PHYSICS-ENHANCED L-SYSTEMS

Hansrudi Noser[1], Stephan Rudolph[2], Peter Stucki[1]

[1]Department of Informatics
University of Zurich, Winterthurerstr. 190
CH-8057 Zurich
Switzerland
noser(stucki)@ifi.unizh.ch, http://www.ifi.unizh.ch/~noser(~stucki)

[2]Department for Statics and Dynamics of Aerospace Structures
University of Stuttgart, Pfaffenwaldring 27
D-70569 Stuttgart
Germany
rudolph@isd.uni-stuttgart, http://www.isd.uni-stuttgart.de/~rudolph

ABSTRACT

In computer graphics and engineering many classes of complex objects can be designed with L-systems. We present a concept for enhancing timed and parametric L-systems with physics. This simplifies considerably the physically correct design of certain classes of computer animations or technical objects modelled by production rules. The focus is on structural extensions in timed and parametric L-system theory necessary for constraint propagation management for the treatment of hierarchical objects and on physics enhanced grammar-language extensions. The proposed concept is illustrated with a design model incorporating the statics of arbitrary tree structures.

Keywords: L-systems, rewriting, physics, computer graphics, design, animation, engineering, conceptual design.

1. INTRODUCTION

In physics- and engineering-based computer graphics simulation, the solution of large systems of equations is common practice. Very often the symbolic solution of large numbers of constraints further complicates the situation. A careful information management becomes necessary in order to guarantee solvability and success. In [YR99] the authors report that the conceptual design phase is rather important in the engineering design process, and despite its importance, this design stage is the least understood and only few supporting tools exist [FRS96]. Approaches for supporting the conceptual Design phase are described in [D94].

The objective of our work is to develop a computer-based framework that supports selected classes of design models in order to improve and reduce the time need in the conceptual engineering design phase. The basic idea is to use rewriting systems for defining such design models. Rewriting systems are synonym for L-systems, L-grammars, or so-called production systems [PL90]. Rewriting is a technique for building complex objects by successively replacing parts of a simple initial object using 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, which can be called a grammar since it describes the structure of the object. From the axiom and the rules - the grammar - the computer can derive in subsequent iterations 3D objects of a given structure (or grammar). Furthermore, by using the concept of turtle graphics, the symbolic objects can be visualised.

Traditionally, L-systems (Lindenmayer-systems) [PL90, PHM93, PJM94] are used for efficient plant and fractal modelling. In [NT99] we describe a behavioural animation system where production rules are not only used to define growth and topology of objects, but also behaviour and

animation of objects in real-time. In this paper we suggest to extend such a behavioural animation system to a physics-enhanced L-system application. Our work is inspired from [RN00] where we described a concept on engineering design generation with XML-based knowledge-enhanced parametric grammars.

The main focus of the project is to extend a timed and parametric L-system application [NT99] in such a way, that it produces not only the geometry but also the corresponding system of equations or constraints that will have to be solved automatically for a given target application.

Rewriting systems are well suited for embedding geometry and physics in the same primitives, and combining them to objects of predefined design models. In particular, they

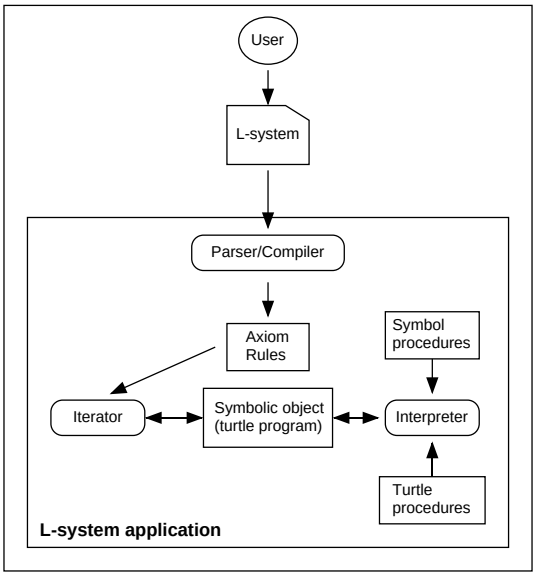
- -support the construction of a *family* of similar objects that assure solvability
- -are well suited for the construction of objects with similar elements such as plants, bridges, aeroplanes, etc.
- -can be used for defining geometry and the corresponding system of equations
- -exhibit a high *data amplification* factor because of their rule-based definition of objects.

2. CONCEPT

In order to enhance timed L-systems with physics, we propose a general concept for incorporating design models. The main idea is to associate not only geometry but also physics to symbols. When the designer uses them in rules for constructing certain classes of objects from a given design model, the application should be able to visualise the object, as well as to compute the correct physics.

In [RN00] we proposed a generic concept supporting the conceptual design phase of engineering application. In a XML based parametric grammar similar to a parametric L-system [PL90], the user can first associate equations to symbols and then use them in rules for defining objects. At iteration and interpretation of the rules the object was visualised, and the application generates automatically the set of equations of the corresponding physical description of the object. These equations can be solved automatically by techniques described in [YR99]. Thus, an essential support of a conceptual design phase for engineering problems is assured. XML was chosen to get a standardised representation of the L-system-like parametric grammar. This new standard simplifies representation, edition, parsing, and internal representation of the grammars in applications that use the existing XML supporting Java packages.

While the described method is best suited for the purpose of conceptual engineering design with human interaction, however, it is not suited for real-time applications based on timed L-systems. In this work we describe how we can use similar principals in timed parametric L-systems suitable for real-time applications. In order to improve speed we tend to focus on particular design models and to implement directly the treatment of physics in the application, instead of maintaining a universal approach suitable for large classes of engineering. We maintain the optional output of the set of equations in order to be compatible with the work of [YR99]. This concept reduces the generality of the application, but it makes it suitable for real-time applications and user friendlier, as most physics can be hidden to designers.



Architecture of the timed and parametric L-system application that is the starting point of our work

Figure 1

The starting point of our work is the real-time L-system application described in [NT99]. Fig.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, for each frame, the iterator iterates the symbolic object, which corresponds to the axiom at the beginning of a simulation, according to the rules. 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 we use is timed, parametric, conditional, stochastic, and evolutionary. 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 environment functions.

Symbols can also have "growth" or "evolution" functions (attributes) that determine the growth or behaviour of the symbol during their existence. Such functions can describe the geometric growth of an object, or the movement of a camera in space, for instance. Growth or evolution 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.

Properties of timed and parametric L-systems have been described here for clarity. In the next sections we propose new features for enhancing L-systems with physics. First, we describe a concept for managing parent-child relationships between relevant parts of a design model. Such relationships are needed for mutual interaction of body-parts. Then, we focus on a concept for introducing particular design models into L-systems.

Relationship Mangement

In most traditional L-system based animation systems the formal graphics object is represented by a linear symbol string (list), which corresponds to the turtle program that visualises something when interpreted. In this formal or symbolic object no parent-child relationships between relevant symbols are explicitly maintained for further use, such as constraint propagation. But in tree-like objects, for example, consisting of rigid rods that are rigidly linked, and that propagate forces and moments to their parent elements, parent-child relationships are needed for computing forces and moments in the tree.

Therefore we need a mechanism in L-system based applications enabling us to control parent-child relationships of certain symbols. One possibility to solve this problem is to implement graph rewriting, where rules and formal objects exist as graph instances. Here we can perform all algorithms and computations directly on these graph instances where pointers link parents and children explicitly. But there exist many symbol-string-like implementations of L-system-based applications. Therefore, we propose an extension of the concept to manage parent-child relations for such types of applications. We need only three new elements, namely:

- a parent-child relationship table (*relTable*)
- a function *getIdOfLeftneighbour(i)*
- a function *isChildOf(parent)*.

The parent-child relationship table *relTable* is a global table maintaining its state from frame to frame during animation. It contains child identifier and parent identifier fields that express our parent-child relationships. We also introduce a function *isChildOf(parent)* that makes an entry into *relTable*. The argument of the function sets the parent identifier field of *relTable*, and the child identifier field is set with the symbol identifier of the symbol in which the function is called. This function is only called in parameter expressions of symbols that are only evaluated once at a derivation step, where also the corresponding symbol identifier is created.

We still need a second function, which is called only in parameter expressions, and which serves to get a symbol identifier. The proposed function *getIdOfLeftneighbour(i)* returns the symbol identifier of the i^{th} left neighbour of the symbol where the function has been called. This function enables us to import symbol identifiers into the formal parameter space of L-systems.

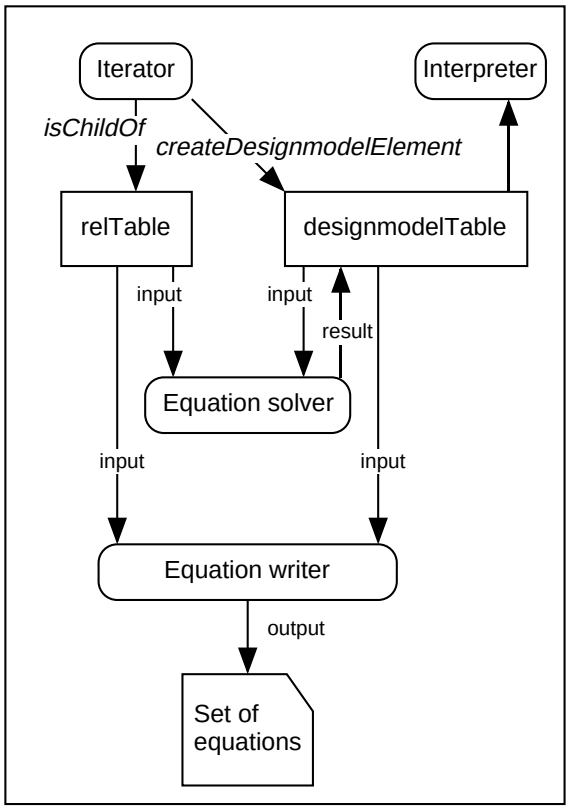
With these three new elements rule designers have now the possibility to define in a very flexible manner parent-child relationships of relevant symbols. An example is shown in section three.

Integration of Design Models

We propose to introduce a design model by implementing five new elements (see Fig.2), namely

- a design model table (*designmodelTable*)
- a design model symbol
- a function *createDesignmodelElement()*
- a design model equation writer
- a design model equation solver.

The design model table defines all the attributes of the elements of a given design model that are necessary to describe their physics. The key field of this table is the symbol identifier.



Architecture of extensions
Figure 2

The alphabet of the L-system application has to be extended by one ore several design model symbols, which enable designers to set certain physical attributes of the corresponding design model. These user-defined attributes correspond to growth or evolution functions of the symbols and can depend on time.

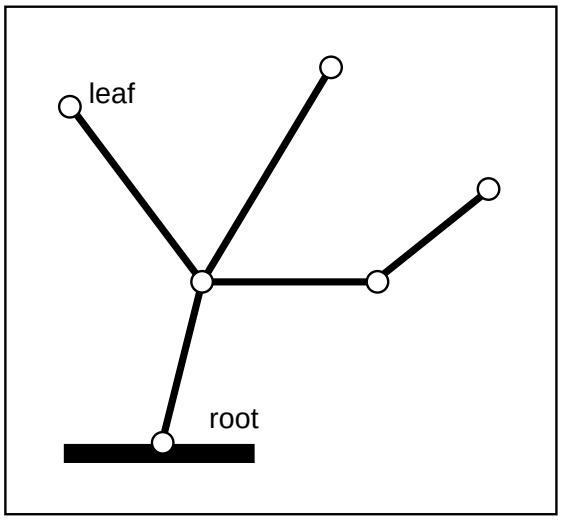
To add a new element to *designmodelTable,* we use the function *createDesignmodelElement()*, which can be called in parameter expressions of the design model symbol.

The design model equation writer is used to output the system of equations describing the physics of the design model. This procedure needs the relationship table and the design model table as input. It can be called at the end of an interpretation step at each frame.

The design model equation solver computes the solution of the actual set of equations describing the state of the on-going animation. As input it uses *relTable* and *desginModelTable*. It updates all relevant fields of *designmodelTable* that can be used by symbols for visualisation in a subsequent interpretation step.

3. EXAMPLE OF A DESIGN MODEL

This section illustrates the proposed concept of enhancing timed and parametric L-systems by realising a particular design model, namely the statics of arbitrary rigid tree constructs with a fixed root and free children (see Fig.3). For any tree structure and applied external, time dependent forces that are designed with rules, the statics will be computed automatically by the application. Moreover, the geometry, the resulting forces and moments at the joints are displayed at each frame.



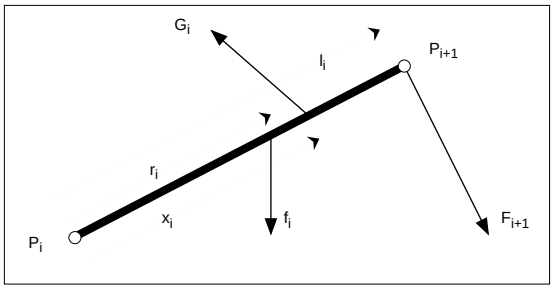
Design model of arbitrary 3D tree structures with a fixed root and free leaves
Figure 3

This simple example is well suited to illustrate our proposed approach. First, we describe the physical model of the design model. Then, we focus on the structure of the particular design model table *treeTab*

and the algorithms of the equation solver and the optional equation writer. Finally, we discuss an L-system representing a binary tree.

Design model of Rigid Trees

An element of our tree structure is a rigid rod with rigid joints propagating forces and moments to the root (see Fig.4).



P_i: position vector of the root side part
P_{i+1}: position vector of the child side
m_i: mass density of the element
f_i: Force vector caused by the mass of the element and gravity acting at r_i
F_{i+1}: Force vector propagated by the child and acting at l_i
G_i: External force acting on the element at the position x_i
computed attributes: $l_i = P_{i+1} - P_i$, $r_i = 0.5 * l_i$

A primitive rod element of a tree structure
Figure 4

Eq. 1 shows the force acting on a joint of a rod element in the tree. It contains the sum of the contribution of all children, the external force, and its own contribution caused by gravity and its mass.

$$F_i = f_i + G_i + \sum_{k=childOfNode_i} F_k \quad (1)$$

The moment, which acts on the joint and which is propagated to the parent, is given by Eq. 2.

$$M_i = r_i \times f_i + x_i \times G_i + \sum_{k=childOfNode_i} l_i \times F_k + \sum_{k=childOfNode_i} M_k \quad (2)$$

Note that Eq. 1 and Eq. 2 contain the contributions of their children that are again parents of their children, etc. Therefore, when solving the resulting system of equations, all interactions are propagated from the leaves to the root. This means, that this model is physically correct. The root, for instance, will see all the contributions of the whole tree elements.

According to Fig.4, we need the following data fields for our tree design model table (*treeTab*):

symbolId:	The identifier of the symbol
massDensity:	The mass density of the symbol
M[3]:	The moment vector
F[3] :	The child force vector
G[3] :	The external force vector
l[3]:	The rod unit vector
len:	The rod length
P[3]:	The rod position vector
r :	The attack point for gravity
x[3] :	The attack point for G

As a next step we introduce the design model symbol called *rod*. Its evolution functions determine the attributes – length, radius, mass-density, x, external force -, that a designer can define. The start position P_i of a rod is given by the current position of the turtle. Its direction is determined by the turtle's heading vector. When interpreted, first the evolution functions are evaluated in order to update the corresponding attributes in the design model table *treeTab*. Then, the rod, the external force, and the computed moment and joint force are drawn.

The equation solver has to compute the forces and moments according to Eq. 1 and Eq. 2. Every parent can have multiple children. Therefore, the forces and moments for the rod elements can be calculated in reverse sense by starting from leaves and going back to the root. A possible solution is to mark first all lines in the relationship table with the topological distance of the children from the root. Then, in a second step we can calculate all the forces and moments starting from the highest distances down to the root. The code of Fig.5 accomplishes this task. It is executed after a complete iteration and interpretation step.

```
resetDistanceMarks(relTable, 0)
maxLength = markWithDistance(relTable)
for i = maxLength downto 0
    for all relTable.childId with
    i=relTable.distanceMark
      calculate force and moment with all child
      contributions
```

Pseudo code of the equation solver
Figure 5

After an interpretation step we can also write the system of equations in a text file for further external processing. Fig.6 illustrates the pseudo code of the equation writer. Variables are indexed by the corresponding symbol-identifier. Parts of the equations that can be evaluated are calculated

directly. The algorithm of Fig. 5 outputs the equations. These equations can also be solved automatically by constraint propagation techniques described in [YR99], which are used in conceptual design in engineering. For our real-time application, however, it is more efficient to implement directly our proposed numerical solution of Fig.5.

```
for all rod elements el of treeTab do {
    /* computation of numerical values */
    i = el.symbolId;
    gravity =
    (0,el.massDensity*el.len*el.radius²*Pi*9.91, 0);
    F1 = gravity + el.G;
    M1 = el.r × gravity + el.x × el.G;

    /* The output strings are generated and initialized */
    String F = "F_" + i + " =  " + F1;
    String M = "M_" + i + " = " + M1;
    /* The child contributions are added as variables */
    for all childs j = childOfParent(relTable, i);
      F = F + "F_" + j;
      M = M + "+" + el.r + " × F_" + j + " + M_"+ j;
    }
    print F;
    print M;
}
```

Code of the equation writer that produces a text file with the set of equations describing the statics of a tree structure

Figure 6

L-system of a Binary Tree

The L-system of a binary tree without external forces is given by the pseudo code of Fig.7. The axiom consists of a germ-like symbol *z* with three parameters that correspond to the maximal number of iterations, the root identifier (-1), and the initial length (5) of a rod element passed by the unique rule to its right side symbols. Note, that the parameter x_1 is used to propagate the parent-identifier to its children. The function *getIdOfLeftNeighbour(i)* puts the parent-identifier into the parameter space of the L-system.

```
Constants
a = 30 /* rotation angle */
b = 0.8 /* rod length reduction factor */
Axiom
z (x0=4, x1=-1, x2=5)

Rule1: z is replaced if (t>maxAge) and (x0>0) by

rod ( x0=isChildOf(x1),
      x1=createTreeTableElement(),
      x2=x2,
      length=x2, radius=1, massDensity=1)

push
rot (angle = a)
z ( x0 = x0-1,
    x1=getIdOfLeftNeighbour(3),
    x2 = x2*b)
pop

push
rot (angle = -a)
z ( x0 = x0-1,
    x1=getIdOfLeftNeighbour(7),
    x2 = x2*b)
pop
```

Pseudo code of the L-system of a binary tree

Figure 7

Let us now apply the iterator on this L-system and look at the produced symbolic objects. Fig.8 shows two iterations of the axiom. At iteration the right side of the unique rule replaces the dummy symbol *z* of the current symbolic object. For each right-side symbol a unique identifier is created that indexes it in the symbolic object.

Axiom

z_1 (x_0=5, x_1=-1, x_2=5)

First iteration

rod_2(x_0=isChildOf(-1), x_1=createTreeTableEl(), x_2=5,
length=5, radius=1, massDensity=1)
$push_3$ rot_4(angle=30) z_5 (x_0=4, x_1=2, x_2=4) pop_6
$push_7$ rot_8 (angle=-30) z_9 (x_0=4, x_1=2, x_2=4) pop_{10}

Second iteration

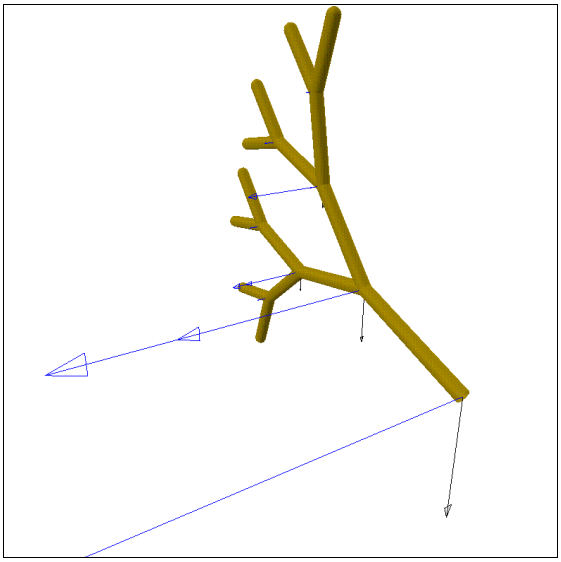
rod_2(x_0=isChildOf(-1), x_1=createTreeTableEl(), x_2=5,
length=5, radius=1, massDensity=1)
$push_3$ rot_4(angle=30)

rod_{11}(x_0=isChildOf(-1), x_1=createTreeTableEl(), x_2=4,
length=5, radius=1, massDensity=1)
$push_{12}$ rot_{13}(angle=30) z_{14} (x_0=3, x_1=11, x_2=3.2) pop_{15}
$push_{16}$ rot_{17}(angle=-30) z_{18} (x_0=3, x_1=11, x_2=3.2) pop_{19}
pop_6

$push_7$ rot_8(angle=-30)
rod_{20}(x_0=isChildOf(-1), x_1=createTreeTableEl(), x_2=4,
length=5, radius=1, massDensity=1)
$push_{21}$ rot_{22}(angle=30) z_{23} (x_0=3, x_1=20, x_2=3.2) pop_{24}
$push_{25}$ rot_{26} (angle=-30) z_{27} (x_0=3, x_1=20, x_2=3.2) pop_{28}
pop_{10}

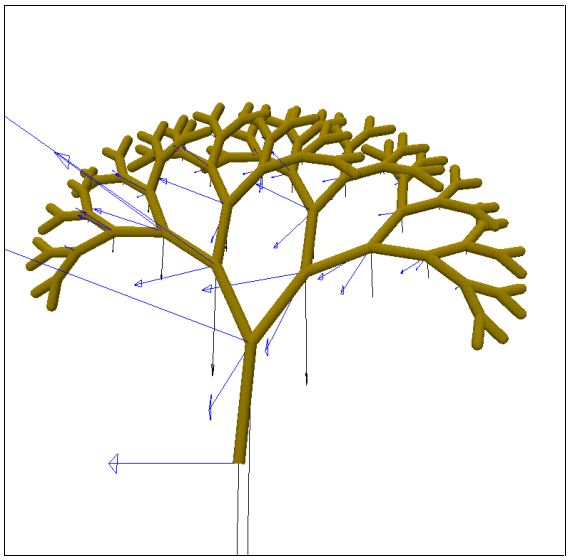
The symbolic objects of two iterations of the axiom

Figure 8



Visualization of the formal object, the forces, and the moments after four iterations of the axiom

Figure 9

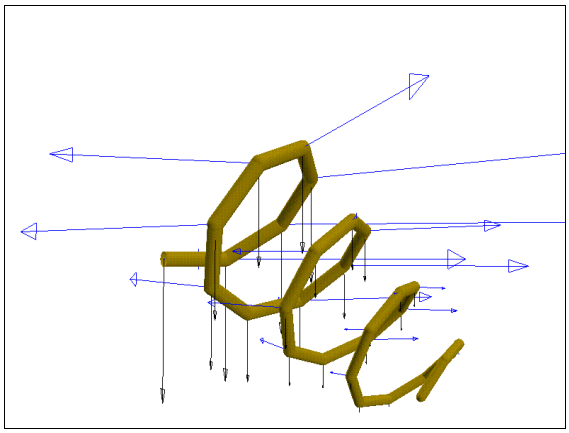


A non-flat binary tree with visualized forces and moments

Figure 10

According to the rule and the parameter x_0 of the symbol *z* the L-system is iterated four times. Fig. 9 illustrates the visualised tree object after four iterations. The interpreter of the L-system application (Figure 1) is responsible for the visualisation of the symbolic object (Figure 8) at a given animation time. It interprets each symbol according to its semantics and its parameters. The rod symbol draws not only a cylinder in the turtle's coordinate system, but also the force and moment vectors at its root-side joint. The symbol can access the computed resulting force and moment in the design model table (treeTab) by its unique symbol identifier. Please note that this design model only treats the statics of rigid tree structures, and therefore, the effects of forces and moments cannot be visualised as deformations or movements. A design model of the dynamics of tree structures is much more complex. Its implementation is left to future work.

The forces and moments caused by gravity are drawn as vectors starting from the joints of the rod elements.



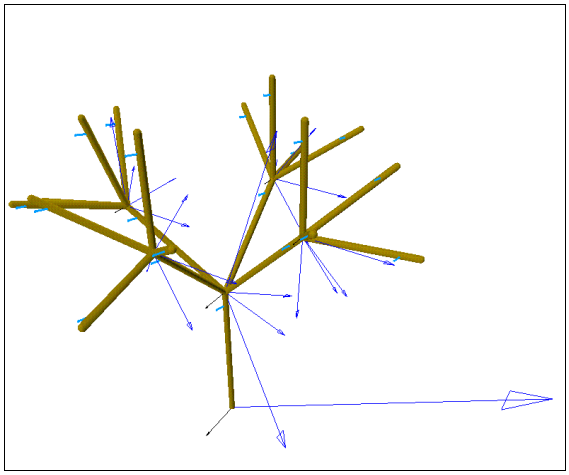
A spiral composed of rod elements

Figure 11

The Fig.10 to 12 show other examples of the static tree design model with one fixed root and free leaves. Each figure is defined by an appropriate L-system, containing rod symbols of the static-tree-design-model. A detailed description of all of these L-systems is beyond the scope of this paper. The examples serve only to illustrate the fact that, once a design model is implemented, it is possible to investigate, visualise, and analyse a multitude of instances of a given design model.

In all figures gravity is acting in vertical direction. Only in Fig.12 there are external forces that are visualised as thick vectors. They are attacking at each rod element of the quad tree.

As a last illustration of the descriptive power of L-systems, Figure 13 shows the L-system of the spiral-like object of Figure 11 consisting of 29 linked rod elements. The only rule *Rule1* adds at each time unit a rod element and adjusts the turtle by two rotations in order to get a spiral-like shape of the final object.



A tree with four branches at each node. External forces attack at each rod element

Figure 12

Axiom
z (x_0=29, x_1=-1, x_2=5)

Rule1: z is replaced if (t>1) and (x_0>0) by

rod (x_0=isChildOf(x_1),
x_1=createTreeTableElement(),
x_2=x_2,
length=x_2, radius=x_2/9, massDensity=3)

rotUp (angle =37)
pitchDown (angle = 28)
z (x_0 = x_0-1,
x_1=getIdOfLeftNeighbour(3),
x_2 = x_2 * 0.9)

L-system of the spiral-like object that is illustrated in Figure 11. This object consists of 29 linked rod elements. The spiral-like shape is obtained by the two turtle rotations rotUp and pitchDown after each rod element.

Figure 13

4. CONCLUSIONS

It is a visionary fact that the tight coupling of physics- and engineering-based simulation tasks with computer-graphics visualisation procedures bears a high potential in the field of advanced conceptual system design and development. The presented work aims in this direction. It shows how to extend real-time L-system based applications in such a way, that they produce not only geometry but also the corresponding system of equations that are solved automatically for a given target application. We illustrate our proposed concept with a simple design model dealing with the statics of arbitrary trees.

Future work will focus on the integration of more design models such as the dynamics of trees, particle systems, and most especially, design models containing general parent-child relationships with cycles as they are found in most technical constructions.

REFERENCES

[D94] Dym C. L., *Engineering Design: A Synthesis of Views*, Cambridge University Press, 1994.

[FRS96] Fertig K. W., Reddy Y. S., Smith D. E., *Constraint Management Methodology for Conceptual Design Tradeoff Studies*, Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference, August 18-22, Irvine, California, USA, August 1996.

[NT99] 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.

[PHM93] P. Prusinkiewicz, M.S. Hammel, E. Mjolsness, *Animation of Plant Development*, Computer Graphics Proceedings, SIGGRAPH '93, Annual Conference Series, ACM Press, pp. 351, 1993.

[PJM94] P. Prusinkiewicz, M. James, R. Mech, *Synthetic Topiary* , SIGGRAPH 94, Computer Graphics Proceedings, Annual Conference Series, pp. 351-358, 1994.

[PL90] P. Prusinkiewicz, A. Lindenmayer, *The Algorithmic Beauty of Plants*, Springer Verlag, 1990.

[RN00] S. Rudolph, H. Noser, *On Engineering Design Generation with XML-Based Knowledge-Enhanced Grammars*, Proceedings IFIP WG5.2 Workshop on Knowledge Intensive CAD (KIC-4), Parma, Italy, May 22-24, 2000.

[YR99] Yusan, H. and Rudolph, S., *A Study of Constraint Management Integration into the Conceptual Design Phase*, Proceedings 25th Design Automation Conference, Las Vegas, NV, September 12-15, 1999.