Dynamic Terrain

Sibo Yao

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

December 2008

Your file Votre référence
ISBN: 978-0-494-63160-7
Our file Notre référence
ISBN: 978-0-494-63160-7

ABSTRACT

Dynamic Terrain

Sibo Yao

In computer graphics, realistic 3D modeling has been becoming a challenging research area. The terrain synthesis, one of the 3D models, is being developed with two approaches: real geographical data based and fractal terrain. Evolutionary Computation (EC) mimics Darwin's principles of natural evolution processes by representing a chromosome and then applying crossover and mutation.

In this thesis, we are interested in how the EC applies to terrain modeling and generates a self-adaptive dynamic terrain system, which can adapt to the personal mannerism of different users. The thesis reviews the background and related works about terrain synthesis and evolutionary computation including related theories and practices. This thesis presents a dynamic adaptive terrain system based on Evolutionary Computation - a specific genetic algorithm. The EC algorithm has been designed and implemented in the EC module; the graphic terrain module shows the result of terrain. The system requires a user's input – mouse click event, drive the interaction between the EC module and the dynamic graphic terrain module.

Based on the results that the system has given, the thesis gives out some analysis and conclusions – strengths and weaknesses. The thesis also proposes future works so that researchers picking up this work in future have the benefit of the ideas that thesis generated.

Acknowledgements

I am extremely thankful to my supervisor, Dr. Peter Grogono, who has helped me throughout this work in many profound ways, with guidance, advice, encouragement, support, and patience. This work would not have been completed without his instructions and supervision. I thank him for giving me the opportunity to be his Master's student, for leaving me the freedom to pursue my research with my interests, and for providing me the flexible study atmosphere. I am grateful to him for the rigorous academic style, which makes my work exciting, challenging, and rewarding. This thesis owes much to him.

I would like to express appreciation to my wife Jane for her support, encouragement, and patience during my studies at Concordia University.

I would like to thank my parents and my brother for their support and encouragement towards my graduate studies.

I miss my grandma. Her encouragement always supports me when I encounter difficulties. Moreover, her encouragement will be with me forever.

I would also like to thank Maria at Student Learning Services (SLS), Concordia University.

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

1.1 Motivation and Objective

Computer graphics is an important field in computer science and is used in many other disciplines. Today, computers and computer-generated images touch many aspects of our daily life. Major applications of computer graphics include video games, computer simulation, and computer animation. Graphics need realistic 3D models, and modeling has become one of the major research areas and goals in computer graphics. In the 3D modeling area, terrain synthesis has become a requirement for many applications such as video games. Terrain modeling has been developed and applied in many fields. Depending on the way in which terrain modelling is constructed and implemented, it can be divided into two categories: *real geographical data input based terrain* and *fractal terrain.* The former uses raw data as input or retrieves geographical information stored in satellite or aerial images. The latter mainly uses algorithms such as the mid-point displacement algorithm to construct terrain models

The focus of this thesis is on how Evolutionary Computing can be applied to terrain modeling. We describe a self-adaptive dynamic terrain system that can adapt to the personal mannerisms of different users. The proposal requires the dynamic terrain to be innovative, truly new and original. Moreover, the thesis explores the feasibility of applying Evolutionary Computation to terrain modeling. The areas of evolutionary computation and terrain modeling have both been investigated thoroughly, with many interesting results. Nevertheless, to our knowledge, no one has previously used a genetic algorithm or a genetic program to construct a fractal terrain. This is the central contribution of this thesis.

1.2 Evolutionary Computation

Evolutionary Computation (EC) mimics Darwin's principles of natural evolution processes by representing a chromosome and then applying crossover and mutation to it. EC enables a computer program to be *adaptive*—to continue to perform well in a changing environment. In evolutionary computation, the anticipated emerging behavior is the design of high-quality solutions and the ability to adapt these solutions when confronted with an ever-changing environment [1].

There are several varieties of EC — evolution strategies, evolutionary programming, genetic algorithms, and genetic programming form the backbone of the field of evolutionary computation. In this thesis, we concentrate on genetic algorithms and genetic programming and we design an evolutionary algorithm that is suitable for evolving a terrain with a user's need and feedback.

1.3 Thesis Organization

The remainder of this thesis is structured as follows. All the background about graphics, terrain, fractal, and EC and the related work about terrain modeling and evolutionary computation is reviewed in Chapter 2. Chapter 3 examines the design of the evolutionary algorithm adopted in this thesis and the design of the aspects of the graphics. Chapter 4 provides the implementation details of this dynamic terrain system. After that, the principal results of the thesis are presented in Chapter 5. The conclusion and future work are discussed in Chapter 6.

2. BACKGROUND AND RELATED WORK

In this thesis, we investigate the possibility of applying EC, in the form of Genetic Algorithms (GA) and Genetic Programming (GP), to terrain modeling, develop an adaptive dynamic transformation terrain system, and provide users with an adaptive self-feedback environment. In more detail, we design a suitable specific evolutionary algorithm, based on GA and the GP theories and algorithms, for dynamic terrain modeling. We also implement a system to present the result of our algorithm.

The system requires a user's input – a mouse click event – to drive the interaction between the EC module and the dynamic graphic terrain module. The EC algorithm is implemented in the EC module; the graphic terrain module shows the resulting terrain.

EC has not previously been applied to dynamic terrain modeling. Our study is worthwhile in two senses. Firstly, the EC enables a computer program to constantly adapt to a changing environment. For example, it allows a computer interfaces to be adaptable to the idiosyncrasies of different users. Also, it allows computer programs to be innovative in the sense that they can produce truly new and original results. Secondly, current terrain modeling is limited as it uses real life geographical data as input to construct terrain. Another current alternative applies the mid-point algorithm to generate fractal terrain.

The EC generated dynamic adaptive terrain may be used for applications that require virtual reality. For example, in a video game environment, the terrain is adaptable to the personal mannerisms of different players. As another example, graphic designers may use a dynamic adaptive terrain system to design their own favorite terrain models.

2.1 Graphics

Computer Graphics studies the uses of computers to generate or manipulate images. The core technology of rendering and homogeneous coordinates was developed in the 1960s mainly at MIT, Harvard, and the University of Utah. In computer graphics, *modeling* deals with the mathematical specification of shape and appearance properties in a way that can be stored on the computer. *User interaction* handles the interface between input devices such as mice and tablets, the application and feedback to the user in imagery and other sensory feedback. *Virtual reality* is intended to immerse a user into a 3D virtual world, which typically requires at least stereo graphics and response to head motion. Video gaming, one of the most important graphics applications, uses sophisticated 3D models and rendering algorithms. For example, 3D terrain modeling is often used in video games applications. Also, terrain modeling is needed as software functionality for a graphic designer.

A graphics Application Programming Interface (API) is a software interface that provides a model for an application program to access system functionality, such as drawing an image into a window [2]. Current popular APIs include OpenGL, Direct3D, and Java3D. In this thesis, we use OpenGL as an API during the implementation stage.

Graphics programs need to use 3D geometric models, which describe 3D objects using mathematical primitives such as spheres, cubes, cones, and polygons.

> "Mathematically defined, a polygon is a plane figure specified by a set of three or more coordinate positions, called *vertices*, that are connected in sequence by straight-line segments, called the *edges* or *sides* of the polygon." "By definition, a polygon must have all its vertices within a single plane and there can be no edge crossings." [3]

A polygon mesh is a set of polygons that share vertices and edges, it may contain polygons with any number of vertices and require a moderately sophisticated data structure to store and display efficiently [4]. The figure 2.1 below shows a polygon (rectangular) mesh.

> "The most ubiquitous type of model is composed of 3D triangles with shared vertices, which is often called a triangle mesh." [2]

A typical triangle mesh comprises a set of 3D triangles that are connected by their common edges. In this thesis, triangle mesh is used to construct the terrain model. The speed of most modern graphics pipelines is roughly proportional to the number of triangles being drawn.

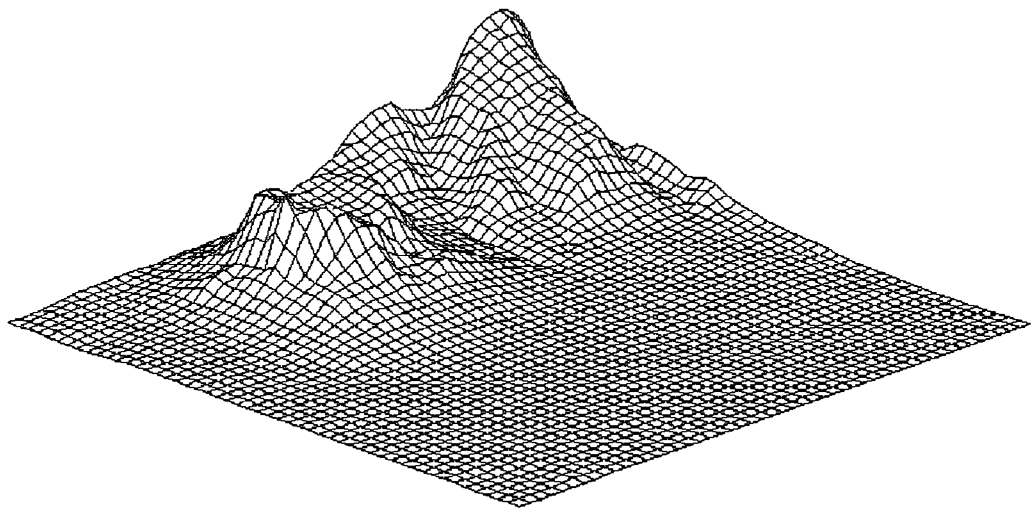

Figure 2.1 Mesh plot of Honolulu data [4]

2.1.1 Terrain Modeling related work

Currently, there are two main approaches used to create the terrain models.

One basic approach for terrain modeling implementation is based on the natural geographical height data of real-life terrain, which are stored in a raw data file. The data are plotted in a 3D coordinates system to form a 3D polygon mesh. Also, texture mapping is usually used for adding details, surface texture, or color.

In this approach, terrain modeling is hard because it needs a lot of data to construct realistic terrain.

2.1.1.1 Fractal Approaches

A terrain may also be generated using a stochastic algorithm designed to produce fractal behavior, which mimics the appearance of natural terrain.

The concept of fractal geometry can be traced back to the IBM mathematician Benoit B. Mandelbrot and the 1977 publication of his book “The Fractal Geometry of Nature". A **fractal** is generally "a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole," [5] a property called self-similarity. A set *F* is a fractal if it has (some) of the following properties:

1. *F* has detail at every scale.

2. *F* is (exactly, approximately, or statistically) self-similar.

3. The "*fractal dimension*" of *F* is greater than its *topological dimension*.

4. There is a simple algorithm description of *F* [6].

The figure below is a fractal surface of a mountain.

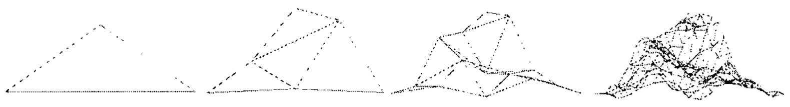

Figure 2.2 A fractal that models the surface of a mountain [7]

The repetition of form over a variety of scales is called *self-similarity*: a fractal looks similar to itself on a variety of scales. In other words, a fractal is an object that is invariant under change of scale. Self-similarity is the key concept behind any fractal. A little piece of a mountain looks a lot like a bigger piece of a mountain and vice versa [8].

Terrain (or Mountain) is perhaps the best-known example of fractals. A smaller part of terrain looks just as terrain-like as larger part, but they are not exactly the same, which is the distinction between *statistical self-similarity*, where only the statistics of random geometry are similar at different scales, and *exact self-similarity*, where the smaller components are exactly the same as the larger ones [8].

Several techniques for generating fractal terrain synthesis exist. The original method for generating fractional Brownian surface uses Poisson faulting [5]. This method involves application of random faults having the Gaussian distribution (see Figure 2.3) resulting in statistically indistinguishable surfaces (planes or spheres).

Most fractal terrain models are based on two faster methods for generating fractal terrain surface: the random midpoint displacement method and Fourier synthesis [9].

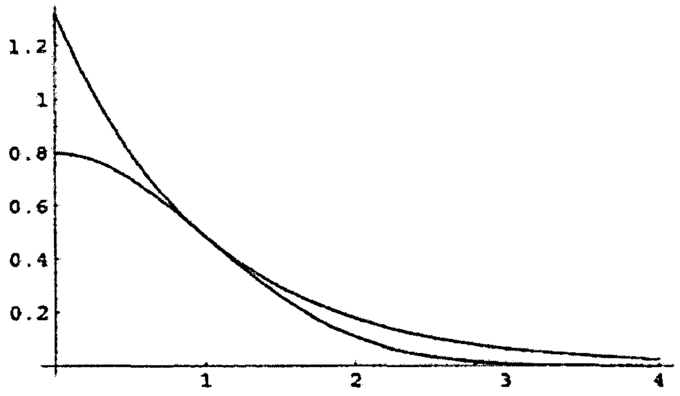

Figure 2.3 Bounding the positive normal density [10]

Musgrave and Kolb have described a new approach. This approach, termed *noise synthesis*, for the synthesis of fractal terrain height fields is presented which, in contrast to previous techniques, features locally independent control of the frequencies composing the surface, and thus local control of fractal dimension and other statistical characteristics. The new technique is intermediate in difficulty of implementation, between simple stochastic subdivision and Fourier filtering or generalized stochastic subdivision [11].

2.2 Evolutionary Computation

Evolutionary computation (EC) is being developed along with the goal to inspire computer programs with intelligence, with the life-like ability to self-reproduce, and with the adaptive capability to learn and to control their environments. In the 1950s and the 1960s several computer scientists independently studied evolutionary systems with the idea that evolution could be used as an optimization tool for engineering problems. The idea in all these systems was to evolve a population of candidate solutions to a given problem, using operators inspired by natural genetic variation and natural selection [1]. Genetic algorithms (GAs) were invented by John Holland in the 1960s and were developed by Holland and his students and colleagues at the University of Michigan in the 1960s and the 1970s. Other evolutionary algorithms include evolution strategies (Rechenberg, 1973; Schwefel, 1981), evolutionary programming (Fogel et al., 1966), and genetic programming (Koza, 1992). Together, evolution strategies, evolutionary programming, genetic algorithms, and genetic programming form the backbone of the field of evolutionary computation.

Nowadays, many computational problems require complex solutions that are difficult to program by hand. The best route to artificial intelligence is through a "bottom-up" paradigm in which humans write only very simple rules, and complex behaviors such as intelligence emerge from the massively parallel application and interaction of these simple rules. In evolutionary computation the rules are typically "natural selection" with variation due to crossover and/or mutation.

Evolutionary algorithms are composed of a class of search, selection, adaptation, and optimization techniques based on Darwin's principles of natural evolution [12]. Nowadays, evolutionary algorithms have been successfully applied to a large number of problems from different domains, including engineering, computer science, cognitive science, economics, management science, optimization, automatic programming, machine learning, operations research, immune systems, ecology, population genetics, social systems, and other fields.

Computer Graphics

Karl Sims describes how evolutionary techniques of variation and selection can be used to create complex simulated 3D plant structures, solid textures, and motions for use in computer graphics and animation [13]. He also describes a system for the evolution and co-evolution of virtual creatures that compete in physically simulated three-dimensional worlds [14]. He applied the EC to 3D modeling and an adaptive interactive system. A terrain model is a complex 3D model simulating a natural terrain structure, but being constructed artificially. The objective of this thesis is an adaptive system and realistic terrain models.

Machine Learning

EC has been applied to tackle problems in robot control in which a robot has to perform a task in a variable environment. For example, Andre and Teller [15] provided the genetic programming algorithm for robots in a soccer game with a set of primitive control functions such as turning, moving, kicking, and so on. In this thesis, we are interested in implementing a system that provides a variable environment and in modeling a terrain to adapt the changing environment.

Engineering

Evolution strategies have been used to optimize the final shape of a jet nozzle [16].

Altshuler and Linden used their GA to design a circularly polarized seven-segment antenna with hemispherical coverage.

Weismann, Hammel and Bäck [17] applied evolutionary algorithms to a "nontrivial" industrial problem: the design of multilayer optical coatings used for filters that reflect, transmit or absorb light of specified frequencies. EC is a useful tool to design and optimize models in Engineering.

In this thesis, we use a specific GA as a tool to evolve the terrain model and produce the desirable shape of the terrain.

Mathematics and algorithms

Haupt and Haupt [18] discuss the use of GAs to solve high-order nonlinear partial differential equations, typically by finding the values for which the equations equal zero, and give as an example a near-perfect GA solution for the coefficients of the fifth-order Super Korteweg-de Vries equation.

Koza et al. [19] used genetic programming to evolve minimal sorting networks for 7-item sets (16 comparisons), 8-item sets (19 comparisons), and 9-item sets (25 comparisons).

It is hard to find a simple mathematical representation for a realistic terrain model. In this thesis, EC, in the form of Genetic Algorithms, is used as a relatively easier tool to evolve the terrain models.

Pattern Recognition

Rizki, Zmuda and Tamburino [20] used evolutionary algorithms to evolve a complex pattern recognition system with a wide variety of potential uses. In this thesis, we use an evolutionary algorithm to evolve an adaptive self-feedback terrain system that can adapt to the personal mannerisms of different users.

Routing and scheduling

He and Mort [21] applied genetic algorithms to the problem of finding optimal routing paths in telecommunications networks (such as phone networks and the Internet) which are used to relay data from senders to recipients. In this thesis, we apply a suitable genetic algorithm to the problem of finding desirable shapes of terrain model in an adaptive environment with users' feedbacks.

Business and economics

Genetic algorithms have been applied to theoretical questions in economic markets, to time series forecasting, and to econometric estimation. String-based genetic algorithms have been applied to finding market-timing strategies based on fundamental data for stock and bond markets by Bauer (1994) [22]. Franklin Allen and Risto Karjalainen [23] use genetic algorithms to find technical trading rules. British bank evolved creditability model to predict loan paying behavior of new applicant.

In this thesis, the system is dynamic and is able to find specific terrain models adaptive to users' feedbacks.

2.2.1 Evolutionary Algorithms

An evolutionary algorithm maintains a population of solution candidates and evaluates the quality of each solution candidate according to a problem-specific fitness function, which defines the environment for the evolution. New solution candidates are created by selecting relatively fit members of the population and recombining them through various operators such as crossover and mutation. Specific evolutionary algorithms may differ in some or all of the followings: the representation schemes of solutions, the selection mechanism, the details of the recombination operators, and the termination criteria [12]. In EC, chromosomes represent individuals (solution candidates), which are either solutions for GA or programs providing solutions for GP. Crossover happens between selected individual pairs, and selected single individuals will mutate.

Initially, a group of randomly created individuals forms the first generation of population. Then the fitness function is applied to each individual to evaluation how it fits the environment. The better its fitness value is, the greater is the chance that it will be selected for mating. If two individuals are selected for crossover, they exchange part of their information with each other. Mutation modifies a single individual randomly. This whole process iterates until the population reaches a termination criterion [12].

In Genetic Algorithms (GA), finite binary strings, known as the genomes, are used to represent chromosomes (individuals) – i.e., possible solutions to the problem. The string encodes a possible solution in a given problem space, which is referred to as the *search space*. The genetic algorithm is usually applied to spaces which are too large to be searched exhaustively. When two individuals perform crossover, they swap substrings. For example, the following 2 individuals perform crossover. They swap underlying part of themselves with each other.

110001011001

101011001011

After the crossover is done, they become

110001001011

10101**1011001**

The mutation just randomly changes a bit of a string. For the following individual, the underlying bold bit is randomly selected. 101**1**0100101.

After mutation, it becomes 10100100101.

Genetic programming also maintains a population of genetic structures. Solution candidates are represented as hierarchical compositions of functions — tree structures. The initial population consists of random trees. The root node of a tree is chosen at random among functions of the same type as the desired composite function. Each argument of that function is then selected among the functions of the appropriate type, proceeding recursively down the tree until a function with no arguments (a terminal node) is reached. The evolution takes place much as in the basic genetic algorithms, selecting relatively fit solution candidates to be recombined and replacing unfit individuals with the offspring. In genetic programming, the crossover operator recombines two solution candidates by replacing a randomly selected sub-tree in the first parent with a sub-tree from the second parent.

The following two sections describe two specific evolutionary algorithms: genetic algorithm and genetic programming, which we concentrate on in this thesis.

2.2.1.1 Genetic Algorithms

A *genetic algorithm* is an iterative procedure that consists of a fixed-size population of individuals, a fitness function, and operators. Each individual is represented by a finite string of symbols, known as the *genome*, encoding a possible solution in a given problem search space. The standard genetic algorithm starts with an initial randomly generated population of individuals. Every evolutionary step, known as a generation, the individuals in the current population are decoded and evaluated according to some predefined quality criterion, referred to as the fitness, or fitness function. The operators crossover and mutate are performed to introduce the new individuals. Figure 2.4 presents the standard genetic algorithm in pseudo-code format.

```
begin GA
   g:=0 { generation counter }
   Initialize population P(g)
   Evaluate population P(g) { i.e., compute fitness values }
   while not done do
      g:=g+1
      Select P(g) from P(g - 1)
      Crossover P(g)
      Mutate P(g)
      Evaluate P(g)
   end while
end GA
```

Figure 2.4 Pseudo-code of the standard genetic algorithm.

Initialization Step

The standard genetic algorithm proceeds with an initial population of individuals which is generated randomly.

Evaluation Step

During every evolutionary step, known as a *generation*, the individuals in the current population are decoded and evaluated according to some predefined quality criterion, referred to as the *fitness*, or the *fitness function*.

Selection Step

To form a new population (the next generation), individuals are selected according to their fitness with a probability proportional to their relative fitness, which ensures that the expected number of times an individual is chosen is approximately proportional to its relative performance in the population. Thus, high-fitness ("good" or "fitter") individuals stand better chances to be selected to reproduce, while low-fitness ones are more likely to disappear.

Crossover Step

Selection alone cannot introduce any new individuals into the population, i.e., it cannot find new points in the search space; these are created by operators inspired by genetics — crossover and mutation. Crossover is performed between two selected individuals, called parents, by exchanging parts of their genomes (i.e., encodings) to form two new individuals, called *offspring*. In its simplest form, substrings are exchanged at a randomly-selected point. The crossover operator roughly mimics biological recombination between two single-chromosome organisms.

Mutation Step

This crossover operator tends to enable the evolutionary process to move toward "promising" regions of the search space. The mutation operator is introduced to prevent premature convergence to local optima by randomly sampling new points in the search space. Premature convergence happens when a population for a problem converged too early, resulting in being non-optimal. In this context, the parental solutions are not able to generate offspring that are superior to their parents. Premature Convergence can happen in case of loss of genetic diversity. Mutation is carried out by flipping bits at random, with small probability.

Termination

The termination condition may be specified as some fixed maximal number of generations or as the attainment of a fitness level that is acceptable to the user.

2.2.1.2 Genetic Programming

Genetic Programming (GP) differs from GA in its representation. In GA, genetic structures are represented as character or bit strings of fixed length. GP partly breaks the restrictions of the fixed-length representation of genetic structures. In GP, syntax trees with terminal and non-terminal nodes are used to represent chromosomes. In these tree-like structures, the non-terminal nodes (i.e., nodes with successors) are operators. The terminal nodes (i.e., nodes with no successors) correspond to the operands. As a result, a solution candidate is represented as a hierarchical composition of function – a tree-like structure [12].

The entire tree is evaluated recursively by evaluating each node of the tree with preorder tree traversal. The syntax trees representation omits the grouping parentheses, since in a syntax tree the grouping of operands is implicit in the tree structure.

Crossover

In GP, crossover is performed by swapping sub-trees of two individuals.

Before crossover, there are individuals A and B.

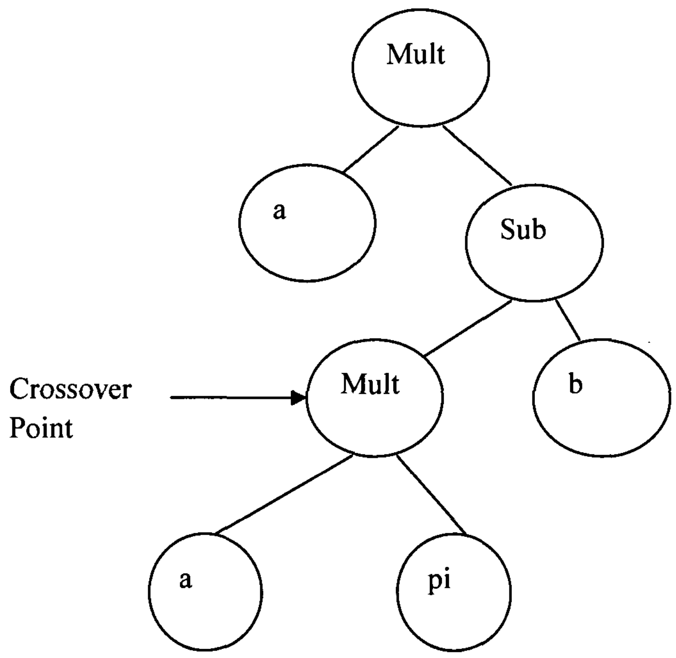

Figure 2.5 Individual A

The individual A is evaluated as a * ((a * pi) – b) with the use of the algorithm described below.

Evaluate the syntax tree algorithm

```
eval(node) =
  if node is a constant
    return constant value
  else if node is a variable
    return variable value
  else // node is an operator
    lhs = eval(left_subtree)
    rhs = eval(right_subtree)
    return lhs <op> rhs [24]
```

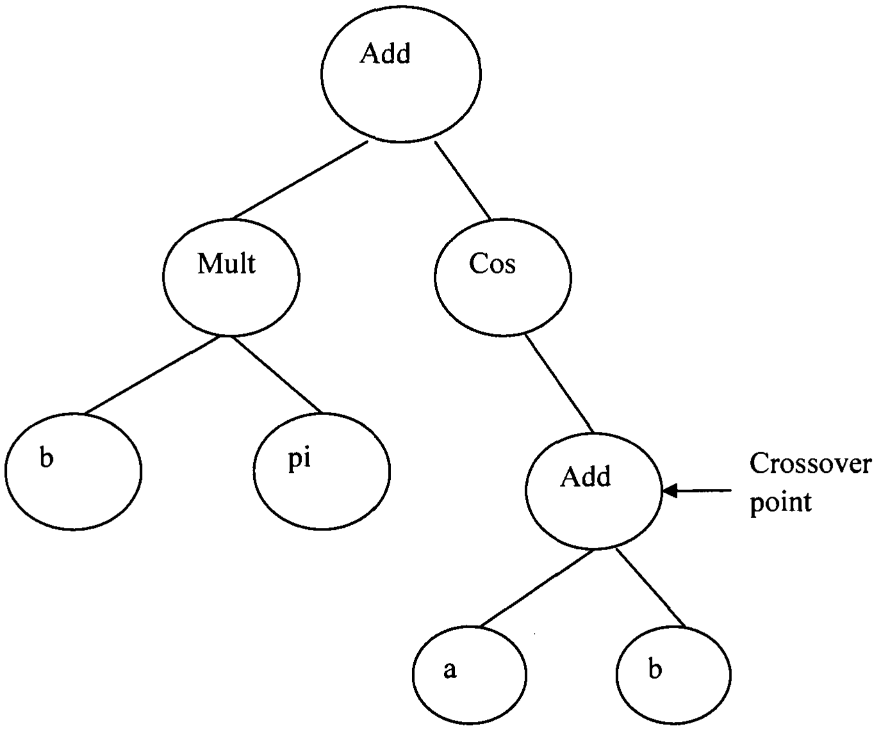

Figure 2.6 Individual B

The individual B is evaluated as (b* pi) + cos (a + b)

After the crossover operation between individual A and B, individual A becomes A', and individual B becomes B'.

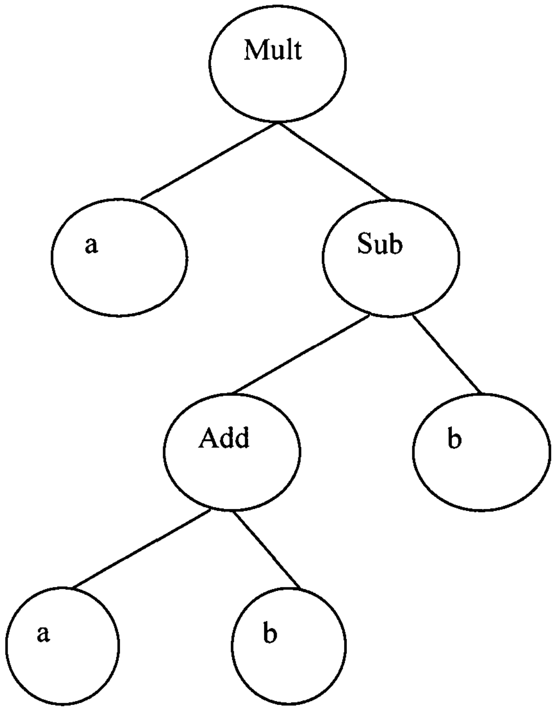

Figure 2.7 Individual A’

The individual A’ is evaluated as a*((a + b) – b).

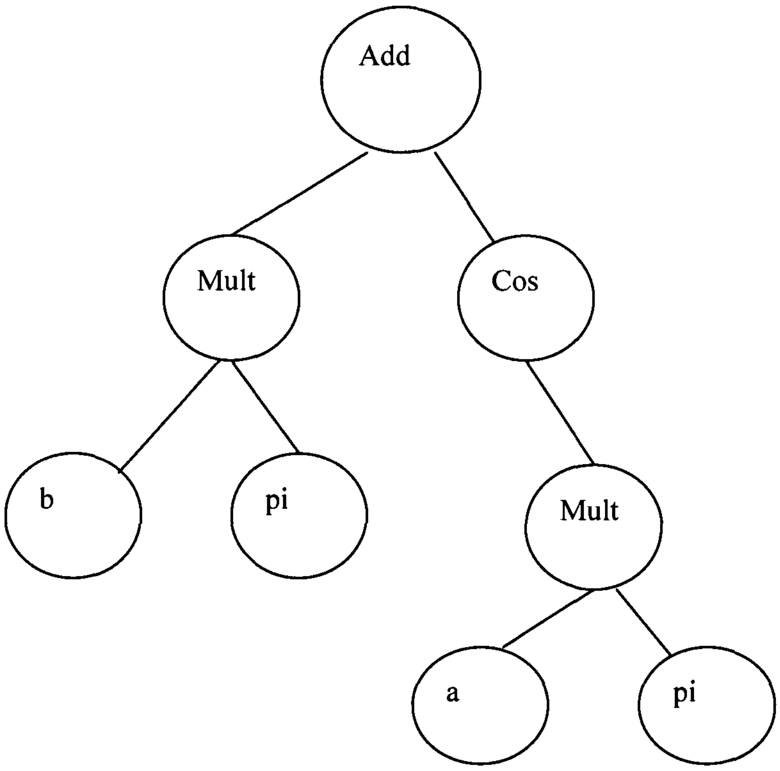

Figure 2.8 Individual B'

The individual B' is evaluated as (b * pi) + Cos (a * pi) [12].

During the mutation operation, a single individual (program) is selected with low probability. Then, a mutation point (Node) is randomly chosen from the individual. That Node is deemed as root of a sub-tree, which is deleted. A new sub-tree grows at that mutation point. The final overall effect is a new sub-tree replaces the original sub-tree [12].

3. DESIGN

3.1 Graphics

The simplest form of terrain representation is a regular height information grid, i.e. considering a regular grid in the plane XZ, with evenly spaced points, a height is attributed to each point. This representation saves a lot of space when storing the terrain because we will only need the heights, and a reference point in the terrain, for instance, the centre point in the plane XZ. Since the grid is evenly spaced, we do not need to store both *x* and *z* values for each point

3.1.1 Fractal and Midpoint Displacement Algorithm

The mid-point replacement algorithm is applied in this thesis to add more details to make the terrain look smoother and finer. The midpoint displacement algorithm itself is relatively simple, but it can produce complex graphics relatively quickly. The algorithm for one-dimension basically iterates a large number of times for the following steps: find the midpoint of the line segment and displace the midpoint's y value by a random amount.

Figure 3.1 (3.2 for more iterations) illustrates this recursive method in the xy plane.

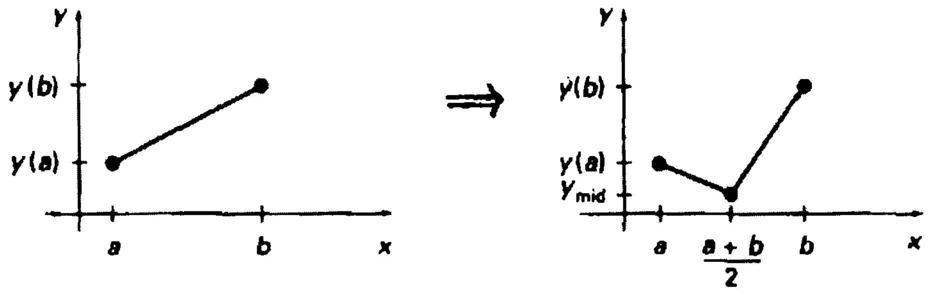

Figure 3.1 Random midpoint-displacement of a straight-line segment [3]

It starts with a straight-line segment, and then calculates a displaced y value for the mid-position of the line as the average of the two endpoint y values plus a random offset:

$$y_{mid} = 1/2\ [y(a) + y(b)] + r$$

The random value r is calculated based on a random number from a Gaussian distribution in order to approximate functional Brownian motion. One way to calculate a random offset is to take $r = s * r_g * |\ b - a\ |$.

In this equation, the parameter s is a selected "surface-roughness" factor; r_g is a Gaussian random value. The process is then iterated by computing a displaced y value for the mid-position of each half segments of the subdivided line. We continue the subdivision until the subdivided line sections are less than a selected value. At each step, the value of the random variable r reduces because it is proportional to the width $|\ b - a\ |$ of the line segment to be subdivided. The offset value r can be negative, in which case the value of the midpoint in vertical direction is decreased, and the point moves down as the point D_{22} illustrated in figure 3.2 [3].

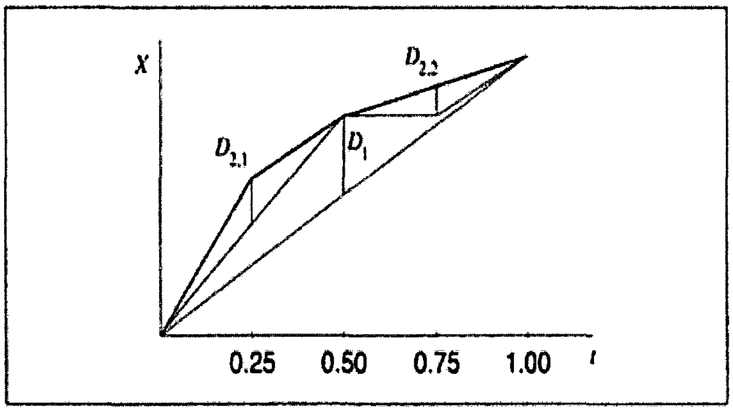

Figure 3.2 A two iterations of the random midpoint-displacement procedure [25]

Figure 3.3 illustrates a fractal curve obtained with this random midpoint displacement method.

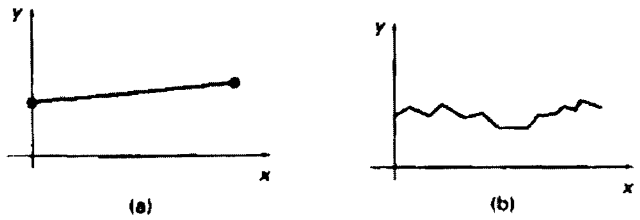

Figure 3.3 A random-walk path generated from a straight-line segment with four iterations of the random midpoint-displacement procedure [3]

Terrain features are generated by applying the random midpoint-displacement procedures to a rectangular or a square ground plane (Fig. 3.4).

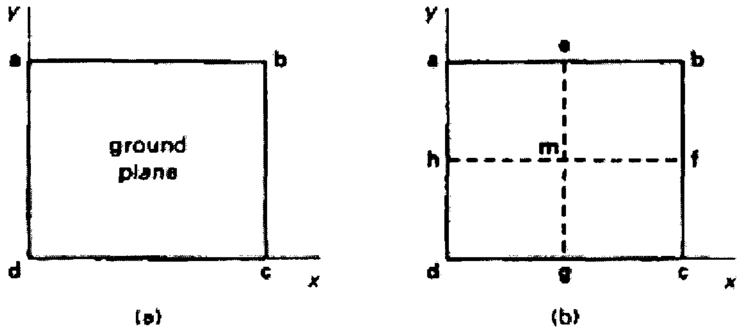

Figure 3.4 A rectangular plane subdivided into four sections in a random midpoint displacement procedure to calculate terrain elevations [3]

We begin the procedures by assigning an elevation (height value, here called z value) to each of the four corners (a, b, c, and d in fig.3.4) of the plane. Then we subdivide the ground plane at the midpoint of each edge to generate the five new grid positions: e, f, g, h, and m. Elevations at mid-positions e, f, g, and h of the ground-plane edges can be calculated as the average elevation of the nearest adjacent two vertices plus a random offset. For example, elevation at mid-position e is calculated using vertices a and b:

$$z_e = (z_a + z_e)/2 + r_e$$

Also, height value z at mid-position f is calculated using vertices b and c with the same way as shown in the equation above. The random value r_e can be obtained as the product of surface-roughness factor times the grid separation times a Gaussian random number. The elevation z_m of the ground-plane mid-position m can be calculated using positions e and g, or positions f and h. An alternative to calculate z_m is to use the assigned height values (elevations) of the four ground-plane corners:

$$z_e = (z_a + z_b + z_c + z_d)/4 + r_e$$

This process proceeds iteratively for each of the four new grid sections at each step until the grid separation becomes smaller than a preset value [3].

Figure 3.5 and 3.6 are used to simply illustrate the 3D effects of the algorithm after the first pass and second pass.

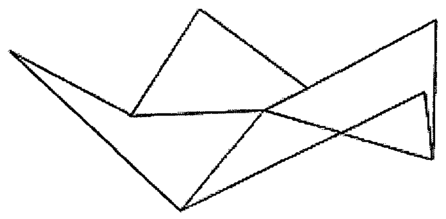
Figure 3.5 A 3D image result of the midpoint displacement algorithm [26]

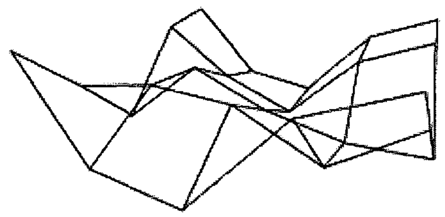
Figure 3.6 Another 3D image result of the midpoint displacement algorithm – one more pass [26]

3.1.2 Texturing and Viewing the Terrain

The terrain model is textured with 2D texture mapping. We use 2D coordinate, called *uv*, to create a reflectance R (*u, v*). Then, we take an image and associate a (*u, v*) coordinate system on it so that it can, in turn, be associated with points on a 3D surface. We need to load the texture data (i.e., a terrain bitmap) and use the functions in the *glaux* library to store the bitmap data, and bind the texture to the texture arrays index and initialize the texture.

The view of the terrain scene is rendered through a polar camera, so the viewer is able to hover over the terrain. A *polar* camera is a camera positioned with polar coordinates so that it moves around the surface of a sphere and always looking at the origin. The viewer can change the radius of the camera track sphere, using the f/F keys to move forward and backwards respectively on the line joining the point, an approximate center of gravity, centrally located on the terrain and the center of projection. The viewer can use the up/down arrow keys to move/rotate the center of projection up/down about the above stated point on the terrain. Also, the viewer can use the left/right arrow keys to move/rotate the center of projection left/right about the point on the terrain.

3.2 Genetic Algorithm Design

The algorithm adopted in this thesis is basically a genetic algorithm, but replacing the binary string representation with GP's tree structure representation. The main advantage of GP is that the dynamic sizes and shapes of the individuals in the population provide diversity.

3.2.1 Design a Representation.

Each individual is represented by a grow-able tree, composed with tree nodes. When growing from root of the tree, three different types of tree node can be randomly chosen – binary node, unary node, and leaf node. The tree root is forced to be a binary node. Each tree node has a value. A binary node has two pointers pointing to tree nodes. A unary node has one pointer, and leaf node have no pointers. The maximum height of trees is predefined. Each tree (individual) has a fitness value, which is initially set to be zero. Also, an individual has a fitness function to evaluate its fitness.

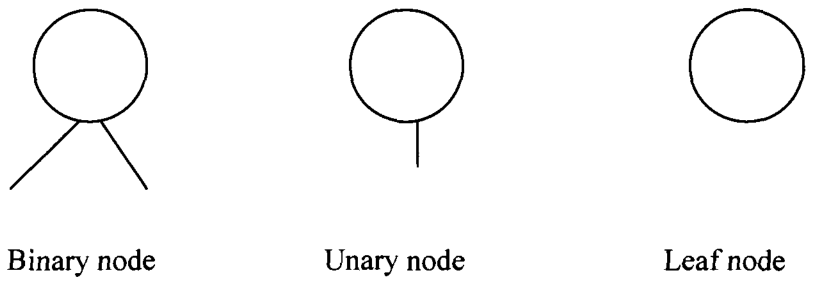

Figure 3.7 The types of tree nodes

3.2.2 Mapping a Genotype to a Phenotype

Once the tree is generated, the system traverses the tree, retrieves the value stored in each node, and pushes it into a list – the *height values list*. Every tree has a *height values list*. For the whole population, an array is created, each element of which is inserted into a *height values list*. Consequently, the system ends up with a 2-D array of height values for the whole population. In the graphics module, the system outputs the value in the array at [x][z] as the y coordinate of the point at (x, z) in 3-D coordinates. By connecting the points together through drawing primitive objects in the `GL_TRIANGLE_STRIP` mode provided by the OpenGL API, the system then forms a 3D triangle mesh. Finally, texture mapping is applied to construct the final terrain module.

The terrain is constructed from the entire population, with each individual representing an array of points – along either x-axis or z-axis direction – on the terrain surface. This is unusual because usually a single individual is selected as the 'best' solution to represent the terrain, so the approach used in this thesis is non-standard. For the first reason, in this thesis, the terrain should show viewers some sort of diversity in order to make it look like real and natural. For example, if the flat terrain shape is desired, the ideal terrain should look like flat for the most part, but spiky on very small area. The desired flat region is due to the individuals with good fitness; while the spiky area is caused by the individuals with bad fitness. As the population evolves, more and more individuals in each generation tend to have better fitness, leading to the desired parts of the terrain. On the other hand, the individuals having non-competitive fitness values become fewer and fewer. They, resulting in the less desired parts of the terrain, always exist but with less and less proportion. As another example, in a video game scene, when a user goes through the terrain, the system interacts with the user by constantly changing the terrain to be flat or spiky for the most part and changing the small area to be the opposite shape to make it look and feel real and natural. If the user goes towards the flat part, the system changes the coming terrain shape to be spiky with much probability, but with little probability of changing the coming terrain shape to be flat, to make it feel real.

For the second reason, this terrain system provides a user chances to evolve the terrain from one shape to another. For instance, the user can click the spiky part of the terrain to make the terrain shape to evolve from flat to spiky.

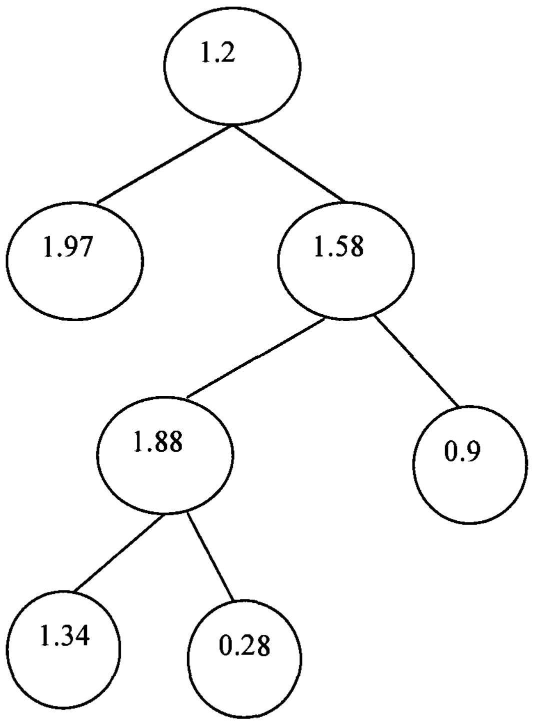

Figure 3.8 An example of an individual (tree structure)

The following table 3.1 Terrain height data map table shows the y coordinate of the points in each individual.

Table 3.1 Terrain height data map table

Individual-1	Individual-2	Individual-3	Individual-i	...	Individual-n
1.2	1.81	0.91			
1.97	1.50	1.34			
1.58	0.74	0.99			
1.88	0	1.81			
1.34	0	0.91			
0.28	0	1.34			
0.9	0	0.74			

All the points in one individual have the same x coordinate. For example, the x coordinates of the points in the individual-1 are 0. And, those in individual-2 are 1. In each individual, the z coordinates of the total m points (m stands for the number of points in the individual) are in the range of [0, m-1]. The above height values of Individual-1 can be extended into 3-D coordinates and formed to points (0, 1.2, 0), (0, 1.97, 1), (0, 1.58, 2), (0, 1.88, 3), (0, 1.34, 4), (0, 0.28, 5), (0, 0.9, 6)

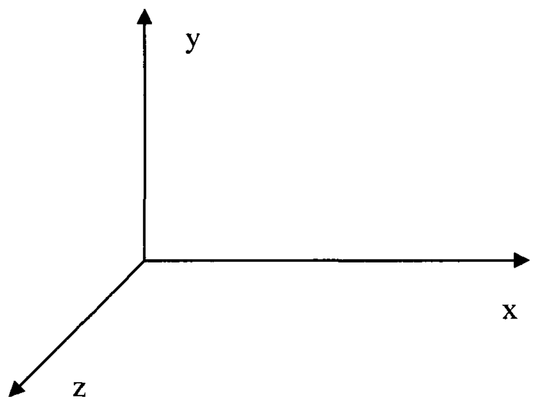

Figure 3.9 3D coordinates system used in this thesis

3.2.3 Evaluating an Individual – Fitness Function

In this system, a user uses mouse to click a point on the terrain. Then, the system maps the specified window coordinates into object coordinates. The GLU function gluUnProject is used to convert Windows screen coordinates to OpenGL coordinates (X, Y, Z). From X value and Z value, the system looks up this pair of values in the terrain map and finds the corresponding height value Y (3.2.2 Mapping a genotype to a phenotype).

The process of evaluating individuals is based on the height Y of the clicked point, which is passed to the fitness function. In a sense, the user is doing the "evaluation" by clicking. After the user has clicked, the fittest individuals are those nearest in average height. Hence, the fitness calculation is similar to standard deviation calculation on a discrete random variable or data set.

This calculation is described by the following formula:

$$s_N = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}.$$

In this formula, xi stands for each value (i varies from 1 to N), N means the number of values, and $\overline{\mathbf{X}}$ is the mean of N values. The goal is to calculate by how much these numbers differ from their mean.

However, in this thesis, we use the height value Y of the clicked point to replace $\overline{\mathbf{X}}$ in the above standard deviation formula to calculate how far points in each individual differ from the clicked height. The closer it is the better fitness it has.

Given a mouse click by the user, the overall evaluation proceeds as follows:

- Get height value Y from the user
- Use Y as the mean, compute S_N for each individual
- Choose individuals with small values of S_N as the "fittest"

3.2.4 Starting an Initial Population

An initial population starts with a group of initial individuals (tree representation) by keeping adding individuals into the population. An initial individual starts with a tree root. A tree root is force to be a binary node. All of the other nodes are randomly chosen from types of binary, unary, and leaf node. The tree keeps growing until reaching leaf node or tree height reaches the predefined limit.

Gaussian distribution

The value of each tree node, which will be used as height of terrain map, is created randomly and distributed with Gaussian distribution. A set of such variables are defined by two parameters *location* and *scale*: the mean ("average", μ) and variance (standard deviation squared, σ^2) respectively. Given a mean value M and a standard deviation value S, the system generates a set of random variables, the mean of which yields M, and the standard deviation value of which yields S. In this thesis, the mean value M and standard deviation value S are predefined before the program runs, and the standard deviation value S will be changed when the population evolves to the next generation. The new standard deviation value S is re-calculated based on the height value Y of clicked point and mean M. The further Y is away from M, the greater the new standard deviation value S will be.

3.2.5 Selection Mechanism

Once the system evaluates each individual's fitness value, it sorts the population by each individual's fitness value. According to statistical principal, most of high fitness individuals are paired to crossover; some in the middle are kept to next generation. And, 1% of population will be mutated.

3.2.6 Crossover

When a pair of individuals selected from front part of the fitness-sorted list perform crossover, trees A, B swap their sub-trees and turn to be A', B'. Some of individuals' properties, for example, fitness values, will be re-calculated before the evolution to the next generation.

3.2.7 Mutation

A selected individual's random chosen sub-tree is removed, and a new type of random sub-tree grows there with predefined height. The point, the root of removed sub-tree, is called a *mutation point*. First, a new Node, randomly chosen from types of

Binary-Node, Unary-Node, and Leaf-Node, is created as the root of new sub-tree at the mutation point. Then, a sub-tree grows from this root until desired height is reached or leaf-Node is created. When creating a new tree node, the standard deviation value S is changed, and the changed S will affect the random value of each new tree node (3.2.4).

3.2.8 Population Evolves to Next Generation

The basic program flow of evolving one generation is as follows. Firstly, from the entire population, the system constructs terrain, where the midpoint displacement algorithm is applied (see 3.1.1). Secondly, the user clicks on the terrain surface with the mouse, the system passes the height value Y of the clicked point as a parameter to the population. Thirdly, system evaluates the fitness for each individual, and sorts all individuals by fitness values. Lastly, it selects individuals to perform crossover and mutation, resulting in a new generation.

When user clicks a point not on the terrain surface, the operation is discarded. The standard deviation value S is changed according to the height value Y of the clicked point and mean M (3.2.4). This will affect crossover and mutation operations.

3.2.9 Termination Criteria

The system terminates if it meets the user's expectation, and the user ends it. If the user keeps clicking spiky part of the terrain, the dynamic terrain system self-adapts to be spiky for most parts of itself. The system detects that the clicked part is spiky by evaluating how each point's height value far away (or close to) from the height value of clicked point. The further they are, the spikier the shape is. The math calculation is standard deviation.

3.3 User Interaction

Before the mouse click, from the entire population, the system constructs the terrain, where the midpoint displacement algorithm is applied for more detailed effect (see 3.1.1). At this point, the terrain is constructed from the population and is rendered to a fractal. Then, a user clicks on the terrain surface with the mouse's left button; the system calculates the height value Y of the clicked point and passes it as a parameter to the GA module population class. When the user clicks a point not on the terrain surface, that operation is discarded. The user needs to clicks a point on the surface of the terrain to perform a valid operation. Thirdly, the system evaluates the fitness value for each individual and sorts all individuals by fitness values. The standard deviation value S is changed according to the height value Y of the clicked point and the mean M, and will affect the crossover and the mutation operations. In this system, the new value S is set to $|Y\text{-}M|$. Lastly, the system selects individuals to perform crossover and mutation according to their fitness with probabilities proportional to their relative fitness, which ensures that the expected number of times an individual is chosen is approximately proportional to its relative performance in the population. Until now, the system has resulted in a new generation. Again, from the entire population of the new generation, the system constructs a new terrain. A new mouse click event by the user will drive a new cycle of evolution.

4. IMPLEMENTATION

In this thesis, we describe a dynamic terrain system implemented with two main modules – the GA Engine module and the Graphics terrain module. The GA Engine is where the GA is applied and performed. Also, it drives the dynamic transformation of the terrain. The graphics module represents the terrain and provides parameter feedback to the GA Engine module as well. In this chapter, we describe the implementation of the two modules and their interaction as well.

4.1 Graphic Terrain Module Implementation

In the Graphic Terrain module, the height values are plotted into the scene to form a terrain, and the mid-point displacement algorithm is applied to generate a fractal terrain for more detailed effect. Then, when a mouse click event happens, the Windows screen coordinates of the point clicked on the terrain surface are converted to OpenGL coordinates. The height value Y of the clicked point is sent to the GA Engine module as a parameter for the fitness evaluation. After the population evolves to the next generation, the updated data of height values are plotted into the scene so that the terrain transforms to a new shape.

4.1.1 Height Data Grid Forms 3D Terrain

The function `setMap(vector<vector<nodetype> > popuList)` is used to form a map, which is a two-dimensional vector, as a container to hold the height values retrieved with the Population method getPopuList(). During this progress, since different lists of height values of individuals may have different lengths (due to diverse shapes and sizes of trees), every list will be extended with zeros until its length is the same as the longest list in the whole population.

Also, the fractal procedure is performed here to make the terrain look more detailed, smoother, and finer, which is discussed in the next section. Then, the function `drawTerrain()` is used to draw the terrain in the `GL_TRIANGLE_STRIP` mode to form a 3D polygon mesh.

4.1.2 Fractal and Mid-point Displacement

The Mid-point displacement algorithm is applied to make a fractal terrain more detailed, smoother, and finer. Hence, even though fractals are not the key point of this thesis, they still play an important role for the final effect of the terrain. In this implementation, the algorithm calculates the mid-point value of two adjacent points in the X direction and adds an amount to the mid-point value with a calculated number based on a Gaussian distributed random number. And, the algorithm goes through the points in the Z direction to calculate the values the same way. The loop for these X, Z direction procedures continues until the desired effect is reached.

The code below generates the new value at mid-point between x and y. The new value is the addition of a random amount and the average value of x, y.

```
double temp = (x+y)*0.5;
temp += abs(y-x)*box_muller(0.2,0.125);
```

4.1.3 OpenGL 3-D Position Conversion

In this system, a user uses the mouse's left button to click a point on the terrain surface. Then, the function gluUnProject() is the essential function used to convert Windows screen coordinates to OpenGL coordinates. The function glGetDoublev() is used to retrieve the ModelView Matrix, which determines how the vertices of OpenGL primitives are transformed to eye coordinates. The function glGetDoublev() is used again to retrieve the Projection Matrix, which transforms vertices in eye coordinates to clip coordinates. The function glGetIntegerv() is used to retrieve the Viewport Values: the starting X and Y positions of the GL viewport along with the viewport width and height. Since Windows coordinates start with (0, 0) being at the top left, whereas OpenGL coordinates start at the lower left, winY = (float)viewport[3] - winY converts winY to OpenGL coordinates. The function glReadPixels() gets the z coordinate. The function gluUnProject() is used to convert Windows screen coordinates to OpenGL coordinates (X, Y, Z).

```
vector<GLdouble> GetOGLPos(int x, int y)
{
   //...other code omitted
   glGetDoublev( GL_MODELVIEW_MATRIX, modelview );
   glGetDoublev( GL_PROJECTION_MATRIX, projection );
   glGetIntegerv( GL_VIEWPORT, viewport );
   winX = (float)x;
   winY = (float)viewport[3] - (float)y;
   glReadPixels( x, int(winY), 1, 1, GL_DEPTH_COMPONENT, GL_FLOAT,
                 &winZ );
   gluUnProject( winX, winY, winZ, modelview, projection, viewport,
                 &posX, &posY, &posZ);
   //...other code omitted
}[27]
```

Given the OpenGL X and Y coordinates, and because the terrain coordinates are known, the system looks up this pair of values in the terrain map and finds the corresponding height value Y of the point clicked on terrain surface (4.3.2 Mapping a genotype to a phenotype). It will be passed to the GA module.

4.1.4 Viewing the Terrain

In order to view the terrain from all angles, a polar camera is implemented in this system. The array `viewer[3]` is used to keep track of the location of the camera. The array `center[3]` is used to keep track of the look at position of the camera. And, the array `up[3]` is used to keep track of the up Vector of the camera. Three variables are used to keep track of the angles and radius of the camera and are adjusted respectively when the control keys are pressed. Also, these values simultaneously update three vectors – *viewer[], center[], and up[].* Since the camera's up direction is always same as y axis, *up[1]* is always 1.0, *up[0]* and *up[2]* are always 0.0. Finally, the up-to-date vectors are used as parameters in function *gluLookAt(viewer[0],viewer[1],viewer[2], center[0], center[1], center[2], up[0], up[1], up[2]).*

4.2 GA Engine Module Implementation

In the GA Engine module, three main classes are designed and implemented. They are Tree Node family classes, Tree (individual) class, and Population class. A tree is formed through Tree Node growth. Node growth is a process where the pointers of a Tree Node (usually Binary Tree for the first time as root) are recursively assigned new Tree Nodes. First of all, a Binary Node is instantiated as the root of a tree. The Binary Node class has two pointers to Node – *left and *right. Then, two Nodes, randomly chosen from types Binary-Node, Unary-Node, and Leaf-Node, are created and assigned, one to *left and one to *right. This growth process will continue until a pre-defined tree height is reached or a leaf-Node is created. Given a tree, a Population class adds the pointer to the tree into its list. As shown in the hierarchy diagram below, a population is composed of one or more trees, stored in its data member `trees`. A tree may be composed of several sub-trees (recursive composition). A tree is composed of one or more Nodes. There are three types of Nodes: Binary Node, Unary Node and Leaf Node. They are all derived from abstract base class Node. The Binary Node class has two pointers to the Node, *left and *right. The Unary Node has one pointer to the Node, *next. The Leaf Node is the terminal Node and does not have a pointer.

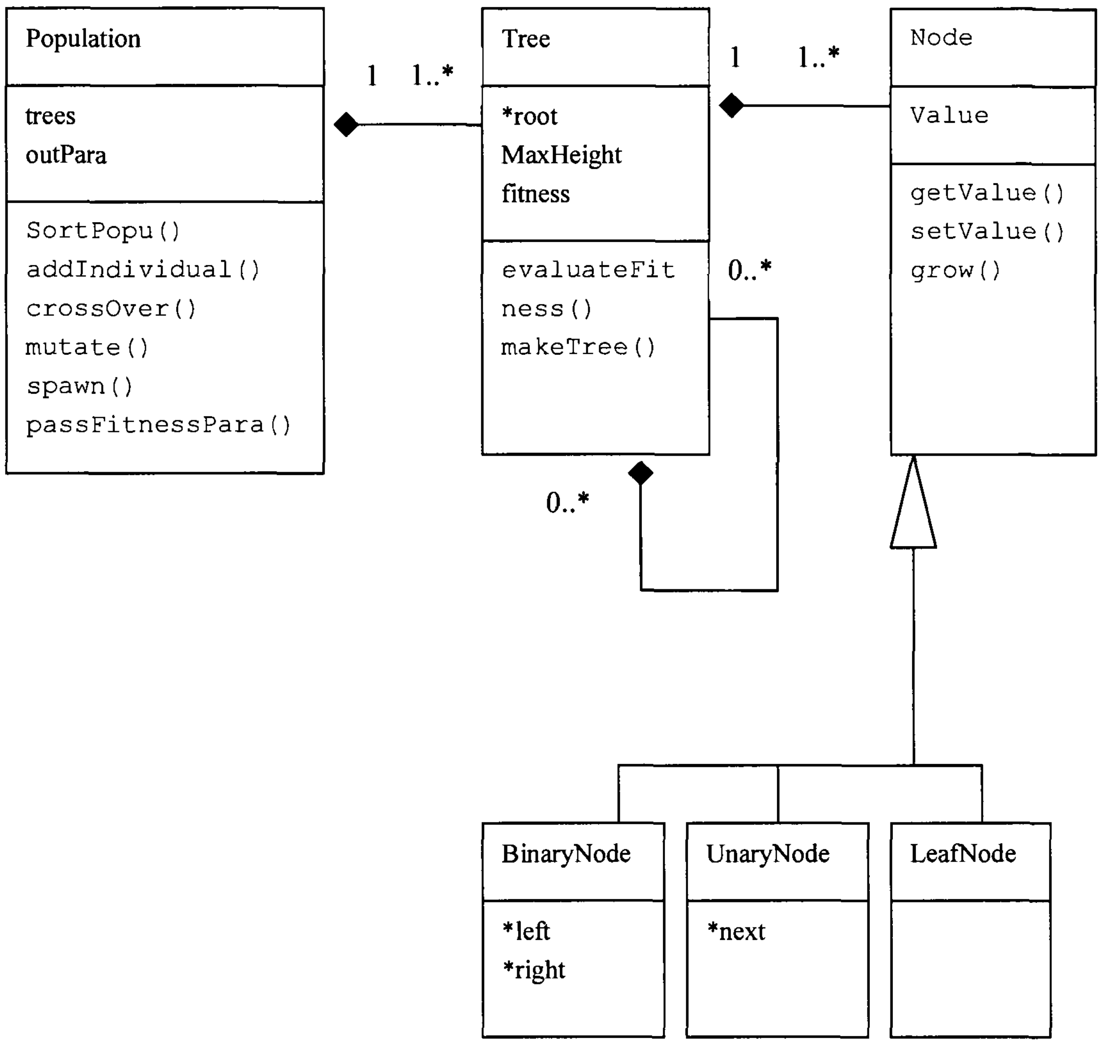

Figure 4.1 The GA engine module hierarchy diagram

Table 4.1 Multiplicity Indicators

Indicator	Meaning
1	One only
0..*	Zero or more
1..*	One or more

4.2.1 Implement Tree Node Family of Classes

In addition to the discussion of Tree Nodes above, the Tree Node class has a variable representing the height of a point on the terrain surface. When traversing each Node, the tree gets the height value stored in each Node and ends up with a list of height values for the tree to be used later for terrain construction.

Since the representation is a tree structure, the implementation of the tree should make the tree shape diverse so that the representation is diverse at run time. The dynamic sizes and shapes of the trees (individuals) provide diversity, which prevents a population of solutions for a problem from being converged too early, resulting in such a situation that the parental solutions are not able to generate offspring that are superior to their parents. Hence, a tree is constructed with Nodes randomly chosen from three types of Node at run time. Inheritance polymorphism is implemented, which provides the advantage that without knowing which derived classes the common methods belong to at compile time, the desired behavior is performed for a specific derived class at run-time. This is a specific advantage for this research.

The method Node::grow() is used to provide the prototype of the growth behavior and is re-defined in derived classes.

```
class Node
{
public:
   virtual void grow(int heigh) = 0;
   // ... other methods omitted
private:
};
```

The re-defined method in each derived class has its own meaning and handles the case of different Node types.

```
class BinaryNode: public Node
{
public:
   void grow(int heigh);
   // ... other methods omitted
private:
   Node *left;
   Node *right;
};

class UnaryNode: public Node
{
public:
   void grow(int heigh);
   // ... other methods omitted
private:
   Node *next;
};

class LeafNode: public Node
{
public:
   void grow(int heigh)
   {std::cout<<"leaf, cannot grow...\n";return;}
   // ... other methods omitted
};
```

4.2.2 Implement Tree (Individual)

The tree constructor is used to build the tree. As stated in section 5.1, a Binary Node is originally instantiated as the root of the tree. Then, the Tree Node growth process will continue until the pre-defined tree height (limited by data member MaxHeight) is reached or a leaf-Node is created. In this process, the re-defined method grow() in each derived class is used to instantiate a new Node. Figure 5.2 below shows an example of a tree, with actual numbers in it.

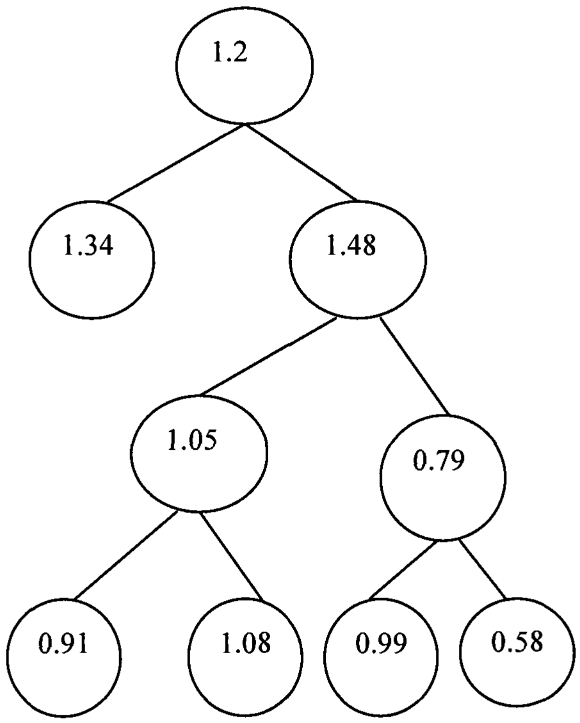

Figure 4.2 an example of a tree with actual numbers

In addition to a list of height values to be used later as one tree's contribution to the whole terrain construction, a tree stores a pointer to the root of the tree, called *root. Moreover, a tree has a `double` variable, called `fitness`, to store the fitness value of the tree. An `int` variable called MaxHeight is also stored in a tree to control the maximum height of the tree.

Because one tree structure has all the information for an individual in the whole population, as the representation of one individual, it represents only part of a terrain. The whole population represents the whole terrain.

The fitness function is a critical and key point in the implementation. The method Tree::evaluateFitness() is used to evaluate each individual's fitness. The process of evaluating individuals is based on the height value Y of the clicked point, which is passed to the fitness function. The calculation method calculates how close the heights of the points in each individual are to the height Y of the point clicked. This is similar to the standard deviation calculation on a discrete random variable or data set. According to the design we adopted, the closer the points' height values in an individual are to the Y value, the better fitness the individual has. On the other hand, if the height values of points in one individual are far away from the height value Y of the clicked point, this individual has worse fitness. As a result, when evaluating each individual through the method *evaluateFitness()*, the mean value in the standard deviation formula is replaced with the height value Y of the clicked point.

```
class Tree
{
public:
   Tree(int MaxHeight = 1, BinaryNode* root = new
           BinaryNode());
   int evaluateFitness();
   // ... other methods omitted
private:
   BinaryNode *root;
   int MaxHeight;
   double fitness;
};
```

4.2.3 Implement Population

The Population class has a data member to store a list of pointers to Tree. When a new Population is instantiated, the method `Population::addIndividual()` is used to add Tree pointers to its list. The method `Population::passFitnessPara()` is used to pass the height value Y of the clicked point to the Population for further fitness calculation. The class has a function `SortPopu()` to sort the tree list according to their fitness values. The method `Population::spawn()` is used to assign new standard deviation value and call `crossover()` and `mutate()`.

In the following equation, ::standDevia = abs(outerPara-::mean), the value of standDevia will be used to calculate the next set of random variables.

The new standard deviation value S is decided by the height value Y of the clicked point and mean M. The greater the difference between Y and M, the more separate from M the height values of new points are. As a result, the new S value is calculated based on the height value Y of the clicked point and mean M and set by the absolute value of difference between Y and M. As a result, the greater the difference between Y and M, the greater the value S is. Consequently, the discrete random numbers to be generated are more separate from the mean M. The figure below shows how random numbers are distributed with different mean and standard deviation.

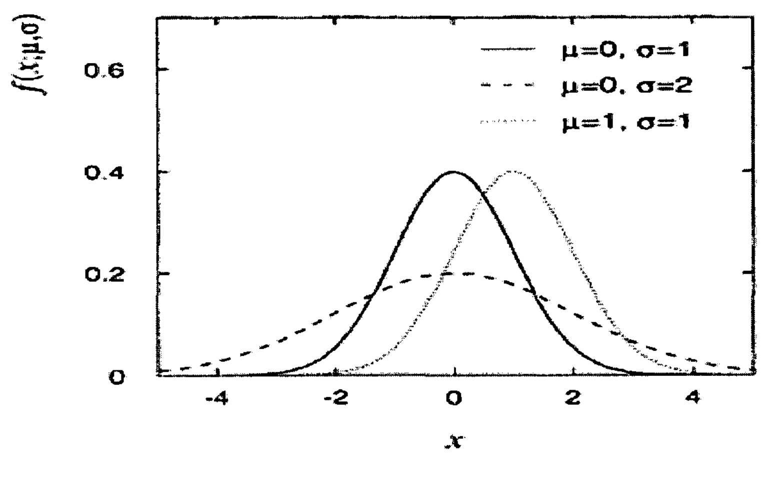

Figure 4.3 The Gaussian probability density for various values of the parameters μ and σ [28]

After the list of individuals is sorted, the method Population::crossOver() is used to perform crossover between two individuals (trees) with good fitness by swapping sub-Trees of them. The method Population::mutate() is used to randomly pick a sub-Tree in it, destroy the whole sub-Tree, and create a new sub-Tree with totally random tree Nodes. As a result, the shape of the new sub-Tree is different from the original deleted one. The maximum height of a new sub-Tree is set the same as the original height of the sub-Tree for this thesis implementation, but the actual resulting height of the tree is not guaranteed since the type of Nodes is randomly decided, and it might be Leaf-Node, resulting in the growth termination. The new sub-Tree grows with Node values generated with a new standard deviation value for random number generation.

```
class Population
{
public:
   void SortPopu();
   void addIndividual(Tree *);
   void crossOver(Tree* pA, Tree* pB);
   void mutate(Tree* pT);
   void spawn(double birthrate = 1, double crossover = 0.6,
               double mutate = 0.1 );
   void passFitnessPara(double clickHeight, int count);
   // ... other methods omitted
private:
   TreeList trees;
   double outerPara;
};
```

4.2.4 Gaussian Random Number Generation

Gaussian distribution random number generator is implemented in this thesis to generate random numbers other than uniformly distributed random numbers. The numbers are used for Tree Node values and when applying the mid-point displacement algorithm (implementation discussion later) as well.

The pseudo-random number generation functions available in most standard math libraries generate random numbers with a uniform distribution. If many "random" numbers are drawn from a particular region (say [0,1), where the value v is in the range 0 <= v < 1) and a histogram plot is generated from the result, the histogram will form a block, which is why it is called flat, or uniformly distributed random numbers. A number of applications, including the generation of Brownian random walks, require random numbers that fall into a Gaussian distribution (e.g., a bell curve). An example of 100,000 random numbers by throwing six dice in an approximate Gaussian distribution is shown below:

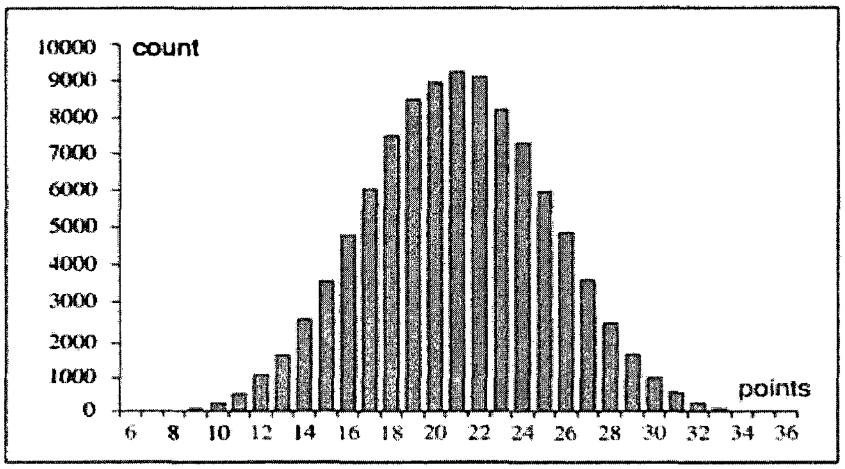

Figure 4.4 100,000 random numbers in a Gaussian distribution [25]

The implementation of converting a flat distribution to a Gaussian distribution is described as follows.

In *Chaos and Fractals,* Peitgen *et al.*, propose a simple equation for converting random numbers with a uniform distribution into random numbers with a Gaussian distribution. They write [25]:

> "In fact, the Gaussian distribution arises in all cases where independent and similar (i.e. identically distributed) random events are summed up or averaged. This is the context of an important mathematical theorem called the *central limit theorem.*"

An example of a "random event" would be a throw of six dice having six sides each. The values on the dice are added, yielding a value from 6 to 36. If 10,000 throws are made, the distribution of values will fall in a Gaussian curve. The principle of adding multiple random events is used by Peitgen *et al.* to convert the values produced by a uniform random number generator into an approximation of a Gaussian distribution using the formula below:

$$D = \frac{1}{A}\sqrt{\frac{12}{n}}\left(\sum_{i=1}^{n} Y_i\right) - \sqrt{3n}$$

- D: a Gaussian random number
- A: the upper limit of the random number generator, which returns numbers 0, 1, ... A
- n: number of independent events (e.g., dice)
- Y_1, Y_2, ... Y_n: results of an independent event (e.g., a dice throw) [25]

Carter has given a more rigorous method for generating Gaussian random numbers. Carter's [29] implementation is quoted below with a modification to avoid division by zero.

```
/* normal random variate generator */
```

```
float box_muller(float m, float s)
{                                              /* mean m, standard deviation s */
        //...other code omitted
                do {
                        x1 = 2.0 * ranf() - 1.0;
                        x2 = 2.0 * ranf() - 1.0;
                        w = x1 * x1 + x2 * x2;
                } while ( w >= 1.0 || w==0.0);
                w = sqrt( (-2.0 * log( w ) ) / w );
                y1 = x1 * w;
                y2 = x2 * w;
                use_last = 1;
        //...other code omitted
}
```
[29]

The while loop condition `|| w==0.0` is added to avoid division by zero.

4.3 GA Engine and Graphics Interaction

The interaction between the Graphics module and the GA Engine module is driven by a mouse click event as described above. The GA interacts with the dynamic terrain shape in two senses. Firstly, this terrain system provides an adaptive feedback environment. When a user clicks the terrain surface, the system interacts with the user by constantly changing the terrain to be flat or spiky. For example, if a user clicks the flat part, the system changes the next frame of the terrain image to be more flat, with a little part the opposite shape - spiky - to make the terrain look and feel more real and natural. Secondly, evolving from one generation to the next, this terrain system provides users chances to evolve the terrain from one shape to another, as the whole population constructs the terrain, and mutation creates diversity. For instance, given a terrain that is flat for the most part and spiky on a small area, a user can click the spiky part of the terrain to make the terrain evolve from flat to spiky one.

When a mouse's left button click event happens, the function `GetOGLPos()` returns OpenGL coordinates and then passes them to the function `getHeight`(). The height value is then calculated and passed to the population class method `passFitnessPara()` to evaluate the fitness of each individual. Method `spawn()` is then used to evolve the population to the to the next.

```
if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
{
    vector<GLdouble> v = GetOGLPos(x,y);
    double heightPara = getHeight(v.at(0),v.at(1),v.at(2));
    if (heightPara != -999.0)
    {
        pP->passFitnessPara(heightPara,getMapTreeSize());
        pP->spawn();
        setMap(pP->getPopuList());
    }
}
```

5. RESULTS

5.1 Algorithm

In this thesis, we have investigated how the EC applies to the terrain modeling. A specific evolutionary algorithm, with the combination of the GA and the GP, has been designed. The algorithm is hybrid because it is based on the GA, but it uses the tree structure representation traditionally applied in the GP. Also, the entire population of individuals is used to represent the terrain. Lastly, we implemented an adaptive dynamic terrain system based on the algorithm designed. In this thesis, we have demonstrated the first use of the GC to dynamic terrain modeling and found that it is feasible for the EC to be applied to the terrain modeling.

5.2 Terrain Model

The figures below show the initial terrain model and the resulting model after 16 rounds of evolutions. During the evolution, the user clicked the mouse on flat areas. The resulting terrain shape looks flat and smooth compared with the initial terrain shape. The user does not obtain an ideal result every time. In this case, the user has to click the terrain more times and drive more rounds of evolution.

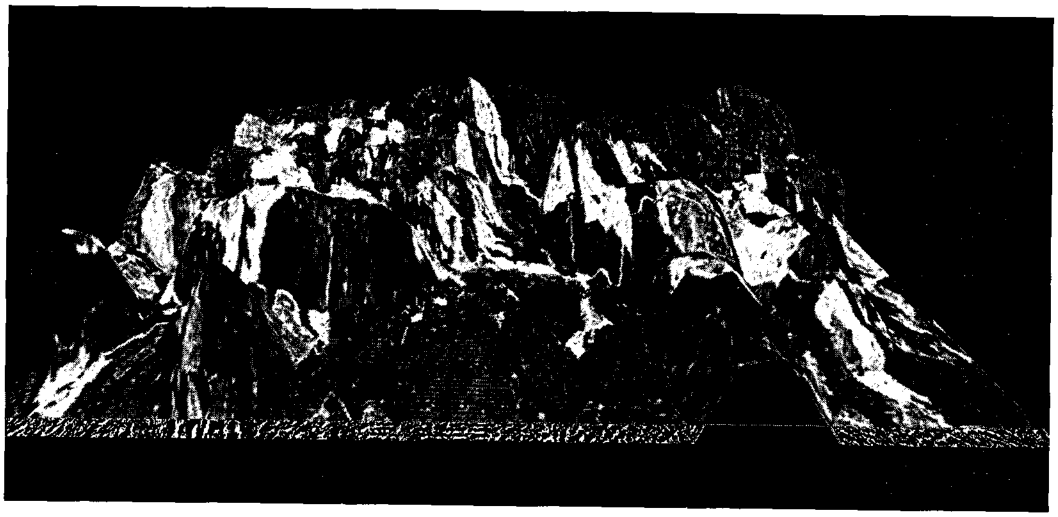
Figure 5.1 the initial terrain model

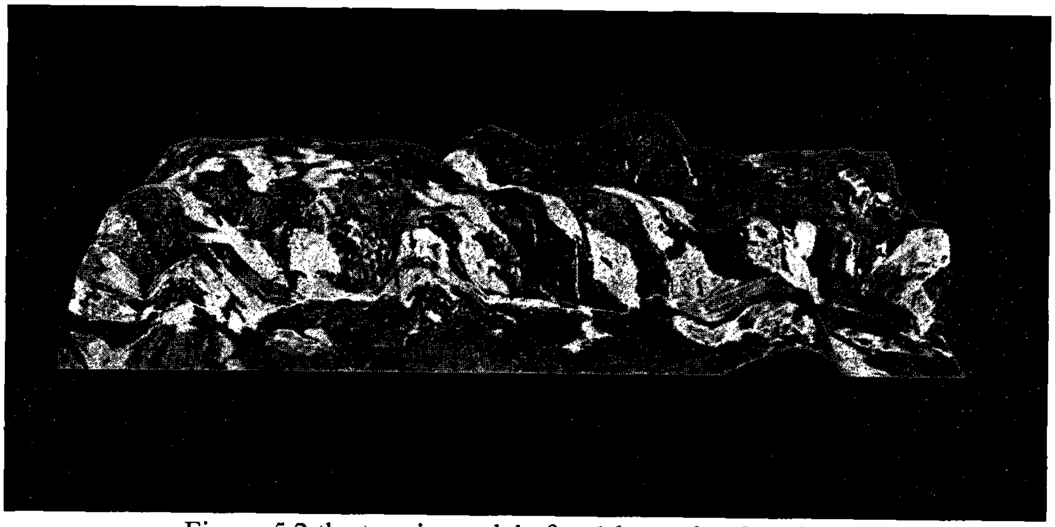
Figure 5.2 the terrain model after 16 rounds of evolution

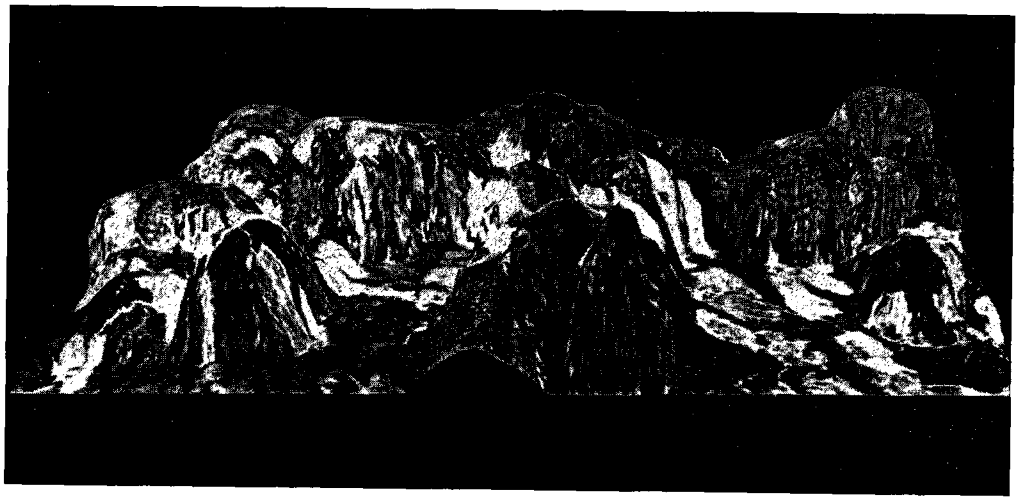
Figure 5.3 the terrain model after 18 rounds of evolution

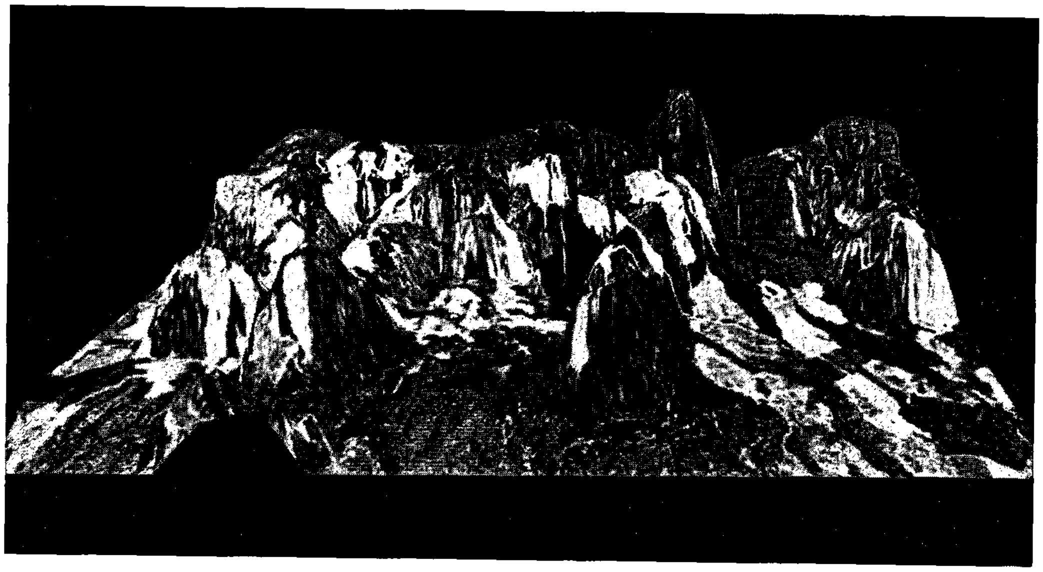
Figure 5.4 the terrain model after 30 rounds of evolution

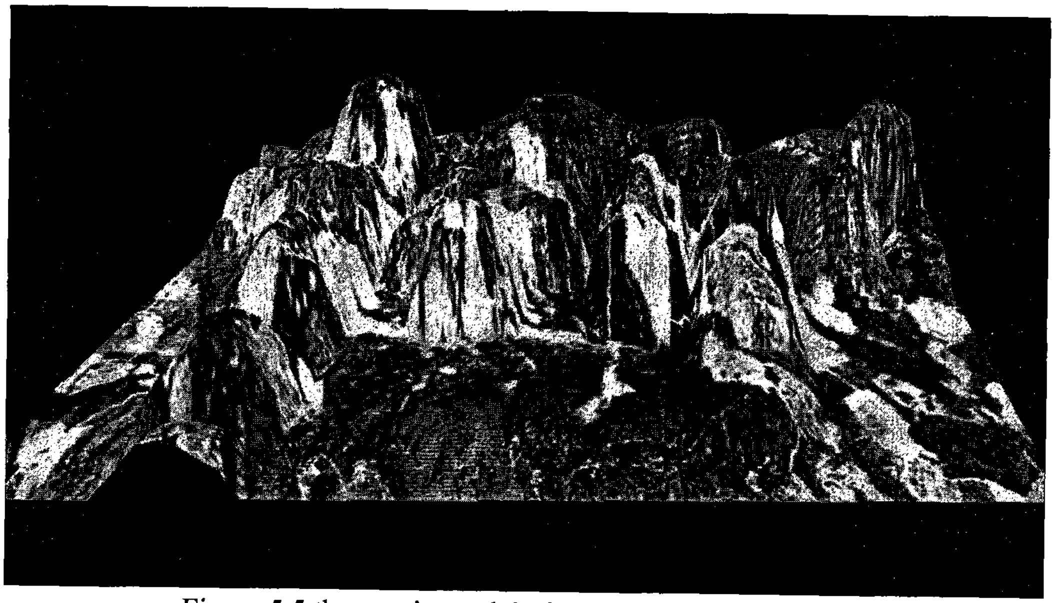
Figure 5.5 the terrain model after 36 rounds of evolution

The following group of images illustrates the terrain constructed with different criteria.

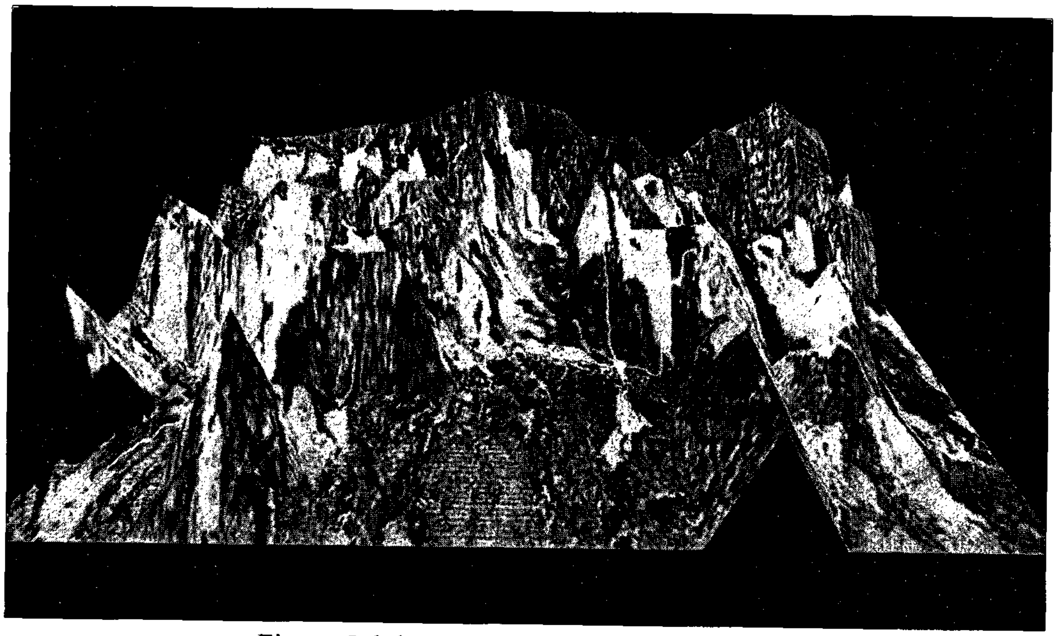
Figure 5.6 the terrain without fractal – rough

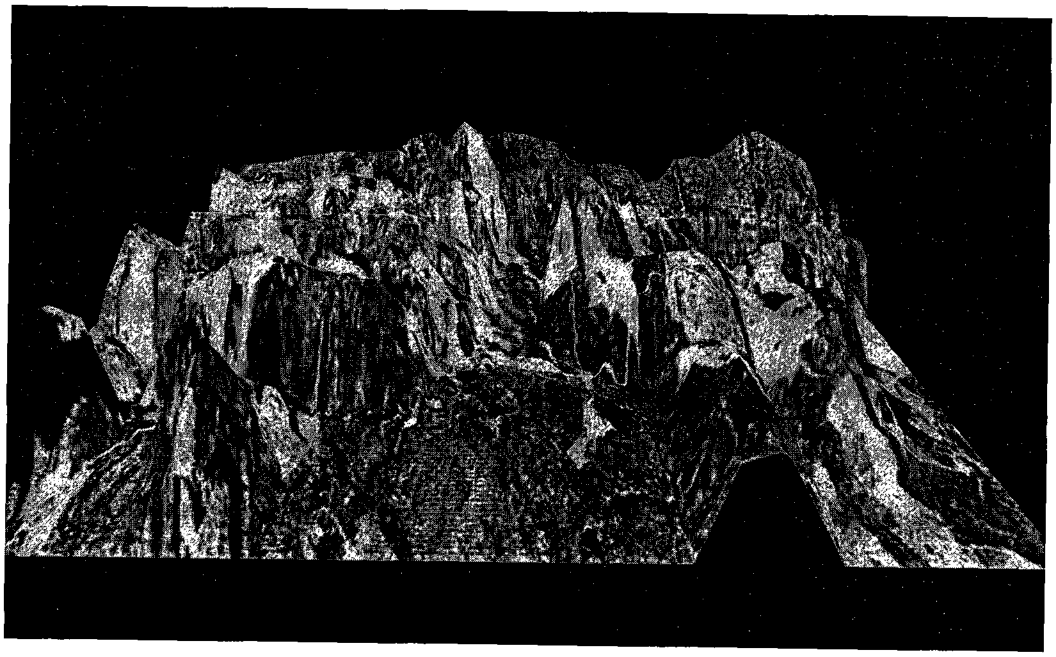
Figure 5.7 the terrain with 2 rounds of fractals – smooth

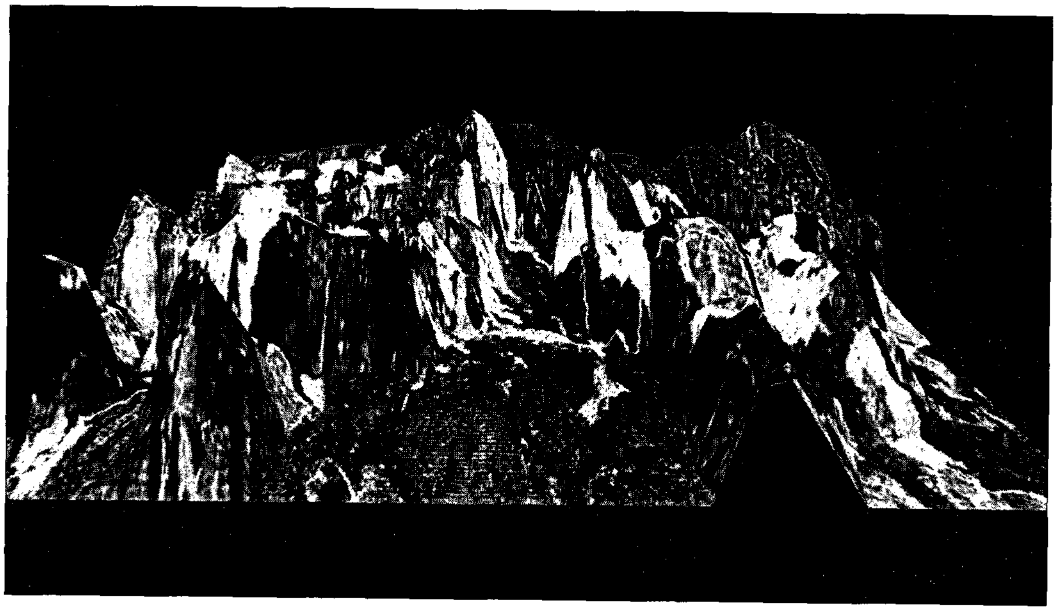

Figure 5.8 the terrain with 3 rounds of fractals – more smooth

In order to create distinct initial terrain shape every time the application is run, the wall-clock time is used to seed the random number generator. The following images show different initial terrain shapes at different times.

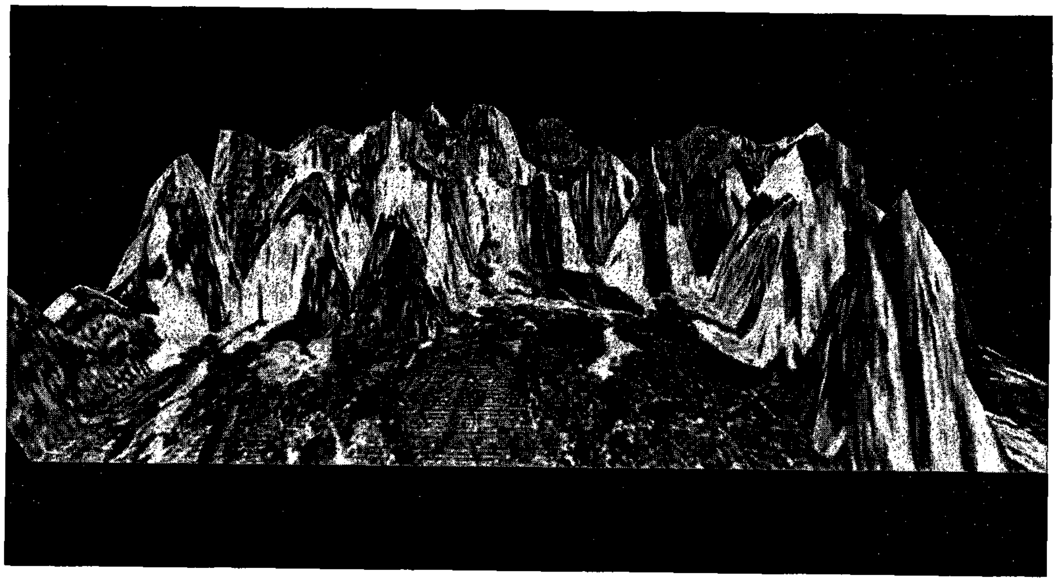

Figure 5.9 the initial terrain model at different time of run

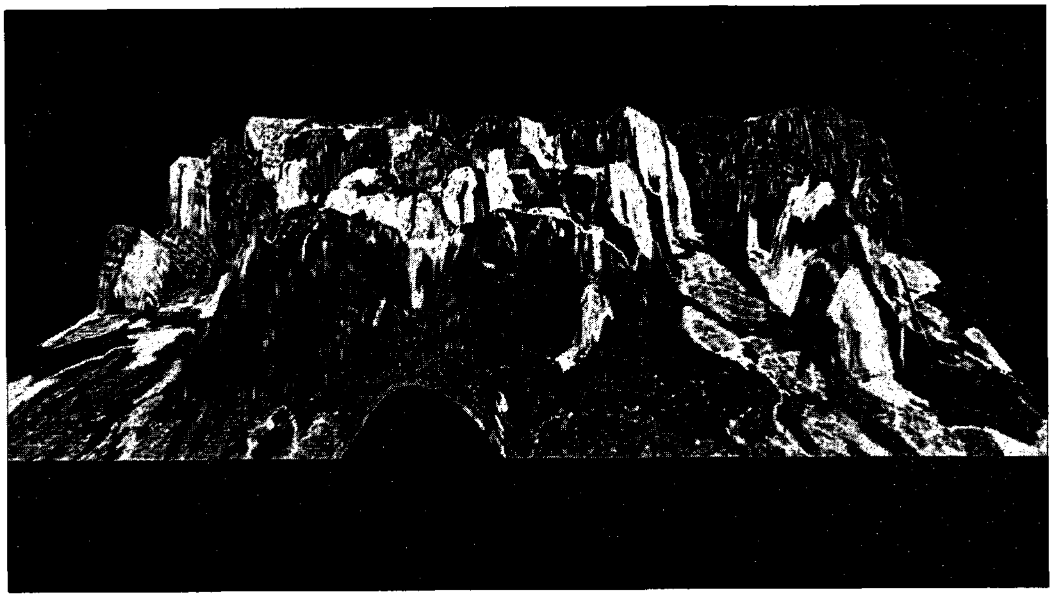

Figure 5.10 the initial terrain model at different time of run

6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The idea stated in Chapter 1 has been answered as shown in chapters 3 through 5. We have designed a specific hybrid evolutionary algorithm, which is based on the GA, but replaces the representation with the tree structure representation traditionally applied in the GP. Also, the algorithm is unusual in that the entire population of individuals is used to represent the terrain. Based on the algorithm we have designed, an adaptive dynamic terrain system has been implemented. Our contribution is to combine two existing techniques: evolutionary computation and terrain modeling. This first use of the EC's application to terrain modeling has found that it is feasible to apply the EC to the terrain modeling.

There is a distinguishing difference between natural evolution and evolutionary computation. Natural evolution proceeds with no explicit goal, but the EC, where the fitness measure (i.e. fitness function) has been predefined, aims to search for individuals with higher or highest possible fitness values. Hence, the fitness function plays a key role in EC and decides the soundness of the specific evolutionary algorithm.

6.1.1 Strengths

The evolutionary algorithms have certain advantages. Some computational problems require complex solutions that are difficult to program by hand. With the EC, there might be no need for humans to care about the details, and detailed aspects of the problem may even be unknown. The EC is adaptable to the changing environment. The EC generated systems are adaptable, creative, and able to deal with complexity. Even though evolutionary algorithms are not guaranteed to find the global optimum, they can find an acceptable solution relatively quickly in a wide range of problems – they are robust. The evolutionary algorithms are promising methods for solving difficult technological problems, for machine learning, and for simulating natural systems in a wide variety of scientific fields. Models created with EC often perform better than precise simulations attempting to match real-world data. The purposes of these ideal models are to make ideas precise and to test their plausibility by implementing them as computer programs, to understand and predict general tendencies of natural systems, and to see how these tendencies are affected by changes in details of the model. These models allow scientists to perform experiments that would not be possible in the real world, and to simulate phenomena that are difficult or impossible to capture and analyze in a set of equations.

6.1.2 Weaknesses

The evolutionary algorithms also have some limitations. They are computationally intensive and usually slower than traditional method. Fitness function plays a key role in the algorithm and needs to be very suitable for the algorithm. The evolutionary algorithm does not guarantee success all the time. It is a stochastic system, and a genetic pool may be too far from the solution, or for example, a too fast convergence may terminate the process of evolution. Hence, it needs to perform a sufficient number of independent runs and/or use statistical measures (e.g. standard deviation).

6.2 Future Work

This thesis can be improved and extended in many aspects.

The implementation of the system can be extended by adding more controls on the rates of birth, crossover, and mutation. The result of this would be to make the system more robust, efficient, and flexible.

The implementation can be optimized by taking the consideration that the algorithm controls the size of individuals when the population evolves. This needs more work to verify the correctness because the diverse sizes of individuals provide diversity, which prevents a population for a problem being converged too early, resulting in such a situation that the parental solutions are not able to generate offspring that are superior to their parents.

To extend this algorithm, the further work may adopt a multiple-population algorithm as an extension. The EC is about comparison, so more competition is supposed to produce a better result. We might design and implement multiple populations with parallel computation. Each population will be applied the same evaluation function

The fitness function might have space to improve so that the system can be more adaptive. For example, we may divide the evaluation process into three steps: pre-evaluation, evaluation, and post-evaluation.

The future work may include the idea that changes how each individual's position in the 3D coordinates. For example, the position for each individual may be non-lineal.

To make the system more compatible to a terrain model designer, we can add an option that a mouse click event just changes one part of terrain intensively while others remain the same or have minor change.

The future work may include enrich/improve UI for more options. Firstly, we may add a GUI and add the options that are important factors to the terrain shape. Also, we need to investigate how the terrain factors to be controlled. Then, we modify the user interface and add more options based on the requirement of professional users in order to fulfill their requirement.

The final version of the system may be developed to a multi-purpose dynamic terrain modeling system.

References

[1] Melanie Mitchell, *An Introduction to Genetic Algorithms*, Cambridge, Mass.:MIT Press, c1996.

[2] Peter Shirley, *Fundamentals of Computer Graphics,* A K Peters, Ltd. Natick, MA, 2002.

[3] Donald Hearn, M. Pauline Baker, *Computer Graphics with OpenGL*, Upper Saddle River, NJ : Pearson Prentice Hall, c2004, p. 124, p. 490-493.

[4] Edward Angel, *Interactive Computer Graphics*, Boston: Pearson/Addison-Wesley, c2006, p. 262-264.

[5] Benoit B. Mandelbrot, *The Fractal Geometry of Nature.* W.H. Freeman and Company, San Francisco, 1982.

[6] Y. Fisher. *Fractal Image Compression: Theory and Application*, Springer Verlag, New York, USA, 1995, p. 26.

[7] António Miguel de Campos http://en.wikipedia.org/wiki/Image:Animated_fractal_mountain.gif, (current Nov 28, 2008)

[8] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steven Worley, *Texturing and Modeling,* Amsterdam; Boston: Academic Press, 2003, p.280-281, p. 435, p. 572.

[9] Ivo Marák, "On Synthetic Terrain Erosion Modeling:A Survey", *Proceedings of CESCG (Central European Seminar on Computer Graphics),* Austria,1997.

[10] Reuven Y. Rubinstein, Dirk P. Kroese, *Simulation and the Monte Carlo method,* Hoboken, N.J. : John Wiley & Sons, c2008, p. 60.

[11] F. K. Musgrave C. E. Kolb, "The synthesis and rendering of eroded fractal terrains," *International Conference on Computer Graphics and Interactive Techniques Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, 1989.

[12] S. N. Sivanandam, S. N. Deepa, *Introduction to Genetic Algorithms,* Berlin ; New York : Springer, 2007, p. 1-3, p. 15-16, p. 131-154.

[13] Karl Sims, "Artificial Evolution for Computer Graphics." *Proceedings of the 18th Annual International Conference on Computer graphics and Interactive Techniques*, 1991.

[14] Karl Sims, "Evolving 3D Morphology and Behavior by competition," *Artificial Life IV Proceedings*, ed.by Brooks & Maes, MIT Press, 1994.

[15] David Andre and Astro Teller,. "*Evolving team Darwin United.*" In RoboCup-98: Robot Soccer World Cup II, Minoru Asada and Hiroaki Kitano (eds). Lecture Notes in Computer Science, Springer-Verlag, vol.1604, 1999, p. 346-352.

[16] A.E. Eiben and J.E. Smith, *Introduction to Evolutionary Computing*, Springer, Natural Computing Series 1st edition, 2003.

[17] Dirk Weismann, Ulrich Hammel, and Thomas Bäck, "Robust design of multilayer optical coatings by means of evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol.2, no.4, November 1998, p.162-167.

[18] Randy Haupt and Sue Ellen Haupt. *Practical Genetic Algorithms*, John Wiley & Sons, 1998

[19] John Koza, Forest Bennett, David Andre and Martin Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers, 1999.

[20] Mateen Rizki, Michael Zmuda and Louis Tamburino, "Evolving pattern recognition systems," *IEEE Transactions on Evolutionary Computation*, vol.6, no.6, December 2002, p. 594-609.

[21] L. He and N. Mort. "Hybrid genetic algorithms for telecommunications network back-up routeing," *BT Technology Journal*, vol.18, no.4, Oct 2000, p. 42-50.

[22] Bauer, R.J. Jr., *Genetic Algorithms and Investment Strategies*. Wiley, New York, 1994.

[23] F. Allen, R. Karjalainen, "Using genetic algorithms to find technical trading rules1," *Journal of Financial Economics,* Vol. 51, pp. 245-271, 1999.

[24] Peter Grogono, Concordia University, personal correspondence, 2008

[25] Heinz-Otto Peitgen, Hartmut Jürgens, and Dietmar Saupe, *Chaos and Fractals: New Frontiers of Science*, Springer, 2004, p. 448-464.

[26] Paul Martz, "Generating Random Fractal Terrain," gameprogrammer, http://www.gameprogrammer.com/fractal.html, (current Nov 28, 2008).

[27] OpenGL Architecture Review Board; editor, Dave Shreiner, *OpenGL reference manual : the official reference document to OpenGL, version 1.4* , Boston : Addison-Wesley, 2004.

[28] Glen Cowan, *Statistical Data Analysis: with applications from particle physics,* Oxford University Press, 1998, p.32.

[29] Everett F. Carter Jr. "Implements the Polar form of the Box-Muller Transformation", 1994, ftp://ftp.taygeta.com/pub/c/boxmuller.c, (current Nov 28, 2008).