FURY Documentation

Release beta

Contributors

Jul 22, 2021

ABOUT HELIOS


Helios

CHAPTER

ONE

ABOUT

1.1 Overview

1.2 What is Helios ?

1.3 Mission Statement

The purpose of Helios is to make it easy to perform graph/network Visualization …

- …
- …

1.4 Features

Efficient

Robust

Multiplatform

1.5 License

Helios is distributed under the MIT License

1.6 Credits

Go to *Community page* to see who have been involved in the development of Helios.

1.7 Bug reports and support

Please report any issues via https://github.com/fury-gl/helios/issues. All types of issues are welcome including bug reports, documentation typos, feature requests and so on.

CHAPTER

TWO

LICENSE

CHAPTER

THREE

BLOG

CHAPTER

FOUR

INSTALLATION

Helios supports Python 3.7+.

4.1 Dependencies

The mandatory dependencies are:

- numpy >= 1.7.1
- vtk >= 8.1.0
- fury

The optional dependencies are:

- opencv
- cugraph

4.2 Installation with PyPi

In a terminal, issue the following command

```
pip install helios
```

4.3 Installation via Source

Step 1. Get the latest source by cloning this repo

```
git clone https://github.com/fury-gl/helios.git
```

Step 2. Install requirements

```
pip install -r requirements.txt
```

Step 3. Install helios via

```
pip install -e .
```

Step 4: Enjoy!

4.4 Running the Tests

Let's install all required packages for the running the test

```
pip install -r requirements.txt
pip install -r requirements_dev.txt
```

There are two ways to run Helios tests:

- From the command line. You need to be on the Helios package folder

```
pytest -svv helios
```

- To run a specific test file

```
pytest -svv helios/tests/test_actor.py
```

- To run a specific test directory

```
pytest -svv helios/tests
```

- To run a specific test function

```
pytest -svv -k "test_my_function_name"
```

4.5 Running the Tests Offscreen

Helios is based on VTK which uses OpenGL for all its rendering. For a headless rendering, we recommend to install and use Xvfb software on linux or OSX. Since Xvfb will require an X server (we also recommend to install XQuartz package on OSX). After Xvfb is installed you have 2 options to run Helios tests:

- First option

```
export DISPLAY=:0
Xvfb :0 -screen 1920x1080x24 > /dev/null 2>1 &
pytest -svv fury
```

- Second option

```
export DISPLAY=:0
xvfb-run --server-args="-screen 0 1920x1080x24" pytest -svv fury
```

4.6 Populating our Documentation

In our `docs` folder structure above:

- `source` is the folder that contains all `*.rst` files.
- `tutorials` is the directory where we have all python scripts that describe how to use the api.
- `examples` being the Helios app showcases.

4.6.1 Building the documentation

Step 1. Install all required packages for the documentation generation

```
pip install -U -r requirements.txt
pip install -U -r requirements_docs_sys.txt
```

Step 2. Go to the `docs` folder and run the following command to generate it (Linux and macOS)

```
make -C . clean && make -C . html
```

To generate the documentation without running the examples

```
make -C . clean && make -C . html-no-examples
```

or under Windows

```
make clean
make html
```

To generate the documentation without running the examples under Windows

```
make clean
make html-no-examples
```

Step 3. Congratulations! the `build` folder has been generated! Go to `build/html` and open with browser `index.html` to see your generated documentation.

CHAPTER

FIVE

GETTING STARTED

Start by importing Helios.

```
import numpy as np
```

CHAPTER

SIX

EXAMPLES

6.1 Visualize Interdisciplinary map of the journals network

The goal of this app is to show an overview of the journals network structure as a complex network. Each journal is shown as a node and their connections indicates a citation between two of them.

First, let's import some useful functions

```
from os.path import join as pjoin
from fury import colormap as cmap
from fury.window import record
import numpy as np
```

Then let's download some available datasets.

```
from fury.data.fetcher import fetch_viz_wiki_nw

from helios import NetworkDraw
from helios.layouts.force_directed import HeliosFr

files, folder = fetch_viz_wiki_nw()
categories_file, edges_file, positions_file = sorted(files.keys())
```

Out:

```
Dataset is already in place. If you want to fetch it again please first remove the␣
↪folder /home/devmessias/.fury/examples/wiki_nw
More information about complex networks can be found in this papers: https://arxiv.org/
↪abs/0711.3199
```

We read our datasets

```
positions = np.loadtxt(pjoin(folder, positions_file))
positions = np.random.normal(scale=10, size=positions.shape)
categories = np.loadtxt(pjoin(folder, categories_file), dtype=str)
edges = np.loadtxt(pjoin(folder, edges_file), dtype=int)
```

We attribute a color to each category of our dataset which correspond to our nodes colors.

```
category2index = {category: i
                  for i, category in enumerate(np.unique(categories))}
```

(continues on next page)

(continued from previous page)

```
index2category = np.unique(categories)

categoryColors = cmap.distinguishable_colormap(nb_colors=len(index2category))

colors = np.array([categoryColors[category2index[category]]
                   for category in categories])
```

We define our node size

```
radii = 1 + np.random.rand(len(positions))
```

Lets create our edges now. They will indicate a citation between two nodes. OF course, the colors of each edges will be an interpolation between the two node that it connects.

```
#edgesPositions = []
edgesColors = []
for source, target in edges:
    #edgesPositions.append(np.array([positions[source], positions[target]]))
    edgesColors.append(np.array([colors[source], colors[target]]))

#edgesPositions = np.array(edgesPositions)
edgesColors = np.average(np.array(edgesColors), axis=1)
```

Our data preparation is ready, it is time to visualize them all. We start to build 2 actors that we represent our data : sphere_actor for the nodes and lines_actor for the edges.

```
network_draw = NetworkDraw(
        positions=positions,
        colors=colors,
        scales=4,
        node_edge_width=0,
        edge_line_color=edgesColors,
        marker='3d',
        edges=edges,
)
layout = HeliosFr(edges, network_draw, update_interval_workers=10)

layout.start()
```

The final step ! Visualize and save the result of our creation! Please, switch interactive variable to True if you want to visualize it.

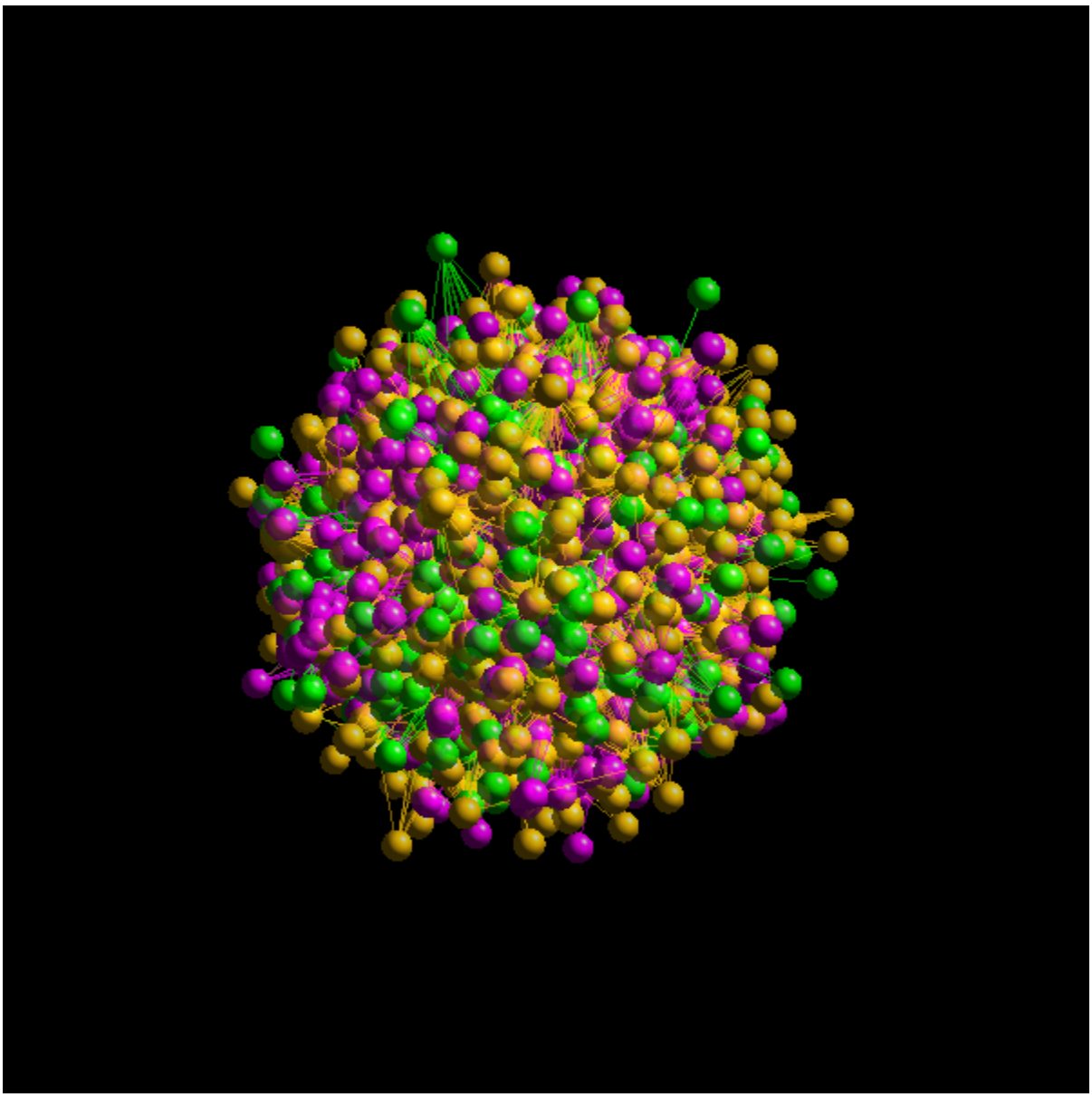
```
interactive = False
if not interactive:
    import time
    time.sleep(10)
    layout.stop()

if interactive:
    network_draw.showm.initialize()
    network_draw.showm.start()
```

(continues on next page)

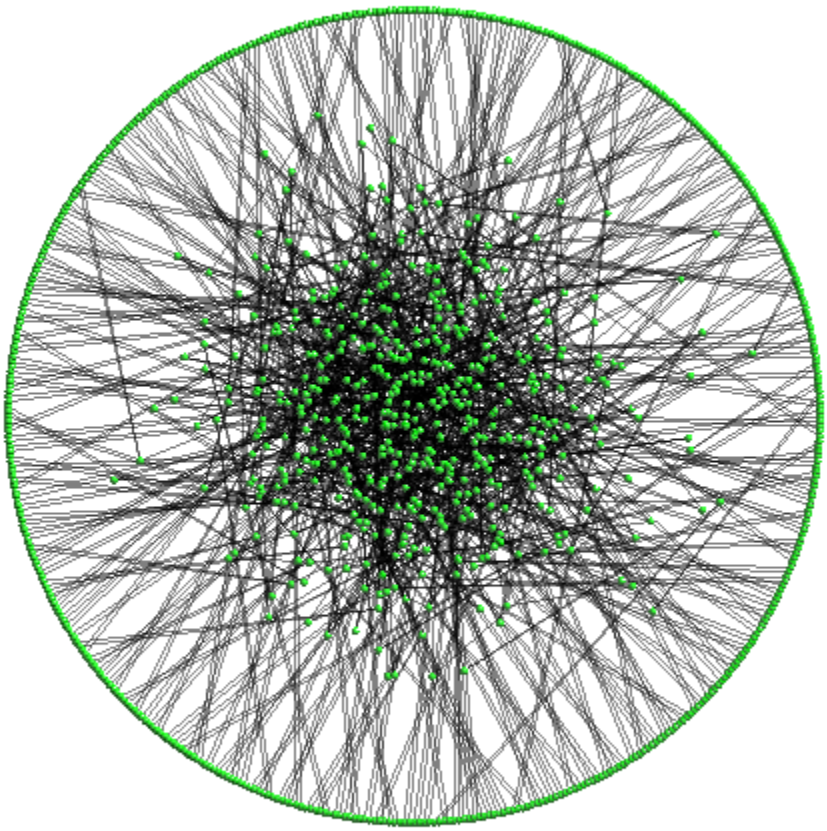
(continued from previous page)

```
record(
    network_draw.showm.scene, out_path='viz_helios.png', size=(600, 600))
```



Total running time of the script: (0 minutes 14.288 seconds)

6.2 Minmum Distortion Embedding with Anchored Constraints



```
import numpy as np
import time
from fury.window import record

from helios import NetworkDraw
from helios.layouts.mde import MDE

# from https://github.com/cvxgrp/pymde/blob/main/examples/anchor_constraints.ipynb

depth = 9
n_items = 2**(depth + 1) - 1
```

(continues on next page)

(continued from previous page)

```
edges = []
stack = [0]
while stack:
    root = stack.pop()
    first_child = root*2 + 1
    second_child = root*2 + 2
    if first_child < n_items:
        edges.append([root, first_child])
        stack.append(first_child)
    if second_child < n_items:
        edges.append([root, second_child])
        stack.append(second_child)

# these are the indices of the nodes that we will pin in place
anchors = np.arange(2**depth) + 2**depth - 1


radius = 20

# pin the root to be at (0, 0), and the leaves to be spaced uniformly on a circle
angles = np.linspace(0, 2*np.pi, anchors.shape[0] + 1)[1:]
anchors_pos = radius * np.stack([np.sin(angles), np.cos(angles)], axis=1)
centers = np.random.normal(size=(n_items, 2))*5
centers[anchors] = anchors_pos.copy()


network_draw = NetworkDraw(
        positions=centers,
        scales=.4,
        node_edge_width=0,
        #colors=(1, 0,0),
        edge_line_opacity=.5,
        edge_line_color=(0, 0, 0),
        marker='3d',
        window_size=(500, 500),
        edges=np.array(edges)
)

mde = MDE(
    np.array(edges), network_draw,
    constraint_name='anchored',
    anchors=anchors.astype('float32'),
    anchors_pos=anchors_pos.astype('float32'),
    use_shortest_path=True
)


interactive = False
if not interactive:
    mde.start(
        100, 1, 300,
        record_positions=False, without_iren_start=True)
```

(continues on next page)

(continued from previous page)

```
    time.sleep(30)
    mde.stop()
else:
    mde.start(
        3, 300, 1,
        record_positions=True, without_iren_start=False)

if interactive:
    network_draw.showm.initialize()
    network_draw.showm.start()

record(
    network_draw.showm.scene, out_path='viz_mde.png', size=(600, 600))
```

Total running time of the script: (0 minutes 30.180 seconds)

CHAPTER

SEVEN

API

helios.layouts	Layouts
helios.backends.fury	
helios.core	

7.1 helios.layouts

Layouts

helios.layouts.base	Network Layout Abstract Classes
helios.layouts.force_directed	Force-Directed Layout oct-tree
helios.layouts.forceatlas2gpu	ForceAtlas2 cuGraph
helios.layouts.ipc_tools	Inter-Process communication tools
helios.layouts.mde	Minimum-Distortion Embedding

7.1.1 helios.layouts.base

Network Layout Abstract Classes

This module provides a set of abstract classes to deal with different network layouts algorithms using different communication strategies.

Functions

—

helios.layouts.base.**is_running**(*p*, *timeout=0*)

Check if the process *p* is running

Parameters

- **p** (*process*) –
- **timeout** (*float, optional*) – positive float

Returns running

Return type bool

Classes

—

class helios.layouts.base.**NetworkLayout**

- *Visualize Interdisciplinary map of the journals network*

class helios.layouts.base.**NetworkLayoutAsync**

- *Visualize Interdisciplinary map of the journals network*

class helios.layouts.base.**NetworkLayoutIPCRender**(*network_draw*, *edges*, *weights=None*)

An abstract class which reads the network information and creates the shared memory resources.

Parameters

- **network_draw** (NetworkDraw) – A NetworkDraw object which will be used to draw the network
- **edges** (*ndarray*) – a bi-dimensional array with the edges list
- **weights** (*array, optional*) – a one-dimensional array with the edge weights

- *Minmum Distortion Embedding with Anchored Constraints*

class helios.layouts.base.**NetworkLayoutIPCServerCalc**(*num_nodes*, *num_edges*, *edges_buffer_name*, *positions_buffer_name*, *info_buffer_name*, *weights_buffer_name=None*, *dimension=3*, *snaphosts_buffer_name=None*, *num_snapshots=0*)

An abstract class which reads the network information from the shared memory resources.

This should be used inside of a subprocess which will update the network layout positions

Parameters

- **num_nodes** (*int*) –
- **num_edges** (*int*) –
- **edges_buffer_name** (*str*) –
- **positions_buffer_name** (*str*) –
- **info_buffer_name** (*str*) –
- **weights_buffer_name** (*str, optional*) –
- **dimension** (*int*) –
- **snaphosts_buffer_name** (*str, optional*) –
- **num_snapshots** (*int, optional*) –

7.1.2 helios.layouts.force_directed

Force-Directed Layout oct-tree

Classes

—

class helios.layouts.force_directed.**HeliosFr**(*edges*, *network_draw*, *viscosity=0.3*, *a=0.0006*, *b=1*, *max_workers=8*, *update_interval_workers=0*, *velocities=None*)

A 2D/3D Force-directed layout method

This method is a wrapper for the helios force-directed algorithm, heliosFR. HeliosFr is a force-directed layout algorithm that is based on oct-trees. The algorithm is designed to work with a large number of nodes and edges.

References

[1] Fruchterman, T. M. J., & Reingold, E. M. (1991). Graph Drawing by Force-Directed Placement. Software: Practice and Experience, 21(11).

Parameters

- **edges** (*ndarray*) –
- **network_draw** (NetworkDraw) –
- **viscosity** (*float, optional*) –
- **a** (*float, optional*) –
- **b** (*float, optional*) –
- **max_workers** (*int, optional*) – number of threads
- **update_interval_workers** (*float, optional*) – When you set this to a value greater than zero the helios-fr will wait to perform each step. This can be used to reduce the CPU consumption

- *Visualize Interdisciplinary map of the journals network*

7.1.3 helios.layouts.forceatlas2gpu

ForceAtlas2 cuGraph

ForceAtlas2 layout algorithm through IPC using cuGraph.

Classes

—

class helios.layouts.forceatlas2gpu.**ForceAtlas2**(*edges*, *network_draw*, *weights=None*, *lin_log_mode=False*, *edge_weight_influence=1.0*, *jitter_tolerance=1.0*, *barnes_hut_optimize=True*, *barnes_hut_theta=1.0*, *scaling_ratio=2.0*, *strong_gravity_mode=False*, *gravity=1.0*)

Performs the ForceAtlas2 algorithm using the cugraph lib.

The ForceAtlas will be called inside of a different process which comunicates with this object through the SharedMemory

---

Notes

Python 3.8+ is required to use this

Parameters

- **edges** (*ndarray*) –
- **network_draw** (NetworkDraw) –
- **weights** (*array, optional*) – edge weights
- **lin_log_mode** (*bool, default False*) –
- **edge_weight_influence** (*float, default 1.0*) –
- **jitter_tolerance** (*float, default 1.0*) –
- **barnes_hut_optimize** (*bool, default True*) –
- **barnes_hut_theta** (*float, default 1.0*) –
- **scaling_ratio** (*float, default 2.0*) –
- **strong_gravity_mode** (*bool, default False*) –
- **gravity** (*float, default 1.0*) –

class helios.layouts.forceatlas2gpu.**ForceAtlas2ServerCalc**(*edges_buffer_name*, *positions_buffer_name*, *info_buffer_name*, *weights_buffer_name=None*, *snapshots_buffer_name=None*, *num_snapshots=0*, *lin_log_mode=False*, *edge_weight_influence=1.0*, *jitter_tolerance=1.0*, *barnes_hut_optimize=True*, *barnes_hut_theta=1.0*, *scaling_ratio=2.0*, *strong_gravity_mode=False*, *gravity=1.0*)

This Obj. reads the network information stored in a shared memory resource and execute the ForceAtlas2 layout algorithm

Parameters

- **edges_buffer_name** (*str*) – The name of the shared memory buffer where the edges are stored
- **positions_buffer_name** (*str*) – The name of the shared memory buffer where the positions are stored.
- **info_buffer_name** (*str*) –
- **weights_buffer_name** (*str, optional*) –
- **snapshots_buffer_name** (*str, optional*) –
- **num_snapshots** (*int, optional*) –
- **lin_log_mode** (*bool, default False*) –
- **edge_weight_influence** (*float, default 1.0*) –
- **jitter_tolerance** (*float, default 1.0*) –
- **barnes_hut_optimize** (*bool, default True*) –
- **barnes_hut_theta** (*float, default 1.0*) –
- **scaling_ratio** (*float, default 2.0*) –
- **strong_gravity_mode** (*bool, default False*) –
- **gravity** (*float, default 1.0*) –

7.1.4 helios.layouts.ipc_tools

Inter-Process communication tools

This Module provides some objects to deal with inter-process communication

Classes

---

class helios.layouts.ipc_tools.**GenericArrayBufferManager**(*dimension*, *dtype='float64'*, *num_elements=None*)

This implements a abstract (generic) ArrayBufferManager.

The GenericArrayBufferManager is used for example to share the positions, edges and weights between different process.

Parameters

- **dimension** (*int*) –
- **dtype** (*dtype*) –
- **num_elements** (*int, optional*) – In MacOs a shared memory resource can be created with a different number of elements then the original data

class helios.layouts.ipc_tools.**SharedMemArrayManager**(*dimension*, *dtype*, *data=None*, *buffer_name=None*, *num_elements=None*)

An implementation of a GenericArrayBufferManager using SharedMemory

Parameters

- **dimension** (*int*) – number of columns

- **dtype** (*str*) – type of the ndarray
- **data** (*ndarray*) – bi-dimensional array
- **buffer_name** (*str*) – buffer_name, if you pass that, then this Obj. will try to load the memory resource
- **num_elements** (*int, optional*) – In MacOs a shared memory resource can be created with a different number of elements then the original data

class helios.layouts.ipc_tools.**ShmManagerMultiArrays**
This Obj. allows to deal with multiple arrays stored using SharedMemory

7.1.5 helios.layouts.mde

Minimum-Distortion Embedding

Classes

---

class helios.layouts.mde.**MDE**(*edges*, *network_draw*, *weights=None*, *use_shortest_path=True*, *constraint_name=None*, *anchors=None*, *anchors_pos=None*, *penalty_name=None*, *penalty_parameters=None*, *attractive_penalty_name='log1p'*, *repulsive_penalty_name='log'*)
Minimum Distortion Embedding algorithm running on IPC

This call the PyMDE lib running in a different process which comunicates with this object through SharedMemory from python>=3.8.

References

[1] PyMDE

Notes

Python 3.8+ is required to use this

Parameters

- **edges** (*ndarray*) – the edges of the graph. A numpy array of shape (n_edges, 2)
- **network_draw** (*NetworkDraw*) – a NetworkDraw object
- **weights** (*array, optional*) – edge weights. A one dimensional array of shape (n_edges,)
- **use_shortest_path** (*bool, optional*) – If set to True, shortest path is used to compute the layout
- **constraint_name** (*str, optional*) – centered, standardized or anchored
- **anchors** (*array, optional*) – a list of vertex that will be anchored
- **anchors_pos** (*ndarray, optional*) – The positions of the anchored vertex

- **penalty_name** (*str, optional*) – cubic, huber, invpower, linear, log, log1p, logratio, logistic, power, pushandpull or quadratic
- **penalty_parameters** (*array, optional*) – the parameters of the penalty function
- **attractive_penalty_name** (*str, optional*) – cubic, huber, invpower, linear, log, log1p, logratio, logistic, power, pushandpull or quadratic
- **repulsive_penalty_name** (*str, optional*) – cubic, huber, invpower, linear, log, log1p, logratio, logistic, power, pushandpull or quadratic

- *Minmum Distortion Embedding with Anchored Constraints*

class helios.layouts.mde.**MDEServerCalc**(*num_nodes*, *num_edges*, *edges_buffer_name*, *positions_buffer_name*, *info_buffer_name*, *dimension=3*, *weights_buffer_name=None*, *snapshots_buffer_name=None*, *num_snapshots=0*, *penalty_name=None*, *penalty_parameters_buffer_name=None*, *num_penalty_parameters=None*, *attractive_penalty_name='log1p'*, *repulsive_penalty_name='log'*, *use_shortest_path=False*, *constraint_name=None*, *constraint_anchors_buffer_name=None*, *num_anchors=None*)

This Obj. reads the network information stored in a shared memory resource and execute the MDE layout algorithm

Parameters

- **num_nodes** (*int*) –
- **num_edges** (*int*) –
- **edges_buffer_name** (*str*) –
- **positions_buffer_name** (*str*) –
- **info_buffer_name** (*str*) –
- **weights_buffer_name** (*str, optional*) –
- **snapshots_buffer_name** (*str, optional*) –
- **num_snapshots** (*int, optional*) –
- **dimension=3** (*int, optional*) – layout dimension
- **penalty_name** (*str, optional*) –
- **penalty_parameters_buffer_name** (*str, optional*) –
- **num_penalty_parameters** (*int, optional*) –
- **attractive_penalty_name** (*str, optional*) –
- **repulsive_penalty_name** (*str, optional*) –
- **use_shortest_path** (*str, optional*) –
- **constraint_name** (*str, optional*) –
- **constraint_anchors_buffer_name** (*str, optional*) –
- **num_anchors** (*int, optional*) –

7.2 helios.backends.fury

helios.backends.fury.actors	VTK/FURY Super Actors
helios.backends.fury.draw	FURY NetworkDraw
helios.backends.fury.tools	VTK/FURY tools

7.2.1 helios.backends.fury.actors

VTK/FURY Super Actors

Classes

—

class helios.backends.fury.actors.**FurySuperEdge**(*edges*, *positions*, *colors*, *opacity=0.5*, *line_width=3*, *blending='additive'*)

class helios.backends.fury.actors.**FurySuperNode**(*positions*, *colors=(0, 1, 0)*, *scales=1*, *marker='3d'*, *edge_width=0.0*, *edge_opacity=1*, *edge_color=(1, 1, 1)*, *marker_opacity=0.8*, *write_frag_depth=True*)

class helios.backends.fury.actors.**NetworkSuperActor**(*positions*, *edges=None*, *colors=(0, 1, 0)*, *scales=1*, *marker='o'*, *node_edge_width=0.0*, *node_opacity=0.8*, *node_edge_opacity=1*, *node_edge_color=(255, 255, 255)*, *edge_line_color=(1, 1, 1)*, *edge_line_opacity=0.5*, *edge_line_width=1*, *write_frag_depth=True*)

- *Visualize Interdisciplinary map of the journals network*
- *Minmum Distortion Embedding with Anchored Constraints*

7.2.2 helios.backends.fury.draw

FURY NetworkDraw

Classes

—

class helios.backends.fury.draw.**NetworkDraw**(*positions*, *edges=None*, *colors=(0, 1, 0)*, *scales=1*, *marker='o'*, *node_edge_width=0.0*, *node_opacity=0.8*, *node_edge_opacity=1*, *node_edge_color=(255, 255, 255)*, *edge_line_color=(1, 1, 1)*, *edge_line_opacity=0.5*, *edge_line_width=1*, *better_performance=False*, *write_frag_depth=True*, *window_size=(400, 400)*, *showm=None*, ***kwargs*)

This object is responsible to deal with the drawing of the network.

Parameters

- **positions** (*ndarray*) – Array of the nodes positions.
- **edges** (*ndarray, optional*) – Array of the edges.
- **colors** (*tuple or ndarray, optional*) – Tuple of the colors of the nodes.
- **scales** (*float or ndarray, optional*) – Scaling factor for the node.
- **marker** (*str or list, optional*) – Marker for the nodes.
- **node_edge_width** (*float or array, optional*) – Width of the edges around the nodes.
- **node_opacity** (*float or array, optional*) – Opacity of the nodes.
- **node_edge_opacity** (*float or array, optional*) – Opacity of the edges.
- **node_edge_color** (*tuple or ndarray, optional*) – Color of the edges around the nodes.
- **edge_line_color** (*tuple or ndarray, optional*) – Color of the edges.
- **edge_line_opacity** (*float or ndarray, optional*) – Opacity of the edges.
- **edge_line_width** (*float or ndarray, optional*) – Width of the edges.
- **better_performance** (*bool, optional*) – Improves the performance of the draw function.
- **write_frag_depth** (*bool, optional*) – Writes in the depth buffer.
- **window_size** (*tuple, optional*) – Size of the window.
- **showm** (*showm, optional*) – Fury ShowManager instance.

- *Visualize Interdisciplinary map of the journals network*
- *Minmum Distortion Embedding with Anchored Constraints*

7.2.3 helios.backends.fury.tools

VTK/FURY tools

Classes

---

class helios.backends.fury.tools.**Uniform**(*name*, *uniform_type*, *value*)

This creates a uniform shader variable

It's responsible to store the value of a given uniform variable and call the related vtk_program

Parameters

- **name** (*str*) – name of the uniform variable
- **uniform_type** (*str*) – Uniform variable type which will be used inside the shader. Any of this are valid: 1fv, 1iv, 2f, 2fv, 2i, 3f, 3fv,

  3uc, 4f, 4fv, 4uc, GroupUpdateTime, Matrix, Matrix3x3, Matrix4x4, Matrix4x4v, f, i value: float or ndarray
- **value** (*type(uniform_type)*) – should be a value which represent's the shader uniform variable. For example, if uniform_type is 'f' then value should be a float; if uniform_type is '3f' then value should be a 1x3 array.

class helios.backends.fury.tools.**Uniforms**(*uniforms*)

Creates an object which store and execute an uniform variable.

Parameters **uniforms** (*list*) – List of Uniform objects.

Examples

```python uniforms = [

    Uniform(name='edgeWidth', uniform_type='f', value=edgeWidth)...

] CustomUniforms = Uniforms(markerUniforms) add_shader_callback(

    sq_actor, CustomUniforms)

sq_actor.CustomUniforms = CustomUniforms sq_actor.CustomUniforms.edgeWidth = 0.5 ```

---

## 7.3 helios.core

---

*helios.core.network*

---

### 7.3.1 helios.core.network

#### Classes

---

**class** helios.core.network.**Network**
```

CHAPTER

EIGHT

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

8.1 Types of Contributions

8.1.1 Report Bugs

Report bugs at https://github.com/fury-gl/helios/issues.

If you are reporting a bug, please include:

- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

8.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

8.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

8.1.4 Write Documentation

Helios could always use more documentation, whether as part of the official Helios docs, in docstrings, or even on the web in blog posts, articles, and such. Helios uses [Sphinx](http://www.sphinx-doc.org/en/stable/index.html) to generate documentation. Please follow the [numpy coding style](https://numpydoc.readthedocs.io/en/latest/format.html#docstring-standard) - and of course - [PEP8](https://www.python.org/dev/peps/pep-0008/) for docstring documentation.

8.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/fury-gl/helios/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

8.2 Get Started!

8.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.md.
3. The pull request should work for Python3.7, 3.8, 3.9 and for PyPy.

8.4 Publishing Releases

8.4.1 Checklist before Releasing

- Review the open list of Helios issues. Check whether there are outstanding issues that can be closed, and whether there are any issues that should delay the release. Label them !
- Review and update the release notes. Review and update the `Changelog` file. Get a partial list of contributors with something like:

```
git shortlog -nse v0.1.0..
```

  where `v0.1.0` was the last release tag name.

  Then manually go over `git shortlog v0.1.0..` to make sure the release notes are as complete as possible and that every contributor was recognized.
- Use the opportunity to update the `.mailmap` file if there are any duplicate authors listed from `git shortlog -ns`.
- Add any new authors to the `AUTHORS` file.
- Check the copyright years in `docs/source/conf.py` and `LICENSE`
- Check the examples and tutorial - we really need an automated check here.
- Make sure all tests pass on your local machine (from the `<helios root>` directory):

```
cd ..
pytest -s --verbose --doctest-modules helios
cd helios # back to the root directory
```

- Check the documentation doctests:

```
cd docs
make -C . html
cd ..
```

- The release should now be ready.

8.4.2 Doing the release

- Update release-history.rst in the documentation if you have not done so already. You may also highlight any additions, improvements, and bug fixes.
- Type git status and check that you are on the master branch with no uncommitted code.
- Now it's time for the source release. Mark the release with an empty commit, just to leave a marker. It makes it easier to find the release when skimming through the git history:

```
git commit --allow-empty -m "REL: vX.Y.Z"
```

- Tag the commit:

```
git tag -am 'Second public release' vX.Y.Z  # Don't forget the leading v
```

  This will create a tag named vX.Y.Z. The -a flag (strongly recommended) opens up a text editor where you should enter a brief description of the release.

- Verify that the __version__ attribute is correctly updated:

```
import helios
helios.__version__  # should be 'X.Y.Z'
```

  Incidentally, once you resume development and add the first commit after this tag, __version__ will take on a value like X.Y.Z+1.g58ad5f7, where +1 means "1 commit past version X.Y.Z" and 58ad5f7 is the first 7 characters of the hash of the current commit. The letter g stands for "git". This is all managed automatically by versioneer and in accordance with the specification in PEP 440.

- Push the new commit and the tag to master:

```
git push origin master
git push origin vX.Y.Z
```

- Register for a PyPI account and Install twine, a tool for uploading packages to PyPI:

```
python3 -m pip install --upgrade twine
```

- Remove any extraneous files:

```
git clean -dfx
```

  If you happen to have any important files in your project directory that are not committed to git, move them first; this will delete them!

- Publish a release on PyPI:

```
python setup.py sdist
python setup.py bdist_wheel
twine upload dist/*
```

- Check how everything looks on pypi - the description, the packages. If necessary delete the release and try again if it doesn't look right.
- Set up maintenance / development branches

  If this is this is a full release you need to set up two branches, one for further substantial development (often called 'trunk') and another for maintenance releases.

  - Branch to maintenance:

    ```
    git co -b maint/X.Y.Z
    ```

    Push with something like `git push upstream-rw maint/0.6.x --set-upstream`

  - Start next development series:

    ```
    git co main-master
    ```

    Next merge the maintenace branch with the "ours" strategy. This just labels the maintenance branch *info.py* edits as seen but discarded, so we can merge from maintenance in future without getting spurious merge conflicts:

    ```
    git merge -s ours maint/0.6.x
    ```

    Push with something like `git push upstream-rw main-master:master`

  If this is just a maintenance release from `maint/0.6.x` or similar, just tag and set the version number to - say - `0.6.2.dev`.
- Push the tag with `git push upstream-rw 0.6.0`

CHAPTER

NINE

COMMUNITY

9.1 Join Us!

PYTHON MODULE INDEX

h

INDEX