Project 4: convex hull

1. Algorithms used:

- Naive algorithm:
  - Time complicity: $O(n^3), n =$ number of points
- Gift wrapping algorithm:
  - Time complicity (average): $O(n * h)$, where n - number of points; h: number of conv
    $* log(n))$
  - Time complicity (worst case): $O(n^2)$

2. Project implementation

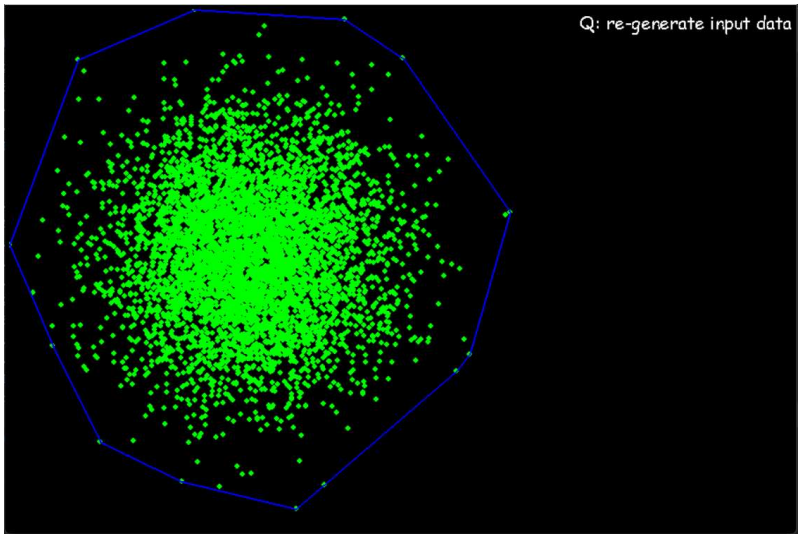
2.1 Project structure

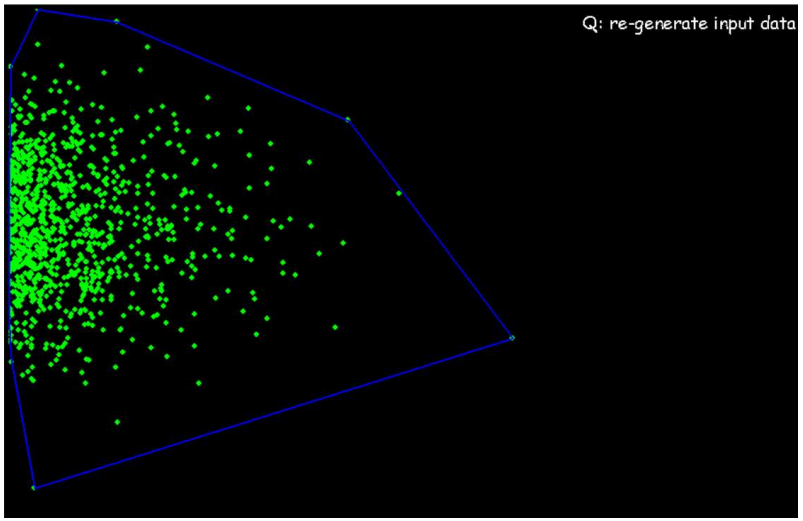
- /test_data/ : Test cases & algorithm execution results
- __init__.py - module initialization
- gui.py - GUI implementation, visualization of gift wrapping algorithm
- project4.py - Algorithms implementation (convex hull, gift wrapping); main() entrypoint of the project; Test data generator (TestCases)
- project4_test.py - main(): generate tests & write using the class TestOutputWriter

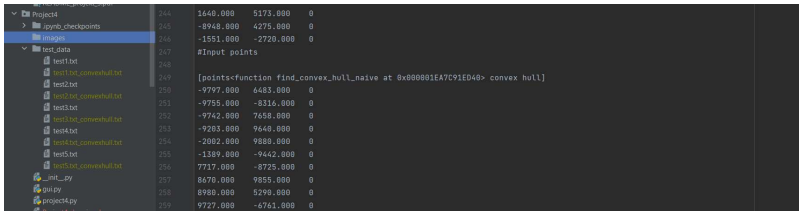
2.2 Short description of python modules

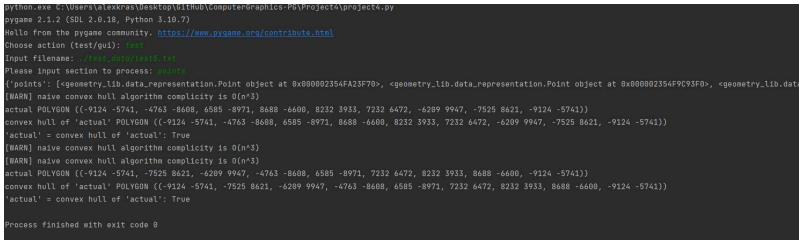
- gui.py:
  - class DisplayConvexHullResults:
    - A pygame-based graphical interface to display the results of a given convex hull algorithm

    1. method init_bbox: init bounding box & scale for a given set of points
    2. setup: setup pygame screen & hotkeys
    3. draw(): display results
    4. _draw(surface): draw self in a given surface
    5. set_data(points, convex hull): update data & clear cache
  - main:
    - display a set of points and segments (convex hull) - gui test
- project4.py:
  - method find_convex_hull_naive:
    - Naive convex hull algorithm (defined in section 1.)
  - method - orientation:
    - returns:
      - 0: Collinear points
      - 1: Clockwise angle
      - 2: Counterclockwise angle
  - method - find_convex_hull_giftwrap:
    - Gift wrapping convex hull algorithm (defined in section 1.)
  - class - TestCases:
    - method generate_linear(count):
      - generate random points on a random line
    - method generate_linear_with_repeating_points:
      - generate points on a line with repeating points
    - method _rand:
      - random number generator (normal distribution)
    - method generate_normal:
      - generate points distributed according to a given 2-D distribution (normal x uniform etc)
    - generate_with_repeating_points:
      - equal to the previous method, but the output data contains repeating points
    - log_exec_time:
      - log execution time to a TestOutputWriter object
    - test_algorithm_correctness:
      - namely, execute a convex hull algorithm and write output to file
    - test_giftwrap_algorithm_correctness:
      - re-definition of the above method for gift-wrapping algorithm
    - test_naive_algorithm_correctness:
      - re-definition of the above method for naive algorithm
  - method - main:
    - test algorithm correctness for a generated set of points & print output to file
  - method - sub:
    - read points from file
    - use Shapely.Polygon object to get the convex hull
    - compare algorithm results (must be equal)
  - method - sub_gui:
    - generate points & find convex hull
    - create a GUI instance to display the results
  - method - __main__:
    - prompt: test/gui
- project4_test.py:
  - method - __main__:
    - generate points & find convex hull
    - write test results to file


Q: re-generate input data


Q: re-generate input data


Project4
.ipynb_checkpoints
images
test_data
test1.txt
test1.txt_convexhull.txt
test2.txt
test2.txt_convexhull.txt
test3.txt
test3.txt_convexhull.txt
test4.txt
test4.txt_convexhull.txt
test5.txt
test5.txt_convexhull.txt
__init__.py
gui.py
project4.py
1640.000 5173.000 0
-8948.000 4275.000 0
-1551.000 -2720.000 0
#Input points
[points<function find_convex_hull_naive at 0x000001EA7C91ED40> convex hull]
-9797.000 6483.000 0
-9755.000 -8316.000 0
-9742.000 7658.000 0
-9203.000 9640.000 0
-2002.000 9888.000 0
-1389.000 -9442.000 0
7717.000 -8725.000 0
8670.000 9855.000 0
8980.000 5298.000 0
9727.000 -6761.000 0


python.exe C:\Users\alexkras\Desktop\GitHub\ComputerGraphics-PG\Project4\project4.py
pygame 2.1.2 (SDL 2.0.18, Python 3.10.7)
Hello from the pygame community. https://www.pygame.org/contribute.html
Choose action (test/gui): test
Input filename: ./test_data/test5.txt
Please input section to process: points
{'points': [<geometry_lib.data_representation.Point object at 0x000002354FA23F70>, <geometry_lib.data_representation.Point object at 0x000002354F9C93F0>, <geometry_lib.dat
[WARN] naive convex hull algorithm complicity is O(n^3)
actual POLYGON ((-9124 -5741, -4763 -8608, 6585 -8971, 8688 -6600, 8232 3933, 7232 6472, -6209 9947, -7525 8621, -9124 -5741))
convex hull of 'actual' POLYGON ((-9124 -5741, -4763 -8608, 6585 -8971, 8688 -6600, 8232 3933, 7232 6472, -6209 9947, -7525 8621, -9124 -5741))
'actual' = convex hull of 'actual': True
[WARN] naive convex hull algorithm complicity is O(n^3)
[WARN] naive convex hull algorithm complicity is O(n^3)
actual POLYGON ((-9124 -5741, -7525 8621, -6209 9947, -4763 -8608, 6585 -8971, 7232 6472, 8232 3933, 8688 -6600, -9124 -5741))
convex hull of 'actual' POLYGON ((-9124 -5741, -7525 8621, -6209 9947, -4763 -8608, 6585 -8971, 7232 6472, 8232 3933, 8688 -6600, -9124 -5741))
'actual' = convex hull of 'actual': True
Process finished with exit code 0