Evolving Artificial Trees Described by Parametric L-systems

Abstract

The paper describes a solution of the inverse problem for parametric Lindenmayer systems with genetic algorithms. Inverse problem means that we try to evolve Lindenmayer grammars to describe plants. A genetic algorithm is used to evolve the rewriting rules and the set of parameters of the system. In this paper we present an extension of the solution how genetic algorithms can be applied to simulate the evolution of artificial plant morphologies of Ochoa [8], and Mock [7]. The three main improvements are that more populations are processed in parallel and an adaptation scheme for the application probability of genetic operators is applied. The turtle graphics system has to be also modified, so that three-dimensional structures such as trees can be evolved and drawn.

1 Introduction

This paper describes the solution of the inverse problem for parametric Lindenmayer systems using genetic algorithms. Inverse problem means that we try to evolve Lindenmayer grammars to describe plants. The program *ELSFI* (*Evolution of Lindenmayer Systems for Fractal Images*) developed by us in order to solve the problem allows the user to select different types of *fractal images*, but in this paper we focus on a parametric tree (see in Figure 1). The goal is to define a tree given as a picture with the help of some kind of grammar, but to show how our system works the Figure1 is generated itself with the help of a L-system. That way it can be demonstrated that the *ELSFI* system can find the correct L-system to describe a parametric tree. The *parametric Lindenmayer grammars* seemed to be a suitable descriptive grammar.

The *Lindenmayer systems* (L-systems) are parallel grammatical rewriting systems. They contain an initial axiom and one or more rules. With the help of such a grammar complex objects can be described starting from the axiom and using the productions [10]. When choosing the representation form, we took into consideration that L-systems can be evolved using genetic algorithms [4].

Since parametric L-systems are evolved, two kinds of genetic operators are applied. The first kind operates on the rewriting rules of the evolved L-system, and the other kind operates on the parameters of the rewriting rules. The first solution of this problem was presented by John R. Koza [5] and refined among others by Jacob [4], Mock [7] and Ochoa [8].

The main idea described in this paper is that we applied an extended version of genetic algorithms: more populations are processed in parallel, and after a while the populations meet. Thus, populations could change each other's individuals. It can happen on the base of either the best or median fitness value or by user intersection; this means that the versatility of populations could be increased.

As further improvement we applied an adaptation scheme for the application probability of genetic operators. With the help of this method, operators which produce better individuals can be used more frequently, so better individuals are more likely produced. This method also increases the speed of convergence. The developed *ELSFI* system is implemented in *C++*. To draw the individuals, the well-known turtle graphics system is applied. It had to be also modified to draw three-dimensional structures such as trees. Our turtle system has been extended in another way, it can produce Postscript output from the turtle commands. This output can be saved to a file, and later viewed with any Postscript-interpreter. The figures in this paper were created in this way.

We proceed as follows. First in Section 2 a brief overview of the concept is presented. The evolution process is presented in Section 3. After that Section 4 shows simulation results and finally Section 5 contains a summary and some comments on future work.

2 Fundamentals of the evolution using parametric L-systems

In this section a short overview of concepts used in this paper is given. First the Lindenmayer system, focusing on parametric L-systems is defined. Then a short description of the 3D turtle system which draws the individuals generated by the grammar on the screen is presented. After it the description of the demonstration tree (see in Figure 1) is given. In the rest of the section the evolution process is explained.

2.1 Lindenmayer systems

Lindenmayer introduced special kinds of descriptive grammars in order to describe natural development processes ([6]). These are called Lindenmayer grammars or *Lindenmayer system*s (*L-system*s). The most important characteristic of *L-system*s compared with Chomsky grammars is *parallel rewriting*. In Lindenmayer systems rewriting rules are applied in parallel and simultaneously replace all letters in a given word.

Different *L-system* classes are introduced, such as deterministic or nondeterministic *L-systems*. Context free and context sensitive *L-systems*. can be distinguised The definition of these classes are similar to the definition of Chomsky language classes. In this paper only *deterministic context free L-systems*, or *D0L-systems* for short are used. To enlarge the number of described objects *L-systems* can also be associated with parameters.

2.2 Parametric D0L-systems

With the idea of parametric *L-systems* the concept of parallel rewriting to parametric words instead of strings of symbols can be applied. Following the notation of Prusinkiewicz ([9]) the word "module" as a synonym for "letter with associated parameters" is used.

The alphabet is denoted by Σ, and the set of parameters is the set of real numbers $\boldsymbol{R}$. A module with letter $a \in \Sigma$ and parameters $p_1, p_2, \ldots, p_n \in \boldsymbol{R}$ is denoted by $a(p_1, p_2, \ldots, p_n)$.

The real-valued *actual* parameters appearing in words have a counterpart in the *formal* parameters, which may occur in the specification of *L-system* productions. Let Π be the set of formal parameters, The combination of formal parameters and numeric constants using the arithmetic operators $(+,-,*,/)$ the relational operators $(<,<=,>,>=,==)$, the logical operators $(!,\&\&,||)$ and parentheses $(())$ generates the set of all correctly constructed logical and arithmetic expressions with parameters from Π are noted $\mathcal{C}(\Pi)$ and $\mathcal{E}(\Pi)$.

DefinitionA *parametric* D0*L-system* is an ordered quadruple $G = (\Sigma, \Pi, \alpha, P)$, where

- $\Sigma = s_1, s_2, ..., s_n$ is the alphabet of the system,
- Π is the set of formal parameters,
- $\alpha \in (\Sigma \times \boldsymbol{R}^*)^+$ is a nonempty parametric word called the axiom,
- $P \subset (\Sigma \times \Pi^*) \times \mathcal{C}(\Pi) \times (\Sigma \times \mathcal{E}(\Pi)^*)^*$ is a finite set of productions.

A production in a D0*L-system* is given in the form $pred \mid cond \rightarrow succ$, where $pred \in \Sigma \times \Pi^*$ denotes the *predecessor* and $succ \in (\Sigma \times \mathcal{E}(\Pi)^*)^*$ is the *successor*; $cond \in \mathcal{C}(\Pi)$ describes the *conditions*. A production *matches* a module if the following conditions are met:

- the letter in the module and the letter in the production predecessor are the same
- the number of actual parameters in the module is equal to the number of formal parameters in the production predecessor, and
- the condition evaluates to true if the actual parameter values are substituted for the formal parameters in the production.

If a production is matching a module then it can be *applied* in order to create a string of modules specified by the production successor. The actual parameter values are substituted for the formal parameters according to their position.

There is no essential difference between *D0L-systems* that operate on strings with or without brackets. In our implementation, brackets are used; however, in this case productions involving brackets are restricted to the following forms:

- $pred \mid cond \to succ$, where *pred* and *cond* are as before, but $succ \in ((\Sigma \times \mathcal{E}(\Pi)^* \cup \{[,]\})^*$, and *succ* is well nested;
- $[\to [$, or $] \to]$.

A string S is called well-nested if and only if for all S' prefix of S the number of '[' symbols is not less than the number of ']' symbols, and the number of these symbols is equal in S. More details and examples of parametric *D0L-systems* can be found in [10].

2.3 Turtle

To draw the generated Lindenmayer-systems, the well known turtle graphics was used. Since its application in the *LOGO* language, turtle graphics became an often used tool to visualize complex objects given in a string format. The turtle graphics system was modified in order to draw *3D images*. The turtle is defined as follows:

$$T = (\vec{P}, O)$$

where $\vec{P}$ is the position vector, and O is the orientation matrix of the turtle. O gives the three main directions: heading, left and up. Thus O can be defined as $[\vec{H}, \vec{L}, \vec{U}]$, where $\vec{P}$, $\vec{H}$, $\vec{L}$ and $\vec{U}$ are three-dimensional vectors.

A simple turtle just moves around and draws the lines, circles, etc. But sometimes it is helpful to get the graphics in other formats as well. So a basic turtle was implemented, which is responsible only for movements. Additionally modified versions of this *moving-only-turtle* were created which are the following: a drawing turtle, a postscript turtle and a pov-ray turtle.

The *drawing turtle* not only moves, but draws to the screen. The two others generate input files for interpreters. The *postscript turtle* generates Postscript output, therefore it can be displayed by an interpreter or can be included in other Postscript documents. The *pov-ray turtle* generates an object description file for the *POV-Ray ray tracer* program. Figure 2 shows the ray-traced version of the demonstration tree (see figure 1). It is useful when one wants to generate 3D graphics, or wants to use the generated graphics as a part of an image. These extensions also allow to put more objects together in a single image.

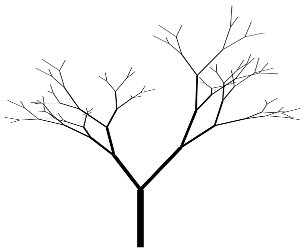

Figure 1: The demonstration tree

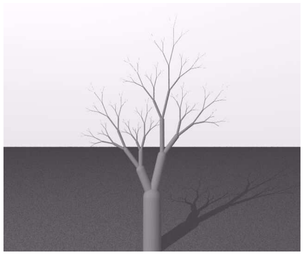

Figure 2: The ray-traced demonstration tree

2.4 Description of the demonstration tree with L-system

In this paper we use a demonstration tree as an illustrative example (see Figure 1). The axiom of the L-system describing it is $A(l_0, w_0)$ and there is only one rewriting rule:

$$A(l,w) \quad | \quad l >= min \quad \rightarrow \quad (w)F(l)[+(\alpha_1) < (\phi_1)A(l \cdot r_1, w \cdot q)][+(\alpha_2) < (\phi_2)A(l \cdot r_2, w \cdot q)].$$

The angle values $\alpha_1, \alpha_2, \phi_1, \phi_2$ determine the orientation of the turtle when drawing a line from a node, and r_1 and r_2 determine the decrease in the length of the line segment as we go upward the tree; q determines the decrease in the length of the line segment as we go upward the tree. For the demonstration tree, these values are the following:
$\alpha_1 = 25°, \alpha_2 = -30°, \phi_1 = 60°, \phi_2 = 120°, r_1 = 0.65, r_2 = 0.85, w_0 = 20$ and $q = 0.65$.
The demonstration tree can be obtained by the 7th iteration of the L-system described above.

2.5 Evolution algorithms

Evolution algorithms are computer-based problem solving systems which use evolution processes as they can be observed in nature [1]. Main types of evolution algorithms are evolution strategies ([11]), genetic algorithms ([3]) and genetic programming [5]. All of them are based on the simulation of evolution of individuals by means of natural selection, mutation and reproduction.

2.5.1 Evolution using more populations

As it was mentioned above, our system can process more populations in parallel. If this mode was chosen by the user, then the system generates the chosen number of populations with the same size, and the populations are processed in the same way. Of course the weights of the operators (see subsection 2.3.3) are computed independently for the different populations. This means that, for example, the weight of the operation mutation can be different for population 1 and for population 2 at the same state of the evolution process.

After a number of generations had been processed (the number can be also changed by the user), the populations are made to meet. This means that genetic operators are applied to the population of populations. At this level, for example, the operation recombination means that some individuals of the populations are exchanged. The function of other genetic operators can be defined in a similar way.

The advantage of this method is that if more populations are processed, then on the one hand they can exchange each other's best individuals, and so the speed of convergence could be increased. On the other hand, if only one population is processed, then the evolution process can get stuck. With more populations, this is less likely. More details can be found in [2].

2.5.2 Adaptation of the weight of the genetic operators

As it was mentioned earlier the descendants during the process are created by recombination, mutation and reproduction. The weight of each operator is adapted by observing their effectiveness: during one step (when generating a population of descendants) the effectiveness (the individuals generated with this operator have better fitness value than their parents) of each operator is observed. The more effective the operator is, the more likely we are to use it in the next step. Thus, the weight of the operators is continually changed during the evolution process.

The probability of operator $\omega \in \Omega$ in generation t is

$$\mathcal{P}(\omega) = \frac{\gamma(\omega,t)}{\sum\limits_{\omega' \in \Omega} \gamma(\omega',t)},$$

where Ω is the set of operators, and $\gamma(\omega,t)$ is the weight of operator ω computed by

$$\gamma(\omega,t) = \gamma(\omega,t-1) \cdot \Delta\gamma(\omega,t) \cdot \gamma_{decay}(\omega).$$

The operator-specific constant $0 \leq \gamma_{decay}(\omega) \leq 1$ is introduced in order to prevent an operator from generating too many individuals in a generation. That is, no matter how bad an operator is, it still can produce some individuals (we must not exclude any operators from the process). $\Delta\gamma(\omega,t)$ shows how the operator ω increased the average fitness of the population in the last step. Its calculation depends on the number of operands used in the operator. For operators with one operand, the method is the following:

$$\Delta\gamma(\omega,t) = |P(\omega,t)|^{-1} \cdot \sum_{s \in P(\omega,t)} \frac{\phi(s)}{\phi(Parent(s))},$$

where $P(\omega,t)$ is the set of individuals created by operator ω in generation t, and $\phi(s)$ is the fitness value of the individual $s \in P(\omega,t)$. $Parent(s)$ is the parent of the individual $s \in P(\omega,t)$.

The method of computing the $\Delta\gamma(\omega,t)$ value for operators with two operands (e.g. recombination) is the following:

$$\Delta\gamma(\omega,t) = \frac{1}{2} \cdot |P(\omega,t)|^{-1} \cdot \sum_{s \in P(\omega,t)} \phi^*(s),$$

where

$$\phi^*(s) = \max\left\{ \frac{\phi(s)}{\phi(parent_1(s))}, \frac{\phi(brother(s))}{\phi(parent_2(s))} \right\}.$$

3 The evolution process

At the beginning of the evolution process the individuals in the generation 0 are randomly generated, Since parametric L-systems are evolved, two kinds of evolutionary operators are applied: genetic algorithms (*GA*) are used on the rewriting rules of the evolved L-system, and evolution strategies (*EA*) are applied on the parameters of the rules. To calculate the fitness value, the images described by the L-systems are created. The fitness function is implemented as the distance of two images with the help of the quadratic error but we extended this definition to make a difference between whether the point is 1 on both images or the point is 0 on the destination image, but 1 on the generated image or vice versa Because our image is a finite set of discrete points there is an upper bound for the fitness value. Therefore after finite steps there will be a population such that its fitness is the same as of the previous one. Using that the process can be stopped if the maximum fitness is reached, or nearly reached, or the fitness value was unchanged during the previous *N* steps. *N* can be defined as a parameter. Of course there are several other parameters that can be changed before the process and also during the evolution: the number of populations, the number of iterations of the L-system, and parameters controlling the *EA* and the *GA* process.

4 Simulation result

At the beginning of the evolution process various individuals appear. Then, after a few generations, when the appropriate rewriting rule appears, the generated trees are rather similar, differing only in their parameters. In Figure 3 some trees can be seen which appeared in a later generation. They differ only in their parameters. The parameters of these trees can be seen in Figure 4. Of course the speed of convergence is depending on several parameters of the genetic algorithm given by the user: more individuals mean faster convergence. The same can be observed about the number of populations.

5 Summary, future work

In this paper it is shown how the demonstration tree can be described with parametric Lindenmayer systems. The main goal of *ELSFI* program is to define a tree given as a picture with the help of L-system, but to show how our system works the Figure1 was generated with the help of a L-system using an L-system editor which is a part of the system. A genetic algorithm is used to evolve the