About

Puppet Warp: A Python library for advanced image transformations, inspired by Photoshop's Puppet Warp, designed for automation in scripting environments.

Readme
GPL-3.0 license
Activity
53 stars
1 watching
4 forks
Report repository

Releases

3 tags

Packages

No packages published

Contributors 1

mikecokina Mike

Languages

Python 100.0%

master 5 Branches 3 Tags Go to file Go to file Code

mikecokina test(unittests): basic unittests coverage	adc62aa · 2 months ago	91 Commits
docs	docs: update api documentation	2 months ago
src/pwarp	docs: update api documentation	2 months ago
tests	test(unittests): basic unittests coverage	2 months ago
.gitignore	initial commit	4 years ago
CHANGELOG.md	chore(docs,typing): modernize docstring...	2 months ago
LICENCE	licence	4 years ago
MANIFEST.in	fix(setup): fix documentation file path	2 months ago
README.md	docs(requirements): fix of requirements ...	2 months ago
pyproject.toml	test(unittests): basic unittests coverage	2 months ago
requirements.txt	chore(deps): update requirements	2 months ago
setup.cfg	chore(docs,typing): modernize docstring...	2 months ago
setup.py	fix(setup): fix documentation file path	2 months ago
version.py	python compatibility update	last year

README GPL-3.0 license

version 0.4.dev0 python 3.10 | 3.11 | 3.12 | 3.13 license GNU/GPLv3 os Linux | Windows

Puppet Warp

The goal of the package **puppet-warp** is to provide a plug-and-play solution for image transformation similar to Adobe Photoshop's *Puppet Warp* tool. Since the Photoshop solution is proprietary (and scripting can be painful, especially on unsupported platforms), this project implements Puppet Warp in Python so it can be used programmatically in automation pipelines where advanced deformation is required.

Features

- As-Rigid-as-Possible (ARAP) shape manipulation of a triangular mesh

- Image transfer from a triangular mesh at rest to a mesh defined by ARAP deformation

> Note: Please report issues and feel free to open pull requests.

Requirements

```
numpy>=1.21.5
opencv-contrib-python>=4.5.4.60,<=4.12.0.88
opencv-python>=4.5.4.60,<=4.12.0.88
scikit-image>=0.19.2,<=0.26.0
scikit-learn>=1.0.2,<=1.8.0
```

Optional:

```
triangle>=20200424
```

Installation

```
pip install puppet-warp
```

For the latest version from git:

```
pip install git+https://github.com/mikecokina/puppet-warp.git@dev
```

Install with Jonathan Richard Shewchuk's Triangle bindings:

```
pip install puppet-warp[jrs]
```

Usage

Demo

The package comes with a live interactive demo:

```
from pwarp import Demo
Demo().run()
```

To manipulate the image:

- Select control points by clicking on vertices in the mesh.
- Drag a selected control point to deform the mesh.

Demo also supports saving the transformed mesh:

- **Space**: save current mesh (Wavefront OBJ)
- **Esc**: quit

By default, outputs are stored in `~/pwarp`.

Custom demo

```
import cv2

from pwarp import Demo, triangular_mesh
from pwarp._io import save_wavefront

# Define WIDTH and HEIGHT of your image and DELTA step to create a triangular mesh.
width = 800
height = 492
delta = 100
method = "scipy"  # or "jrs"

# Define paths to your image and the OBJ file.
wavefront_path = "image.obj"
image_path = "image.jpg"

image = cv2.cvtColor(cv2.imread(image_path), cv2.COLOR_BGR2RGB)

# Generate triangular mesh over the image.
r, f = triangular_mesh(width=width, height=height, delta=delta, method=method)

# Save wavefront object.
save_wavefront(wavefront_path, no_vertices=len(r), no_faces=len(f), vertices=r, faces=f)

Demo(
    image=image_path,
    obj_path=wavefront_path,
    screen_height=height,
    screen_width=width,
```

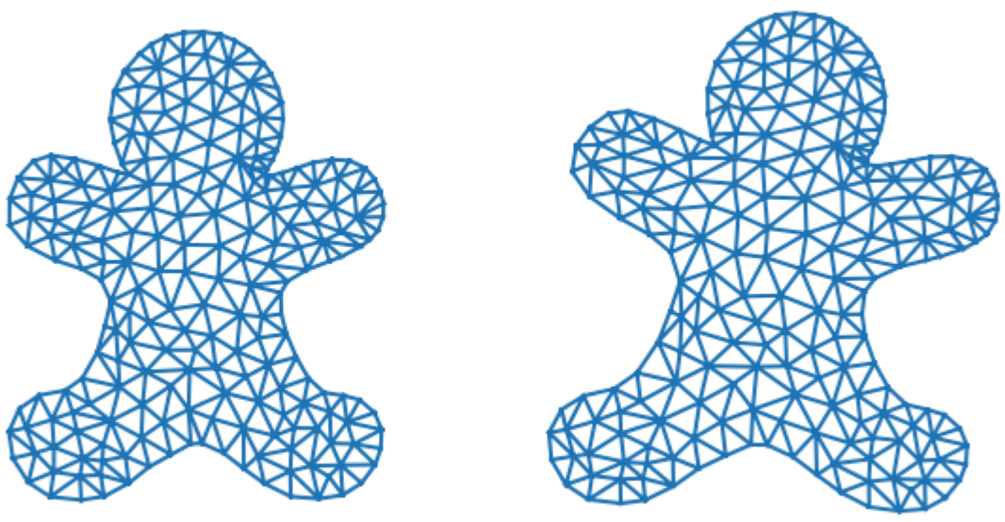
```
    scale=1,
    dx=0,
    dy=0,
    verbose=True,
).run()
```

Graph warp

Graph warp requires vertices and faces (triangulation), control points, and new positions of control points. Based on that information, graph warp computes new positions of the supplied vertices.

Example:

```
import numpy as np

from pwarp import get_default_puppet, graph_warp
from pwarp.core.precompute import arap_precompute

# Control points represent indices of points in original vertex array.
control_pts = np.array([22, 50, 94, 106], dtype=int)

# Shift represents new positions of control points respectively to `control_pts` list.
shift = np.array(
    [
        [0.555, -0.905],
        [-0.965, -0.875],
        [-0.950, 0.460],
        [0.705, 0.285],
    ],
    dtype=float,
)

puppet = get_default_puppet()

# Precompute once per mesh (recommended).
pre = arap_precompute(vertices=puppet.r, faces=puppet.f)

new_vertices = graph_warp(
    vertices=puppet.r,
    faces=puppet.f,
    control_indices=control_pts,
    shifted_locations=shift,
    precomputed=pre,
)
```



Graph defined warp

Graph defined warp transforms image regions covered by source vertices to given destination vertices. It requires:

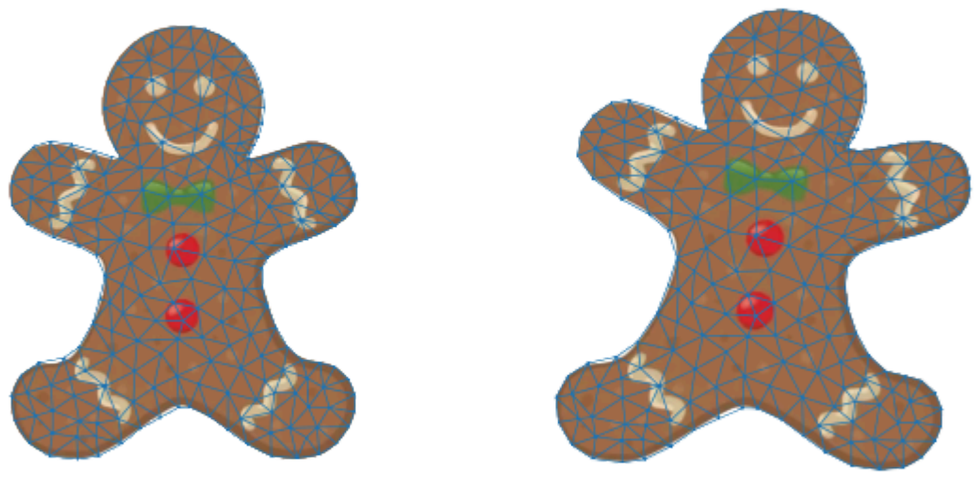
- input image
- source vertices + faces
- destination vertices + faces

Faces (triangles) must correspond pairwise between source and destination.

Example:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

from pwarp import get_default_puppet, graph_defined_warp, graph_warp
from pwarp.core.precompute import arap_precompute
```

```python
control_pts = np.array([22, 50, 94, 106], dtype=int)
shift = np.array(
    [
        [0.555, -0.905],
        [-0.965, -0.875],
        [-0.950, 0.460],
        [0.705, 0.285],
    ],
    dtype=float,
)

puppet = get_default_puppet()
pre = arap_precompute(vertices=puppet.r, faces=puppet.f)

new_r = graph_warp(
    vertices=puppet.r,
    faces=puppet.f,
    control_indices=control_pts,
    shifted_locations=shift,
    precomputed=pre,
)

image = cv2.cvtColor(cv2.imread("../data/puppet.png"), cv2.COLOR_BGR2RGB)
width, height = 1280, 800
dx, dy = int(width // 2), int(height // 2)
scale_x, scale_y = 200, -200

r = puppet.r.copy()
r[:, 0] = r[:, 0] * scale_x + dx
r[:, 1] = r[:, 1] * scale_y + dy

new_r = new_r.copy()
new_r[:, 0] = new_r[:, 0] * scale_x + dx
new_r[:, 1] = new_r[:, 1] * scale_y + dy

image_t = graph_defined_warp(
    image,
    vertices_src=r,
    faces_src=puppet.f,
    vertices_dst=new_r,
    faces_dst=puppet.f,
)

fig, axs = plt.subplots(1, 2, frameon=False)
plt.tight_layout(pad=0)

axs[0].imshow(image)
axs[1].imshow(image_t)
axs[0].triplot(r.T[0], r.T[1], puppet.f, lw=0.5)
axs[1].triplot(new_r.T[0], new_r.T[1], puppet.f, lw=0.5)

for ax in axs:
    ax.set_xlim([380, 900])
    ax.set_ylim([150, 750])
    ax.invert_yaxis()
    ax.axis("off")

plt.show()
```

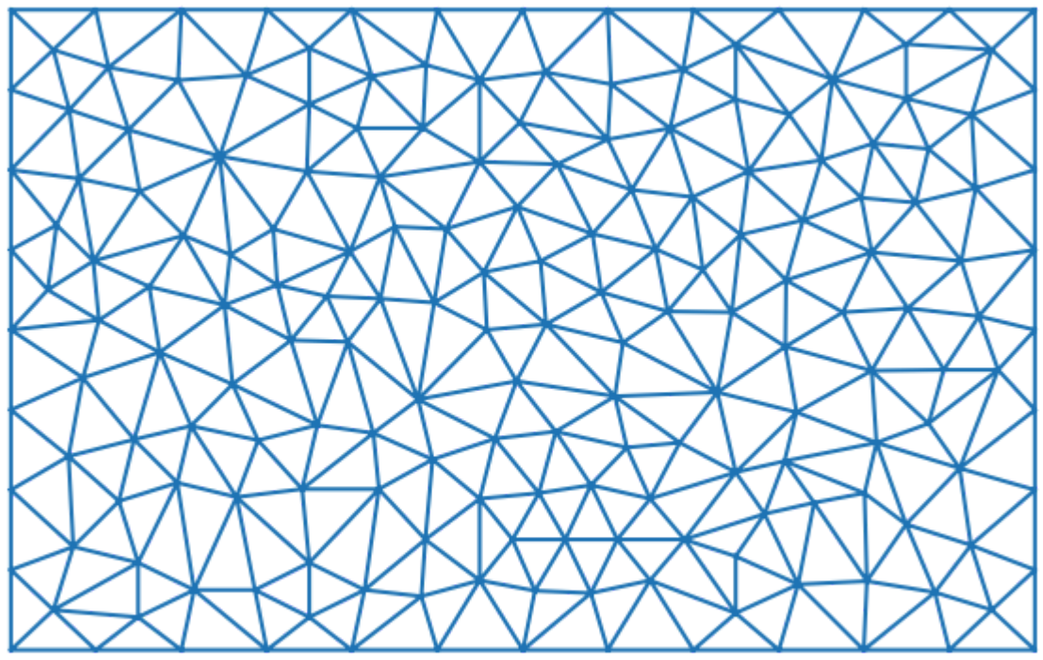


Triangular mesh

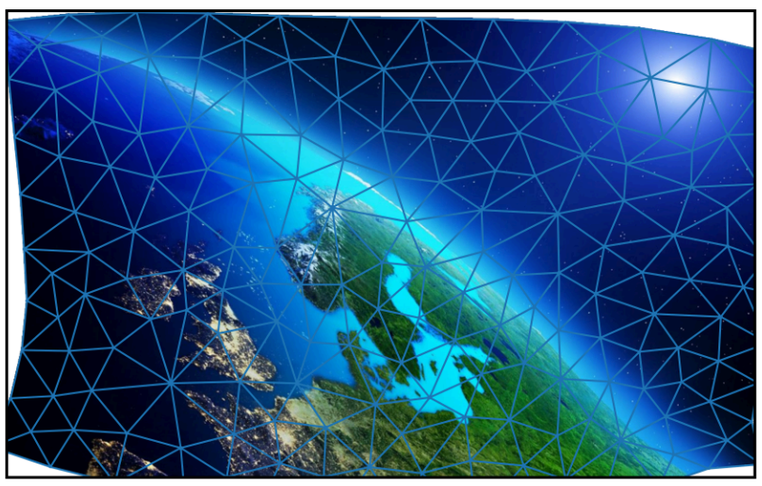
The algorithm generates a triangular mesh within a rectangle defined by its width and height. Mesh density is adjustable via the `delta` parameter.

Example:

```python
from pwarp import triangular_mesh

r, f = triangular_mesh(width=1280, height=800, delta=100)
```



Example on full screen triangular mesh warp:



References

```
[1] https://www-ui.is.s.u-tokyo.ac.jp/~takeo/papers/takeo_jgt09_arapFlattening.pdf
[2] https://github.com/deliagander/ARAPShapeManipulation.git
[3] https://learnopencv.com/warp-one-triangle-to-another-using-opencv-c-python/
[4] https://rufat.be/triangle/
[5] http://www.cs.cmu.edu/~quake/triangle.html
```

Cite

```
@article{journals/jgtools/IgarashiI09,
    author = {Igarashi, Takeo and Igarashi, Yuki},
    ee = {http://dx.doi.org/10.1080/2151237X.2009.10129273},
    journal = {J. Graphics, GPU, & Game Tools},
    number = 1,
    pages = {17-30},
    title = {Implementing As-Rigid-As-Possible Shape Manipulation and Surface Flattening.},
    url = {http://dblp.uni-trier.de/db/journals/jgtools/jgtools14.html#IgarashiI09},
    volume = 14,
    year = 2009
}
```

or

```
@article{10.1145/1073204.1073323,
    author = {Igarashi, Takeo and Moscovich, Tomer and Hughes, John F.},
    title = {As-Rigid-as-Possible Shape Manipulation},
    year = {2005},
    publisher = {Association for Computing Machinery},
    address = {New York, NY, USA},
    volume = {24},
    number = {3},
    doi = {10.1145/1073204.1073323},
    journal = {ACM Trans. Graph.},
    month = {jul},
    pages = {1134–1141},
    numpages = {8}
}
```