Bigstream Alignment Library for the RNR-ExM Challenge

ISBI 2023

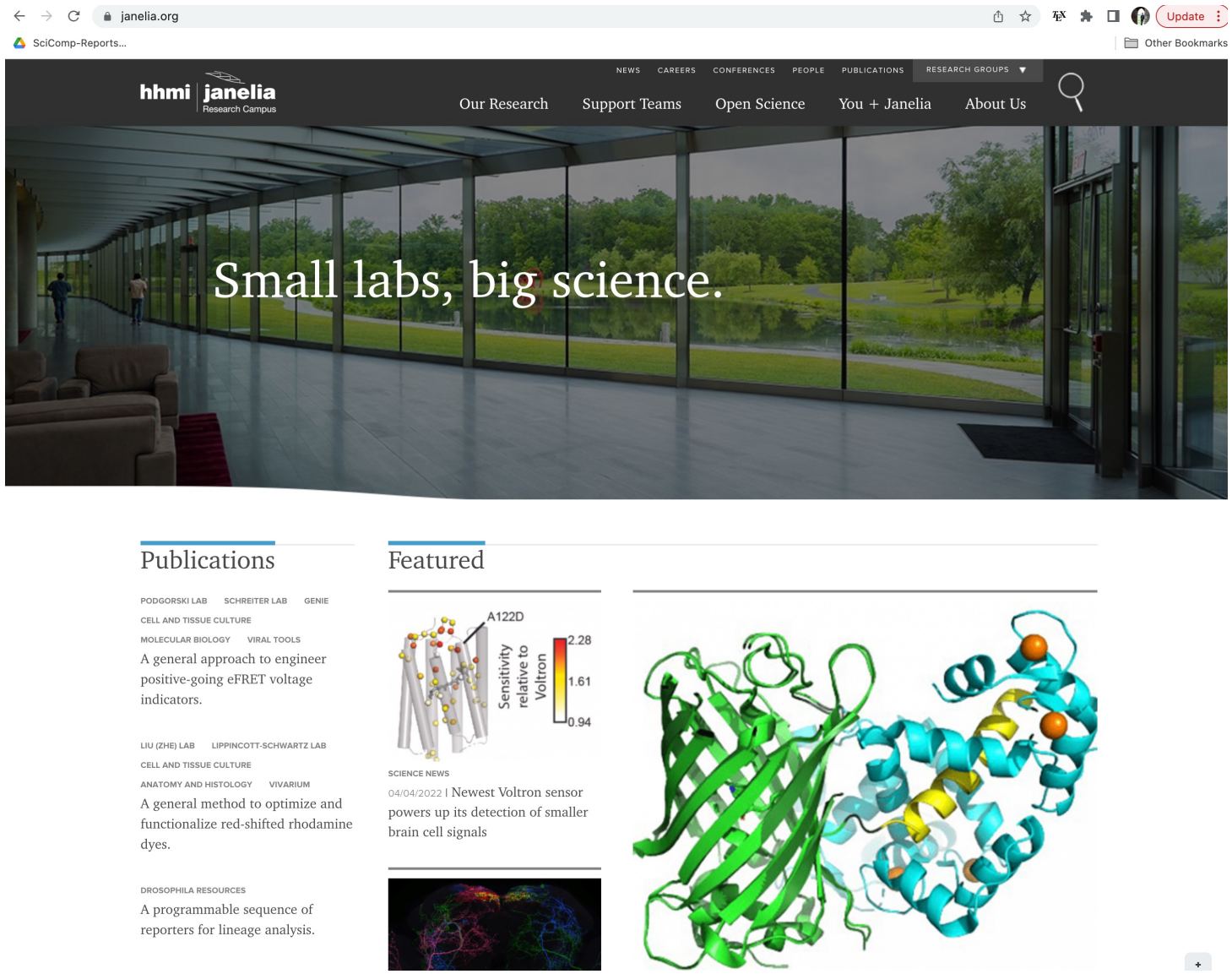
Greg M. Fleishman

Scientific Software Engineer, Janelia Research Campus HHMI

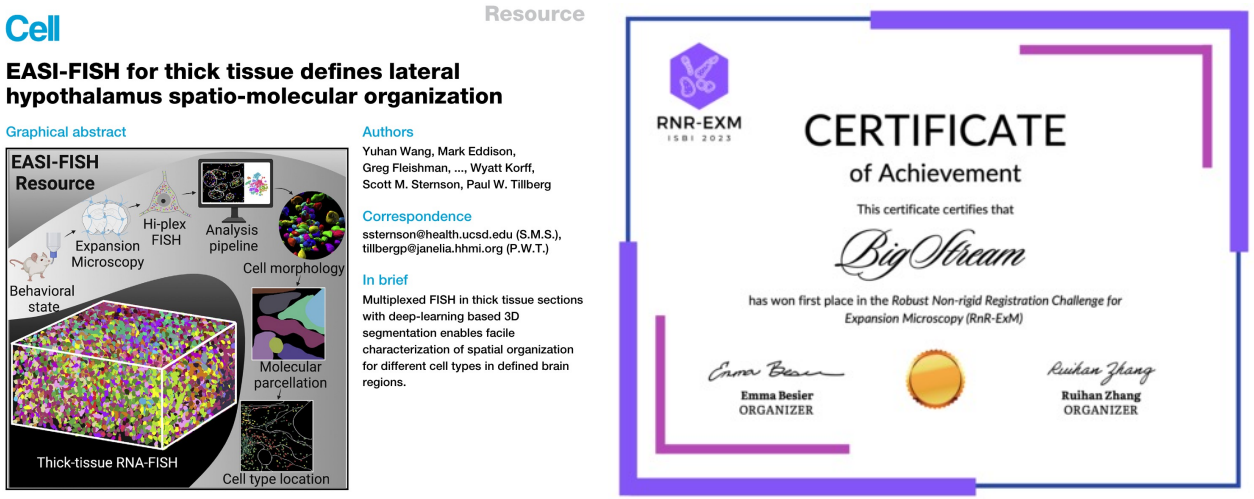
Janelia Research Campus - HHMI

- Physical campus of the Howard Hughes Medical Institute
- Focus on Neuroscience and Cell biology
- Focus on new technology development
  - Microscopy, molecular tools, **computational methods**

- Scientific Computing Department
  - Core facility providing computational expertise to labs



What is Bigstream?



Cell

Resource

EASI-FISH for thick tissue defines lateral hypothalamus spatio-molecular organization

Graphical abstract

Authors
Yuhan Wang, Mark Eddison, Greg Fleishman, ..., Wyatt Korff, Scott M. Sternson, Paul W. Tillberg

Correspondence
ssternson@health.ucsd.edu (S.M.S.), tillbergp@janelia.hhmi.org (P.W.T.)

In brief

Multiplexed FISH in thick tissue sections with deep-learning based 3D segmentation enables facile characterization of spatial organization for different cell types in defined brain regions.

RNR-EXM ISBI 2023

CERTIFICATE of Achievement

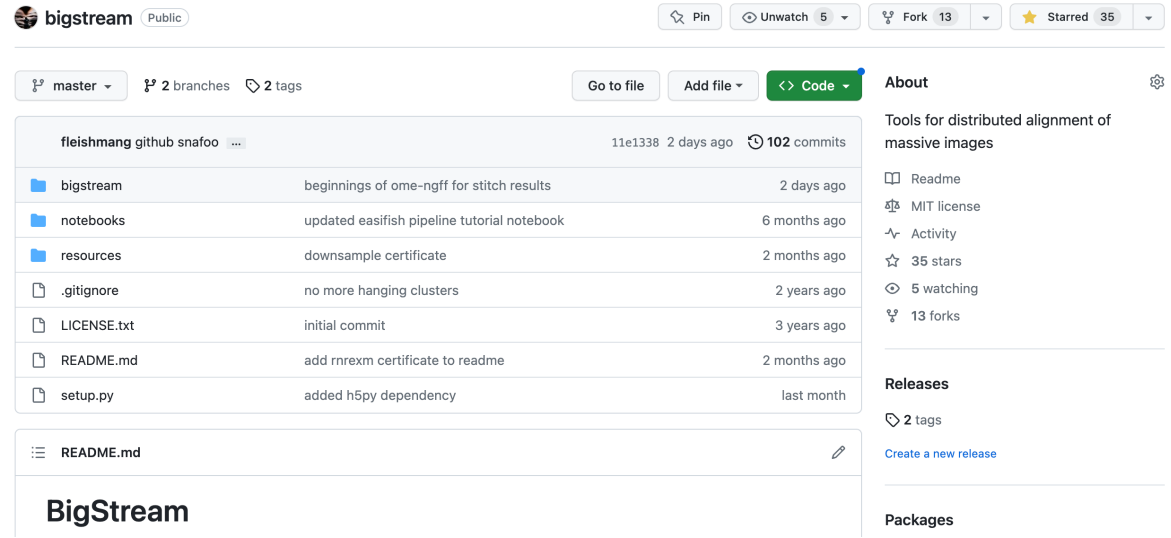
This certificate certifies that

BigStream

has won first place in the *Robust Non-rigid Registration Challenge for Expansion Microscopy (RnR-ExM)*

Emma Besier ORGANIZER

Ruihan Zhang ORGANIZER



bigstream Public

Pin | Unwatch 5 | Fork 13 | Starred 35

master | 2 branches | 2 tags | Go to file | Add file | Code

fleishmang github snafoo ... 11e1338 2 days ago 102 commits

bigstream	beginnings of ome-ngff for stitch results	2 days ago
notebooks	updated easifish pipeline tutorial notebook	6 months ago
resources	downsample certificate	2 months ago
.gitignore	no more hanging clusters	2 years ago
LICENSE.txt	initial commit	3 years ago
README.md	add rnrexm certificate to readme	2 months ago
setup.py	added h5py dependency	last month

README.md

BigStream

About

Tools for distributed alignment of massive images

Readme
MIT license
Activity
35 stars
5 watching
13 forks

Releases

2 tags

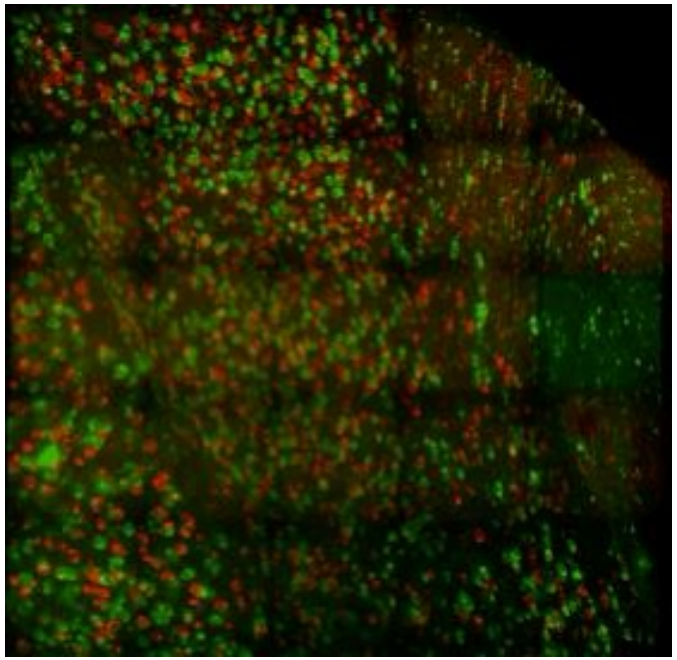
Create a new release

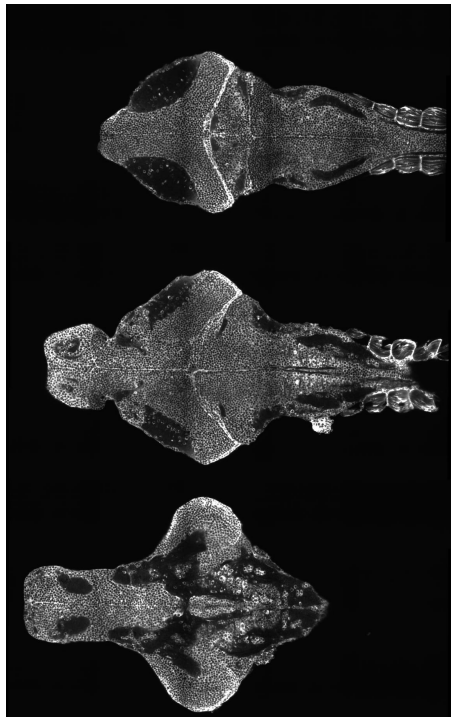
Packages

- Library of tools for 3D registration
- Extremely customizable pipelines
- Scales to clusters and images too-large-for-memory

- Used in several publications already
- Currently being used in projects at:
  Janelia, UCLA, MIT, CIBR, E11 Bio, Harvard, Allen institute
- Used on data from C. elegans, Mouse, and Zebrafish

- **Real contribution**: Combining SimpleITK and Dask to scale all SITK registrations; and the logic to stitch things back together seamlessly.



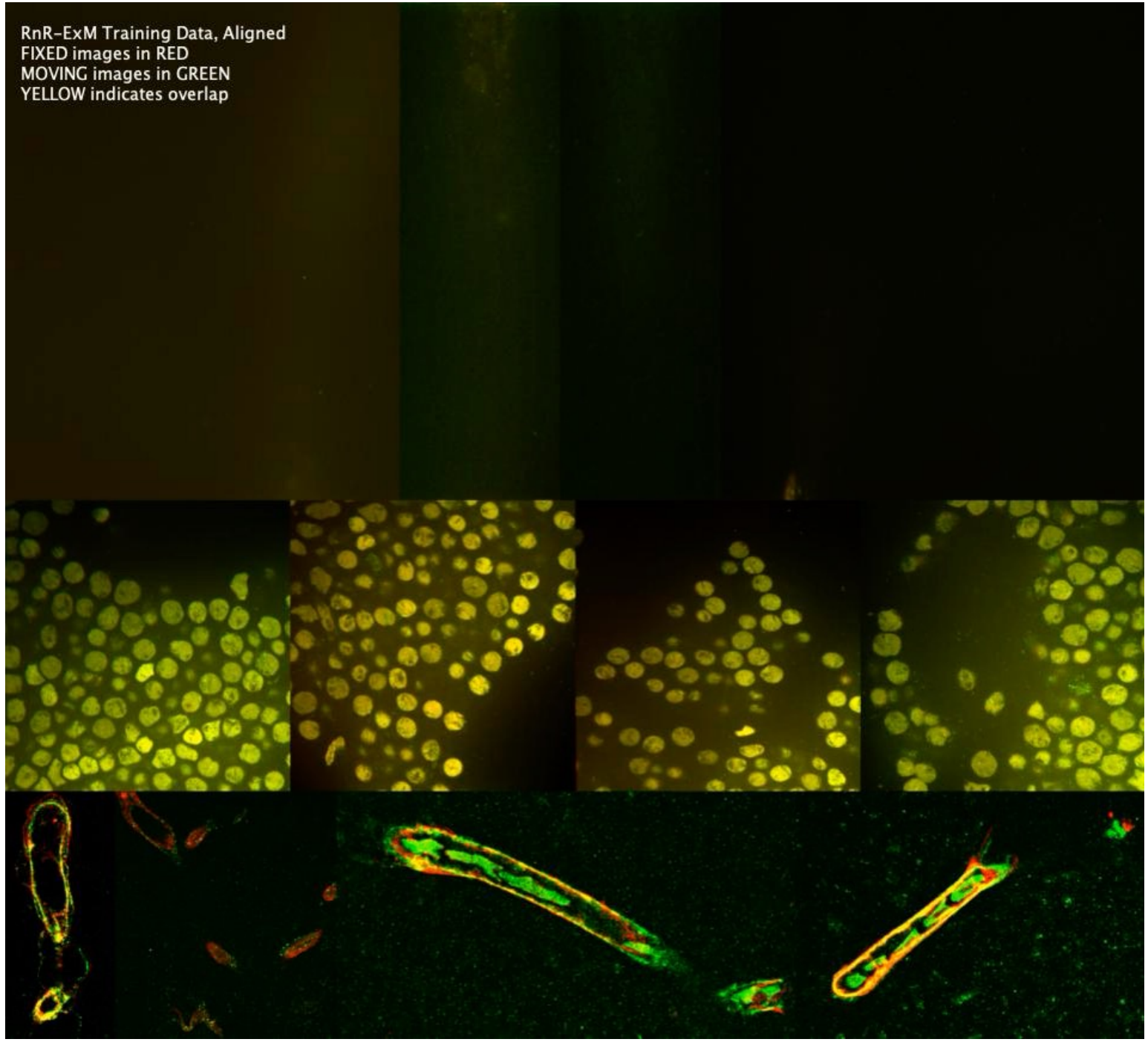


Training Data: aligned

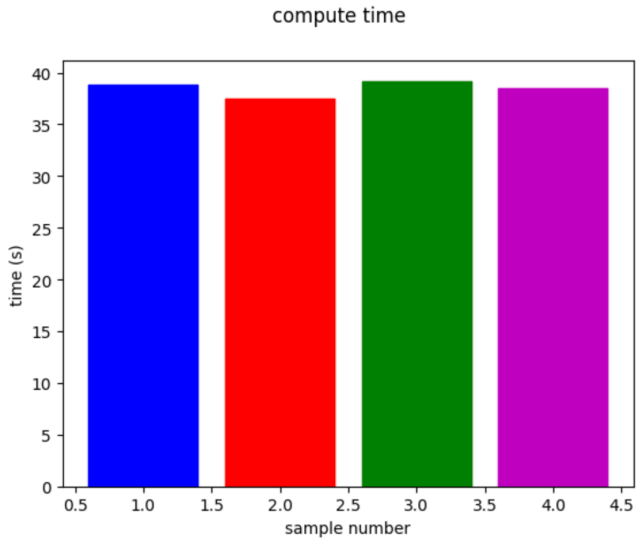
C. elegans

Zebrafish

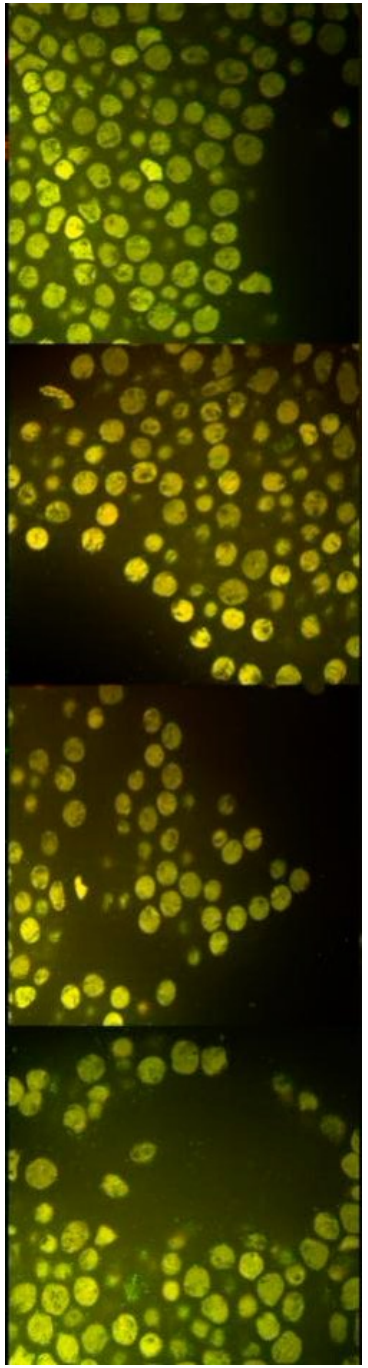
Mouse



Training Data: Zebrafish

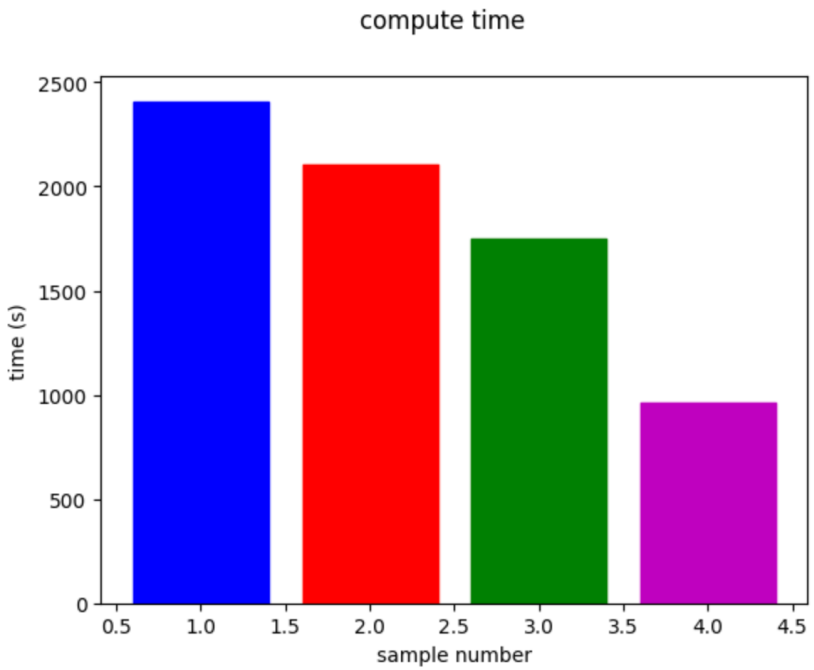


16 cores: 2.7 GHz Intel SkyLake Platinum 8168
Oracle Linux 8.3
0 GPUs



```
%%time

# alignment functions
from bigstream.align import alignment_pipeline
from bigstream.transform import apply_transform

# define alignment steps
common_kwargs = {
    'alignment_spacing':4.0,
    'shrink_factors':(2,),
    'smooth_sigmas':(2.,),
    'optimizer_args':{
        'learningRate':0.25,
        'minStep':0.,
        'numberOfIterations':400,
    },
}

affine_kwargs = {
    'initial_condition':'CENTER',
}

steps = [('affine', {**common_kwargs, **affine_kwargs},),]

# align
affine = alignment_pipeline(
    fix, mov,
    fix_spacing, mov_spacing,
    steps,
)

# apply affine only
affine_aligned = apply_transform(
    fix, mov,
    fix_spacing, mov_spacing,
    transform_list=[affine,],
)

# write results
np.savetxt('affine.mat', affine)
nrrd.write('./affine.nrrd', affine_aligned.transpose(2,1,0), compression_level=2)

# load precomputed results
affine = np.loadtxt('./affine.mat')
```

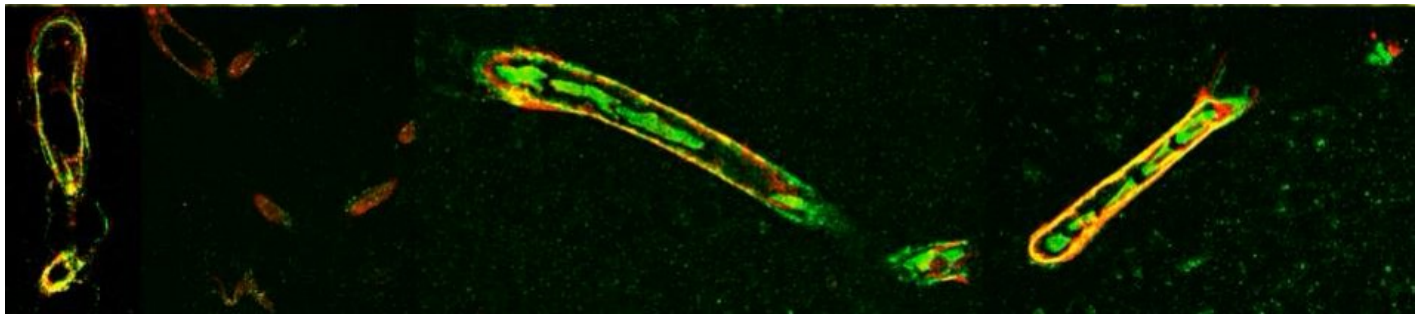
Training Data: Mouse



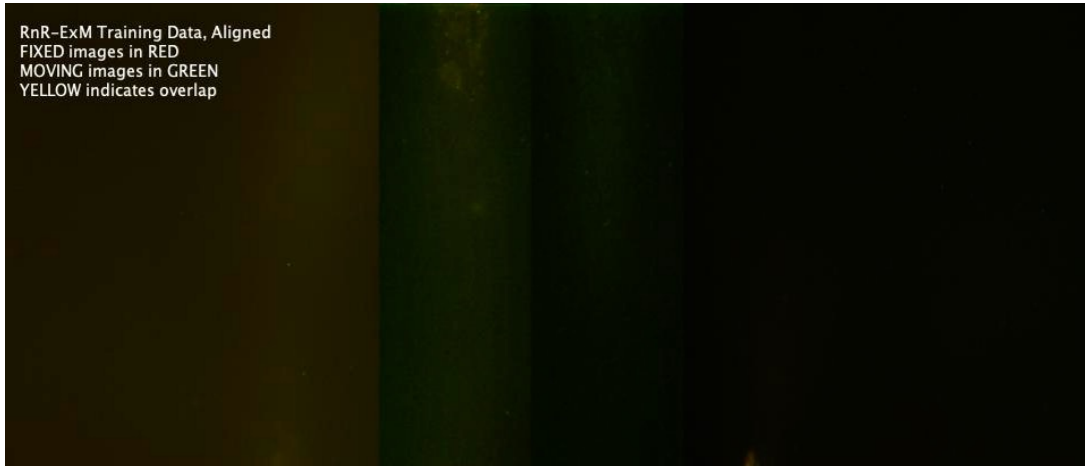
16 cores: 2.7 GHz Intel SkyLake Platinum 8168
Oracle Linux 8.3
0 GPUs

```
%%time

# alignment functions
from bigstream.align import alignment_pipeline
from bigstream.transform import apply_transform

# define alignment steps
common_kwargs = {
    'alignment_spacing':1.0,
    'shrink_factors':(4,2,1,),
    'smooth_sigmas':(4.,2.,1.,),
    'optimizer_args':{
        'learningRate':0.25,
        'minStep':0.,
        'numberOfIterations':400,
    },
}

affine_kwargs = {
    'alignment_spacing':0.5,
    'shrink_factors':(8,4,2,1),
    'smooth_sigmas':(4.,4.,2.,1.),
}

deform_kwargs = {
    'control_point_spacing':32.0,
    'control_point_levels':(1,2,2),
    'fix_mask':fix_mask,
    'mov_mask':mov_mask,
    'optimizer_args':{
        'learningRate':10.0,
        'minStep':0.,
        'numberOfIterations':600,
    },
}

steps = [('affine', {**common_kwargs, **affine_kwargs}),
         ('deform', {**common_kwargs, **deform_kwargs}),]
```

```
# align
affine, deform = alignment_pipeline(
    fix, mov,
    fix_spacing, mov_spacing,
    steps,
    return_format='compressed',
)

# apply affine only
affine_aligned = apply_transform(
    fix, mov,
    fix_spacing, mov_spacing,
    transform_list=[affine,],
)

# apply both
deform_aligned = apply_transform(
    fix, mov,
    fix_spacing, mov_spacing,
    transform_list=[affine, deform],
    extrapolate_with_nn=True,
)
```



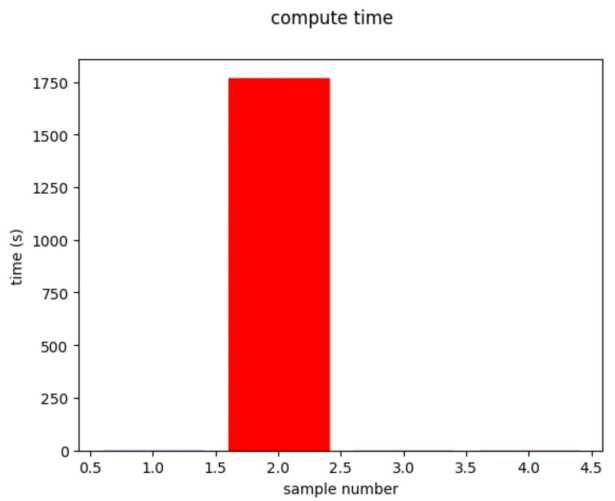
Training Data: C. elegans



16 cores: 2.7 GHz Intel SkyLake Platinum 8168
Oracle Linux 8.3
0 GPUs

```
%%time

# alignment functions
from bigstream.align import alignment_pipeline
from bigstream.transform import apply_transform

# define alignment steps
common_kwargs = {
    'alignment_spacing':1.0,
    'shrink_factors':(2,),
    'smooth_sigmas':(1.5,),
    'optimizer_args':{
        'learningRate':0.25,
        'minStep':0.,
        'numberOfIterations':400,
    },
}

affine_kwargs = {
    'initial_condition':'CENTER',
}

affine2_kwargs = {
    'fix_mask':fix_mask,
    'mov_mask':mov_mask,
    'optimizer_args':{
        'learningRate':1.0,
        'minStep':0.,
        'numberOfIterations':3600,
    },
}

deform_kwargs = {
    'fix_mask':fix_mask,
    'mov_mask':mov_mask,
    'control_point_spacing':32.0,
    'control_point_levels':(1,),
    'alignment_spacing':1.0,
    'shrink_factors':(2,),
    'smooth_sigmas':(1.5,),
    'optimizer_args':{
        'learningRate':10.0,
        'minStep':0.,
        'numberOfIterations':60,
    },
}

steps = [('affine', {**common_kwargs, **affine_kwargs},),
         ('affine', {**common_kwargs, **affine2_kwargs},),
         ('deform', {**common_kwargs, **deform_kwargs},),]
```

```
# align
affine, deform = alignment_pipeline(
    fix, mov,
    fix_spacing, mov_spacing,
    steps,
    return_format='compressed',
)

# apply affine only
affine_aligned = apply_transform(
    fix, mov,
    fix_spacing, mov_spacing,
    transform_list=[affine,],
)

# deform aligned
deform_aligned = apply_transform(
    fix, mov,
    fix_spacing, mov_spacing,
    transform_list=[affine, deform],
)
```

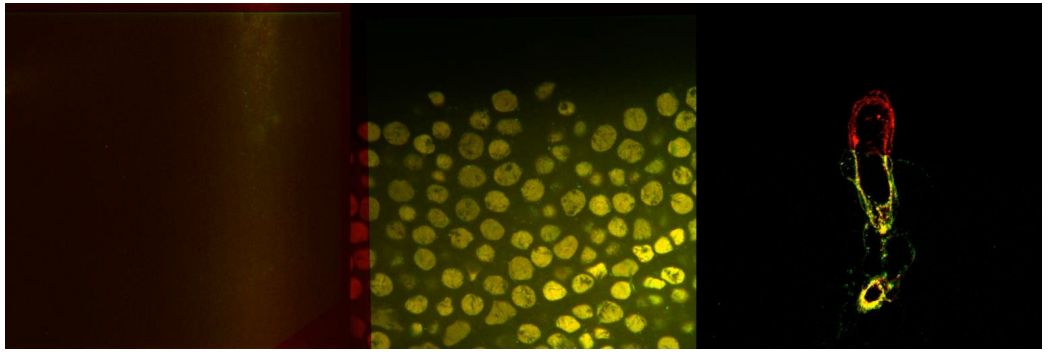


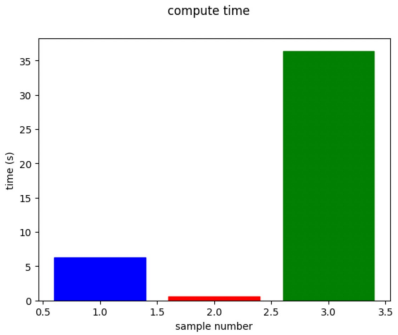
Lost the compute times for samples 1, 3, and 4

Conclusions from training data

- Bigstream can successfully align all the data types
- Bigstream can handle simple and complex alignments
- Bigstream can align data pairs with significantly different intensity characteristics and SNR
- Multiple alignment strategies are possible with different cost/benefit ratios
- In-memory alignments run efficiently on typical workstation size resources

Validation Data





16 cores: 2.7 GHz Intel SkyLake Platinum 8168
Oracle Linux 8.3
0 GPUs

C elegans

```
# alignment functions
from bigstream.align import alignment_pipeline
from bigstream.transform import apply_transform

# define alignment steps
common_kwargs = {
    'alignment_spacing':2.0,
    'shrink_factors':(2,),
    'smooth_sigmas':(2.,),
    'optimizer_args':{
        'learningRate':0.25,
        'minStep':0.,
        'numberOfIterations':2000,
    },
}

rigid_kwargs = {
    'initial_condition':'CENTER',
}

steps = [('rigid',  {**common_kwargs, **rigid_kwargs},),]
```

```
%%time

# run alignment
affine = alignment_pipeline(
    fix, mov,
    fix_spacing, mov_spacing,
    steps,
    # fix_mask=fix_mask,
    # mov_mask=mov_mask,
)
```

Zebrafish

```
# define alignment
from bigstream.align import alignment_pipeline
from bigstream.transform import apply_transform

# define alignment steps
common_kwargs = {
    'alignment_spacing':4.0,
    'shrink_factors':(2,),
    'smooth_sigmas':(2.,),
    'optimizer_args':{
        'learningRate':0.25,
        'minStep':0.,
        'numberOfIterations':400,
    },
}

affine_kwargs = {
    'initial_condition':'CENTER',
}

steps = [('affine', {**common_kwargs, **affine_kwargs},)]
```

```
%%time

# run it
affine = alignment_pipeline(
    fix, mov,
    fix_spacing, mov_spacing,
    steps,
)
```

Mouse

```
# alignment functions
from bigstream.align import alignment_pipeline
from bigstream.transform import apply_transform

# define alignment steps
common_kwargs = {
    'alignment_spacing':0.5,
    'shrink_factors':(8,4,2,1),
    'smooth_sigmas':(4.,4.,2.,1.),
    'optimizer_args':{
        'learningRate':0.25,
        'minStep':0.,
        'numberOfIterations':400,
    },
}

steps = [('affine', common_kwargs),]
```

```
%%time

# align
affine = alignment_pipeline(
    fix, mov,
    fix_spacing, mov_spacing,
    steps,
)
```

Conclusions from validation data

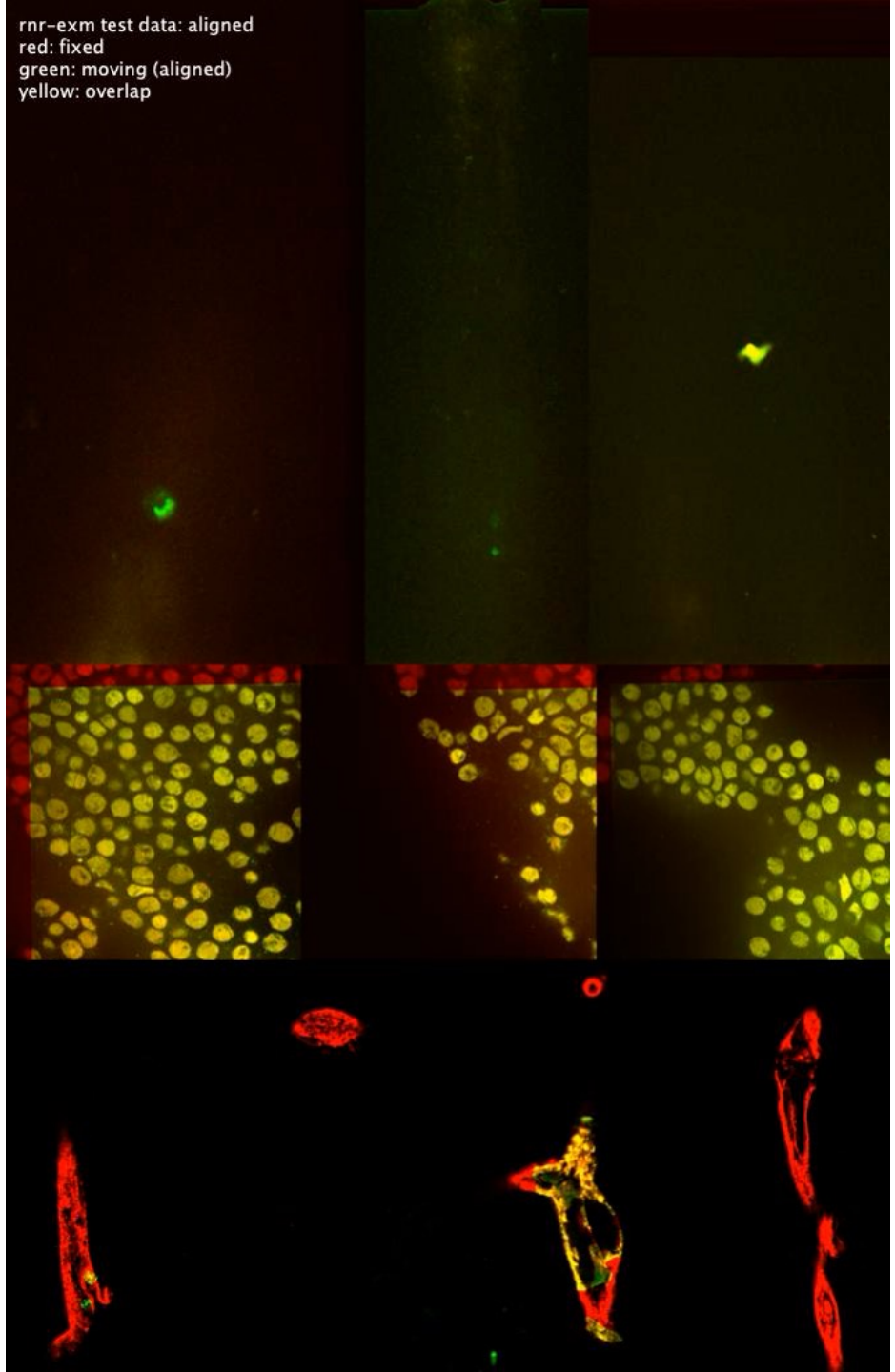
- Very simple and fast alignment can solve most of the discrepancy for these datasets

Test Data: aligned

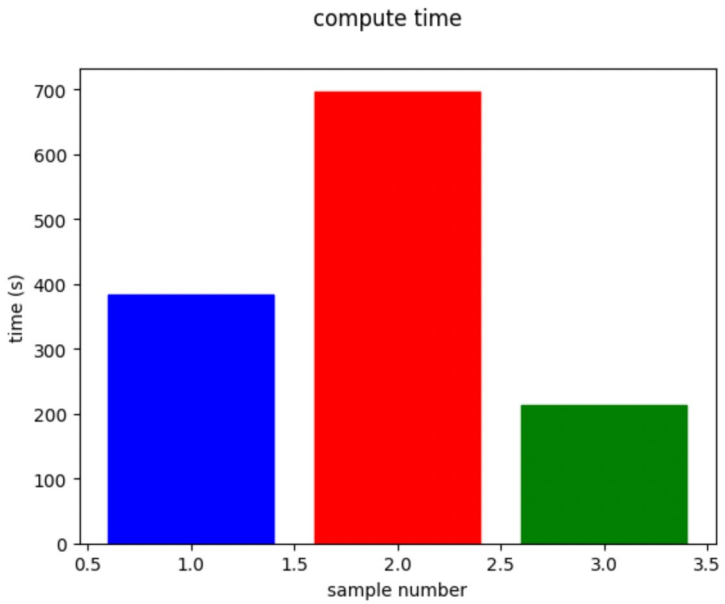
C. elegans

Zebrafish

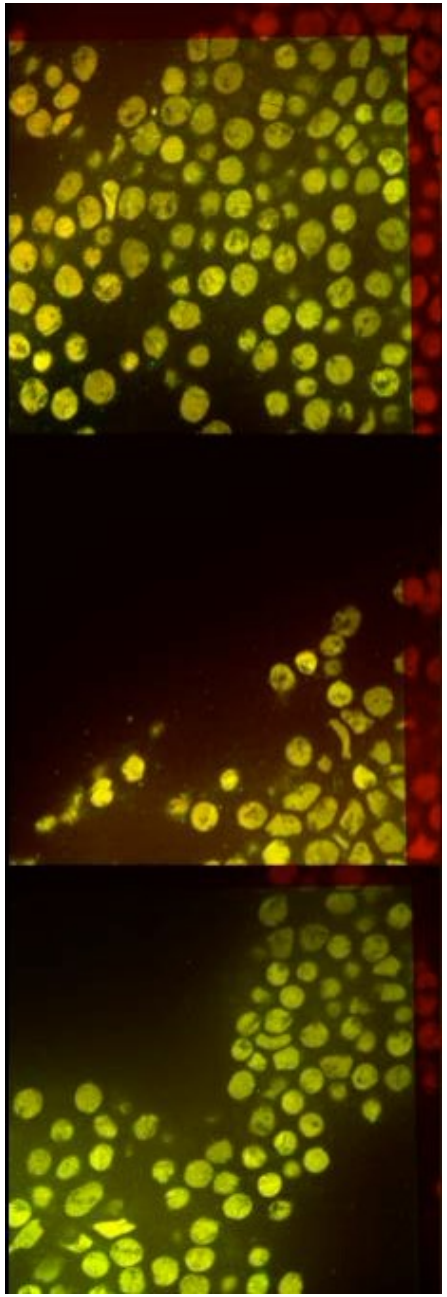
Mouse



Test Data: Zebrafish



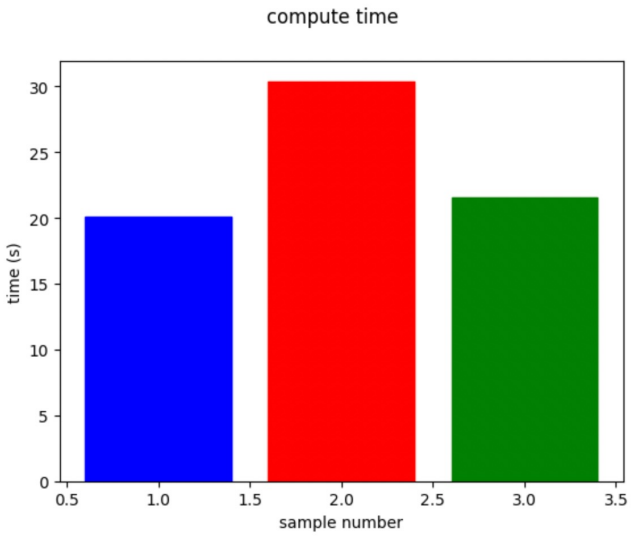
16 cores: 2.7 GHz Intel SkyLake Platinum 8168
Oracle Linux 8.3
0 GPUs



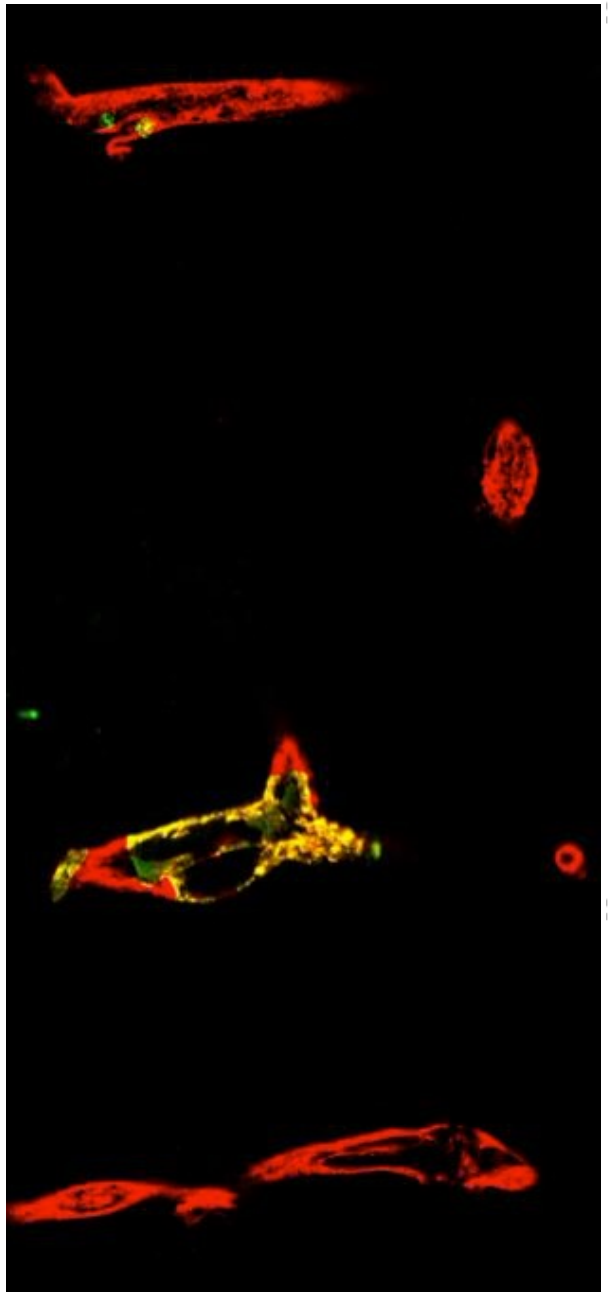
```python
# define alignment
from bigstream.align import alignment_pipeline
from bigstream.transform import apply_transform

# define alignment steps
common_kwargs = {
    'alignment_spacing':0.4,
    'shrink_factors':(2,2,),
    'smooth_sigmas':(2.,0.2,),
    'optimizer_args':{
        'learningRate':0.25,
        'minStep':0.,
        'numberOfIterations':500,
    },
}

affine_kwargs = {
    'initial_condition':'CENTER',
}

steps = [('affine', {**common_kwargs, **affine_kwargs},),]
```

```python
%%time

# run it
affine = alignment_pipeline(
    fix, mov,
    fix_spacing, mov_spacing,
    steps,
)
```

Test Data: Mouse



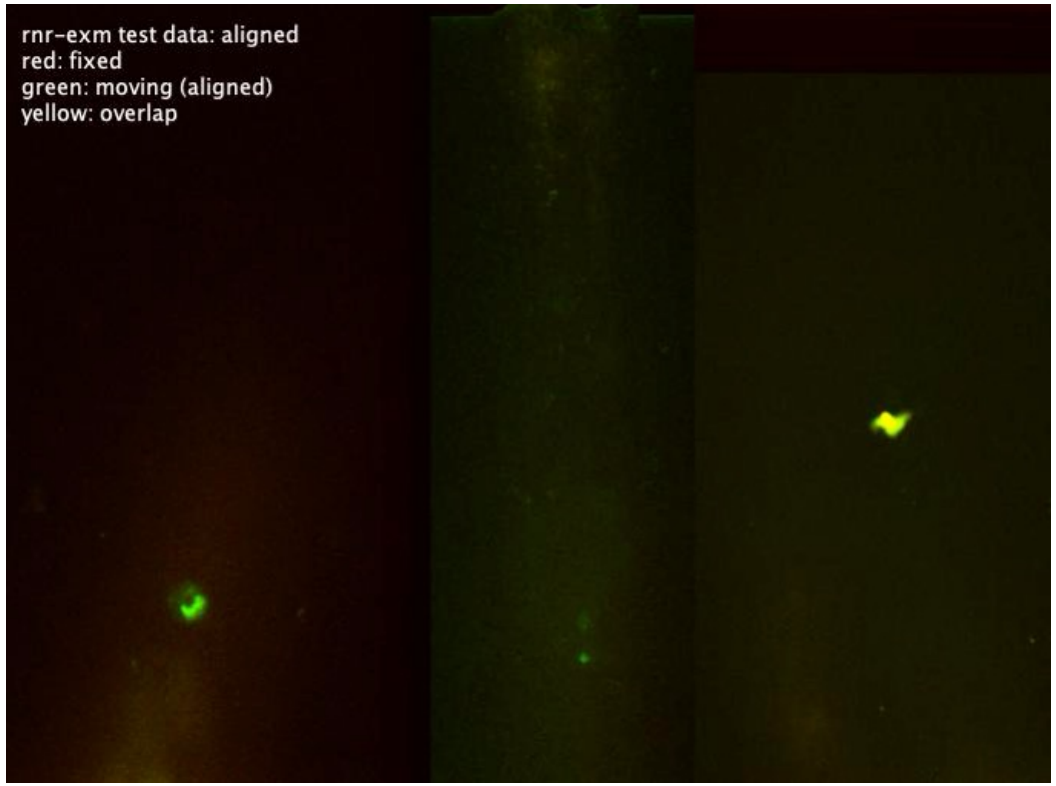
16 cores: 2.7 GHz Intel SkyLake Platinum 8168
Oracle Linux 8.3
0 GPUs



```
# alignment functions
from bigstream.align import alignment_pipeline
from bigstream.transform import apply_transform

# define alignment steps
common_kwargs = {
    'alignment_spacing':0.5,
    'shrink_factors':(8,4,2,1),
    'smooth_sigmas':(4.,4.,2.,1.),
    'optimizer_args':{
        'learningRate':0.25,
        'minStep':0.,
        'numberOfIterations':400,
    },
}

steps = [('affine', common_kwargs),]
```

```
%%time

# align
affine = alignment_pipeline(
    fix, mov,
    fix_spacing, mov_spacing,
    steps,
)
```

Test Data:
C. elegans

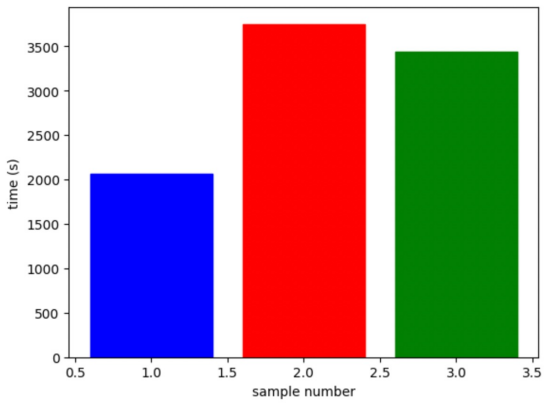



```python
# alignment functions
from bigstream.align import alignment_pipeline
from bigstream.transform import apply_transform
from bigstream.piecewise_align import distributed_piecewise_alignment_pipeline

# define alignment steps
common_kwargs = {
    'alignment_spacing':2.0,
    'shrink_factors':(2,1),
    'smooth_sigmas':(12.,2.,),
    'optimizer_args':{
        'learningRate':0.25,
        'minStep':0.,
        'numberOfIterations':6000,
    },
}

rigid_kwargs = {
    'initial_condition':'CENTER',
}

affine_kwargs = {
    'smooth_sigmas':(2.,2.,),
}

steps = [('rigid',  {**common_kwargs, **rigid_kwargs},),
         ('affine', {**common_kwargs, **affine_kwargs},),]

dpap_affine_kwargs = {
    'alignment_spacing':None,
    'shrink_factors':(1,),
    'smooth_sigmas':(0.4,),
    'optimizer_args':{
        'learningRate':0.25,
        'minStep':0.,
        'numberOfIterations':800,
    },
}

dpap_deform_kwargs = {
    'control_point_spacing':3.0,
    'control_point_levels':(1,),
    'alignment_spacing':None,
    'shrink_factors':(1,),
    'smooth_sigmas':(0.1625,),
    'optimizer_args':{
        'learningRate':0.25,
        'minStep':0.,
        'numberOfIterations':200,
    },
}

dpap_steps = [('affine', dpap_affine_kwargs,),
              ('deform', dpap_deform_kwargs,),]
```

```python
cluster_kwargs = {
    'project':'ahrens',
    'ncpus':1,
    'threads':1,
    'min_workers':200,
    'max_workers':800,
}

blocksize = [24, 48, 48]
```

```python
%%time

# run alignment
affine = alignment_pipeline(
    fix, mov,
    fix_spacing, mov_spacing,
    steps,
    fix_mask=fix_mask,
    mov_mask=mov_mask,
)

# run piecewise alignment
deform = distributed_piecewise_alignment_pipeline(
    fix, mov,
    fix_spacing, mov_spacing,
    dpap_steps,
    blocksize,
    fix_mask=fix_mask,
    mov_mask=mov_mask,
    metric='C',
    static_transform_list=[affine,],
    cluster_kwargs=cluster_kwargs,
)
```

compute time



800 cores: 2.7 GHz Intel SkyLake Platinum 8168
Oracle Linux 8.3
0 GPUs

Conclusions from test data

- Simple and quick alignments are still very effective
- Costly distributed alignments are possible and can give increased precision if you want to pay the cost

Conclusions from the overall challenge

- DSC is a good metric, but we all overfit to the single annotation
- Both traditional methods and learning methods can achieve best performance but with different cost/benefit trade offs
  - This is very fruitful discussion topic in person and in the literature
- Challenges are fun and push the sharing of tools and advance the state of the art

Give Bigstream a try! github.com/GFleishman/bigstream

I've shown some of the general features of the package but there are a lot more available