Assignment 2

Evolutionary Algorithms

April 8, 2021

Author: Vladislav Lamzenkov, BS1905

Part 1. Implementation

For this assignment, I was required to implement an evolutionary algorithm that generates a picture based on the input picture. Firstly, I was trying to choose an optimal stack of technologies that allows me to write pure code to work with images and provides good performance. Frankly speaking, I fastly chose Python since this language has pure syntax with a really good library that helps to work with images: *OpenCV* [1]. However, this programming language doesn't provide a high speed of execution of the code. To speed up the code, I optimized some parts of the code, but the main thing is the *OpenCV* library which is written in C/C++. In other words, when I am calling some functions from this library in Python, what runs is the underlying C/C++ source. As a result, there will be no big problems with the performance. Besides, I used the *NumPy* library [2] which is used in the implementation of the *OpenCV* library, it is a better way to work with lists in Python. Also, I wrote the code using the OOP approach, thus, it will be easier to support and extend the program. Additionally, I am using the *argparse* library [3] to parse and validate the arguments from the command line.

Actually, I wrote 2 main classes: *ExtendedImage* and *GeneticAlgorithm*. The first class(*ExtendedImage*) contains the image itself(the image is represented as NumPy array) and some useful methods:

- *view()* - public method to show image in a separate window.
- *get_color_in_region(start, end)* - public method that returns the average color of the image in the rectangle area specified by *start* and *end* arguments.
- *get_width()* - public method that returns the width of the image.
- *get_height()* - public method that returns the height of the image.
- *create_empty_image(width, height)* - public method that returns the fully filled black image(meaning: empty one) of the specified size(via arguments *width* and *height*).

The second class(*GeneticAlgorithm)* consists of the following methods:

- *_generate_population()* - protected method to generate the initial population and store it inside the object.
- *_fit_test(img)* - protected method to evaluate the goodness of the given chromosome(*img*) and returns the fitness value.
- *_mutate(img)* - protected method that creates a copy of the given chromosome(*img*) and mutates its genes via adding one new rectangle(at a random place) on the copy. The size of the rectangle usually varies from 4px up to 8 pixels, however, the size can be less than 4 pixels. After all, the method returns a new independent offspring.

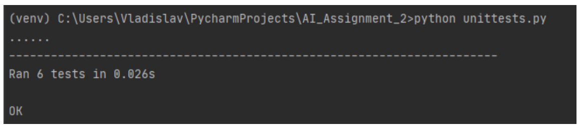
- *_crossing_over(img1, img2)* - protected method which is responsible for chromosomal crossover between *img1* and *img2*, as a result, returns a new independent offspring.
- *_selection()* - protected method to generate the next population based on the previous one(or initial one). In the end, the obtained array is sorted according to the fitness value of obtained chromosomes. This method saves the new generation inside the object.
- *run()* - public method to run the genetic algorithm, as a result, it returns the tuple of 2 elements: output image, fitness value of the image.

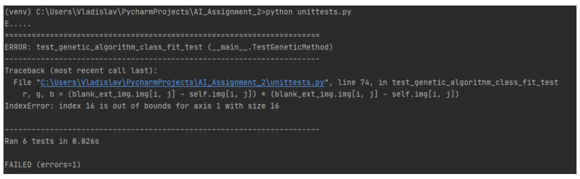
After all, the algorithm works in the following way. Firstly, the method *run()* is called, as a result, the initial population is generated. Secondly, the algorithm iterates MAX_NUM_OF_ITERATIONS times or until STOPPING_CRITERIA is reached. On each iteration the program calls *_selection()* method. In this method, 50% of the best chromosomes from the previous population are copied to the new generation, and the algorithm mutates 25% of the best ones via *_mutate()*. In addition, the algorithm applies chromosomal crossover to 25% of the best offsprings from the previous population using method *_crossing_over()*. To each mutated or obtained via crossing over progeny the algorithm calculates its fitness value calling method *_fit_test()*. Then all chromosomes are sorted in increasing order. Finally, when the *run()* method terminates it returns the resulting picture(the best chromosome from the last generation) and its fitness value.

Besides, I implemented a testing module for my program to be sure about the correctness of the written code. So, I decided to test my program using one of the most traditional ways: unit testing. Therefore, I used the Python framework *unittest* [4]. I separated all tests from the main code into a separate file called *unittests.py*. I tested all functionality provided by the *ExtendedImage* and *GeneticAlgorithm* classes.

Figure 1. Testing the program using unit tests.

Command to run the tests	**Possible output**

python unittests.py	
	`(venv) C:\Users\Vladislav\PycharmProjects\AI_Assignment_2>python unittests.py`
	`......`
	`----------------------------------------------------------------------`
	`Ran 6 tests in 0.026s`
	
	`OK`
	If everything is ok, you will get such a successful message.

	
	`(venv) C:\Users\Vladislav\PycharmProjects\AI_Assignment_2>python unittests.py`
	`E.....`
	`======================================================================`
	`ERROR: test_genetic_algorithm_class_fit_test (__main__.TestGeneticMethod)`
	`----------------------------------------------------------------------`
	`Traceback (most recent call last):`
	`  File "C:\Users\Vladislav\PycharmProjects\AI_Assignment_2\unittests.py", line 74, in test_genetic_algorithm_class_fit_test`
	`    r, g, b = (blank_ext_img.img[i, j] - self.img[i, j]) * (blank_ext_img.img[i, j] - self.img[i, j])`
	`IndexError: index 16 is out of bounds for axis 1 with size 16`
	
	`----------------------------------------------------------------------`
	`Ran 6 tests in 0.026s`
	
	`FAILED (errors=1)`
	If something is wrong, you will get a detailed message with the error.

Part 2. Description of Algorithm’s Parameters

By definition, a chromosome consists of genes. In my implementation, the gene is a rectangle. The size of the rectangle usually varies from 4 pixels up to 8 pixels. However, the algorithm also generates a little number of figures which have a size of fewer than 4 pixels. I tested the algorithm with different sizes of figures, and the most optimal size is between 4 and 8 pixels. Also, I am taking a rectangle area on the original picture and calculate the average color inside it just to optimize the code. I tried to generate color randomly, but in that case, the approximation to the original picture is unpredictable. Moreover, the algorithm will require more iterations to generate the output, as a consequence, the execution time of the program will grow.

By the way, I need to say why the algorithm only works with rectangles - I tried different shapes(from circles up to polygons), but all of them had a problem with finding the average color of the figure. For example, imagine that my algorithm generates a polygon. Then, we need to set up the color of the obtained polygon. In other words, we want to find the average color of the original picture, we need to check the specific area taking into account each point and trying to understand: does this concrete point [X, Y] on the original picture belong to the generated polygon? To answer the question, we need to solve the task *“Point in polygon”* [5] for each point, and this is one of the biggest problems since it requires time. As a result, I chose rectangles as shapes since it will be much simpler and faster to find the average color for them.

In addition, it is not worthy of storing shapes separately from the resulting picture since the program will be wasting time on translating the shapes to pixels on an empty image during the computation of the fitness value. As a result, I decided to store all shapes exactly in the picture(meaning: in pixels).

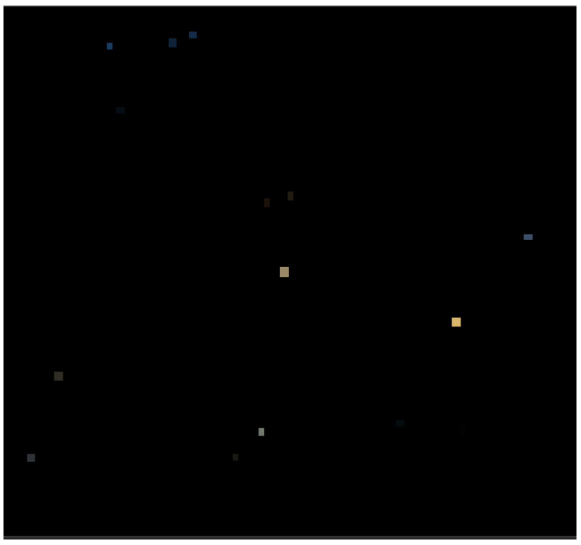
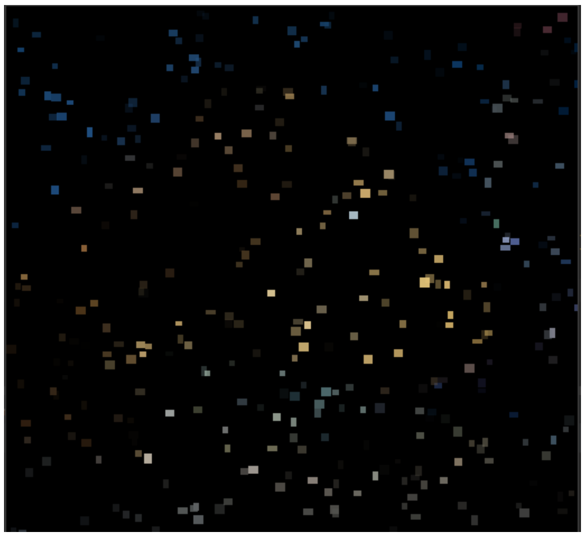
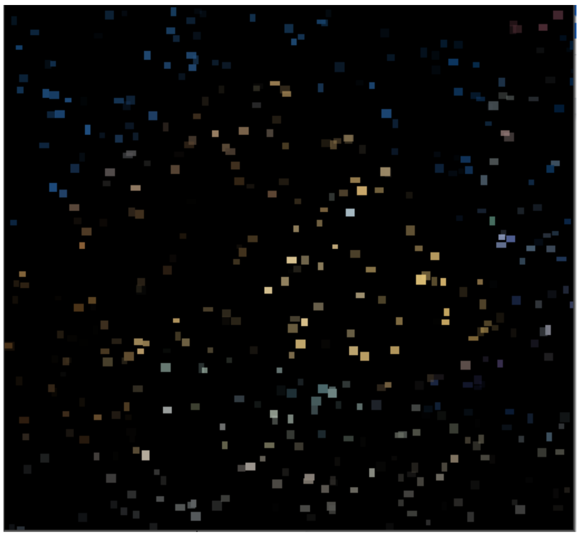
<u>The size of the population in my implementation is 10 chromosomes</u>. I decided to choose such a size after different tests. On the one hand, I tried to have a big population(1000 chromosomes) and applied 1 mutation to each chromosome on one iteration. But this approach requires a big amount of free memory to store all chromosomes in the memory(1 chromosome is 1 picture of the size 512*512 pixels). On the other hand, I used a small population(10 chromosomes) and applied from 1 to 100 mutations to each chromosome on one iteration. To tell the truth, the second approach required less amount of memory and each iteration was handled fastly. Moreover, the second algorithm was producing not worse results than the first one.

<u>The selection technique is to choose 1 chromosome with the smallest fitness value out of the generation.</u>

The fitness function is the squared sum of the difference between the original one and a produced image. A small sum means a better approximation to the original picture. I am using the squared difference and not just difference since the difference between the original pixel and obtained one can be negative(for example, original pixel: [255, 255, 255] and pixel on the generated image is [0, 0, 0]; in this case, the difference between the original and generated pixels is [-255, -255, -255]). It is also possible to use the absolute difference in this case, but it works slowly in Python. However, NumPy arrays work fastly with simple math operations. As a result, I chose such a fitness function.

The mutation is a changing of the gene at a chromosome. This gene is a rectangle which size is usually between 4 and 8 pixels with some opacity(from 0 up to 1). The size of the figure is defined randomly. But it needs to say, that the size can be less than 4 pixels to better highlight the frame of the picture(meaning: pixels that surraround the image). The position of the figure is generated randomly, but it is limited only by the width and height of the original picture. The color of the rectangle is the average color of the same figure in the original picture. Besides, I am applying mutations to 25% of the generation. My algorithm applies from 1 up to 100 mutations to each of the given offsprings on each iteration, as a result, my program already provides some results even after a small number of iterations.

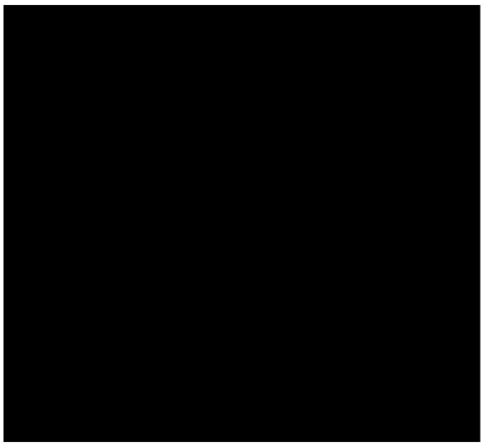
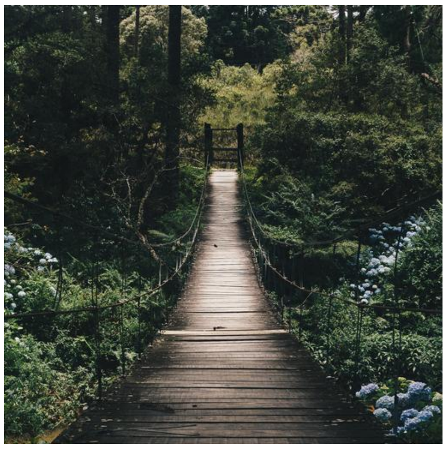
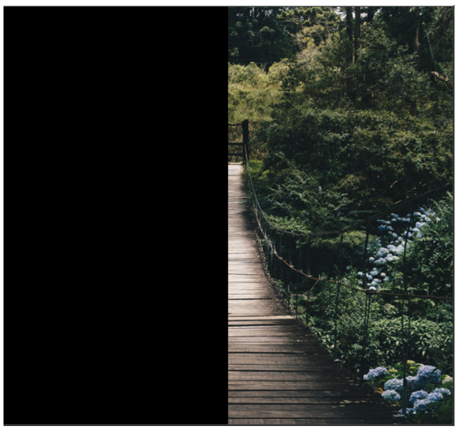
Figure 2. Example of mutations.

Input image(before mutation)	**Output image(after some number of mutations)**
	
	

<u>The crossover is implemented in the following way: the left half of the first chromosome is united with the right half of the second chromosome</u>

creating a new offspring. I am applying crossing overs to 25% of the population. By the way, I tested my algorithm with a different percentage, but at the beginning, my algorithm produces more accurate results when I increase the percentage of mutations. However, after some time the offsprings obtained by crossing over have better fitness values than mutation due to uniting of different offsprings. As a result, I decided to apply 25% of mutations and 25% of chromosomal crossover to one population.

Figure 3. Example of chromosomal crossover.

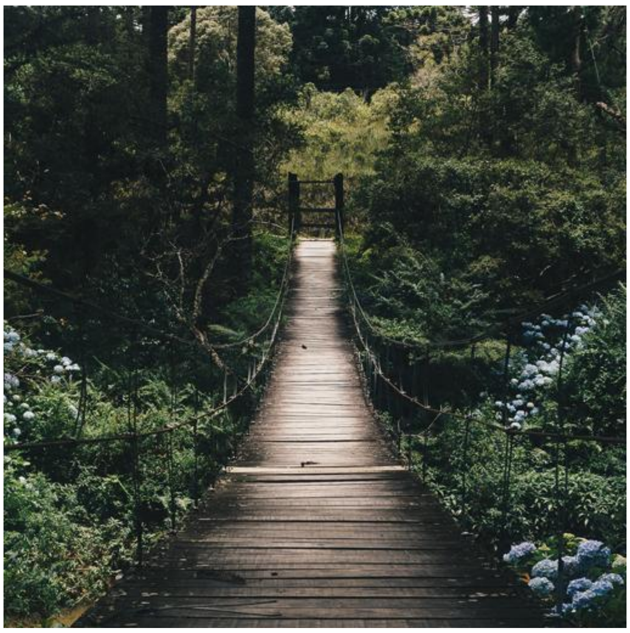
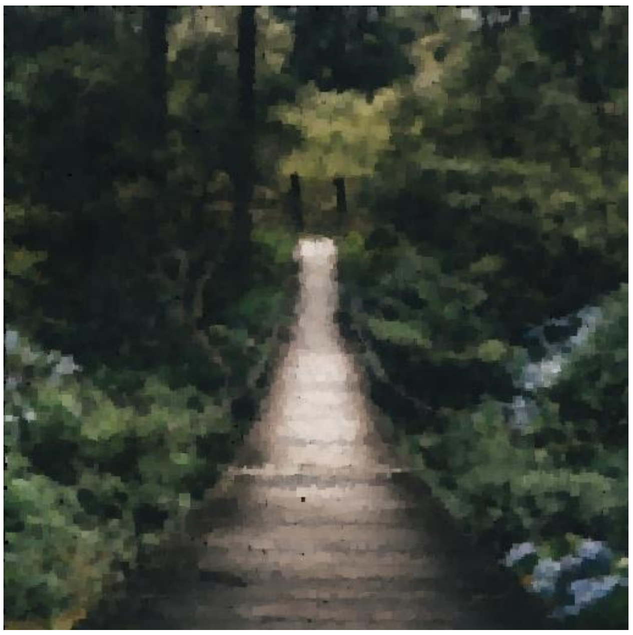
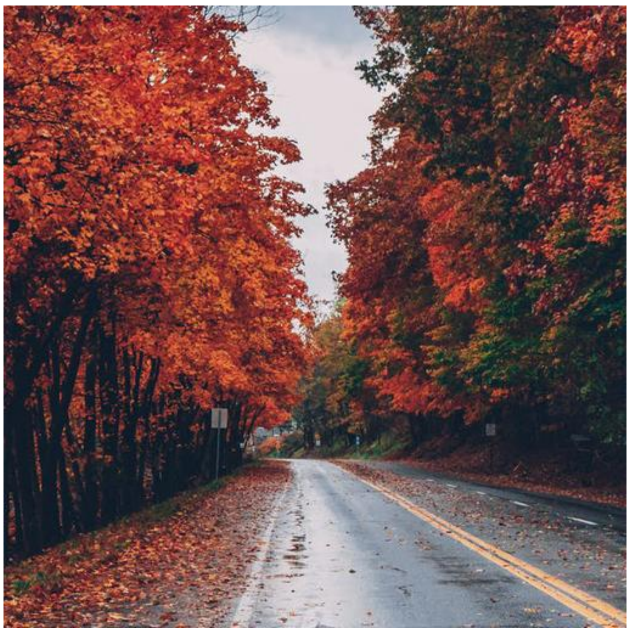
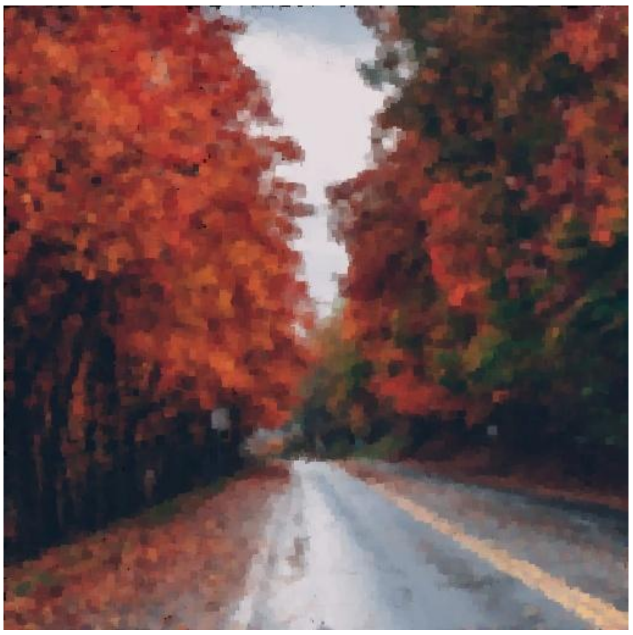
First input image(before crossing over)	**Second input image(before crossing over)**
	
Resulting output image(After crossing over)	
	

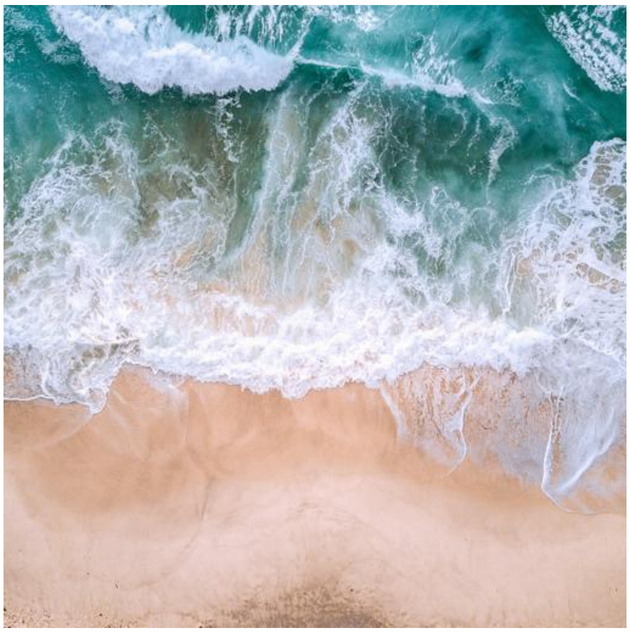
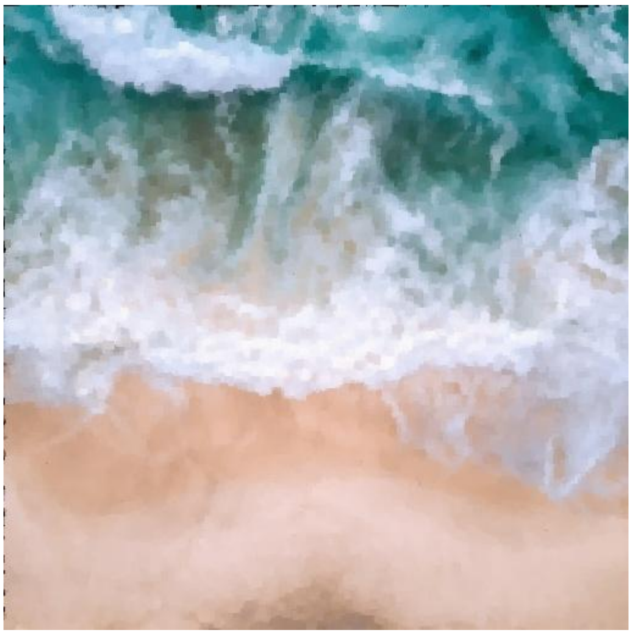
Besides, I need to say that while creating a new population I am taking 50% of the best offsprings from the previous population. Due to the fact that I am using crossing over, it is easier to get progenies with good fitness values just by exchanging the parts of the best offsprings.

In conclusion, I can say that my algorithm is genetic since it is applying chromosomal crossovers.

Part 3. Examples of Input and Output Images

Figure 4. Examples of input and output images.

Input image	**Corresponding output image**
	
	

Part 4. What Is Art

One must admit that there is no exact definition of what is art. However, I want to start with my own understanding of what is art. For me, art is an object that is trying to provide you some information about the world and evoke your emotions. An object can be anything in the world, for example, it can be: physical item, image, sculpture, voice - anything that matters. Thus, art is not necessarily the picture of a famous artist, it can be even a piece of dust if the particle matters. If a person sees beauty in something unusual - it will affect his emotions, what is more, this can change his understanding of the world. One should, however, not forget that everyone decides by themselves what is art.

Pictures that are generated by my algorithm are true objects. It is important to note that the objects may not be physical. The computer shows the output result on the screen - we cannot touch the image itself, obviously, we can only touch the screen. However, this output picture really changes my emotions. For example, if the output image seems to be good for me and similar to the original picture - I am glad. Otherwise, I can be unhappy. Sometimes I even try to understand the beauty of the generated picture - the beauty of randomness. Moreover, the interpretation of the original picture and generate one can be entirely different. This interpretation is a piece of hidden information inside of the picture.

After all, I consider my work as art since the picture can store some hidden pieces of hidden meaning inside it and it can evoke people's emotions.

Part 5. References

1. “OpenCV-Python Tutorials,” OpenCV. [Online]. Available: https://docs.opencv.org/master/d6/d00/tutorial_py_root.html. [Accessed: 14-Apr-2021].

2. “NumPy v1.20 Manual,” Overview - NumPy v1.20 Manual. [Online]. Available: https://numpy.org/doc/stable/. [Accessed: 14-Apr-2021].

3. “argparse - Parser for command-line options, arguments and sub-commands¶,” argparse - Parser for command-line options, arguments and sub-commands - Python 3.9.4 documentation. [Online]. Available: https://docs.python.org/3/library/argparse.html. [Accessed: 14-Apr-2021].

4. “unittest - Unit testing framework¶,” unittest - Unit testing framework - Python 3.9.4 documentation. [Online]. Available: https://docs.python.org/3/library/unittest.html. [Accessed: 14-Apr-2021].

5. “Point in polygon,” Wikipedia, 01-Apr-2021. [Online]. Available: https://en.wikipedia.org/wiki/Point_in_polygon#:~:text=In%20computational%20geometry%2C%20the%20point,the%20boundary%20of%20a%20polygon. [Accessed: 14-Apr-2021].