Inverse Procedural Modeling by Automatic Generation of L-systems

O. Šťava[1], B. Beneš[1], R. Měch[2], D. G. Aliaga[1] and P. Krištof[1]

[1]Purdue University, USA   [2]Adobe Systems Incorporated, USA

Abstract

We present an important step towards the solution of the problem of inverse procedural modeling by generating parametric context-free L-systems that represent an input 2D model. The L-system rules efficiently code the regular structures and the parameters represent the properties of the structure transformations. The algorithm takes as input a 2D vector image that is composed of atomic elements, such as curves and poly-lines. Similar elements are recognized and assigned terminal symbols of an L-system alphabet. The terminal symbols' position and orientation are pair-wise compared and the transformations are stored as points in multiple 4D transformation spaces. By careful analysis of the clusters in the transformation spaces, we detect sequences of elements and code them as L-system rules. The coded elements are then removed from the clusters, the clusters are updated, and then the analysis attempts to code groups of elements in (hierarchies) the same way. The analysis ends with a single group of elements that is coded as an L-system axiom. We recognize and code branching sequences of linearly translated, scaled, and rotated elements and their hierarchies. The L-system not only represents the input image, but it can also be used for various editing operations. By changing the L-system parameters, the image can be randomized, symmetrized, and groups of elements and regular structures can be edited. By changing the terminal and non-terminal symbols, elements or groups of elements can be replaced.

Categories and Subject Descriptors (according to ACM CCS): Mathematical Logic and Formal Languages [F.4.2]: Grammars and Other Rewriting Systems—Parallel rewriting systems - L-systems; Computer Graphics [I.3.5]: Computational Geometry and Object Modeling— Geometric algorithms, languages, and systems;

1. Introduction

We present an important step towards the solution of the problem of *inverse procedural modeling*. Procedural techniques in computer graphics are commonly defined [EMP*03] as algorithms that generate a model, texture, or behavior. One of the most significant classes of procedural techniques are grammar-based methods, out of which the most common are methods based on Lindenmayer's systems (L-systems) [PL90]. L-systems use compact rules for model description yet they can generate a wide class of models.

The key challenge of procedural techniques is the definition of the rules. This task is not intuitive, lacks controllability, requires in-depth knowledge of procedural modeling, and resembles programming in an advanced language. Automatic generation of procedural rules, or *inverse procedural modeling*, has been an open problem for more than 20 years. Its goal is to find the rules of a procedural system that would

generate a given model. Applications of such an automatic system would be immense. For example in Figure 1, changing the rules allows generation of classes of similar models, the internal structure of the model could be modified, the model could be represented by its generative rules (compressed), the syntactic analysis of the rules could be used for image analysis, and so forth.

Our key observation is that models frequently consist of repeated elements organized into structures and the structures themselves appear multiple times as well. Such regularities are an ideal fit for procedural representation. Recently, several papers studying symmetries (e.g., [MGP06, PMW*08]) have exploited repetition for the purpose of model completion and symmetrization. Our use of the repetition is to create a procedural representation of the model.

Our algorithm takes an input vector image with objects formed as groups of line segments or Bézier curves and it
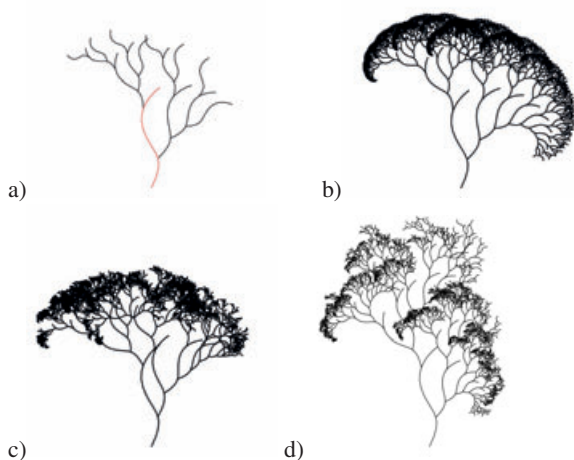
Figure 1: *a) A branching structure with varying scaling is automatically coded as an L-system. The basic element is a Bézier curve (in red), the generated rule has form $R(m) \rightarrow A[-(17)f(64)*(0.8)-(3)R(m-1)][+(7)f(181)*(0.6)+(23)R(m-1)]$, and the axiom is $R(3)$. b) A structure generated using the detected rule from axiom $R(10)$ and c) a branching structure generated from $R(10)$ with randomized angles and d) randomized angles and scaling.*

produces an L-system that generates the given input. We use various transformation spaces to detect sequential and branching structures of repeating elements (or groups of elements) with changing angle, size, and rotation.The main steps of our method are as follows (also see Figure 2).

Inverse Instantiation. The input atomic elements with an associated local coordinate system are grouped based on geometric similarity. Each similarity group is represented by one element, a set of transformations from the origin into each element, and a terminal symbol for use in an L-system.

Transformation Spaces Creation. The transformations between all pairs of elements are calculated. The parameters of each transformation are used as the coordinates of a point in a transformation space. Elements from each similarity group are compared to elements in all other similarity groups. The results from each pairing of similarity groups are stored in different transformation spaces. Once all the transformation spaces are filled, the points in them are clustered.

Rules Generation. The clusters are weighted and sorted to iteratively output an L-system rule using the elements of the most important cluster. The cluster is analyzed, the most relevant sequence of elements is selected, and an L-system rule is generated. References to the elements from the transformations of the cluster are erased from all clusters. A new symbol that corresponds to the sequence of elements is created and inserted into the transformation spaces. This process of iterative rules generation terminates when there is only a single element left.

We demonstrate our framework on theoretical and real-world examples. We analyze various structures, code them as L-systems, and generate similar structures, structures with more levels of hierarchy, and randomized structures. Another application of our system is a symmetrization of the model. The user can change elements in the input scene by replacing terminal symbols of the L-system. The parameters of the rules, such as the distance and angles between the elements, can be manipulated interactively, resulting in several forms of high-level editing operations. Groups of elements on a higher level of hierarchy can be replaced by a single element and the terminal symbol can be replaced by another procedural rule. This allows the user to connect different structures to each other. Moreover, storing the input scene as a set of procedural rules has applications in automatic object generation or data compression.

The main contributions of our paper are:

- an algorithm for matching of elements in vector images,
- automatic detection of various types of hierarchical repeated structures in vector images,
- an automatic coding of the structures as L-system rules,
- various applications of the detected rules such as image randomization, symmetrization, element replacement, and editing at different levels.

2. Previous Work

There are three main categories of related work: L-systems, attempts to generate procedural rules interactively, and symmetry detection.

L-systems were introduced by Lindenmayer [Lin68] as a mechanism for simulation of linear cellular subdivision. Prusinkiewicz [Pru86] extended the concept of L-systems by the graphical interpretation using a turtle. L-systems present the most advanced formal mechanism for simulation of growing structures [PL90]. They were also applied to determine plant distribution in plant ecosystems in [DHL*98], to simulate river formation in artificial terrains in [PH93], and to describe subdivision curves [PSSK03]. Recently, grammar-based procedural modeling has been applied to urban space modeling. Parish et al. used Open L-systems to generate street layout in virtual cities in [PM01]. Aliaga et al. introduced inverse procedural system to generate floors in virtual buildings in [ARB07]. Müller et al. introduced CGA shape for interactive building generation [MWH*06], and [MZWG07] used simple recursive grammar for façade generation. Most of the previous grammar-based approaches assume the rules are provided by the user. Automatic finding of the rules for a given model (the focus of this paper) has been recognized as one of the key problems of procedural modeling (see an early attempt to detect linear fractals [HCF97]).

Some recent related works describe attempts to actually create procedural rules from user input. Ijiri et al. [IOI06]
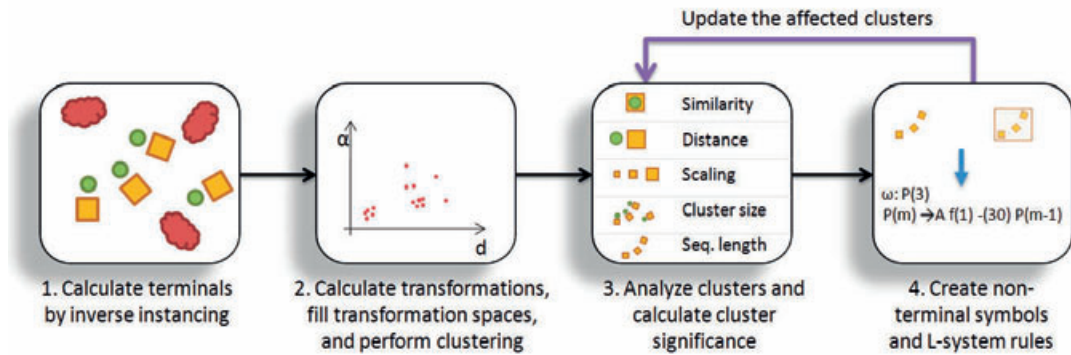
Figure 2: *System pipeline: 1. Similar elements of the input image are detected and terminal symbols are generated for each group of similar elements. 2. Pairwise transformations are calculated and transformation spaces are filled with points corresponding to each transformation. Mean shift clustering determines clusters of similarly oriented elements. 3. Cluster significance is calculated based on user-defined criteria. 4. Groups of elements are represented as L-system rules. Each group is represented as a new non-terminal and the clusters in the transformations space that contain references to them are updated. The process ends with a single element that represents the axiom of the L-system.*

used a sketch-based system to code user strokes as an L-system rule that is then used for an interactive plant generation. However, more complex rules cannot be generated and the system allows only for matching one predefined rule. Similarly Lipp et al. [LWW08] generate shape grammars from user input. Again, in their system only a predefined set of rules can be used. Ijiri et al. [IMIM08] introduced an example-based framework for procedural element arrangement generation and recently [YM09] used transformation spaces to detect symmetries and curvilinear arrangement in vector images. The above two user-assisted systems use simple algorithms for procedural models generation and they can express only a limited number of effects and a limited class of procedural models. The main difference between our approach and previous work is that we express the output using automatically determined rules for an L-system.

Our work is inspired by the recent work on symmetry detection. Mitra et al. [MGP06] introduced algorithms for detecting symmetries in unstructured 3D models by selecting dominant transformations in a transformation space. Similar concepts have since been used by others [LE06, PSG*06, XZT*09]. The model is sampled to create local signatures. The signatures are compared pairwise, and the results are mapped as points into the transformation space. As clusters of points correspond to potential symmetries, the clusters are detected and the symmetries are stored in a symmetry graph. The next paper by the same authors [MGP07] uses the symmetries to modify objects to express more symmetries than the originals. Pauly et al. [PMW*08] presented a paper on discovering regular and repeated geometric structures. They detect scaling, rotation, translation, and the combination of structural elements in an input scene. The principal finding of this work is that repetitive structures present themselves as regularly spaced sets of clusters in the transformation space. Other techniques for symmetry detection focus on detect-

ing features by first connecting them into a graph and then using a RANSAC-based randomized subgraph searching algorithm [BBW*08, BBW*09]. We exploit the concept of transformation spaces in our work, with the following differences. First, we do not use a single scene or a continuous object. Instead our input scene is a set of atomic, oriented elements. This allows us to use a particular transformation space for pairs of elements of specific types, resulting in little noise in each transformation space. Second, since our goal is the creation of L-system rules, the transformation space exploration is primarily driven by the desire to find the procedural rules and goes beyond just discovering symmetries. Third, we explore subspaces of the transformation space to detect rotated, translated, and scaled sequences of repetitive objects. Finally, we update the transformation spaces by one or multiple rules.

3. L-systems

We provide a brief description of L-systems and extensions that have been introduced for our purposes. Readers familiar with L-systems can skip this section keeping in mind that there are the following extensions: 1) we introduce scaling as a new command for the turtle, 2) the alphabet is divided into two groups of symbols: terminal and non-terminal symbols. Terminals are interpreted as turtle commands, and non-terminals are used in the rewriting process and 3) the terminal symbols are the standard turtle commands plus new symbols that represent the elements from the input image.

Our L-system is defined as a tuple

$$G = \langle M, \omega, \Re \rangle, \qquad (1)$$

where M is the L-system alphabet that contains elements $\{A(p_1, p_2, \ldots, p_n), B(p_1, p_2, \ldots, p_m), \ldots\}$ 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 (axiom). Finally, $\Re$ is a set of productions (rules)

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

where *label* is the production identifier, *cond* is a Boolean expression involving the parameters, and M^* 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. We drop the *label* if the rule is not reused.

The *implicit rule* for rewriting a single module is:

$$r_0 : A(p_0, p_1, \ldots, p_n) \rightarrow A(p_0, p_1, \ldots, p_n), \qquad (2)$$

and it rewrites a module with its exact copy. This is applied when there is no corresponding rule in $\Re$ for a module from M. The *epsilon rule* erases the symbol $A(p_0, p_1, \ldots, p_n)$:

$$r_\varepsilon : A(p_0, p_1, \ldots, p_n) \rightarrow \varepsilon. \qquad (3)$$

Turtle Interpretation. The string of modules is interpreted geometrically by a turtle [PH93]. The turtle has a position $[x, y]$, an uniform scaling factor s, and heading angle ϕ that together define its state (x, y, s, ϕ). The turtle reads sequentially the modules from the string and interprets them geometrically as described by Table 1.

module	interpretation
$f(\Delta)$	translate in the direction of heading by Δ
$+(\delta)$	rotate by δ to the left
$-(\delta)$	rotate by δ to the right
$*(s)$	set scaling to s
[	push the state on the stack
]	pop the state from the stack, move to $[x, y]$, set scaling to s, and head in the direction ϕ

Table 1: *Turtle interpretation of the modules.*

Traditionally, the scaling is not associated with the turtle. However, we found it convenient to include scaling as a turtle command $*(s)$ that affects all subsequent geometric commands (e.g., rendering, translation).

Terminal and Non-Terminal Symbols. Using the commands from the Table 1 the turtle can freely move in the 2D plane. However, our goal is the generation of an input image that is a collection of certain basic elements. Therefore, we expand the turtle commands by a special "print" command that draws an image element on a given turtle position. The content of the print command depends on the input image and it is not known in advance. Therefore, the Table 1 is extended dynamically by the elements from the input image.

Higher level rules use symbols that have no direct geometric interpretation and are used as a proxy for groups of elements. That is why we define the alphabet as the union of two subsets - the terminal and the non-terminal symbols.

The terminal symbols are denoted by V, non-terminal symbols by S and the alphabet $M = V \cup S$. The terminal symbols have a geometrical interpretation either given by the Table 1, or are printed as the elements of the input image, whereas the non-terminals have no direct geometrical interpretation as they represent a group of elements. The L-system rules operate on the alphabet and if a terminal symbol appears on the left side of the rule, the implicit rule r_0 is applied.

4. Transformation Space Initialization

This section describes the two steps that initialize our algorithm. First we detect all similar structures and we assign them a single representative. Second, we fill the transformation spaces with information about pairwise transformations between the elements and find clusters.

4.1. Input Element Similarity Detection

We define an image to be a collection of atomic structures called *elements*. An element is denoted by e_i, and the set of all elements is $E = \{e_i, i = 1, 2, \ldots, |E|\}$. The objective of this step is to find all possible similar elements, regardless of their position, orientation, or scale in the input image.

An input element is a sequence of connected lines or curves. It is the smallest unit and the similar objects are then detected during the process of rules generation. Unfortunately, as vectorization software packages may produce different curves for similar elements we need to provide a consistent way of measuring similarity of two input elements. Let's denote the similarity of elements e_i and e_j by $\phi(e_i, e_j)$. We do the following:

1. We first uniformly sample both elements by a set of points S_i and S_j.
2. For each element set we evaluate the maximum distance between two sample points d_i^{max} and d_j^{max} and calculate a scaling factor $s(e_i, e_j) = d_j^{max} / d_i^{max}$.
3. The element e_j is rescaled by $s(e_i, e_j)$ and resampled yielding a new set of sample points S_j. Thus both elements e_i and e_j are sampled at comparable sizes.
4. For all samples in S_i and S_j we compute their normal vector and curvature as described in [YM09].
5. For all samples in S_i we detect corresponding samples from S_j that have a similar value of curvature. For every corresponding pair of samples, we evaluate a transformation T_k that transforms element e_i to e_j so that the positions and orientation of both samples are matched.
6. All computed transformations T_k are inserted into a 3D transformation space, mean shift clustering is performed, and the cluster with the maximum number of points n is determined.
7. The similarity of the two elements is then computed as: $\phi(e_i, e_j) = n / |S_j|$ and two elements are considered similar when their similarity is greater than some user defined threshold value (0.6 in our application).

The result of this step is a unique ID that identifies elements that are visually similar independently on their orientation, position, and scaling. The result depends on the choice of the similarity threshold and can fail for highly dissimilar inputs.

4.2. Terminal Symbols by Inverse Instancing

All similar elements determined in the previous step are grouped and the transformations necessary to locate them at the corresponding position in the input image are determined. Each *similarity group* of elements is represented by a single terminal symbol $A \in V$ in the origin of the coordinate system and a list of transformations T_{A_i}, specifying the position, uniform scaling, and orientation of the *i*-th instance of the element with respect to the origin (see Figure 3). All similar elements can be generated from the representative and the list of transformation.
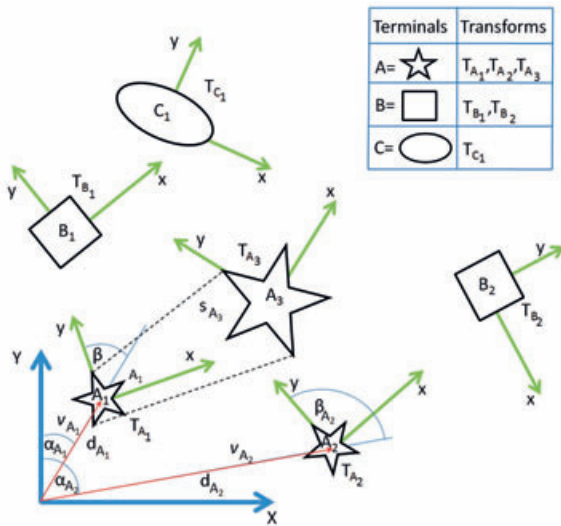

Figure 3: *Inverse instancing. Similar elements of the input image are represented by a terminal symbol and a set of transformations.*

Using the L-system notation the transformation T_{A_i} can be described by symbols $+(\alpha_{A_i})f(d_{A_i})*(s_{A_i})+(\beta_{A_i})$, where d_{A_i} is the distance between the origins of the world coordinate system and the element's local coordinate system, α_{A_i} is the angle of the axis *y* to the vector *v* connecting the origins, and β_{A_i} is the angle of the element to the same vector *v*. Scale s_{A_i} is given as the size ratio of one chosen representative and the instance A_i.

4.3. Transformation Spaces Filling and Clustering

In this step we determine mutual transformations between pairs of elements represented as instances of terminal symbols, and create various transformation spaces for them.
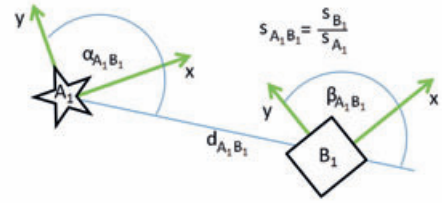

Figure 4: *Transformation $T_{A_1 B_1}$ is a sequence of turtle commands $+(\alpha_{A_1 B_1})f(d_{A_1 B_1})*(s_{A_1 B_1})+(\beta_{A_1 B_1})$.*

If T_{A_i} and T_{B_j} are transformations from the world coordinate system to the element's coordinate system we express the transformation between A_i and B_j (see Figure 4) as

$$T_{A_i B_j} = T_{A_i}^{-1} T_{B_j},$$

where $T_{A_i}^{-1}$ is the inverse transformation to T_{A_i}. The transformation $T_{A_i B_j}$ is described by the four values of $t_0 = (\alpha, d, s, \beta)$ that identify a point in a 4D transformation space. We store each point as well as the backward link to the elements A_i and B_j. There can be multiple points at the same location representing multiple pairs of elements that undergo the same transformation.

L-systems have a great expressive power for structures with a high level of repetition and that is why we want to discover relations between groups of similar elements. So instead of using a single transformation space [MGP06, PMW*08], we use multiple transformation spaces. Specifically, having *n* terminal symbols we create $(n \times n)$ transformation spaces, one for each pair of terminals. In practical cases such as in Figure 12 we have hundreds of transformation spaces. However, the transformation spaces are usually not dense and their analysis is not time consuming.

Let's recall that multiple points in the dual space correspond to similarly distributed elements in the input image. So in the next step, similar to the symmetry recognition approach by [MGP06], we perform mean shift clustering to determine groups of elements with similar transformations.

5. Procedural Rules Generation

5.1. L-system Parameters

We chose to generate parametric context-free L-systems that use only specific parameters and expressions in the rules yet are powerful enough to capture a wide variety of repeating structures in distance, angle, and scalings. Ideally, we would like to use a single rule that would be easy to match yet allow for a wide variety of combinations. We have found matching this rule difficult and that is why we generate two kinds of rules. The first rule provides sequences of symbols and the second rule generates branching. It is important to note that these two rules are not the direct output of the L-system generation but are the generic rules that are merged into more complex rules during the analysis.

The condition *cond* of a rule in a parametric deterministic context free L-systems (Section 3) or the values of parameters on the right side of each production can result from a mathematical expression that uses the parameters $p_0, ..., p_n$. Arbitrary expressions add too many degrees of freedom and they make it difficult to fit such an L-system to a given set of terminal symbols. We want to capture repetitive linear structures. These can be described by a single parameter m that is used as a counter of the number of occurrences, and a vector of parameters $t_0 = (\alpha_0, d_0, s_0, \beta_0)$ that define the initial value of the transformation. We limit the expressions used on the right side of rules to a linear function of m:

$$t(m) = t_0 + t_\Delta \, m, \qquad (4)$$

where t_0 are the initial values and t_Δ is a vector expressing the difference in one step.

5.2. Atomic Rules

The transformation spaces provide pairs of elements so it is convenient to use rules that work with pairs of elements. We make an observation that any production with k symbols on the right side can be replaced with $k-1$ productions that generate at most two symbols. For example, the production $A \rightarrow BCD$ can be replaced by these three productions:

$$A \rightarrow XY, \; X \rightarrow BC, \; Y \rightarrow D$$

Note that two additional non-terminal symbols X, Y are introduced and that more steps are necessary to generate the same string. Based on the observation about the length of rules and the assumption of detecting structures with linear dependency of the transformations between elements, we define sequence rules and branching rules.

Sequence Rules creating a sequence of symbols are:

$$r_1 : P(m, t_0) : m > 0 \quad \rightarrow \quad [A] \, T(t(m)) \, P(m-1, t_0)$$
$$r_2 : P(m, t_0) : m == 0 \quad \rightarrow \quad [B]$$

where P is a non-terminal and A can be either a terminal or a non-terminal (representing an element or a group of elements, respectively). Symbol $T(t(m))$ denotes a transformation that in L-system notation can be expressed as $+(\alpha(m)) f(d(m)) * (s(m)) + (\beta(m))$. The rule r_2 terminates the sequence of symbols A with a symbol B (symbol B can be equal to symbol A or it can be ε). The brackets [and] are necessary when symbol A or B is a non-terminal, and will be further expanded. Note that for $t_\Delta = 0$ we can write these rules without the parameter t_0. Figure 5 illustrates the use of rule r_1 for a case of $P(3, t_0)$, $t_0 = (\alpha, d, s, \beta)$, $\alpha = 60^o$, $\beta = -50^o$, $s = 1$, and d and $t_\Delta = (10^o, 0, 0, 10^o)$.

Note that the rules r_1 and r_2 can be used not only to generate sequences of equal symbols or sequences of equal symbols terminated by a different symbol, but also an arbitrary pair of elements A and B. More importantly, these rules can express any parametric L-system with expressions (4). The
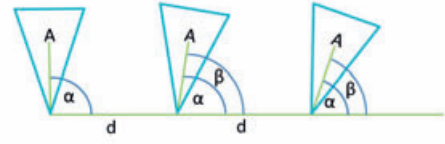

Figure 5: *Geometric interpretation of the string generated by the rule r_1 with the axiom $\omega = P(3, t_0)$, $t_0 = (60^o, 1, 1, -50^o)$, and $t_\Delta = (10^o, 0, 0, 10^o)$.*

type of expressions allows us to represent sequences of elements with the combination of: rotation, translation, scale, and an additional rotation of the object itself.

Branching Rules. Arrangements such as circular, translational, or scaling sequences, or their combinations (see Figure 9) can be directly expressed by one instance of rule r_1 and r_2. However, another arrangement that appears often in procedural patterns is branching that can be generated by several instances of sequence rules. The structure from Figure 1a) can be generated as four sequences of the symbol A generated
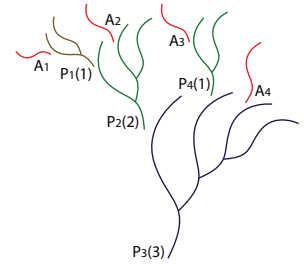

Figure 6: *Detecting branching structures using sequence rules would only lead to too many new non-terminals P_i.*

by rule r_1 of lengths 3, 2, and 1, with initial symbols $P_i(i, t_0)$, $i = 1, 2, 3$, and $P_4(1, t_0)$, where $t_0 = (7, 181, 0.6, 23)$ and $t_\Delta = 0$ (see Figure 6). These symbols would be connected together with each other and with the symbols A_i in subsequent steps of our algorithm, but not in one sequence, since the symbols P_i would be different from each other. The resulting L-system would be valid, but the branching would be described in a cumbersome way. Consequently, we decided to add an explicit branching rule:

$$r_3 : R(m, t_0, t_0') : m > 0 \quad \rightarrow \quad [A] \, [T(t(m)) \, R(m-1, t_0, t_0')]$$
$$\qquad\qquad\qquad\qquad\qquad T'(t'(m)) \, R(m-1, t_0, t_0')$$
$$r_4 : R(m, t_0, t_0') : m == 0 \quad \rightarrow \quad [B]$$

The example from Figure 6 can be then generated using a pair of rules r_3 and r_4 with $t_0 = (-17, 64, 0.8, -3)$, $t_0' = (7, 181, 0.6, 23)$ and the initial symbol $R(3, t_0, t_0')$ (see also Figure 1a). The following section describes how the branching and sequence rules are constructed from clusters in the transformation spaces.

5.3. Cluster Analysis

A cluster in the transformation space represents pairs of elements of the general form AB or AA that can be generated using the same rules. We could replace each such pair with a

non-terminal symbol $P(1,t_0)$ and the sequence rules r_1 and r_2. However, the pair can be a part of a larger sequence of similarly oriented elements, so we link the elements in the cluster into a sequence that can be generated by $P(m,t_0)$, where m is the number of the elements in the sequence.

We want to detect rules in which the values of α, d, s, and β change linearly with respect to m as expressed by (4). Transformations between elements generated by such a rule are not represented by a single point in the transformation space. Instead, they form regular structures. If only one value changes with respect to m, say the value of α, there is a sequence of points in the transformation space with a constant distance from each other in the α dimension. A similar problem has been addressed by Pauly et al. [PMW*08] where they find a regular grid in a three-dimensional space representing symmetry transformations in 3D. Because of the choice of the rules, we need to search for regularly spaced points on a line in our transformation spaces only.

However, not all linear sequences of points in the transformation space correspond to interesting structures. Structures with linearly changing distance or scale do not appear frequently in the input scenes and we assume that only the parameters α and β may change. This can be efficiently restated as a task of finding clusters in each of the 2D subspace of the transformation space (d,s) and determining if the values in the remaining two subspaces form regularly spaced points and if they belong to one or more sequences of symbols that can be generated by the rule r_1.

The cluster analysis algorithm attempts to identify sequential or branching structures:

1. Find clusters $\mathbf{C_i}$ in the space (α,d,s,β).
2. Find clusters $\mathbf{D_j}$ in the two-dimensional subspace (d,s).
3. a. *Sequential rules:* Each point in a cluster represents a pair of elements. If the second element of a pair is equal to the first element of another pair, we connect these two into a sequence. We find the longest sequence in each cluster $\mathbf{C_i}$ and $\mathbf{D_j}$.
   b. *Branching rules:* If the second element of a pair is the first element of two different pairs (from two different clusters) it is a branching sequence. We find the most frequently repeated branching sequences in each cluster $\mathbf{C_i}$ and $\mathbf{D_j}$.

5.4. Cluster Selection

Clusters vary in the number of elements and their content. In order to generate "reasonable" rules, and to allow user control over the process of rules generation, we define *a weighted cluster importance function* that assigns a value of importance to each cluster. The parameters have been determined based on assumptions we have made after visual inspection of a number of input images. The function automatically selects the branching or the sequence rule.

The first assumption is that the distance between elements should be taken into account. The generated L-system rules are more intuitive if we merge elements that are close to each other rather than distant elements. An example would be a sequence of equally-spaced elements where it is reasonable to merge neighbors first. The second assumption is that it is better to start with clusters with a higher number of points, because they represent repeating patterns. The third assumption is that it is better to merge similar elements first, since that would be a natural way of creating procedural systems by a human. Finally, the last assumption is to give priority to clusters with long sequences of symbols as they lead to a higher value of m in terminals $P(m,t_0)$ for rules r_1 and r_3. The four assumptions are quantified into the cluster importance function that assigns a value to each cluster:

$$w = w_n n + w_h h + w_\phi \phi + w_l l \tag{5}$$

where n is the number of the sequential elements or number of branches (depending on the rule) in the cluster, h is the proximity of two elements, ϕ is the element similarity, and l is the average length of the sequences in the cluster. The weights $0 \leq w_n, w_h, w_\phi, w_l \leq 1$ are defined by the user.

The normalized distance $d(e_i, e_j)$ is defined as the closest distance between point samples on the elements (not the distance between local origins of two elements). d ranges from 0 (overlapping elements) to 1 (for the maximum distance of two elements in the input data). The similarity $\phi(e_i, e_j)$ is also calculated from the point samples using the algorithm described in Section 4.1. In order to account for symmetries, we also flip the sets around the x and y axis. The number l is the average length of all sequences in a cluster divided by the maximum average length of sequences in all clusters.

Intuitively, setting the distance weight w_h to a small value allows merging of objects far from each other. Changing the similarity w_ϕ defines the level of error allowed for two different but similar elements (or group of elements) to be merged (see Figure 7). Manipulating the weight w_l can lead to rules generating smaller or larger sequences of elements. The weight w_n gives higher priorities to repeating structures regardless of whether they are in a sequence.

Figure 7: *Cluster importance function. Color identifies grouped elements. a) Users prefer distance to define grouping of elements, whereas in b) element similarity is preferred.*

The values of n, h, ϕ, and l are associated with each cluster. Note that all pairs of the elements in the cluster have equal or similar values. The clusters are then sorted according to the value of the cluster importance function (5). The procedural rules generation then starts with the most important cluster.

An example in Figure 7 shows influence of the sensitivity to the distance and similarity parameters. In Figure 7a)

the proximity of objects is preferred and the result is two sequences. On the other hand, in Figure 7b) the object similarity is preferred and the system detects three different sequences. The weight selection is intuitive but can be problematic for highly noisy input data.

5.5. Rule Generation

In this step we take the cluster with the highest weight and create the corresponding sequential or branching rule. The new rule is described by a new non-terminal symbol $P(m, t_0)$ (or $B(m, t_0, t_0')$ for the branching rule), where m is the recursion depth of the rule and t_0 is the non-parametric portion of the transformation described by the rule. While t_0 (and t_0') are constant for all points in the processed cluster, the value of m can vary as one cluster can contain multiple sequences with different lengths. Because identical rules with different values of m produce different elements, we need to create a new non-terminal symbol for every possible sequence length m that is contained in the cluster.

5.6. Higher Level Rules Generation

When the rule is generated we eliminate its elements (or group of elements) from the scene and substitute them with the non-terminal symbol(s). The affected clusters are regenerated efficiently by keeping mutual links between elements and the clusters.

Consequently, we replace each occurrence of an element described by symbol A and transformation T_A in any cluster by the new non-terminal element P and transformation $T_P = T_A$. In this way a group of elements becomes a new element of the scene. Elements of a sequence or branching other than the first one are removed from the associated clusters. This mechanism prevents reclustering. However, the cluster importance function must be recalculated for all clusters. We use point sampling to determine distance and similarity of the higher structural elements (groups) in the same way as for the elements of the input image.

When the clusters are updated with the new element, it can be merged into higher-level elements in the next iteration. In this way we generate complex hierarchies where structures on lower levels are either more abundant or more important than the structures on higher levels. The algorithm completes when there is a single non-terminal symbol $X(m, t_0)$ left. This symbol and its transformation T_X form the axiom of the L-system $\omega : T_X X(m)$.

Reinserting the new non-terminal symbols into transformation spaces assures that we find not only the basic combination of transformations: rotation, translation, scale, and an additional rotation of the object itself, but also their combinations. For example, we can find a grid of elements as translation $\times$ translation, but also a grid of elements where elements at each row are rotated around its origin by an increasing angle: translation $\times$ translation $\times$ rotation, etc.

5.7. Post-processing rules

The generated rules are further processed and simplified. First we clean up the generated rules. For rules that do not use linear dependency of parameters on m, we skip the parameter t_0 and use fixed values in the rule. We eliminate meaningless transformations such as $+(0)$ and $-(0)$, scaling by one, etc. We also combine together rules r_1 and r_2 for $P(1, t_0)$ to a new rule: $P \rightarrow A\ T\ B$.

Fig	ele-ments	rules	init. [s]	generation of rules [s]	total time [s]
10	72	4	14	2	16
11	1184	182	605	610	1215
12	500	42	55	154	209

Table 2: *Time needed to analyze the figures from this paper.*

6. Implementation and Results

We have implemented our system in C++. It uses OpenGL for visualization of the structures. It reads standard SVG files and generates corresponding L-systems. Most of the examples in this paper were analyzed within seconds or minutes. The most complicated example (Figure 11) needed twenty minutes (see Table 2).
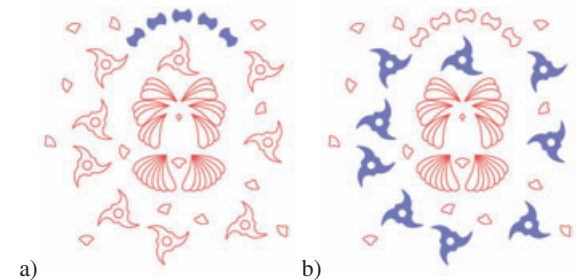

a)   b)

Figure 8: *Two extreme cases detected (blue color) by the system. a) The regular structure is captured by the rule r_1 with $P(4)$. b) The structure has no regularities and is represented as a sequence of transformations of pairs by the rule r_1 with multiple non-terminals $P(1, t_i)$.*

Figure 8 shows examples that demonstrate a sequence detection using our framework. On the left, a (blue) sequence of five elements is detected and coded by the rule r_1 with parameters $t_0 = (7^0, 50, 1, 7^0)$ and the axiom $P(4)$. The right image shows a clutter of similar elements that are detected by the inverse instancing, but there is no repetition of transformations. This is the worst case scenario and the elements are coded as a set of pairs produced by the rule r_1.

An opposite case is the helix in Figure 9 a) that shows high structural symmetry and is stored as the L-system rule r_1 with $t_0 = (0, 27.5, 0.993, 8.5^o)$. i.e., $P(m) \rightarrow A f(27.5) *(0.993) + (8.5) P(m - 1)$ and an axiom $\omega : P(336)$. Figure 9 demonstrates the possibility of editing the generated structures by modifying L-system parameters.
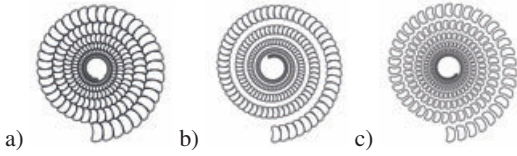
Figure 9: *A) Helix detected from an input image is represented by the rule $P(m) \rightarrow Af(27.5) * (0.993) + (8.5)P(m - 1)$ and includes $m == 336$ repetitions. b) Helix generated from the same rule with increased angle between two elements, and c) with increased distance between elements.*

A branching structure in Figure 1 a) is recognized and stored as a rule:

$$R(m) \rightarrow A[-(17)f(64) - (3)R(m-1)] \\ [+(7)f(181) + (23)R(m-1)]$$

Figure 1 shows different outputs produced by changing the number of recursive calls, internal angles, and scaling. This example shows the expressive power of L-systems.
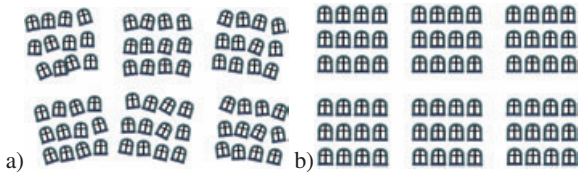

Figure 10: *Symmetrization using L-systems. Objects distributed with repetition but large randomness (left) are recognized by setting relaxed sensitivity to angle and distance. The image on the right is generated from the detected rules.*

An example of symmetrization on the level of L-system rules is presented in Figure 10, where hierarchies of similar elements are randomly rotated and translated. The relaxed clustering merges the elements into clusters, the L-system generator produces the axiom $P_3(2)$ and rules:

$$\begin{aligned} A &\rightarrow Window \\ P_0(m) &\rightarrow Af(13)P_0(m-1) \\ P_1(m) &\rightarrow P_0(3) + (90)f(10) - (90)P_1(m-1) \\ P_2(m) &\rightarrow P_1(4) + (90)f(52) - (90)P_2(m-1) \\ P_3(m) &\rightarrow P_2(3)f(51)P_3(m-1) \end{aligned}$$

that lead to the symmetrical composition in Figure 10 b).

Terminal or non-terminal symbol replacement is another operation that can be applied directly to the rules of the generated L-systems and has an important effect on the generated image. This operation is demonstrated in Figure 11. The input scene is analyzed and coded as L-system. The scene is horizontally extended by changing the parameter m of the axiom and some non-terminal symbols (i.e., different levels of hierarchy) are then replaced by a terminal symbol.

A complex scene in Figures 12 demonstrates the power of an editing system based on L-systems. The scene is continuously analyzed as the user defines different weights of

Figure 11: *A complex structure is coded as an L-system (up), enlarged, (middle), and a non-terminal symbol is replaced by a terminal that substitutes a group of elements with an element (down).*

the cluster weighting function. By giving preferences to different weights in the cluster importance function, the system finds different structures that are expressed as L-systems. By changing the L-system parameters, the scene is modified, resulting in a new image in Figure 12 b). Generation of a similar result by manual editing of a vector image would take significantly longer.

7. Conclusions and Future Work

We presented an important step towards the solution of the problem of inverse procedural modeling by generating L-systems that represent an input 2D scene. The key concept behind our algorithm is an efficient analysis of various transformation spaces that code mutual transformations between pairs of elements. We efficiently recognize and code branching and non-branching sequences of linearly translated, scaled, and rotated elements and their hierarchies. The L-system is a representation of the input image and it can be used, for example, for scene randomization, symmetrization, editing of different structures, or substitution of elements and groups of elements by new symbols.

As future work, we would like to enhance the class of L-systems that we can generate. We want to investigate the use of more complex expressions in the L-system rules, such as polynomials that can be evaluated using sequences of simple
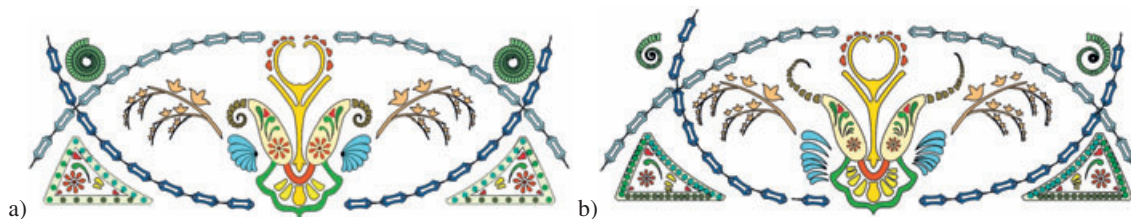
Figure 12: *a) An input image is automatically coded as an L-system and b) edited by user by manipulating L-system parameters.*

rules. We want to explore coding of structures, for which the L-system rules require context as well as similarity among different elements. An analysis of the resulting rules and detecting the regularities between the rules at different levels of the hierarchy would make it possible to automatically detect other regular patterns in the rules besides branching. Another are of future work is the creation of a comprehensive editing system based on L-systems that could interactively generate L-system rules and allow local changes to propagate through the L-system rules. We would like to extend our system to the automatic coding of 3D structures and improve the robustness of the algorithm and its sensitivity to noise.

Acknowledgements

References

[ARB07] ALIAGA D. G., ROSEN P. A., BEKINS D. R.: Style grammars for interactive visualization of architecture. *IEEE Trans. on Vis. and Comp. Graph. 13*, 4 (2007), 786–797. 2

[BBW*08] BERNER A., BOKELOH M., WAND M., SCHILLING A., SEIDEL H.-P.: A graph-based approach to symmetry detection. In *Symposium on Volume and Point-Based Graphics* (Los Angeles, CA, 2008), Eurographics Association, pp. 1–8. 3

[BBW*09] BOKELOH M., BERNER A., WAND M., SEIDEL H.-P., SCHILLING A.: Symmetry detection using line features. *Comp. Graph. Forum 28*, 2 (2009), 697–706. 3

[DHL*98] DEUSSEN O., HANRAHAN P., LINTERMANN B., MĚCH R., PHARR M., PRUSINKIEWICZ P.: Realistic modeling and rendering of plant ecosystems. In *SIGGRAPH '98 Proceedings* (New York, NY, USA, 1998), ACM Press, 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

[HCF97] HART J. C., COCHRAN W. O., FLYNN P. J.: Similarity Hashing: A computer vision solution to the inverse problem of linear fractals. *Fractals 5*, (1997), 35–50. 2

[IMIM08] IJIRI T., MECH R., IGARASHI T., MILLER G.: An example-based procedural system for element arrangement. *Comput. Graph. Forum 27*, 2 (2008), 429–436. 3

[IOI06] IJIRI T., OWADA S., IGARASHI T.: The sketch l-system: Global control of tree modeling using free-form strokes. In *Smart Graphics* (2006), vol. 4073 of *LNCS*, Springer, pp. 138–146. 2

[LE06] LOY G., EKLUNDH J.-O.: Detecting symmetry and symmetric constellations of features. In *In ECCV* (2006), vol. 2. 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. *ACM Trans. Graph. 27*, 3 (2008), 102:1–10. Article No. 102. 3

[MGP06] MITRA N. J., GUIBAS L. J., PAULY M.: Partial and approximate symmetry detection for 3d geometry. *ACM Trans. Graph. 25*, 3 (2006), 560–568. 1, 3, 5

[MGP07] MITRA N. J., GUIBAS L. J., PAULY M.: Symmetrization. *ACM Trans. Graph. 26*, 3 (2007), 63. 3

[MP96] MĚCH R., PRUSINKIEWICZ P.: Visual models of plants interacting with their environment. In *SIGGRAPH '96 Proceedings* (New York, NY, USA, 1996), ACM, pp. 397–410. 2

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., GOOL L. V.: Procedural modeling of buildings. In *ACM Trans. Graph. 26*, 2 (2006) pp. 614–623. 2

[MZWG07] MÜLLER P., ZENG G., WONKA P., GOOL L. V.: Image-based procedural modeling of facades. In *ACM Trans. Graph. 26* 3, (2007), p. 85. 2

[PH93] PRUSINKIEWICZ P., HAMMEL M.: A fractal model of mountains with rivers. In *Proceedings of Graphics Interface'93* (1993), vol. 30(4), pp. 174–180. 2, 4

[PJM94] PRUSINKIEWICZ P., JAMES M., MĚCH R.: Synthetic topiary. In *SIGGRAPH '94 Proceedings* (New York, NY, USA, 1994), ACM Press, pp. 351–358.

[PL90] PRUSINKIEWICZ P., LINDENMAYER A.: *The Algorithmic Beauty of Plants*. Springer–Verlag, New York, 1990. 1, 2

[PM01] PARISH Y. I. H., MÜLLER P.: Procedural modeling of cities. In *SIGGRAPH '01 Proceedings* (New York, NY, USA, 2001), ACM, pp. 301–308. 2

[PMW*08] PAULY M., MITRA N., WALLNER J., POTTMANN H., GUIBAS L.: Discovering structural regularity in 3d geometry. *ACM Trans. Graph. 27*, 3 (2008), 1–11. 1, 3, 5, 7

[Pru86] PRUSINKIEWICZ P.: Graphical applications of L-systems. In *Proceedings on Graphics Interface'86,* Canadian Information Processing Society, pp. 247–253. 2

[PSG*06] PODOLAK J., SHILANE P., GOLOVINSKIY A., RUSINKIEWICZ S., FUNKHOUSER T.: A planar-reflective symmetry transform for 3d shapes. *ACM Trans. Graph. 25*, 3 (2006), 549–559. 3

[PSSK03] PRUSINKIEWICZ P., SAMAVATI F. F., SMITH C., KARWOWSKI R.: L-system description of subdivision curves. *International Journal of Shape Modeling 9*, 1 (2003), 41–59. 2

[YM09] YEH Y.-T., MĚCH R.: Detecting symmetries and curvilinear arrangements in vector art. *Comp. Graph. Forum (Proc. EUROGRAPHICS) 28*, 2 (2009), pp. 707–716. 3, 4

[XZT*09] XU K., ZHANG H., TAGLIASACCHI A., LIU L., LI G., MENG M., XIONG Y.: Partial Intrinsic Reflectional Symmetry of 3D Shapes. *ACM Trans. Graph. 28*, 5 (2009) 3