Biological Algorithms and a Real World Application in Image Synthesis

Jason Gray

12046280

University of South Wales

Supervisor: Dr. Colin Morris

April 13, 2016

Abstract

Biological algorithms are a complex set of constantly evolving systems that have many uses within the real world, specifically this project was started to investigate the power of these algorithms in image synthesis and improve upon existing implementations.

The project was split into two halves, the 1st involved an in depth research phase detailing a range of biological algorithms and the 2nd focusing on design and implementation of said algorithm along with a conclusive evaluation of the final product and any relevant processes.

The result is a high quality detached program that efficiently generates an approximation of a source image over many genetic generations. This was achieved in the face of a lack of research available specifically aimed at similar types of image synthesis and hopefully this will aid future research.

Statement of Originality

This is to certify that, except where specific reference is made, the work described within this project is the result of the investigation carried out by myself, and that neither this project, nor any part of it, has been submitted in candidature for any other award other than this being presently studied.

Any material taken from published texts or computerised sources have been fully referenced, and I fully realise the consequences of plagiarising any of these sources.

Student Name (Printed)	JASON GRAY
Student Signature	
Registered Scheme of Study	BSc Computer Science
Date of Signing	11/04/2016

Acknowledgements

I would like to express my appreciation to my main supervisor, Dr. Colin Morris for his support in this project with respect to giving verbal guidance and pushing to keep the project on track.

I would also like to thank Dr. Paul Angel for allowing me to use his OpenGL and OpenCL set-up code and helper functions. This resulted in a faster implementation time in relation to code which allowed me more time to concentrate on a detailed report and less on the initial set-up of the base code.

Furthermore I would like to thank my partner and work colleges for understanding the amount of extra time that was needed to finish this project in a high quality and timely manner.

Contents

List of Figures

1 Introduction

1.1 Aims and Objectives

This report is based around the topic of biological algorithms, the main objective is to come to a greater understanding of biological based algorithms and the uses in the IT industry and beyond and to apply that knowledge to a software development project.

Objectives:

1. Research biological algorithms in detail and develop a documentation that shows the working of each so that it can be used for future development.
2. Design the development in detail.
3. Create and develop a biological algorithm that solves a problem.
4. Research any other information pertaining to each individual biological algorithm.

The objectives are important to have listed, as these are set tasks that must be achieved to reach the final goal. The aims listed below, tie into the objectives but give a less wide view of that the final report and deliverable could achieve. This will help to keep productivity higher by allowing a overview.

Aims:

1. Expand upon the deliverable by making it cross platform.

2. Keep track of time with a initially developed time plan and stick to this plan throughout the project.

3. Document each stage of development, including screen-shots and time tests.

4. Have a design documentation that can be used as a reference in developing the final deliverable.

1.2 Dissertation Outline

This project will involve using the knowledge gained from previous research to create a genetic algorithm to draw a provided sample image using a specific shape. The report will detail research about genetic algorithms, cellular automata, particle swarm optimization and ant colony optimization. These were specifically selected out of the huge range of biological algorithms because of how they could work to solve the problem in the final deliverable.

The problem to be solved is as follows:

> Using at least one of the algorithms reported in this document, demonstrate its potential in problem solving.

The outcome will be to use an genetically algorithm that is designed to draw a set amount of squares each of which has a position, RGB and an alpha what will give a visual approximation of an input image. The specifics of the program will be set out in the design documentation which will give an overall plan on how to tackle the problem along with reasoning on why certain methodology was used.

A evaluation of the final deliverable including the processes taken throughout the project is to be done. This involves the creation of an evaluation plan that is to be used on relevant members of society to gain critical feedback.

2 Plan

The time plan shown in Figure 1 is to give a guide of as to what needs to be done and when, including which order. It is only a basic time plan, only being a month to month basis, but it should still help the project to keep on track nevertheless.

It is important to keep to this plan as not doing so will undoubtedly reduce the quality of the final deliverables and with such a important project, quality is of the up most importance.

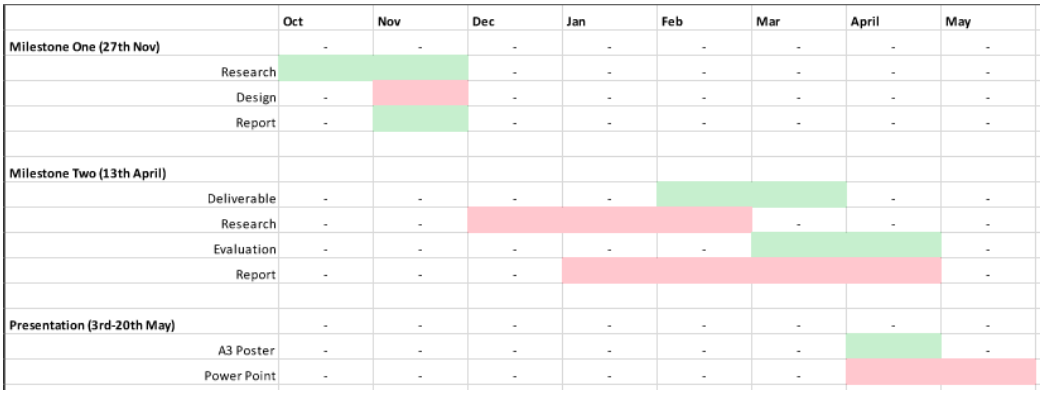


Figure 1: Project Time Plan

Milestone one's objectives will be to do all the initial research required to make the final deliverable along with researching other biological algorithm even though they may not directly link into the final project. This stage will mainly be about the research as described in the project handbook but will

also involve a basic design and overview.

Milestone two will focus on the implementation of the final deliverable, using the knowledge gained from the first milestone to help towards that cause. Design will initially be done and then an evaluation of the created product to prove its worth and give critical feedback.

This will all be congregated into one large project report as described along with a overall software solution.

3 Literature Review

3.1 Genetic Algorithms

The field of Genetic Algorithms was invented by American scientist John H Holland in the 1960s when he detailed a number of theorems relating to Genetic Algorithms before the term was coined. The idea behind his theory was to mimic the aspect of genetics which manipulate chromosomes to combine the information stored in the chromosomes of the parents into one hopefully better which is known as the child.

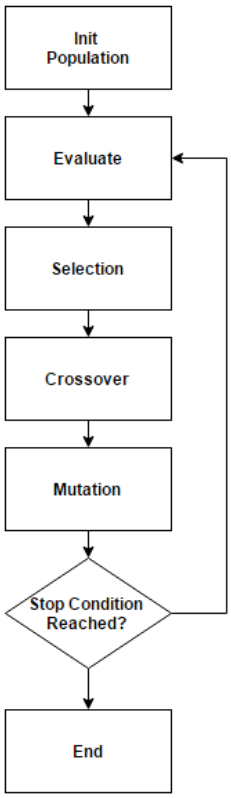


Figure 2: Genetic Algorithm Stages

Since then this concept has been applied to many fields in the industry including artificial creativity, automated design, chemistry, code breaking, economics and many more. Because genetic algorithms are search heuristics, they are well suited for finding a optimal but not perfect solution within a search space, the result of this is a process which is many fold faster then an iterative loop to find the best solution. This is due to the nature of the algorithm itself being one of evolution which is pushed towards the global optima by functions designed to do so.

Below is an overview of how genetic algorithms work, looking from a top-down view as shown by (Davis 1991) and is graphically shown in Figure 2. The structure is simple to understand but there is many methods available for doing each function and some may result in a less than satisfactory result.

Genetic Algorithm Stages

1. Initialize a population of chromosomes.
2. Evaluate each chromosomes in the population.
3. Create new chromosomes by mating current chromosomes; apply mutation and recombination as the parent chromosomes mate.
4. Delete members of the population to make room for the new chromosomes.
5. Evaluate the new chromosomes and insert them into the population.
6. If time is up, stop and return the best chromosomes; if not, go to 3.

3.1.1 Evaluation

Davis (1991) notes that only in the evaluation function is information stored about the problem to be solved. This is interesting because it shows that the rest of the program is dedicated to the evolution heuristics but also shows that the algorithms will obviously need to be tailored to a specific problem or set of problems. That being said, this is not always true as in some scenarios, specific types of functions will need to be used to get an optimal result, this can include tailor picking selection, crossover and mutation functions that suit the problem. There are some methods that have been specifically made to solve one type of problem but in general the standard approaches are used instead of these specific methods.

The evaluation function also known as the fitness function is problem specific. For example, Qing et al (2007), evaluates each of the population members chromosomes bits then converts it into 2 real number based on how

the designer want it to; these real numbers can then be the inputs of the following mathematical functions shown by :

Binary F6 Global maximum of 1.0 at the point of x1 = x2 = 0, search range of -100, 100

$$f(x,y) = 0.5 - \frac{(\sin\sqrt{x^2+y^2})^2 + 0.5}{1.0 + 0.001(x^2+y^2)^2}$$

Rosenbrock Function Global minimum of 0.0 at the point x1 = x2 = 0. Range is -2.048, 2.048

$$F(x,y) = 100(x^2 - y)^2 + (1-x)^2$$

Rastrigin Function Global minimum of 0.0 at the point x1 = x2 = x3 = x4 = x5 = 0. Range is -5.12, 5.12

$$f(x1,x2,x3,x4,x5) = 15.0 + \sum_{i=1}^{5}(x_i^2 - 3.0cos(2\pi x_i))$$

These functions may work for another scenario but as each is different, a tailored evaluation must be used and this method comes from knowing the problem domain and generating the solution from there. In a comparative situation where the goal is to create a close as possible likeness of an original image, sound or design; because the original is already known, the predicted result can be directly compared to the original and this serves as a fitness function, that being a 100% match is a perfect fitness and 0% match being completely wrong.

Sometimes, it is not possible to get a near accurate fitness value and so it must be estimated with functions that give a predictable result. This is in contrast to a direct comparative situation where a 100% accurate fitness result can be achieved.

Once the fitness value is determined, the information can be passed on to the next stage to do with as it needs and that stage is almost always the selection function which determines the population members that will be used to carry their genetic data into the next generation through a crossover function.

3.1.2 Selection

Selection refers to picking a portion of the existing population to mate which will then become the new generation. This selection process is impacted by the fitness of each of the population, meaning that fitter solutions will more likely be picked and carry their chromosomes onto the next generation; this serves in the idea that fit members of the population will generate fitter solutions.

Miller, B (1995) notes that an ideal selection scheme would be simple to code and efficient for both non-parallel and parallel architectures. some selection processes like Tournament selection are designed, and in nature stick to this rule. This would seem a the best way to implement a selection process because of its simplicity yet tried and tested approach but that is not always the case, indeed there will be cases where this method will not be sufficient and another method or a problem specific one may need to be implemented. That being said, a tournament selection method in general is more computationally intensive than other standard methods of selection due to the many stages that are taken place.

There are a number of ways to select specific members to breed, some of the more popular ones are

1. Tournament selection picks a certain number of the population at random, the fittest one is then selected. The tournament size can be

adjusted as needed, increasing or decreasing the size will affect the pressure of selection.

2. Boltzmann Tournament selection is an offshoot of the basic Tournament selection process introduced by David E. Goldberg which varies the selection pressure depending on the time since the start of the solution search.

3. Roulette wheel selection puts each member of the population on a kind of wheel, fitter members have a higher percentage of the wheel and weaker have less so it will always be more likely that a fitter one will be picked but still giving the weaker ones a chance at selection.

4. Elitist selection involves taking only the fittest members and mating that or simply taking them and carrying them over to the next generation without any chances at all, guaranteeing that there will always be the fittest member so far in the current generation.

5. Rank selection gives each population member a rank based on their position within the current generation. This works by giving each member a number rank, for example the worst fitness chromosome could be 1, second worst 2 and so on. This gives the highest fitness members a much higher change of being selected for mating when all of the population are of similar fitness values.

There are two factors that compete in the selection process; pressure and diversity, both of these can be adjusted in sense by changing parameters of the selection function and/or using a different method entirely. Both of these methods tie into avoiding premature convergence on local minimum and to

aid that, Comte et al (1998, pp.265) suggest the use of an elite selection. These factors are described below:

1. Selection pressure is the trend of genetics and of the algorithm to pick only the best candidates that will carry over to the next generation, this will push the algorithm towards the optima. Any change in selection pressure will affect the genetic diversity, too much will limit the diversity, especially if towards the start of the solution search. Having the best solutions picked every time will mean that all avenues are not searched and may get stuck in a local type search space.

2. Genetic diversity is the need to maintain a diverse population so that a suboptimal solution is not reached; if a population is not diverse enough, the final solution may not ever be reached. Many processes in the selection function can affect the diversity of the population like an incorrectly matched selection method or an improperly tuned one.

3.1.3 Crossover

Genes within a member chromosome and generally from the fittest are swapped to attempt to produce the best. This is the only stage where new population members are generated as it follows the basics of biology where 2 individuals of a population mate. Davis (1991) believes that crossover is an extremely important feature of a genetic algorithm. He states that most genetic algorithm practitioners would agree that without the crossover method it would no longer be classed as a genetic algorithm and this has baring on how our children's genes are made up from a mixture of parents DNA.

Crossover has many different methods available, most tailored to a specific type of search space whereas other are tied directly into the method of

chromosome gene encoding. The most popular methods are described below:

Single point crossover A single point is selected and everything from that to the beginning is copied from the first parent. Everything else is copied from the second parent.

Two point crossover Two points are selected. Every binary bit from the first point to the second point is copied from parent two, everything else from parent one.

Uniform crossover Each bit, sequentially is copied, randomly from either parent one or parent two.

Arithmetic crossover A mathematical operation is performed on the two parent to generate the child.

A crossover can essentially be created however is seen fit but generally the common techniques are used because of they are known to work. There are problem specific types of crossover techniques like ones tailored for the Multiple Travelling Salesman Problem as shown by Yuan, S et al (2007) which uses multiple Genetic Algorithms and a two part crossover method and as they state each portion is treated separately and is processed using two independent crossover methods.

3.1.4 Mutation

A required aspect in a GA, it is a function that is used and applied after crossover. This alters a small portion of the chromosome to allow random mutation into the population. This is generally accomplished by bit flipping; one bit, usually one of the lesser bits may at random be flipped from 1 to 0 or 0 to 1. This gives the illusion of randomness and helps to create solutions that

are unique when compared to the population pool members. The probability of mutation should be set low, this is because if it is not the search will tend to become random.

Originally, mutation was the only method of change between generations as crossover was not included in Holland's original design. Later it was added to the concept and was seen to improve performance of the algorithm in terms of computational time and accuracy to find an optimal solution. This is not always the case as papers have been written that show in certain circumstances, a mutation only method brings better results.

Boundary The selected value is replaced randomly with either the lowest possible value or the highest. This tends to be for optimization problems where the solution is at the bounds of the search space.

Non-Uniform With this method, mutation begins at a high percentage but decreases as number of generations increase. This allows the algorithm to have more freedom of the search space towards the start of the program but concentrate on a more specific section towards the end.

Uniform A data value is replaced with a random number.

These are a small selection of methods used for value based mutation.

3.1.5 Existing Examples

A TV commercial scheduling genetic algorithm has been proposed by Ghassemi and Alaei (2013) via a mathematical model, they made use of combinatorial auctions and optimisation and winner determination to achieve an efficient way to determine the best time to show adverts between shows.

Outlined in a paper by Hornby et al (2006) NASA made use of a Genetic Algorithm to design an antenna for three spacecraft in NASAs Space Technology 5 Mission which launched on 22nd of March 2006 and stayed in operation for 3 months. Hornby et al (2006) also suggested that their algorithm tested thousands of new designs and that a human couldn't possibly of tested so many manually in the same time-scale.

The Travelling Salesman is a classic example of a solution that Genetic Algorithms brings to this problem. The problem is that a travelling salesman wants to know which order to visit the cities on his list to get back to his starting city quickest. He knows the distance and must only visit each city once. This problem is a combinatorial optimization problem which Genetic Algorithms are suited for.

3.2 Cellular Automata

John Von Neumann was the scientist who developed the theory of cellular automatons back in the 1950s, he died in 1957 at the age of 53 from cancer. Born in 1903 in Budapest, he moved to the United States in 1931 at which point he started teaching at Princeton University, ironically the same place where he was buried. Von Neumann himself worked on 1D and 2D implementations of the field when first working on it but never went on to fully research the field as he died before it could be completed.

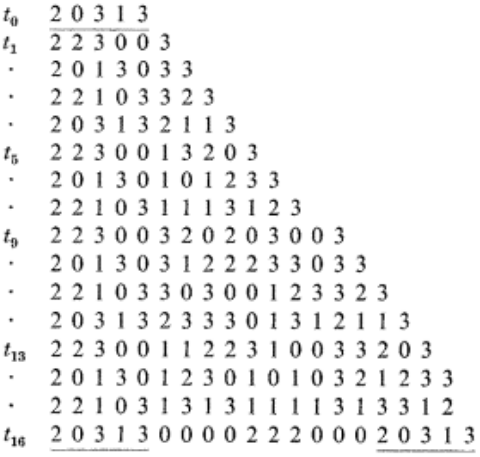


Figure 3: Tessellation Structure

Cellular automatons are rule based systems which are applied to a grid that can be perceived in any dimension, generally the algorithm is implemented in a 1D or 2D environment. Each generation, the global rules from the CA are applied to every cell on the grid. These rules are based on the surrounding cells of each cell and determine if it should be alive or dead (1 or 0).

These simple rules allow the environment to change and flow, giving life to so called gliders which constantly move across the scene. The main concept with this is that the initial configuration will determine the results because the rules are applied every generation, there must be something to apply them to in the first instance.

Amoroso and Cooper (1971) described what they called a tessellation

structure as shown in Figure 3, which after a certain number of steps in a 1D or 2D dimension, shows that automata can duplicate its initial state. Sarkar (2000) delves into this research and states that the rule for this in 1D is the sum of the left neighbour and itself modulo k (k = total states), he also states that the rule is modified in 2D to include the neighbour above the cell.

3.2.1 Computation Universality

The Computational Universal theory is the idea that the history of the universe is computable. First proposed by Konrad Zuse in 1967, he created what is now known at Zuse's Thesis, In 1997 it was extended by Jurgen Schmidhuber to include every universe, not just our own. In a Reddit interview (AMA) recently, Jurgen Schmidhuber (2015) wrote: that the entire universe is being deterministically computed on some sort of discrete computer, possibly a cellular automaton. Cellular automata is said to be capable of universal computation. A basic concept is that a Cellular Automaton can simulate a Turing Machine. As detailed by Sarkar (2000), the steps of the Turing machine can be broken down into the local rule for a Cellular Automata; for this example there are 3 rules of which only one may happen at one time and because it is a step by step simulation, this destroys the standard parallelism of the Cellular Automata system.

3.2.2 Features

Sarkar (2000) states that a Cellular Automata consists of 4 distinct features: Geometry, States, Neighbourhood and the Local rule. Below they are discussed in more detail.

States for every Automaton design can have anything up from two states, the two being 0 or 1. The design of how many states an Automaton can have,

has to do with the global solution that the designer is trying to solve. The automatons that Von Neumann theorised have a total of 29 states including 1 base/start state. Conways Game of Life has the lowest amount of states possible, two. Even with a low amount of states, there is still a huge amount of complexity in the scenes that can be generated.

Geometry can be defined as a d-dimensional grid, possible infinite but commonly finite. In finite grids it is simple to define conditions for the boundary of the grid. The geometry can also be defined as a group graph, Graph grouping is a more abstract way of defining geometry. The mathematical formulae for this which was defined by Harao and Noguchi (1978) is:

$$N = (G, h)$$

$$h(g) = (h_1 \cdot g, \ldots, h_k \cdot g)$$

$$h_i \in G$$

Neighbourhoods in a cellular automata are the cells around each cell that it can interact with. In the von Neumann neighbourhood (diamond), the top, bottom, left and right cells are used whereas in the other popular neighbourhood type, Moore neighbourhood, it uses all cells surrounding a centre cell; in a 2-dimensional grid, 8 surrounding cells are used. The Moore neighbourhood is what Conways game of life uses for its underlying rule decisions.

3.2.3 Existing Applications

There are many kinds of Cellular Automatons ranging from 1D linear types to full 3D reversible function calculating instances. Cellular automata works by trying to take aspects of cell division into account to produce self-replicating

automatons that react to certain rules usually based on surrounding cells or even local or global neighbourhoods.

Conway's Game of Life developed by John Horton Conway in 1970. It has only 2 states, either 0 or 1. Each cell has only 8 neighbours which are used to determine the state of each cell after each step. The rules are as follows:

1. If a live cell has less than two live neighbours, it is dead (0).
2. If a live cell has more than three neighbours, it is dead (0).
3. If a dead cell has three neighbours, is alive (1).

Reversible automata start at a set state for each cell and essentially work backwards, this has been used in the past to calculate equations and many other applications.

SmoothLife is a rule based automaton developed and studied by Rafler, S (2011) in his publication called Generalization of Conways Game of Life to a continuous domain- SmoothLife. The results of his study produced a floating point based automaton where the cells are disks instead of blocks. This creates a smooth, under the microscope view of the entire domain.

3.3 Ant Colony Optimization (ACO)

Proposed by Marco Dorigo in 1992. The algorithm is based on the way ants use pheromones to find their way back to the ant colony, thus the name. As an example, as ants follow a trail, they leave a small pheromone behind which other ants can follow, as it is a pheromone it will over time dissipate. The shorter paths will have a stronger pheromone trail behind it is being followed more often and thus longer paths will have less; this is ideal for path finding as the shortest path usually will be needed.

Many problems like the Travelling Salesman Problem are classed as NP-Hard which means they almost certainly cannot be solved in any speedy fashion with an iterative approach unless there are few solutions due to there being few nodes in the problem. The Ant Colony Optimization algorithm is a solution to many problems but mainly a solution to problems relating to combinatorial optimization like the TSP.

3.3.1 Initialization

The population members are initialized to their starting positions as well as pheromone trails being set. The pheromones on the trail must be set to a value that is slightly more than what will be evaporated after the first few steps. Without this, routes would become dead quickly and the algorithm will most likely fail.

3.3.2 Pheromone Update

Increases pheromone values in good solutions and decreases bad ones. Over the course of many steps, the shorter routes will become a higher pheromone value whereas longer routes will have a lesser or non-existent pheromone

value. Higher value routes are picked in the solution phase as they are considered the shortest.

3.3.3 Solution Phase

Once the conditions are met, the best solution must be picked from the available solutions. This involves scanning through the solution space from the start to the end via each consecutive node that has the highest pheromone levels. This results in an optimal path from A to B.

3.3.4 Existing Examples

Silva, C.A. et al (2007) proposed and simulated a technique to distribute and manage a generic supply chain with suppliers, logistics and distributors. There have been many of this type of study into optimizing supply chains before and there will be in the future, but it shows the application of an ACO to be useful in this kind of problem solution as there are so many different algorithms and options to take to find the optimal solution.

Edge Detection is an interesting use of the algorithm, it can be applied to a 2-d image to find the edge of an item in that image based on the variation of the local image intensity. The image in Figure 4 is shown by Jevtic, A et al (2009) as a before and after image upon applying the algorithm to the said image.

This example is a common one to use and the results are above average, missing things like the chimney stack and a couple of edges. The chimney is missed because of how the algorithms uses per-pixel infor-

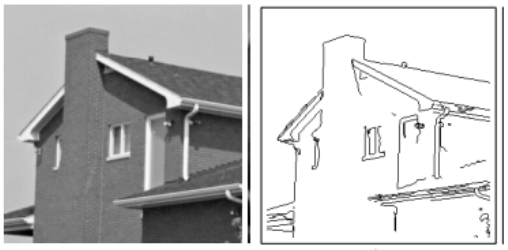


Figure 4: Edge Detection

mation by taking local image intensity to differentiate between edges.
In this image the chimney blends in with the wall because of the lack of shadows and surface reflection making it hard to see even for us humans, especially when in grey scale.

There are many common solutions to data mining and with current advancements in the field, the ant colony algorithm has been used to solve this problem too. Kumar and Rao (2009) proposed ant colony optimization combined with data mining techniques to solve scheduling. Although an ant colony can be used on its own to solve scheduling, a combination was proposed to increase productivity. Interestingly, they include mutation and crossover functions into their article, this allows for a more optimal solution.

3.4 Neural Networks

Neural networks are designed to simulate the way that neurons within the human brain work, having a large collection of interlinked nodes that are trained over time to adapt to specific situations, mainly pattern recognition and prediction which is verified by Crespin (2013, pp.5). McCulloch and Pitts in 1943 created the first neural network from a computational model making use of mathematics and algorithms. Since then neural networks have advanced much especially since the discovery of back-propagation

The standard neural network is a 3 layer type consisting of input, hidden and output nodes which have weighting between them that are modified in a defined way with some function usually back-prorogation. The effect of this is the ability of neural networks to learn from a training data set which adjusts the nodes, and then have real data fed into the system for the neural network to predict the results.

The best implementations of neural networks are on a purely hardware basis. Sarle (1994) suggests that because of the design of neural networks, they are made to be run on parallel computer systems and computer networks, not standard desktop computers. This does not mean that the methods will not work on desktop computers, but that a true solution should have some hardware basis as well.

3.5 Particle Swarm Optimization

```
while currentP < totalP do
    Init particle;
end
while criteria not met do
    foreach particle do
        Calculate fitness;
        particleBest = fitness;
        if particleBest > gBest then
            globalBest = pBest
        end
    end
    foreach particle do
        Calculate and set particle velocity;
    end
end
```

Figure 5: Particle Swarm Pseudo Code

Proposed by Kennedy, Eberhart and Shi in 1995 during a Neural Network conference, it is a population based search algorithm Particle Swarm Optimization (PSO) works exactly as you might think, it has many Particles that act in a Swarm. Each particle is a population member that is gravitated towards its nearest swarm member; the population member closest to the solution will be pulled towards the solution instead of the closest member to it. Basic pseudocode is shown in Figure 5. This shows how simple compared to other algorithms out there a basic PSO can be. This is not a bad aspect

as some situations may require a simpler solution.

A PSO is similar to a Genetic Algorithm in that each particle is initialized with a random starting position as put by Eberhart and Shi (2001). The initial velocity is randomized unlike a GA, which differentiates it, meaning that a population set to the same coordinates each time with the same problem location will undoubtedly not reach the same solution. It was also shown by Eberhart and Shi (2001) when comparing the original PSO to different versions that the optimal approach is to use the constriction factor method which was originally done by Clerk in 1999 along with a constraint on the maximum velocity Vmax to the dynamic range of the variable Xmax on every dimension. Vmax is a crucial part of the velocity calculation. It is a problem specific number that determines the fineness of the solution. If Vmax is too high, members of the swarm may swing past the optimal solution limiting the global optima. If Vmax is too low, members may not get far from the starting position before the search ends.

Each particle keeps track of three sets of variables, velocity, its coordinates within the problem space and its current fitness. Each of these variables will be updated every step based on different parameters. There are 3 core components of this algorithm:

Initialization Place each particle into the solution problem. This function should only be called once.

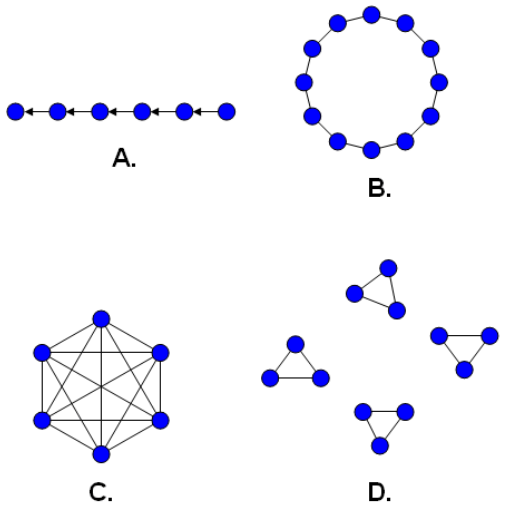


Figure 6: Topologies

Fitness This is calculated for each particle, the best fitness will be the one closest to the global optimal solution. Can be calculated multiple ways but in general it is just the distance from the particle to the solution.

Velocity Based on many factors which include distance, fitness and direction.

There are many optional topologies available for a Particle Swarm, but for this report, have chosen these four to show a varying range:

1. Single-sighted (A)
2. Ring (B)
3. Connected (C)
4. Isolated (D)

These topologies which are shown by Mnemosyne Studio (2012) in Figure 6, allow a particle swarm to act in a certain way; mainly to stop a swarm from getting stuck in local minima. The topologies relate to how each particle act in relation to other particles and this practice affects the final solution.

3.5.1 Potential Applications

PSOs have been shown to be used to evolve or train Neural Networks. Neural networks are essentially a complex function that accepts an input and outputs a set numerical number or numbers. A technique called backpropagation is the standard technique to train the weights in a neural network but a particle swarm optimization algorithm has been shown to have this ability too.

Using a PSO to find the shortest route is a typical example of the algorithm. The swarm is released from its starting positions and each particle pushed towards the solution taking into account its own random velocity and direction. The swarm will spread out in the direction of the solution and whichever member arrives at the end first is considered to be the best route, time wise.

4 Design and Development

4.1 Software

The purpose of this section will evaluate the underlying software used to create the final deliverable, it is imperative to use the correct language when designing and creating possibly computer intensive applications as some languages are faster than others in runtime for example where many are faster at compile time. This can carry forward to functions within a language too, some being poorly optimized compared to function in other languages. Prechelt (1999) compiled the results of 40 programs into graphs that compared the runtime speed and memory usage of Java, C and C++. As shown in Figure 7 taken from that paper, Java was at a huge disadvantage over the C languages in both runtime and memory usage. This is not a true test of the speed of a certain programming language compared to another because each language will have its own functions that are faster than another. It is not an easy test to perform.

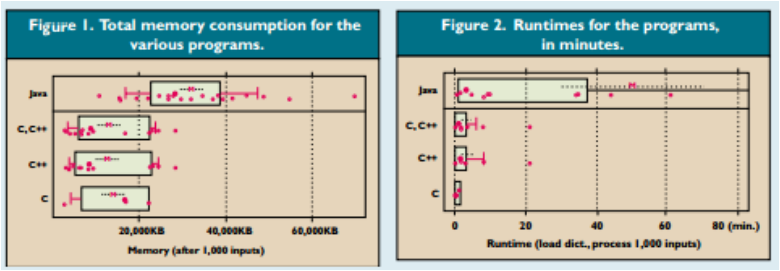


Figure 7: Language Speeds

As the final deliverable will be a mixture of algorithmic code and front end GUI, a language with an optimized front end that can display possibly 3d tick based graphics will be necessary. It will also be more efficient if the language supports the OO paradigm so that design patterns can be

incorporated into the design, enabling a more professional and interchangeable outcome. From there requirements, the following languages have been initially selected because of their popularity in mainstream programming.

1. C, not being an object orientated language would not be ideal as the final solution may require inheritance, possibly relating to the design patterns implemented.

2. C++ contains a mixture of high performance optimized code and is object oriented which will be required. It does not however contain any built in Graphical User Interface which will be required, but this is easily added in with the use of a graphical library.

3. C# although meeting all the initial requirements, tends to be slower because of the layers upon layers of complexity in relation to the graphical element. The graphical rendering will surely be less optimized than a lower down programming solution. The ideal aspect of using C# would be the ease of use via its built in graphical interface and drag and drop features.

4. Python being an interpreted language, does not compile the code beforehand but runs it all in real time, this would tend, although not in all cases to slow down the run time execution of algorithmic program.

5. Java is similar in the technical features to C# but works on a huge range of devices. According to Oracle, the creators of Java, there are 3+ billion devices out there that the software runs on. (Java, 2015)

From the below table, Java or C# would seem to be the ideal software language to use because it ticks all of the criteria needed but in practice they

tend to be slower because of all the extra functionality like garbage collection. Because of this C++ would seem to be the ideal solution along with using OpenGL as a graphical library. C++ is cross-platform, OpenGL is not, and it is platform independent, similar but different library files will be needed for each platform.

	GUI Built-in	Object Oriented	Garbage Collection	.Exe File Type
C	N	N	N	.exe
C++	N	Y	N	.exe
C#	Y	Y	Y	.exe
Python	N	Y	Y	.py
Java	Y	Y	Y	.exe

Figure 8: Programming Language Comparison

4.2 Design Patterns

Design patterns initially, at the start of this report were considered to be used as they tend to be used in professional soft wares because of their tried and tested application in the past. Unfortunately at this stage no design patterns were found that would fit into the final deliverable. Of considerable note though, is the availability and documentation of different design patterns out there each designed for a specific purpose.

The most commonly used pool of design patterns stems from the so called Gang of Four book, titled: Design Patterns: Elements of Reusable Object-Oriented Software. The authors provide twenty three design patterns in detail.

4.3 Development Platform

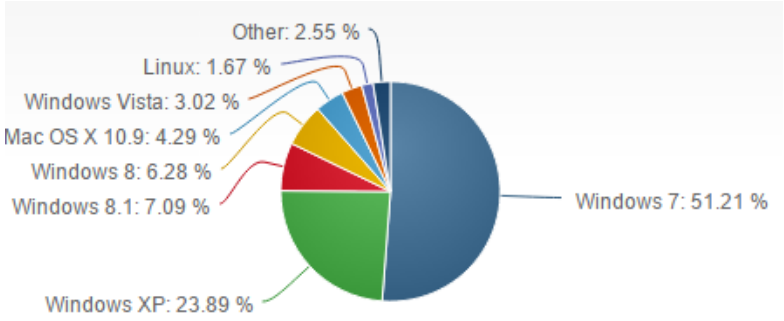


Figure 9: Operating System Usage %

As the final outcome is a technical demo that can if need be, be used by members of the IT community, basing it around the correct operating systems for that specific demographic is extremely important and merits discussion. Windows as shown in Figure 9 (netmarketshare, 2015), has a huge percentage of the market share as shown in 2015 but this has been the case now for multiple decades. Mac OS X, being Apple Inc.s flagship operating system is in second place with a 10 percent share, this has grown over the last few year because of the success of Apple in their smart phone and tablet market. These results may lead some to believe that there would little to no interest for software developed on anything but windows as they have a huge portion of the market share. Because the final deliverable will be aimed at a specific portion of that population, the market share will undoubtedly change as we should be looking at the share for software developers and alike. There was a lack of meaningful data on market share for the target audience while writing this report, but as us developers should know, many professionals out there or even activists do use the other minor operating systems, especially Linux with its focus on command-line interface programs and the main facts that give it precedence over Windows

in this case, its free and open-source.

Relating to the nature of this development with note to the fact that it will be an open source software, the best solution would be to develop it for both windows and Linux. Mac OS even though has higher market share than Linux will be avoided in this context because of the closed nature of the system in relation to only being available on Mac machines, at least officially. Also it would seem that the development community is focused mainly on windows and Linux as its core programming operating systems.

4.4 Design Structure

The project program can be thought of as being separated into four distinct sections. Each of these sections communicate with each other in some fashion and are clearly defined for readability purposes. This is shown in Figure 10 and broken down below:

1. Main Program Loop
2. OpenGL
3. Genetic Algorithm
4. OpenCL Kernels

The main handles the creation of core elements like the OpenGL context and the Genetic Algorithmm. There is hard coded loop within the main as once each other section is instantiated they contain their own loops.

OpenGL being on the same thread as main in essence takes control of that thread. This means that any computer intensive commands that are not related to the rendering through it should not be called within the thread

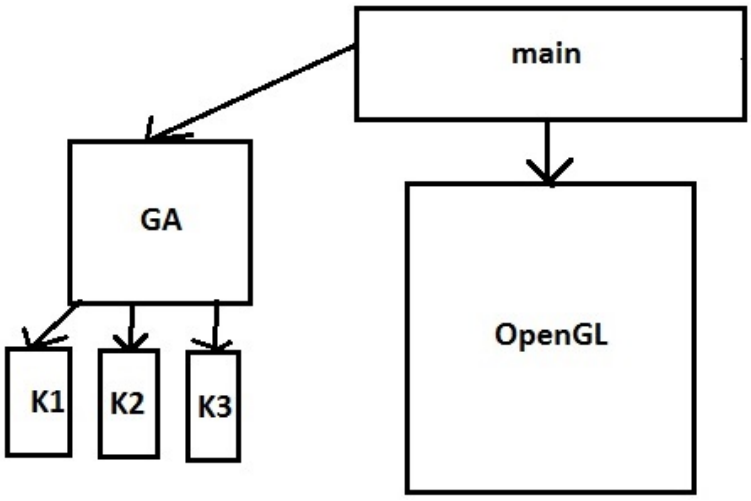


Figure 10: Design Structure

itself. This allows the rendering to be a high performance system that should give top frames per second almost always unless the graphics card is taxed to the point of lock.

The Genetic Algorithm code initially runs within the main thread but on creation launches its own thread and passes control back to main to do what it needs. Because this code is on a separate thread it has the ability to use the entirety of a single core if possible and not interfere with the main loop thread. All of this ties in to the ability of OpenCL to push a thread to the GPU when needed but also to as many threads as necessary.

Moving on, the four sections are also grouped into 2 distinct parts. This allows each group to function without the other allowing for code to be detached and inserted into other programs if necessary. This is a important feature because it means that the genetic algorithm does not decide what is drawn to screen or how, it just does what it needs to and exposes choice variables that allow the program designer to utilize to their wishes. Figure 11

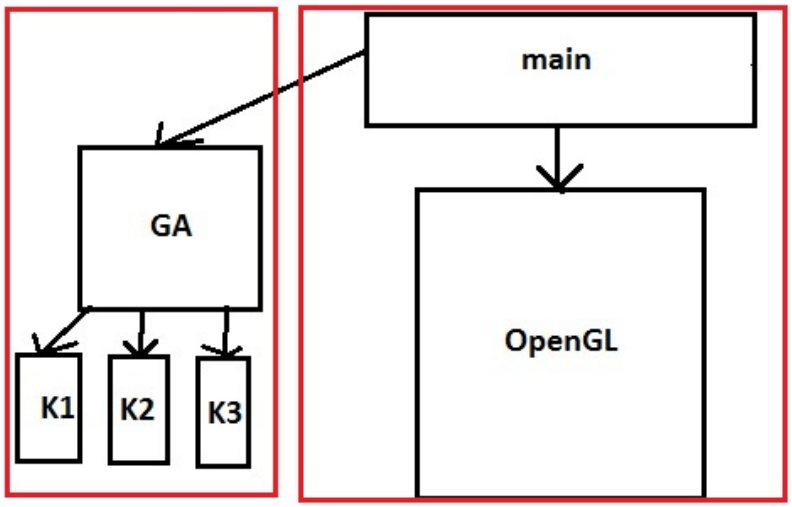


Figure 11: Attached Groups

emphasises this with red boundary boxes and also has the effect of showcasing the two main CPU thread loops.

In essence, two different programs are being created for this project, the actual genetic algorithm which is a cumulation of the previous research in this report and a GUI front end so that the user can see and experience the outcomes of the algorithm. In reality, the front-end could be anything, even just a simple console output.

4.5 Genetic Algorithm

Initial data in the form of an image array will passed through from the main program to the genetic algorithm class as well as any default parameters. This data passed through is an important aspect to cater for as it is the structure that the rest of the program and the algorithm its self will use for the entirely of its use. Inefficient structures or unoptimized data may cause

performance issues along with development difficulty.

Below are the default settings that should be passed into the class on creation:

Image Squared for simplicity and a multiple of the power of 2 as this is the standard when possible.

Population Size Minimum 2 as a population of 1 cannot function correctly using a Genetic Algorithm. If it were possible for a GA to get towards global optima it would undoubted do so slowly.

Selection, Crossover, Mutation Type C++ structure containing all possible varieties.

From the settings shown above, the only real issue that needs to be addressed is the Image which is loaded from file and stored in temporary memory for duration of the program; others are simple settings that are used for aspects of the algorithm its self.

The data represented within an image file will be stored at a low level as an RGB value, this usually ranges from 0 to 255 but some formats store values as a floating point number between 0.0 and 1.0. This is directly convertible to a 0 to 255 range. On a binary level the maximum value of 255 can be stored within a 8 bit (1 byte) data store as shown in Figure 12. Figure 13 shows the available fundamental types in C++ as shown by the official MSDN (Microsoft Developer Network). From the data gained from these tables, it is clear to see that only a single byte will be needed per pixel location and colour, from this either c __int8 or unsigned char should be used.

From these results using either a __int8 or unsigned char will result in minimal memory use and when this comes to loading and pushing data between

128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0

Figure 12: 8 bit Binary Representation (Max 255)

Type	Size
bool, char, unsigned char, signed char, __int8	1 byte
__int16, short, unsigned short, wchar_t, __wchar_t	2 bytes
float, __int32, int, unsigned int, long, unsigned long	4 bytes
double, __int64, long double, long long	8 bytes
__int128	16 bytes

Figure 13: C++ Fundamental Types

GPU and CPU memory, every piece of memory and computation saved will result in a faster result.

4.5.1 Data Encoding

Data for each generation member must be encoded into a format that is usable by the genetic algorithm code as well as taking into account many other aspects, these are summarised about below: http://www.garph.co.uk/ijarie/mar2013/1.pdf

Computational Speed Most methods of encoding data have the side effect of reducing computational speed either by a small margin or more. This is mainly the result of the amount of processor clock cycles that are needed to put this data back into a correct format.

Crossover Compatability The selection of crossover must be one that is compatible with the way genome data is encoded. Some are specifically designed for encoding types like binary or value for example.

Sustainability The data encoded in the genome must be able to be sustained over a unspecified period of time. Data size as well as scalability must be taken into account, the former being more important in most cases.

The data that needs to be encoded is the XYZ coordinate of each shape and its colour information in the form of an RGBA value. This gives a total of 7 char variables for each shape that need to be encoded into the genome.

The other issue that must be addressed is the format which encoding will take form of. There are multiple ways to have the data encoded and these are detailed below:

Binary A string containing a series of binary bits each of which represents a solution in the problem space.

Permutation For ordering problems. Typically a string containing numbers with the maximum number being the total amount of items in the ordering list. For example when trying to find the optimal route between cities in the Travelling Salesman Problem, if there are 20 cities, all number would range from 0 to 19 giving a total of 20 cities.

Value Can be an integer or floating point value but also char or an object. Any values can be stored this way and typically most crossover functions that work for binary encoding will work for this type.

Tree Evolving programs tend to use this method where a linked list of commands comes into play.

As can be seen, Value and Binary encoding are the only two options suitable for the problem of encoding for this project. Permutation encoding is

not suitable because of the type of data that will be stored which consists of XYZ coordinates and colour information all in a range of 0 to 255. Similarly tree encoding does not conform to the problem task as there is no tree structure to be had, only 8 bit values are to be used and no form of tree structure is present within the algorithm.

Binary encoding would consist of 48 bits per shape present within the population member, this is a result of there being 6 pieces of data per shape and each data variable consisting of 8 bits which results in 48 (6 * 8). For the full genome with an example of 50 shapes, this would give 2,400 bits or 300 bytes and as shown in Figure 14, this has the effect of making for one the genome less human readable but this is not a problem for any computer because binary information is what computers are designed to deal with.

x	y	z	r	g	b	a
0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1	1 1 1 1 1 1 1 1	0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1
85	85	85	255	85	85	85

Figure 14: Binary vs Value Encoding

Value encoding as also displayed in Figure 14 but on the bottom section, gives a more human readable result. The main benefit of this approach is that there is no need to convert from binary to decimal for use within the code or OpenCL kernel, this will theoretically give a large performance boost as every processor cycle is important.

For future reference, Algorithm 15 shows the pseudo code of how to convert a 8 bit binary string into decimal.

When looking at the design of the DNA for each population member, how OpenCL accepts the data is an important component to consider. OpenCL can only accept certain C99 standard types, this means the use of strings or vectors are excluded from the data types that can be passed to and from

```
Declare Total;
while i < 8 do
    Increment i;
    if binary bit i == 1 then
        Set total = total + (128 / 2 to the power of i);
    end
end
```

Figure 15: Convert from binary to decimal

the kernel. Because of this and previous design, every data item will have to be its own char variable which is formed into an array because that is an acceptable format. From this information, the results of the encoding process for value and binary encoding are shown below:

Binary Encoding 8 chars per data variable and there are 7 data variables per shape. On a 50 shape example, the results will be 2,800 chars needed per population members.

Value Encoding 1 char per data variable and there are 7 data variables per shape. On a 50 shape example, the results will be 350 chars needed per population members.

From all of this information it can be judged that value encoding is the most suitable way to encode the data for this scenario because of the lower memory profile per population member and removing the need to convert between decimal and binary. Also it meets the requirements shown at the top of this section which are computational speed, crossover compatibility and sustainability. Because of this only value encoding will be used.

4.5.2 Shape Type

The type of shape used by the algorithm to draw the interpreted image is important, not only for visual purposes but also speed of execution within the code.

As the drawing into the image array will be done manually through mathematical calculations, a poor implementation or overly complex shape type will reduce runtime speed significantly. Another aspect that must be noted in the choice of shape is complexity to code, some shapes will by nature be more difficult to design and implement and this must be taken into account as this project is not about designing of shapes but of the genetic algorithm implementation.

1. Square
2. Rectangle
3. Triangle
4. Circle

The shapes above can be grouped into 2 classes, point based (circle) and polygon (square, rectangle and triangle). A point based method involves only position and size whereas a polygon has only position but this is per vertex, allowing for a variety of shapes in affect. The first 3 can simply get implemented as a point based system and this would be the simplest implementation but this would undoubtedly result in sharp edges in many cases.

Taking into account the fact that the image array is a simple WxH grid, having shapes that possess diagonals or anything other than straight vertical

or horizontal lines would be extremely difficult to implement and this is amplified when the points on say a polygon can be at any position within the image space. This is why with graphical libraries, the programmer rarely has the ability to code per pixel information as this has been taken care of by optimised code within the library itself. Due to this, triangles, circles and any per vertex properties will be omitted from possible implementation and thus squares and rectangles are the only possible option for this project.

A square would only need a size property and a rectangle would need a height and width property. This in essence means that a rectangle would have more freedom of movement and allow the algorithm to explore the search space more accurately, as well as making more use of the available processing power. Because of this, rectangle shapes will be used over any other shape type.

4.5.3 Evaluation / Fitness

Fitness in this project is judged by the difference between the source image used and the predicted image.

A simple loop through the data images sequentially, comparing each RGB value as the loop progresses will return meaningful data that will form a basis of the fitness for each population member.

Fitness should be in a range of 0.0 to 1.0 for coding purposes as well as negating the possibility of the fitness value surpassing the maximum value of the variable type.

Furthermore, there are 2 fitness methods that are to be considered for this project as they seem to fit the problem best and are both described below, many more are available to be used but tend to increase in complexity and computation time because of the mathematical functions required.

Sum of Absolute Differences (SAD) Takes the absolute difference between each point and sums them.

$$1 - d/(imageSize * imageSize * 3 * 256)$$

Sum of Squared Differences (SSD) Takes the squared difference between each point and sums them. Squaring the difference has the effect of always giving a positive value but also gives a much larger range because of the extra 256 multiplication.

$$1 - d/(imageSize * imageSize * 3 * 256 * 256)$$

Both of these methods will give a fitness for each result as a range of 0.0 to 1.0 but SSD will give much more detail that in this project may not be needed. A change on such a small magnitude is not something that seems to be required as undoubtedly the best crossovers will give a much greater fitness jump and that is what is needed.

Overall, the SAD method is the that should be used by default because of the aspects mentioned previously.

4.5.4 Selection

As shown in previous research there are multiple ways of doing selection to pick the next parents that will create the next generation. Although the option will be given to the user to pick the selection method used, a survey or the methods in relation to speed and suitability to the scenario must be completed to give a satisfactory design.

Selection each generation must if possible pick the best candidates. A simple example of this would be to have a population size of 2, the first

being the best solution so far and the 2nd being a random DNA strand. This would give the next generation always the best candidate so far but also a potential best solution. The result would be a constantly improving algorithm where each generation is guaranteed to be the same as the last or better but will result in much slower results as the design is of random nature, not using crossover to feed the future generations. An example of this that has a similar outcome as this project was found to of been achieved through research where a random DNA strand is rendered, if that image is better than the last then call this the best.

Roulette Wheel Each member has a chance to be picked as a parent of the next generation equal to their fitness divided by the total fitness of the generation.

Tournament A number of the population are picked, the highest fitness member is chosen as a parent for the new generation.

Elitism A portion of the best population members are carried on to the next generation without any modification.

Rank Each population member is sorted and given a value equal to that position within the sorted array. The highest fitness member will have a rank value equal to the population count. This gives the best fitness members a much higher chance of being selected.

A partial elitist approach would be a step in the right direction as having fitness revert back, which may happen in this type of project would be a waste of time and may reduce the final result. Taking a select portion of the population, the best 2 would allow the algorithm to constantly improve upon itself and that combined with one of the suitable selection methods shown

above should in theory take us towards the global optima within a sensible time span.

Looking at existing examples that are available, the elitist method is one that is used throughout them all. One example uses a population size of 2 and the first is the best solution so far. This seems a bit excessive as it would be more along the lines of a random search but saying that, the aim of this project is not to copy existing creations but to improve upon the concept where possible and even though other do use the elitist approach does not mean that it is the best way.

When comparing roulette wheel selection vs tournament selection, one of the main aspects between them to note is the speed to get a total selection pool. Roulette wheel is a much faster solution because of the minimal amount of loops needed but compare that to tournament, there are many more iterations that are needed to be run and again referring back to the need of a fast program process, this would not be an ideal selection method. From this, a ranked roulette wheel will be used with a partial elitism method but tournament selection will still be available to the program user if they wish to allow for more freedom of choice and diversity.

4.5.5 Crossover

Crossover can be achieved via many different and wide ranging methods. This arguably is the most important aspect for the GA within this project as an incorrect crossover selection will result in unsatisfactory results. Research has shown that there are a wide range of available methods out there and each has their own advantages and disadvantages.

As the type of data encoding will be value based as shown in Figure 14, there are only 7 values per shape that are needed to crossover at each gen-

eration compared to the 56 that would of been utilized for binary encoding. This means that the computational speed of the crossover will be much faster as well as a much simpler job to do so.

From the needs of the algorithm, two methods of crossover will be available with the first being the default. These are:

One Point All genes at a selected random point are taken from the 1st parent and anything after that point from the second parent.

Uniform Genes at random are taken from either parent.

Both methods will be available to the user but what must be decided is whether to take each shape as a whole, that meaning when a selection is make to take the genes from parent A, all of the data from that specific shape will be carried over, or allow the crossover method to select deeper into each shape and allowing access to each point in the shape.

The concept of this is shown in Figure 16 and 17.

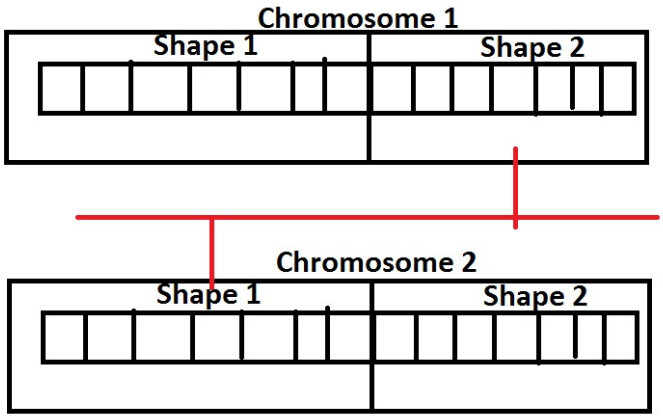


Figure 16: Crossover Per Shape

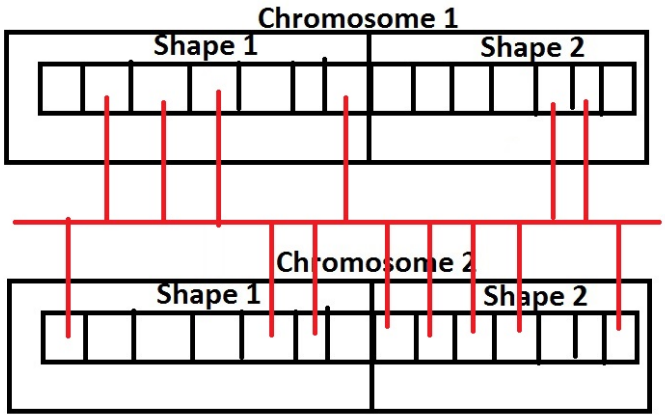


Figure 17: Crossover Per Gene

The examples of crossover selection types are based upon the uniform method but the same concept applies to all types.

Using a per shape type has the effect of not changing the data of a shape after crossover (not including mutation) which would produce a crossover that can take the best shapes available and copy them into the new generation. The possible downside of this is that without mutation, only the current shapes in the total population are the ones that are possible to use as a shape will never change per generation.

Using a per gene type will give the crossover the ability to change the properties of each shape but this may have the consequence of reducing the fitness of each shape individually. Obviously this varies with crossover methods for example one point crossover could pick a point in the middle of a shape, so all shapes before and after the picked one would be the same but every property left of the selection point in the picked shape would come from parent A where as the right would come from parent B. Because of this, only 1 shape per chromosome will be changed, compare this to uniform

crossover, every shape's properties will more than likely be different due to the random selection process between the 2 parents.

From this discussion, it has been decided that uniform crossover will be the default method along with a per gene crossover type. This will give the algorithm a much improved ability to navigate through the search space due to its increased possibilities and more diverse population.

4.5.6 Mutation

Mutation being an assist towards the problem of the algorithm getting stuck in local optima does not have a massive selection of methods that would apply to the current project. A simple bit flip operation should suffice but because the data encoding is value based, this is not feasible.

Instead, a few methods are summarised below that are tailored for value based DNA encoding.

Boundary The selected value is replaced randomly with either the lowest possible value or the highest. This tends to be for optimization problems where the solution is at the bounds of the search space.

Uniform A data value is replaced with a random number.

Non-Uniform With this method, mutation begins at a high percentage but decreases as number of generations increase. This allows the algorithm to have more freedom of the search space towards the start of the program but concentrate on a more specific section towards the end.

Both uniform methods seem to be suited for the problem whereas boundary would lead to results focused on the edge of the search space which is not wanted for this scenario. It is suggested that a standard uniform

method is primarily used but also implement non-uniform. This is because the user may want the program to go on for much longer than intended but a non-uniform approach sets the mutation rate to 0 when a design specified generation number has been hit. This is how the non-uniform is able to scale the mutation.

4.6 Graphical Design

The genetic algorithm will be visually shown via an OpenGL interface GUI. Although the concept of the program is to be 2D only because of the need to input one image and output another, a 3D implementation can still be had via a 2D plane rendered in the 3D environment. This may make the program unnecessarily complex but, this gives the opportunity to add a static HUD along with extra quads for increased information. Figure 18 shows a design where just a single quad is rendered in the 3D scene for which the best fitness genome so far can be rendered after each genetic algorithm population loop.

Available Settings:

1. Population Size
2. Mutation Rate
3. Number of Shapes
4. Selection Type
5. Crossover Type
6. Mutation Type

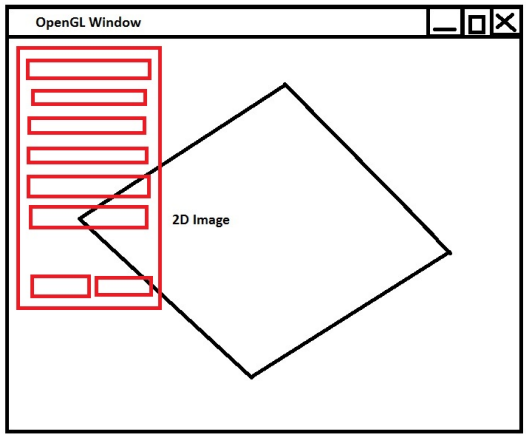


Figure 19: Basic Concept + GUI Design

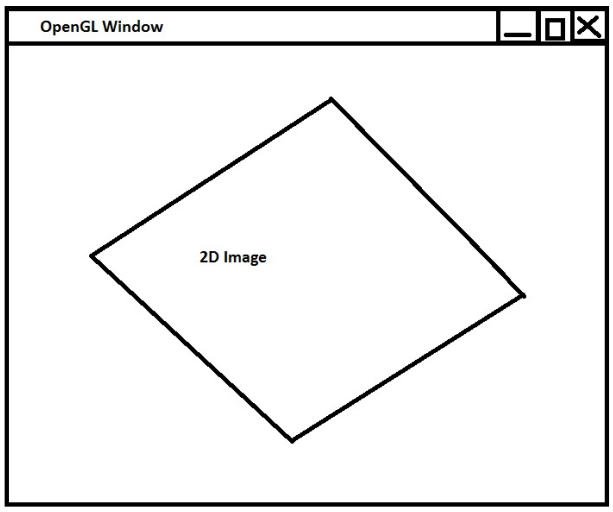


Figure 18: Basic Concept

Building upon this, there will be a selection of static GUI options that are rendered to screen allowing the user to control the starting settings for the GA. This will be a mixture of text strings and buttons that tie in intuitively to give a excellent user experience. Figure 19 & 20 show the desired design of the settings making use of plus and minus buttons as well as drop down menus for feature selections. All of this gives the user a fair amount of control over the program and also allows the user to start/restart and pause when necessary.

It is important to note that

because of how the Genetic Algorithm program works, changing these settings during algorithm runtime will not be possible as a change of settings will create difficulty with the state structure of the code and would make it much for difficult to program as it would have to dynamically change most aspects of the design structure like genome layout for example.

The windows header can be made use of for displaying information. This section should be concise as there is not a huge amount of character space available and because of that only FPS, Current Generations and Best Fitness will be shown along with the program title.

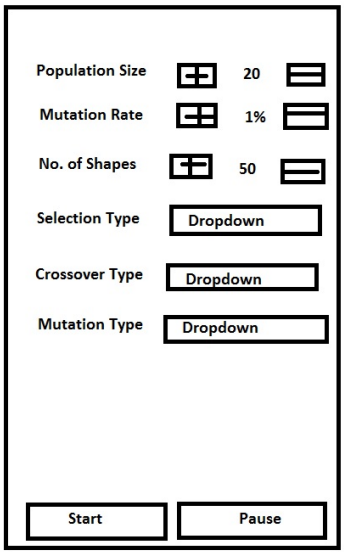


Figure 20: GUI Design

4.7 Multi Threading

There are two threading solutions that are to be used in this project. CPU threads which are the basis of the program code and GPU threads which are there to just speed up CPU computation time by pushing the time consuming loops

and calculations to the GPU which has many more cores and threading abilities.

Designing the program using just the default thread would create a sequential program and seeing as most computers in today's age tend to have 4 cores as default, this would be a waste of computational time and in a program that could possibly get to hundreds of thousands of genetic algorithm generations, time is an important factor to consider.

Another aspect that must be talked about is the aim of developing the program to work on both Linux and windows, this means that different threading methodologies must be used depending on the platform the program is running on. Each have their own threading system and as such must be designed to use each specific thread function when necessary.

4.8 OpenCL

As we are not using the graphic drawing power and multi threading capabilities of the standard OpenGL implementation, something must be done to speed up runtime. This is where OpenCL comes into play, allowing small pieces of code called kernels to be pushed into the GPU, or any compatible device in fact with the result of speed increases of many fold.

There will be two used kernels, one for drawing coloured shapes into an image array on the fly and another to calculate the fitness of each population DNA based upon that image. This creates two distinct kernels but the output from the first instance will feed into the input of the second, creating a kind of chain of events.

Figure 21 and 22 show pseudo code that links into the previous statement talking about a chain of events, one kernel in a sense feeding into another. Because of this design and the way OpenCL is, both kernel queues can be

pushed into the platform, giving our GA loop time to do extra computation tasks as OpenCL will notify the program when it has been completed. This is better than a true block wait.

Using this loop we can calculate the amount of loops that will executed on average by substituting parameters into the loop and calculating the total. From that with a population size of 50 and a crossover percentage of 30%, this gives the total selected members as 15. Each member has 50 shapes and each shape has 7 data points. This gives the total number of loop per generation as shown below with an estimated variance of roughly 10% because of the crossover function.

$$15 * 50 * 7 = 5250$$

From this information, it can be deduced that the program will not benefit from using an OpenCL implementation for this section of code as the time to write and read the data between the CPU and GPU would more than likely take more time that a sequential implementation. This is simply because there is overhead in the read and write functions but this is minimized when the total number of threads needed is such a high number.

Input: X, Y, Z, R, G, B, A
Output: image array
Result: Draws a shape into the image array based upon the inputs
for *each pixel* **do**
  if *pixel within new shape* **then**
    Set pixel to RGBA;
  end
end

Figure 21: Draw Kernel

Input: pixel, source image, generated image, type

Output: fitness

Result: Returns the fitness for the given image

```
if type is 0 then
    Set fitness to fitness + absolute diff;
end
else if type is 1 then
    Set fitness to fitness + squared diff;
end
```

Figure 22: Fitness Kernel

Another important process to look into is if the two kernels mentioned above are the only ones needed; will the program benefit from taking a section of code and converting it to a kernel instance that will be run on a OpenCl device. Looking at Figures 23, 24 and 25, the first and the last seem to be already optimised as much as possible, but in Figure 24 in the crossover and mutation code towards the bottom, there is a substantial loop that will be run through once every generation.

4.9 Pseudo Code

This section is to show an initial design specification via pseudo code. It is important to have a structure to work towards in a project, albeit a simple one. Having a design beforehand can help eliminate and potential problems later down the road.

The pseudo code in Figure 23 shows a overview of the evaluation function without going into detail for OpenCL kernel code. As shown there is a sizeable amount of processing to be done every generation and could simply not

be done in any reasonable amount of time using a single CPU thread/core. Figure 24 shows the step taken after evaluation which in order are; selection, crossover and mutation. As these functions are generally not tailored to the problem scenario, they are not expanded on in the pseudo code because it can be one of many different functions.

Figure 25 the simple main loop within the genetic algorithm that end when a specific termination criteria is met.

```
while i < population member do
    while j < shape count do
        Draw shape into image array i;
        Increment j;
    end
    while j < screen size * screen size do
        Set Fitness i to Fitness i + pixel difference;
        Increment j;
    end
    Increment i;
end
while i < populationSize do
    if fitness i > bestFitness i then
        Set bestFitness;
    end
    Increment i;
end
if best image changed then
    Read best image from kernel;
end
```

Figure 23: Evaluation Pseudo Code

```
while selection criteria not met do
    Select member from population;
    Insert member into selection array;
    Increment i;
end
for selected member do
    Crossover population a with b;
    Add new member;
    for shape in selected member do
        for data in shape do
            Set r to random number;
            if r > mutation chance then
                Mutate data;
            end
        end
    end
end
```

Figure 24: Selection, Crossover and Mutation Pseudo Code

```
while termination criteria not met do
    Call Evaluation Function;
    Call Selection Function;
    Call Crossover Function;
end
```

Figure 25: GA Loop Pseudo Code

5 Prototype

5.1 Prototype 1

From the design documentation, an initial prototype was developed but with certain aspects changed to test the validity of the design procedure.

Figure 26 shows the best chromosome strand from the program after 482 generations and even a couple of thousand seem to have similar results where no discernible image is seen within the quad. Below are the changes that were implemented differently from the design.

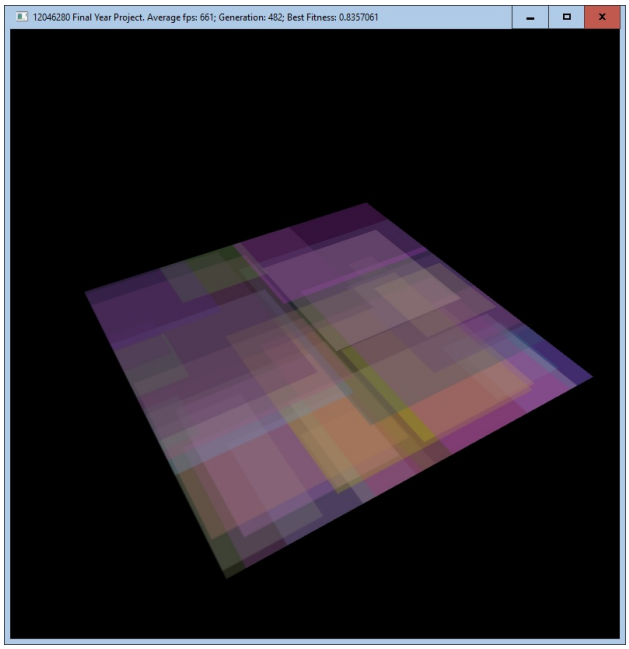


Figure 26: Prototype 1

1. Constant alpha of 30%.
2. Binary based encoding instead of value based.
3. Only 2 population members are selected each generation.

The result is a far from a satisfactory outcome with a best fitness of only 0.83 . This seems to be a result of the constant alpha but also the selection process mainly. The binary based encoding should not affect the outcome

only the speed of the program itself. Having a constant alpha of 30% with a total shape count of 50 means that most shapes are hidden below the recently rendered shapes and giving the algorithm less of a chance to navigate through the search space effectively.

Based on this, for the next prototype alpha will be implemented into the solution so that the difference can be observed effectively.

5.2 Prototype 2

Alpha was implemented along with changing the genome encoding to value based. Unexpectedly, the results were very similar in the best fitness found within 500 generations.

The extra data within each shape does give the program more flexibility to find an optimal solution but because there is little difference between this prototype and the previous other than a speed boost, it is assumed that there is either a bug within the code or the design is flawed. This could be because of genome encoding which may not let the algorithm search the space efficiently or the use of incorrect functions like selection, crossover and mutation.

Because of this, the development of prototypes will be stopped for which more research and time will be put into design to counteract these issues.

6 Evaluation

Evaluation is an important part of any programming project, this gives the means to find issues within a program, report or code. Two aspects of this project will need to be evaluated to obtain an effective outcome. The product, the final deliverable can be evaluated with a mixture of test outputs, human interaction and speed tests. The final program will be multi-threaded, the intention of this is to increase runtime speed but this may not help to the desired factor; this should be tested at each stage and outputted for evaluation. Any processes relating to this project can be evaluated. This relates to the process of creating the final software but as well as evaluating the methods used to select the software used to create it. Also the processes used in evaluation should themselves be evaluated to test they are performing up to par.

6.1 Methodology

According to Gediga et al(2002, pp.3) evaluation goals can be characterised by at least one of the following questions:

1. "Which one is better?"
2. "How good is it?"
3. "Why is it bad?"

Each of these questions serves to evaluate the software and to help improve the system though a multitude of evaluation stages.

The first two questions are summative evaluation methods and the third is formative evaluation. To expand, summative evaluation concerns its self

with the result of the software development and design, giving data that helps make a decision on the project as a whole. Formative evaluation, deals with information that is targeted at the improvement of the system or design and this is classed as the main stage of evaluation.

The questionnaire that is to be designed for evaluation would seem to fall within the formative evaluation concept as the data returned will be analysed to allow for improvements where necessary but also some questions will be designed to give feedback on the selection processes of some elements of the system and this would be considered summative. Thus the above questions, 2 and 3 would be covered by the questionnaire.

In an overall view of things, question 1 above should be directed not at other similar programs that are available out there but at the chosen genetic operators. Speed is one of the main focuses of this project and some of the available operators are bound to be slower or faster and this can be captured and analysed for future developments.

6.2 Target Audience

The evaluation of the final deliverable must be evaluated by a select number of individuals from its target audience. Having someone who is outside that audience will be a failure in evaluation because they may not necessarily know the area in relation to personal expertise and the returned evaluation results may not be accurate.

From all of the documentation and the deliverable itself, it can be shown that the target audience are those interested in genetic or biological algorithms but also those who would use the detached code that was developed in C++. Games developers, computer scientists and computer engineers would benefit from this package as it is something that either they could use

directly or research to their own needs. Only a small group of the selected target audience will be use for evaluation

The next question is to determine the method(s) of evaluation that will be used upon those target audiences.

A simple one or two page questionnaire would seem to be the best method for this scenario as it can capture the individual's thoughts on aspects of the program like design and usefulness if presented in a suitable fashion. This would give good feedback that can be used to improve on design of the GUI and the software itself. This tied into another form of evaluation would solidify the results and give an overall good feedback pool of data.

Before that an evaluation of the state of the program's speed compared to other example available must be completed, this is to ensure that the quality of the program is up to par with existing examples and hopefully beyond. This will take the form of a runtime test based on criteria that is fair to both parties.

6.3 Visual Quality

We will now conduct an evaluation of the images that are generated which will involve a comparison between it and the original as well as an analysis of different shape amounts.

Figure 27 and 28 shows the best image generated so far by that generation. As can be seen, the quality is nowhere near perfect but it does give an overall good outcome which will undoubtedly improve over many more generations. As the generations increase, the image slowly turns into something that can be seen to resemble the original image. There will be a limit to the quality that can be generated and this is constrained by the amount of shapes that are available to each population members. If every pixel has a unique colour

then the only way to get a 100% match would be to have as many shapes as there are colours and this can be seen from the figures where a type of blur effect is seen because there is not enough diversity in the shapes due to the lack of said shapes.

200 Gen | 1000 Gen | 10000 Gen | 50000 Gen | Original

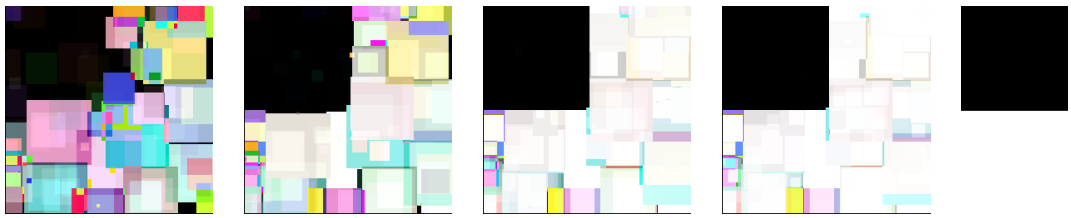
Figure 27: Star Trek's Data

200 Gen | 1000 Gen | 10000 Gen | 50000 Gen | Original

Figure 28: Tom Waits

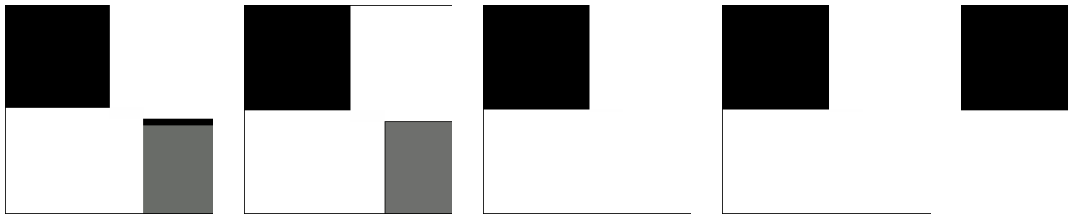
Comparing Figure 29 to Figure 30, it can be seen that because of the simplicity of the image, the optimal shape amount is 3 since 3 white shapes can be placed to form the original image assuming the background is black. Because of this the optimal shape example shown in Figure 30 reaches a perfect solution and in a small amount of time, compare this to Figure 29, the results are far from perfect within the same amount of generations.

The main problem with increasing the total shapes is the extra computational time needed for each generation. The result is a lesser average fitness



200 Gen | 1000 Gen | 10000 Gen | 50000 Gen | Original

Figure 29: Test Image (100 squares per chromosome)



200 Gen | 1000 Gen | 10000 Gen | 50000 Gen | Original

Figure 30: Test Image (3 squares per chromosome)

increase per generation but a much higher possible maximum fitness compared with a lower shape count. Due to this effect, it is best to tailor the settings of the algorithm to the needs of the image and this is mainly done through experimentation and a knowledge of the problem.

Taking the results from Figure 28 and comparing to the image in Figure 31 which is from an existing on-line system similar to this one, the fitness is much higher and gets to that level much quicker. This is mainly due to the use of a different shape as was used in this project, giving the algorithm a much higher fidelity in image quality. The reason for using squares over any other shape have been explained previously but it is interesting to note an actual comparison now that the implementation and design stage have been finalised.

Figure 31: chriscummins.cc (3000 generations)

6.4 Runtime Tests

Tests will now be run to compare the speed of different selection, crossover and mutation methods to see if this ties into previous statements about them. Not only that but the speeds can be noted so that a fast implementation can be used if needed where generation speed is of the utmost importance.

This is a borderline type of evaluation in that it will test the estimated speeds for methods noted throughout this report with the actual speeds shown. It can serve as a small way to test the outcome as well.

Each test result will be an average of the results of the test run 5 times. This gives a better result and minimises error with system times and processes. 500 generations will be the cut-off point and all results will be recorded in milliseconds using the C++ clock() function as the chrono library is not available in the used version of the standard.

The default settings for the test are as follows:

1. Roulette Wheel Selection
2. One Point Crossover
3. Uniform Mutation

Changing the method to the one needed for each individual test will give the different time results while allows a initial testing time to be recorded, for comparison purposes. Mutation will not be tested as there is only 1 method implemented and this speed will be recorded within the initial test speed.

6.4.1 Selection

Type	Test 1	Test 2	Test 3	Test 4	Test 5	Average	100,000 Gen Time
Roulette	16051	16542	16254	16245	17010	16420.4	54.7 Mins
Tournament	17251	17111	18012	17564	17321	17451.8	58.1 Mins

As can be clearly seen from the results, roulette wheel selection is faster than tournament selection as had been previously stated, this is a simple fact of the amount of loop iterations that are executed in comparison to one another.

Although the difference between the results is only roughly a second, over a minute this adds up to 4 seconds which is a speed decrease of about 4%. Looking at the estimated time to reach 100,000 generations, there is a difference of 4 minutes and that difference would be 40 minutes to reach 1 million. A substantial time difference.

6.4.2 Crossover

Type	Test 1	Test 2	Test 3	Test 4	Test 5	Average	100,000 Gen Time
One Point	16051	16542	16254	16245	17010	16420.4	54.7 Mins
Uniform	16811	16211	16872	16921	16215	16606	55.3 Mins
Non-Uniform	16512	16315	16456	16855	16321	16491.8	54.9 Mins

Not much difference is shown between the different crossover methods. The shown difference may even be due to the system at the time.

6.5 Questionnaire Design

Questionnaire on 12046280 Final Year Project

This questionnaire is to help evaluate the software that was developed for the Computer Science final year project.

The underlying software is a genetic algorithm that takes an input image and tries to generate a copy using a specified number of squares at different positions and different colours. This is done through a crossover procedure which mates two chromosomes together to create a new hopefully better one.

If you agree to take part in this questionnaire, your answers and details will be used within a final report, no contact details will be recorded for private purposes. Please answer as honestly as possible so that the feedback can be used in a constructive way.

Thank you for your participation.

Full Name: .....
Current Job / Relevant Study: .....

Does the program load and run without error on your computer system?
— yes —— no —

How would you rate the look of the GUI (Graphical User Interface)
terrible 1 2 3 4 5 6 7 8 9 10 very good

Would you be inclined to use the underlying software in the future for free?
— yes —— no —
If possible please elaborate:

Do you feel the speed of the program could be improved? If possible explain

What kind of shape would you have used for the genetic algorithm drawing? —
square —— rectangle —— circle —— triangle —

What operating systems do you use on a regular basis?
— MacOS —— Windows —— Linux —

If you have any final comments, please do so below:

Thank you.

6.6 Results

Questionnaire on 12046280 Final Year Project

This questionnaire is to help evaluate the software that was developed for the Computer Science final year project.

The underlying software is a genetic algorithm that takes an input image and tries to generate a copy using a specified number of squares at different positions and different colours. This is done through a crossover procedure which mates two chromosomes together to create a new hopefully better one.

If you agree to take part in this questionnaire, your answers and details will be used within a final report, no contact details will be recorded for private purposes. Please answer as honestly as possible so that the feedback can be used in a constructive way.

Thank you for your participation.

Full Name:	Andrew Howson
Current Job / Relevant Study:	Computer Science

Does the program load and run without error on your computer system?

— **yes** —— no —

How would you rate the look of the GUI (Graphical User Interface)

terrible 1 2 3 4 5 6 **7** 8 9 10 very good

Would you be inclined to use the underlying software in the future for free?

— **yes** —— no —

If possible please elaborate:

It may be useful in the future within the industry, having a pre-built multiple operating system software could save much time.

Do you feel the speed of the program could be improved? If possible explain

It does seems slow to generate a good image.

What kind of shape would you have used for the genetic algorithm drawing? —

square —— rectangle —— **circle** —— triangle —

What operating systems do you use on a regular basis?

— MacOS —— **Windows** —— Linux —

If you have any final comments, please do so below:

Thank you.

Questionnaire on 12046280 Final Year Project

This questionnaire is to help evaluate the software that was developed for the Computer Science final year project.

The underlying software is a genetic algorithm that takes an input image and tries to generate a copy using a specified number of squares at different positions and different colours. This is done through a crossover procedure which mates two chromosomes together to create a new hopefully better one.

If you agree to take part in this questionnaire, your answers and details will be used within a final report, no contact details will be recorded for private purposes. Please answer as honestly as possible so that the feedback can be used in a constructive way.

Thank you for your participation.

Full Name:	Steven Stafford
Current Job / Relevant Study:	Graduate Web Developer

Does the program load and run without error on your computer system?

— **yes** —— no —

How would you rate the look of the GUI (Graphical User Interface)

terrible 1 2 3 **4** 5 6 7 8 9 10 very good

Would you be inclined to use the underlying software in the future for free?

— **yes** —— no —

If possible please elaborate:

Having ready build packages is a must within the web development industry.

Do you feel the speed of the program could be improved? If possible explain

What kind of shape would you have used for the genetic algorithm drawing? —

square —— rectangle —— circle —— **triangle)** —

What operating systems do you use on a regular basis?

— **MacOS** —— Windows —— Linux —

If you have any final comments, please do so below:

I use MacOS mainly in my workplace so this program would not be appropriate for me. I used my home computer to test this project. Thanks.

Thank you.

Questionnaire on 12046280 Final Year Project

This questionnaire is to help evaluate the software that was developed for the Computer Science final year project.

The underlying software is a genetic algorithm that takes an input image and tries to generate a copy using a specified number of squares at different positions and different colours. This is done through a crossover procedure which mates two chromosomes together to create a new hopefully better one.

If you agree to take part in this questionnaire, your answers and details will be used within a final report, no contact details will be recorded for private purposes. Please answer as honestly as possible so that the feedback can be used in a constructive way.

Thank you for your participation.

Full Name:	Thomas David
Current Job / Relevant Study:	Computer Science

Does the program load and run without error on your computer system?

— yes —— **no** —

How would you rate the look of the GUI (Graphical User Interface)

terrible 1 2 3 4 5 6 7 8 9 10 very good

Would you be inclined to use the underlying software in the future for free?

— yes —— no —

If possible please elaborate:

Do you feel the speed of the program could be improved? If possible explain

What kind of shape would you have used for the genetic algorithm drawing? —

square —— rectangle —— circle —— triangle —

What operating systems do you use on a regular basis?

— MacOS —— Windows —— Linux —

If you have any final comments, please do so below:

The program wouldn't load on my pc. ADDED by Jason: After enquiring, the user does not have a graphics card installed within his test pc and thus was using a non-opencl integrated graphics chip.

Thank you.

Questionnaire on 12046280 Final Year Project

This questionnaire is to help evaluate the software that was developed for the Computer Science final year project.

The underlying software is a genetic algorithm that takes an input image and tries to generate a copy using a specified number of squares at different positions and different colours. This is done through a crossover procedure which mates two chromosomes together to create a new hopefully better one.

If you agree to take part in this questionnaire, your answers and details will be used within a final report, no contact details will be recorded for private purposes. Please answer as honestly as possible so that the feedback can be used in a constructive way.

Thank you for your participation.

Full Name:	Joscelyn Egginton
Current Job / Relevant Study:	Computing Course Representative

Does the program load and run without error on your computer system?

— **yes** —— no —

How would you rate the look of the GUI (Graphical User Interface)

terrible 1 2 3 4 5 6 7 **8** 9 10 very good

Would you be inclined to use the underlying software in the future for free?

— **yes** —— no —

If possible please elaborate:

Free and open source is always good.

Do you feel the speed of the program could be improved? If possible explain

I have seen examples on-line that do a similar thing faster. The generations are much faster but a image that is recognisable is much slower than that example.

What kind of shape would you have used for the genetic algorithm drawing? —

square —— rectangle —— circle —— **triangle** —

What operating systems do you use on a regular basis?

— MacOS —— **Windows** —— Linux —

If you have any final comments, please do so below:

Thank you.

Questionnaire on 12046280 Final Year Project

This questionnaire is to help evaluate the software that was developed for the Computer Science final year project.

The underlying software is a genetic algorithm that takes an input image and tries to generate a copy using a specified number of squares at different positions and different colours. This is done through a crossover procedure which mates two chromosomes together to create a new hopefully better one.

If you agree to take part in this questionnaire, your answers and details will be used within a final report, no contact details will be recorded for private purposes. Please answer as honestly as possible so that the feedback can be used in a constructive way.

Thank you for your participation.

Full Name:	Alex Induchny
Current Job / Relevant Study:	Owner of Nexus Websites Ltd

Does the program load and run without error on your computer system?

— **yes** —— no —

How would you rate the look of the GUI (Graphical User Interface)

terrible 1 2 3 4 5 6 **7** 8 9 10 very good

Would you be inclined to use the underlying software in the future for free?

— **yes** —— no —

If possible please elaborate:

We are always looking for professional pieces of software that bring in new customers.

Do you feel the speed of the program could be improved? If possible explain

N/A

What kind of shape would you have used for the genetic algorithm drawing? —

square —— **rectangle** —— circle —— triangle —

What operating systems do you use on a regular basis?

— MacOS —— **Windows** —— Linux —

If you have any final comments, please do so below:

The software does seem professionally coded and works well. What we think of within my company most of the time is aesthetics and although the quality is good, it is always something we strive to keep at its highest with a number of measures in place to ensure quality.

Thank you.

Questionnaire on 12046280 Final Year Project

This questionnaire is to help evaluate the software that was developed for the Computer Science final year project.

The underlying software is a genetic algorithm that takes an input image and tries to generate a copy using a specified number of squares at different positions and different colours. This is done through a crossover procedure which mates two chromosomes together to create a new hopefully better one.

If you agree to take part in this questionnaire, your answers and details will be used within a final report, no contact details will be recorded for private purposes. Please answer as honestly as possible so that the feedback can be used in a constructive way.

Thank you for your participation.

Full Name:	Nicolas Wearing
Current Job / Relevant Study:	Full Time Developer

Does the program load and run without error on your computer system?

— **yes** —— no —

How would you rate the look of the GUI (Graphical User Interface)

terrible 1 2 3 4 5 6 7 8 **9** 10 very good

Would you be inclined to use the underlying software in the future for free?

— yes —— **no** —

If possible please elaborate:

It is a very interesting program but my work deals with real time programming. The results would have to be near instantaneous to benefit my work.

Do you feel the speed of the program could be improved? If possible explain

Great speed!

What kind of shape would you have used for the genetic algorithm drawing? —

square —— rectangle —— circle —— triangle —

What operating systems do you use on a regular basis?

— MacOS —— **Windows** —— **Linux** —

If you have any final comments, please do so below:

Polygons would definitely be of benefit for the quality of the produced image. Squares are just too rigid for an image that is not at all.

Thank you.

6.7 Analysis

The results of questionnaires have brought about a wide range of information that will now be analysis in order to understand the views from other perspectives.

Going in descending order. The data shows that 1 our of the total people surveyed had problems loading the program itself. Even though this is not a good result, it was expected as with most software, catering for everyone's individual computer system requirements is next to impossible as seen with bigger project like computer games. That being said, after further questioning, the problem seems to be due to the lack of graphics card on the individuals system. This has the result of the OpenCL library attempting to use the integrated GPU which does not have OpenCL compatibility. There are two possible solutions to this; one would be to attempt to use the CPU as the queue device as most these days are compatible with the framework. No, 2 would be to implement the kernel also within C++ itself which would serve as a backup in case OpenCL has no available devices at all. With that being said, having the program work on a completely sequential nature would slow it down to such a degree that it may not even be worth implementing it that way. If a C++ version was needed it would be for a future development and not be part of this project specification.

Graphically, the program was rated fair from a style perspective. That seems a fair appraisal as aesthetic design was not a focus of the project or the design documentation and does not need to be as the front end is completely detached from the back end genetic algorithm code. The GUI is simply a solution to visually show the results to the user on a basic level but also giving them a variety of control for the algorithm.

All but one person did not comment on the issue of speed in relation

to the generating of the image. This is believed to be because there is no reference for the user to compare it to. This exact algorithm is not something that is used every day and because of that is hard to judge, also different specifications of computers will obviously give different speed results. The one person who did comment on this question mentioned that they had found something similar on-line and although the speed of the generations was faster, the time for when a discernible image is shown is slower. This ties into the next question of the kind of shape they would of personally used for the drawing kernel. Squares and indeed triangles are possibly the worst shapes to use to approximate an image because of their corners, jagged edges and lack of size parameters. This is fine though because a square was specifically chosen to see if it was possible and also how much of a difference between it and the best type which is undoubtedly a shape with moveable vertices. The aim of the project was not to copy other existing example but improve in some way, and squares do improve upon polygons in the computational speed.

Moving on, as expected, windows and Linux are the most popular operating systems selected in total for these questionnaires, these are shown from previous research to be the most common in this type of field with graphical designers leaning more towards Mac-OS as their preferred system. MacOS was selected from one of the questionnaires which is not on the list of platforms to design for as stated previously, this is an expected answer as the field of the individual in question was slightly outside the target audience but still relevant. Also it can be seen that those with roles with more graphical backgrounds do seem to rate the GUI design less than those from a more technical field like computer science.

Overall the evaluation have given critical and good feedback, highlighting

issues within the program and design. This shows how a good evaluation can show problems that were not seen or fully understood by those close to the project but also how a lack of evaluation can lead to problems for all.

7 Future Developments

Porting the software over to Linux as well as windows would be an interesting task and was one of the original aims to have accomplished from the project. This would involve changing the threading classes for the genetic algorithm to use Linux threads which would not be difficult, fortunately OpenGL works on both platforms through a little tweaking and OpenCL being a cross platform standard would only need a Linux library. The reason for not implementing these changes was to give more focus to the quality of the core documentation.

As well as multiple platforms, more options for the user through the GUI would be of benefit for all which could include a selection of source images or loading their own but also different image sizes whereas 250x250 is the current pixel size and this is fixed. Also giving the user the ability to not use OpenCL as a multi-threading platform would be notable because some users may have no GPU for example or no devices that are OpenCL enabled. Obviously at the moment anyone without a GPU that is OpenCL enabled cannot use the program but this was thought to be of importance to give a massive increase in speed compared to a purely CPU threaded approach.

The main aspect to develop would be research into using different shapes as a basis for the program, this is because as shown by evaluation and test results, squares are not the best shape to use and even though this was known from the beginning, the aim was not to use the best shape but to go through the process and learn from mistakes. A project into the speed vs quality

of different shapes would be of great benefit for all and would make for an interesting research project.

8 Conclusions

There are many biological inspired algorithms out there, this makes sense as in general we as a species try to mimic aspects of the natural world, even from other species as we believe these traits will help us with developments relating to specific fields. In medicine, regenerative aspects of certain species are being researched and hopefully developed at this very moment. Engineering also has learnt from aspects of the evolved world, as an example bones remove material from areas it is not needed, this lightens the structure, many building materials use this principle. The same thought process goes into software development and principles, we tend to take things if possible from nature and use it to our advantage whenever available, this is the beauty of nature/biological algorithms.

Looking back at the objectives of this project, all four would seem to be been achieved to a high standard. The objectives were not set high because they are simply pulled in essence from the original project specification and those objectives gave a basis of what was needed to be accomplished by the end of the project. Unexpectedly, the one objective that was originally thought to be easy, objective 3 which was to develop the program was found to be much tougher than expected. This was not a issue with the use of the features of the programming language but with the algorithm not giving the results that were expected of it. This has been mainly fixed as an issue by putting more time into the design and research as there were some components that were not reported in enough detail.

The aims have all been accomplished except for the 1st because of time constraints but the precedence has been set for that to be achieved in the future. This is a feature that was designed for from the start with the use of cross platform software, this enables future development for this feature to easily export the system over to different operating systems if need be.

The use of the OpenCL framework has allowed the developed software to run at a speed far beyond that of any sequential program. Although not a required part of the project as a whole, the use of it is to allow the generations of the program to rapidly be increased

This report detailed a selection of biological algorithms, relating loosely to the solving of a factitious problem to demonstrate the potential in programming. The outcome was to produce a set of research documentations that detail a specific set of biological algorithms, aiming to have enough on how to reproduce them in code and the overall feeling is that this was achieved.

8.1 Milestone 1 Comments

The milestone went exactly to plan with daily research being done on the subject from the day the project was released. There were no problems and all three stages were delivered to a hopefully high standard and on time. The only issue with this milestone was the intended project deliverable, it was decided from the start but because of the lack of design in the milestone, it was not put into full perspective and thus has changes a fair amount since then for the better.

8.2 Milestone 2 Comments

This milestone was a much harder task than the 1st being a much larger chunk of the work and with the extra load of creating a final deliverable. Unfortunately the report was delayed by a fair amount of time mainly due to other coursework being released and one in particular taking 3 straight weeks to complete. Fortunately it is believed that the quality of the report has not dropped because of that due to the fact that extra time was taken to catch up including a few into late hours.

The deliverable was also a harder task than originally thought. This was shown with the prototype that would not function to a desirable degree and this was documented within the prototype section.

Objective Settings Proforma

Students Name:	JASON GRAY
First Assessor:	DR COLIN MORRIS
Second Assessor:	DR PAUL ANGEL

Project Title:

Biological Algorithms and a Real World Application in Image Synthesis

Project Objectives & Deliverables

1. Research biological algorithms in detail and develop a documentation that shows the working of each so that it can be used for future development.
2. Design the development in detail.
3. Create and develop a biological algorithm that solves a problem.
4. Research any other information pertaining to each individual biological algorithm.

Agreed Marking Scheme Weightings

Mark Category	Proposed Weighting	Agreed Weighting	Mark Allocated
Project Management (Only set by Supervisor 1)	50-80		
Originality & Self-Direction	40-80		
Technical Complexity	20-80		
Solutions, Evaluation & Conclusions	50-120		
Final & Sub-Report Quality	50		
Prototype / System Demo Or Project Deliverable	50-100		
Sponsor Mark	0-60		
Sub-Total Marks	——		
Sub-Total Percentage	——	60%	%
Milestone 1 (initial)	——	20%	%
Research & Research Applied			
Final Presentation	——	20%	%
TOTAL PERCENTAGE	——	100%	%

References

[1] Angel, P. (2016) Core Structures.

[2] Angel, P. (2016) OpenCL Setup and Helper Functions.

[3] Amoroso, S and Cooper, G. (1971) Tessellation Structures for Reproduction of Arbitrary Patterns, *Journal of Computer and Systems Sciences*. [Online] Science Direct. 5(5). pp.455-464. Available at: http://www.sciencedirect.com. (Accessed: 20th October 2015).

[4] Cummings, C. (2010) *Grow Your Own Picture*. Available at: http://chriscummins.cc/s/genetics/ (Accessed: 1st April 2016).

[5] Davis, L. (1991) *Handbook of Genetic Algorithms*. 1st Ed. New York: Van Nostrand Reinhold.

[6] Comte, P. Vassiliev, S. Houghten, S and Bruce, D. (2011) 'Genetic algorithm with alternating selection pressure for protein side-chain packing and pK(a) prediction', *Biosystems*. 105(3), pp.263-270. ScienceDirect [Online] Available at: http://www.sciencedirect.com (Accessed: 6th April 2016).

[7] Crespin, D. (2013) *Equivalence of Polyhedrons and Perceptrons*. [Online] Available at: https://www.academia.edu/3420217/Equivalence_of_Polyhedrons_and_Perceptrons (Accessed: 13th March 2016)

[8] Eberhart, Russell C and Shi, Yuhui. (2001) Particle Swarm Optimization: Developments, Applications and Resources [Online] Available at: http://www.researchgate.net/profile/Yuhui_Shi2/publication/3903911

_Particle_swarm_optimization_developments_applications_and_resources/links/54d9a96e0cf2970e4e7c55d4.pdf (Accessed: 17th November 2015).

[9] Gediga, G. Hamborg, KC. and Duntsch, I. (2002) 'Evaluation of Software Systems', In: A. Kent and J. G. Williams, eds. *Encyclopedia of Computer Science and Technology, Vol 44*, Marcel Dekker Incorporated. pp 162-196.

[10] Ghassemi Tari, Farhad and Alaei, Reza (2013) Scheduling TV commercials using genetic algorithms, *International Journal of Production Research*. [Online] EBSCOhost. 51 (16). pp. 4921-4929. Available at: https://www.ebscohost.com. (Accessed: 9th November 2015).

[11] Harao, Masateru and Noguchi, Shoichi. (1978) On Some Dynamical Properties of Finite Cellular Automaton, *IEEE Transactions on Computers*. [Online] IEEEXplore Vol 27(1), pp.42-52. Available at: http://ieeexplore.ieee.org.ergo.glam.ac.uk. (Accessed: 20th November 2015).

[12] Hornby, Gregory S. Globus, Al. Linden, Derek S. Lohn and Jason D. (2006) *Automated Antenna Design with Evolutionary Algorithms*. [Online] Available at: http://ti.arc.nasa.gov/m/pub-archive/1244h/1244

[13] Java (2015) *Learn about java technology*. [Online] Available at: https://www.java.com/en/about/. (Accessed 26th November 2015).

[14] Jevtic, Aleksandar. Melgar, Ignacio and Andina, Diego. (2009) 2009 35th Annual Conference of IEEE Industrial Electronics, Nov. 2009, *Ant based edge linking algorithm*. [Online] IEEE Xplore Digital Library. Vol. 51(16) p.3353-3358. Available at: http://ieeexplore.ieee.org/. (Accessed 6th November 2015).

[15] Kumar, Surendra and Rao, C.S.P (2009) Robotics and Computer Integrated Manufacturing, *Application of ant colony, genetic algorithm and data mining-based techniques for scheduling*. [Online] ScienceDirect. Vol. 25(6), pp.901-908. Available at: www.sciencedirect.com. (Accessed: 24th November 2015).

[16] Li, Qing. Yin, Yixin. Wang, Zhiliang and Liu, Guangjun. (2007) Advances in Neural Networks - ISNN 2007. [Online]. Available at: http://books.google.co.uk (Accessed: 9th November 2015).

[17] Miller, Brand L and Goldberg, David E. (1996) Evolutionary Computation, *Genetic Algorithms, Selection Schemes, and the Varying Effects of Noise*. [Online] EBSCOhost 4(2). pp.113-131. Available at: https://www.ebscohost.com/ (Accessed 10th November 2015).

[18] Mnemosyne Studio. (2012) *Particle Swarm Optimization introduction*. [Online] Available at: http://www.mnemstudio.org/particle-swarm-introduction.htm. ()Accessed: 21th November 2015).

[19] NetMarketShare. (2015) *Operating system market share*. [Online] Available at: https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0 (Accessed: 19th November 2015).

[20] Prechelt, Lutz (1999) Communications of the ACM, *Technical Opinion: Comparing Java vs. C/C++ Efficiency Differences to interpersonal Differences*. [Online] ACM Digital Library Vol.42 (10), pp.109-122. Available at: http://www.acm.org.ergo.glam.ac.uk/. (Accessed 26th November 2015).

[21] Sarkar, P. (2000) ACM Computing Surveys, *a brief history of cellular automata*. [Online] ACM Digital Library. 31(1) pp.80-107. Available at: http://dl.acm.org.ergo.glam.ac.uk/ (Accessed: 20th October 2015).

[22] Sarle, WS (1994) Neural Networks and Statistical Models, *Proceedings of the Nineteenth Annual SAS. 10-13 April*. SAS [Online] Available at: http://people.orie.cornell.edu/davidr/or474/nn_sas.pdf (Accessed: 8th March 2016)

[23] Schmidhuber, J. (2015). *I am Jrgen Schmidhuber, AMA! Reddit*. [Online] Available at: https://www.reddit.com/r/MachineLearning/comments/2xcyrl/i_am_j%C3%BCrgen_schmidhuber_ama (Accessed: 16th November 2015).

[24] Yuan, Shuai. Skinner, Bradley. Huang, Shoudong and Liu, Dikia. (2013) European Journal of Operational Research, *a new crossover approach for solving the multiple travelling salesmen problem using genetic algorithms*. [Online] Science Direct Vol. 228(1). pp.72-82. Available at: http://www.sciencedirect.com/ (Accessed 10th November).