Integration of Optimization by Genetic Algorithms into an L-System-Based Animation System

Dr. Hansrudi Noser, Prof. Dr. Peter Stucki, Hans-Peter Walser
University of Zurich, Institute of Computer Sciences, MultiMedia Laboratory
Winterthurerstrasse 190, 8056 Zurich, Switzerland
noser@ifi.unizh.ch, stucki@ifi.unizh.ch, hp@ggaweb.ch

Abstract

In Computer Graphics L-systems represent a powerful rule-based language for modeling complex objects and their animation. However, designing objects and animations by rules is a difficult task, because designers often cannot foresee the consequences of rules. This is especially true for non-experts in the domain. Therefore, we propose to enhance an L-system based animation system with evolutionary features based on genetic algorithms (GAs). These features support the designers' task of interactively modeling objects and animations. Starting from an initial population of L-system-defined objects, the computer proposes iteratively new populations based on fitness value that are determined by the designers' creative or functional criteria. Moreover, automatic optimization of L-System-defined objects/animations is possible if an appropriate fitness function can be found for a given problem. We present a concept to integrate optimization by genetic algorithms into an L-system based animation system. Typical examples, such as automatic function optimization and creative interactive design of objects, illustrate our work.

1. Introduction

L-systems represent an alternative way of modeling and animating 3D virtual scenes. They are particularly advantageous in modeling complex and dynamic scenes because of their inherent high data amplification factor. L-systems, L-grammars, or so-called production systems [1] belong to the family of parallel rewriting systems. Rewriting is a technique for building complex objects by successively replacing parts of a simple initial object according to a set of rewriting rules or productions.

In computer graphics, an L-system describes a 3D object by an axiom and a set of production rules, also called its grammar. From such an L-system, the computer can derive iteratively 3D objects of a given structure. Furthermore, the computer can visualize the derived symbolic objects by interpreting them as a kind of turtle-graphics language.

Traditionally, L-systems (Lindenmayer-systems) [1, 2, 3] model formally plants and fractals. In [4] we describe a behavioral animation system where production rules define not only growth and topology of objects but also their real-time behavior. L-system based animation systems are not wide spread. One reason might be, that people are not familiar to programming with rules. Particularly, the design of highly parametric, time dependent, and parallel sets of rules is a difficult task, because it is not evident to foresee the consequences of rules. Therefore, expert knowledge and exhaustive tests are necessary for a successful design. To improve this non-satisfactory situation, we propose to enhance L-system based animation systems with an evolutionary, goal-oriented optimization mechanism supporting the difficult task of designing by L-systems.

As an evolutionary approach to this problem, Jacob [5] presented the genetic L-system programming paradigm for creation and development of parallel rewrite systems. He describes genetic operators that dynamically change the structure of the axiom and the rules, resulting in a huge search-space. For our purposes, however, genetic programming, which changes the rules and not only its parameterization, is to uncontrollable and would produce too many unfit individuals in interactive examples. Therefore, we prefer to use genetic algorithms that optimize only parameters of L-systems. A parameter set, characterizing an L-system defined object, represents a chromosome of the L-system. By defining a whole population of individuals, characterized by their chromosomes and a suitable fitness-function for goal-oriented evolution, a genetic algorithm can manage many kinds of optimizations. By evolving only sets of parameter, we can also influence in a controllable way the structure of rules if we associate evolved parameters to probabilities of stochastic rules.

Like L-systems, GAs are inspired from Biology and represent a universal optimization method. Genetic algorithms are well treated in literature. Since John Holland et al. have developed genetic algorithms in the 1960s, many applications have been described in literature. Introductions and references to genetic algorithms can be found in [6] or [7], for example.

2. The L-system based animation system

Our L-system application is timed, parametric, and suitable for interactive real-time simulations if the corresponding L-system defined objects are not to complex. It can also be used for raytraced movie generation containing many complex objects and environments such as plants and fractal mountains. The basic features and implementation details can be found in [4]. The most recent extensions with Virtual Reality features and Physics integration are described in [8] and [9]. The Webpage [10] contains an exhaustive list of references and illustrates the program's capabilities with pictures from typical applications.

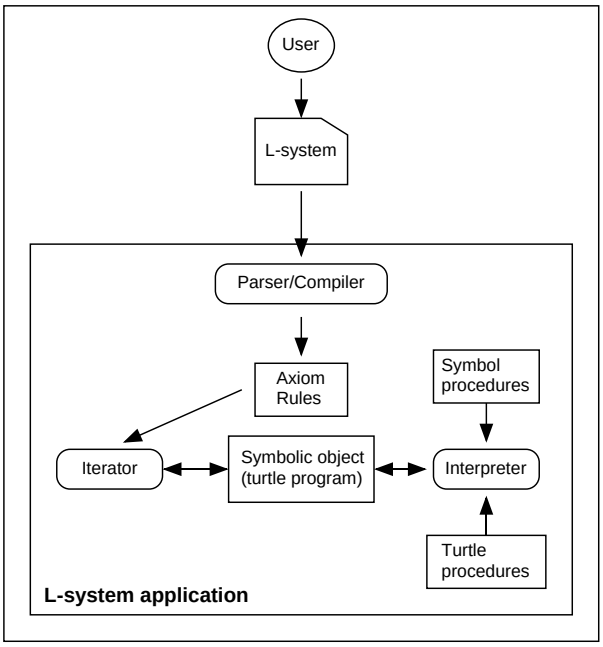

Figure 1. Architecture of the L-system based animation system.

The starting point of this work is the real-time L-system application described in [4]. Figure 1 illustrates its main components. A user designs an object by the corresponding L-system. The parser and the compiler convert it into an internal representation of the axiom and the rules. Then, at each frame, the iterator iterates the symbolic object according to the rules. At the beginning of a simulation the symbolic object corresponds to the axiom. After the iteration, the interpreter interprets this symbolic turtle program by using procedures of the *turtle* and the *symbol* modules, and produces the current frame of the virtual scene.

The L-system application is timed, parametric, conditional, and stochastic. Timed symbols of the alphabet of an L-system depend on time. They have a local age, and they can only be replaced by a rule if they have reached their maximal age. This time dependency is necessary to model continuous animation. The L-system is also parametric and conditional. The parameters allow the parent symbols (left side of production rule) to pass their parameters to their children (right side of rule) and to modify them. With each rule we can associate a condition that triggers the rule if it is true. In general, conditions depend on the parameters of the parent symbol and some environmentally sensitive functions.

Symbols can also have "growth"-functions (attributes) that determine their growth or behavior during their lifetime. Such functions can describe the geometric growth of an object, or the movement of a camera in space, for instance. Growth functions can depend on the local age, the global time, the symbol parameters, and other functions.

In a real-time L-system based application, at each frame (time step), the symbolic object is first iterated and then interpreted. At an iteration step, each symbol of the symbolic object that has reached its maximal age and whose rule is triggered by a Boolean expression (the condition), is replaced by the right side of the rule. Otherwise, only its local age is increased by the time step of the animation loop.

At an interpretation step the symbolic object is interpreted from left to right. The symbolic object can be considered as the actual turtle program, which builds and controls the virtual environment of the current frame. Each symbol of this "program" corresponds to a more or less high-level procedure with a given semantics.

3. The genetic algorithm

We use a genetic algorithm similar to that described in [Gold89]. It contains non-overlapping string populations, reproduction, crossover, and mutation. The genetic algorithm consists of the following main steps.

1. Initialize population.
2. Determine fitness of all individuals during simulation period.
3. Generate new population (reproduction, crossover, mutation).
4. Go to 2.

At the initialization step the individuals are randomly generated or set according to the designers preferences.

The fitness values are determined at a simulation step or period of the application. When all or a part of the fitness values are determined, the application triggers the generation of a new population or parts of it. Then, the whole process continues iteratively at point 2 of the above list.

The generation of a new population uses reproduction, crossover, and mutation operators. It consists of the following steps.

3.1 Select $parent_1$ and $parent_2$ from the population.
3.2 Generate two new children by eventually mating the parents and by eventually mutating them.
3.3 Go to 3.1 until *n* new children are generated.

Reproduction is realized by the selection of two parents. We use roulette wheel selection with slots weighted in proportion to fitness values. In order to improve the behavior of the genetic algorithm at the begin and at the end of an optimization run, we use a linear fitness scaling as described in [Gold89]. This fitness scaling prevents at the beginning of a run a premature convergence to a possible local optimum. At the end of a run, however, it focuses on the best individuals.

The following steps describe the generation procedure.

3.2.1 Randomly choose crossover according to crossover rate.
3.2.2 If crossover, then produce two new children by determining randomly a crossover point and crossing them. Mutate them according to the mutation rate.
3.2.3 If no crossover, copy the parents to the new children. Mutate them according to the mutation rate.

This simple but general genetic algorithm can be used for a variety of optimization problems. Before treating particular examples, however, the next section describes first its integration into the L-system based real-time and interactive animation system.

4. Integration of genetic algorithm

The main idea of this paper is to enhance the L-system based animation system with a versatile optimization mechanism for L-systems based on genetic algorithms. Many solutions to this goal are possible. We propose a concept that is characterized by the following features.

a. Each L-system can contain a population that is declared in the header of the L-system definition. A population contains *n* individuals or chromosomes.
b. An individual or chromosome is defined by *m* numbers between 0 and 1. A multi-parameter coding is used to map the parameters to the string of each individual.
c. There exists a function *getGen(i,j)* returning parameter *j* of individual *i*, which can be called in parameter expressions, growth-function expressions, and condition expressions of an L-system.
d. There exists a function *getFitness(i)* returning the fitness value of individual *i*, which can be called in expressions of the rules' conditions.
e. There exists a function *setFittness(i, value)* setting the fitness of individual *i*, which can be called in growth-function expressions.
f. There exists a function *addFittness(i, value)* adding a value to the fitness of individual *i*, which can be called in growth-function expressions.
g. There exists a function *setIndexLength(length)* that determines the current size of a subset of the population.
h. There exists a function *setIndex(i, j)* setting at entry *i* of the subset-index the individual *j*. The subset index determines the individuals that are evolved at the next generation. This function can be called by parameter expressions or by growth-function expressions.
i. There exists a function *evolveSet()* triggering the generation of the part of the population, that is determined by the set-index.

According to point b. an individual is given by *m* floating point parameters between 0 and 1. It must be converted to a sub-string representation of *k* bits. In the coding process each parameter is first linearly scaled to an integer value between 0 and 2^k. This integer value is then coded in binary notation of *k* bits. Thus, each parameter is represented by a sub-string of *k* bits, forming the whole chromosome or individual of *k***m* bits.

The GA implementation is based on a server-client architecture. Server and clients communicate through a TCP/IP network. They use a simple, text-based protocol to exchange genetic data and requests. This architecture was chosen because of their versatility and because of reusability considerations. As GA are very general tools for many optimization tasks, the GA server is completely decoupled from the L-system based application. Therefore, it can serve also many other types of clients.

The server handles the general string populations, the genetic algorithm, and the statistics. It receives real-valued population parameters, fitness-values, and evolution requests from the clients. Then, it encodes the parameters to its binary string-chromosomes of the population, makes an optional fitness-scaling, executes the genetic algorithm, converts the new chromosomes to real parameters, and sends them back to the clients. Note that the server saves the newly evolved chromosomes in a separate data structure, as its fitness values have to be

determined first by the client. The new chromosomes are copied to the active population when the client sends them back with valid fitness values. Therefore, in interactive real-time simulations, this particular implementation can handle the evolution of subsets of individuals by maintaining a complete population with valid fitness values.

The client is a real-timed L-system application that uses the services of the GA server. According to its current L-system program, the client uses and manipulates extensively the real-valued chromosomes and fitness values by using the functions defined above. When the L-system client needs a new population or parts of it, it sends the corresponding individuals with the actual fitness values to the server and waits for a new population.

These features extend essentially the animation system by enhancing it with a genetic algorithm for versatile optimizations. In order to illustrate their application and their power, the next section shows two typical examples of their use in L-systems.

5. Examples

In the following sections, two different examples of L-systems optimize functions and an interactive design of a tree with the genetic algorithm. The examples illustrate the proposed extensions. However, many other applications are possible too.

replace *symbol*(t, ${}_p x_0$, .. , ${}_p x_n$)
if condition(t, ${}_p x_0$, .. ,${}_p x_n$)
by *symbol*(x_0=expr(${}_p x_0$,..,${}_p x_n$), .. , x_m=expr(${}_p x_0$,..,${}_p x_n$))
{attribute$_o$=expr(t, x_i, .. ,x_m), .. ,
attribute$_q$=expr(t, x_i, .. ,x_m))}*

t: local age of the symbol
n: number of parameters of a parent symbol
m: number of parameters of a child symbol
q: number of attribute functions of a child symbol

Figure 2. Syntax of an L-system rule that rewrites a symbol, if the condition is true, by its right side consisting of a sequence of parametric symbols.

Figure 2 shows the description of the rule-syntax of L-systems that is used in this paper. The symbolic object is the axiom at the beginning of an animation. Triggered by its true condition, the rule's right side replaces a parametric, timed symbol of the symbolic object. The condition can depend on the local age of the symbol, its parameters, and other specialized functions making part of a particular implementation. The right side of the rule consists of a sequence of timed and parametric rules. These symbols can contain expressions for their parameters and attributes.

Note that parameter expressions, enclosed by parenthesis *()*, are only evaluated once when the rule is triggered by a true condition. However, attribute expressions, enclosed by accolades *{}*, are evaluated at each frame of an animation. Parameter expressions can depend on the parent parameter values – i.e. the parameters of the symbol that is replaced - and other specialized functions. Attribute expressions can depend on the local age, their symbol's parameters which are set by the parameter expressions, and other specialized functions.

5.1. Function optimization

The first example, a classic optimization of a function with only one argument, illustrates the application of the genetic algorithm in the animation system. The task is to find the maximal value of the function $f(x) = x^5$, and to visualize the function and the evolution process in a virtual 3D space. In this example the parameter x corresponds to the chromosome of an individual, and the function value *f(x)* to its fitness value. Figure 3 shows the corresponding L-system pseudo code.

Define Population: nPop=30, nGene = 1
evolutionPeriod = 3
cubeSize = 0.2

Axiom: *individual*(x_0=0)

Rule$_1$:
replace *individual*(${}_p x_0$) if (${}_p x_0$ < nPop) by
goTo(x_0 = ${}_p x_0$)
{xPos = getGene(x_0, 1), zPos = 0,
yPos = setFitness(x_0, (getGene(x_0, 1)^5))}
drawCube{length = cubeSize, width = cubeSize,
height = cubeSize}
individual(x_0 = ${}_p x_0$+1)

Rule$_2$:
replace *individual*(${}_p x_0$) if ((${}_p x_0$==nPop) AND
(t > evolutionPeriod)) by
individual(x_0=${}_p x_0$, x_1=evolveSet())

Figure 3. An L-system that searches for the maximal value of the function $f(x)=x^5$ in the interval [0,1], and that visualizes the function and the evolution process.

In the header of the text file that defines the L-system, the designer declares a population of thirty individuals, each consisting of one parameter *x*. This parameter *x* corresponds to the unique argument of the function to be

optimized. The axiom of the L-system consists of only one symbol *individual* whose parameter x_0 serves as individual counter and is set to zero at the begin. The first rule builds the complete population by replacing iteratively the symbol *individual* by the right side of the rule, i.e. the symbols *goTo*, *drawCube*, and *individual*. The *goTo* symbol places the turtle at the position where the symbol *drawCube* draws a cube representing the data point at (x, $f(x) = x^5$). The function *getGen* returns the gene of the individual x_0, and the function *setFitness* sets the individual's fitness value. The attributes *xPos*, *yPos*, and *zPos* of the symbol *goTo* determine the coordinates of the turtle position to be set. Finally, the symbol *individual* increments the individual counter. It will be again replaced at a subsequent iteration step as long as the rule's condition is true. The second rule is responsible for periodically triggering an evolution step as soon as the complete population has been created. Note that the functions *setFitness* and *getGen* are called in attribute expressions, which are evaluated at each frame of the animation. Therefore, the *setFitness* and *addFitness* functions can update an individual's fitness-value throughout his or her life. The evolve function, however, is called in a parameter expression which is only evaluated once at the iteration step when the corresponding symbol *individual* is created by a rewriting step. Therefore, an evolution step is triggered only once after each evolution period.

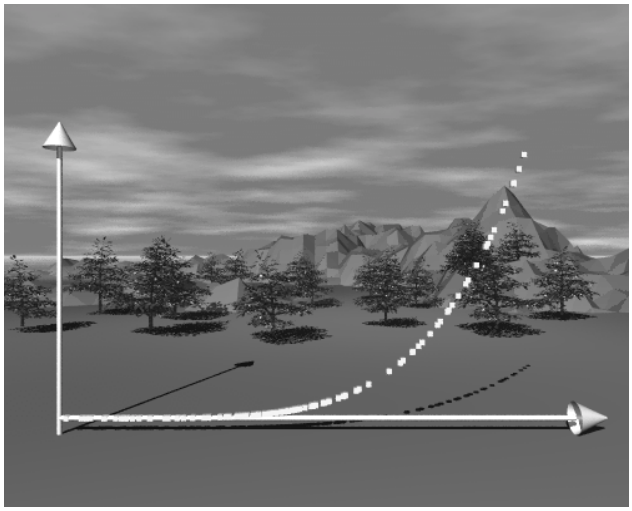

Figure 4. The visualized L-system of Figure 3. The data points, which correspond to the phenotypes of the population, are embedded in a 3D virtual scene that is rendered with a raytracer (Rayshade, Craig E. Kolb).

Figure 4 shows a snapshot of the animated L-system of Figure 3. Note that the L-system of Figure 3 contains only the description of the data points of the function representing the population. The definition of the rest of the virtual scene by L-systems is not included in the code as it is not relevant to the topic of the paper.

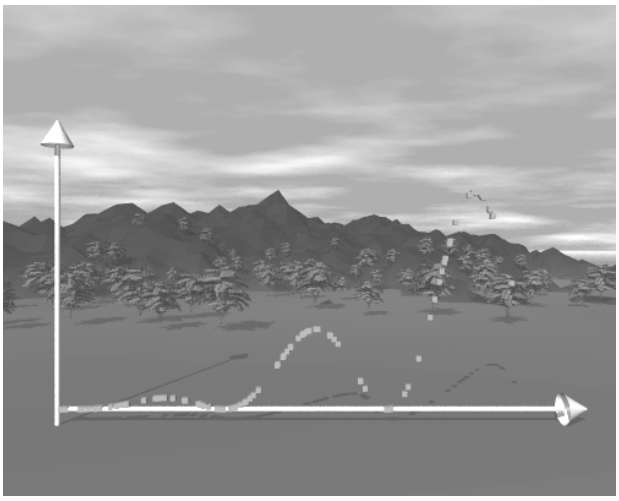

Figure 5. The 1D function $(x*\sin(x))^2$ with several local maximums.

```
nGene = 2
goTo( x0 = px0)
        {xPos = getGene(x0, 1), zPos = getGene(x0, 2),
        yPos = fitnessSet(x0, |getGene(x0, 1)*
                    getGene(x0, 2)*sin(getGene(x0, 1))|)}
```

Figure 6. Modifications of the rule of Figure 3 for a function optimization depending on two parameters.

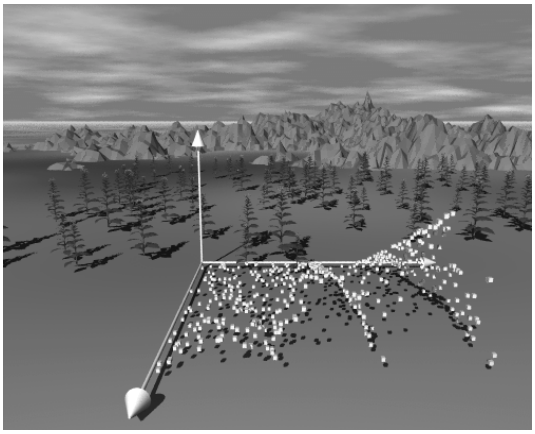

Figure 7. The visualized L-system that searches for the maximal value of the 2D function f(x,y)=(|x*y*sin(x)|.

Figure 5 and Figure 7 show two more examples of function optimization. The function $(x*sin(x))^2$ contains several local maximums in a given interval. A GA finds without problems the global maximum of such functions. The other 2D function $f(x,y)=(|x*y*sin(x)|)$ has two parameters *x* and *y*, that correspond to the chromosome of an indiviual with a fitness value of $f(x,y)$. As indicated in Figure 6, the design of the corresponding L-system is straightforward. In the declaration of the population the number of genes is set to two, and the *goTo* symbol places the turtle at the coordinate $(x, y, fitness(x,y)= (|x*y*sin(x)|)$.

5.2. GA supported design

The next example illustrates the use of genetic algorithms in order to support interactive optimization of parametric objects defined with parametric L-systems. A part of the parameters of the L-system consists of the chromosomes of individuals. For timed L-systems the space of possible phenotypes is huge, and in general, it can't be visualized completely. Therefore, the computer visualizes only a small population, and the user assigns to each individual interactively a fitness value. When all individuals are assessed, a new population is generated and presented to the designer. After many iterations of this creative process, the population should evolve to an object according to the designers taste.

Figure 8 shows the pseudo code of the L-system that manages the user interaction and displays the phenotypes of the population. In the axiom the whole population, represented by the symbol *individual*, as well as the user interaction symbol, represented by the symbol *userInteraction*, are placed between push and pop symbols. These stack operations act on the turtle state given by its position and orientation. Such a branch of an L-system, bracketed by these two symbols, can be completely eliminated by a *cut* symbol when a new population is needed. The x_0 parameter of the symbol *individual* represents the individual counter needed to build up the entire population. The parameter x_0 of the symbol *userInteraction* serves to manage the individual during user interaction. The symbols *goTo* and *drawCone* mark the individual to be assessed by the designer.

Only one rule is necessary for modeling the user interaction. Every time-step, given by *scanTime*, the keyboard is scanned. If a key has been pressed, its value serves as fitness value and triggers the rule. In this case the *userInteraction* parameter x_0 is incremented, the fitness value of the current individual is set with the key value, and the position of the marker is translated by a relative translation to the next individual. This is possible by a relative translation if the individuals are sequentially positioned and evenly spaced.

```
Define Population: nGene=3, nPop=6
ScanTime = 0.1

Axiom:
push
individual(x0=0)
goTo{xPos= -2, yPos=1, zPos=0}
userInteraction(x0=0)
drawCone{length=0.3, base=0.1}
pop

Rule1:
replace userInteraction(px0)  if (t >scanTime) AND
        (keyValue()>= 0) by
relativeMove{xTranslation = 2}
userInteraction(x0 = px0+1, x1=setFitness(px0,
        keyValue())

Rule2:
replace userInteraction(px0)  if (x0>=nPop) by
cut
push
individual(x0=0)
goTo{xPos= -2, yPos=0, zPos=0}
userInteraction(x0=0, x1=evolveSet() )
drawCone{length=0.3, base=0.1}
pop

Rule3:
Replace individual ….
```

Figure 8. L-system code that allows interactive designers to assign fitness values to individuals.

Rule 2 is triggered if the designer has assessed all individuals. In this case the cut symbol first eliminates completely the branch with the old population. Then a new branch is added, that is a copy of the axiom. In the same time the *evolveSet()* function triggers the generation of a new population. Now, the L-system builds up a new population determined by the new chromosome set, and the designer can assess them again.

Figure 9 shows an example of a tree, which is parameterized by three genes: a branching angle, the branch length, and a complexity number. The L-system application proposes six trees characterized by their chromosome. The designer can assess the currently marked tree by a keystroke (0 to 5) that determines the tree's fitness value.

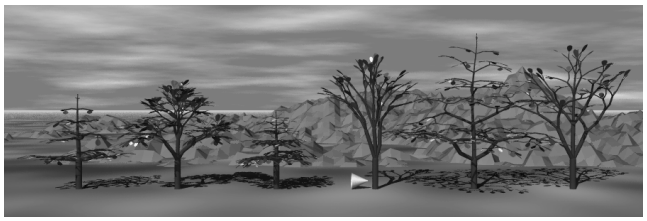

Figure 9. Trees parameterized with three parameters representing their chromosome. The computer proposes the trees for assessment by the designer.

6. Conclusions

This paper presents a method for integrating genetic algorithms for optimization purposes into an L-system based modeling and animation system. Only a small set of functions useable in conditions of rules and parameter/attribute functions of symbols is necessary for the L-system programming interface to apply the new features. These versatile and powerful extensions enable designers to develop animations with function optimization, GA supported interactive design, or elements of Artificial Life.

Future work will focus on artificial life simulations similar to that indicated in Figure 10, where a population of butterflies flies around a flower. The group animation is driven by force fields, which are parameterized by the genes of the butterflies. We think there is a large potential for new insights into L-system-animated and modeled virtual scenes populated by groups of actors and autonomous actors.

Figure 10. A population of butterflies, animated by force fields that are parameterized by the butterflies' genes.

Acknowledgements

The idea of using genetic algorithms for the optimization of L-systems originated some years ago in the Computer Graphics Laboratory LIG of EPFL. The authors would like to thank its director Prof. Daniel Thalmann, and also Paolo Baerlocher, who made investigations in this direction during a project work for students.

References

[1] P. Prusinkiewicz, A. Lindenmayer, *The Algorithmic Beauty of Plants*, Springer Verlag, 1990.

[2] P. Prusinkiewicz, M.S. Hammel, E. Mjolsness, *Animation of Plant Development*, Computer Graphics Proceedings, SIGGRAPH '93, Annual Conference Series, ACM Press, pp. 351, 1993.

[3] P. Prusinkiewicz, M. James, R. Mech, *Synthetic Topiary* , SIGGRAPH 94, Computer Graphics Proceedings, Annual Conference Series, pp. 351-358, 1994.

[4] H. Noser, D.Thalmann, *A Rule-Based Interactive Behavioral Animation System for Humanoids*, IEEE Transactions on Visualization and Computer Graphics, Vol. 5, No. 4, October-December 1999.

[5] Jacob, C., Genetic L-System Programming, PPSN III - Parallel Problem Solving from Nature, International Conference on Evolutionary Computation, Lecture Notes in Computer Science 866, Springer-Verlag, Berlin, 1994, pp. 334 - 343.

[6] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley Longman, 1998.

[7] M. Mitchell, An Introduction to Genetic Algorithms, A Braqdford Book, The MIT Press, Cambridge, Massachusetts, London, England, 1998.

[8] H. Noser, C. Stern, P. Stucki, *Distributed Virtual Reality Environments Based on Rewriting Systems***,** submitted to IEEE Transactions on Visualization & Computer Graphics.

[9] H. Noser, S. Rudolph, P. Stucki, Physics-Enhanced L-Systems, WSCG'2001, The 9-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2001, Plzen, Czech Republic, February 5. - 9. 2001, Conference Proceedings, Vol 2, pp 214-221, 2001.

[10] H. Noser, *Publications*, http://www.ifi.unizh.ch/~noser/work.html