ML-System: Generating Complex Geometries using L-Systems

José L. Hidalgo, Emilio Camahort, Francisco J. Abad, and Alejandro Domingo

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
{jhidalgo,camahort,fjabad,adomingo}@dsic.upv.es

Abstract. We introduce Modular L-System (ML-System), a generic rewriting engine designed to derive complex geometries based on C/C++ objects. The advantage of ML-System is that it implements solutions in the same domain as the problem, unlike traditional L-Systems that require changing the representation domain. Our modeling language is highly expressive and can describe complex geometric objects that can be used in game design. We provide some examples of objects generated with our modeling system.

1 Introduction

Each new generation of games requires more realistic renderings. But quality image rendering is not enough. Games should allow the user to experience more complex environments. Most of the effort needed to develop a game is done by artists and designers.

To facilitate modeling, techniques based on procedural generation of geometry can be applied. For example, parametric L-Systems have been used for plants and trees [1], cities [2] and other complex geometries [3].

L-Systems are easy to use, they scale well, and they allow modeling using multiresolution techniques [4]. Additionally, they allow creating geometry on the fly for dynamic game world generation.

We present an extension to L-Systems, Modular L-System (ML-System), that uses general C/C++ objects and data structures to easily and efficiently model objects like the tower shown in Figure 1. The system also employs a scripting language to describe the actions of the derivation rules.

2 Previous Work

Two extensions have been previously proposed to extend L-Systems. The L+C modeling language uses C code instead of rules allowing C code and structures in the rewriting modules [5]. The system requires compiling and linking models before display, thus slowing down interactive design. It is also based on the turtle metaphor and it only allows plant-and-tree modeling.

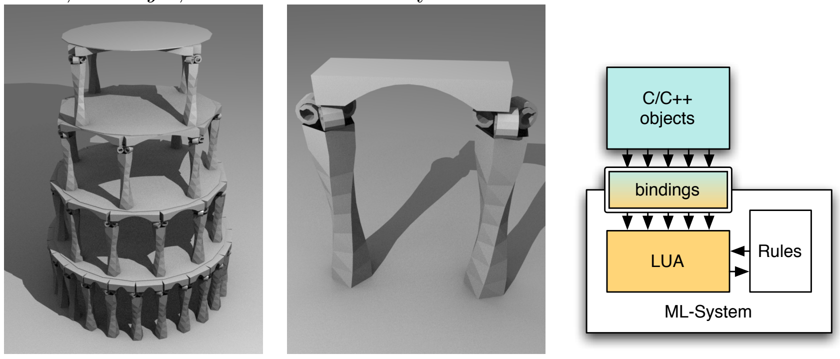
The FL-System associates functions to terminal symbols [6]. Those functions generate VRML code, but they do not provide the expressiveness of L+C Systems. Our goal is to provide a system that can handle general objects with the same expressiveness as L+C Systems and without the shortcomings of offline generation and display.

3 The ML-System

ML-System is a general rewriting engine based on parametric L-Systems. It starts with an axiom and repeatedly applies rules to each symbol in parallel thus obtaining a new derivation chain. It uses the Lua [7] scripting language to describe rule actions. Thus, no recompiling is needed and rules can be changed on the fly.

Figure 1(right) shows how the system works. Given an object written in C/C++ the user declares which methods can be accessed by the engine during derivation. Those methods can be accessed using Lua-C/C++ bindings.

Fig. 1. *Left*, a tower generated with ML-System, *center*, one of the building blocks of the tower, and *right*, architecture of ML-Sytem.



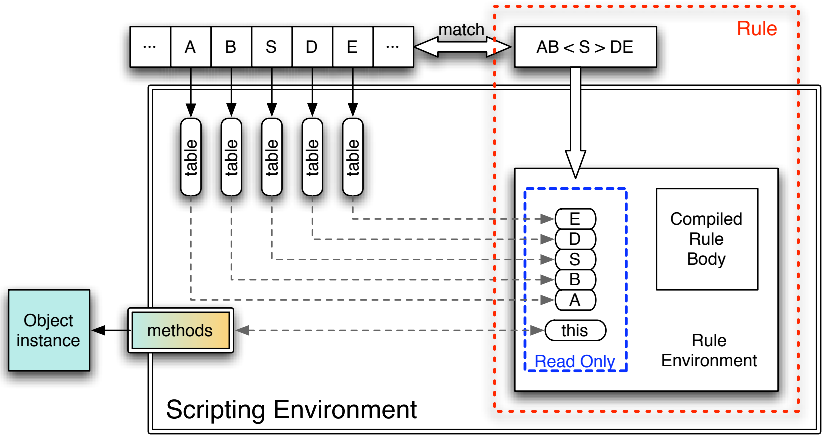
There are two types of rules. Rewriting rules modify the derivation chain without changing the C/C++ object state. Interpretation rules change the objects' state without modifying the derivation chain.

Rewriting rules have a left-hand side (lhs) and a right-hand side (rhs). The lhs has the form "AB < S > DE" where S is the symbol being rewritten and AB and DE are the left and right contexts, respectively. These contexts are optional. To match a rule we compare the lhs's symbol and its contexts.

The rhs of a rule is a pre-compiled chunk of scripting code that computes the output of the rewriting process. Each chunk of code can only access and modify variables within its own scope. It can also access (read only) the global variables, the state of the object associated to the rule and the parameters of the symbols of the lhs. The parameters of the symbols can be any object known to the scripting language: a number, a string, a table, or any other user-defined object.

When a rule matches, the system places in the rule's scope variables with the same name as the symbols of the lhs (see Figure 2). The object associated to the rule (*this*) is also placed in the rule's scope. This allows the rule to call methods of the object to check its state. A rule's rhs can return (i) no value, (ii) *nil*, or (iii) a list of one or more symbols. (i) leaves the derived chain unchanged; (ii) removes the symbol of the lhs; and (iii) replaces the symbol of the lhs with the new list of symbols.

Fig. 2. Components involved in the rewriting process.



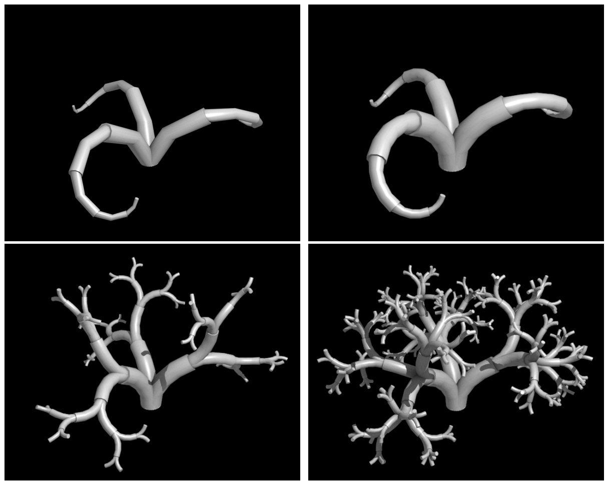
Interpretation rules are executed after each rewriting step. These rules do not modify the derivation chain, they only affect the state of the C/C++ objects. Using a combination of rewriting rules and interpretation rules, ML-System can handle user's objects and modify them.

4 Results and Conclusions

As an example we have developed a C++ extruder object. It has methods to define the shape, the extruding direction, the resolution, the length, the curvature and the cross-section scale factor of each segment. Figure 3 shows four models generated with this extruder object. The model of Figure 1 was also generated with the same extruder using different rules and symbols.

We have presented a new approach that can handle any C/C++ object combined with parametric L-Systems. It allows the user to define L-System rules in the same domain as the application's. In games, this supports generating of geometry on the fly. It can also be used to model other structures such as paths, textures, Artificial Inteligence states, hierarchical models, and higher level objects like buildings, cities and landscapes.

Fig. 3. Four models generated using a C++ extruder object with different recursion depths: *top row* low and high resolution models with $recursion = 1$, and *bottom row* high resolution models with $recursion = 2$ and $recursion = 3$.



Acknowledgements

This work was partially supported by grant TIN2005-08863-C03-01 of the Spanish Ministry of Education and Science, STREP project IST-004363 of the 6th Framework Program of the European Union, and by an R+D support grant of the Universidad Politécnica de Valencia under program PAID-04-06.

References

1. Prusinkiewicz, P., Lindenmayer, A.: The algorithmic beauty of plants. Springer-Verlag, New York (1990)
2. Parish, Y.I.H., Müller, P.: Procedural modeling of cities. In: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press (2001) 301–308
3. Tobler, R.F., Maierhofer, S., Wilkie, A.: Mesh-based parametrized l-systems and generalized subdivision for generating complex geometry. International Journal of Shape Modeling **8**(2) (2002)
4. Lluch, J., Camahort, E., Vivó, R.: Procedural multiresolution for plant and tree rendering. In: AFRIGRAPH '03: Proceedings of the 2nd international conference on Computer graphics, virtual Reality, visualisation and interaction in Africa, New York, NY, USA, ACM Press (2003) 31–38
5. Karwowski, R., Prusinkiewicz, P.: Design and implementation of the l+c modeling language. Electronic Notes in Theoretical Computer Science **86**(2) (2003) 141–159
6. Marvie, J.E., Perret, J., Bouatouch, K.: The fl-system: a functional l-system for procedural geometric modeling. The Visual Computer **21**(5) (2005) 329–339
7. Ierusalimschy, R.: Programming in Lua 2ed. Ingram and Baker & Taylor (March 2006)