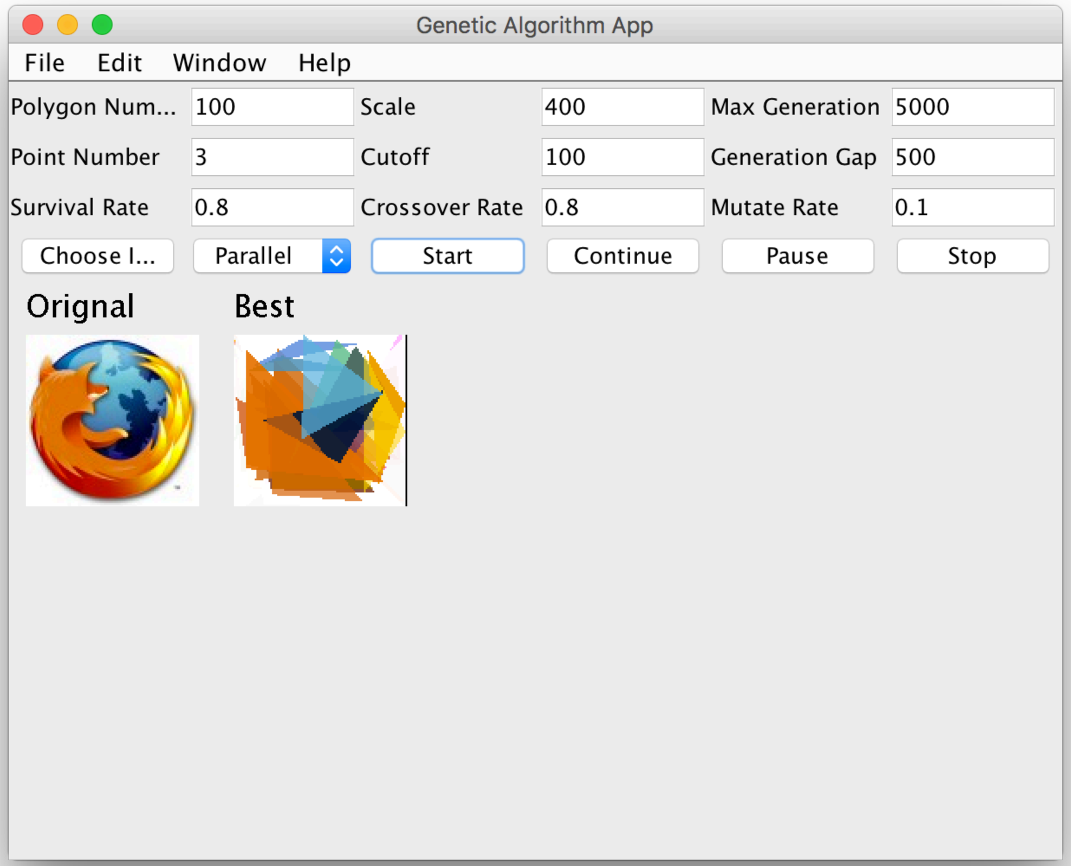
Genetic Algorithm: Draw Images Via Genetic Programming

Team Members: Feng Huang(001230993) Zixuan Yu(001263991)
Team Number : 327
Final Draft: April 14, 2018

Problem Inspiration:

My teammate and I created an application using genetic algorithm to get an image represented as a collection of overlapping polygons of various colors and transparencies. We start from random 100 polygons. In each optimization step, we randomly modify one parameter (like color, stacking or position of vertices) and check whether such new variant looks more like the target image. If it is, we keep it, and continue to mutate this one instead. Fitness is a sum of pixel-by-pixel differences from the original image. Lower number is better. Finally, we made this image as the graph below by genetic algorithm:(left is target image, right is our image)



Polygon Num...	100	Scale	400	Max Generation	5000
Point Number	3	Cutoff	100	Generation Gap	500
Survival Rate	0.8	Crossover Rate	0.8	Mutate Rate	0.1

In our algorithm, users are given freedom to configure these independent variables:

1. Polygon Number
2. Point Number
3. Scale
4. Cutoff (for parallel processing)
5. Max Generation
6. Generation Gap (for parallel processing)
7. Survival Rate
8. Crossover Rate
9. Mutate Rate

Implementation Design

Genetic code:

By changing the parameter (like color, stacking or position of vertices) and checking whether such new variant looks more like the target image, we keep the appropriate parameter. By comparing with the pixel of target image using RGB, we can get the absolute value of adding difference. Finally, by sorting difference(fitness) to keep better individuals.

Gene expression:

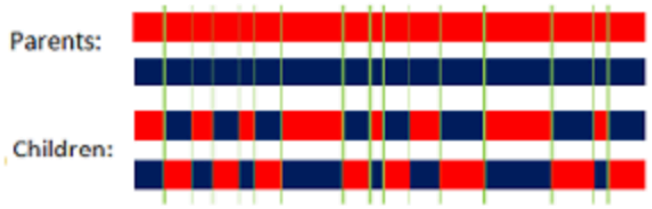
Each target image made by buffered images, and each buffered image is represented as 100 overlapping polygons of various colors and transparencies.

Fitness function:

Fitness is a sum of pixel-by-pixel differences from the original image. Lower number is better. By comparing with the pixel of target image using RGB, we can get the absolute value of adding difference.

Crossing Over:

We use the uniform crossover, which evaluates each bit in the parent strings for exchange with a probability of 0.5. The offspring has approximately half of the genes from first parent and the other half from second parent, although cross over points can be randomly chosen as seen below:



With a probability of 0.5, children have 50% genes from first parent and 50% of genes from second parent even with randomly chosen crossover points.

Selector:

We use the Tournament selection to choose two parents in crossover. it run several "tournaments" among a few individuals (or "chromosomes") chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected for crossover.

Mutation:

When we copy the gene from the parents to child, we will get a random number. If it lower than mutation rate, one of parameter (like color, stacking or position of vertices) will change.

Evolution:

The population is stored as a priority queue, and we will sort the population by their fitness. We select a rate of these population. A part of population can be to evaluate, while other population is discarded. The first one (the lowest difference) will copy directly, and paste into next generation. Besides, each population who can be kept has a random rate to mutation or crossover, and generate a next generation.

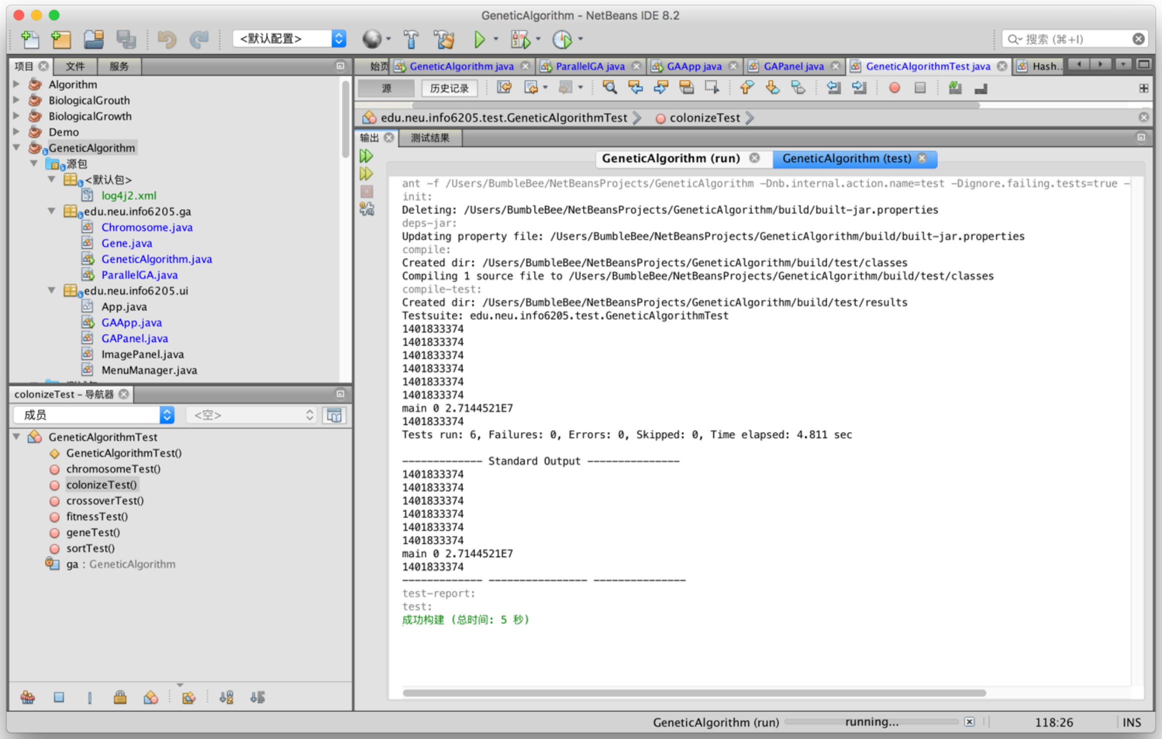
For example, the first generation is 100 individuals. We will sort them by their fitness, and select former 80%(80 individuals) to keep. Each of these individuals will get a number (0-1). When the individual who has number more than 0.8, it will crossover, else it will mutate.

Parallel Processing:

We use the parallel processing to improve the speed of calculation. We use two variables (cutoff and gap). The population will be divided into several parts (each part is a thread), each part will get their own generation by evolution function. And after acquiring several generations (gap means several generations), we will take these parts together to build a new generation. Later, repeat this loop until require the result.

Results

Our project builds and passes our unit test cases:



```
ant -f /Users/BumbleBee/NetBeansProjects/GeneticAlgorithm -Dnb.internal.action.name=test -Dignore.failing.tests=true -
init:
Deleting: /Users/BumbleBee/NetBeansProjects/GeneticAlgorithm/build/built-jar.properties
deps-jar:
Updating property file: /Users/BumbleBee/NetBeansProjects/GeneticAlgorithm/build/built-jar.properties
compile:
Created dir: /Users/BumbleBee/NetBeansProjects/GeneticAlgorithm/build/test/classes
Compiling 1 source file to /Users/BumbleBee/NetBeansProjects/GeneticAlgorithm/build/test/classes
compile-test:
Created dir: /Users/BumbleBee/NetBeansProjects/GeneticAlgorithm/build/test/results
Testsuite: edu.neu.info6205.test.GeneticAlgorithmTest
1401833374
1401833374
1401833374
1401833374
1401833374
1401833374
main 0 2.7144521E7
1401833374
Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 4.811 sec

------------- Standard Output ---------------
1401833374
1401833374
1401833374
1401833374
1401833374
1401833374
main 0 2.7144521E7
1401833374
------------- ---------------- ---------------
test-report:
test:
成功构建 (总时间: 5 秒)
```

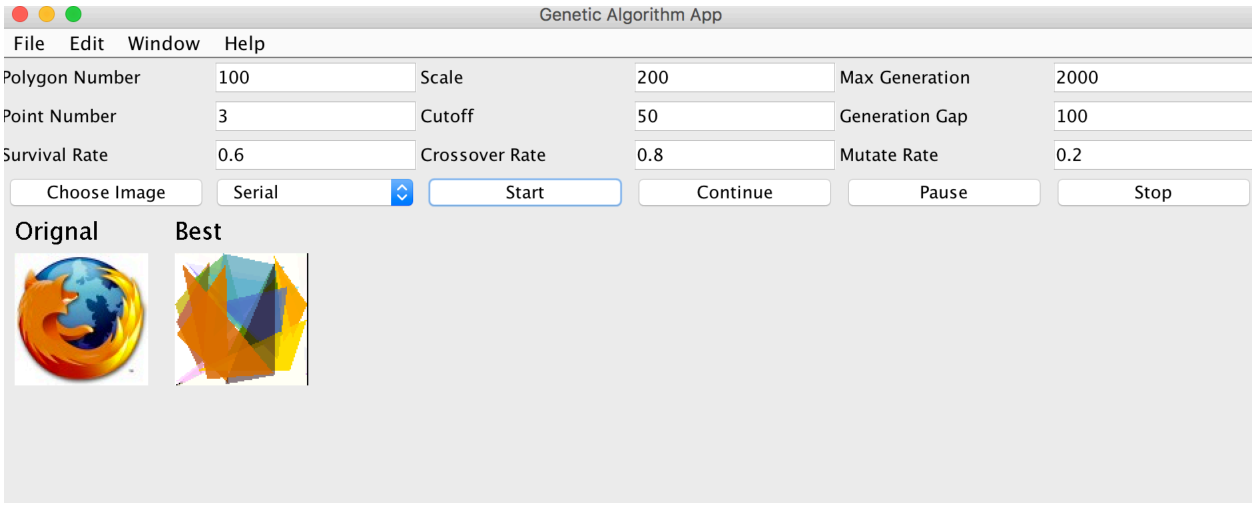
Observations

We use our application to test the algorithm in these parameter with 0.6 survive rate, 0.8 crossover rate and 0.2 mutate rate at first. Fitness is a sum of pixel-by-pixel differences from the original image. Lower number is better. And then, we found the result is not our expectation.

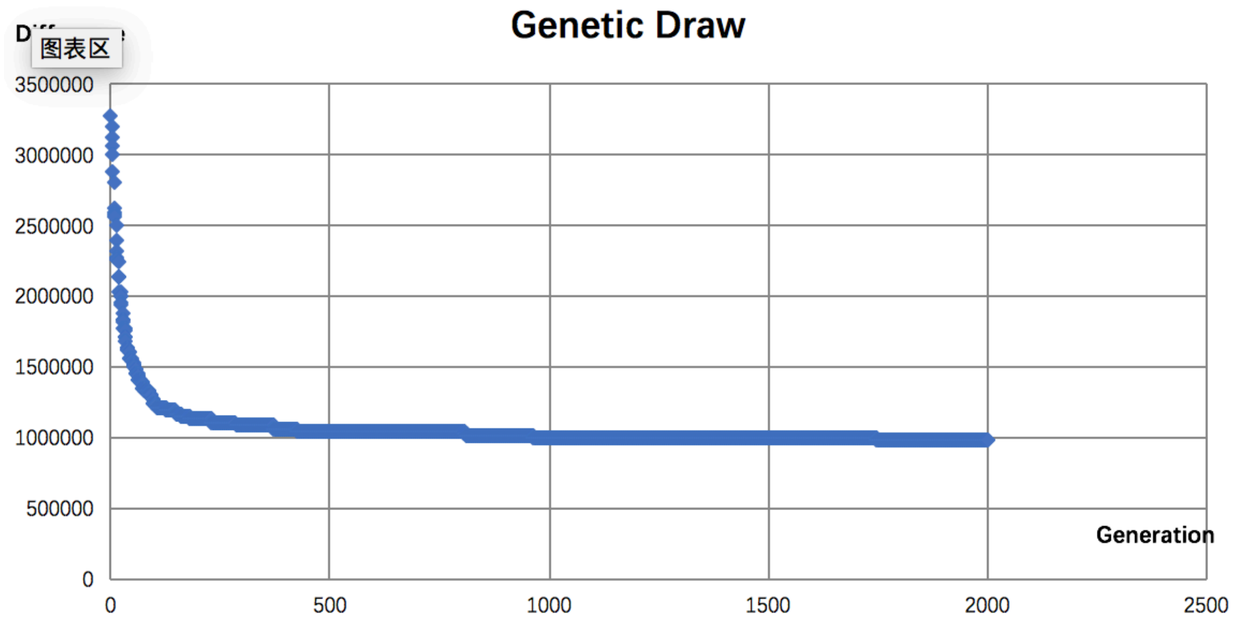
We change these parameter, and we found that 0.8 survive rate, 0.8 crossover rate and 0.01 mutate rate are best solution, but it needs too much time, so we use parallel processing.

First step

Survive rate	Crossover rate	Mutate rate	time	Difference
0.6	0.8	0.2	67min	984367

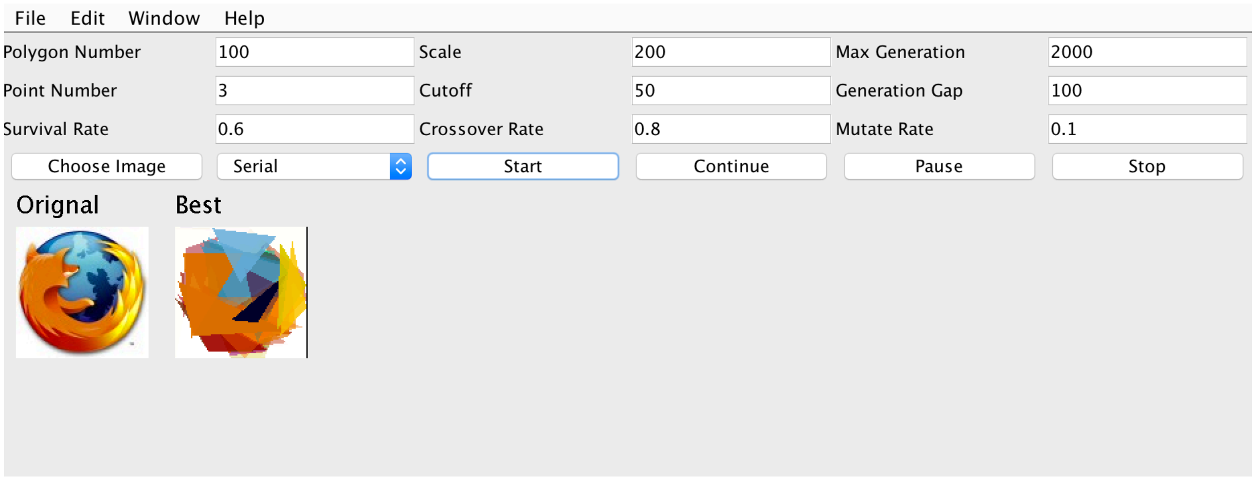


Polygon Number	100	Scale	200	Max Generation	2000
Point Number	3	Cutoff	50	Generation Gap	100
Survival Rate	0.6	Crossover Rate	0.8	Mutate Rate	0.2

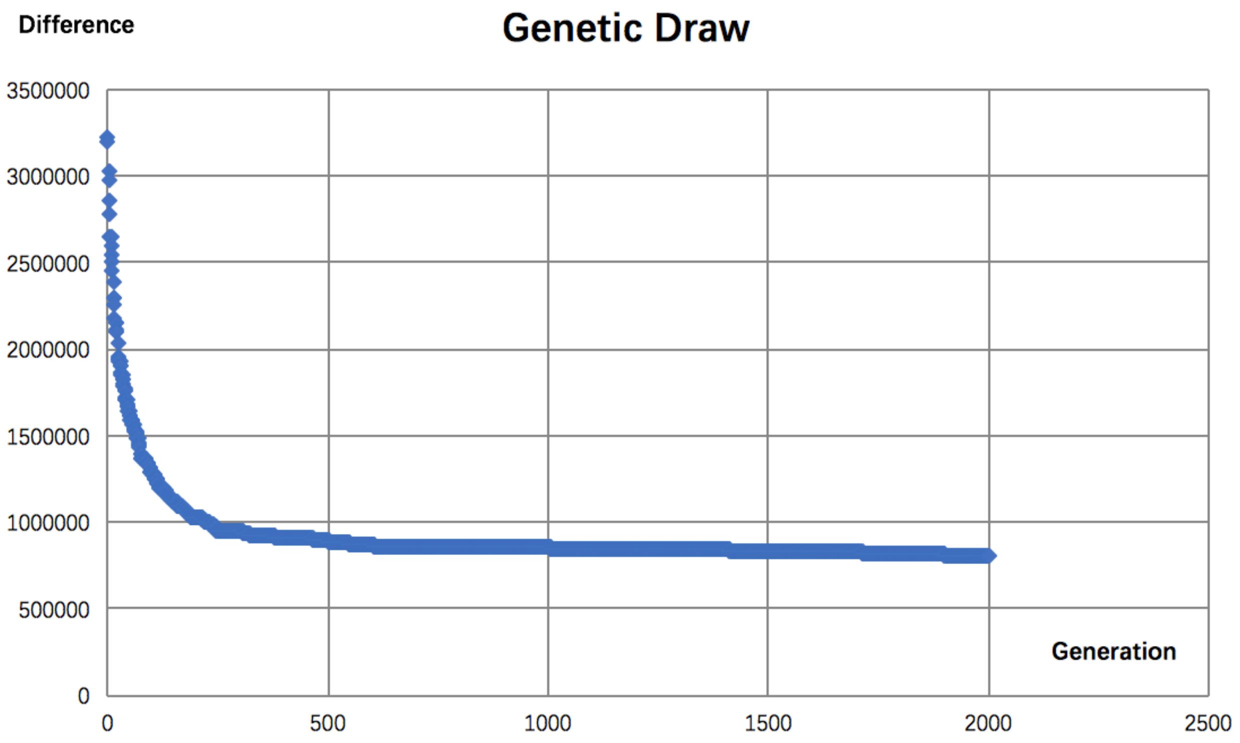


Second step

Survive rate	Crossover rate	Mutate rate	time	Difference
0.6	0.8	0.1	60min	805737

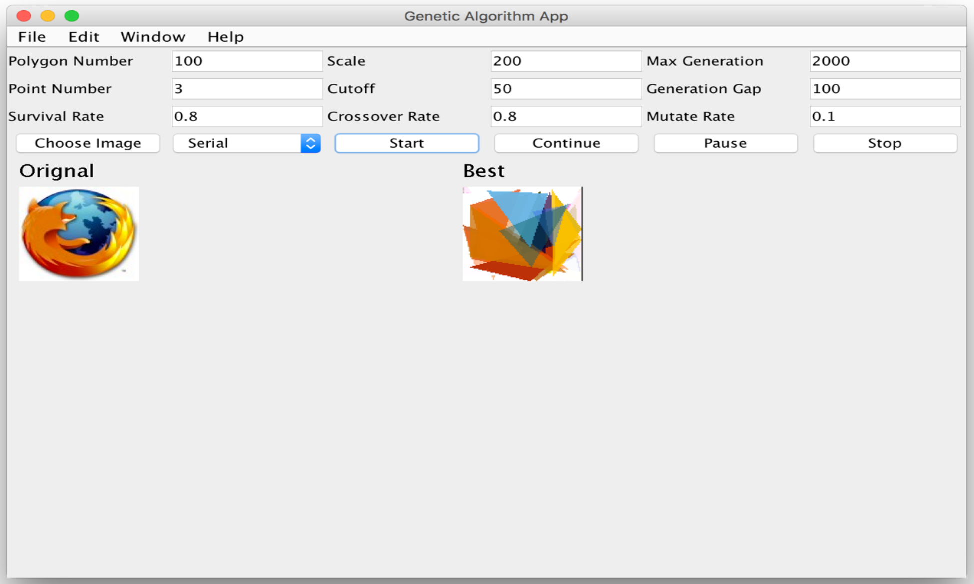


Polygon Number	100	Scale	200	Max Generation	2000
Point Number	3	Cutoff	50	Generation Gap	100
Survival Rate	0.6	Crossover Rate	0.8	Mutate Rate	0.1

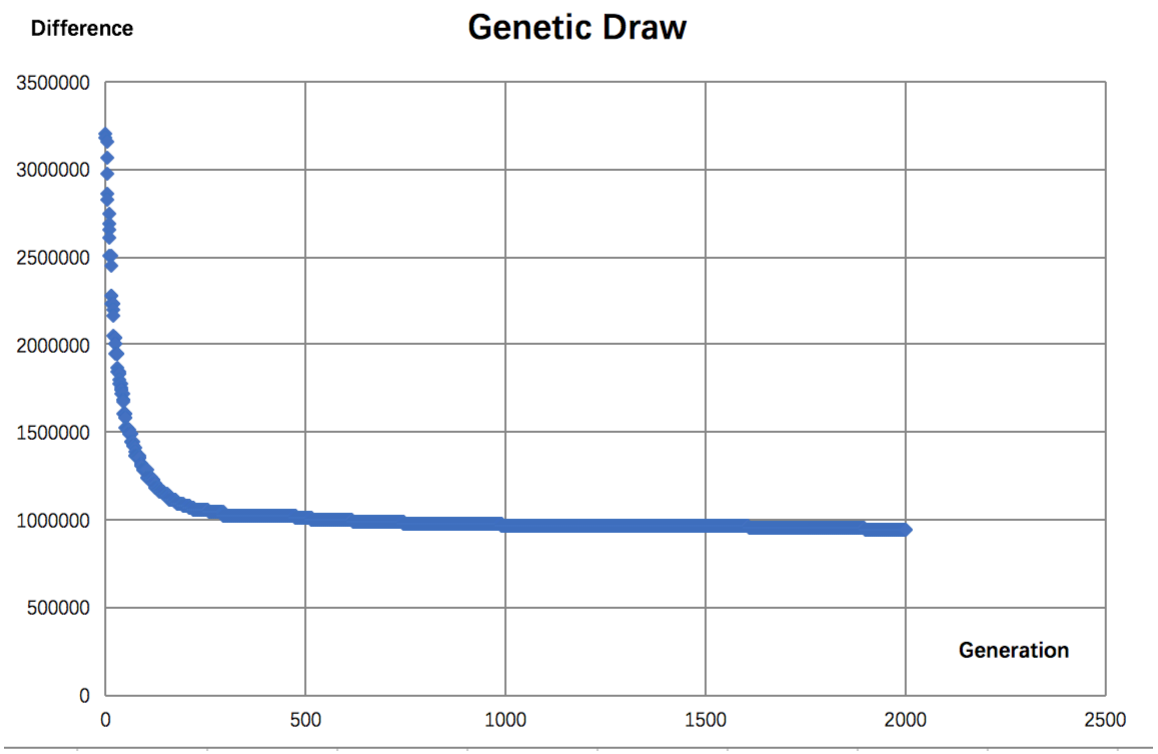


Third step

Survive rate	Crossover rate	Mutate rate	time	Difference
0.8	0.8	0.1	45min	945015

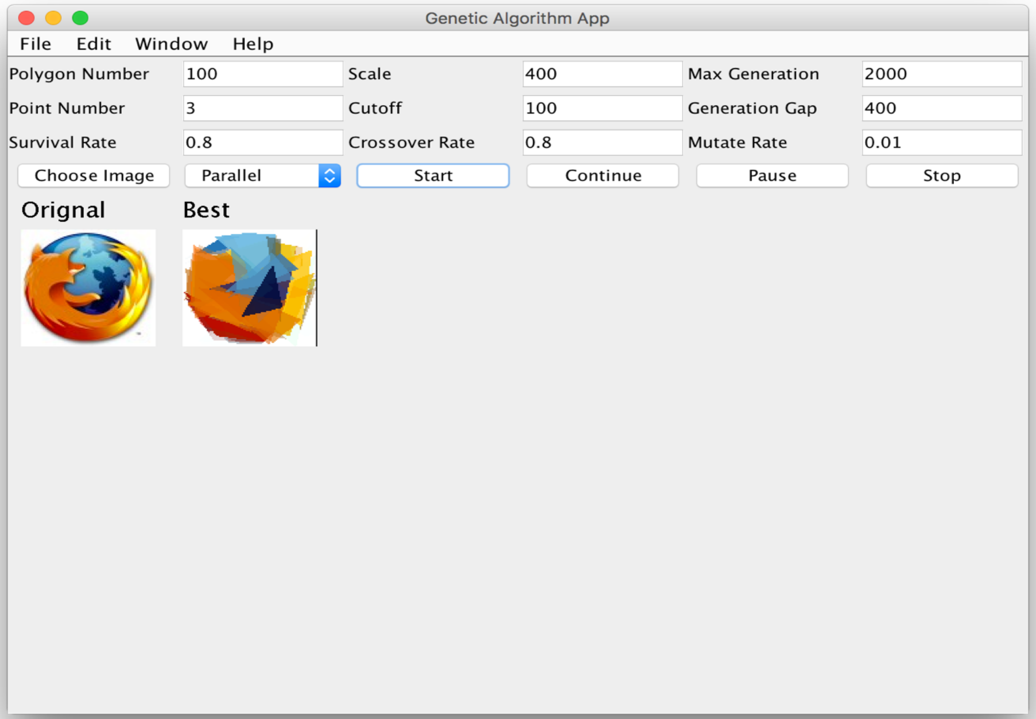

Genetic Algorithm App

Polygon Number	100	Scale	200	Max Generation	2000
Point Number	3	Cutoff	50	Generation Gap	100
Survival Rate	0.8	Crossover Rate	0.8	Mutate Rate	0.1



Final step (use parallel processing)

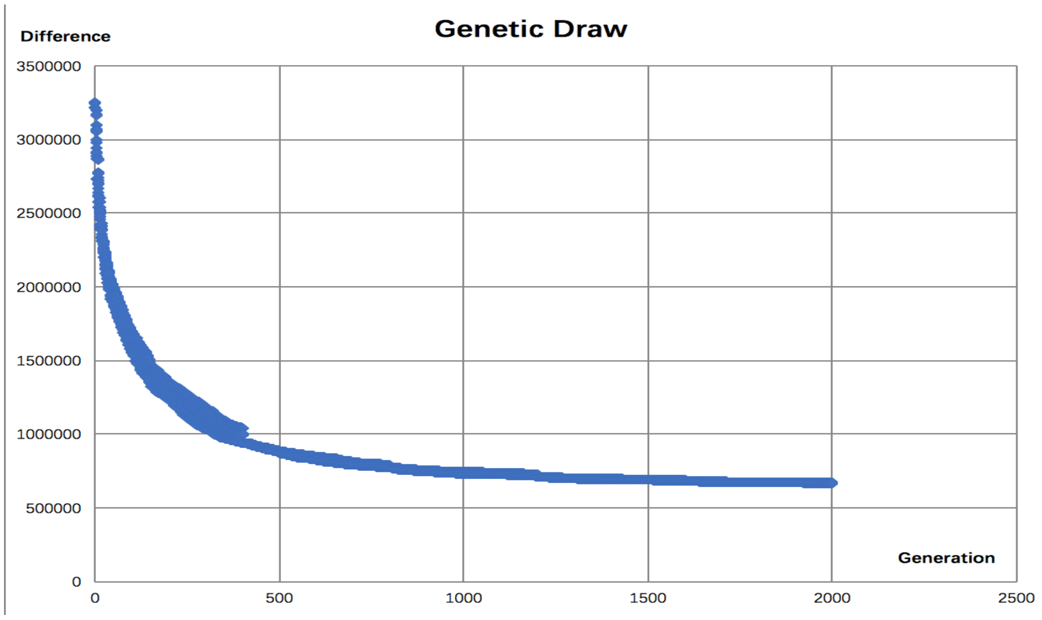
Survive rate	Crossover rate	Mutate rate	time	Difference
0.8	0.8	0.01	60min	665659



Genetic Algorithm App

Polygon Number	100	Scale	400	Max Generation	2000
Point Number	3	Cutoff	100	Generation Gap	400
Survival Rate	0.8	Crossover Rate	0.8	Mutate Rate	0.01

Parallel



Conclusion:

1. The relationship between number of generations and differences appears to be logarithmic.

2. Mutate rate affect the difference apparently, while survival rate and crossover are not.

Apply: