# Lworld – An Animation System Based on Rewriting

Noser Hansrudi
Multimedia Laboratory
Institute of Computer Technology
Winterthurerstrasse 190
CH-8057 Zurich
Switzerland
noser@ifi.unizh.ch

## Abstract

*Lworld is a computer graphics animation system based on L-systems, a parallel rewriting technique used primarily in computer graphics for plant modeling. Because rule-based programming is a powerful technique, we use it as a basis for a general-purpose animation system called Lworld (L-System world). We describe the architecture, the features, and the programming language of the animation system. Examples of fractal curves, plants, fractal landscapes, group animation, visualization, and evolutionary optimizations illustrate its capabilities. Lworld allows users to create real-time animations as well as raytraced image sequences for further movie production. It is freely available, and runs on PCs.*

## 1 Introduction

There are many excellent modeling, animation, and rendering software such as Maya [1], 3D Studio Max [2], Bryce [3], or the free raytracers POV Ray [4] and Rayshade [5]. Most of them are based upon traditional modeling techniques where designers model the objects interactively by composing them directly piece by piece. Many animations are added by hand or by extensions supporting particle systems or physical modeling.

L-systems, Lindenmayer-systems, L-grammars, or so-called production systems [6] belong to the family of parallel rewriting systems and correspond to another modeling paradigm, the rule-based or rewriting-based modeling. 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.

Traditionally, L-systems [6, 7, 8] model formally plants and fractals. In [9] the authors describe a behavioral animation system running on SGI workstations where production rules define not only growth and topology of objects but also their real-time behavior. In [10] they proposed an extension for modeling distributed virtual reality environments by rules.

Modeling and animation by rules is not wide spread. L-system based software are mostly research tools with a cryptic modeling language, or educational software serving to demonstrate the mechanism of production rules. The work of Prusinkiewicz [6] or Parish [11] shows the inherent potential of rule-based programming by modeling with few rules complex plants and huge cities. Therefore, we developed a general-purpose animation system based on rewriting that should enable a larger public to experience rule-based programming. The software is freely available, small, easily installable, running on most PCs, and contains many features that make it suitable for many tasks in visualization, interactive animation, research, education, and raytraced movie generation.

## 2 Concept of the animation system

As rewriting is a very powerful concept for modeling complex objects, we decided to use this technique not only for modeling plants, but also as the basic principle for a general-purpose animation system. Parallel rules can model not only hierarchical objects, but also their animation, complex behaviors, and interaction. Timed and parametric rules combine in one language the modeling of complete virtual worlds including structured objects, their piecewise-continuous development, their animation, and user interaction.

## 2.1 Modeling and animation by rewriting

In computer graphics, an L-system describes a 3D object by an axiom and a set of production rules, also called its grammar. The axiom and the rules are composed of symbols of the L-system's alphabet. From such an L-system, the computer can derive formal symbol sequences from the axiom by a series of parallel rewriting steps where symbols are replaced according to the rules. Furthermore, the computer can visualize the derived symbolic objects by interpreting them as a kind of turtle-graphics language. This modeling principle is particularly advantageous for complex, structured objects for which we can find its grammar, i.e. the rules for its construction. Such a description is very compact and can provide immense data amplification factors.

Symbols, rules, iteration, and interpretation are key-notions of rewriting techniques. In our system a, parametric symbol of the axiom and the rules is given by a symbol name, a parameter block, and an attribute block containing specific statements like assignments and procedure calls.

$$\text{Symbol } (x_0{=}...;\ ...;\ x_n{=}...;)\ \{f_0{=}...;\ ...;\ f_m{=}...;\}$$

Parameter blocks define formal parameters, named $x_i$ or ${}_p x_i$ if they indicate parameters of parent symbols, needed to pass information from left sides to right sides of rules at iteration. They are only evaluated once when a rule is applied and the symbols are written to the symbolic object. Attribute blocks, however, are evaluated each time the interpreter interprets the symbols. They describe the semantics of the symbols' attributes that are named $f_i$, and can depend on the formal parameters and the age of symbols represented by the parameter $t$.

A rule is composed of a left side, a condition, and a right side:

REPLACE *leftSide* IF *conditon* BY

< *probability rightSide*>*

The left side of the rule is a symbol that is replaced at an iteration step by the rule's right side in the symbolic object if the condition is true. In stochastic L-systems there exists a probability distribution that maps the set of productions for a left side into the set of production probabilities whose sum has to yield one. The right side is a sequence of timed parametric symbols. Iteration, which builds the symbolic objects, and interpretation, which interprets the symbols of the symbolic objects and draws a frame of the animation, can be separated clearly. At each instant of the animation, the whole virtual scene exists only as a symbolic object, and not as scene-graph containing vertices and triangles.

Rules are not limited for modeling the shape and structure of objects. Parametric and timed rules are also particularly suited for defining the growth and development of objects and their behavior. Therefore, a timed parametric L-system can define a complete animation with user interaction. As it is based on a parallel rewriting technique, synchronized parallelism is inherently supported in an animation.

## 2.2 Features

The system we propose supports most features of advanced parallel rewriting techniques, such as timed, parametric, stochastic, bracketed, environmentally sensitive, networked, and behavioral L-systems. At a higher level, we integrated fractal mountains, sound playback, networking, optimization and evolution by genetic algorithms, tropism forces, a particle system, synthetic vision for autonomous actors, as well as a design model for modeling physically correct static trees that visualizes the forces and moments at the tree joints.

We can look from different perspectives to the animation system. From the system's point of view, the application *Lworld* is a virtual machine capable to interpret animation files. At runtime, it reads an animation file, parses it, compiles it into an internal representation of the axiom and the rules, and starts the animation. No separate compilation of an animation file is necessary.

A designer or programmer sees a programming language with a specific syntax in which he can write the rules that model his virtual, animated world. He can model real-time virtual worlds for interactive users, or scenes for raytraced movie generation.

The user or consumer of interactive virtual worlds starts the application by simply clicking on an animation file created by a designer. He interacts with the animation through mouse, keyboard, and screen. Ball games, virtual parks, or visualizations in many domains are typical interactive examples.

## 2.3 Architecture of the animation system

Figure 1 illustrates the architecture of the animation system. At runtime the application parses user defined L-systems and compiles them into an internal representation of the axiom and the rules.

The iterator and interpreter modules are essential parts of the real-time animation loop representing the heart of the animation system given in more detail as pseudo code in Figure 2. At each frame the loop executes first code for frame initialization. Then, the iterator applies all triggering rules to the symbolic object by rewriting the corresponding symbols. The *interpreter* module then executes the resulting symbolic objects that can be considered as a turtle program, and visualizes the frame on the screen. If autonomous actors with synthetic vision have been declared in an animation, the frame is rendered for each actor into their vision window. Finally, the frame is terminated and the global animation time is incremented by the time step.

Optionally, the scene of each frame can be written as a numbered text file in Rayshade format for further raytracing. The calculator module interprets at runtime all numeric expressions of the parameter and attribute blocks of the symbols. The *procedure*-module contains the high-level procedures that implement the semantics of the language's symbols.

The software, written in C/C++ and including the OpenGL/GLUT API, is implemented for PCs and runs on Windows98/NT/2000/XP. It is freely available on the Internet [12].

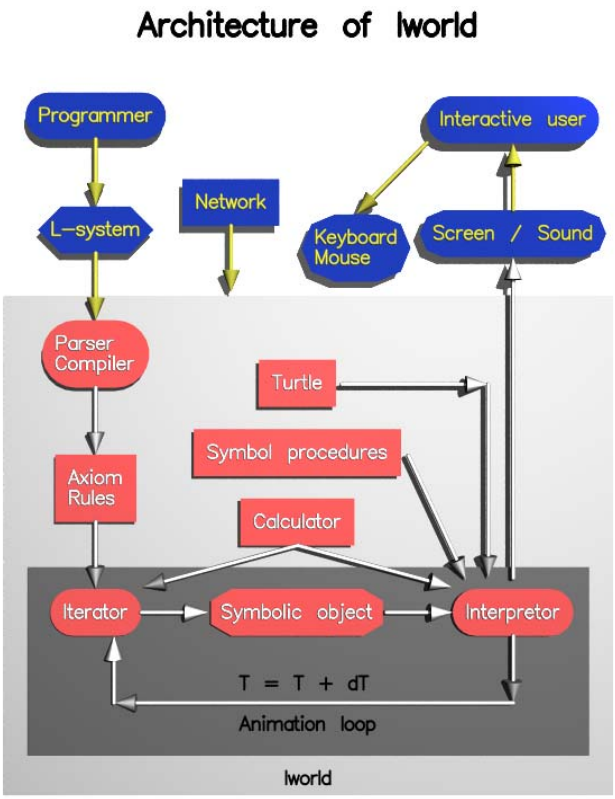


**Figure 1. The architecture of the parallel rewriting-based animation system.**

```
T=0
loop {
   /* frame initialization */
   startTime = clock()
   if (Networking) then read, parse, reset NetBuffer
   integration step for differential equations
   if (Rayshade) then open rayshade file for raytracer

   /* iteration of L-systems */
   for i=0 to NumberOfLsystems do { iterate(i) }

   /* display for user window */
   Place camera (i)
   Place headlight

   for j=0 to NumberOfLsystems do { interprete(j) }

   /* display for autonomous actors */
   for i=0 to NumberOfActors do {
      Place camera of Actor(i)
      for j=0 to NumberOfLsystems do { interprete(j) }
      treat visual input
   }

   /* frame termination */
   if (Rayshade) then close rayshade file and render it
   if (Networking) then send NetBuffer
   T = T + timeStep
   if T > tFinal then terminate program
   workTime = clock() - startTime
   if (workTime < frameTime) then
      wait(frameTime – workTime)
}
```

**Figure 2. Pseudo code of the animation loop.**

## 3 Programming language

Most existing languages for defining L-system rules are somewhat cryptic and difficult to read for new programmers of rules. Therefore, we developed a new rule-language similar to C, which is syntactically familiar to most programmers. The execution model, however, is not sequential and must be carefully studied by designers.

The animation system can be viewed as a virtual machine, defined by some status variables be set by assignments or procedure calls within parameter and attribute blocks of symbols. They define partially the viewing parameters, rendering features, animation control, and physical models. Actually, we implemented more than 80 symbols representing the alphabet of the L-systems. They can be grouped according to their semantic. Some of them manipulate the turtle state or the rendering mode. Others correspond to geometric primitives, sounds, or particular design models such as tropism forces or particles. Others control user interaction, networking, or the interpretation of the symbolic object. The actual set of implemented symbols allows programmers to design a vast variety of virtual scenes such as presented in the example section.

### 3.1 Example code

In this section we present the syntax of the language for defining animations with L-systems by discussing the code that produces the raytraced scene of Figure 4 containing some geometric primitives and binary trees. The general syntax of animation code is partially given by the Extended Backus Naur Form (EBNF) in Figure 3. An animtion is given by the definition of some animation parameters, the declaration of global parameters, and one or several L-systems that are each given by some local variables and a set of rules including the axiom.

```
Animation ::= AnimationParameters Globals Lsystem+
AnimationParameters ::= “ANIMATIONPARAMETERS”
      statement+
Globals ::= “GLOBALS” statement*
Lsystem ::= “DECLARE” statement+ “RULES” Rule+
Rule ::= “REPLACE” LeftSide “IF” Assignment “BY”
      RightSide+
LeftSide ::= SymbolName
SymbolName ::= “Cube” | “Sphere” | “Plane” ...
RightSide ::= < Probability Symbol* >
Probability ::= NumericExpression
```

```
Symbol ::= SymbolName | SymbolName ParameterBlock |
      SymbolName AttributeBlock | SymbolName ParameterBlock
      AttributeBlock
ParameterBlock ::= "(" statement+ ")"
AttributeBlock ::= "{" statement+ "}"
statement ::= Assignment ";" | Function ";"
Function ::= FunctionName"(" Arguments ")"
Arguments ::=(NumericExpression ",")* NumericExpression
Assignment  ::= VariableName "=" NumericExpression | String
```

**Figure 3. The syntax of the language.**

The first two sections of the code in Figure 5, starting with the keywords ANIMATIONPARAMETERS and GLOBALS, serve to define some animation parameters and user defined global variables that are visible to all subsequent L-system definitions.

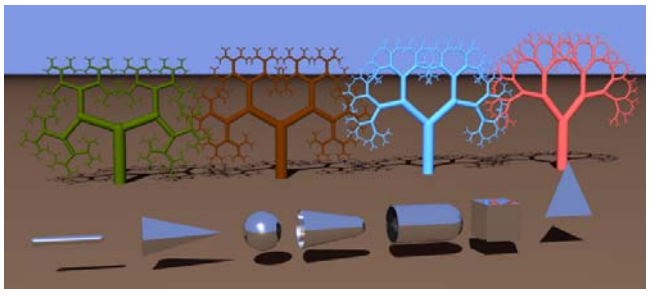

**Figure 4. The raytraced scene of some geometric primitives and binary trees described by the example code.**

```
ANIMATIONPARAMETERS

timestep=0.01;       /* The time step of the animation */
timefinal=10;        /* The end time of the animation */
...

GLOBALS

d=60;                /* branching angle */
```

**Figure 5. The first two sections of an animation file declare and define some viewing and animation parameters, and user-defined global variables.**

An animation file can contain several L-system declarations. Each L-system starts with the DECLARE keyword followed by user defined local variables visible only within the L-system. In the RULES section the programmer can place the rewriting rules.

Figure 6 illustrates the code of the first L-system containing the symbols of some geometric primitives and the call of the second L-system given in Figure 7. Note that the *Do* and *While* symbols represent a loop that is four times iterated during an interpretation step drawing four times the binary tree represented by the second L-system.

```
DECLARE

step=2;
a=45;
b=0.8;
i=0;

RULES                /* Section for rules */
    /* first rule of that replaces the implicit symbol "Axiom" */
REPLACE Axiom IF condition = 1 ; BY

< 1          /* probability of this right side of the rule */
CamMouse             /* The mouse controls the camera */

TurtPush
     /* Initial, absolute placement of the turtle at (f0, f1, f2) */
          TurtMoveAbs {f0=-4; f1=0; f2=0;}
          Line {lighting(0); f0=3;}
          TurtForward {f0=step;}
          /* Material f0 is activated */
          Material {lighting(1); f0=5;}
          Pyramid {f0=4; f1=1; f2=1;}
          TurtForward {f0=step;}
          Sphere {f0=1; f1=1; f2=1;}
          TurtForward {f0=step;}
          Trunk {f0=3; f1=1; f2=0.5;}
          TurtForward {f0=step;}
          Cylinder {f0=3; f1=1; f2=1;}
          TurtForward {f0=step;}
          Cube {f0=1; f1=1; f2=1;}
          TurtForward {f0=step;}

                              /* A triangle */
          PolyPush
          PolyVertex
          TurtMoveRel {f0=3; f1= 0; f2=0;}
          PolyVertex
          TurtMoveRel {f0=-1; f1= 3; f2=0;}
          PolyVertex
          PolyPop
TurtPop

  /* Places and draws 4 times L-system number 1 (this code is L-
                              system number 0) */
Do {i=4;}
          Material {f0=i;d=80-i*10;}
          TurtMoveAbs {f0=(i-1)*15; f1=2; f2=-2;}
          RunLsys {f0=1;}
While {i=i-1; f0=i;}

     /* Places the turtle for drawing the following L-system. */
TurtMoveAbs {f0=0; f1=0; f2=10;}
>
```

**Figure 6. The first L-system containing some basic symbols.**

```
DECLARE

a=3;       /* initial segment length */
b=0.7;     /* segment reduction factor */
```

```
RULES

REPLACE Axiom IF condition = 1; BY
< 1
TurtPush
            TurtPitch {f0=90;} /* The turtle points upwards now */
            S0 (x0=a;)             /* the Symbol for a tree */
TurtPop
>

    /* All 0.3 time units the production is applied as long as the
 parameter of the parent symbol is bigger than 0.5; in other words:
 the tree iteration is stopped when the branch lenght is smaller than
                                        0.5 */

REPLACE S0 IF condition = (px0>0.5) and (t>0.3); BY
< 1

/* The trunk of the binary tree. It grows smoothly according to the
   hermite function. The parameter x0 determines its length and
                                        radius */
Cylinder (x0=px0;) {f0=x0*hermite(t); f1=f0/9; f2=f0/9;}

                /* The first branch of the binary tree */

TurtPush
            TurtPitch {f0=d;}
            S0 (x0=px0*b;)

  /* Recursive placement of the symbol S0 representing the tree.
            The parameter x0 is reduced by the factor b */
TurtPop

              /* The second branch of the binary tree */
TurtPush
            TurtPitch {f0=-d;}
            S0 (x0=px0*b;)
TurtPop
>
```

**Figure 7. The second L-system defining a piecewise continously growing binary tree.**

## 4 Examples

In this section we present some typical applications that one may realize with our rewriting-based animation system. Some of the figures illustrate interactive real-time examples rendered by OpenGL, others are raytraced pictures rendered with the freeware raytracer Rayshade from C. E Kolb, which has been adapted to our needs.

The first example is given in Figure 1. The 3D diagram showing the architecture of the animation system is a raytraced picture of an L-system that uses extensively 3D text and arrow symbols. However, the animation system serves not only to make static diagrams, but also to visualize and animate processes, procedures, and physical behaviors. Figure 8, for instance, is a picture of an animation that illustrates some turtle operations represented by some symbols of the L-system alphabet.

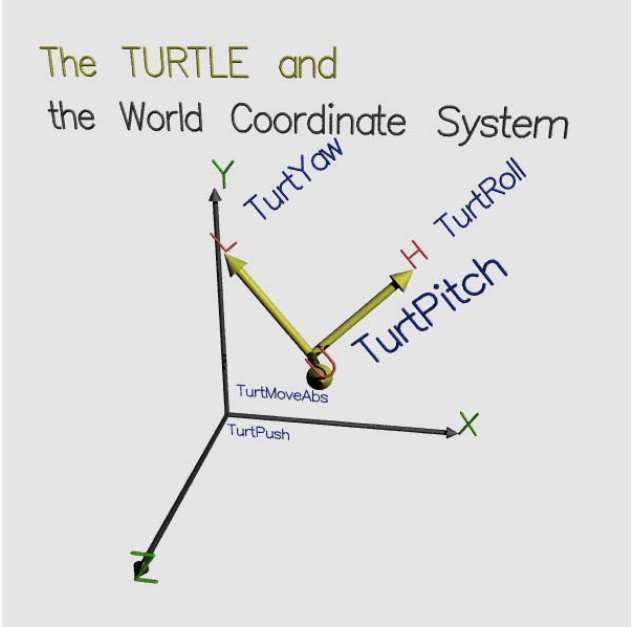


**Figure 8. Animation that illustrates some turtle commands.**

Another type of applications is 3D visualization of functions and physical effects. Figure 9 illustrates the physics of the differential equation of a damped particle fixed by a spring at the origin. This example shows four different types of visualization that can be easily realized by some rules – the visualization of the particles speed by a vector, the visualization of the x-position of the particle by a curve, the real-time animation of the particle in 3D, and finally the textual display of some values. Note that the animation is very dynamic, as all these visualized values are time dependent. Figure 10 shows the forces and moments of the spiral-like static structure. The built in design model of static tree structures visualizes the forces and moments at the joints of any tree structure with a fixed root that is modeled by an L-system.

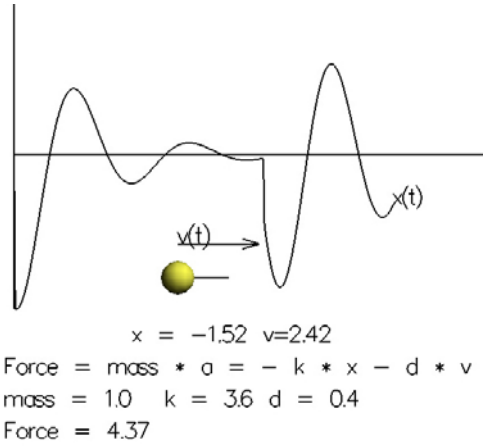


**Figure 9. Visualization of a damped particle movement.**

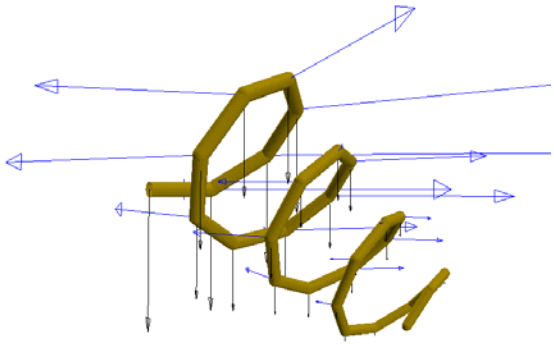

**Figure 10. Visualization of the forces and moments of a static 3D spiral-like structure by vectors.**

Figure 11 illustrates lworld's features in creating complex raytraced landscapes including fractal mountains with thousands of automatically placed growing plants and group animation realized with a force field based particle system. In the example, the butterflies fly around a flower. Differential equations that are defined by the designer in the corresponding animation file determine their individual behavior and are responsible for collision treatment.

**Figure 11. Complex raytraced landscapes with fractal mountains, many automatically placed plants, and a group of butterflies animated by the built-in particle system.**

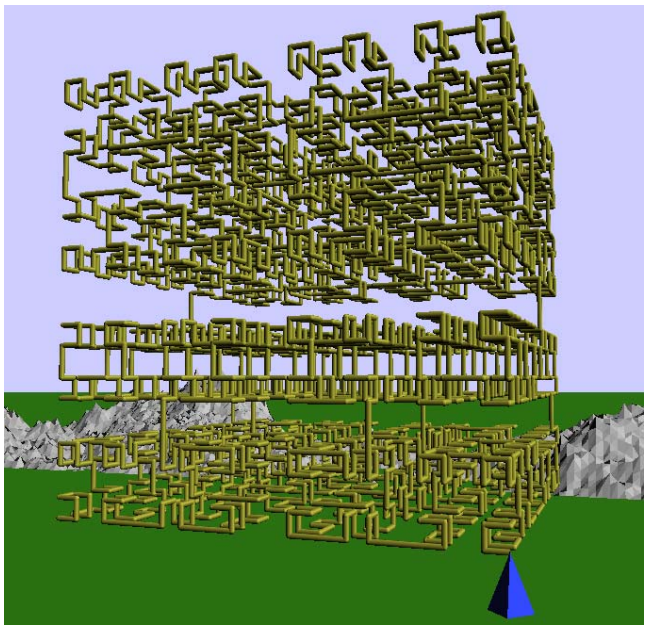

**Figure 12. 3D Hilbert curve in a virtual park (modeled by W. Wellauer [13]).**

An example of an interactive application is a virtual park that can be visited by a user. The visitor can freely navigate around. As soon as he approaches a labeled pedestal the corresponding object, defined by an L-system, starts to grow and its behavior can be watched. When the visitor turns to another object the old one is eliminated from the scene. The rules responsible for this proximity effect are described in [10]. Figure 12 shows a typical fractal object in a virtual park.

Built-in optimization by genetic algorithms (GA) is another feature that enriches the animation system. It supports interactive and automatic evolution of populations producing fit individuals. The first picture of Figure 13 illustrates an interactive example of GA supported design, where the computer proposes a population of flowers to the user. He can assess the flowers by giving fitness values to them. In a subsequent evolution step a new, fitter population is generated. By repeating these steps the flower shapes evolve according to the taste of the user. The second picture illustrates an automatic 2D function optimization. The function value corresponds to the fitness value, and the 2 arguments of the function represent the individual's genes. Users can observe the optimization process embedded in a 3D environment.

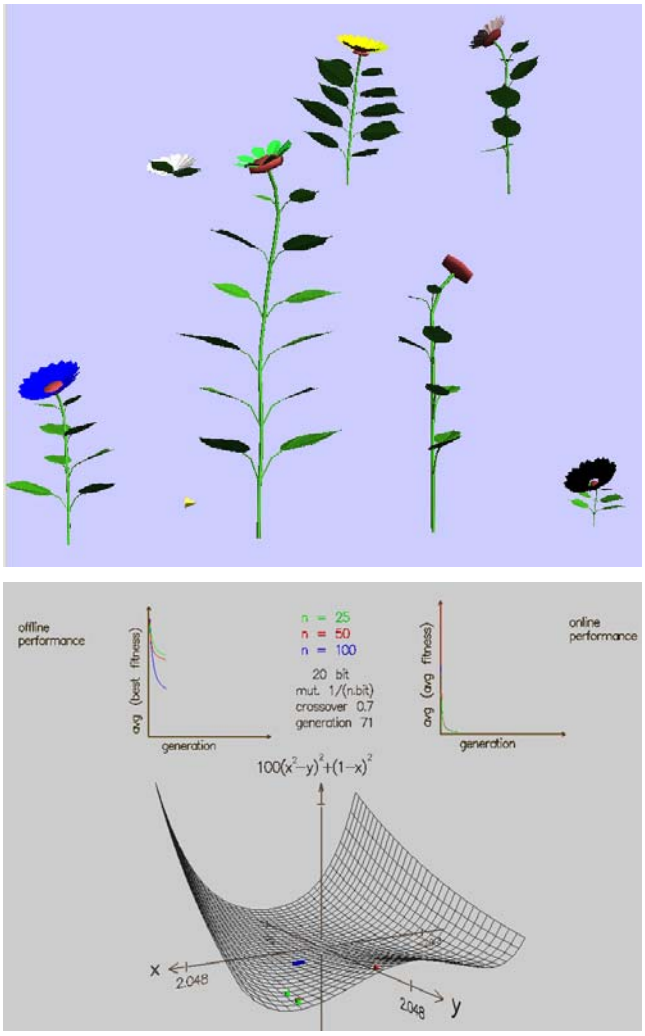


**Figure 13. Built-in optimization by using genetic algorithms. The upper picture shows an interactive, GA supported design of flowers. The lower picture illustrates an automatic function optimization (modeled by W. Wellauer [13]).**

Lworld offers also features for experimenting with autonomous synthetic vision-based actors [14]. Figure 14, for instance, shows a picture from an animation where an actor with synthetic vision looks for the exit of a maze containing impasses and circuits. The behavior and the maze are completely modeled with rules according to [15]. The window of the synthetic vision of the actor is in the lower left corner of the user's window. From this image, rendered from the actor's point of view, it can extract information about obstacles in front of it by using color and z-buffer values of pixels. Lworld supports this kind of test-bed for autonomous actors. The rule-language includes some procedure calls for declaring actors with synthetic vision and for querying color and distance values of pixels.

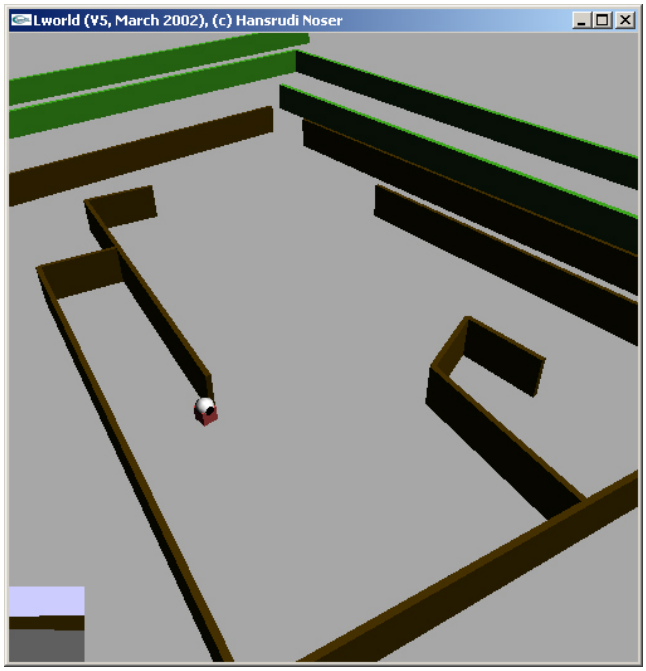


**Figure 14. An autonomous actor with synthetic vision escaping from a maze.**

## 5 Conclusions and future work

We present an animation system based on a parallel rewriting technique. It includes a language for defining and animating virtual worlds. The rule-based paradigm has a large potential for high data amplification resulting in compact code for complex scenes. Additionally, the concept of the language integrates modeling of objects as well as animations and behaviors through parametric and timed symbols with time-dependent attributes. The animation system supports the design of real-time interactive worlds as well as raytraced image sequence generation for high-quality movie productions. As rewriting is not a widespread technique in spite of its unexploited potential, we put a PC demonstration version to everybody's disposal. From our WWW site it can be freely downloaded.

Future work will focus on integration of more physical modeling and Artificial Life (AL) elements, in order to create mutable, interactive, and realistic looking virtual environments for virtual reality applications and synthetic image by image movie generation. We think that there is still a large unexplored potential for virtual worlds modeled, animated, and controlled by rules.

## References

[1] Alias / Wavefront, Maya, http://www.aliaswavefront.com, 2002.

[2] Discreet,3ds max, http://www.discreet.com/products/3dsmax/, 2001.

[3] Corel, Bryce, http://www.corel.com, 2002.

[4] Persistence of Vision, POV-Ray, a free raytracer, http://www.povray.org, 2002.

[5] Craig Kolb, Rayhshade homepage, Stanford Computer Graphics Laboratory, Stanford University, Stanford, CA http://graphics.stanford.edu/~cek/rayshade.

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

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

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

[9] 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.

[10] H. Noser, Ch. Stern, P. Stucki, Distributed Virtual Reality Environments Based on Rewriting Systems, to be published in IEEE Transcations on Visualization and Computer Graphics, 2002.

[11] Y.I.H. Parish, P.Müller, Procedural Modeling of Cities, SIGGRAPH 2001, Conference Proceedings, August 12-17, 2001, pp. 301-308.

[12] H. Noser, Lworld, April 2002, http://www.ifi.unizh.ch/~noser/Lworld/lworld.html,

[13] W. Wellauer, Anwendung genetischer Algorithmen in regelbasierten virtuellen Welten, Master Thesis, Institut für Informatik der Universität Zürich, 2002.

[14] O. Renault, N.M. Thalmann, D. Thalmann, A Vision-based Approach to Behavioral Animation, The Journal of Visualization and Computer Animation, Vol 1, No 1, pp 18-21.

[15] H. Noser, D. Thalmann, The Animation of Autonomous Actors Based on Production Rules, Proceedings Computer Animation ’96, June 3-4, 1996, Geneva Switzerland, IEEE Computer Society Press, Los Alamitos, California, pp 47-57.