Using Genetic Algorithms to Improve the Visual Quality of Fractal Plants Generated with CSG-PL-Systems

Christoph Traxler and Michael Gervautz

Institute of Computer Graphics
Vienna University of Technology, Austria
e-mail: traxler|gervautz @cg.tuwien.ac.at - http://www.cg.tuwien.ac.at

Abstract

PL-systems are a powerful and flexible technique for plant modeling. Unfortunately it is a hard task to specify a PL-system, that generates a desired plant. Especially the tuning of the parameter values is time consuming and demands a lot of experience from the user. In this paper we describe how to apply genetic algorithms to CSG-PL-systems, which are a special class of PL-systems. A decomposition of CSG-PL-systems is introduced to extract those parts, which can serve as genotype. Mutation and mating, the two major operations of evolution techniques, are applied to this data set. With the described method it is possible to find easily natural looking individuals out of a species that is described in an abstract way by the underlying CSG-PL-system.

Key Words: genetic algorithms, artificial evolution, CSG-PL-systems, natural phenomena

1 Introduction

Genetic algorithms are a reliable and powerful search strategy to find sub-optimal solutions in huge data spaces. The principles of evolution, namely reproduction, mutation and mating are thereby applied to a great class of problems, which are frequently known to be NP-complete [GOLD89]. Like in nature, genetic algorithms usually do not find the very optimal solution, but one that satisfies certain qualitative requirements. This is a property they share with neural networks. The notion genetic algorithm was first introduced by Bagley in [BAGL67], who used them to solve problems of game theory. Rosenberg simulated the evolution of single cell organisms [ROSE67] to examine how nature works. Cavicchio introduced a genetic algorithm in [CAVI70] to create an efficient machine for pattern recognition. Holland compared adaptation in natural and artificial systems [HOLL75], and DeJong analyzed a class of genetic algorithms [JONG75] by comparing their efficiency for function optimization. In all cases the principle of evolution is applied to a set of data, which serves as genotype, for example a stream of bits or arithmetic expressions. The evolving individuals are tested for their fitness to solve a given problem. This is done by the so called fitness function, which has to be defined carefully to obtain the desired results. The complexity of the fitness function depends on the nature of the problem to be solved.

In this paper we describe an approach to apply genetic algorithms to find PL-systems that generate natural looking plants of a desired species. PL-systems are known to be a powerful tool for plant modeling. They are an important extension to the classical L-systems, which were firstly introduced by the biologist Lindenmayr and are fully described in [PRUS91]. Though PL-systems allow to simulate the development of plants with high accuracy, but it is a well-known fact, that it is a hard task to specify one that generates the desired plant. The reason for that is called data base amplification by Smith [SMIT84] and characterizes the problem of tuning a small data set to obtain a complex object as result of a recursive feedback process. This problem clearly belongs to a class that can easily be solved by a genetic algorithm. Real plants are a product of evolution. Thus it is obvious to create them by using genetic algorithms.

In [SIMS91] evolution techniques were firstly applied to evolve several species of plants out of a unique and powerful model. The underlying model was a procedural one, much as that introduced in [DERE88], where the development of a plant depends on a set of specific parameters and probabilities. Thus this set of values is used as genotype. Hi did not implement a fitness function but let the user decide, which of the evolved individuals should be used for further reproduction. Within PL-systems the problem occurs to define the proper genotype. In fact the whole PL-system could be conceived as genotype of a subclass of species but this would not make much sense. A careful selection of some of the components of a species description has to be made to guarantee that evolution converges to the user's goals. Therefore we took only two of the components of a PL-system under consideration. In spite of this restriction it is possible to release the user from time consuming parameter tuning and to find a natural looking and proportional individual plant in a specific subclass of species in relatively short time.

In the first part of this paper we give an introduction to the methods that were used in [SIMS91] since we apply them to PL-systems. A short description of CSG-PL-systems follows which were introduced in [GERV95]. The remainder of this paper explains how genetic algorithms are applied to components of these systems and we further discuss future research possibilities.

2. Genetic algorithms for the evolution of plants

Genetic algorithms can be divided into two operations, mating and mutation. Two or more parents are selected out of a given or already evolved population to derive a new one. For that purpose the genotypes of the parents are reproduced and mated. Mutation guarantees variations within the new generation, so that the new population is not simply a mixture of inherited characters. The fittest individuals of the new generation are selected for further reproductions.

Karl Sims was the first, who applied genetic algorithms to evolve a variety of plant species [SIMS91]. Some of his results can be seen in his animation “Panspermia”. For the plant model he used a similar model as those introduced by deReffye [DERE88]. The plants are generated procedurally and their final shape depends on a set of parameter values and probabilities. These values control the activity of buds, which either generate branches with new buds, flowers, or simply die. The probabilities are modified with respect to the hierarchical position of a segment during growth. The geometric aspects of plants like branching angles and scaling factors depend also on parameter values. Sims used a set of only 21 parameter values as genotype. The selection of the fittest completely relies to the human user, so that a selection function is not necessary.

Mutation is done by adding random values out of a certain interval to each element of the value set with a mutation probability. Thus the effects of mutation can be adjusted for each gene of the genotype. If the mutation of a particular value is not intended, its mutation probability is set to zero. The intervals for the random numbers are chosen with respect to the valid range of each parameter value. A Gaussian distribution is used to make small changes more likelihood than large ones. The mutation rates and amounts are usually higher than in natural systems to accelerate evolution and allow a greater spread of different individuals in each generation.

For the mating of parameter value sets, Sims suggested four methods. After the selection of two individuals of a certain population as parents for a new generation, their genotypes can be reproduced and mated in the following ways:

1) Genes are taken from both parents alternately with a constant frequency *f*. The new genotype is thus an interleaved mixture of the parent genotypes, where *f* genes of the first parent are concatenated with *f* genes of the second parent in an iterative way as long as there are genes left. This method is called *crossover* and has the advantage, that adjacent genes of both genotypes are concatenated with each other.

2) Each gene of the new genotype is taken randomly from one of the parents. Here the genes are combined independently from each other. This was the method Sims preferred for his artificial evolution, maybe because the genotypes of the new generation are spread widely enough among the possible combinations of two initial genotypes.

3) The new genotype is a linear interpolation between the corresponding parent genotypes. The resulting genes are taken randomly from the line connecting the parent genes. This is achieved by the linear equation: $\textit{new gene} = p \bullet \textit{gene of } 1^{st} \textit{ parent} + (1-p) \bullet \textit{gene of } 2^{nd} \textit{ parent}$, where *p* is a random value out of [0,1]. For all genes the same random number *p* is taken in contrary to the next method.

4) Each new gene is a random value between the values of the corresponding parent genes independently of each other. This is the method, which causes the widest spread among the value ranges of both parent genotypes.

We exactly used these methods to tune parameter values of CSG-PL-systems. Before we explain how to apply Sims methods to CSG-PL-systems in greater detail, we give a short description of these systems in the next chapter.

3. CSG-PL-systems and their components

CSG-PL-systems [GERV95] are an adaptation of the classical PL-systems as described in [PRUS91] to the concept of Constructive Solid Geometry (CSG). The combination of this two powerful modeling schemes allows the description of a wide range of natural phenomena like plants, objects defined by IFS, Fractal terrain in a single unified way.

CSG is a well-known object representation, which can be rendered efficiently using ray tracing. Complex objects are constructed by the binary combination of properly transformed primitive objects, like cube, sphere, cone, cylinder and so on. The primitives are represented procedurally by their volumes and can be combined by the three Boolean operations (union ($\cup$), intersection ($\cap$) and subtraction (/)). The desired object is thus specified by a binary expression.

Since PL-systems operate on parametric strings, i.e., strings consisting of symbols with associated parameters, they could also generate CSG-expressions. The formal language of such PL-systems is a subset of the formal language of all valid CSG-expressions. In CSG-PL-systems the symbols are distinguished between *variables* and *terminals*. Only the variables are substituted in parallel during a derivation step. The terminals are syntactic elements of a valid CSG expression and are kept in the string. This implies, that for each variable at least one production must exist, which substitutes it with terminals to obtain a valid CSG-expression as final string. Such a production is called *terminating production*, whereas one that defines a recursive derivation is called *generating production*.

Productions in CSG-PL-systems are controlled by parameters. Parameters can be used to describe some geometric properties like branching angles or proportions of the system or to influence the topology of the described object. CSG-PL-systems can be decomposed into three components. The most important part are the productions themselves, which define, how the objects are constructed recursively in the feedback system, that is called derivation. A variable and a set of corresponding productions are called *selection*. Selections are denoted in a selection statement, and allow to obtain a particular production during derivation. Selections are usually controlled by parameters.

Linear transformations are used in the productions to consider geometric aspects. Changing transformations by parameters during derivation allows to modify not only the topology but also the geometry of plants as a result of growth. Transformations can be conceived as an extension to binary CSG-expressions by unary operators. In the following we will denote transformations as a sequence of atomic linear transformations like scaling, rotation, and translation that are applied one after the other.

Calculations are used to modify parameters during derivation and to initialize them before the derivation. Each calculation or initialization is denoted as a sequence of calculation statements, which can be seen as unary operators in the productions scheme.

To make this decomposition of CSG-PL-systems clear, we illustrate it on an example. The following system is capable to generate different branching structures and can be conceived as implementation of a small subset of deReffye's plant model [DERE88].

```
Calculation C_initialize              // Definition of a calculation
{
     cntbush     =    8;              // age (order) of the plant
     1st_phase   =    2;              // time of first growing phase
     trunk_phase =    4;              // phase of trunk growing
     brtyp1      =    0.6;            // probability for pattern 1 vs. 2
     brtyp2      =    0.4;            // probability for bend vs. pattern 2
     beta_br     =    44.0;           // branching angle
}
Transformation T_branch               // Definition of a Transformation
{
     trans  0.0  0.0  1.99;
     rotz   80.0;
     roty   beta_br;
     scale  0.8  0.8  0.8;
}
PL-system bush                        // Specification of the PL-system
{
     Initialization by C_initialize;
     Selection bush                   // The selection statement
     {
        if (cntbush > 1st_phase)
          if (cntbush > trunk_phase)  // build up trunk
            production no 2
          else if (brtyp1 < 0.5)      // build up monopodial pattern
              production no 3
            else                      // build up sympodial pattern
              production no 4
        else if (cntbush > 0)
          if (brtyp2 < 0.5)           // bent segment without branches
            production no 5
          else                        // build up sympodial pattern
            production no 4
        else                          // terminate with a cone
            production no 1
     }
     Production section               // Specification of productions
     {
        1: bush -> T_cone Cone;       // terminating production
                                      // generating productions
        2: bush -> T_cyl Cylinder + C_bush T_trunk bush;
        3: bush -> T_cyl Cylinder + C_bush (T_trunk bush + T_branch bush);
        4: bush -> T_cyl Cylinder + C_bush T_1branch bush +
                                    C_bush T_2branch bush;
        5: bush -> T_cyl Cylinder + C_bush T_bend bush;
     }
}
```

Fig. 1 A CSG-PL-system describing a general branching structure for a bush or tree.

In fig. 1 the three parts of a CSG-PL-system (calculations, transformations, and selections) can be seen. In the selection's part the plant is built up with only one variable (*Bush)* and 5 corresponding productions. The first production terminates a segment with a cone, if the counter *cntbush* is equal to zero. All other productions are generating productions. The second production builds up the trunk in the first phase of growth (*cntbush* greater than *trunk_phase*). Furthermore there are two possible branching patterns that are selected with respect to the parameter *brtyp1*. A monopodial and a sympodial branching structure are built up by the productions 3 and 4 respectively. If *cntbush* gets less than *1st_phase* the second phase of growth is induced. Here the parameter *brtyp2* determines if branches are generated with sympodial branching (production number 4) or if only a bent segment is built up by the fifth production. In the transformations' part an example for the definition of the transformation *T_branch*, which scales, rotates and translates a segment, that is connected as a branch by production 3 is given. The notation of the definition of all other transformations has been omitted to keep the example short. A couple of parameters is initialized in the calculation *C_Initialize*, which determine essentially the topology of the final plant. Only one parameter, namely *beta_br*, is used as branching angle in the transformation *T_branch* and therefore influences the geometry. The other parameters are used in the selection statement *Bush* and control the selection of the branching pattern. Again because of simplicity the description of other calculations has been omitted. This example is only a very limited subset of deReffye's model translated into the notation of CSG-PL-systems, but it is a good test-grammar to examine the application of genetic algorithms to those systems.

4 Applying genetic algorithms to CSG-PL-systems

It is not trivial to find a useful genotype for CSG-PL-systems. Obviously all components contribute to the final shape of a plant. Therefore the naive approach would be to use the whole system description as genotype. Nevertheless it is difficult to define the operations of mutation or mating for productions. We must guarantee that they will still produce valid CSG-expressions. On the other hand we have to deal with the sensitivity of CSG-PL-systems towards the starting conditions. They are dynamic systems, which often react with great deviations on small changes. Even a slight modification of a production can destroy the initial shape of a plant due to its recursive amplification. The same is true for arithmetic expressions in calculations and the ordering of transformations. For a very selective fitness function the approach to take the whole PL-system as genotype might be of use, but such fitness functions are very hard to define. In our approach the user should be able to select individuals out of a generated population. So we need small populations with also small variations among the individuals. To achieve this, only some signified parts of the PL-system can form the genotype.

Thus we apply genetic algorithms to only the numerical components of a CSG-PL-system. These are the initial values of parameters and the arguments of transformations. That means our genotype consists of the numerical parts of calculations and transformations. With this restriction it is easy to use the same methods like Sims. Mutation is done by adding a random value out of a given interval to each parameter or transformation argument with a certain probability. For mating the genotypes of two parents have to be compared. In the general case the underlying CSG-PL-system will be the same for both parents. Therefore it is simple to mate corresponding transformation arguments and parameter initializations. Beside that, we wanted to allow the more general case of two different CSG-PL-systems for the two parents. Therefore we need a scheme to match equivalent structures in the two systems. In different CSG-PL-systems only adjacent group of genes can be mated to make sure that the topological and geometrical semantic of a parameter or a transformation is preserved. We use the simple approach to identify similarities by name. For that purpose a list of all parameters in initializing calculations and a list of all transformations is built up and sorted by name. Now the subsets of all parameters and transformations, which have the same name in both parent systems are taken under consideration. The subsets of parameters can be mated with one of Sims' methods directly to reproduce a new one as described in section 2. Transformations are usually built up by a sequence of atomic linear transformations and the structure of the two parent sequences could be different. Therefore an additional comparison is necessary. Two corresponding transformations are searched for equal combinations of atomic transformations, because only the arguments of those transformations can be mated. The following example shows how this is done:

```
First Parent:
Transformation T_branch
{
    trans  0.0  0.0  1.99
    rotz   80.0
    rotx   alpha_br;
    scale  0.8  0.8  0.8

}
```

```
Second parent:
Transformation T_branch
{
trans     0.0  0.2  1.32
rotz      40.0
roty      beta_br;
scale     0.3  0.3  0.3
flipxy
}
```

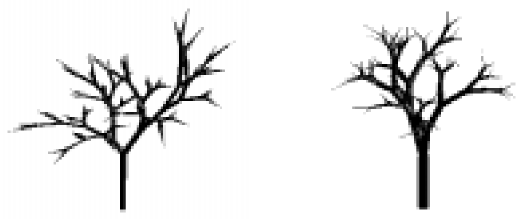
```
Reproduced Transformation:
Transformation T_branch
{
    trans  0.0  0.2  1.99
    rotz   60.0
    roty   beta_br;
    scale  0.3  0.3  0.8
    flipxy
}
```

Fig. 2   Mating of two transformations -- the left parent is defined as dominator

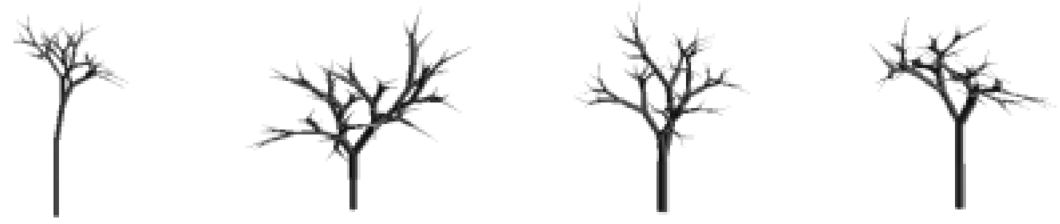
Figure 2 shows the mating of two parent transformations consisting of different atoms. The atoms *trans*, *rotz* and *scale* are the equal parts of both transformations and are mated. For illustration different mating methods were used within this transformation. The translation is mated by crossover, the rotation by linear interpolation and the scaling by random selection. Those parts of the combination, which do not coincide are taken without any changes from the parent that is defined as dominating. In the example the left parent is the dominant one and thus the *roty*-gene survives instead of the *rotx*-gene. For the same reason the *flipxy*-gene is added to the reproduced transformation. In our system the dominant parent can either be selected randomly or by the user.

As we learned out of some tests, it depends strongly on the capability of the underlying CSG-PL-system to generate a wide range of different species or only a few related ones or just one particular. The CSG-PL-system introduced in the last section can create closely related species of tree or bush like structures. Out of that example we can see that much of the success of a genetic algorithm applied to CSG-PL-systems depends on the formulation of the CSG-PL-system itself. If the system is designed to cover a broad range of structures, the evolution will take longer time but the variance of the resulting models will be richer. So this is more a design problem than an algorithmic problem. To design richer CSG-PL-systems two general observations can be made: If we define the structure of the system representing the topology of the plant as general as possible we need just a few different systems to cover many different species. We have to carefully select only relevant parameters for the evolution process. It does not make sense to allow the absolute value of the scaling factor of leafs to be altered, for example, because leafs have to keep their size proportional to the rest of the tree or bush. Following these two ideas, the definition of Fractal plants can be done in an interactive way with non experienced users.

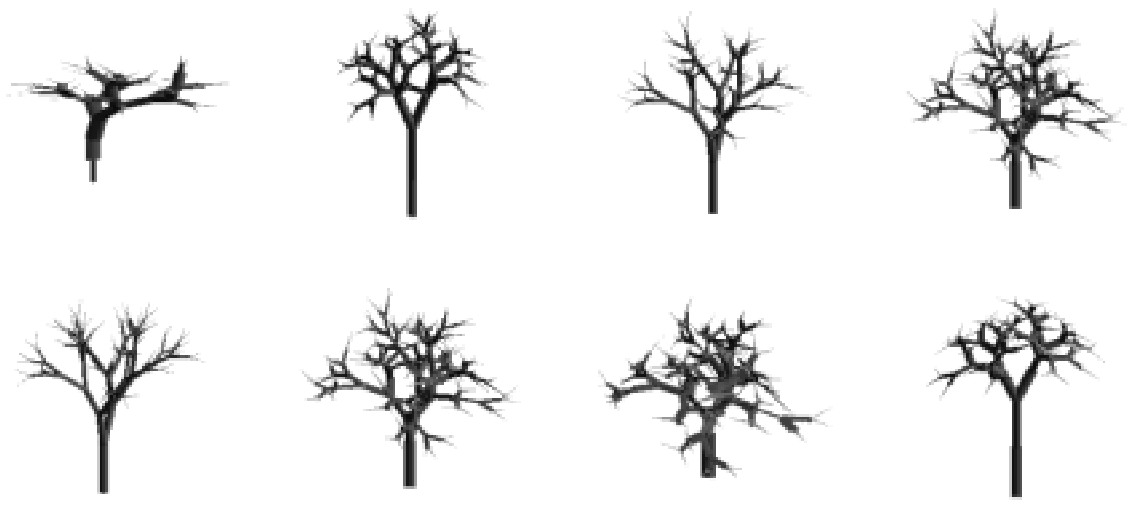
In our tests an amount of 20 individuals has been reproduced for each new generation out of those selected by the user. For the example of figure 1 it was sufficient to calculate not more than 15-20 generations to satisfy the users' ideas. Picture 1 shows some results of the last 3 generations of this CSG-PL-system. Picture 2 is the result of an evolution of a conifer tree. The underlying CSG-PL-system for that kind of tree is much more complicated, because a lot more parameters, transformations and selections are needed to describe the geometry and topology of such a tree.



Two individuals of the last evolved population

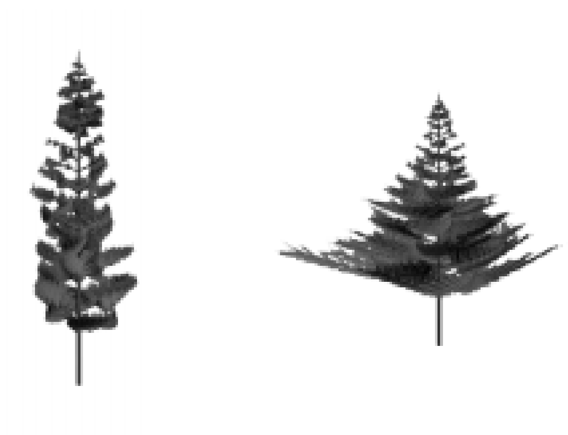


Some individuals selected from the new generation

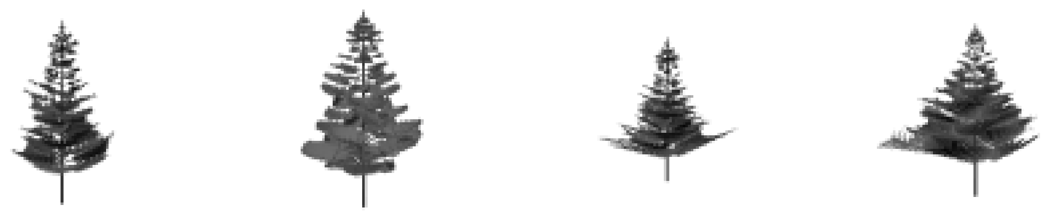


Selected parents of a young generation

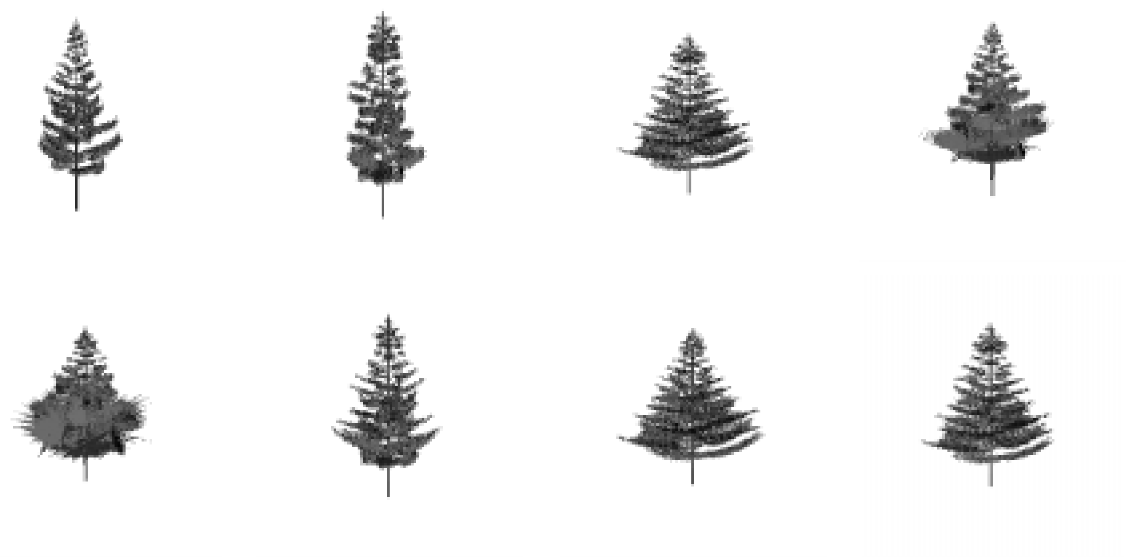
Picture 1: The last 3 generations evolved out of the CSG-PL-system depicted in figure 1.



Two individuals of the last evolved population



Some individuals of the next population



Selected parents of a young generation

Picture 2: The last 3 generations of a CSG-PL-system that generates conifer trees

5 Conclusion and future work

Genetic algorithms are an efficient method to generate new plants by just selecting individuals rather than tuning parameters of CSG-PL-systems tediously by hand. Specifying a CSG-PL-system that describes the common structure of a desired species in an abstract way and then using genetic algorithms to obtain a concrete individual out of this species is an efficient way to specify plants. To define productions that build up some kind of conifer tree for example is not difficult, but it is hard to set all parameters and transformation arguments accordingly to obtain a natural looking tree. We apply the evolution techniques of Sims to only the parameter sets and the transformation arguments of our CSG-PL-systems. This restriction avoids a great spread among the infinite class of plants, that can be generated with that kind of systems. In this way, we are able to generate individuals from a subclass of species. Like in nature, only closely related species can be mated to produce new ones. It is a topic of our future research to define a very general CSG-PL-system to obtain out of one system a large range of different species.

We will try to translate deReffye's model into the notation of CSG-PL-systems completely although this is not an easy task loosing major advantages of CSG-PL-systems. In this way the final shape of a plant will only depend on parameter values and probabilities. Thus we can apply genetic algorithms as explained in the last section hopefully with fast convergence to the users' ideas..

Acknowledgments

This work is supported by the "Fond zur Förderung der wissenschaftlichen Forschung (FWF)", Austria, (project number: P09818). We would like to thank Ferdinand Rödlich for doing the implementation of the genetic algorithms.

References

[BAGL67] Bagley: The behavior of adaptive systems which employ genetic and correlation algorithms, Doctoral Dissertation, University of Michigan, Dissertation Abstracts International 28(12)

[CAVI70] Cavicchio: Adaptive search using simulated evolution, Unpublished doctoral dissertation, University of Michigan, Ann Arbor

[DERE88] DeReffye, Edelin, Francon, Jaeger,. Puech: Plant Models faithful to botanical structure and development, ACM Computer Graphics SIGGRAPH Proc., Vol 22(4), (1988)

[GERV95] Gervautz, Traxler: Representation and Realistic Rendering of Natural Scenes with Cyclic CSG graphs, accepted for publication in Visual Computer, 1995

[GOLD89] Goldberg,D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning, 1989, Addison-Wesley Publishing Co.

[HOLL75] Holland: Adaptation in Natural and Artificial Systems, Ann Arbor, Mi: University of Michigan Press, 1975

[JONG75] De Jong: An Analysis of the behavior of a class of genetic adaptive systems, Doctoral Dissertation, University of Michigan, Dissertation Abstracts International 36(10)

[PRUS90] Prusinkiewicz, Lindenmayer: The algorithmic beauty of plants, Springer Verlag, New York, 1990

[ROSE67] Rosenberg: Simulation of genetic populations with biochemical properties, Doctoral Dissertation, University of Michigan, Dissertation Abstracts International 28(7)

[SIMS91] Sims: Artificial Evolution for Computer Graphics, ACM Computer Graphics SIGGRAPH Proc., Vol 25(4), (1991)

[SMIT84] Smith: Plants, fractals and formal lanquages, ACM Computer Graphics SIGGRAPH Proc., Vol 18(3), (1984)