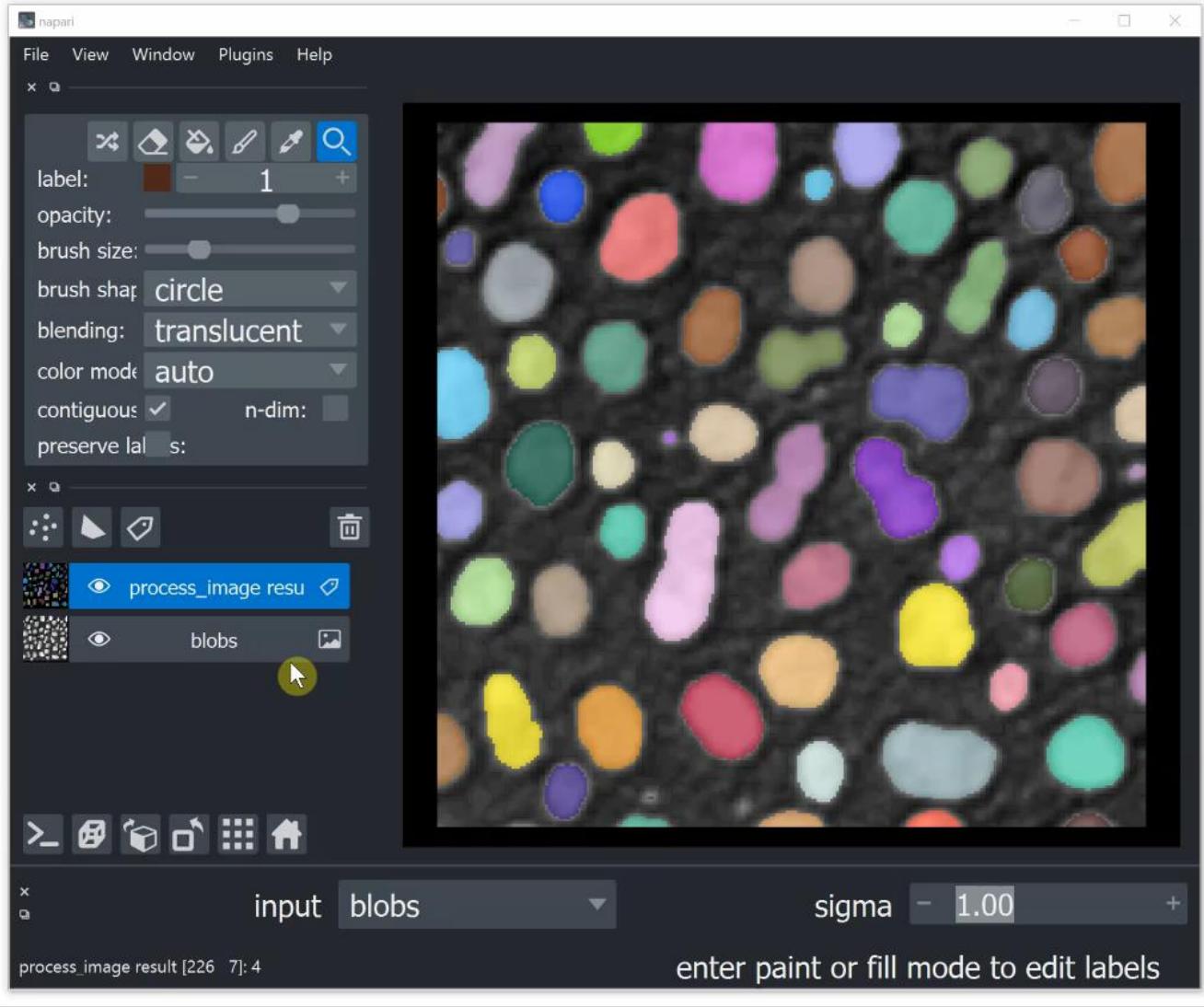
```
1 # load data
2 from skimage.io import imread
3 image = imread('https://samples.fiji.sc/blobs.png')
4
5 # start up napari
6 import napari
7 import pyclesperanto_prototype as cle
8 with napari.gui_qt():
9     viewer = napari.Viewer()
10    viewer.add_image(image, name='blobs')
11
12    # push image to GPU
13    input = cle.push_zyx(image)
14
15    # process the mage
16    sigma = 3
17    blurred = cle.gaussian_blur(input, sigma_x=sigma, sigma_y=sigma)
18    binary = cle.threshold_otsu(blurred)
19    labels = cle.connected_components_labeling_box(binary)
20
21    # pull result back
22    output = cle.pull_zyx(labels)
23
24    # add it to napari
25    viewer.add_labels(output)
```



pyclesperanto + napari + magicgui



```
1 # Inspired by
2 # https://github.com/pr4deepn/pyclesperanto_prototype/blob/master/napari_clij_widget.py
3 import napari
4 import pyclesperanto_prototype as cle
5 from magicgui import magicgui
6 from napari.layers import Image, Labels
7
8 @magicgui(auto_call=True)
9 def process_image(input: Image, sigma: float = 5) -> Image:
10     if input:
11         # push image to GPU
12         input = cle.push_zyx(input.data)
13
14         # process the mage
15         blurred = cle.gaussian_blur(input, sigma_x=sigma, sigma_y=sigma)
16         binary = cle.threshold_otsu(blurred)
17         labels = cle.connected_components_labeling_box(binary)
18
19         # pull result back
20         output = cle.pull_zyx(labels)
21         return output
22
23 # load data
24 from skimage.io import imread
25 image = imread('https://samples.fiji.sc/blobs.png')
26
27 # start up napari
28 with napari.gui_qt():
29     viewer = napari.Viewer()
30     viewer.add_image(image, name='blobs')
31
32     # generate a Graphical User Interface from the function above magically
33     gui = process_image.Gui()
34     viewer.window.add_dock_widget(gui)
```



pyclesperanto + napari + dask



- “Lazy” / “Delayed” image processing: Process a frame as soon as the user wants to see it.

```
def process_image(image):
    import time

    start_time = time.time()

    # push image to GPU memory and show it
    gpu_input = cle.push_zyx(image)
    # print(gpu_input)

    # gaussian blur
    sigma = 2.0
    gpu_blurred = cle.gaussian_blur(gpu_input, sigma_x=sigma, sigma_y=sigma, sigma_z=sigma)

    # detect maxima
    gpu_detected_maxima = cle.detect_maxima_box(gpu_blurred)
```

```
# adapted from: https://github.com/tLambert03/napari-dask-example/blob/master/dask_napari.ipynb
import dask
import dask.array as da

# create dask stack of lazy image readers
lazy_process_image = dask.delayed(process_image) # Lazy reader
lazy_arrays = [lazy_process_image(timelapse[n]) for n in range(0, timelapse.shape[0])]
dask_arrays = [
    da.from_delayed(lazy_array, shape=timelapse[0].shape, dtype=timelapse[0].dtype)
    for lazy_array in lazy_arrays
]
# Stack into one Large dask.array
dask_stack = da.stack(dask_arrays, axis=0)
dask_stack
```

https://github.com/cI Esperanto/pyclesperanto_prototype/blob/master/demo/napari_gui/napari_dask.ipynb

- “Lazy” / “Delayed” image processing: Process a frame as soon as the user wants to see it.

```

def process_image(image):
    import time

    start_time = time.time()

    # push image to GPU memory and show it
    gpu_input = cle.push_zyx(image)
    # print(gpu_input)

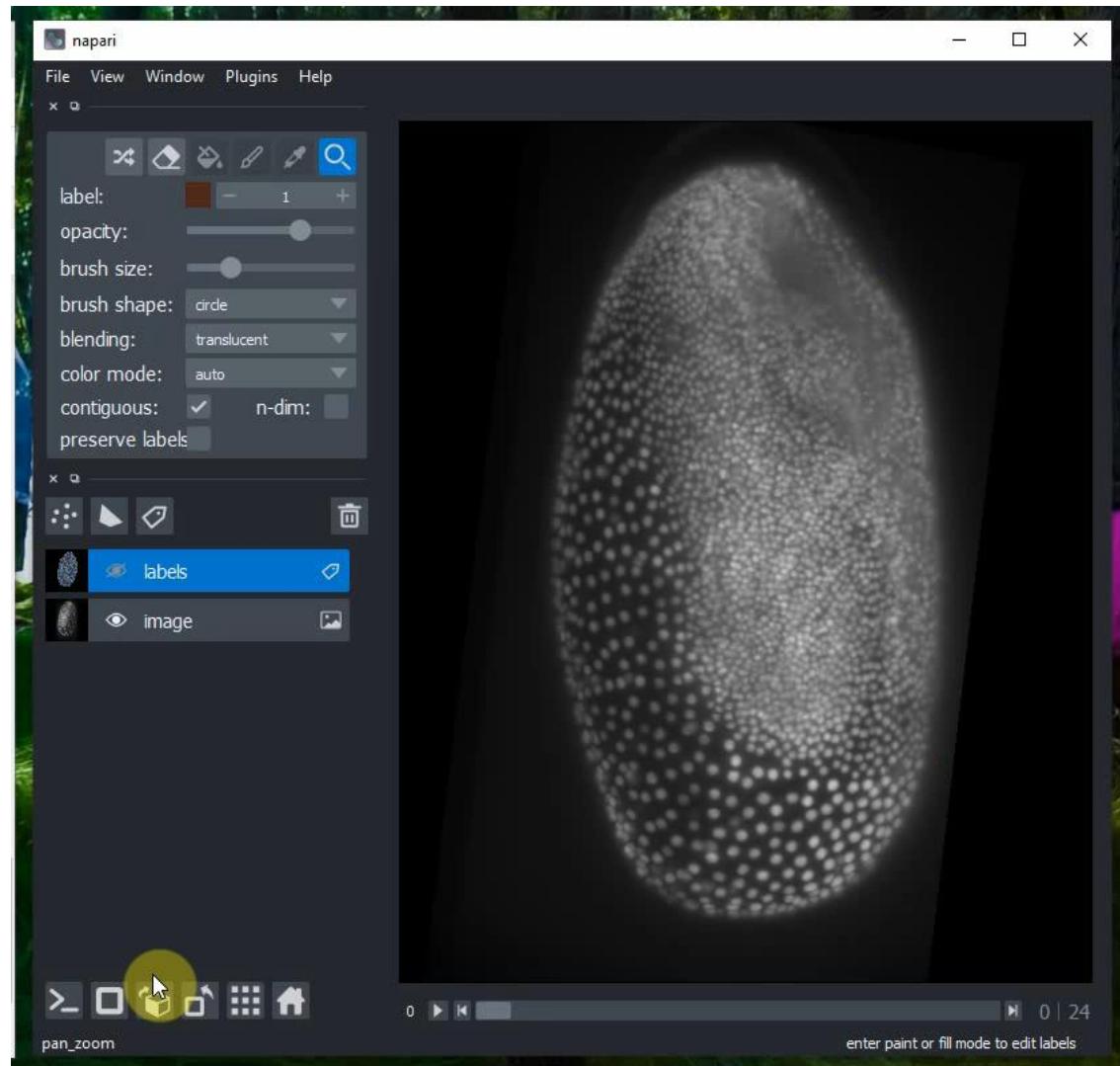
    # gaussian blur
    sigma = 2.0
    gpu_blurred = cle.gaussian_blur(gpu_input, sigma_x=sigma, sigma_y=sigma, sigma_z=sigma)

    # detect maxima
    gpu_detected_maxima = cle.detect_maxima_box(gpu_blurred)

# adapted from: https://github.com/tLambert03/napari-dask-example/blob/master/dask_napari.ipynb
import dask
import dask.array as da

# create dask stack of Lazy image readers
lazy_process_image = dask.delayed(process_image) # Lazy reader
lazy_arrays = [lazy_process_image(timelapse[n]) for n in range(0, timelapse.shape[0])]
dask_arrays = [
    da.from_delayed(lazy_array, shape=timelapse[0].shape, dtype=timelapse[0].dtype)
    for lazy_array in lazy_arrays
]
# Stack into one Large dask.array
dask_stack = da.stack(dask_arrays, axis=0)
dask_stack

```



From napari: developer services coming soon!



New!

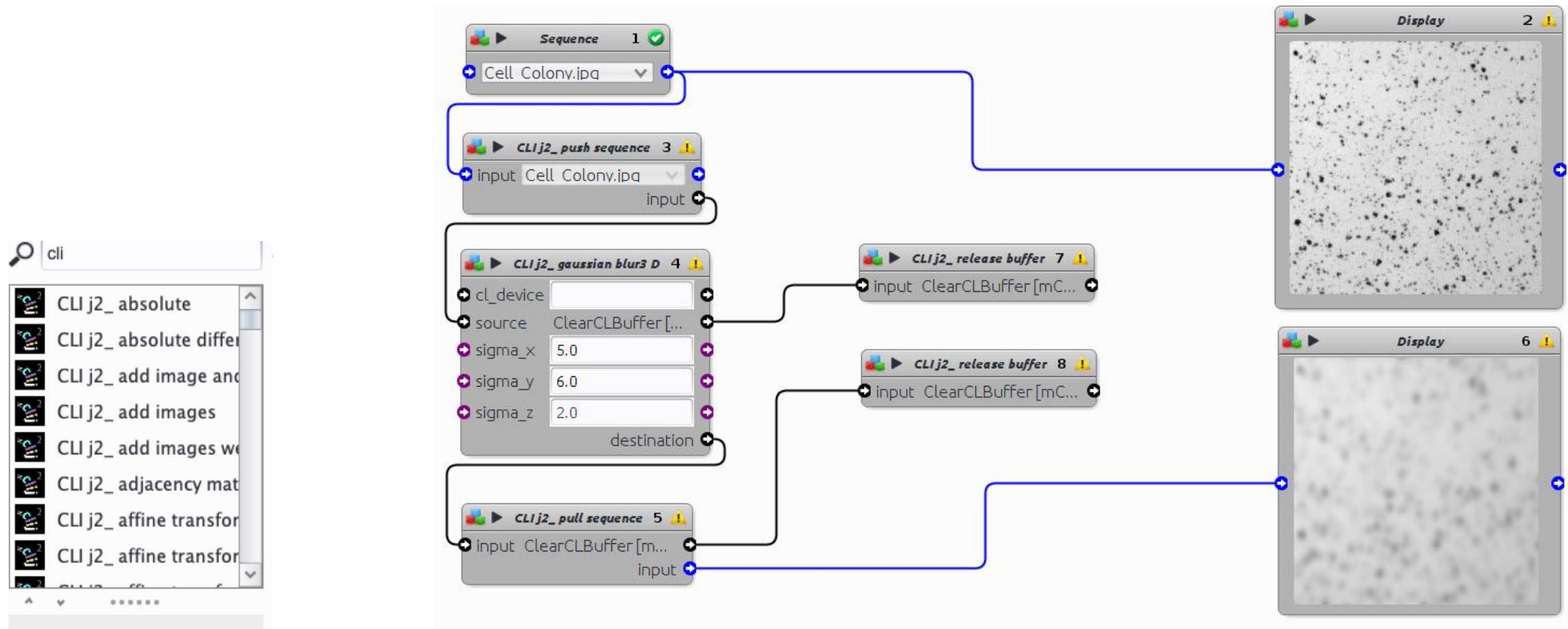
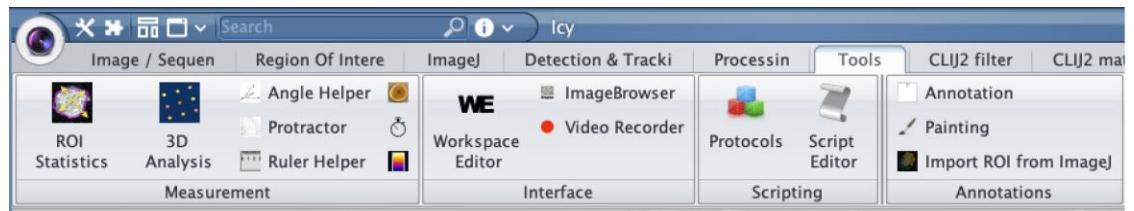
<https://www.napari.dev>

- **Mission:** help developers build their best Python plugin
- Will host services to support the **easy packaging, testing, integration and maintenance** of plugins
- help you **easily distribute** your plugin to users through a plugin store
- **Sign up today** to get early access notification and chat with the developer services team about your plugin!

The screenshot shows a web browser window for the napari Developer Portal at napari.dev. The main heading reads: "Share your analysis tool with researchers everywhere – No GUI development required." Below this, a paragraph explains the mission: "We're building developer services to help bring Python-driven analysis to the imaging community, and we want you to be involved." Another paragraph describes napari: "Led by microscopy and Python experts and built by a growing community, napari is quickly becoming an essential tool for visualizing and exploring imaging data. Our soon-to-debut napari Developer Portal, managed by the Imaging team at the Chan Zuckerberg Initiative, will support napari's plugin developer community every step of the way." To the right, there is a large, colorful fluorescence microscopy image of tissue sections. A sidebar on the right contains text: "Get notified when the napari developer portal launches." and "Help shape the future of bioimage analysis." Below this is a sign-up form with fields for "Email" and a "Sign up!" button. A small note below the button states: "We will use your contact information to keep you updated about the napari developer portal's progress, including announcements and other news." At the bottom, it says "Photo provided by Anna Bäckström and Emma Lundberg".

Icy Protocol generator

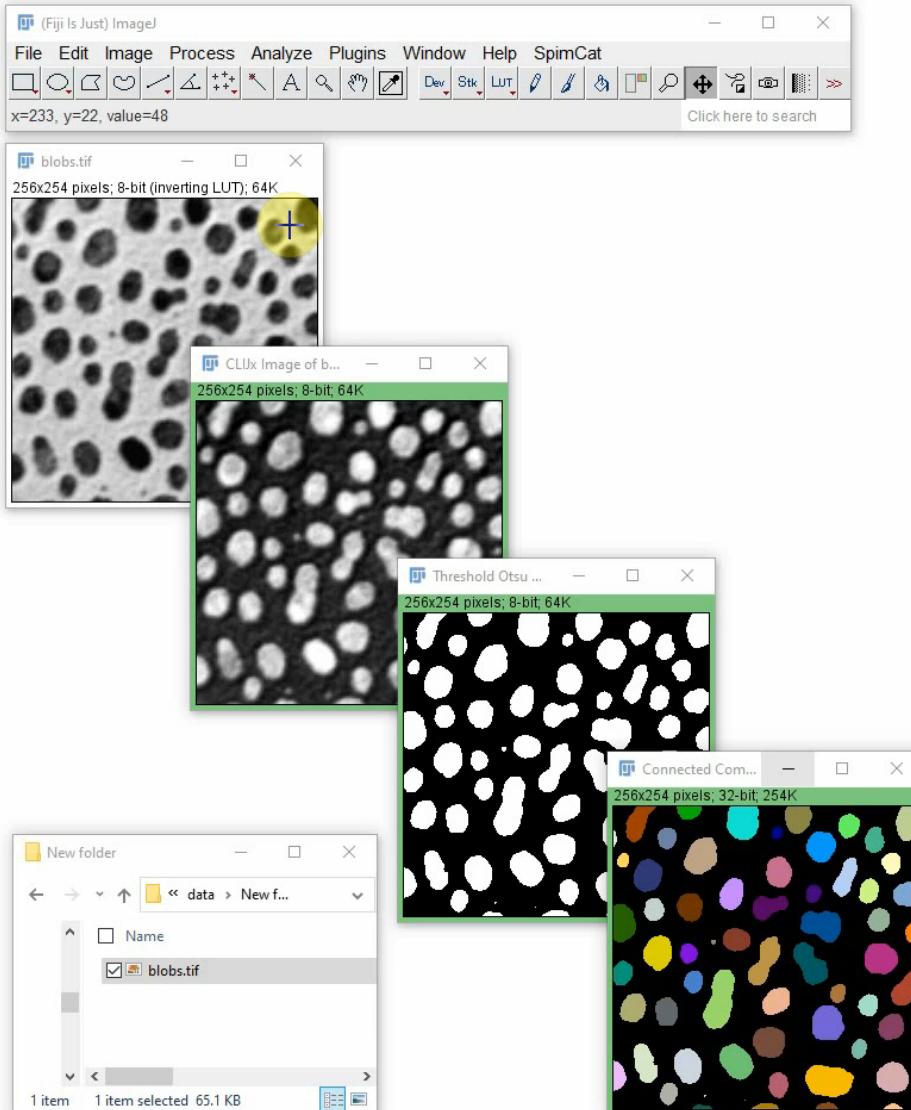
CLIcy in ICY protocols



Export ICY Protocols from Fiji



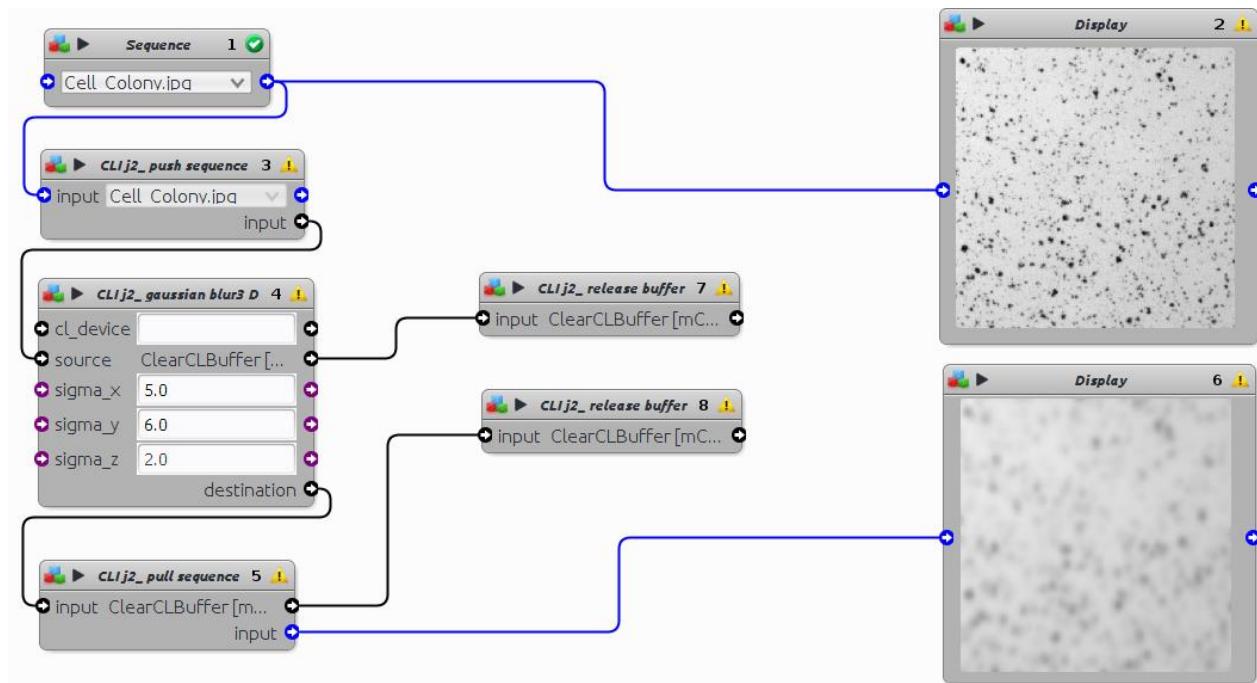
- <https://clij.github.io/assistant/installation#icy>



CLIcy in ICY protocols



- Protocols can be easily shared, and published in journals for reproducibility
- You can also upload them on the ICY website for reference
- You can then also find other example of protocols there!



CLICy in ICY protocols



<http://icy.bioimageanalysis.org>

The screenshot shows the ICY website's main navigation bar with links to HOME, RESOURCES (highlighted in blue), SUPPORT, ICYSERVICES, CONTRIBUTORS, GET INVOLVED, and ABOUT. Below the navigation is a search bar and a user icon. The main content area features three large circular statistics: 'PLUGINS' (424), 'SCRIPTS' (81), and 'PROTOCOLS' (77), each with a corresponding icon.

- Protocols can be easily shared, and published in journals for reproducibility
- You can also upload them on the ICY website for reference
- You can then also find other example of protocols there!

The screenshot shows a protocol page for 'Easy Cell shape with HK-Means' by IcyLyd. It includes a short description, documentation, review options, and a changelog. The right side features a documentation section with an image of a neuron and a workflow diagram titled 'Cell shape with HK-Means protocol'.

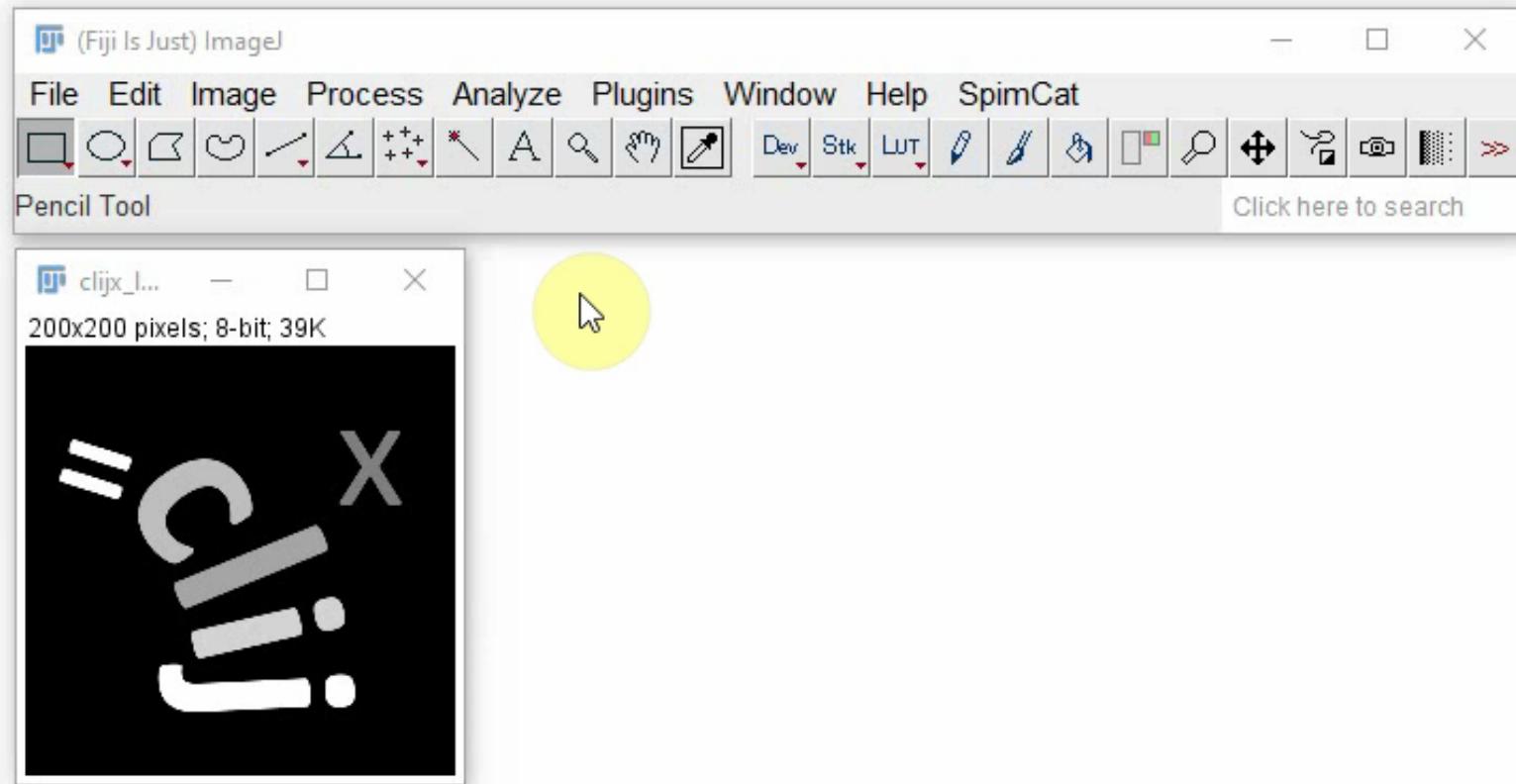
Fiji Plugin generator

Fiji plugin generator



- Generate Fiji plugins using Maven

<https://clij.github.io/assistant/installation#maven>



Fiji plugin generator



- You can explore the generated folder: The only Java file contains your algorithm

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "temp1606221774741" and contains a package "net.haesleinhuepf.clijx.plugins" with a class "Iso_Cylinder_Max_Projection".
- Code Editor:** The file "Iso_Cylinder_Max_Projection.java" is open, showing Java code for a CLIJ macro plugin. It includes annotations like @Plugin and @Override, and methods like getParameterHelpText() and getDefaultValues().
- Maven Tool Window:** On the right, the Maven tool window shows the project structure under "Lifecycle" with various goals like clean, validate, compile, test, package, verify, install, site, and deploy.
- Run Tab:** The "Run" tab shows a successful build for the package "cliж-assistant-iso_cylinder_max_projection_". The output log shows:

```
[INFO] Skipping packaging of the test-jar  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 10.970 s  
[INFO] Finished at: 2020-11-24T13:46:18+01:00  
[INFO] -----
```

Fiji plugin generator



- You can explore the generated folder: The only Java file contains your algorithm

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "temp1606221774741" and contains a module named "clijx-assistant-iso_cylinder". This module has a source directory "src" containing a package "net.haesleinhuepf.clijx.plugins" which holds a class "Iso_Cylinder_Max_Projection.java".
- Code Editor:** The "Iso_Cylinder_Max_Projection.java" file is open, displaying Java code that performs operations like copying between CLBuffers, making an image isotropic, applying a cylinder transform, and performing a maximum Z projection.
- Run Tab:** The "Run" tab shows a successful build for the package "clijx-assistant-iso_cylinder_max_projection_". The log output indicates a total time of 10.970 seconds and a finish time of 2020-11-24T13:46:18+01:00.
- Maven Tab:** The "Maven" tab shows the project's build configuration, including profiles, lifecycle phases (clean, validate, compile, test, package, verify, install), and dependencies.

Fiji plugin generator



- The plugins.config file contains the menu path where your plugin can be found in ImageJ

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "temp1606221774741". The "src/main/resources" folder contains "plugins.config".
- Code Editor:** Displays the content of "plugins.config":

```
Plugins>ImageJ on GPU (CLIJx)>Custom>Robert Haase, "Iso_Cylinder_Max_Projection", net.haesleinhuepf.clijx.plugins.
```
- Maven Tool Window:** Shows the Maven lifecycle with "install" selected.
- Run Tab:** Displays the build output:

```
[INFO] Skipping packaging of the test-jar  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 10.970 s  
[INFO] Finished at: 2020-11-24T13:46:18+01:00  
[INFO] -----
```
- Bottom Navigation:** Includes tabs for Run, TODO, Problems, Terminal, and Build.

Fiji plugin generator



- The pom.xml file contains project information

The screenshot shows the IntelliJ IDEA interface with the following details:

- File Bar:** File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help.
- Title Bar:** clijx-assistant-iso_cylinder_max_projection_ - pom.xml (clijx-assistant-iso_cylinder_max_projection_)
- Project Tree:** temp1606221774741 [clijx-assistant-iso_cylinder], .idea, src, main, java, net.haesleinhuepf.cljx.plugins (Iso_Cylinder_Max_Projection), resources, plugins.config, .gitignore, clijx-assistant-iso_cylinder_max_projection_.iml, pom.xml, readme.md.
- Code Editor:** Content pane showing the XML code of the pom.xml file. The code includes parent information, group ID, artifact ID, version, name, description, and developer details.
- Maven Tool Window:** Shows Maven profiles, lifecycle (clean, validate, compile, test, package, verify, install, site, deploy), and dependencies.
- Run Tab:** clijx-assistant-iso_cylinder_max_projection_[package] is selected, showing a green checkmark and build logs.
- Bottom Status Bar:** Event Log, 1:1 CRLF, UTF-8, Tab*, and a file icon.



PoL
Physics of Life
TU Dresden

 **INSTITUT
PASTEUR**



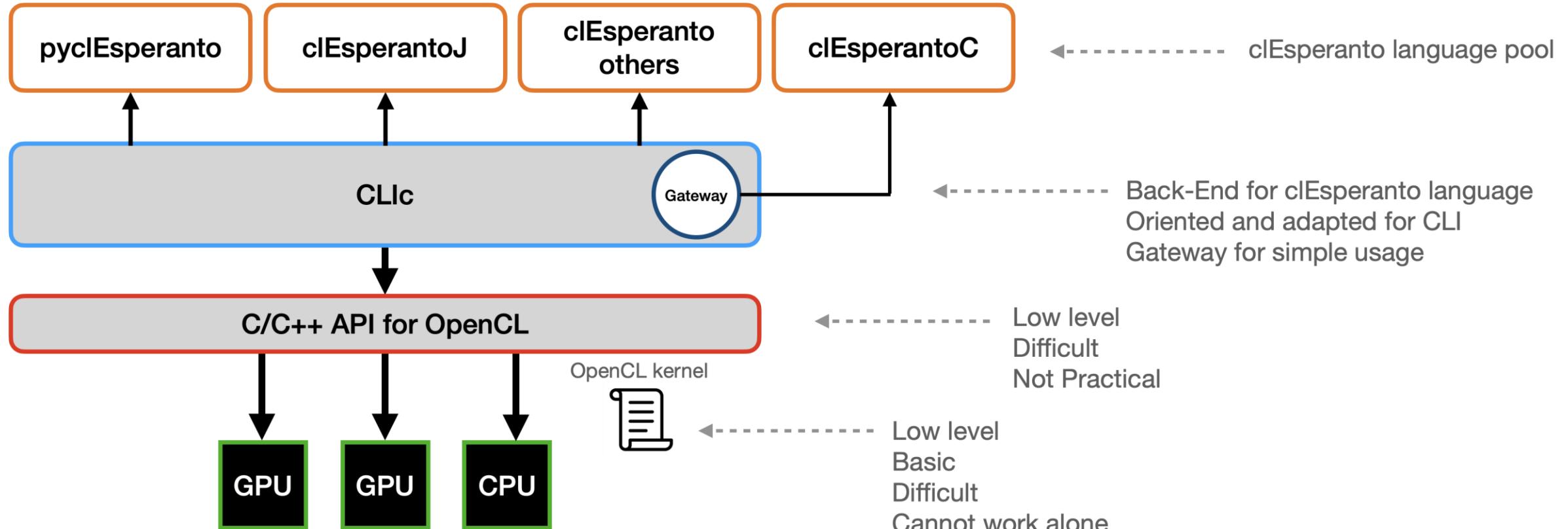
CENTER FOR
SYSTEMS BIOLOGY
DRESDEN



CBG
Max Planck Institute
of Molecular Cell Biology
and Genetics

CLlc: One language to (try to) rule them all

CLIC: One language to ^(try to) rule them all



API / Back-End

Kernel Classes

mean filter

gaussian filter

maximum project

Object Classes

buffer

scalar

image3d

image2d

CLE

Manager Classes

command queue

device

gpu

context

clEsperantoC script

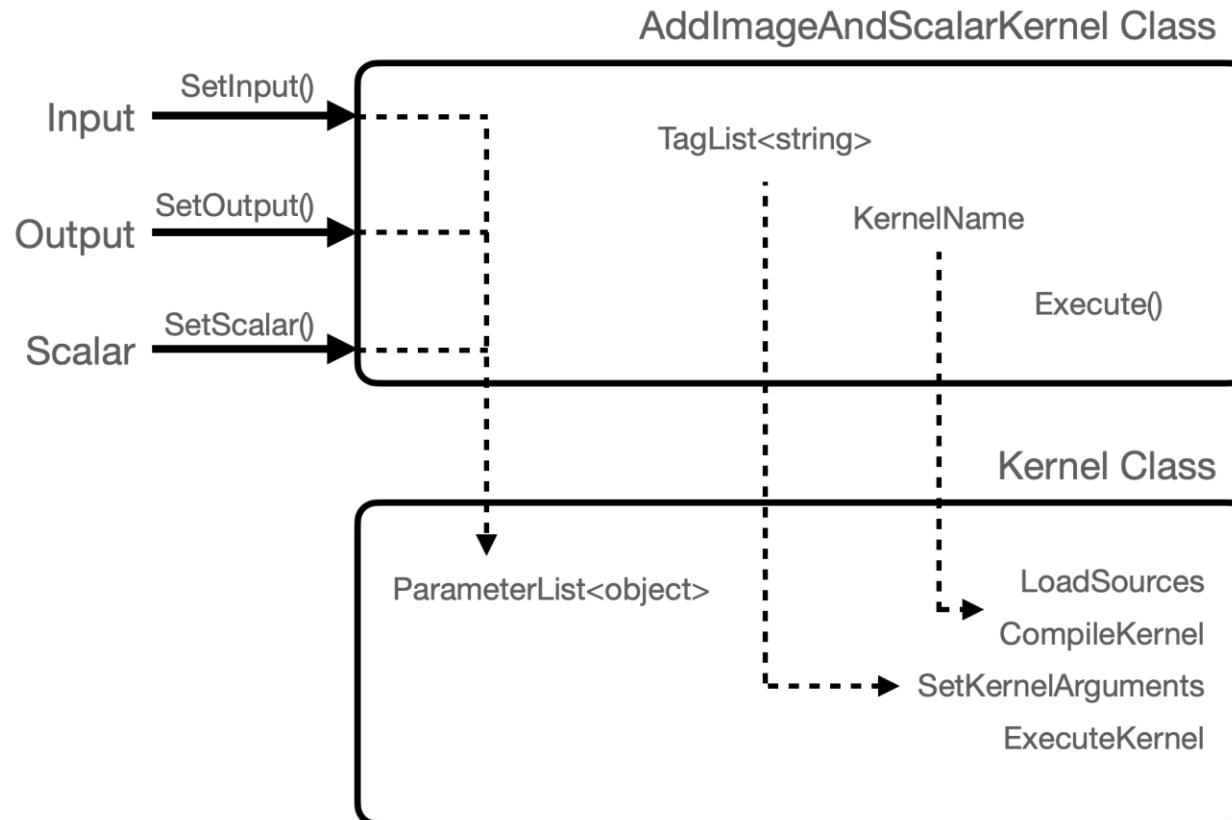
```
// Initialise GPU information.
cle::GPU gpu;
cle::CLE cle(gpu);

// Push image
Image<unsigned char> raw_img (raw, 256, 254, "unsigned char");
cle::Buffer img1 = cle.Push<unsigned char>(raw_img);

// Create buffer
std::array<unsigned int, 2> dimensions = {256, 254};
cle::Buffer img2 = cle.Create<unsigned char>(dimensions.data(),
                                              "unsigned char");

// Smaller Or Equal Constant
cle.smaller_or_equal_constant(img1, img2, 110);

// Pull output into container
Image<unsigned char> res = gpu.Pull<unsigned char>(img2);
```



- Clear and simple Set methods name
- Manage parameter name and order
 - Need to correspond to .cl implementation
- Kernel class manage dimensionality (2d/3d)
- Provide an Execute() method
 - Allow mini-pipeline of kernels
 - Allow complexe operations

- Manage input size and type
- Compilation of kernel and program
- Connect parameter to kernel
- Set operation to GPU queue
- Do the hard lifting behind the curtains

User visibility through the gateway:

```
cle.AddImageAndScalar(Buffer _input, Buffer _output, float _scalar);
```



header

```
namespace cle
{
    class AddImageAndScalarKernel : public Kernel
    {
        private:
            void DefineDimensionality();
        public:
            AddImageAndScalarKernel(GPU& gpu) : Kernel(gpu)
            {
                kernelName = "add_image_and_scalar"; // kernel name
                tagList = {"src", "dst", "scalar"}; // parameter tag/order
            }

            void SetInput(Object&);

            void SetOutput(Object&);

            void SetScalar(float);

            void Execute();

            ~AddImageAndScalarKernel() = default;
    };
}
```

source

```
void AddImageAndScalarKernel::SetInput(Object& x)
{
    this->AddObject(&x, "src");
}

void AddImageAndScalarKernel::SetOutput(Object& x)
{
    this->AddObject(&x, "dst");
}

void AddImageAndScalarKernel::SetScalar(float x)
{
    Float* val = new Float(x); // convert float into generic object
    this->AddObject(val, "scalar");
}

void AddImageAndScalarKernel::Execute()
{
    DefineDimensionality(); // manage 2d or 3d data
    CompileKernel(); // compile .cl code into a gpu program
    AddArgumentsToKernel(); // link parameters to the compiled kernel
    DefineRangeKernel(); // set execution range
}
```



Initialise GPU

```
// Initialise GPU information.  
cle::GPU gpu;  
cle::CLE cle(gpu);
```



Initialise GPU

```
// Initialise GPU information.  
cle::GPU gpu;  
cle::CLE cle(gpu);
```

Use CLIjx-Assistant script generator!
Or trust the auto-completion of your C++ IDE

```
// Create buffer  
std::array<unsigned char> data(110);  
cle::Buffer img2 = cle.Create("img2", data);
```

The screenshot shows a code editor with a tooltip displaying auto-completion suggestions for the `cle.` prefix. The suggestions include:

- Absolute
- AddImageAndScalar
- Create
- GetGPU
- MaximumOfAllPixels
- MaximumXProjection
- MaximumYProjection
- MaximumZProjection
- Mean2DSphere
- Pull
- Push
- SmallerOrEqualConstant

Practical links

Code test example : https://github.com/clEsperanto/CLIC_prototype/test

Pre-made C++ empty project : https://github.com/StRigaud/CLIC_project_template

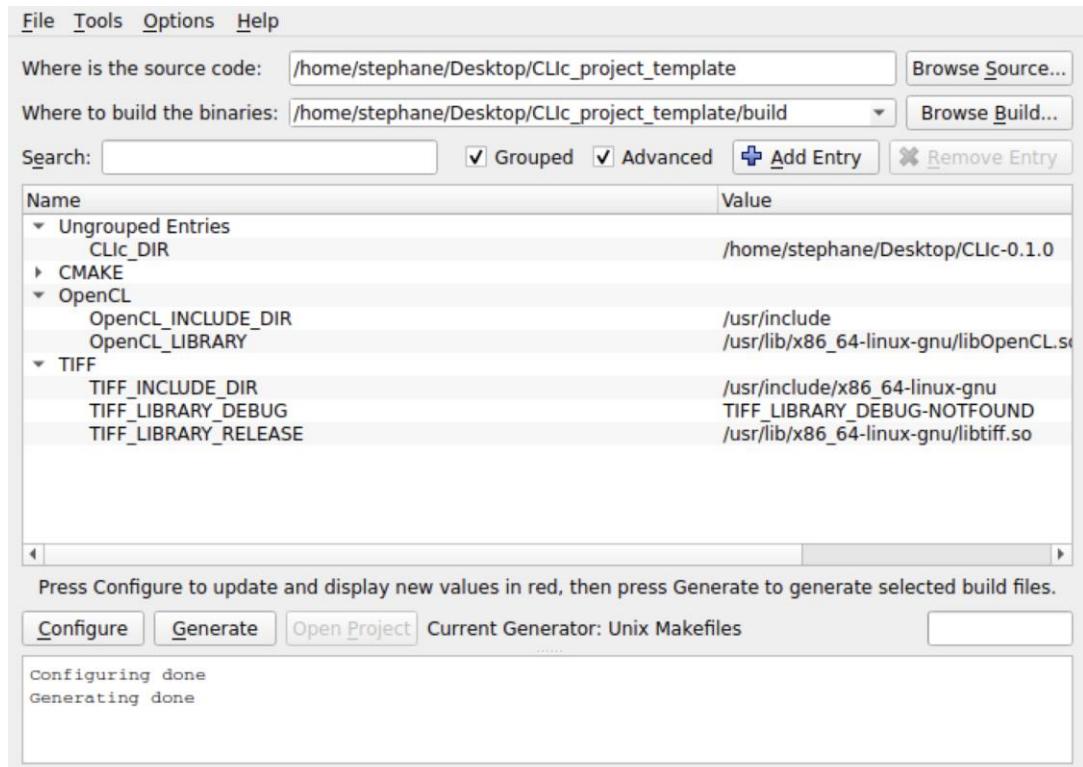
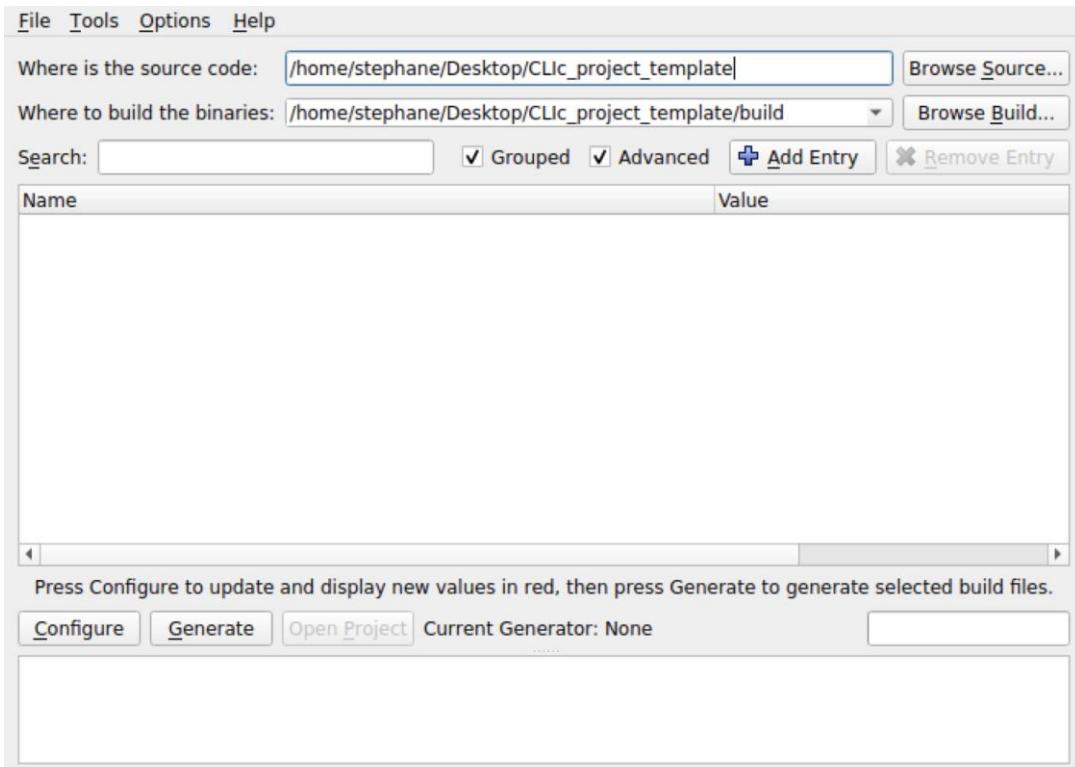
```
    pull  
    // full output into container  
    Image<unsigned char> res = gpu.Pull<unsigned char>(img2);
```

CLIC: Installation and Compilation



Requirement: OpenCL, LibTiff, Cmake

And let CMake do all the hard work of project creation and configuration!



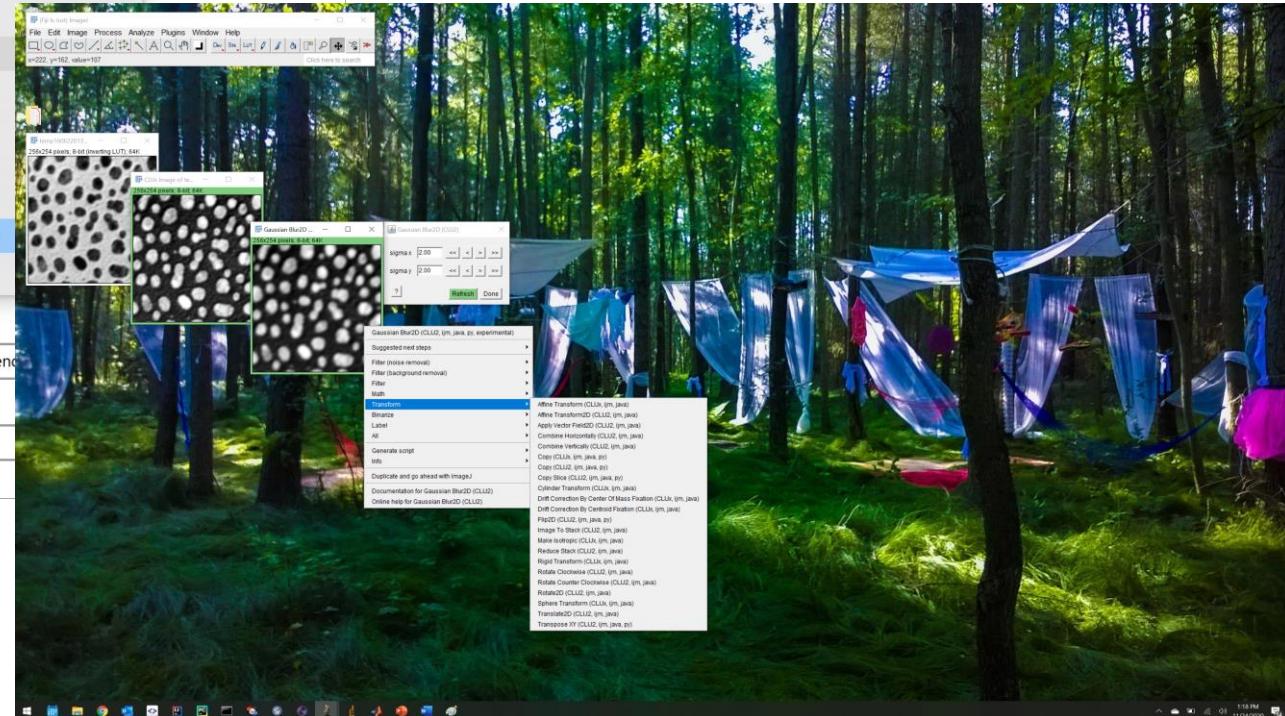
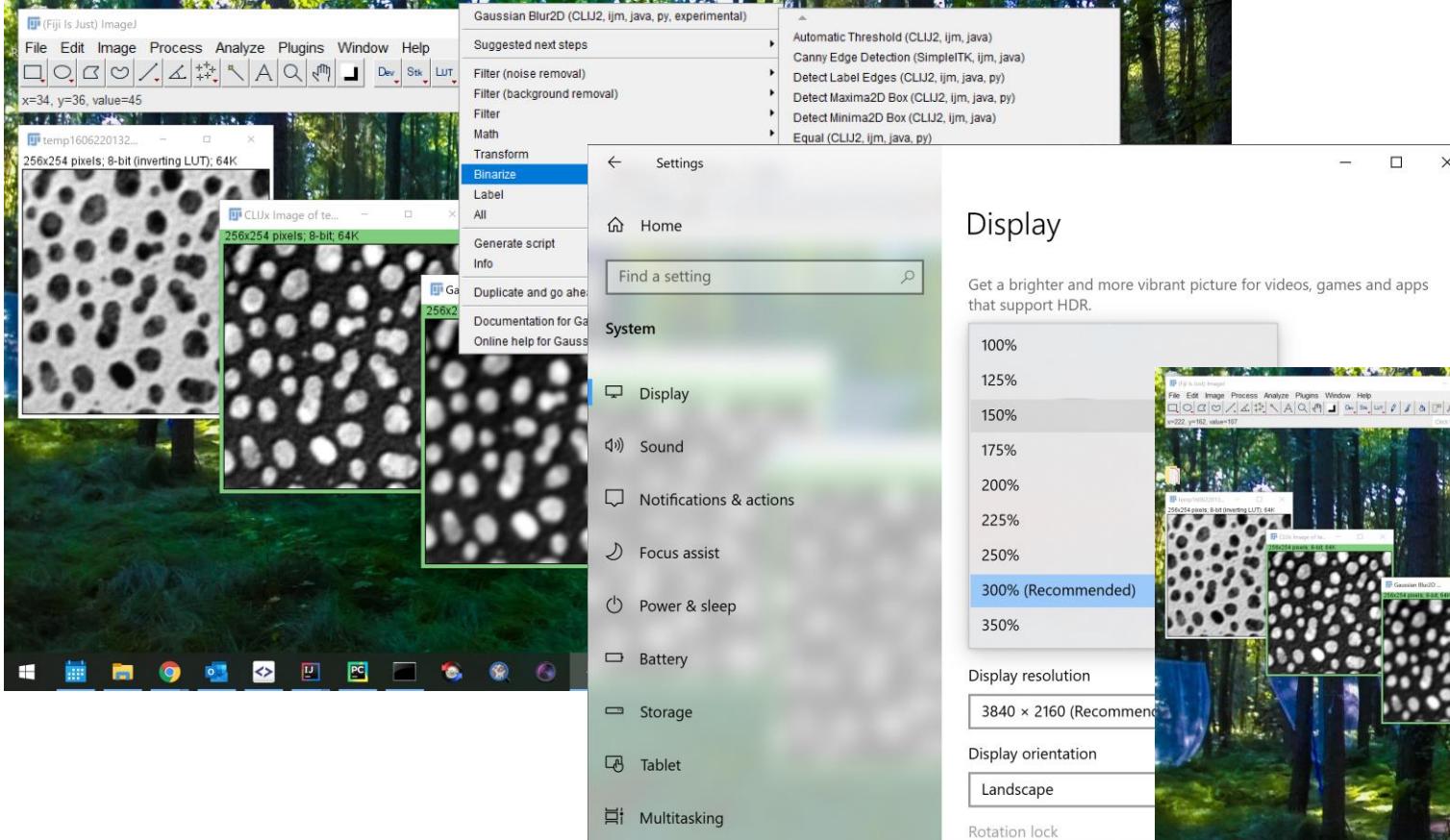
Guideline and troubleshooting here → https://github.com/clEsperanto/CLIC_prototype

Troubleshooting

Hints & troubleshooting



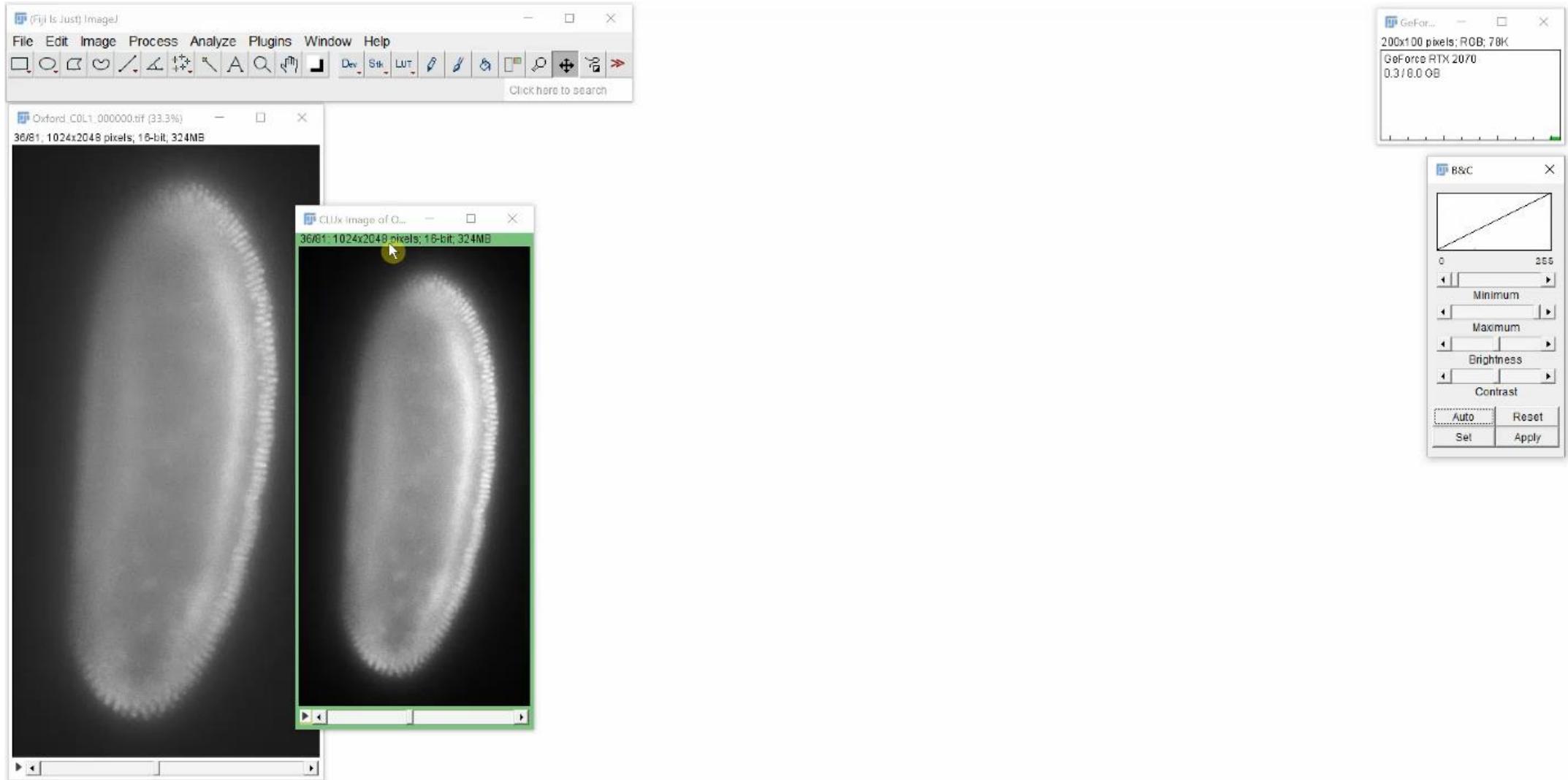
- Windows screen full of windows? Check display settings!



Hints & troubleshooting



- Out of memory? Wrap a chain of operations in a Fiji plugin!



Trouble shooting



- Check the website!

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

The screenshot shows two browser windows. The left window displays the main CLIJ2 website at https://clij.github.io. A green arrow points from the 'FAQ / support' link in the sidebar to the right window, which shows the troubleshooting page at https://clij.github.io/clij2-docs/troubleshooting. The right window has a large title 'Black images on Intel GPUs / Linux'. Below it, there is text about the issue, a command to run ('sudo apt-get install intel-opencl-icd'), and another section titled 'MacOS shows login screen after running CLIJ for a while'.

CLIJ2

GPU accelerated image processing for everyone

CLIJ2 home

This application is maintained by [haesleinhuepf](#), [Icy](#), [Matlab](#)

Imprint

Reference Cheat sheets Source code

FAQ / support

- Frequently asked questions
- CLIJ versus CLIJ2
 - CLIJ documentation (archived)
 - CLIJ BioRxiv preprint (archived)
 - What's different between CLIJ1, CLIJ2 and CLIJx?
 - CLIJ - CLIJ2 transition guide (under construction)
- Troubleshooting
- Support
- Imprint

Acknowledgements

Development of CLIJ is a community effort. We would like to thank everybody who helped developing and testing. In particular thanks to Alex Herbert (University of Sussex), Bertrand Vernay (IGBMC), Bram van den Broek (Netherlands Cancer Institute), Brenton Rongen (RCSI), Brian Northan (True North Intelligent Algorithms), Bruno Vorkel (MPI CBG), Curtis Rueden (UW-Madison LOCI), Damir Krunić (MPI CBG), Daniel J. White (GE), Eduardo Conde-Sousa (Universidade do Porto), Erick Ratamero (The Jackson Laboratory), Gaby G. Matisse (University of Michigan), Guillaume Witz (Bern University), Giovanni Cardone (MPI Biochemistry), Jean-Yves Tinevez (Institute Pasteur), John Girstmair (MPI CBG), Juergen Gluch (Fraunhofer IKT), Kota Nagahama (Acquiifer), Matthew Foley (University of Sydney), Michael Huisman (Fiji), and others.

CLIJ2

GPU accelerated image processing for everyone

CLIJ2 home

This application is maintained by [haesleinhuepf](#), [Icy](#), [Matlab](#)

Imprint

Reference Cheat sheets Source code

Black images on Intel GPUs / Linux

On an "Intel(R) HD Graphics Kabylake Desktop GT1.5" used from Ubuntu Linux 20.04 it was observed that some operations lead to empty images. Furthermore, a warning is shown on std::err Beignet: "unable to find good values for local_work_size[i], please provide\n" "local_work_size[] explicitly, you can find good values with\n" "trial-and-error method.". This issue can be solved by installing a recent Intel OpenCL ICD, e.g. with this command:

```
sudo apt-get install intel-opencl-icd
```

Furthermore, use the device "Intel HD Graphics Gen 9 NEO".

Repeated initialisation fails on AMD Vega 10

When creating CLIJ instances and closing them repeatedly, it crashes after about 40 attempts. This test allows reproducing the issue on specified hardware. Workaround: Don't close the CLIJ instance and keep working with the singleton instance.

MacOS shows login screen after running CLIJ for a while



- Ask the online!

[CLIJ does not seem to be using GPU during test](http://forum.image.sc/t/clij-does-not-seem-to-be-using-gpu-during-test/29432)

NicoDF

Hi @haesleinhuepf !

I finally got to try CLIJ on my computer. I have a large collection of images to give it a shot and test it out. I have an ASUS VivoBook S, running nVidia GeForce 940MX (I also have an in-board intel GPU).

The thing is I could not get past the initial tests. When I run a benchmarking to loops, one on GPU, one on CPU, **CLIJ takes 10+ times longer than the standard**. All the load seems to be on the CPU during the whole test.

Here's a couple of shots from the results for repeated 2° rotation (I know...)

```

blobs_b - 256x256 pixels; 32-bit; 254K
GPU_L - 256x256 pixels; 32-bit; 254K
CPU_L - 256x256 pixels; 32-bit; 254K

Log
File Edit Font
CLJ: 22231 ms
CPU: 136614 ms

success.png 1177x660 133 KB
  
```

@haesleinhuepf @StRigaud

[CLIJ does not seem to be using GPU during test](http://forum.image.sc/t/clij-does-not-seem-to-be-using-gpu-during-test/29432/4)

haesleinhuepf Community Forum Team member

NicoDF

because the CPU took sooo long

Yes, indeed! Image size was the key. Now we are talking! 😊

CLIJ loop	CPU loop
CPU 40% 3.4 GHz	CPU 3% 1.12 GHz
Memo 5.7/7.9 G	Memoria 5.5/7.9 GB (70%)
Disco 0 (C:) 0%	Disco 0 (C:) 0%
Disco 1 (D:) 0%	Disco 1 (D:) 0%
Wi-Fi E: 0 R: 0	Wi-Fi E: 0 R: 0 Kbps
GPU 0 Intel(R) HD Graphics 620 0%	GPU 0 Intel(R) HD Graphics 620 0%
GPU 1 NVIDIA GeForce 940MX 0%	GPU 1 NVIDIA GeForce 940MX 0%

I'm a little bit curious about the small size of the GPU blip in the performance graph (while CPU gets a higher, sustained hit). Still, this doesn't change the *actual* performance. 😊

I'm going to run some more tests, and I'll keep you posted.

Thanks a lot!

Nico

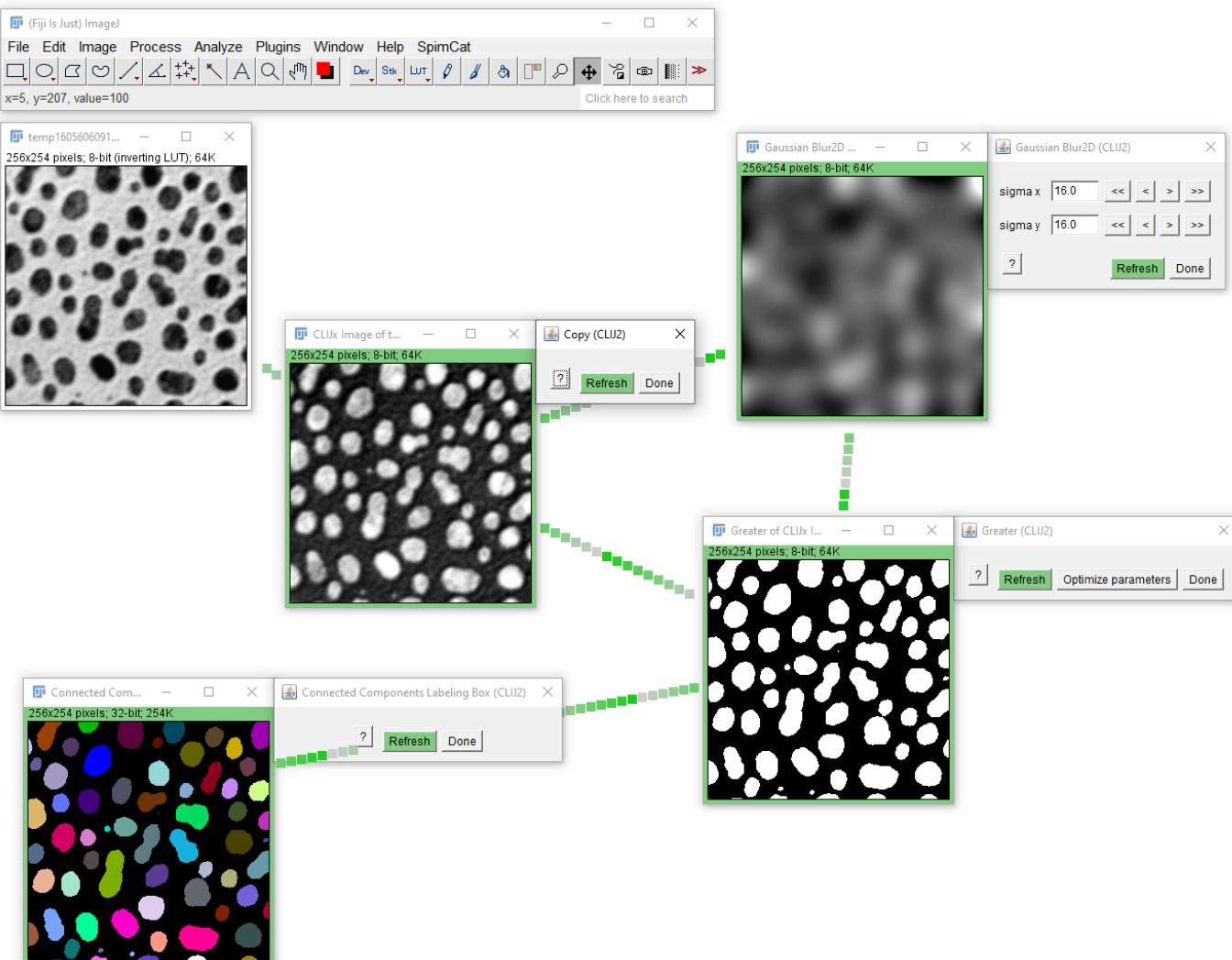
elias

Exercises

Exercise 5: Interactive workflow design



- Design a workflow for segmenting blobs.gif
- 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 clEsperanto-based Jupyter notebook and a Python script that uses Napari.

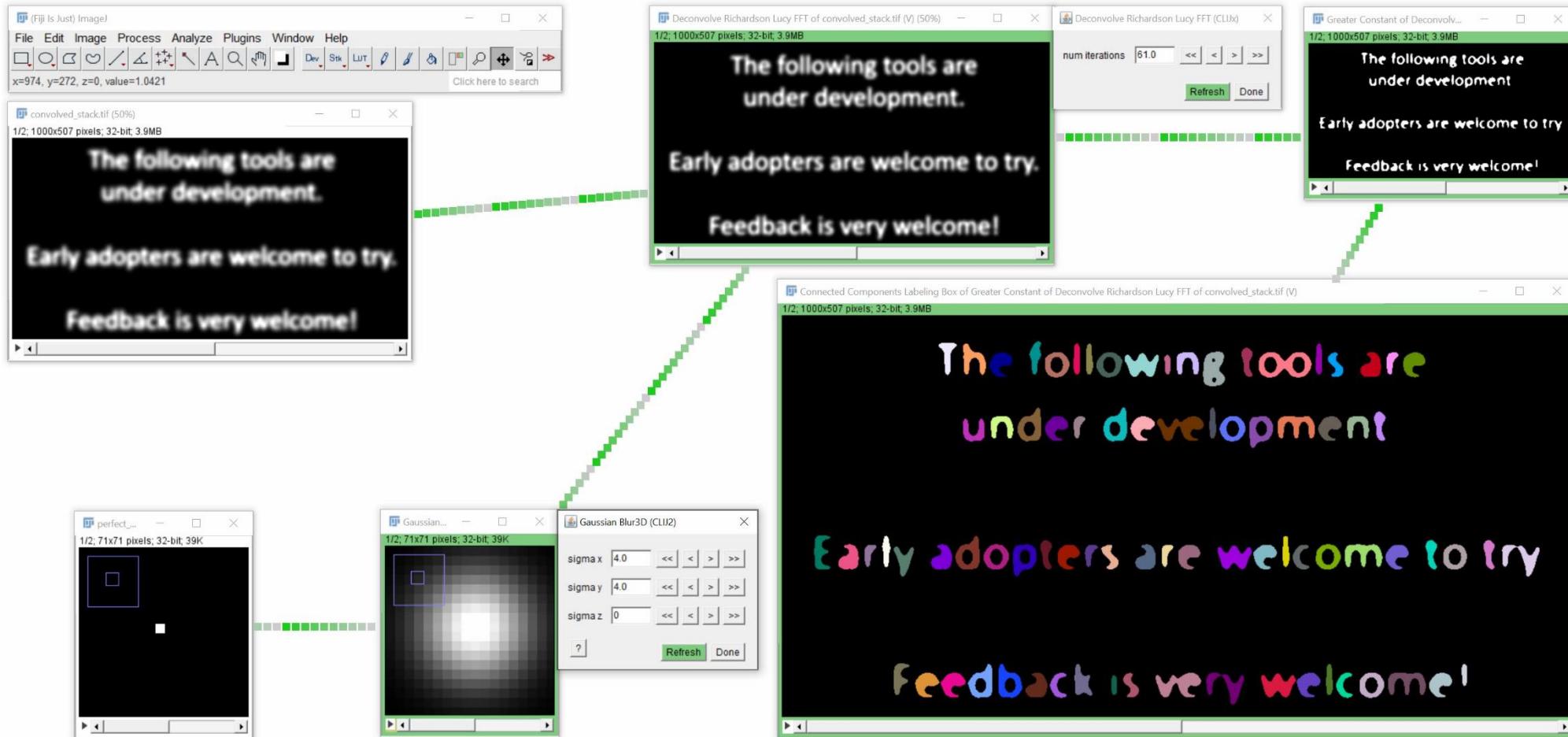


https://github.com/clEsperanto/i2k2020Tutorial_clij_clesperanto

Exercie 6: Deconvolution + Segmentation



- Reproduce the deconvolution + segmentation workflow from the beginning of this session

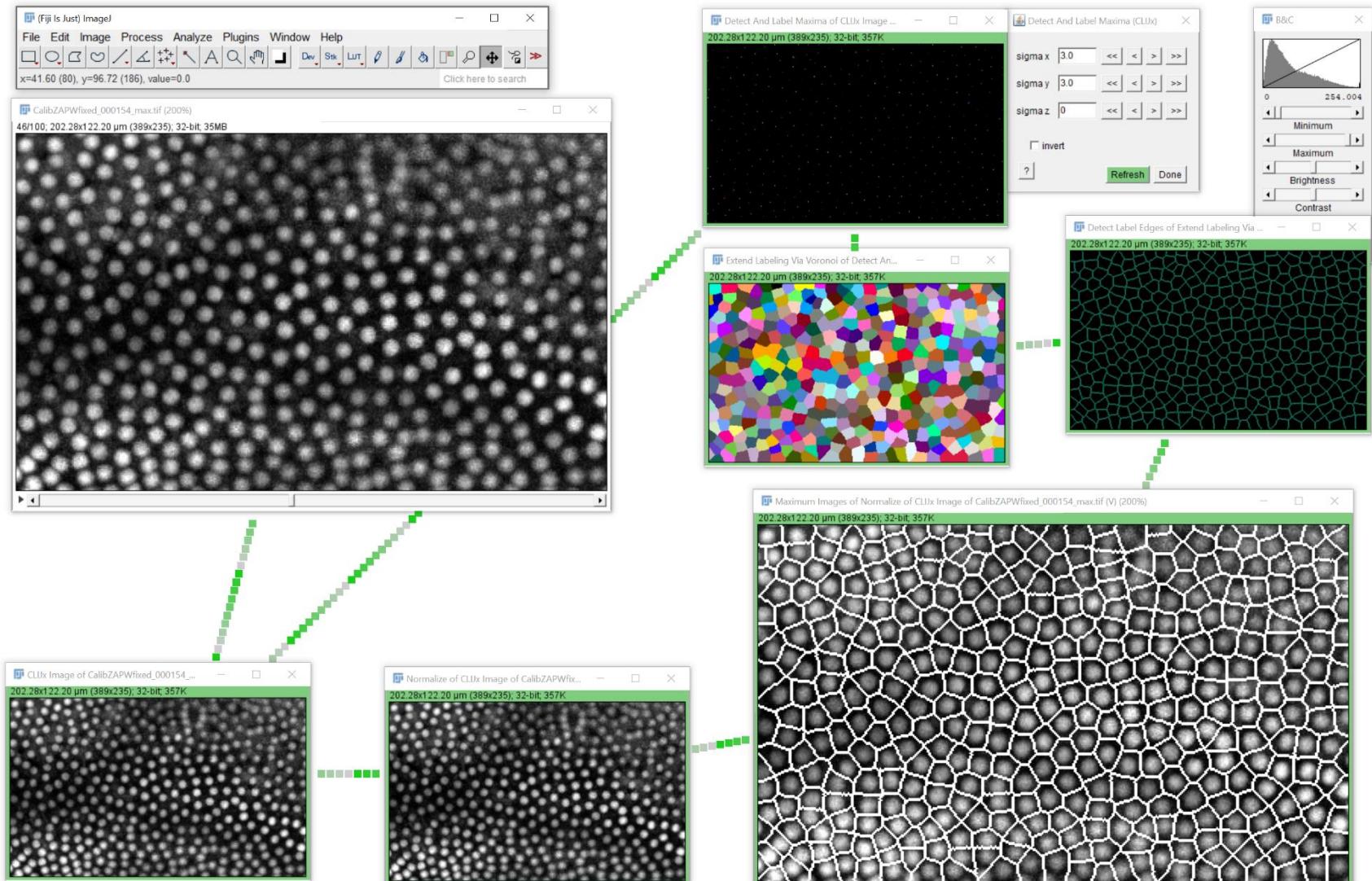


Exercise 7: Segmentation in time lapse data



- Setup a workflow for estimating cell borders between nuclei

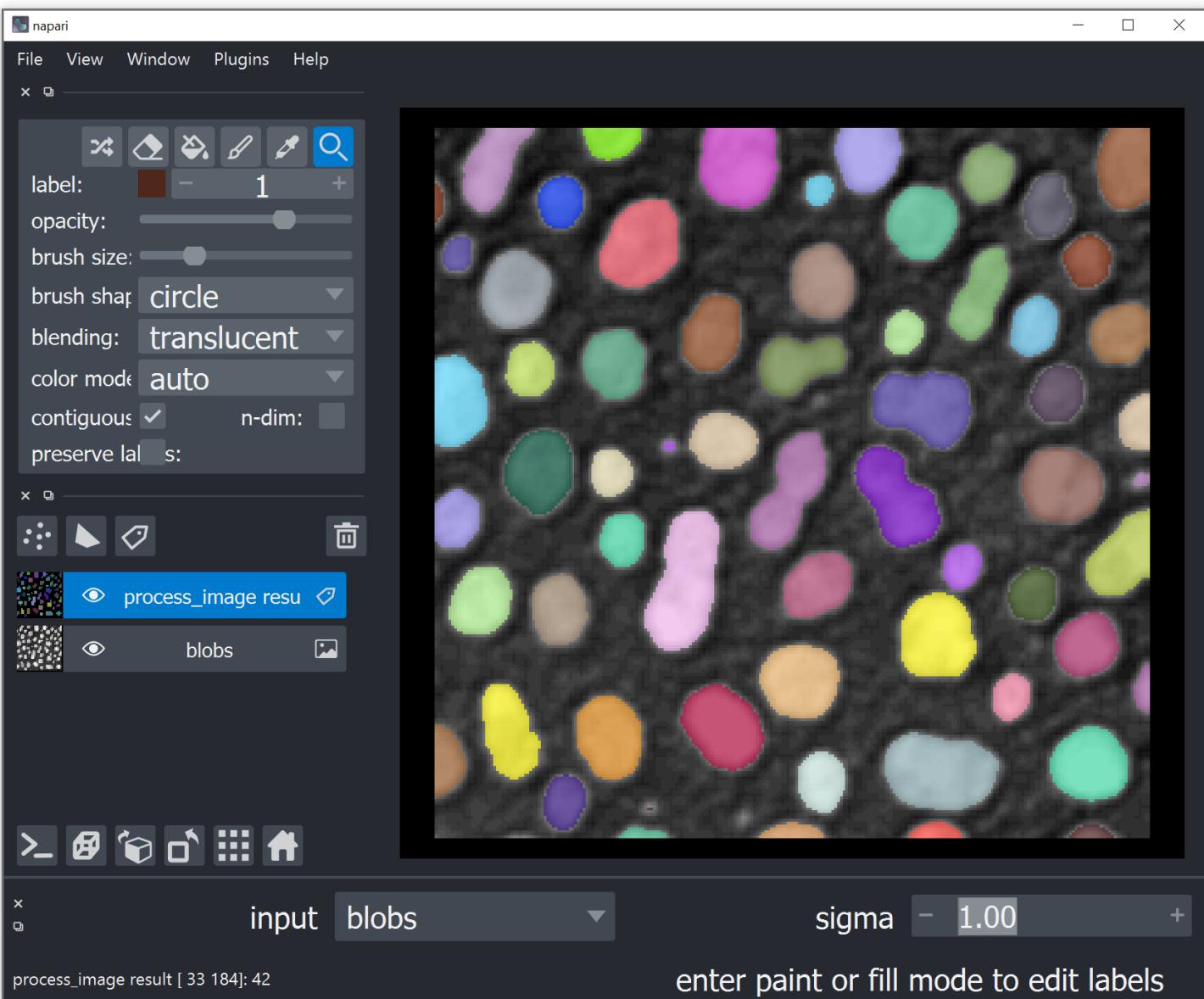
- Export an ImageJ Macro and run it.
- Note: This exercise will teach you how to deal with errors in generated macro code and how to build in a for loop to process frames individually.



Exercise 8: Napari + magicgui



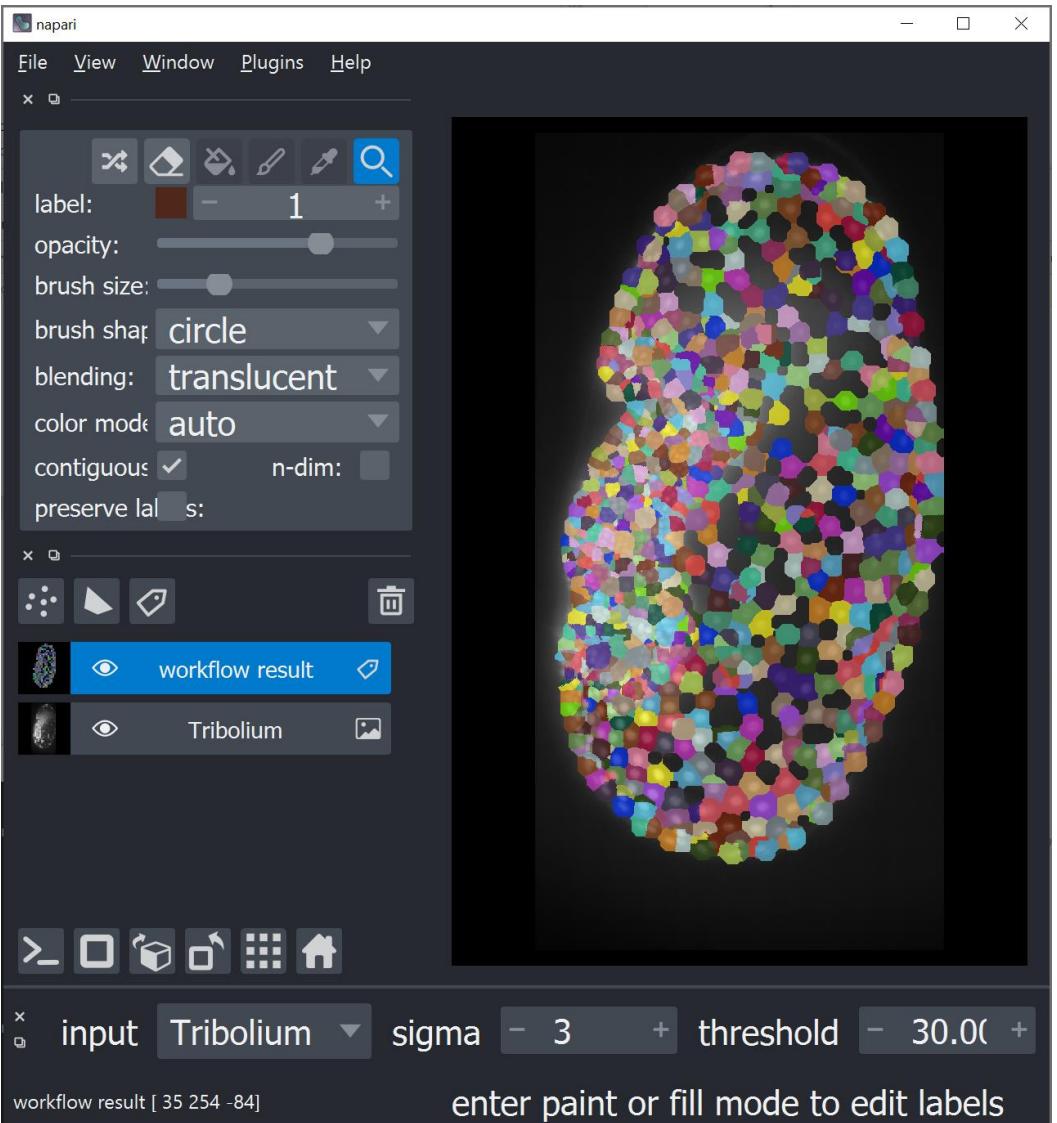
- After installing python, also install magicgui:
pip install magicgui
- Check out the minimal napari example and the reference of pycleasperanto.
- Write an interactive user interface for changing parameters while segmenting / labeling blobs.



Exercise 9: Tribolium Morphometry in Python



- Design a workflow that can segment cells starting from nuclei center positions in this Tribolium embryo
- Check out the ImageJ macro tutorial and rewrite the workflow in Python
https://clij.github.io/clij2-docs/md/tribolium_morphometry/



Exercise 10: Export to cluPath



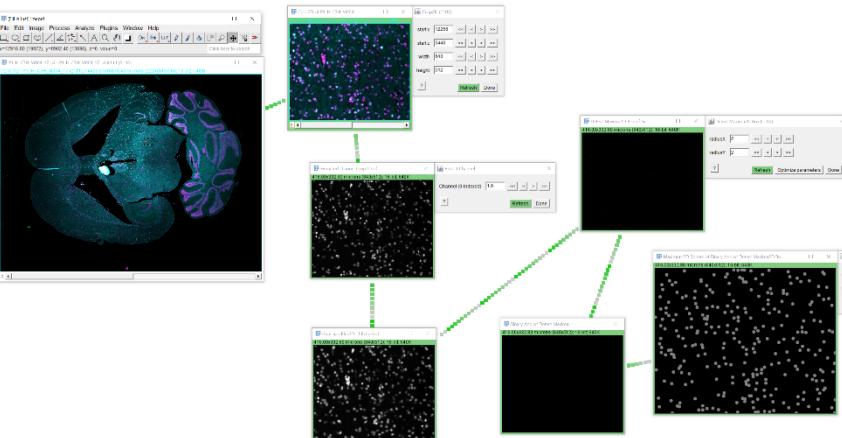
Setup a workflow for detecting nuclei in P1_H_C3H_M004_17.czi - P1_H_C3H_M004_17.czi #1.tif from

https://zenodo.org/record/4276076#.X7_7cM1KguU

(Courtesy of Theresa Suckert, OncoRay, University Hospital Carl Gustav Carus, TU Dresden, license CC-BY 4.0)

Use operations such as

- Crop 2D
- Extract Channel
- Detect Maxima 2D?
- Threshold Otsu 2D?
- Binary Weka Pixel Classifier?



Exercise 11: 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)
```

```
python flip_flop.py
Used GPU: Intel(R) UHD Graphics
Image size: (50, 1024, 1024)
C:\Users\rober\miniconda3\lib\site-packages\pyopencl\__init__.py:252: CompilerWarning: Non-empty compiler output encountered. Set the environment variable PYOPENCL_COMPILER_OUTPUT=1 to see more.
  warn("Non-empty compiler output encountered. Set the "
Flip-flop took 0.08331894874572754s
Flip-flop took 0.015999555587768555s
Flip-flop took 0.00940251350402832s
Flip-flop took 0.006979227066040039s
Flip-flop took 0.002180814743041992s
Flip-flop took 0.00948190689086914s
Flip-flop took 0.004110097885131836s
Flip-flop took 0.0s
Flip-flop took 0.009145736694335938s
Flip-flop took 0.004566669464111328s
Re-alloc took 0.07775282859802246s
Re-alloc took 0.01401066780090332s
Re-alloc took 0.01613020896911621s
Re-alloc took 0.26037096977233887s
Re-alloc took 0.36661481857299805s
Re-alloc took 0.3394293785095215s
Re-alloc took 0.43366098403930664s
Re-alloc took 0.5396988391876221s
Re-alloc took 0.5865137577056885s
Re-alloc took 0.6842575073242188s

(te_oki) C:\structure\code\i2k2020_tutorial_clij_clesperanto\exercise11>
```

Exercise 12: CLIC script compile and run



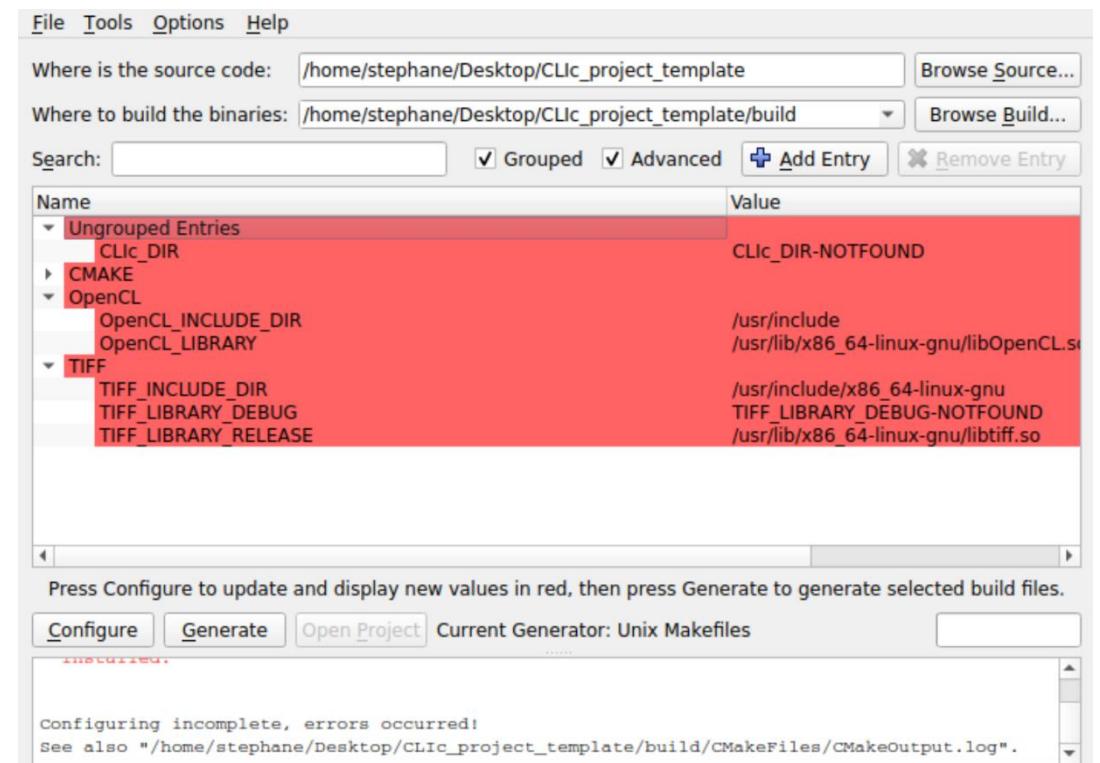
Let's try to configure and compile your first clEsperantoC script code!

Download the CLIC API and the CLIC template project:

https://github.com/clEsperanto/CLIC_prototype

https://github.com/StRigaud/CLIC_project_template

- Use CMake to configure and generate a C++ project
- Compile the empty CLIC template project
- Add a 2D Gaussian Blur filter



Exercise 13: Count nuclei with CLIC

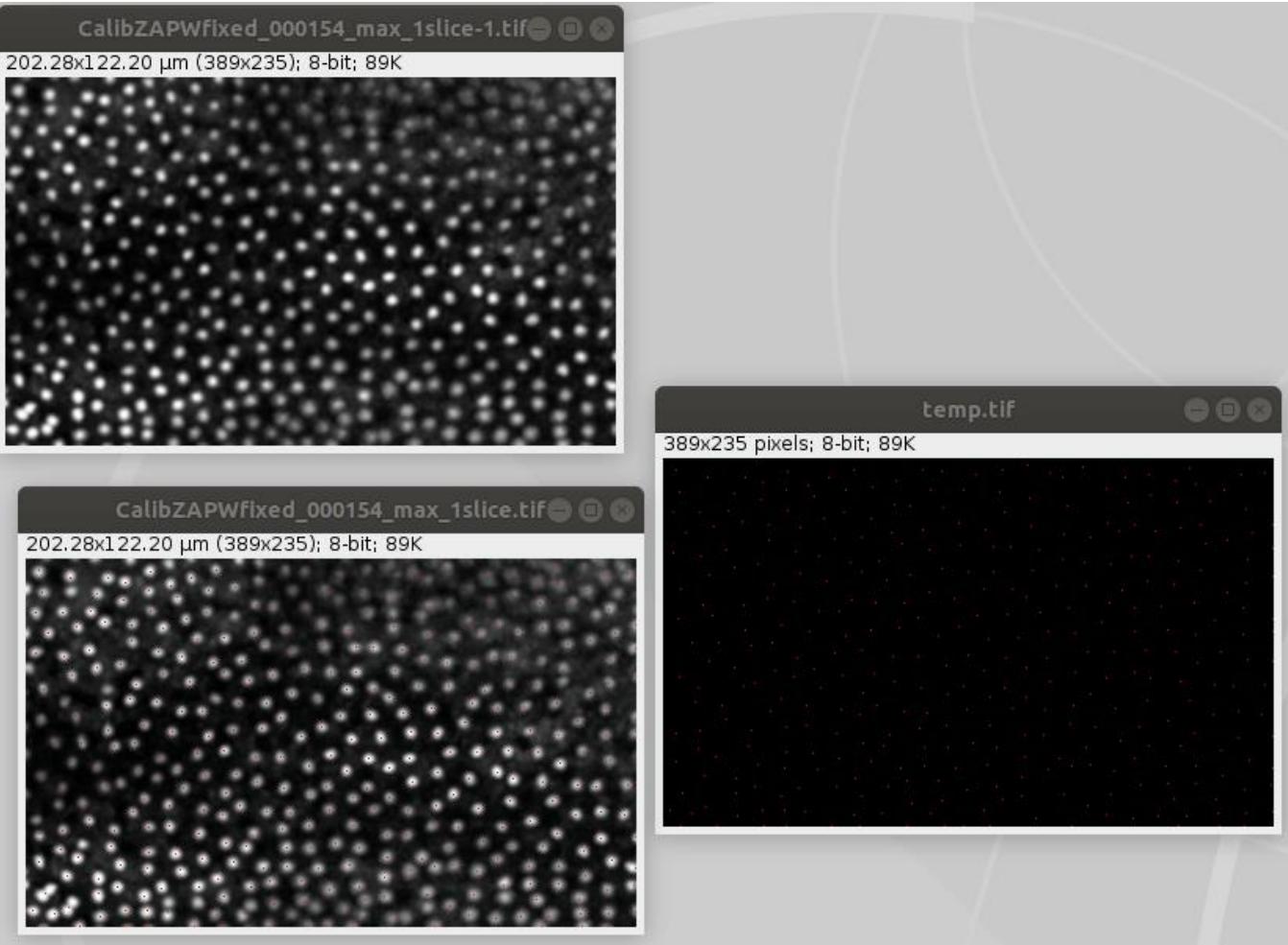


Why make it simple when you can make it complicate!

Reusing the CLIC project from Exercise 12, make a CLIC script to count the number of cell in an image

- Build a C++ workflow
- Manage input and output buffer
- Use CLIjx-Assistant for help/starter

(we should be around 362 nuclei)



Exercise 14: User 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 CLIJx Usage Miner GitHub page displays information about the plugin, including its purpose, how to extract usage statistics, and a screenshot of the Fiji interface showing a text file with usage statistics.

Packages
No packages published
Publish your first package

Environments 1
github-pages (Active)

Languages
Java 100.0%

How to extract usage statistics

Download and install Fiji, activate the CLIJ update site and the download CLIJx usage-miner plugin and put the jar file in the '/plugins/' directory of your Fiji.

Start Fiji, and execute the menu Plugins > ImageJ on GPU (CLIJx) > Options > CLIJ Usage Miner. Please select a folder of imagej macros which use CLIJ. Such a file will open up:

(Fiji Is Just) Image

File Edit Image Process Analyze Plugins Window Help SpimCat

Stacks Menu

New_ File Edit Language Templates Tools Tabs Options

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
```

Exercise 15: clEsperantoJ: Camel or Snake?

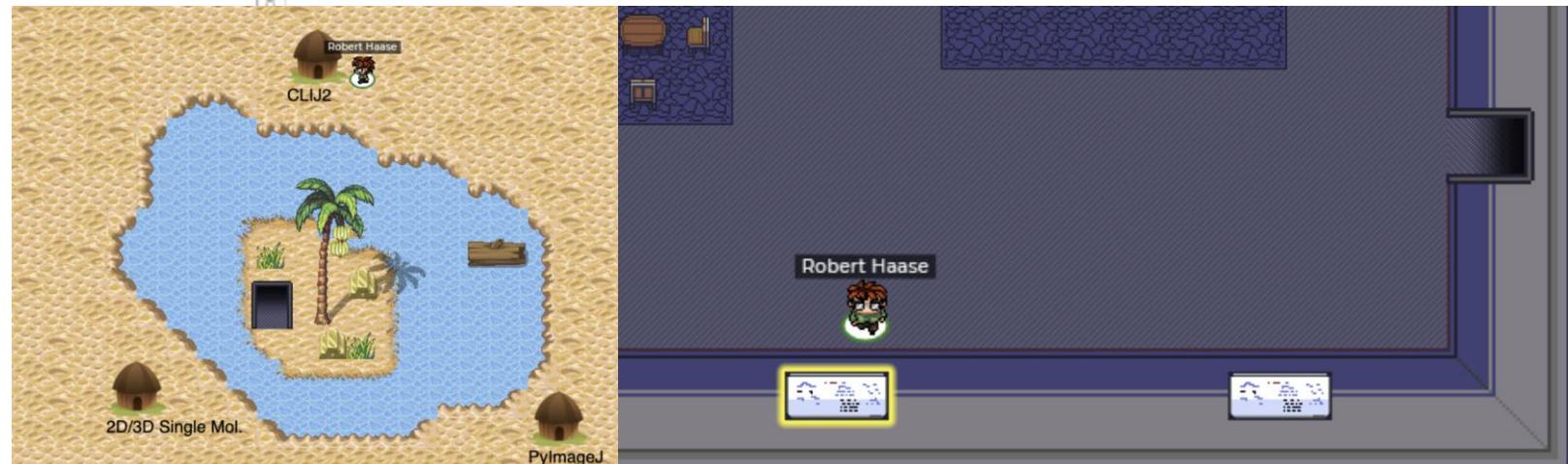


- Please try CamelCase snake_case coding using clEsperantoJ in Fiji Java/Jython/Groovy.
- After the session: Go to the CLIJ2 hut in gather.town and place your vote!

```
1 #
2 import net.clesperanto.javaprotoype.Camel as cle
3 from ij import IJ
4
5 maximum_distance = 72.0
6
7 # Load image from disc
8 imp = IJ.openImage("C:/structure/data/blobs.tif")
9 image1 = cle.push(imp);
10 image2 = cle.create(image1)
11 image3 = cle.create(image2.getDimensions(), cle.Float )
12 image4 = cle.create(image3)
13
14 # image processing
15 cle.thresholdOtsu(image1, image2)
16 cle.connectedComponentsLabelingBox(image2, image3)
17 cle.drawMeshBetweenProximalLabels(image3, image4, maximum_distance)
18
19 # show result
20 cle.pull(image4).show();
21
22 # cleanup
23 cle.clear()
```



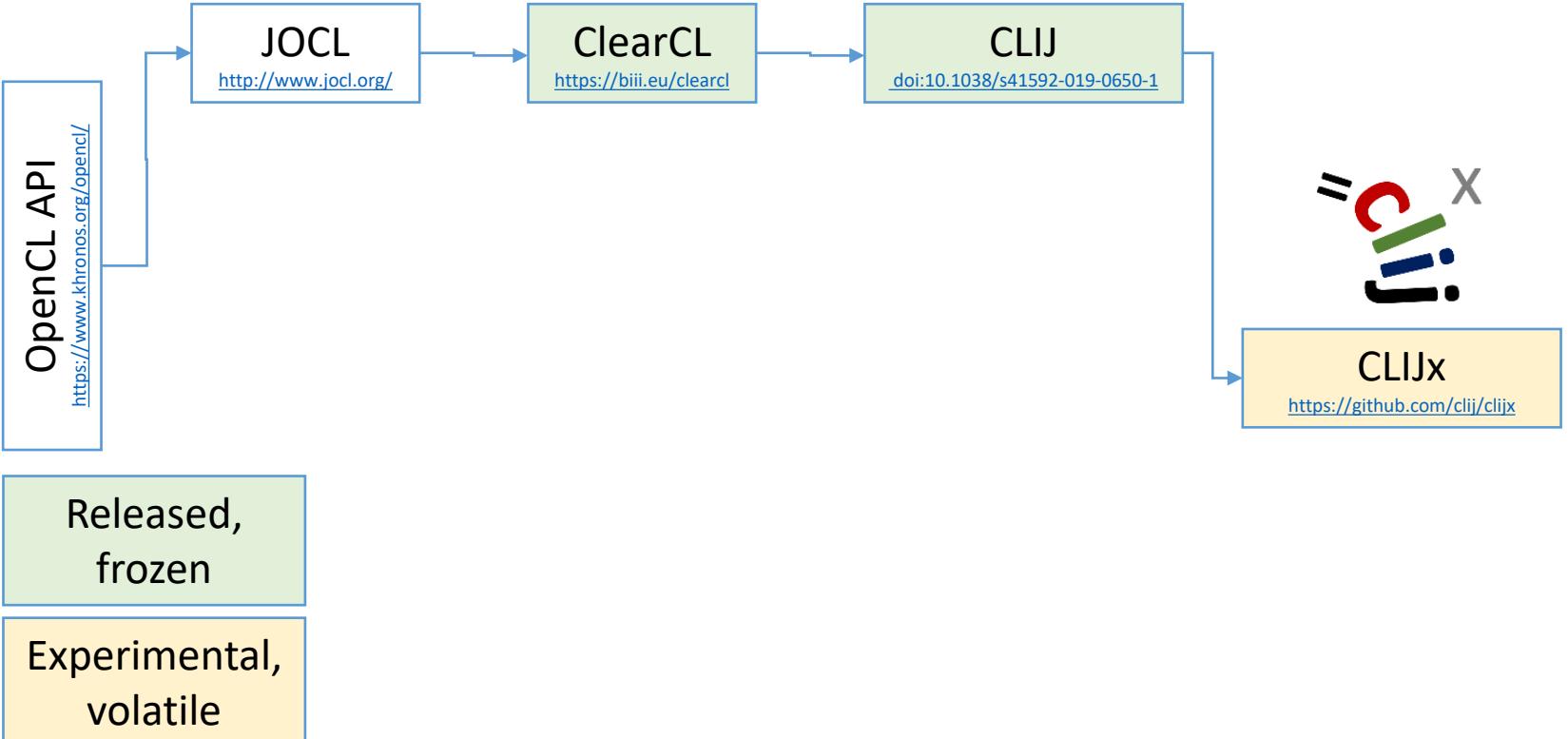
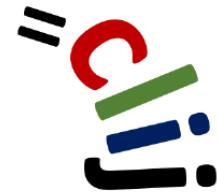
```
1 #
2 import net.clesperanto.javaprotoype.Snake as cle
3 from ij import IJ
4
5 maximum_distance = 72.0
6
7 # Load image from disc
8 imp = IJ.openImage("C:/structure/data/blobs.tif")
9 image1 = cle.push(imp);
10 image2 = cle.create(image1)
11 image3 = cle.create(image2.getDimensions(), cle.Float )
12 image4 = cle.create(image3)
13
14 # image processing
15 cle.threshold_otsu(image1, image2)
16 cle.connected_components_labeling_box(image2, image3)
17 cle.draw mesh between proximal labels(image3, image4, maximum_distance)
18
```



Summary



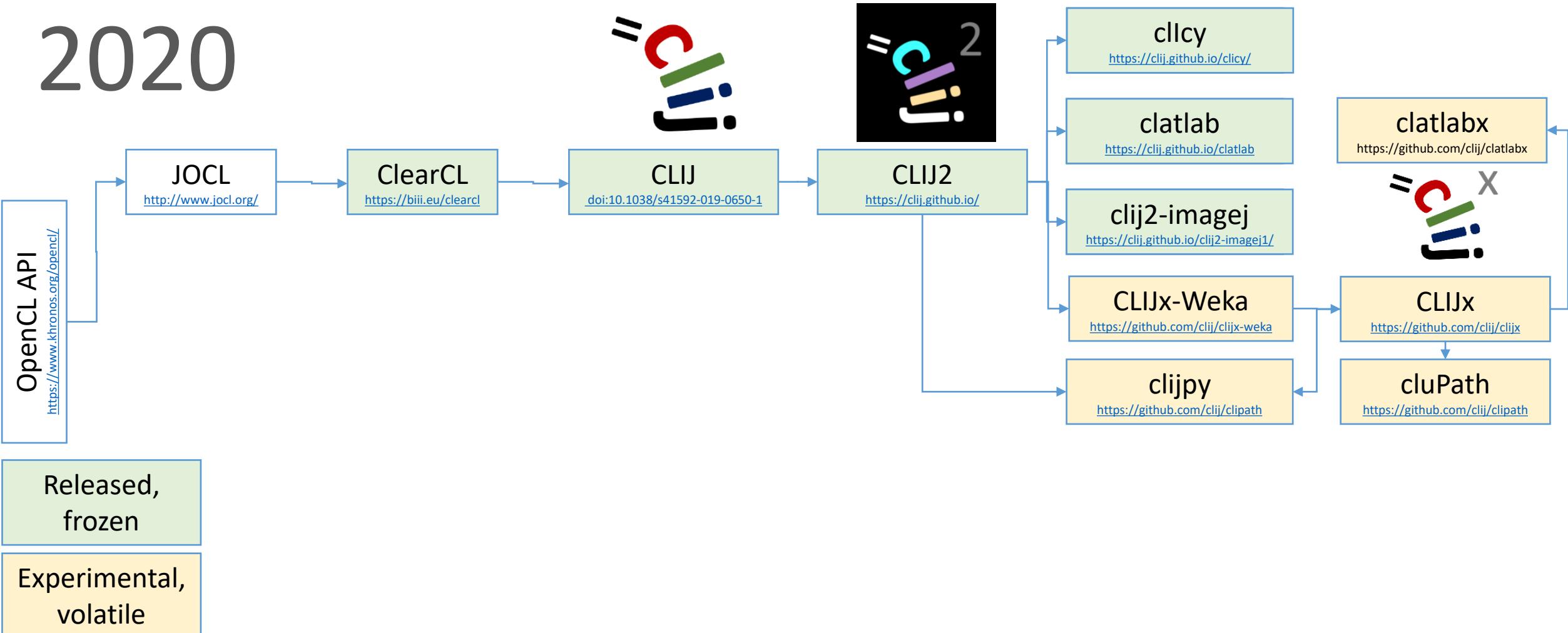
2019



clEsperanto lineage tree



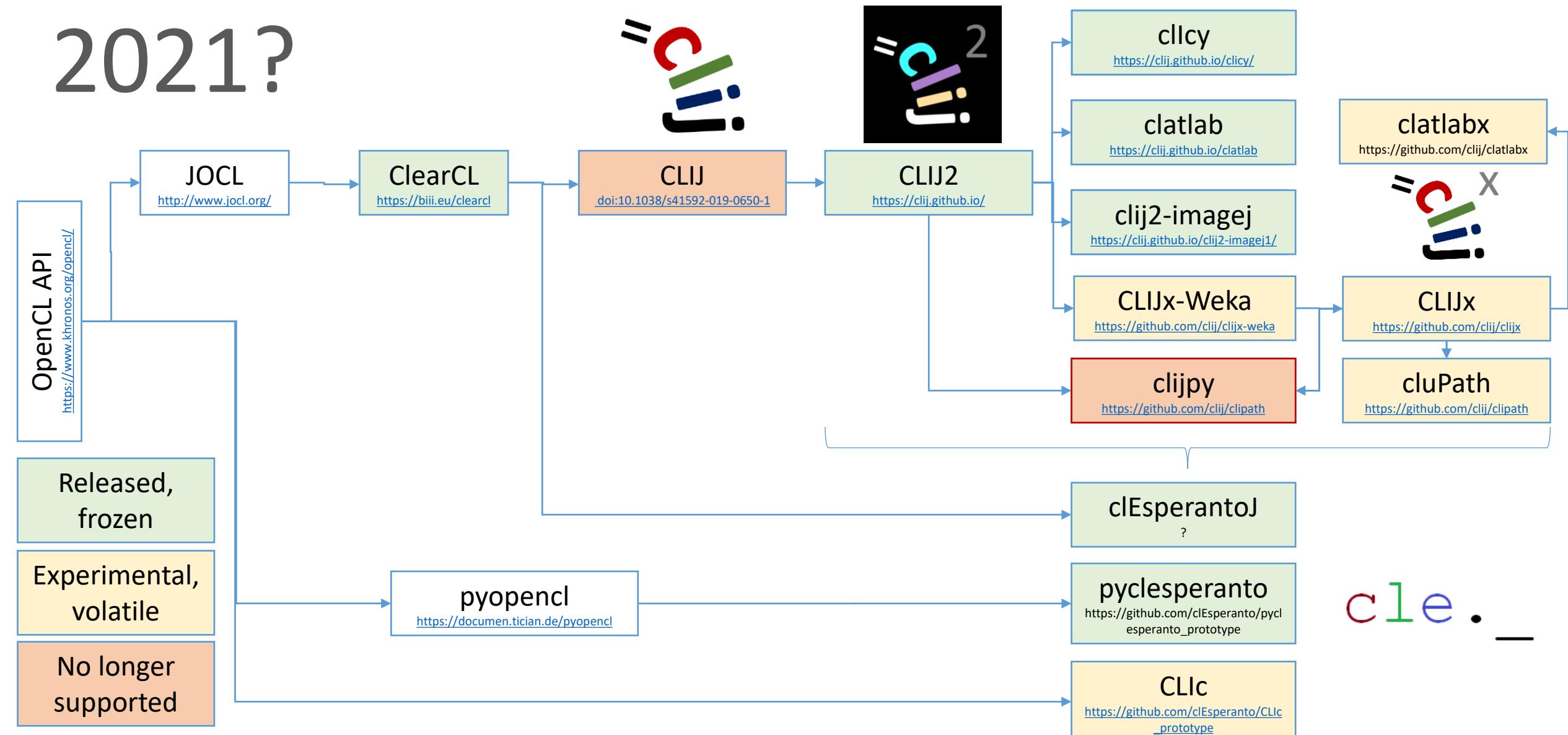
2020



clEsperanto lineage tree



2021?



Last but not least



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

The image shows a side-by-side comparison of two research outputs. On the left is a screenshot of a **Nature Methods** article titled "CLIJ: GPU-accelerated Image Processing for everyone". The article is a correspondence published in **Nature Methods** 17, 5–6(2020), with authors Robert Haase, Loic A. Royer, Dibrov, Uwe Schmidt, Martin Weinert, and Eugene W. Myers. The abstract discusses GPU-accelerated image processing, mentioning variables, for-loops, and workflows. On the right is a screenshot of a **bioRxiv** preprint titled "Interactive design of GPU-accelerated Image Data Flow Graphs and cross-platform deployment using multi-lingual code generation". The preprint is submitted to the **Cold Spring Harbor Laboratory** and lists authors Robert Haase, Akanksha Jain, Stéphane Rigaud, Daniela Vorkel, Pradeep Rajasekhar, Theresa Suckert, Talley J. Lambert, Juan Nunez-Iglesias, Daniel P. Poole, Pavel Tomancak, and Eugene W. Myers. Both pages include navigation links like Abstract, Full Text, Info/History, Metrics, and Preview PDF.

nature > nature methods > correspondence > article

nature meth Cornell University the Sir

Correspondence | Published: 18 CLIJ: GPU-accelerated Image Processing for everyone

Robert Haase, Loic A. Royer, Dibrov, Uwe Schmidt, Martin Weinert, Eugene W. Myers

Nature Methods 17, 5–6(2020)

arXiv.org > cs > arXiv:2008.11799

Computer Science > Mathematical

[Submitted on 26 Aug 2020]

GPU-accelerating Image Processing

Daniela Vorkel, Robert Haase

This chapter introduces GPU-accelerated image processing. Core concepts such as variables, for-loops, and workflows. We present in a step-by-step

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)

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 Rajasekhar, 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

Thank you! 😊

Acknowledgements



Akanksha Jain
(Treutlein lab)
@jain_akanksha



Dani Vorkel
(Myers lab)
@happifocus



Theresa Suckert
(Krause lab,
Oncoray)



Pradeep Rajasekhar
(Poole lab, Monash U)
@pr4deepr



Talley J. Lambert
(HMS)
@TalleyJLambert



Juan Nunez-Iglesias
(Monash)
@jnuneziglesias



Pavel Tomancak
(CSBD / MPI-CBG)
@PavelTomancak



Gene Myers
(CSBD / MPI-CBG)
@TheGeneMyers



Jean-Yves Tinevez
(Institut Pasteur)
@jytinevez



Florian Jug
(CSBD / MPI-CBG)
@florianjug



Szabolcs Horvát
(Modes lab)
@schorvat



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 (University 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)

Jean-Yves Tinevez (Institut Pasteur)

Johannes Girstmair (MPI CBG)

Juergen Gluch (Fraunhofer IKTS)

Kisha Sivanathan (Harvard Medical School)

Kota Miura

Laurent Thomas (Acquierer)

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 (Institut Pasteur)

Tanner Fader (UNC-Chapel Hill)

Thomas Irmer (Zeiss)

Tobias Pietzsch (MPI-CBG)

Uwe Schmidt (MPI CBG)

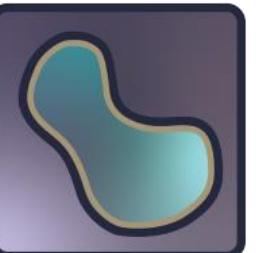
Wilson Adams (VU Biophotonics)

MPI CBG Core Facilities

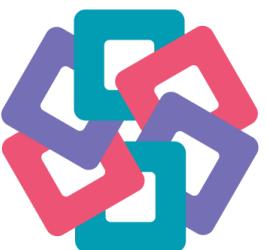
- Advanced Imaging Facility
- Light Microscopy Facility
- Scientific Computing
- IT Department



<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 - EXC2068 - Cluster of Excellence Physics of Life of TU Dresden.

Summary



- GPU-acceleration works best with
 - workflows with many steps between push and pull
 - processing many images
 - on dedicated GPUs.
- IDFGs work best if you
 - browse suggestions
 - move the windows
 - use Fiji search!

The screenshot shows three windows side-by-side. On the left is the CLIJ2 documentation page for 'Neighbors of neighbors' (CLIJ2), featuring the CLIJ2 logo and a brief description. In the center is the PyPI page for 'pyclesperanto-prototype', showing the package details and a 'pip install' command. On the right is the 'CLIC' interface, which is a prototype for a multi-language framework. It includes a control panel with settings like 'label', 'opacity', and 'brush size', and a main canvas displaying a colorful segmented image. Below the canvas are input parameters ('input: Tribolium', 'sigma: -3', 'threshold: -30.0') and a status bar.

This is a screenshot of the 'CLIJ2 cheat sheet: ImageJ macro II' document. It provides a quick reference for GPU-accelerated image processing in Fiji. The top section shows the 'CLIJ2' logo and the 'CENTER FOR SYSTEMS BIOLOGY DRESDEN' logo. Below is a table with operations, parameters, results, dimensions, and examples. A 'Filters' sidebar on the left allows for filtering operations. The table rows include:

Operation	Parameters	Result	Dim	Examples
Gaussian blur	, 10, 10		2D 3D	Ext.CLIJ2_gaussianBlur2D(input, result, sigmaX, sigmaY);
Difference of Gaussian	, 2, 2, 20, 20		2D 3D	Ext.CLIJ2_differenceOfGaussian2D(input, result, sigmaX, sigmaY, sigma2x, sigma2y);
Invert			2D 3D	Ext.CLIJ2_invert(input, result);
Laplace			2D 3D	Ext.CLIJ2_laplaceBox(input, result);
Mean	, 5, 5		2D 3D	Ext.CLIJ2_mean2DBox(input, result, radiusX, radiusY);
Median	, 5, 5		2D 3D	Ext.CLIJ2_medianSliceBySliceBox(input, result, radiusX, radiusY);
Minimum	, 5, 5		2D 3D	Ext.CLIJ2_minimum2DBox(input, result, radiusX, radiusY);
Maximum	, 5, 5		2D	Ext.CLIJ2_maximum2DBox(input, result, radiusX, radiusY);

This screenshot shows two ImageJ macro editors. The left one displays a script for 'tribolium_morphometry.ijm' with several comments and function calls related to distance matrices and neighbor analysis. The right one shows the 'Extend Labeling Via Voronoi' interface, which is a plugin for CLIJ2. It includes a preview window showing a segmented image, a 'Suggested next steps' list, and various processing options like 'Transform', 'Binarize', and 'Label measurements'. A status bar at the bottom indicates '202.28x122.20 µm (389x235); 32-bit; 357K'.