

Image Preprocessing for Immersed Boundary Method

In this notebook, general image processing techniques are employed on 2D images and subsequently applied to a numerical simulation program utilizing A new unified governing equation of flow incorporating immersed solid boundaries.

> 1. Preparation before execution

[] 4 6 個のセルが非表示

✓ 2. Image processing

The image file is converted to grayscale, and a blur is applied as a variant distance function to generate a porosity distribution.

✓ 2-1. File Input

Enable use_upload: [check] to choose a local file. Alternatively, manually input the file path into file_path: .

> File Input

use_upload: ☐

file_path: "/content/henoheno.png"

コードの表示

```
henoheno successfully read
Original resolution: (256, 256)
Save the figure in ./tmp/original.png
```



✓ 2-2. Cropping and Adding Margins

Add padding to the loaded file. The original resolution of the file can be found in the code block above [File Input] with Original resolution: (WIDTH, HEIGHT) , allowing you to verify it. Add a margin of MARGIN pixels to match that resolution. Choose surroundings for padding around the image.

> Cropping and Adding Margins

MARGIN: 170

コードの表示

```
Original resolution: (256, 256)
Processed resolution: (596, 596)
Save the figure in /content/tmp/cropped.png
```



✓ 2-3. Grayscale

Grayscale can be easily achieved by loading the previously resized image using OpenCV's IMREAD_GRAYSCALE function. Here, we explain the process that takes place within the image during the grayscaleing process.

Each pixel in the image has a specific RGB (Red, Green, Blue) value, where each color has 256 steps ranging from 0 to 255 (2⁸). By performing the following operations on these RGB values, a grayscale image can be generated. This method follows the "Luminosity Method (ITU-R BT.601)."

$$\text{Gray} = \text{Red} \times 0.229 + \text{Green} \times 0.587 + \text{Blue} \times 0.114$$

The following code block (the grayscale tester) conducts a test to convert an original BGR color (specified by user input) to grayscale by applying custom weights to each channel (B, G, R). Users can set these custom weights through sliders. The resulting grayscale intensity is calculated based on the specified weights, and the original and grayscale images are displayed side by side for visual comparison.

✓ 2-3-1. Threshold

When generating a grayscale image, set all values greater than the threshold to 255 (white). Adjust the grayscale weighting and threshold value to effectively lighten the background by enhancing its white tones.

> Grayscale and Threshold

$$\text{Gray} = \text{Red} \times w_1/w + \text{Green} \times w_2/w + \text{Blue} \times w_3/w$$
$$w = w_1 + w_2 + w_3$$

w1: 0

w2: 1

w3: 0

threshold: 122

inverse: ☐

コードの表示

Custom weights: [0.0, 1.0, 0.0]



2-3-2. Outline (Option)

As the boundary areas exhibit significant gradients in brightness data, applying a differentiation kernel allows for the extraction of these boundary regions. By setting the extracted boundary parts to 0 and the rest to 1, multiplying this mask with the original image creates a composite image where only the brightness data of the boundary parts is set to 0.

Meaning of Variables

- **kernel_size**: The larger the size, the thicker the extracted boundary lines.
- **threshold1**: If you make it smaller, more main lines will be extracted.
- **threshold2**: smaller size extracts more branching lines.

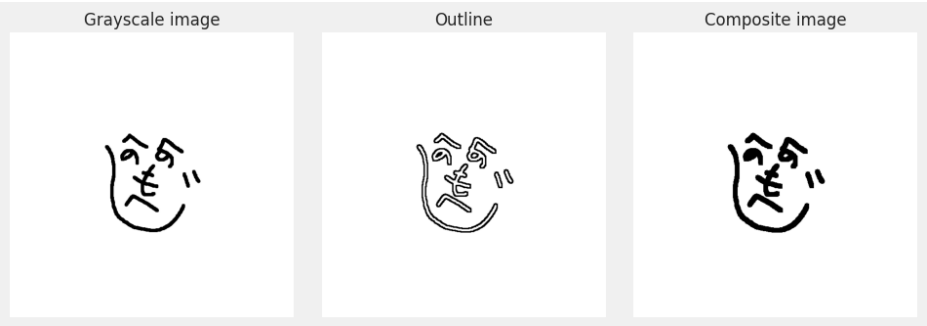
Option: outline

kernel_size:

threshold1: 312

threshold2: 8

コードの表示



2-4. Perform resizing and calculate fluid occupancy

1. **Resize**: Adjust the dimensions of the image using appropriate resizing techniques.
2. **Fluid Occupancy**: Determine the fluid occupancy rate by analyzing the resized image. This involves assessing the portion of the image occupied by the fluid, providing valuable insights into the fluid dynamics within the simulated environment.

2-4-1. Resize

Resizing is done by the function `resize` of the OpenCV module. This function can scale an image by giving a scale factor. For example, a scale factor of 0.5 on a 100x100 image results in a 50x50 image.

Therefore, in order to convert an image to an arbitrary resolution, it is sufficient to know this scale factor. Let (W, H) denote the number of pixels in the original image and (W', H') the number of pixels in the converted image. Resolution is the total number of pixels, so if the original resolution is R , the following equation holds.

$$R = WH$$

Here, the number of pixels in the height and width of the converted image (W', H') is defined using the scale factor $s(> 0)$ and (W, H) as follows.

$$(W', H') = (sW, sH)$$

This allows the transformed resolution R' to be expressed as follows.

$$R' = W'H' = s^2WH$$

Solving this for s , we obtain.

$$s = \sqrt{\frac{R'}{WH}}$$

Therefore, given an arbitrary resolution, the coefficient s can be obtained and resizing can be performed. Run the code block below and set the resolution by changing the slider bar of `resolution`. The last value you set will be the resolution used in the following program. The images are stored in the `tmp` folder.

2-4-2. Fluid Occupancy

Defenition of porosity distribution:

$$\epsilon = \frac{1}{2} \tanh(x) + \frac{1}{2}, x = \frac{\xi}{\Delta}$$

where ξ is the signed distance function.

> Max Resolution Check

コードの表示

```
Max resolution: (2698, 2698)
```

< Porosity Generation

```
# @title Porosity Generation { vertical-output: true }
```

```
import seaborn as sns
import numpy as np
import csv
import cv2
from tqdm import tqdm
import matplotlib.pyplot as plt
import os
```

```
resolution = 256 # @param {type: "integer"}
outline = False # @param {type: "boolean"}
generation_type = "GaussianBlur"
thickness = 2.0 # @param {type: "raw"}
```

```
CWD = os.getcwd()
```

```
def CustomBlurPorosity(img, thickness, blur_type, resolution):
    # Function to apply custom blur and generate porosity
    alpha = 0.8576
    h, w = img.shape[:2]
    scale_factor = resolution / max(h, w)
```

```
    ksize = int(thickness/scale_factor)*4 + (int(thickness/scale_factor)*4 + 1) % 2
    sigma = alpha * thickness / scale_factor
    img_blur_median = cv2.medianBlur(img, ksize=3)
```

```
    if blur_type == 'Blur':
        img_blur = cv2.blur(img_blur_median, ksize=(ksize, ksize))
    elif blur_type == 'Gaussian':
        img_blur = cv2.GaussianBlur(img_blur_median, ksize=(ksize, ksize), sigmaX=sigma)
```

```
    img_blur_norm = img_blur / 255.0
    porosity = cv2.resize(img_blur_norm, (int(w * scale_factor), int(h * scale_factor)), cv2.INTER_LINEAR)
```

```
    return porosity, ksize, sigma
```

```
# File paths
if outline:
    file_name_input = f"{CWD}/tmp/com.png"
else:
    file_name_input = f"{CWD}/tmp/gray.png"
```

```
file_name_output = f"{CWD}/tmp/porosity_image.png"
```

```
# Read the input image
input_image = cv2.imread(file_name_input, cv2.IMREAD_UNCHANGED)
```

```
# Resize the image to the specified resolution
h, w = input_image.shape[:2]
scale_factor = resolution / max(h, w)
```

```
porosity, ksize, sigma = CustomBlurPorosity(input_image, thickness, 'Gaussian', resolution)
```

```
# Divergence control of porosity distributions
h_resized, w_resized = porosity.shape[:2]
```

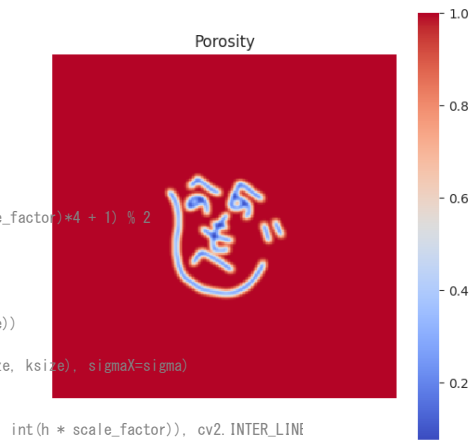
resolution: 256

outline: ☐

thickness: 2.0

Porosity Distribution

Complete
Boundary area = 666.12
Make a porosity distribution
Complete



----- Processing Results -----
Setting resolution = 65536 (256, 256)
Porosity resolution = 65536 (256, 256)
Threshold : 122
Shade factor : 1
Generation Type : GaussianBlur
Kernel size : 17
Sigma : 3.993
Boundary area = 666.1

```
print('Complete')
```

```
S = np.sum(porosity - porosity**2)
print("Boundary area = ", format(S, ".2f"))
```

```
shade_factor = 1
```

```
# Initialization
data_write = np.empty((int(h_resized+1), int(w_resized)), dtype=object)
data_display = porosity
```

```
##markdown # Porosity Distribution
```

```
print('make a porosity distribution')
```

```
# Write porosity distribution
```

```
with open('data/porosity.csv', "w") as file:
    # Write header
    file.write(f"{int(w_resized)}, {int(h_resized)}, 1, porosity\n")
```

```
    # Write data
    for j in range(1, h_resized+1):
        for i in range(1, w_resized+1):
            file.write(f"{int(i)}, {int(j)}, 1, {(porosity[h_resized - (j), (i - 1)]):.5f}\n")
```

```
with open('logs/process.txt', 'w', encoding='utf-8') as f:
    print('----- Processing Results -----', file=f)
    print('Setting resolution = ', porosity.shape[0]*porosity.shape[1], porosity.shape, file=f)
    print('Porosity resolution = ', (data_write.shape[0]-1)*data_write.shape[1], data_write[1,:].shape, file=f)
    print('Threshold : ', threshold, file=f)
    print('Shade factor : ', shade_factor, file=f)
    print('Generation Type : ', generation_type, file=f)
    if generation_type == 'GaussianBlur':
        print('Kernel size : ', ksize, file=f)
        print('Sigma : ', format(sigma, ".4"), file=f)
    print('Boundary area = ', format(S, ".4"), file=f)
    print('-----', file=f)
```

```
print('Complete')
```

```
# Display images side by side using subplot
fig, ax = plt.subplots(figsize=(6, 6))
```

```
# Visualize porosity distribution using heatmap
sns.heatmap(porosity, cmap="coolwarm", xticklabels=False, yticklabels=False, square=True, ax=ax)
ax.set_title("Porosity")
```

```
plt.savefig(file_name_output)
plt.show()
```

```
# Display the content of the process.txt file
%cat logs/process.txt
```

< 3. Simulation

Program [2] for laminar flow calculations of the MAC algorithm using the Immersed boundary method.

< 3-1. Program

The program code is written using FORTRAN. The data/porosity.csv generated by the above code is read in and the flow field is calculated. The calculation results are output as a file in vtk format.

Running Simulation Program will generate a file `main.f90` in the `src` folder. running Compile will compile `src/main.f90` and save it in `bin.` folder and an executable file named `run` is generated in the `bin` folder. In a terminal, type `./run` in the terminal to run the program.

> `main.f90`

コードの表示

Writing src/main.f90

> `Compile`

コードの表示

gfortran -O3 -fopenmp -fno-automatic -o bin/run src/main.f90
f951: Warning: Flag '-fno-automatic' overwrites '-frecursive' implied by '-fopenmp'

▼ `3-2. Settings`

A `controlDict.txt` file enables to change some values from it. We need not recompile the program when we change any variables because of using `namelist`.

$$CFL = \frac{U \Delta t}{\Delta x} < 1$$
$$Re = \frac{Ul}{\nu} = \frac{\rho Ul}{\mu}$$
$$Pe = \frac{U \Delta x}{\nu} < 2$$
$$D = \frac{\nu \Delta t}{\Delta x^2} < \frac{1}{2}$$

- Where,
- U = Inlet velocity
 - l = Representative length in flow direction
 - ν = Kinematic viscosity
 - μ = Dynamic viscosity
 - ρ = Density of the fluid

Large Peclet numbers mean that the advection term is dominant, which can lead to unstable calculations. As the program uses central differences, the stability condition for the Peclet number is $Pe < 2$.

Variable Description

- re: Reynolds number
- dt: time increment
- timeval: Simulation time
- safety_factor: safety factor (3 or more is recommended)
- ratio_of_objects: Ratio of the representative length of objects to the entire region
- AoA: Angle of Acceptance
- slip: Wall condition (noslip or slip)
- Run: When ☒ is checked, the simulation will start at runtime.
- Log: When ☒ is checked, the log file will be writed into `logs/log.txt`

> `Run`

re: 30

dt: 0.01

timeval: 10

safety_factor: 3

ratio_of_objects: 0.3

AoA: 15

slip: noslip

Run: ☒

Log: ☒

コードの表示

Width (w): 256.0, Height (h): 256.0

Settings	
CFL number	= 0.065104
Reynolds number	= 30
Peclet number	= 0.39062
Diffusion factor	= 0.16667
dx	= 3.873e-02 m
dt	= 1.000e-02 s
inlet velocity	= 0.25215 m/s
length	= 9.9148 m
height	= 9.9148 m
steps	= 1000

Running...writing logs to logs/log.txt in progress

xnue = 2.50000004E-02
xlamda = 0.00000000
density = 1.00000000
width = 9.91483688
height = 9.91483688
time = 10.0000000
inlet_velocity = 0.252146989
outlet_pressure = 0.00000000
Angle of inlet_velocity (AoA) = 15.0000000

m, n = 256 256
istep_max = 1000
dx, dy = 3.88817117E-02 3.88817117E-02
dt = 9.9999978E-03
cfl_no = 6.48497641E-02
pecret_no = 0.392156243
diffusion_factor = 0.165367171
thickness = 2.00000000
threshold = 9.9999978E-03
istep_max= 1000 istep_out= 1001
--- time_steps= 1 -- time = 9.9999978E-03
residual= p (fluid): 0.00000000 u: 0.243555292 v: 6.52604401E-02
--- time_steps= 2 -- time = 1.99999996E-02
residual= p (fluid): 3.99294066 u: 0.434651017 v: 0.253714859
--- time_steps= 3 -- time = 2.99999993E-02
...
--- time_steps= 998 -- time = 9.97999954
residual= p (fluid): 1.00854784E-04 u: 1.48020685E-04 v: 1.74194574E-04
--- time_steps= 999 -- time = 9.98999977
residual= p (fluid): 9.80496407E-05 u: 1.47871673E-04 v: 1.74053013E-04
--- time_steps= 1000 -- time = 10.00000000
residual= p (fluid): 9.54195857E-05 u: 1.47737563E-04 v: 1.73896551E-04

program finished
CPU times: user 1.47 s, sys: 204 ms, total: 1.68 s
Wall time: 2min 43s

4. Visualization

For visualization, utilize the Python module pyvista. It reads the VTK file generated by the Fortran program earlier and produces a colored image based on the values.

4-1. Surface profile

The porosity value of 0.5 represents the interface between the solid and fluid phases. As the interface is smoothly distributed, coordinates where the porosity values range from 0.25 to 0.75 are output to the file `output/surface_profile.csv`. By reading and visualizing this file as a colored scatter plot, you can inspect the boundary region.

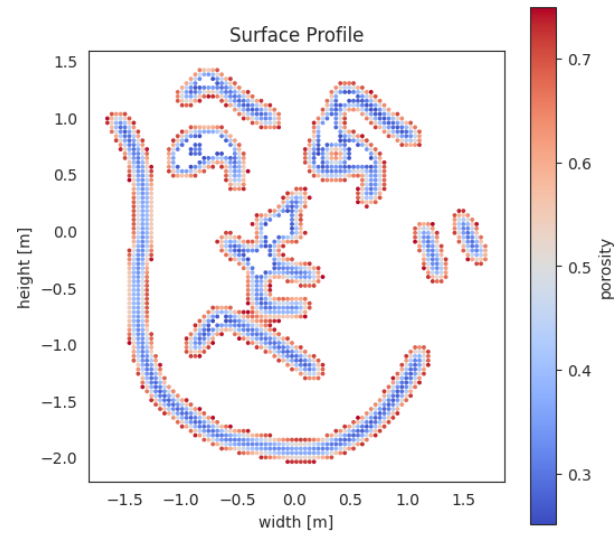
Variable Description

- colors:** Select the color map for data.

Surface profile

colors: coolwarm

コードの表示



4-2. Display Output Data

The simulation results are output to a file named `output/solution_uvp.csv`. By reading this file, you can visualize the data as a colored scatter plot.

Variable Description

- data:** [velocity, pressure, porosit y]
- colors:** Select the color map for data.
- streamlines:** When ☒ is checked, display streamlines
- streamlines_color:** Select the color of the streamlines

- density:** Specify the number of streamlines

Display Results

data: porosity

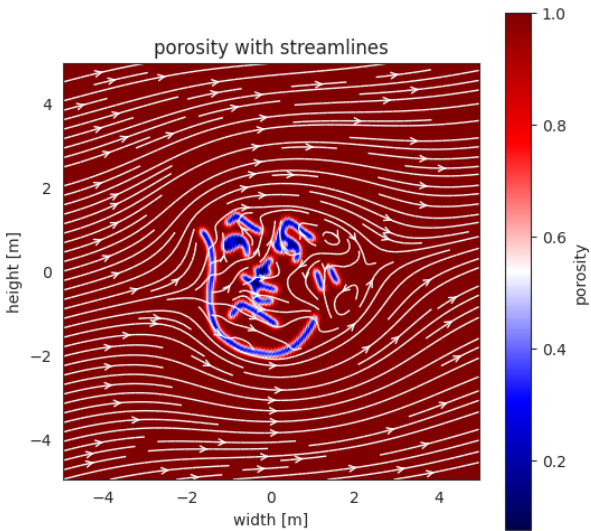
colors: seismic

streamlines: ☒

streamlines_colors: white

streamlines_density: 1.5

コードの表示



Zip output file for download

コードの表示

5. Reference

[1] Oshima,N, A Novel approach for wall-boundary immersed flow simulation: proposal of modified Navier-Stokes equation, Mechanical Engineering Journal. Vol.18, No.4 (2023)

[2] 大島, 流れの数値解析:固体境界が埋め込まれた改良ナビエ・ストークス方程式の解法, 北海道大学学術成果コレクション(HUBCAP), 資源タイプsoftware (2023), URL: <https://eprints.lib.hokudai.ac.jp/dspace/handle/2115/89344>