EUROGRAPHICS 2011 / M. Chen and O. Deussen
(Guest Editors)

Volume 30 (2011), Number 2

Guided Procedural Modeling

B. Beneš[†1] and O.Šťava[1] and R.Měch[2] and G.Miller[2]

[1]Purdue University [2]Adobe Inc.

Abstract

Procedural methods present one of the most powerful techniques for authoring a vast variety of computer graphics models. However, their massive applicability is hindered by the lack of control and a low predictability of the results. In the classical procedural modeling pipeline, the user usually defines a set of rules, executes the procedural system, and by examining the results attempts to infer what should be changed in the system definition in order to achieve the desired output. We present guided procedural modeling, a new approach that allows a high level of top-down control by breaking the system into smaller building blocks that communicate. In our work we generalize the concept of the environment. The user creates a set of guides. Each guide defines a region in which a specific procedural model operates. These guides are connected by a set of links that serve for message passing between the procedural models attached to each guide. The entire model consists of a set of guides with procedural models, a graph representing their connection, and the method in which the guides interact. The modeling process is performed by modifying each of the described elements. The user can control the high-level description by editing the guides or manipulate the low-level description by changing the procedural rules. Changing the connectivity allows the user to create new complex forms in an easy and intuitive way. We show several examples of procedural structures, including an ornamental pattern, a street layout, a bridge, and a model of trees. We also demonstrate interactive examples for quick and intuitive editing using physics-based mass-spring system.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing Algorithms

1. Introduction

Procedural modeling means generating content by a procedure or a program, and it is one of the most exciting areas of computer graphics because of the intriguing concept of *database amplification* [Smi84]. However, for nearly twenty years procedural models were used only in their classical areas, such as plants, noise generation [Per85], particle systems, or fractals [EMP*03]. In the past few years, procedural models have undergone a little renaissance and have found a way into new areas. They are used in urban modeling [ARB07, MWH*06, WWSR03], and can be combined with physics [BCNG10, WOD09], sketching [APS09, IMIM08], and animation [BCNG10].

Among the most important problems of procedural models that hinder their real-life application is the gap between the model description and the control of its execution. The description is usually cryptic, is difficult to understand, and involves complicated relations between the model elements, therefore the steps performed when the system is executed are usually beyond imagination. The resulting interaction with a procedural system is thus in the trial-and-error loop.

Our key observation is that *the concept of environment can be generalized*. A complex procedural model does not need to be described as a whole, and it can be divided into simpler blocks that work in parallel, have no direct influence on each other, but can communicate by using an exogenous mechanism, such as Open L-systems.

We propose a solution to the problem of low controllability of the procedural models by defining *guided procedural models*. The key idea is to divide the space into *guides* – separate geometrical objects with closed boundaries. Each guide contains one procedural model. Procedural systems inside two separate guides cannot interact directly. Guides are connected by *links* that serve as message passing mechanism. When a procedural model inside one guide touches the link, the message is sent to the corresponding guide and

† e-mail:bbenes@purdue.edu

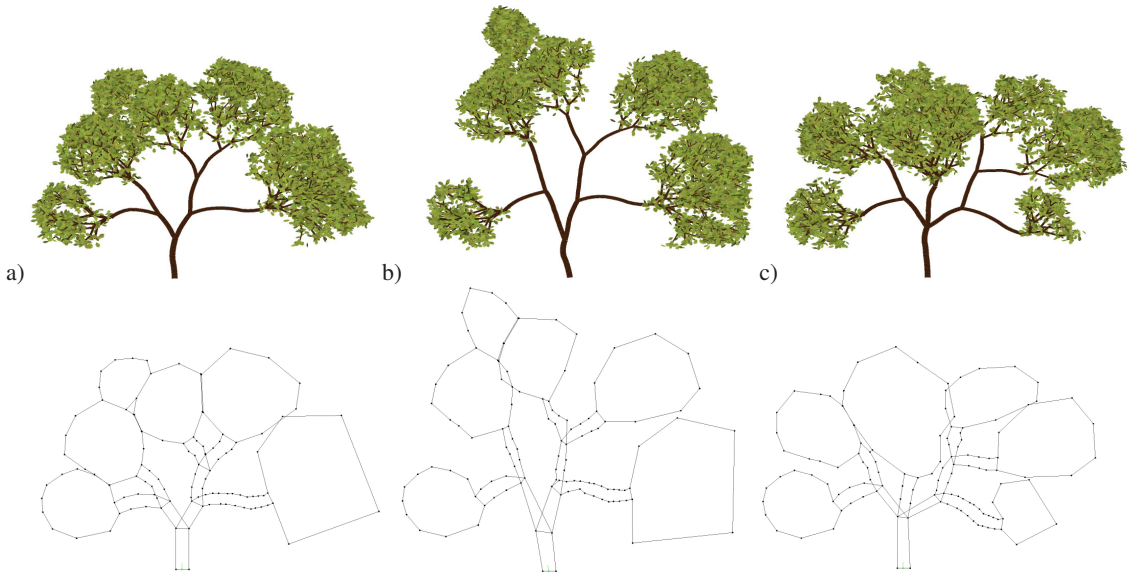

Figure 1: *Guided procedural system used to generate and edit a procedural model of a tree. a) The original tree is generated, b) its guides (lower row) are interactively edited using mass-spring model, and c) some guides are erased and edited individually.*

another procedural model is executed. The main advantage of this control is that the user can handle the overall shape of the entire model by manipulating the guides, whereas the local changes are handled on the detailed level of procedural systems. In this way, we allow for "top down" iterative modeling, where first the overall layout is defined, and then the details are added. Another advantage is the *simplicity* of the editing. Instead of manipulating the procedural system as a whole, we manipulate small blocks separately. Guides can be connected using a mass-spring system as shown in the editing sequence Figure 1.

Our paper continues with a description of the previous work. The method is discussed in Section 3, which is followed by a description of interactions with the system. Implementation and results are in Section 5, and the paper concludes with Section 6, which also discusses future work.

2. Previous Work

One of the first areas of procedural models, and probably the most advanced one, includes biological modeling. In his seminal work [Lin68], Lindenmayer introduced parallel string rewriting systems for a description of cellular subdivision and endogenous information exchange between cells. The Lindenmayer systems (L-systems) were extended in many different directions, from which the most important present geometrical interpretations of the string letters and the introduction of special symbols allowing higher linear topological structures via branching [Pru86]. Various extensions exist, and most of them are summarized in the book [PL90]. Recent work include Open L-systems that allow for exogenous information exchange via query modules [PJM94,MP96]. A feedback system, where an L-system can detect positional information and modify it by an interpretation of the rules was introduced in [PMKL01] and multiset L-systems were used to describe the communication of plant ecosystems in [DHL*98]. Metropolis procedural modeling was introduced in [TLL*10]. L-systems share common problems of procedural systems, the most relevant to our work being the problem of their controllability.

L-systems are linear as they describe the connection of one and/or more symbols in a consecutive way. Shape grammars [SG71] deal with connecting of 2-D elements and they define the replacement or the connections of shapes. The concept of shape grammars was recently extended to applications in urban modeling, where the split grammars [WWSR03] were introduced, but the problem of controllability is not solved.

Procedural models can be found in the works that contribute to computer vision and computer graphics. Müller *et al.* described a user-assisted system for façade description in [MZWG07]. Fit grammars were applied to urban model reconstruction from images and point clouds in [HKHF09]. and Vanegas *et al.* matched a predefined set of rules to reconstruct buildings bottom-up from their footprints [VAB10].

Procedural urban modeling has seen an increasing interest in the computer graphics community since a paper [PM01] where the authors used Open L-systems to describe the street and road layout that was completed with 3-D models of buildings. Recently Müler *et al.* [MWH*06] introduced CGA - a grammar-based description of buildings.

The procedural generation of street layouts was described in [CEW*07], and a procedural generation of roads, tunnels, and bridges was described in [GPMG10]. The controllability of procedural models is leveraged by user interaction, but in general, the user has limited control over the final results. In [GPMG10], user-sketched graphs of influences define global behavior of the system; local control however, is not available. The interactive editing of procedural rules in [LWW08] exploits higher levels of user interaction and allows for an immediate visual feedback of the results.

Ijiri *et al.* introduced a sketching interface to a procedural model in [IOI06]. A parametric procedural rule has the actual values of its formal parameters set by a user sketch. This allows for a limited level of control, because the actual modeling space is defined by a single rule. Chen *et al.* [CNX*08] used Markov processes for sketch-based tree modeling and surface trees for sketch-based procedural surface modeling were introduced in [SS08].

Recently, several attempts to combine physically-based modeling and procedural systems have appeared. Arvin and House [AH02] used a mass-spring model to create architectural design with a high level of control over a floor layout. This idea was an inspiration for our system, however, we also provide a low-level control by means of embedded procedural systems. Moreover, our guides can communicate and do not act independently. Baxter *et al.* [BCNG10] used procedurally built models of plants to interact with physics-based simulations. Procedural models were directly combined with physics in [WOD09]. Including physics in the procedural modeling solves many problems, providing, for example, realistic and feasible results, but it imposes an extra level of complexity on the creator of the procedural system.

An important open problem of inverse procedural modeling was addressed by [vBM*10], where the authors found a complete L-system for a 2-D vector input. Similarly, [BWS10] generates a procedural description of a 3-D point cloud. In the context of urban models [ARB07] reconstructs floor rules from photographs.

3. System Overview

The guided procedural model consists of three basic elements: the *guides* that are closed shapes, the *procedural systems* inside the guides, and *links* that connect the guides and provide communication among them. Without loss of generality our guides are closed planar polygons. Our procedural systems are Open L-systems [MP96] that allow for a wide variety of geometric structures and include an advanced control for exogenous information exchange.

3.1. Guides

Let's denote the set of guides $G = \{g_1, g_2, \ldots, g_{|G|}\}$ and its individual elements g_i (see Figure 2). Guide g_i has its shape defined by k_i vertices $v^i_1, v^i_2, \ldots, v^i_{k_i}$ connected with edges $e^i_1, e^i_2, \ldots, e^i_{k_i}$. A set of oriented *links* is denoted by $L = \{l_{ij}\}$ and we say that the link l_{ij} starts in g_i and ends in g_j. The guides and the links form an oriented graph $H = \langle G, L \rangle$ with vertices G and oriented edges L.

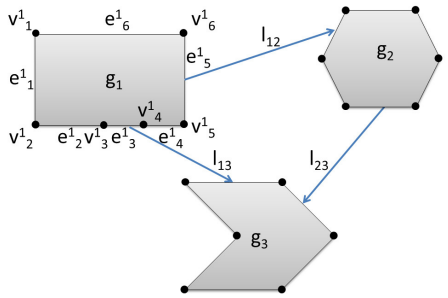

Figure 2: *The guided procedural system is a collection of guides connected with oriented links. Each guide hosts a procedural system.*

The links define topological connectivity of the guides. However, each guide also stores geometrical information about the edges from which the links originate and end. For example, guide g_1 in Figure 2 has two outgoing links: l_{12} starts on edge e^1_5 and l_{13} starts on edge e^1_3. Correspondingly, guides g_2 and g_3 store information about the links that connect them with the other guides. The finer subdivision of the lower side of guide g_1 to edges e^1_2, e^1_3, and e^1_4 constrains the position of the link on the guide. To provide a wide variety of shapes and a global control over the editing process, we can also define geometric relations between guides. We can snap guides together; we can define an exact location of the connection, or we can manipulate the guides using a mass-spring system.

3.2. Open L-systems

An Open L-system is a parallel string rewriting system and is defined as a tuple $P = \langle M, \omega, \Re \rangle$,where M is the L-system alphabet that contains elements $A(p_1, p_2, \ldots, p_n)$ called modules. Modules consist of the letters $A, B, \ldots$ and their parameters $p_1, p_2, \ldots p_n \in R$. The symbol $\omega \in M^+$ (M^+ denotes the reflexive closure) is a non-empty initial string of modules called the axiom. A set of productions (rules) is denoted by $\Re$. The rules have form

$$label : A(p_0, p_1, \ldots, p_n) : cond \rightarrow A^*,$$

where *label* is the rule identifier, *cond* is a boolean expression, and A^* denotes the reflexive-transitive closure i.e., the list of all strings including the empty string ε. The symbol $\rightarrow$ denotes rewriting of the module $A(p_0, p_1, \ldots, p_n)$ with the string on the right side of the rule. The *epsilon rule* erases the symbol $A(p_0, p_1, \ldots, p_n)$:

$$r_\varepsilon : A(p_0, p_1, \ldots, p_n) \rightarrow \varepsilon. \quad (1)$$

The *query module* has form $?D(p_0, p_1, \ldots, p_{|P|})$ and it sets the parameters p_i by values obtained from the environment. An example is module $?D(d)$ that measures distance to an obstacle. The actual semantics of each query module is defined by the designer of the L-system. The list of query modules used in our paper is in Table 2.

Modules are interpreted geometrically by a turtle [Pru86] as described by Table 1 The turtle has a position $[x, y]$ and a heading angle ϕ that define the turtle's state (x, y, ϕ) and the *seed* of an L-system is a tuple

$$[(x, y, \phi), P = \langle M, \omega, \Re \rangle].$$

module	interpretation
$F(\Delta)$	draw a line of length Δ in the heading dir.
$f(\Delta)$	move in the direction of heading by Δ
$+(\delta)$	rotate by δ to the left
$-(\delta)$	rotate by δ to the right
[	push the state on the stack
]	pop the state from the stack, move to $[x, y]$, and head in the direction ϕ

Table 1: *Turtle interpretation of the modules.*

3.3. Open L-systems and Guides

The geometry produced by the L-system can be limited by the guide. An element of the L-system can be clipped either partially if its part is outside the guide, or entirely, if its center is outside. We use the latter solution which allows for a partial overlap with the guide as can be seen in Figure 4. Two or more guides may overlap. However, the embedded L-systems cannot communicate directly, because the guides and their L-systems are permitted to communicate only via message passing as described below.

We assign at most one L-system to each guide g_i. This could seem to be a limiting factor, however, any pair of L-systems $P_1 = \langle M_1, \omega_1, \Re_1 \rangle$ and $P_2 = \langle M_2, \omega_2, \Re_2 \rangle$ can be converted into one by

$$P = \langle M_1 \cup M_2, \omega, \Re_1 \cup \Re_2 \cup \{\omega \rightarrow [\omega_1][\omega_2]\} \rangle .$$

We have merged both alphabets and rules, and we have created a new axiom ω. We have also augmented the rules by a new one that rewrites the new axiom to the sequence of the old axioms $\omega \rightarrow [\omega_1][\omega_2]$.

3.3.1. Seeds and Query Modules

Each guide has a set of *seeds* that define the initial status of the turtle (its position and heading) and that store the axiom ω of an L-system. Multiple seeds are concurrently expanded by the set of rules from the defining L-system because all the rules are applied in parallel. Seeds can be created in two different ways, either interactively by the user, or by receiving a message from another guide via a link.

Our Open L-system has three query modules that allow the guides to communicate (Table 2). First, a query module $?D_L$ returns in its parameter d the distance to the closest edge of the guide g_i that contains link l_{ij} to another guide g_j. Second, a query module $?N_L$ returns the distance to the closest link in the direction of the actual turtle heading. Third, a query module $?X$ detects when an L-system crosses the edge that contains a link and returns its ID. It is also used for collision detection.

The modules $?D$ and $?N$ are used to navigate the L-system to the edge that stores the link that sends a message to another guide. The number of steps in which an edge is found is directed by the user because it depends on the definition of the L-system. It can be found in many different ways, for example deterministically in a single step or by a random walk of the turtle where the result is not guaranteed.

After a link is intersected the module $?X$ informs the environment (the guide g_i) that it should send a message to another guide as described below.

3.3.2. Message Passing

Sending a Message To provide communication between guides, we enhance each link by a set of *tokens*, a concept borrowed from Petri Nets [Pet77]. A token is stored on the edge of the start guide g_i. When a communication is required, a token is converted into a message that is sent to the end guide g_j associated with the link l_{ij}. Two conditions must be satisfied to send the message: (1) a token is available on the link and, (2) the module $?X$ of the L-system touches the edge that contains a link. When the message is sent the number of tokens on the edge is decreased.

Multiple modules can concurrently compete for a token. For this problem to be solved, a token on a link can be in one of three states: *free, reserved,* or *used*. A free token is available for any module. However, once a module decides to use the token, it will change its state to *reserved* which prohibits other modules from using it. If the module that reserved the token does not reach it, the state is changed back to *free*.

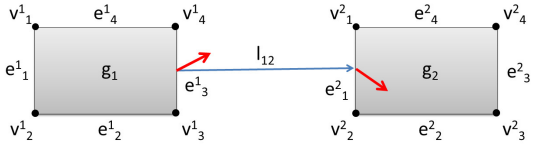

Figure 3: *Link l_{ij} between guides g_1 and g_2. The arrows indicate the incident angle under which the L-system in g_1 approaches the edge of the guide and the growth direction of the new L-system (turtle heading) in guide g_2.*

The above-described mechanism for linking the guides is topological. The geometry of the connection is defined by the user when the link is defined. The incident direction of the turtle, the vector the L-system attempts to match, is given when the link between the two guides is defined as shown in

module	input	output	description
$?D_L$, $?D_G$, $?D_A$	id_{test}	$found$, d, β, id_{out}	Distance to a link. An optional input is an id id_{test}. Parameter d is the distance to the closest link, guide, or element of the given id_{test} (or any if id_{test} is not set), the angle β to the closest point, and the id_{out} of the closest object. If no such element exists, the parameter $found$ is set to 0.
$?N_L$, $?N_G$, $?N_A$	id_{test}	$found$, d, id_{out}	Forward distance. An optional input is an id id_{test}. Parameter d is the distance to the closest link, guide, or element of the given id_{test} (or any if id_{test} is not set) in the turtle direction, and the id_{out} of the closest object. If no such element exists, the parameter $found$ is set to 0.
$?X$	Δ, id_{in}, id_{test}	$found$, id_{out}	Collision detection. The input is the length Δ of an element to be tested for collision. Optionally, an id id_{in} can be associated with the object. The collision test can be restricted to an object with an id id_{test}. The parameter $found$ is set to 1 if there is a collision, in which case the id id_{out} is set to the id of the intersecting element, guide, or a link.

Table 2: *Communication modules of our Open L-systems. An input parameter not listed is passed through unchanged.*

Figure 3. Note that the parameters of the module X as it intersects the link are available in the seed module in the end guide. This way the L-system can send additional information through the link.

Receiving a Message When the end guide g_j receives a message, it seeds an L-system on the receiving edge. The seeded L-system is defined by the user when the system is created simply by dragging the L-system from the menu to the guide. The exact location of the seed on the edge is defined by the intersection point on the starting edge, and the heading direction is defined by the definition of the link as depicted in Figure 3. In this way we can make the transition between edges of two guides C^0 or C^1 continuous.

The condition of seeding the L-system on an exact location of the edge is not restrictive because the L-system can perform a linear transformation by executing the rule

$$A \rightarrow +(\alpha)f(\Delta).$$

3.3.3. L-systems Execution

Let's recall that the guided procedural model is stored as an oriented graph $H = \langle G, L \rangle$ with vertices G and oriented edges L. There are two kinds of seeds possible in each guide. *Explicit* seeds are defined by the user, and *implicit* seeds result from a message passed from another guide.

We create a set of active L-systems that are being executed, which we denote by L_e. This set is initialized by the explicitly defined seeds. We execute all the L-systems in the guides from L_e in parallel using a technique similar to [LWW09]. If the message-passing module $?X$ touches an edge, we add the generated seed into the L_e and execute it immediately. This process repeats until L_e is not empty. Because the number of tokens is finite, the procedure will finish in finite time. New seeds can be created while another L-system is already being executed.

3.3.4. L-systems Update

The L-systems in a guide may require recalculation. This can happen when an interactive operation with the guide occurs, if there is a change of the structure of links, or if it is requested manually by the user. When the update of a guide is required, we perform a local incremental update of the guide and its children. We first erase all of the L-systems in the guide, and reset the explicit seeds as well as the tokens on the edges. The L-systems of all included seeds are then recalculated in parallel, regardless of whether they are implicit or explicit. Messages are sent to all dependent guides and their L-systems are recalculated in the same way.

The communication overhead is negligible compared to the time required to recalculate the L-systems. However, only a part of the model is recalculated, so our approach is faster than calculating the entire model from the scratch as would be the case of a single procedural model.

4. Interaction

The main objective of user interaction is a minimal, intuitive, easy-to use way creation and manipulation of guides. The L-systems can be created beforehand, stored as XML files, loaded at the beginning of the execution, and available from a drag-and-drop menu. We have created various procedural models that we show in Section 5 and on the video.

4.1. Guides and Links Definition and Editing

Creating a guide is done by entering the vertices of the input polygon. The seeds of the L-systems are then defined interactively as a position and orientation, and the L-system is attached from a menu. Once the guides are defined, their links are created by clicking inside a guide and crossing the corresponding edge. This operation is done twice, first for the start guide, then for the end guide. The number of associated tokens can be set from a menu.

Our system allows interactive translation, rotation, and

scaling of the guides. These operations can be performed before the L-system execution for fine-tuning the shapes. If they are applied after the L-system execution, the guides are updated using the mechanism described in Section 3.3.4.

Manual manipulation of guides can be tedious if a large change is required. To simplify this, our system also allows for higher-level operations with guides. Guides can be attached to each other by use of a snap operation.

4.2. Mass-Spring Editing

Another feature that allows for high-level guide editing is a physics-based mass-spring system [WB97]. Vertices of the guides are treated as particles with mass, and the edges as springs with the resting length set to their initial lengths. Selected vertices of each guide are also connected by diagonal springs to better preserve the shape of the guides. When two guides are snapped together, their particles and springs are shared on the overlapping vertices and edges. Guides that are not located next to each other may require a hard link that does not change. This is provided by user-defined stiff springs between the guides. In this way two links will not move relatively to each other when edited.

When editing the guide, we select a particle and we constrain its relative position to the position of the mouse. The positions of the remaining particles are then obtained from a solution of the mass-spring system.

5. Implementation and Results

We have implemented our system in C++ with the support for OpenGL visualization. All examples were generated on an Intel® Xeon® CPU E5530 quad core running at 2.4GHz. The computer is equipped with 12GB of memory and NVIDIA® GTX 480 GPU with 1.5GB of memory.

The creation of L-systems in the examples presented below took 30 minutes to 1 h for each, based on their complexity. Writing an L-system is still a difficult task, however, each L-system is created and edited separately, and it can use the links and guide edges to guide the growth in a desired direction. Thus this task is significantly simplified as compared to the editing of a large L-system, in which the overall shape and local behavior are hard to control. The creation of the guides and their connection took less than 3 minutes. The fine-tuning of the guides communication took less than 10 minutes in the most complicated case in Section 5.4.

The L-systems are stored in an XML file and each is about 10-30 lines long. The most complicated L-system has 11 rules. The response of the system depends on the rules, the number of seeds, the number of concurrently running L-systems, and the number of their elements. If a large number of elements is used and a large number of collisions is checked, the system slows down as it did in the example in Figure 7, where more than 85,000 elements were used and the generation of the model took 45 seconds. For most of the other examples, this took less than one second. The actual bottleneck of the application is the collision detection, so we use kd-trees to speed-up the collision detection. On the other hand, collisions are checked only within the elements inside each guide and not with all the elements in the scene, therefor the guides implicitly simplify the collision detection.

5.1. Palm Tree

The example in Figure 4 shows guided growth, where multiple guides with simple L-systems and intuitive connections are used to create the structure.

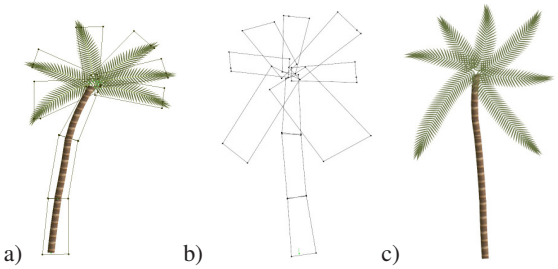

Figure 4: *A structure generated by two simple L-systems (trunk and leaves) and a set of guides. a) The original guide layout and the generated structure. b) The new structure is created by modifying the guides. c) The resulting structure is generated by re-running the same procedural model.*

Figure 4 a) shows the set of guides and the generated structure, and Figure 4 b) shows the modified guides. We find that the palm shape can be easily controlled top-down by changing the shape of the guide without actually modifying the underlying L-systems. A similar structure could be generated bottom-up by a single L-system; however, making the local changes would require complicated modification of the rules. The single L-system would also be more complicated. The rules that produce our structure are rather simple. The set of rules for the trunk guides is:

$$
\begin{aligned}
&(1) \quad ?seed(pos,\alpha) \rightarrow +(\alpha)[?D_L(\Delta)] \\
&(2) \quad ?D_L(found,\beta,\Delta) : found \rightarrow \\
&\qquad +(\beta * 0.5)?X(\Delta)f(\Delta)?D_L(\Delta) \\
&(3) \quad ?D_L(found) \rightarrow \varepsilon \\
&(4) \quad ?X(found,\Delta) : found \rightarrow F(\Delta)cut \\
&(5) \quad ?X(found,\Delta) \rightarrow F(\Delta)
\end{aligned}
$$

The L-system grows a palm trunk from the seed point toward the link to the next guide. The first rule rotates the turtle by the angle α given by the seed and inserts the query symbol $?D_L$ which queries the direction to a link. The symbol $?D_L$ has a parameter Δ that stores the desired length of a segment. The parameter $found$ of the symbol $?D_L$ is set to one if a link is found. In this case, rule (2) creates a new symbol $?X$

rotated towards the link by a half the angle β between the direction to the link and the current turtle orientation. Rule (3) stops growth if a link is not found. The parameter *found* of the symbol $?X$ is set to one if there is a collision with the guide. In this case rule (4) is applied, the growth stops, the trunk segment $F(\Delta)$ of the length Δ is inserted, and the rest of the string is cut. A symbol that intersects a guide initiates the link and thus the seed in the linked guide. Rule (5) places a trunk segment F if there is no collision.

Once the growth reaches guides for compound leaves, the second L-system is executed in each of these guides. The set of rules for the compound leaf guides is

(1) $?seed(pos, \alpha_{LINK}) \rightarrow +(\alpha)[?D_L(\Delta)]$

(2) $?D_L(found, d, \beta, \Delta) : found \rightarrow$
$+(\beta)?X(\Delta, d)f(\Delta)?D_L(\Delta)$

(3) $?D_L(found) \rightarrow \varepsilon$

(4) $?X(found) : found \rightarrow cut$

(5) $?X(found, \Delta, d) \rightarrow F(\Delta)L(d, \Delta * 5)$

(6) $L(d, \Delta) : d > 0.1 \rightarrow [+(60)E(\Delta)][-(60)E(\Delta)]$

(7) $L(d, \Delta) \rightarrow [+(600 * d)E(\Delta)][-(600 * d)E(\Delta)]$

Rules (1-5) have a similar function as in the previous L-system. In rule (2) the distance d to the link, returned by symbol $?D_L$, is stored with the symbol $?X$. Rule (5) creates an additional module L that puts leaf blades E on the branch, using rules (6-7). Rule (6) places two opposite blades at 60^o angle, as long as the distance d to the link is above 0.1. Rule (7) is applied for $d \leq 0.1$ and it gradually reduces the blade angle from 60^o to zero as the stem approaches the link. Rules (2-5) act as global environment sensor. As depicted in Figure 4, the L-system tends to grow to the link on the opposite side of the guide. However, this link has no token and therefore produces no messages. It is used only as a navigation of the L-system growth. The runtime of the model was 0.11 second with 481 elements in the final image.

5.2. Spirals

A more complex example in Figure 5 shows a spiral that initiates lateral spirals in the attached guides. The spiral in the main guide is generated by an L-system that has 11 rules, but the core functionality is captured by the following two:

(1) $?N_A(found, d, \Delta) : found\ AND\ d > \Delta * 10 \rightarrow$
$+(15 * \Delta_0/d)?X(\Delta)f(\Delta)?N_A(\Delta * 0.995)$

(2) $?N_A(found, d, \Delta) : found\ AND\ d > \Delta_0/3 \rightarrow$
$+(50 * \Delta_0/d)?X(\Delta)f(\Delta)?N_A(\Delta * 0.995)$

The tip of the spiral senses the distance to the nearest guide boundary or the nearest element in the guide in the current turtle direction using the symbol $?N_A$. The growth direction is changed by an angle that is inversely proportional to the returned distance d. Moreover, when the spiral gets too close to another element the angle increases in rule (2). The new segment is shorter by a factor of 0.995. If a collision is detected or the segment becomes shorter than the third of the initial length Δ_0 the growth stops.

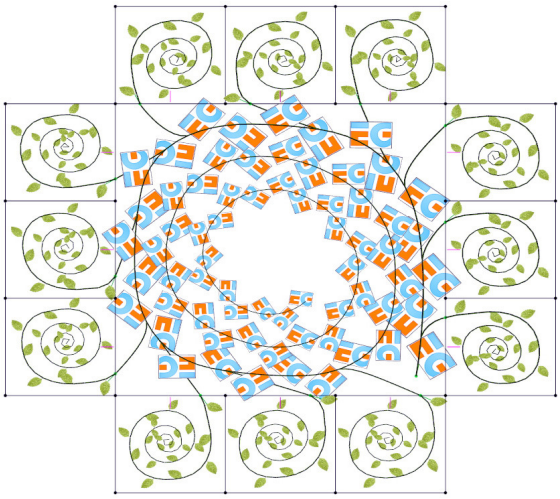

Figure 5: *A spiral generated by two simple L-systems and a set of guides. The main spiral spawns a lateral branch when it sees the proximity of a link. When the lateral branch touches the link, it sends a message to the end guide that generates another spiral.*

Other rules are used to sense the closest edge that contains a link. If the edge is within its proximity, it sends a lateral branch to it. When the branch reaches the edge, it sends a message to the other guide which generates another spiral. A feedback loop mechanism is also implemented to avoid multiple branches reaching one edge. Once a branch is sent to an edge, no more branches are generated to the link with the same ID by marking the token as *reserved*.

The lateral guides have a simplified version of the same algorithm for spiral generation that does not provide the mechanism for sending lateral branches. The second L-system has only 7 rules. The total time to generate this example was 8 seconds with 1,900 elements in the scene.

5.3. Dinosaur

The dinosaur from Figure 6 was inspired by [MP96] and has been created by two L-systems. One was used for legs, the neck, and the tail. It grows segments that follow the guide, similarly to the mechanism captured by the two rules in the previous section that are modifying the growth direction based on the distance to an edge. The leaves in the body, the head, and the tail tip are created at random positions, and their rotations and colors are also randomized.

Figure 6: *The overall layout of this structure is controlled by the guides (upper image), and the inner structure is generated by two simple L-systems.*

This is an example, in which the guided procedural modeling enabled the modeler to capture the result by using much simpler L-systems compared to a complicated L-system of a branching structure that would try to grow into the dinosaur's neck and tail. The entire structure was generated in 2 seconds and contains more than 10,000 elements.

5.4. Urban Layout

Figure 7 shows an image of an automatically created urban layout. This example was inspired by the paper [PM01] where the authors used Open L-systems to describe a street-and-road layout that was completed with 3-D models of buildings. The layout has five guides, and the execution starts in the middle with the star like street layout that sends a large number of tokens over the edges to the neighboring guides that create regular patterns.

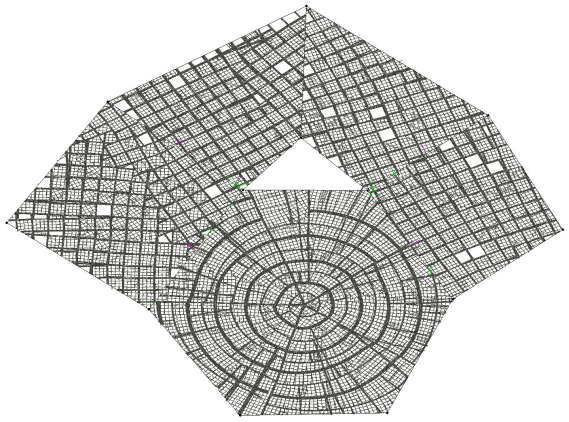

Figure 7: *A guided procedural system generated this complex urban layout, while street continuity across the guides is provided by passing messages.*

The L-system to generate the circular street layout has 51 lines and 9 rules, and the rectangular has 29 lines with 4 rules. The circular layout generates, in parallel, five arterial streets in a star like pattern. They generate circular arterial streets that automatically detect the corresponding points on the neighboring arteries and attempt to connect to them. All arterial streets can detect edges with a link and send messages to the neighboring guide. If this happens, the next guide starts another arterial street that continues in the same direction. In contrast, the second guide generates a regular pattern. Once the arterial street cannot continue, it executes a rule that generates a simple rectangular layout of parcels.

This is the most resource-demanding example. Every rule has a query symbol that detects distance to an intersection, and one that detects edges with links. The total number of elements generated in 70 iterations was 85,000, the maximum number of checked collisions was 6 million, and the maximum time required to regenerate the model was 45 second.

Figure 8: *An individual tree grown inside a single guide.*

5.5. Tree model

Here we want to show the advantage of using guides for classical procedural structures. The tree in Figure 8 is an example of a simple branching L-system grown inside a single guide. As the model grows, each branch creates one or two lateral branches. The probability of creating the lateral branches depends on the desired branch density, and the distance from the root, which is measured in the number of segments. Branches above a certain distance from the root create cluster of leaves that bend downwards.

In contrast, the tree in Figure 1 fully utilizes guides. Unlike the first tree that cannot be interactively modified, here the bottom branches and parts of the tree crown are defined separately in respective guides. The branch L-systems are much simpler than in the previous case. They just grow a sequence of segments that avoid the guide boundary, and seek the link at the opposite end of the guide. The branch clusters are grown using the same system as the one for the individual tree, they only start at a higher level. The information about the level is sent from each parent guide as a parameter of symbol $?X$.

These two examples are two extreme cases provided by our description. In the first one a single L-system controls the entire model and nearly no interactivity is possible. The

second case allows a very high level of control. Simpler L-systems are specified as well as their communication. There is virtually a continuous scale between these two cases, and it is the user's decision how much detail in the structure of the guides is necessary.

5.6. Mass-Spring Interaction

The existence of several guides defining the shape makes it possible to interactively adjust it by manipulating the guides without changing the L-systems. Figure 1 shows modifications of the tree model from the previous section using our interactive mass-spring system. Figure 1 a) shows the original image. In Figure 1 b) the structure of the edges was converted into a mass-spring system. Each edge is substituted by a spring and the user provided a set of interactive operations. Figure 1 c) shows the tree, where one guide was erased and two were scaled up. The tree has approximately 15,000 elements and its generation takes about 2 seconds. We refer to the video for examples of the interactive tree editing.

The last example in Figure 9a) shows a suspension bridge defined with three L-systems. One L-system creates the bridge tower, another one the bridge span by placing a set of beams at a different given cross angles. The third L-system defines the main cable with vertical cables placed a fixed horizontal distance. The guides control the placement of the two towers and the lower part of the bridge. The guides for the main cable are snapped together and the overall shape of the main cable can be changed by moving a cable guide. The position of the remaining guides is adjusted using the physics simulation. The L-systems inside the guides regenerate the horizontal cables at the desired spacing, regardless the shape of curve formed by the main cable. Figure 9b) shows a result after moving just a few guides.

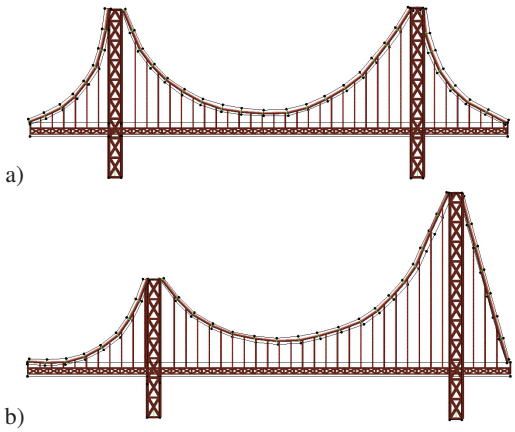

Figure 9: *An example of a suspension bridge, defined with guided procedural model with three simple L-systems.*

6. Conclusions and Future Work

We have presented guided procedural modeling implemented as guided Open L-systems that extends the concept of procedural models with exogenous control by generalizing the concept of environment. Instead of considering the environment as a single entity, we have divided it into mutually exclusive worlds, called guides that can communicate via message passing. The content of the message defines when and where a new L-system is created. The main advantages of our concept are the "top down" approach to modeling and the simplicity of the user control, addressing two of the main open problems of procedural systems. Instead of one large L-system, the user creates a much simpler set of L-systems that can be edited and tuned individually. The user also defines how the guides communicate with each other.

Our system has various limitations. It is easier to create and manipulate L-system models in our system, but it does not eliminate the need of manually writing the L-system productions. Although the L-systems are simple, extra attention is needed to support the communication with guides. In addition, there may be models that do not lend themselves to be used in our system. For example, if an upper branch of a tree should reduce the amount of light received by a lower branch that is defined in another guide an extra communication between guides would be necessary.

There are several avenues for future work. An obvious one is an extension of our system into 3-D. However, special care will be necessary to determine growth directions and also the interactive system would be more complicated. Another extension would be incorporating the concept of guides into a recently introduced procedural system used in urban modeling, such as CGA [MWH*06], split grammars [WWSR03], or shape grammars [SG71]. Another avenue for future work is recognizing guides in the process of inverse procedural modeling [vBM*10, BWS10]. Analogically to multifractals [Har01], a scene does not need to be the result of a single procedural model but it could be a composition. Also, in our system we have showed one level of message passing between guides. Apparently, this process could be done on multiple levels of hierarchy, but this could also complicate the user design. Last but not least, a user study to show the feasibility of design could be done.

Acknowledgements

We would like to thank NVIDIA for providing free graphics hardware. This work has been supported by NSF *IIS-0964302 Integrating Behavioral, Geometrical and Graphical Modeling to Simulate and Visualize Urban Areas* and Adobe Inc. grant *Constrained Procedural Modeling*.

References

[AH02] ARVIN S. A., HOUSE D. H.: Modeling architectural design objectives in physically based space planning. *Automation in Construction 11*, 2 (2002), 213 – 225. 3

[APS09] ANASTACIO F., PRUSINKIEWICZ P., SOUSA M. C.: Sketch-based interfaces and modeling: Sketch-based parameterization of L-systems using illustration-inspired construction lines and depth modulation. *Comput. Graph. 33*, 4 (2009), 440–451. 1

[ARB07] ALIAGA D. G., ROSEN P. A., BEKINS D. R.: Style grammars for interactive visualization of architecture. *IEEE Transactions on Visualization and Computer Graphics 13*, 4 (2007), 786–797. 1, 3

[BCNG10] BAXTER R., CRUMLEY Z., NEESER R., GAIN J.: Automatic addition of physics components to procedural content. In *AFRIGRAPH '10: Proceedings of the 7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa* (New York, NY, USA, 2010), ACM, pp. 101–110. 1, 3

[BWS10] BOKELOH M., WAND M., SEIDEL H.-P.: A connection between partial symmetry and inverse procedural modeling. *ACM Transactions on Computer Graphics* (2010), 1–10. 3, 9

[CEW*07] CHEN G., ESCH G., WONKA P., PASCAL, ZHANG M. E.: Interactive procedural street modeling. *ACM Trans. Graph. 27*, 3 (2007), 35. 3

[CNX*08] CHEN X., NEUBERT B., XU Y.-Q., DEUSSEN O., KANG S. B.: Sketch-based tree modeling using markov random field. *ACM Trans. Graph. 27* (December 2008), 109:1–109:9. 3

[DHL*98] DEUSSEN O., HANRAHAN P., LINTERMANN B., MĚCH R., PHARR M., PRUSINKIEWICZ P.: *Realistic modeling and rendering of plant ecosystems*. ACM, New York, NY, USA, 1998, pp. 275–286. 2

[EMP*03] EBERT D. S., MUSGRAVE F. K., PEACHEY D., PERLIN K., WORLEY S.: *Texturing and Modeling*, 3rd ed. Academic Press, Inc., Orlando, FL, USA, 2003. 1

[GPMG10] GALIN E., PEYTAVIE A., MARÉCHAL N., GUÉRIN E.: Procedural generation of roads. *Computer Graphics Forum (Proceedings of Eurographics) 29*, 2 (2010), 429–438. 3

[Har01] HARTE D.: *Multifractals: Theory and Applications*. Chapman and Hall/CRC, 2001. 9

[HKHF09] HOHMANN B., KRISPEL U., HAVEMANN S., FELLNER D.: Cityfit - high-quality urban reconstruction by fitting shape grammars to image and derived textured point clouds. In *Proceedings of 3D-ARCH 2009* (2009). 2

[IMIM08] IJIRI T., MĚCH R., IGARASHI T., MILLER G.: An example-based procedural system for element arrangement. *Comput. Graph. Forum 27*, 2 (2008), 429–436. 1

[IOI06] IJIRI T., OWADA S., IGARASHI T.: The sketch l-system: Global control of tree modeling using free-form strokes. In *In Smart Graphics* (2006), pp. 138–146. 3

[Lin68] LINDENMAYER A.: Mathematical models for cellular interaction in development. *Journal of Theoretical Biology Parts I and II*, 18 (1968), 280–315. 2

[LWW08] LIPP M., WONKA P., WIMMER M.: Interactive visual editing of grammars for procedural architecture. *SIGGRAPH '08: ACM SIGGRAPH 2008 papers* (2008), 1–10. 3

[LWW09] LIPP M., WONKA P., WIMMER M.: Parallel generation of l-systems. *Vision, Modeling, and Visualization Workshop* (Nov. 2009). 5

[MP96] MĚCH R., PRUSINKIEWICZ P.: Visual models of plants interacting with their environment. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM, pp. 397–410. 2, 3, 7

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., GOOL L. V.: Procedural modeling of buildings. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), ACM Press, pp. 614–623. 1, 2, 9

[MZWG07] MÜLLER P., ZENG G., WONKA P., GOOL L. V.: Image-based procedural modeling of facades. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), ACM, p. 85. 2

[Per85] PERLIN K.: An image synthesizer. *SIGGRAPH Comput. Graph. 19*, 3 (1985), 287–296. 1

[Pet77] PETERSON J. L.: Petri nets. *ACM Comput. Surv. 9*, 3 (1977), 223–252. 4

[PJM94] PRUSINKIEWICZ P., JAMES M., MĚCH R.: Synthetic topiary. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), ACM Press, pp. 351–358. 2

[PL90] PRUSINKIEWICZ P., LINDENMAYER A.: *The Algorithmic Beauty of Plants*. Springer–Verlag, New York, 1990. 2

[PM01] 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* (2001), ACM Press, pp. 301–308. 2, 8

[PMKL01] PRUSINKIEWICZ P., MÜNDERMANN L., KARWOWSKI R., LANE B.: The use of positional information in the modeling of plants. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM, pp. 289–300. 2

[Pru86] PRUSINKIEWICZ P.: Graphical applications of l-systems. In *Proceedings on Graphics Interface '86/Vision Interface '86* (Toronto, Ont., Canada, Canada, 1986), Canadian Information Processing Society, pp. 247–253. 2, 4

[SG71] STINY G., GIPS J.: Shape grammars and the generative specification of painting and sculpture. In *Segmentation of Buildings for 3DGeneralisation. In: Proc. of the Workshop on generalisation and multiple representation , Leicester* (1971). 2, 9

[Smi84] SMITH A. R.: Plants, fractals, and formal languages. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1984), ACM Press, pp. 1–10. 1

[SS08] SCHMIDT R., SINGH K.: Sketch-based procedural surface modeling and compositing using Surface Trees. *Computer Graphics Forum 27*, 2 (2008), 321–330. 3

[TLL*10] TALTON J., LOU Y., LESSER S., DUKE J., MĚCH R., KOLTUN V.: Metropolis procedural modeling. *ACM Trans. Graphics* (2010). 2

[VAB10] VANEGAS C. A., ALIAGA D. G., BENEŠ B.: Building reconstruction using manhattan-world grammars. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2010), p. 8. 2

[vBM*10] ŠŤAVA O., BENEŠ B., MĚCH R., KRIŠTOF P., ALIAGA D. G.: Inverse proc. modeling by automatic generation of L-systems. *Comp. Graph. Forum 29:2* (2010), 10. 3, 9

[WB97] WITKIN A., BARAFF D.: Physically based modeling: Principles and practice. Siggraph Course Notes, 1997. 6

[WOD09] WHITING E., OCHSENDORF J., DURAND F.: Procedural modeling of structurally-sound masonry buildings. *ACM Transactions on Graphics 28*, 5 (2009), 112. 1, 3

[WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. *ACM Trans. Graph. 22*, 3 (2003), 669–677. 1, 2, 9