WorldBrush: Interactive Example-based Synthesis of Procedural Virtual Worlds

Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, Bedrich Benes

WorldBrush: Interactive Example-based Synthesis of Procedural Virtual Worlds

Arnaud Emilien[1,2]* Ulysse Vimont[1]† Marie-Paule Cani[1]‡ Pierre Poulin[2]§ Bedrich Benes[3]¶
[1]University Grenoble-Alpes, CNRS (LJK), and Inria [2]LIGUM, Dept. I.R.O., Université de Montréal [3]Purdue University

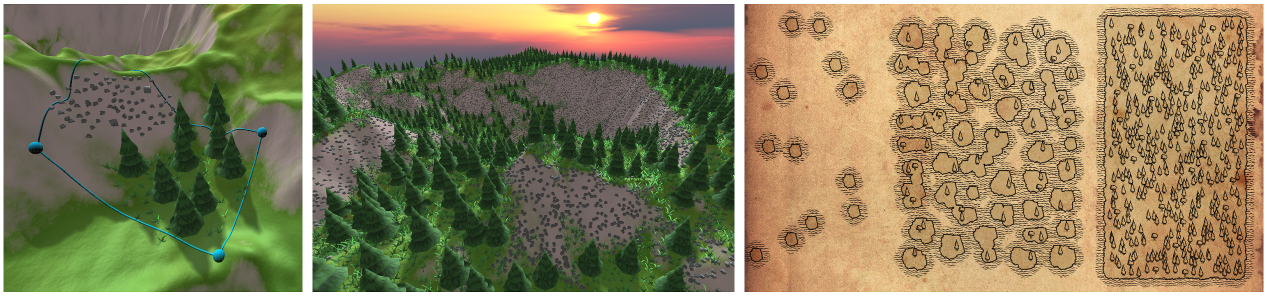

Figure 1: *An example model (left) is used to learn parameters of distributions of trees, grass, and rocks constrained by the terrain's slope. Our model then uses these parameters to create consistent content in a larger region, using copy-paste or with user-controlled interactive brushes (center). Various adapted editing operations are introduced, inspired by painting tools, such as interpolation of parameters, that allow for creation of varying islands with trees (right).*

Abstract

We present a novel approach for the interactive synthesis and editing of virtual worlds. Our method is inspired by painting operations and uses methods for statistical example-based synthesis to automate content synthesis and deformation. Our real-time approach takes a form of local inverse procedural modeling based on intermediate statistical models: selected regions of procedurally and manually constructed example scenes are analyzed, and their parameters are stored as *distributions* in a palette, similar to colors on a painter's palette. These *distributions* can then be interactively applied with brushes and combined in various ways, like in painting systems. Selected regions can also be moved or stretched while maintaining the consistency of their content. Our method captures both distributions of elements and structured objects, and models their interactions. Results range from the interactive editing of 2D artwork maps to the design of 3D virtual worlds, where constraints set by the terrain's slope are also taken into account.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Modeling packages I.3.4 [Computer Graphics]: Graphics Utilities—Graphics editors I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods—Representations (procedural and rule-based)

Keywords: computer graphics, inverse procedural modeling, interactive modeling, virtual worlds, painting systems

*e-mail:emilien@wildmagegames.com
†e-mail:ulysse.vimont@inria.fr
‡e-mail:marie-paule.cani@inria.fr
§e-mail:poulin@iro.umontreal.ca
¶e-mail:bbenes@purdue.edu

1 Introduction

Geometric modeling is one of the fastest developing areas of computer graphics, yet it still presents itself with many open problems. Among them, the creation of virtual worlds is a very challenging one. Because of its diversity and specificity, the creation of virtual worlds cannot be easily automatized, while the large number of elements makes manual editing long and tedious.

Procedural methods have been for long the most popular choice for the generation of large amounts of consistent data from reduced user input [Smith 1984]. While many successful procedural methods were proposed for different domains (terrains, forests, cities, etc.) and some were combined to synthesize complex virtual worlds [Smelik et al. 2011], these methods cannot automatically respond to the designer's intent. The user then finds himself in tedious trial-and-error sessions, with no guarantee that the intended result will be finally produced. When a specific virtual world needs to be created, the user thus ends up manually manipulating a large quantity of objects, such as trees to compose a forest, or complex structured objects like houses and street segments to compose villages. In addition to taking care of scene design, the user must ensure scene consistency, which potentially breaks the artistic flow.

Just very recently, various approaches for inverse procedural modeling have been introduced [Talton et al. 2011; Vanegas et al. 2012; Št'ava et al. 2014]. Their common goal is to learn rules, or their parameters, in order to find a procedural representation of some input structure. The *key observation* of our paper is that we do not need to learn the procedural parameters of a complete scene. This task can be done instead in an interactive manner, on sub-parts of the scene or for separate categories of objects. In such a scenario, the user interactively selects parts of scenes to analyze, their parameters are learned from intermediate representations such as distributions, and they are then used to synthesize more content through interactive operations. This ability to build individual "palettes" of parameters and to reuse them during scene editing brings a simple form of inverse procedural modeling into an interactive artistic workflow.

We present a novel concept for the interactive design of consistent virtual worlds. We exploit the intuitiveness of state-of-the-art painting and editing tools that we combine with the power of procedural modeling. On the one hand, we overcome the main problem of procedural modeling by *learning parameters for intermediate representations* of procedural models from examples. This task is done interactively and locally. On the other hand, we overcome scalability issues of interactive methods by incorporating procedural models into the design process. Using the parameters of an intermediate representation learned for some example scenes, the user can freely create consistent distributions of objects and of structured graphs using procedural brushes, or edit scenes with a variety of operations such as move, copy-paste, or stretch. This does not prevent accurate local editing, where we aim at maintaining the manually specified content when painting other categories of objects on top of previous ones.

Figure 1 shows an example of the usage of our method. An input scene (left) is used to learn the spatial distribution of the input objects, which is then fed as parameter values to a paste tool used to generate a novel scene (center). Note that the scene generation is environmentally sensitive, i.e., it positions objects while considering the context and the other exemplars used for the generation. Our approach comes with several particular usages of the procedural parameter learning. The example in Figure 1 (right) shows how an interpolation in parameter space is used to generate a novel distribution of islands populated with trees.

Our main contributions include:

- the interactive synthesis of virtual worlds by using painting-inspired tools to interactively extract local statistical parameters from examples and storing them as *distributions* in a palette (Section 3);
- the extension of a form of inverse procedural modeling to complex content, where both distributions of individual elements such as rocks, houses, or trees, and structured content such as roads or rivers, are jointly accounted for, together with external constraints such as the terrain's slope (Section 4);
- the extension of state-of-the-art tools provided in interactive painting systems, such as brushes and pipette tools, copy-paste, move, stretch, color blend and gradient, to the synthesis of virtual worlds (Section 5).

2 Related Work

Our work belongs to procedural modeling, and more precisely to inverse procedural methods that learn parameters from user input. It is also related to example-based texture synthesis and sketch-based modeling techniques.

Procedural modeling automates 3D content creation by generating geometric models from generative rules and processes. It has been successfully applied to a wide variety of object categories, such as plants [Měch and Prusinkiewicz 1996], terrains [Génevaux et al. 2013], buildings [Müller et al. 2006], networks of streets and roads [Galin et al. 2011], cities [Parish and Müller 2001], and villages [Emilien et al. 2012]. It is the topic of a recent survey [Smelik et al. 2014]. Yet, specifying generative rules with a new type of content in mind is difficult, and modifying results usually requires re-launching the process with different input parameters. This indirect control has prevented a wider use of procedural modeling tools, despite their efficiency for quickly creating large sets of consistent content.

Our method uses procedural modeling to generate content, and allows a user to manually modify the generated content. It also uses a simple form of inverse procedural modeling to learn statistical models from the generated content, and to adapt new content generation to these learned models.

Inverse procedural methods aim at easing the use of procedural models by automatically inferring input parameters from user-defined output or constraints. Talton et al. [2011; 2012] describe a statistical approach for learning the parameters of an existing procedural model by using Metropolis procedural modeling and Bayesian networks. Unfortunately, this general solution is too slow for being used in an interactive modeling system. More specific methods have been proposed for the inverse procedural modeling of 2D vector graphics [Št'ava et al. 2010], man-made objects [Bokeloh et al. 2010], cities [Aliaga et al. 2008; Vanegas et al. 2012], and trees [Št'ava et al. 2014]. These methods are restricted to their specific category of content; they do not enable the inverse modeling of complex scenes where several types of content affect each other. Close to our work is a recent paper of Xing et al. [2014], who present an approach for autocompleting stylized repetitions learned from an input model. However, their approach is limited to 2D polygons. In image analysis, Lafarge et al. [2010] use point processes to learn spatial distributions of points for the extraction of geometric features. In our work, we also learn spatial distributions of points, but instead with the objective of a new scene generation.

Our method is related to a simple form of inverse procedural modeling with its statistical models, but in contrast to previous work, it enables joint statistical models for the generation of procedural models for structured and unstructured content, while modeling their interactions. This is done in real time, facilitating the use of the method in an interactive modeling system. We extend approaches developed for 2D texture synthesis, presented next.

Example-based texture synthesis methods generate large, self-similar textures from examples [Ashikhmin 2001; Dischler et al. 2002]. They have been applied to point and 2D element arrangements [Ijiri et al. 2008; Hurtut et al. 2009; Öztireli and Gross 2012; Landes et al. 2013], to structured content (graphs) such as city layouts [Aliaga et al. 2008; Lipp et al. 2011] and branching structures [Sibbing et al. 2010], and to terrains [Zhou et al. 2007]. Some of these methods directly reuse the content in the input examples to generate their results, as is typically done for terrains. In contrast, vector texture synthesis methods rely on the statistical analysis of the input examples to generate visually similar ones.

Our approach belongs to example-based modeling of vector content, and builds on the methods by Aliaga et al. [2008] and Hurtut et al. [2009] to respectively handle graphs and point distributions.

Sketch-based modeling methods allow for an intuitive and interactive modeling experience [Olsen et al. 2009]. They have been applied to various components of virtual worlds, such as terrains [Gain et al. 2009; Tasse et al. 2014] and vegetation [Wither et al. 2009; Longay et al. 2012], but since they rely on preexisting procedural models, these methods limit the variety of possible outputs. Artistic brushing techniques make sketching more expressive by enabling the input of complex strokes with width, angle, and/or pressure. These techniques have been applied to a variety of specific content, such as pen-and-ink illustrations [Kazi et al. 2012], textures [Sun et al. 2013], images [Lu et al. 2014], and short animations [Milliez et al. 2014]. We consider that sketching relies more on pen-like sketches of 2D lines to control or produce results. In the context of a painting metaphor, we consider that these methods rely more on "brushes" used to paint generated content "following" a given statistical model within the area covered by the brushing motion. Similar advanced interactions are already available in commercial softwares, such as *Adobe Illustrator*, which provides a variety of tools, including "pattern brushes" that create object distributions. However, they are limited to the parameterization of simple rules for a unique object category, do not take into account existing objects, and are not adapted to complex virtual world editing.

Our method reuses content creation through a painting metaphor, but extends it to complex virtual worlds that include both distribu-

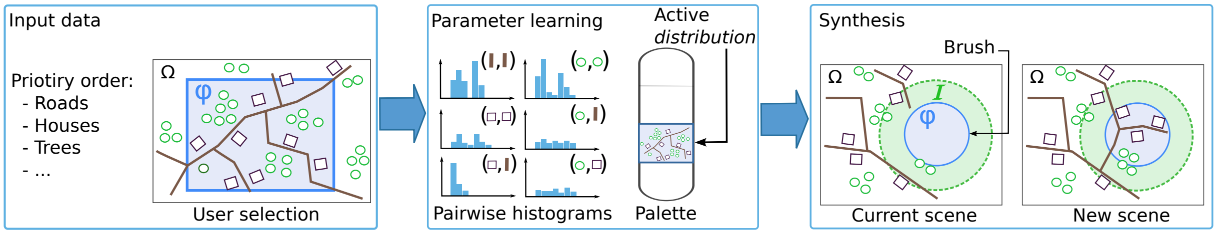

Figure 2: *Method overview. The system takes as input a priority order for categories of scene objects and small scenes exemplars, that may be either imported or created interactively (left). The algorithm learns parameters from selected exemplars, in the form of pairwise histograms linking each category of content with all lower-priority ones. The resulting sets of histograms are stored as one new distribution in a palette (center). Distributions can be interactively applied with brushes to create new content (right), or reused in a variety of other content editing tools inspired from state-of-the-art painting systems.*

tions (trees, houses, etc.) and structured elements (roads, rivers, etc.), enabling the user to paint consistent complex content.

3 Overview

We introduce an example-based synthesis algorithm for virtual worlds, embedded within an interactive editing system based on a **painting metaphor**. Our algorithm captures distributions of individual scene objects (e.g., trees, rocks, or houses positioned on an arbitrary terrain). We also encode more structured content (e.g., roads, rivers) that is represented by graphs. These representations are used to synthesize similar-looking content by analyzing interactions between different categories of content, and their relation with external data, such as a terrain's local elevation or slope. The resulting procedural model is embedded within an interactive editing tool enabling the user to seamlessly paint and edit coherent virtual worlds.

Figure 2 shows an overview of our method. The canvas of our system is a virtual world, from which regions can be selected and used as exemplars. In a first step, parameters of distributions from the procedural generation and manual edition are learned from these exemplars by using statistical models, and each resulting set of parameters is stored as a ***distribution*** in a **palette**, by analogy to colors in a painting system. During an interactive session, the user reuses these *distributions* to synthesize new content with the current active *distribution*, similar to using a brush tool to apply the active color.

Scene objects are generated category by category, generally from larger ones to smaller ones, so that complex (larger) objects can be generated with less constraints during the synthesis step. Consequently, we set a predefined **category priority order**, for instance, stating that houses are built after roads. This ordering simplifies the modeling and also reduces computation time during the learning stage, because only interactions between a category and the preceding (lower-priority) categories in the priority list need to be analyzed.

The analysis and synthesis steps are interleaved during an editing session. *Distributions* are created as needed, and they are updated and modified from one or multiple previous example worlds, or from the edited scene at an earlier stage.

Input data. Let Ω denote the virtual world space that is composed of *scene objects* positioned on a terrain, each belonging to a specific category $\mathcal{C} \in \{$*ground, island, mountain, hill, river, lake, road, castle, house, farm, pine tree, orchard tree*$\}$. Depending on their category, scene objects may correspond to different data types $\mathcal{T} \in \{$*distribution, graph, external*$\}$. Categories corresponding to sets of individual objects are analyzed and synthesized by using *distribution* models. More structured content, such as roads and rivers, belongs to the *graph* type. Maps, such as of elevations or slopes, are assigned to the *ground* category and their type is called *external*, since they define external constraints for scene objects. The object categories can be grouped into a smaller set of layers $\mathcal{L} \in \{$*terrain, water, settlement, vegetation*$\}$, limiting the editing, if needed, to specific layers that are shown in Table 1.

Parameter learning. Our method learns from examplars. The user can either import existing scenes, possibly created from real data, photos, or artwork, or create scene exemplars by manually positioning elements. The user then selects an *active region* $\varphi \subset \Omega$ to be analyzed. In our prototype, we provide three types of regions: axis-aligned bounding boxes (default), circles, and polygons defined by sketching a nonintersecting closed contour. The latter eases the selection of complex exemplars from larger scenes.

The statistics learned from an exemplar are pairwise *interaction histograms* that serve as a set of parameters for a procedural synthesis method generating visually similar content. Some of the histograms represent interactions within a given category of content (such as the ones quantifying spatial distributions of trees), while the other histograms represent interactions between a category and a preceding category in the priority list. The resulting set of histograms is called a *distribution*. A *distribution* is displayed within a palette using a drawing of the input exemplar, to ease subsequent user selections.

Synthesis. *Distribution* parameters serve as input for a procedural model that generates content sharing the same statistical features as the exemplar. *Distributions* can be applied using brushes or reused within other interactive editing tools, such as the ones enabling *distribution* interpolation, *distribution* gradient, moving or stretching operations.

An important aspect of the editing tools is that they are environmentally sensitive [Měch and Prusinkiewicz 1996] and they ensure content consistency with the surrounding. In addition to the user-defined active region $\varphi \subset \Omega$, where the tool is applied, we automatically compute a *region of influence* $\mathcal{I}$ defined as an offset of φ, i.e., $\mathcal{I} \cap \varphi = \emptyset$ (see Figure 2 right). The scene objects preexisting in this region of influence are taken into account when the tool is applied, ensuring consistency. For instance, graph objects are able to connect with their surrounding when a brush tool is applied. The offset radius $r_{\mathcal{I}}$ can be interactively changed or disabled.

4 Inverse Modeling of Virtual Worlds

Various inverse procedural modeling methods exist for both point distributions and for graphs. However, the challenge is to adapt these methods so that the types of content and their mutual interactions can be jointly considered, as well as their response to external

$\mathcal{L}$	terrain				water	settlement				vegetation					
$\mathcal{C}$	ground	island	mountain	hill	river	road	castle	house	farm	big rock	pine tree	red tree	rock	small rock	grass
$\mathcal{T}$	external	dist.	dist.	dist.	graph	graph	dist.	dist.	dist.	dist.	dist.	dist.	dist.	dist.	dist.

Table 1: *The object categories $\mathcal{C}$ used in our examples with their corresponding layers $\mathcal{L}$ and types $\mathcal{T}$ ("dist." stands for distribution).*

constraints.

4.1 Distributions and Their Interactions

We use Metropolis-Hastings sampling [Geyer and Møller 1994; Hurtut et al. 2009] for synthesizing distributions of elements from statistical data learned from examples, because this intermediate model enables to take several interactions into account at once. Let $f(X)$ be a known probability density function of a given point distribution $X = \{x_1, ..., x_{n_X}\}$ of category $\mathcal{C}_X$. A distribution matching f in a region φ of area $\mathcal{A}$ is synthesized by initializing X as a random distribution. Then a fixed number of iterative birth-and-death perturbations is performed, with the same probability for birth or death. A change from X to the new arrangement X' is accepted with probability R_b for a birth, and R_d for a death, where:

$$R_b = \frac{f(X')}{f(X)} \frac{\mathcal{A}}{n_X}, \qquad R_d = \frac{f(X')}{f(X)} \frac{n_X}{\mathcal{A}}. \tag{1}$$

The probability density function $f(X)$ is computed from the statistical properties of category $\mathcal{C}_X$, but also from the interactions between $\mathcal{C}_X$ and all preceding categories of elements $\mathcal{C}_{Y_k} < \mathcal{C}_X$ in the predefined priority order, where Y_k is a distribution of elements with higher priority. We express it as

$$f(X) \propto \prod_{\mathcal{C}_{Y_k} \leq \mathcal{C}_X} \prod_{x_i \in X} \prod_{y_j \in Y_k} h_{X,Y_k}(d(x_i, y_j)),$$

where h_{X,Y_k} is a probability function set to a normalized histogram that measures interaction between categories $\mathcal{C}_X$ and $\mathcal{C}_{Y_k}$, and d is the Euclidean distance normalized by the width of the bins used in the histogram.

We use three ways of computing h_{X,Y_k} from the input exemplar, depending on the types of data in the two categories: if Y_k is of type *distribution* (and in particular when $Y_k = X$), we use a *radial distribution function* h_{rdf} adapted from the work of Öztireli and Gross [2012]; it was chosen for its ability to capture point clusters. If Y_k is of type *graph*, we use a new *distance-to-graph* histogram h_g, and a *map sampling* histogram h_m if Y_k is of type *external*. We describe these three histograms next.

Radial distribution function. Given two input distributions X and Y of respectively n_X and n_Y elements, and the area A of the analyzed region φ, the histogram is computed as follows [Öztireli and Gross 2012]: For each scene object in X, we count the number of objects from Y that are in an annular shell of inner radius $r = d_r k$ and thickness d_r around it, and add this number to $h_{rdf}(k)$ (see Figure 3 left). The result is multiplied by a normalization factor (where dA is the area of the annular shell) to get valid probability values:

$$h_{rdf}(k) = \sum_{x_i \in X} \sum_{\substack{y_j \in Y \\ kd_r \leq d(x_i, y_j) < (k+1)d_r}} \frac{A}{dA\, n_X\, n_Y}. \tag{2}$$

Moreover, we adapted this method to overcome the window-edge effect. Edges of the selected region often cross the annular shells when manipulating small regions with few objects (Figure 3, center). When h_{rdf} is evaluated for a point near the window border, fewer objects are accounted for than for a point at the center, because it can only consider objects in $dA \cap \varphi$. Our solution is to approximate the area $dA' = dA \cap \varphi$ by computing for each point the proportion $\alpha_{x_i,r}$ of the outer circle's circumference that intersects the active region, and by setting:

$$dA' = \alpha_{x_i,r}\, dA.$$

We replace dA by dA' in Equation (2).

While radial distribution functions (*rdf*) are efficient for modeling isotropic point distributions and point clusters, they are not appropriate for anisotropic distributions, such as aligned trees or road nodes. To capture them, we adapt h_{rdf} to an oriented *rdf*, h_{ordf}, by counting points in specific angular ranges in addition to distance ranges (Figure 3 right), resulting into a 2D histogram.

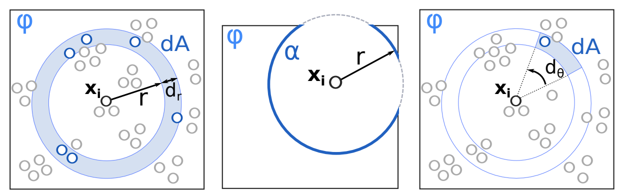

Figure 3: *Left: Radial density function. For a point x_i and a radius $r = kd_r$, we count the number of points within a range of distances $d \in [kd_r, (k+1)d_r[$ of x_i. Center: Window-edge effect correction. We evaluate α as the circumference of circle $C(x_i, r)$ that intersects the active region φ. This value is used to normalize the integration region dA. Right: Oriented radial density function. The range is reduced to points with an angle ranging in $d_\theta \in [ld_\omega, (l+1)d_\omega[$.*

Distance-to-graph is the normalized histogram h_g of the minimal distance from an object in X to all the edges of the graph in Y. For each scene object in X, we compute the closest distance d to an edge and add one to the corresponding part of the histogram, i.e., to $h_g(k)$ such that $kd_r \leq \min\{d(x_i, y_j), y_j \in Y\} < (k+1)d_r$. All the resulting values are divided by the number n_X of objects for normalization. This probability function enables us to capture the interaction between distributions of elements (e.g., houses or trees) and structured content (e.g., roads or rivers). See Figure 4.

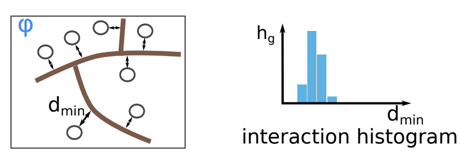

Figure 4: *Left: A graph and a distribution of points (circles). For each point we compute d_{min}, the closest distance to the graph. Right: Corresponding histogram h_g of d_{min} values.*

Map sampling is the histogram h_m of the values defined on a given map, sampled on the distribution X. For each scene object x_i, we look at the map value at the same position $m(x_i)$, and add it to the corresponding part of the histogram, i.e., to $h_m(k)$, such that $kd_r \leq m(x_i) < (k+1)d_r$. The histogram is normalized again by dividing all values by n_X. See Figure 5. The resulting probability function enables us to analyze, for instance, the positions of trees and rocks relatively to the local slope of a rugged terrain.

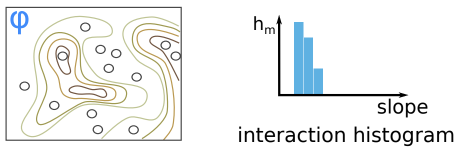

Figure 5: *Left: An elevation map (represented as isocontours) and a distribution of points (circles). Right: Histogram h_m of the slope values at each point.*

4.2 Graphs and Their Interactions

Procedural model. We have modified the inverse procedural model from the work of Aliaga et al. [2008] for synthesizing graphs from examples (Figure 6). The synthesis method first generates a distribution of nodes. Paths are then progressively generated by using a random walk, where edges are constructed from a node to its closest neighbors. The connecting node is chosen based on the statistics of edge lengths and angles along the path.

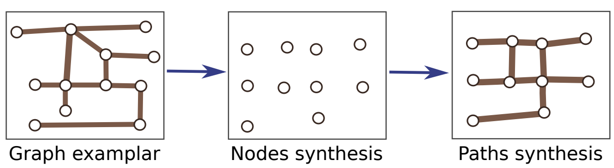

Figure 6: *Procedural graph synthesis. Left: Input graph examplar. Center: New node distribution. Right: New edge generation.*

Graph nodes are generated by using the previously described distribution synthesis method. This enables us to model interactions between graph nodes and other categories of content with higher priorities. First, we add two extra constraints: we do not allow new edges that would collide with existing content. Second, we do not allow new edges if they form an angle smaller than the minimal intersection angle of the examplar with the other edges connecting to the same node, to prevent acute angles at road intersections. Both the statistics and this minimal angle are extracted from the exemplar (see Figure 7).

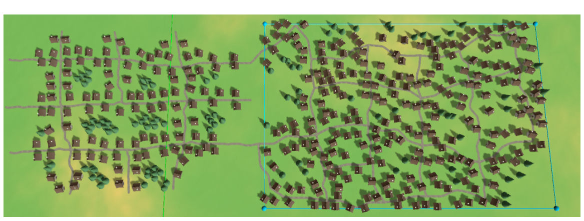

Figure 7: *Village synthesis. Distribution from an exemplar (left) is used to generate a larger village (right).*

Parameter learning. Similarly to the histograms computed for point distributions, we compute normalized histograms of edge lengths and of angles along a path in the input graph. To be able to define successive angles along a path, the input graph is represented as a set of paths that may possibly intersect (such as a set of intersecting roads in a city layout). Nodes are automatically added at the intersection points between these paths before computing the interaction histograms associated to the distribution of graph nodes.

4.3 General Synthesis Algorithm

In the previous steps, we have described how all the histograms are extracted from an example scene. They are stored as one *distribution* in the palette. New content is created within an active region φ using the *distribution*. We successively generate scene objects in this region, each category at a time, using our predefined priority order. For distributions generation we use Metropolis-Hastings Sampling (MHS) (Section 4.1), while graphs are created using MHS for nodes generation and a marching algorithm for edges (Section 4.2).

To enable local synthesis while maintaining consistency with neighboring regions, we apply MHS in the active region φ, but include all the elements in the region of influence $\mathcal{I}$ when evaluating $f(X)$ and $f(X')$ in Equation (1). This allows for adding or suppressing points depending on neighboring data. Moreover, when creating graph edges, we include nodes in the region of influence as candidates for new neighbors along a path, which avoids the generation of too many disconnected components.

5 Example-based Editing

In the previous section, we detailed how we analyze and synthesize scene elements that are used for the generation of new virtual worlds. In this section, we present new operators that are used to edit virtual worlds using example-based synthesis. Our operators are inspired by editing operators commonly used in painting systems, such as copy-paste, color interpolation, gradient, brushes, move, and stretch. In fact, our operators can be thought of as a *generalization* of those operations, because the common operations are their special case.

5.1 Copy and Paste

Painting systems use the copy-paste operation to duplicate a portion of a scene. This is less suitable for randomized parts of virtual worlds, which usually exhibit statistically similar details, but never identical arrangements.

Our approach to the copy-paste operation is not to copy the exact arrangement of elements, but the *distribution* of elements, i.e., the interaction statistics computed from the analysis of the underlying selected part of the scene. When pasting, a new arrangement is generated, with the statistical properties from the exemplar. In this way, each region will be unique, but similar to the input.

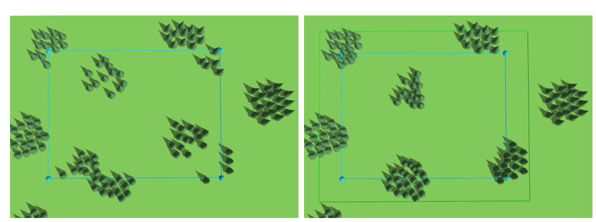

Figure 8: *Pasting without any region of influence results in visually discontinuous clusters (left), while using a region of influence achieves consistent blending with the surroundings (right).*

To perform the copy-paste operation (see also the accompanying video), the user first traces the contour of the region φ_c to copy. The elements in φ_c are analyzed, a new *distribution* is created, and stored in a palette. The user then traces a new active region φ_p for the destination. All the objects in this region are removed, as well as graph edges that intersect this region, and the synthesis method from Section 4 generates new objects. This operation is environmentally sensitive, i.e., it takes the region of influence around φ_p into account for consistency with the surrounding, as illustrated in Figure 8.

Note that φ_c and φ_p can have arbitrary shapes and sizes. However, when copying, the size of the selection region φ_c can have a major impact on the learned histograms, and therefore on the result of a copy operation, as illustrated in Figure 9.

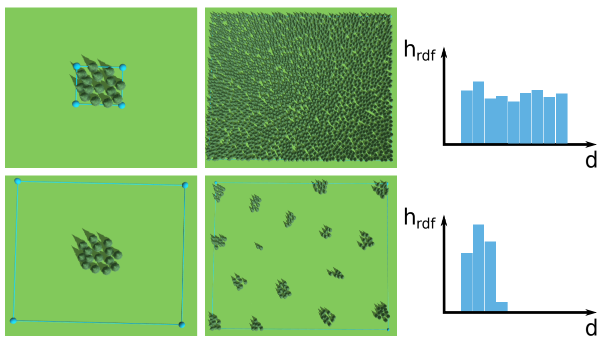

Figure 9: *The same input scene can be used to generate a dense forest (top) or isolated tree clusters (bottom) depending on the selected region (blue contour on the left).*

5.2 *Distribution* Interpolation and Gradients

Following the analogy with painting systems, we provide tools that blend our procedural *distributions* and allow creation of gradients and brushes with opacity masks.

***Distribution* interpolation.** Let us recall that a *distribution* is a set of interaction histograms computed by analyzing an exemplar. *Distribution* interpolation means an interpolation of corresponding histograms.

The histogram interpolation is achieved by using optimal mass transport within the analyzed interaction histograms [Read 1999], which is based on the inverse cumulative density function. Contrary to direct value interpolation, this method performs a more intuitive interpolation of histogram shapes (see Figure 10). Indeed, a naïve interpolation of two Gaussian shapes does not result in a Gaussian shape, but in a fade-in-fade-out between the two shapes. In contrast, mass transport interpolation results in a Gaussian shape with parameters being interpolated from the two initial shapes.

Gradients. Our approach to *distribution* interpolation can be extended to a gradient tool that takes two *distributions* as input, and generates object distributions with a *distribution* interpolated at different levels within the active region. *Distribution* gradient is a generalization of *distribution* interpolation with histograms being interpolated with respect to the position of the evaluated point x_i, i.e., during the synthesis process, $f(X)$ is computed by using a different interpolated *distributions* for each point $x_i \in X$. The value of the interpolation coefficient t is determined by the position of x_i in the active region φ. For instance, for axis-aligned bounding box regions, t is the projection of x_i on the box axis, normalized by its width; for circular regions it is the normalized distance to the center.

To speed up calculation we precompute N interpolated *distributions* and we select on the fly which of them is to be used depending on the point position. N can be a fixed value, or be adaptively calculated from the size of the region. Figure 11 shows an example of gradient application to tree distribution over isolated islands.

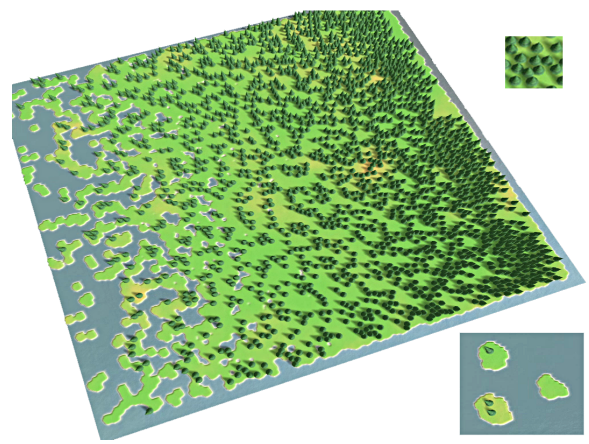

Figure 11: *An example of linear gradient operator applied with the two exemplar distributions in insets.*

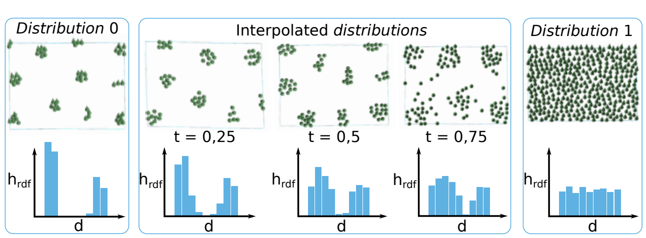

Figure 10: *Interpolation between two distributions using parameter t. Mass transport is used to interpolate between the start and end histograms (bottom), resulting in the expected visual transition (top).*

5.3 Brushes

A key aspect of the interactivity of our method is the ability to paint distributions and structures directly into a scene. We show in this section (and in the accompanying video) how to use the operations presented previously to design a new interactive painting-like tool for modeling virtual worlds.

We use procedural brushes, where the region for synthesizing evolves along the brush path. Within the path we compute a number of synthesis steps, each of them taking into account the previously synthesized regions, similarly to the gradient tool (Figure 12).

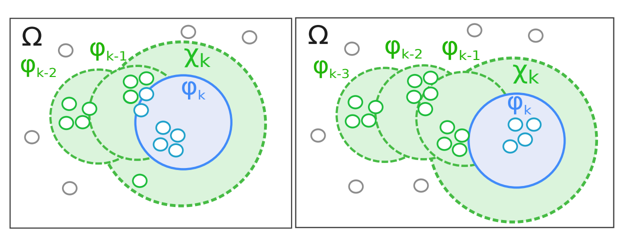

Figure 12: *Two successive steps of a brushing gesture. The blue points are synthesized in φ_k, while taking into account $\mathcal{I}_k = \mathcal{X}_k \cup \{\varphi_i, i \leq k\}$. The new points respect the distance properties with the existing ones.*

During a single brushing gesture, all synthesis regions $\{\varphi_k\}$ are stored. When the brush moves sufficiently far along the brushing path (user-defined parameter), we remove all existing objects in φ_k except those objects in φ_i, for $i < k$. Then we synthesize new objects in φ_k while considering the region of influence $\mathcal{I}_k = \mathcal{X}_k \cup \{\varphi_i, i \leq k\}$. See Figure 13.

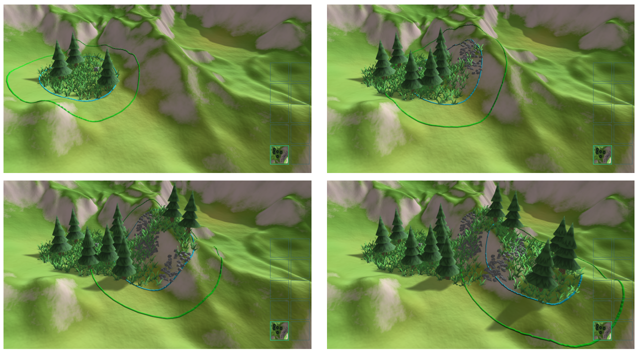

Figure 13: *The brush operator in action (from left to right, top to bottom): objects are successively created along the path followed by the brush, while remaining consistent with the objects in the region of influence, also following the path.*

5.4 Move

If the user moves a region to a new location, the displaced objects should be constrained by the new surrounding objects and by the external constraints of the terrain at the new location, while preserving as much as possible the original arrangement. For instance, when moving a forest along a terrain, the underlying slope should affect the tree arrangement. The copy-paste operator cannot be used, because it would generate completely new arrangements at each displacement step, resulting in annoying visual flickering.

We provide a new algorithm that allows moving an existing selection. The goal of the moving gesture is to adapt a set of objects to a new environment. Our algorithm is based on the Metropolis-Hastings synthesis method. Starting with the objects within the region, we randomly choose an existing point p and apply to it small displacement perturbations, instead of births, to find a new location p'. We favor local displacements to preserve the visual appearance of the original object arrangement that is moved. The new position is accepted using the standard Metropolis acceptance probability (Equation (1)). By also performing death steps, objects with very small probability values are removed from the scene. When objects cannot be adapted and are removed, we allow the equivalent number of births to preserve the total number of objects that are being moved. Consequently, this algorithm enables local adaptation of scene objects to their new environment and removes inconsistent ones. We prevent the generation of new objects in order to best preserve the original arrangement, and to reduce popping of objects while moving. However, this can lead to a new set of objects not respecting the exemplar distributions, for instance having more or less trees than the new environment allows. In such situation, the user can still edit the results to fix the local appearance, or use the copy-paste tool to generate new sets of objects respecting better the exemplar distributions. Figure 14 shows an example of the translation of a selected region.

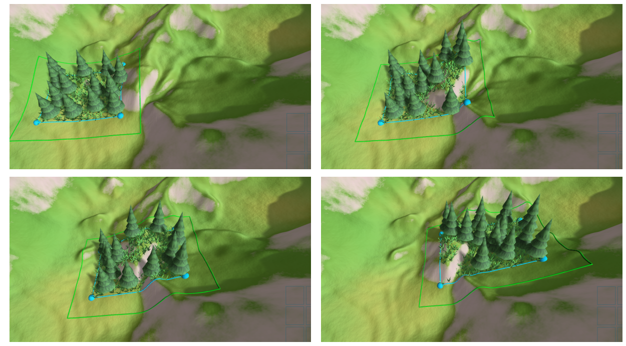

Figure 14: *Left to right, top to bottom: Moving a selection while maintaining constraints, and favoring object displacement rather than births and deaths, to increase temporal coherence.*

5.5 Seamcarving-based Stretching

Linear scaling deforms the arrangement of elements. Instead, we adapted the seamcarving algorithm [Avidan and Shamir 2007], that seamlessly deforms images, by finding paths (cuts) of least-energy in the original image (i.e., traversing smoothly varying pixels) and using these paths to remove pixels (scale down), or add pixels (scale up) with interpolated *distributions*. When the user scales a region in our system, our seamcarving algorithm finds a cut that minimizes scene deformation. We then only deform the objects near the path, thus preserving the rest of the scene.

Energy map. To identify the best cut within the scene, we compute the path of the least-energy within the scene, in the direction perpendicular to the main deformation. The energy function expresses the cost of cutting through the scene at a given location. Our scene is continuous so we first rasterize it into a 2D grid and calculate a repulsion energy $e(\mathbf{p})$ that is inversely proportional to the distance to the nearest objects:

$$e(\mathbf{p}) = \left(\min_{o_i \in O} \{d(\mathbf{p}, o_i)\} \right)^{-1},$$

where $o_i \in O$ is an object, i.e., a point in a distribution or an edge in a graph.

Resizing. To stretch a portion of a scene, we use a move operation for the objects on one side of a cut, and synthesize new objects in the free space left (Figure 15 top). To shrink, we fold the selected region onto itself along a cut by using a move operation (Figure 15 bottom). Figure 16 shows an example of scene stretching.

6 Implementation and Results

We developed a prototype of the interactive application that implements the methods and algorithms discussed in this paper. Our system is implemented in C++ and uses OpenGL. All results were generated on an Intel® Xeon® E5-1650 CPU, running at 3.20 GHz with 16 GB of memory, and rendered with an NVidia 660GTX GPU. All analysis and synthesis computations were performed on a single CPU thread. The terrain geometry and the stylized map are procedurally generated on GPU using GLSL shaders.

Optimizations and considerations. We have taken advantage of a few ways to optimize synthesis. First, synthesizing element categories in order of priority is more efficient than creating them

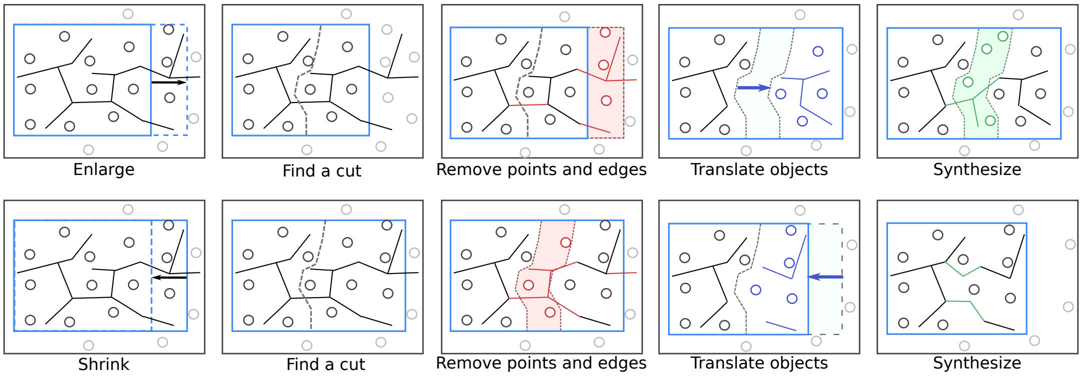

Figure 15: *Stretching (top, from left to right) and shrinking (bottom). The best cut is first computed. Stretching: the region at the right of the cut is moved to the right (which removes overlapping objects), and new objects are synthesized in the free space using a distribution extracted from the moved region. Shrinking: objects within the cut region are removed, those to the right are translated while taking their new surrounding into account, with the synthesis of new arcs.*

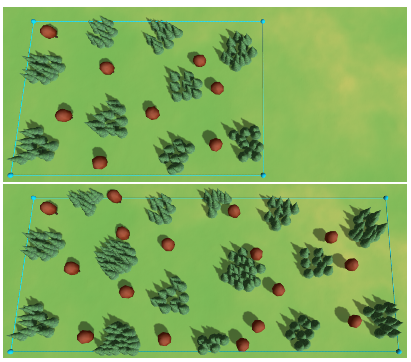

Figure 16: *Seamcarving-based stretching. Top: Initial arrangement. Bottom: New trees have been seamlessly inserted in the empty region, preserving the visual appearance of the distributions.*

within a single loop, because only interactions with the already-created categories of elements are taken into account when computing $f(X)$. Second, the acceptation ratio of a birth in the Metropolis-Hastings algorithm (Equation (1)) can be optimized as follows. Since f is defined by a product of probabilities, and X and $X' = X \cup \{p\}$ are almost identical, the acceptation ratio can be simplified:

$$\begin{aligned} R_b &= \frac{\mathcal{A}}{n_X} \frac{f(X')}{f(X)} \\ &= \frac{\mathcal{A}}{n_X} \frac{\prod_{\mathcal{C}_{Y_k} \leq \mathcal{C}_X} \prod_{x_i \in X'} \prod_{y_i \in Y_k} h_{X,Y_k}(x_i, y_j)}{\prod_{\mathcal{C}_{Y_k} \leq \mathcal{C}_X} \prod_{x_i \in X} \prod_{y_i \in Y_k} h_{X,Y_k}(x_i, y_j)} \\ &= \frac{\mathcal{A}}{n_X} \prod_{\mathcal{C}_{Y_k} \leq \mathcal{C}_X} \prod_{y_i \in Y_k} h_{X,Y_k}(p, y_j). \end{aligned}$$

A similar simplification applies to the death ratio. Consequently, when performing a small perturbation of an arrangement (i.e., adding or removing a few elements), only the probability associated to these elements need to be computed, which greatly reduces computation time.

Since islands are modeled using dense point clusters in our method (see Table 1), while they are generally represented as closed contours in the input scenes, we provide an automatic method for creating dense point clusters from closed contours: we generate a dense set of points in the region within the contours using a jittered grid. Although computed as new dense clusters of particles, synthesized islands and lakes are rendered as contours by tracing a contour around dense clusters. This keeps our model transparent to the user.

Our implementation achieves interactive feedback for example-based synthesis of virtual worlds that include various types of objects, structured or not, while capturing their interactions. While using this inverse method alone would only generate homogeneous scenes, the way we embed it within an interactive painting system, provides the user with a variety of intuitive editing tools, enabling both coarse and fine scene editing.

Examples. Figures 1, 18, and 19, as well as the accompanying video show examples generated with our prototype modeling system. We applied our methods to several example scenes, including 2D scenes built from imported artworks in the form of ancient maps, and 3D vector virtual worlds composed of a variety of elements. Each tool was used on several occasions on several examples.

The large scene synthesis in Figure 18 applies our system to the design in a style of an ancient map. The user imported a vector map (top left) into our system (top right), from which several *distributions* were created and were then used for editing the map (bottom).

Figure 19 shows an example of a virtual world fully created and edited with the methods presented in this paper. For instance, we used gradients to create tree and island distributions, some larger islands were generated with brushes, while roads and small settlements were created using example-based copy-paste. The entire scene (more than 4000 objects) was created in less than two hours.

User evaluation. Our system was tested by fourteen users, including three professional computer artists, one of which with experience on designing ancient-looking vector maps with standard software. Each session started with a 5-minute introduction to the system and its tools. After a short training each subject was asked to recreate an existing scene, composed of mountains, lakes, forests, houses, and orchards in an allocated time of 20 minutes. Then we presented to the users a scene with mountains, and asked them to reproduce a given pattern of trees, rocks, and grass on a large part of the terrain. We asked them to first place each element manually and then using our tools. We measured both times to evaluate the efficiency of our method. After, we asked the users to answer a questionnaire that evaluated the tools and we asked them to provide their general experience. We used four-value scale (1-strongly disagree, 2-disagree, 3-agree, 4-strongly agree).

The results confirmed the ease of use of the different tools and the potential for high impact of our concept. The users enjoyed using the tool, and confirmed that it was easy to use. Moreover, they felt that the tools helped them to be more creative. They appreciated the scene editing efficiency. Their creation task took around 1.9 minutes with our tool and 3.7 minutes manually. The experienced artists declared that our method outperforms classical modeling techniques and would be a great integration in editing systems. More precisely, the users enjoyed the variations produced by our synthesis process, where objects are not copy-pasted, but their statistical properties are used to create new content. They also enjoyed the automatic slope consistency respect when groups of objects are moved over a rugged terrain. The results of the questionnaire were: *"I enjoyed using the method"*: 3.5, *"The method is easy to use"*: 3.5, *"The method helped me to be creative"*: 3.0, *"The method allowed me to create content efficiently"*: 3.4. Moreover, the experienced artists claimed about the map editors: *"This method outperforms classical modeling techniques/editors"*: 3.8.

Our ability to capture interactions between categories of objects rather than only distributions (enabling for instance to create rocks surrounded by grass) was noted as really useful. The map-specialized artist, who used to manually compose maps with *Adobe Photoshop*, was quite impressed. She thought that our system can be a very good tool for composing complex maps. With *Adobe Photoshop*, she had to spend hours designing small islands along a continental coast; she could do it within a few minutes with our tools.

Performance. The analysis of a complex scene is fast, and generally takes less than a few milliseconds. Synthesis, even for large scenes with dense distributions, is usually completed under a few seconds, thus allowing for interactive design. Because objects interact only with other objects of a higher priority, the complexity is $\mathcal{O}(m(m+1)/2) = \mathcal{O}(m^2)$ for m categories. The actual detailed complexity of each operation depends on the number of objects interacting and on the actual algorithm.

We observed that in general, thanks to our extension to radial basis functions (Section 4.1) to overcome window-edge effects, our method is robust to exemplars with only a few (about 10) objects. However, the more complex the interactions between objects of different types, the more cases in exemplars are needed to capture their joint distributions.

To improve computational time, we have used the density of a given category of objects to parameterize the number of iterations for synthesis: small groups of objects can be created in a few iterations, while a dense set of objects may require many iterations. This solution proved to be a good trade-off between accuracy and speed. However, it sometimes fails when objects distributed with low density have complex behaviors due to many interactions over a large region. In this case, more iterations would be needed. The number of iterations could alternatively be controled as a user parameter, to better match the artist's needs.

Table 2 shows statistics of the computation time for the examples from the paper.

Limitations. One limitation of our method is that the object types and categories need to be preset. A more flexible approach would be to analyze and synthesize scenes without *a priori* knowledge, similar to recent example-based synthesis techniques [Hurtut et al. 2009] or inverse procedural methods [Yeh et al. 2012]. The environmental sensitivity could go further into implementing more intricate relations. For example the egress rule for each house is hardcoded in our implementation, but could be learned from the context.

Figure	m	n	Synthesis	Terrain
Fig. 1 (center)	4	83209	13.9	–
Fig. 1 (right)	3	522	0.16	0.35
Fig. 7	4	235	0.13	–
Fig. 11	2	3936	0.71	0.42

Table 2: *Computation times (in sec) of our example-based synthesis algorithm, where m is the number of categories, and n the total number of synthesized objects. Terrain is the generation time of the heightmap or the stylized map, each of resolution 2048×2048.*

Another limitation is the fact that object distributions may have various properties within the same analyzed scene. Adding more granularity to the method would automatically segment the scene into cells and compute independent *distributions* in each cell. This approach would be particularly useful for the seamcarving operation: the latter currently computes a single *distribution* to fill the new part of the scene, while this region should rather been filled with several different *distributions*.

Last, our synthesis algorithms work well for natural scenes with chaotic arrangements, but are not adapted to complex scenes with lots of semantics in the arrangements, such as 2D vector art, or streets and other urban scenes. An example of this arrangement where the trees follow a curve is shown in Figure 17).

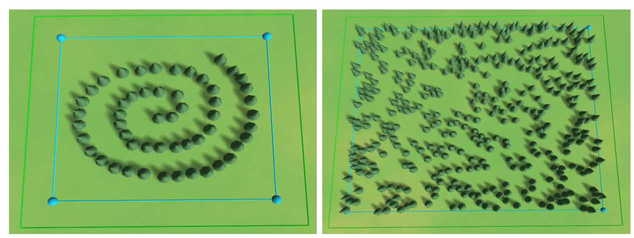

Figure 17: *An example of a failure case. The spiral arrangement of trees (left) cannot be reproduced by our system (right), since we only capture pairwise distributions of distances and orientations.*

7 Conclusion

We presented a framework for the interactive design of consistent virtual worlds. We exploit the intuitiveness of state-of-the-art painting and editing tools that we combine with the power of procedural modeling. We overcome the main problem of procedural modeling, by learning parameters of procedural models into statistical models from examples, and we overcome scalability issues of interactive methods by incorporating procedural models into the design process. Using the parameters learned from some example scenes, the user can freely paint consistent distributions of objects and structures using procedural brushes, or edit scenes with a variety of adapted operations such as move, copy-paste, or stretch. We demonstrate our framework on a variety of examples ranging from simple scenes edited with consistent procedural content generation, to the creation and reuse of large scenes.

Possible avenues for future work include a better model for inverse procedural modeling, where structures can be synthesized as grammars, or the integration of semantics to better synthesize complex scenes with meaningful arrangements, such as houses facing roads or gardens behind houses. The Metropolis-Hastings algorithm is the most time-consuming part of our implementation; it could be further optimized with a parallel implementation. Last but not least, virtual worlds are multi-resolution by nature: an interesting

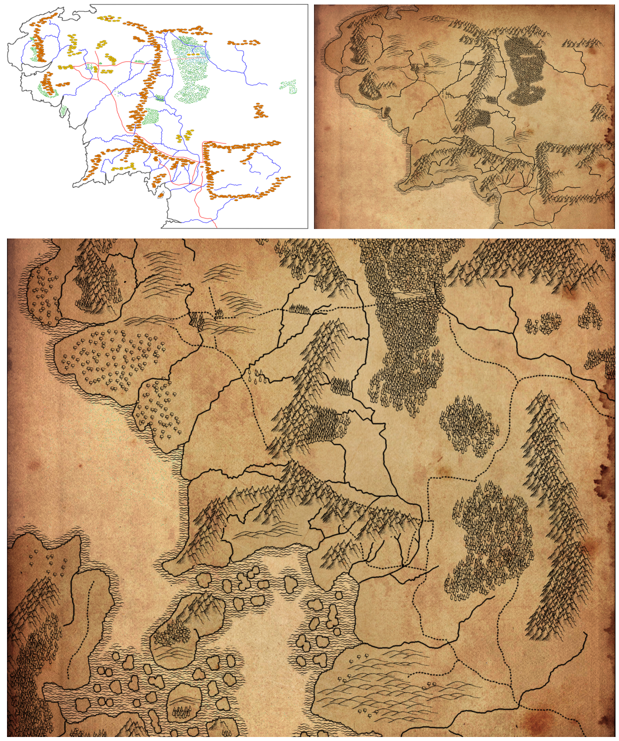

Figure 18: *Top: An example using vector input data (left) was imported in our system and rendered as an ancient map (right). Bottom: Manual editing by using our painting system enabled seamless insertion of the new content, with a perfect match between the new and the old parts.*

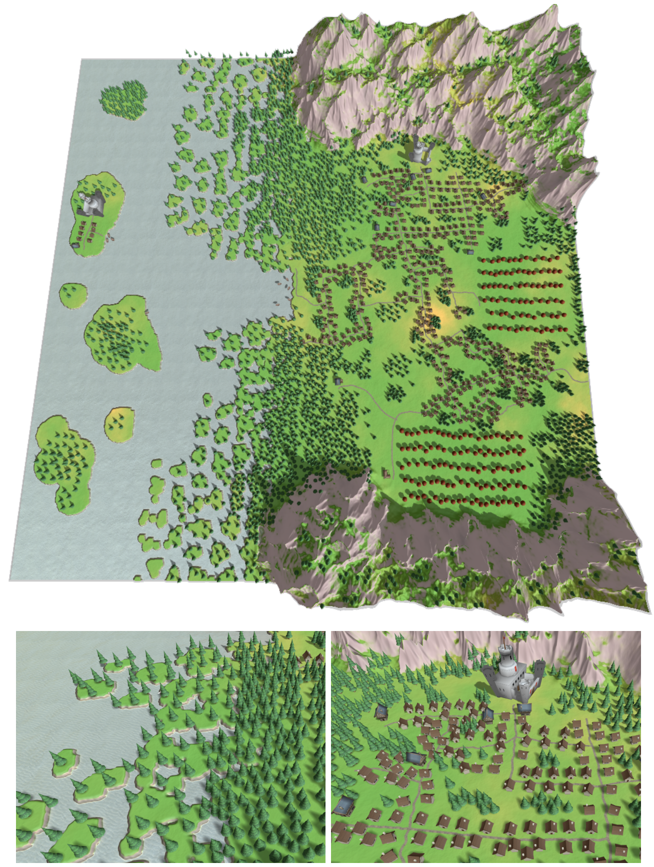

Figure 19: *Three views of a complex scene edited with our system.*

direction of research would be to generalize our concept to a multi-resolution framework, where the user could perform the operations at different levels of details, such as painting village distributions over a map, then zooming in to create distributions of streets and houses, and finally zooming out to again modify consistently entire villages.

Acknowledgments

The authors would like to thank the computer artists Laura Paiardini and Romain Testylier for their contributions to 3D models and 2D map textures. This work was partially funded by the advanced grant no. 291184 "EXPRESSIVE" from the European Research Council (ERC-2011-ADG_20110209). Emilien and Poulin also acknowledge financial support from GRAND.

References

ALIAGA, D. G., VANEGAS, C. A., AND BENES, B. 2008. Interactive example-based urban layout synthesis. *ACM Trans. Graph. (SIGGRAPH Asia) 27*, 5, 160:1–10.

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Proc. Symp. on Interactive 3D Graphics (I3D)*, ACM, 217–226.

AVIDAN, S., AND SHAMIR, A. 2007. Seam carving for content-aware image resizing. *ACM Trans. Graph. (SIGGRAPH) 26*, 3, 10:1–9.

BOKELOH, M., WAND, M., AND SEIDEL, H.-P. 2010. A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graph. (SIGGRAPH) 29*, 4, 104:1–10.

DISCHLER, J.-M., MARITAUD, K., LÉVY, B., AND GHAZANFARPOUR, D. 2002. Texture particles. *Computer Graphics Forum (Eurographics) 21*, 3, 401–410.

EMILIEN, A., BERNHARDT, A., PEYTAVIE, A., CANI, M.-P., AND GALIN, E. 2012. Procedural generation of villages on arbitrary terrains. *The Visual Computer (CGI) 28*, 6-8, 809–818.

GAIN, J., MARAIS, P., AND STRASSER, W. 2009. Terrain sketching. In *Proc. Symp. on Interactive 3D Graphics and Games (I3D)*, ACM, 31–38.

GALIN, E., PEYTAVIE, A., GURIN, E., AND BENES, B. 2011. Authoring hierarchical road networks. *Computer Graphics Forum (Eurographics) 30*, 7, 2021–2030.

GÉNEVAUX, J.-D., GALIN, E., GUÉRIN, E., PEYTAVIE, A., AND BENES, B. 2013. Terrain generation using procedural models based on hydrology. *ACM Trans. Graphics (SIGGRAPH) 32*, 4, 143:1–13.

GEYER, C. J., AND MØLLER, J. 1994. Simulation procedures and likelihood inference for spatial point processes. *Scandinavian journal of statistics 21*, 4, 359–373.

HURTUT, T., LANDES, P.-E., THOLLOT, J., GOUSSEAU, Y., DROUILLHET, R., AND COEURJOLLY, J.-F. 2009. Appearance-guided synthesis of element arrangements by example. In *Proc.*

Symp. on Non-Photorealistic Animation and Rendering (NPAR), ACM, 51–60.

IJIRI, T., MĚCH, R., IGARASHI, T., AND MILLER, G. 2008. An Example-based Procedural System for Element Arrangement. *Computer Graphics Forum (Eurographics) 27*, 2, 429–436.

KAZI, R. H., IGARASHI, T., ZHAO, S., AND DAVIS, R. 2012. Vignette: Interactive texture design and manipulation with freeform gestures for pen-and-ink illustration. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI)*, ACM, 1727–1736.

LAFARGE, F., GIMEL'FARB, G., AND DESCOMBES, X. 2010. Geometric feature extraction by a multimarked point process. *IEEE Trans. Pattern Analysis and Machine Intelligence 32*, 9, 1597–1609.

LANDES, P.-E., GALERNE, B., AND HURTUT, T. 2013. A shape-aware model for discrete texture synthesis. *Computer Graphics Forum (EGSR) 32*, 4, 67–76.

LIPP, M., SCHERZER, D., WONKA, P., AND WIMMER, M. 2011. Interactive modeling of city layouts using layers of procedural content. *Computer Graphics Forum (Eurographics) 30*, 2, 345–354.

LONGAY, S., RUNIONS, A., BOUDON, F., AND PRUSINKIEWICZ, P. 2012. TreeSketch: Interactive procedural modeling of trees on a tablet. In *Proc. Eurographics Symp. on sketch-based interfaces and modeling (SBIM)*, 107–120.

LU, J., BARNES, C., WAN, C., ASENTE, P., MĚCH, R., AND FINKELSTEIN, A. 2014. Decobrush: Drawing structured decorative patterns by example. *ACM Trans. Graph. (SIGGRAPH) 33*, 4, 90:1–9.

MILLIEZ, A., NORIS, G., BARAN, I., COROS, S., CANI, M.-P., NITTI, M., MARRA, A., GROSS, M., AND SUMNER, R. W. 2014. Hierarchical motion brushes for animation instancing. In *Proc. Workshop on Non-Photorealistic Animation and Rendering (NPAR)*, ACM, 71–79.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND VAN GOOL, L. 2006. Procedural modeling of buildings. *ACM Trans. Graph. (SIGGRAPH) 25*, 3, 614–623.

MĚCH, R., AND PRUSINKIEWICZ, P. 1996. Visual models of plants interacting with their environment. In *SIGGRAPH Comput. Graph.*, ACM, 397–410.

OLSEN, L., SAMAVATI, F. F., SOUSA, M. C., AND JORGE, J. A. 2009. Sketch-based modeling: A survey. *Computers & Graphics 33*, 1, 85–103.

ÖZTIRELI, A. C., AND GROSS, M. 2012. Analysis and synthesis of point distributions based on pair correlation. *ACM Trans. Graph. (SIGGRAPH Asia) 31*, 6, 170:1–10.

PARISH, Y. I. H., AND MÜLLER, P. 2001. Procedural modeling of cities. In *SIGGRAPH Comput. Graph.*, ACM, 301–308.

READ, A. L. 1999. Linear interpolation of histograms. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 425*, 1, 357–360.

SIBBING, D., PAVIC, D., AND KOBBELT, L. 2010. Image synthesis for branching structures. *Computer Graphics Forum (Pacific Graphics) 29*, 7, 2135–2144.

SMELIK, R., TUTENEL, T., DE KRAKER, K., AND BIDARRA, R. 2011. A declarative approach to procedural modeling of virtual worlds. *Computers & Graphics 35*, 2, 352 – 363.

SMELIK, R. M., TUTENEL, T., BIDARRA, R., AND BENES, B. 2014. A survey on procedural modelling for virtual worlds. *Computer Graphics Forum 33*, 6, 31–50.

SMITH, A. R. 1984. Plants, fractals, and formal languages. *SIGGRAPH Comput. Graph. 18*, 3 (Jan.), 1–10.

ŠT'AVA, O., BENES, B., MĚCH, R., ALIAGA, D. G., AND KRIŠTOF, P. 2010. Inverse procedural modeling by automatic generation of L-systems. *Computer Graphics Forum (Eurographics) 29*, 2, 665–674.

ŠT'AVA, O., PIRK, S., KRATT, J., CHEN, B., MĚCH, R., DEUSSEN, O., AND BENES, B. 2014. Inverse procedural modelling of trees. *Computer Graphics Forum 33*, 6, 118–131.

SUN, Q., ZHANG, L., ZHANG, M., YING, X., XIN, S.-Q., XIA, J., AND HE, Y. 2013. Texture Brush: An interactive surface texturing interface. In *Proc. Symp. on Interactive 3D Graphics and Games (I3D)*, ACM, 153–160.

TALTON, J. O., LOU, Y., LESSER, S., DUKE, J., MĚCH, R., AND KOLTUN, V. 2011. Metropolis procedural modeling. *ACM Trans. Graph. 30*, 2, 11:1–14.

TALTON, J., YANG, L., KUMAR, R., LIM, M., GOODMAN, N., AND MĚCH, R. 2012. Learning design patterns with Bayesian grammar induction. In *Proc. ACM Symp. on User Interface Software and Technology (UIST)*, ACM, 63–74.

TASSE, F. P., EMILIEN, A., CANI, M.-P., HAHMANN, S., AND BERNHARDT, A. 2014. First person sketch-based terrain editing. In *Proc. Graphics Interface*, 217–224.

VANEGAS, C. A., GARCIA-DORADO, I., ALIAGA, D. G., BENES, B., AND WADDELL, P. 2012. Inverse design of urban procedural models. *ACM Trans. Graph. (SIGGRAPH Asia) 31*, 6, 168:1–11.

WITHER, J., BOUDON, F., CANI, M.-P., AND GODIN, C. 2009. Structure from silhouettes: a new paradigm for fast sketch-based design of trees. *Computer Graphics Forum (Eurographics) 28*, 2, 541–550.

XING, J., CHEN, H.-T., AND WEI, L.-Y. 2014. Autocomplete painting repetitions. *ACM Trans. Graph. 33*, 6, 172:1–11.

YEH, Y.-T., YANG, L., WATSON, M., GOODMAN, N. D., AND HANRAHAN, P. 2012. Synthesizing open worlds with constraints using locally annealed reversible jump MCMC. *ACM Trans. Graph. (SIGGRAPH) 31*, 4, 56:1–11.

ZHOU, H., SUN, J., TURK, G., AND REHG, J. 2007. Terrain synthesis from digital elevation models. *IEEE Trans. Visualization and Computer Graphics 13*, 4, 834–848.