Environmental Objects for Authoring Procedural Scenes

François Grosbellet, Adrien Peytavie, Eric Guérin, Eric Galin, Stéphane Mérillou, Bedrich Benes

Volume 0 (1981), Number 0 pp. 1–13 COMPUTER GRAPHICS *forum*

Environmental Objects for Authoring Procedural Scenes

Francois Grosbellet[1,2], Adrien Peytavie[1], Eric Guérin[1], Eric Galin[1], Stephane Mérillou[2], and Bedrich Benes[3]

[1]Université de Lyon, LIRIS, CNRS UMR 5205, France [2]Université de Limoges, XLIM, CNRS UMR 7252, France [3]Purdue University, USA

Abstract

We propose a novel approach for authoring large scenes with automatic enhancement of objects to create geometric decoration details such as snow cover, icicles, fallen leaves, grass tufts, or even trash. We introduce environmental objects that extend an input object geometry with a set of procedural effects that defines how the object reacts to the environment, and by a set of scalar fields that defines the influence of the object over of the environment. The user controls the scene by modifying environmental variables, such as temperature or humidity fields. The scene definition is hierarchical: objects can be grouped and their behaviors can be set at each level of the hierarchy. Our per object definition allows us to optimize and accelerate the effects computation, which also enables us to generate large scenes with many geometric details at a very high level of detail. In our implementation, a complex urban scene of ten-thousand square meters, represented with details of less than one centimeter, can be locally modified and entirely re-generated in a few seconds.

1. Introduction

Modeling large detailed scenes is an important and challenging problem in computer graphics because it involves the management of an enormous amount of data and complex interactions between objects. Visual details, changes in appearance, as well as aging and weathering, play a central part in the overall realism of a synthetic scene but they are often treated by using textures. Moreover textures are obviously limited in terms of visual impact, because they require tedious work when modifications are needed.

A vast variety of techniques have been proposed for modeling virtual scenes [STBB14] with different kinds of objects, such as trees [SPK*14] or buildings [MWHG06]. The synthetic appearance of generated objects is improved by using aging and weathering algorithms [DRS08, MG08]. Most existing techniques either change the geometry of the object, such as stone erosion [DEJ*99], or only change the texture and surface properties, such as patina or stains produced by flows on surfaces [DPH96, DH96]. Procedural modeling has been addressed so far only in a limited way, usually by considering individual natural phenomena separately and with a global simulation approach which limits the size of the generated scenes.

In this paper, we focus on the generation of small geometric details such as fallen leaves, snow piles, icicles, grass and trash that cover objects. These geometric details play an important part in the overall appearance of a complex scene.

Figure 1: A procedurally decorated city scene. Details such as grass colonizing the sidewalks, fallen leaves, and trash on the ground were automatically generated by extending the geometric objects with environmental properties.

Traditionally, the changes of scene appearance are produced either by carefully adding details to each object in a static scene that has been previously edited or generated, or by performing a global simulation. The composition of a complex scene thus requires either global and computationally demanding processing of the entire scene, or tedious manual editing of the affected areas. These approaches are not compatible with an industrial pipeline context where artists and designers need interactive feedback with a high and intuitive control for authoring effects.

We introduce a novel approach for authoring large scale scenes with automatic procedural effects enhancement at a high level of detail. Inspiration for our work comes from the observation that objects usually influence each other and the environment within a limited range. Moving, deleting, adding objects, or changing their characteristics and parameters does not change the entire scene but only impacts a restricted number of nearby objects.

Instead of applying a global simulation to the entire scene, we introduce *environmental objects.* Our environmental objects are a simple extension of geometry, they are easy to define and control, and allow for an intuitive scene definition. The environmental objects' appearance automatically adapts to its environment. Each environmental object is characterized by a set of *scalar fields* and a set of procedural *effects*. The scalar fields define the impact of the object over the environment and procedural effects describe how the object reacts to the environmental changes by adding geometry to the scene. The final scene is represented by a hierarchical composition of environmental objects and scalar fields.

An example in Fig. 1 shows how procedurally generated leaves are placed automatically on the sidewalk. The leaf field attached to the tree defines the regions where leaves fall to the ground, whereas the pavement reacts to those fields and defines the way leaves are distributed and accumulated into piles. The objects with their fields can be either placed manually during an artistic session or procedurally.

Our main contribution is in developing a novel method for integrating procedural methods. We also develop several fast novel procedural algorithms for snow, icicles, leaves and grass that are intended only for the integration with the framework. More precisely the main contributions of our approach are as follows:

Details Our framework orchestrates heterogeneous object-specific algorithms for generating many different types of details, which change the overall appearance of the scene in a uniform and consistent fashion. Using a per-object definition of effects with instancing mechanisms allows us to generate small details for different types of objects while maintaining the ability to process large scenes. Our procedural per-object approach allows the maximum scale factor between the smallest details and the scene size to reach more than 10^4.

Control Our approach allows for several levels of control. On the lowest level, each individual object and its behavior is defined and controlled. On the second level, the objects can be grouped into hierarchies with defined group fields and instanced. Finally, global scalar fields modify the parameters of the environment and consequently the entire scene behavior. It is important to note that we do not aim for a physically correct simulation: our effects implementations are approximations of physical process that produces plausible and believable results. Instead, we focus on control, interactivity, predictability, and size of the authored scene.

Scalability Each type of object defines how it reacts to the environment in a specific manner, which improves the performance of otherwise time-consuming global simulations. Per-object definition takes into account the objects' specific properties, which enables us not only to optimize the generation of snow, fallen leaves, trash, and grass for different kinds of objects, but also to produce complex effects and features that are difficult to capture with simulations. Different effects can also be selected according to the distance from the camera, therefore our method allows for a level-of-detail generation and partial scene updates.

Our method naturally extends existing scene authoring and procedural generation techniques, and brings the object-specific algorithms for simulation of natural phenomena into a single fast, intuitive, controllable, and unified framework. Note that although environmental objects react to the environment, interaction between different effects is not supported, e.g., our framework does not account for the deposit of snow on leaves.

2. Related work

We present an overview of the most relevant papers dealing with procedural modeling of natural phenomena that add details to a given input scene or model. In particular, we review simulation and generation of snow, leaves, and ice. For a comprehensive and complete overview of erosion, aging, weathering, and material appearance we refer readers to the review [MG08], to the book [DRS08], and for a recent review of procedural methods for virtual worlds to the paper [STBB14].

Snow generation models can be divided into three categories: manual methods, particle simulations, and surface displacement techniques. An early manual model proposed in [NIDN97] uses meta-balls with user-specified snow distribution. Muraoka *et al.* [MC00] used microscopic physical properties of snow and water to model snowfall, snow cover shape, and melting. Fearing *et al.* used particles to represent larger sets of snowflakes, and a surface stability test is used to create realistic snow cover [Fea00]. Another bulk particle simulation method [PTS99] relies on a surface displacement and texturing approach to create the snow coverage of large-scale scenes. The dynamic features of snow, such as wind-driven snow transportation and snowflake animation were addressed in [FO02] and [LZK*04]. The method proposed in [FB07] generates similar scenes by using ambient occlusion. Recently, a voxel-based large-scale winter scenery synthesizer based on a complete thermal simulation was introduced in [MGG*10]. The two models presented in [vFG09] and [vFG11] rely on the construction of height span maps to create snow distribution and produce complex results such as snow bridges.

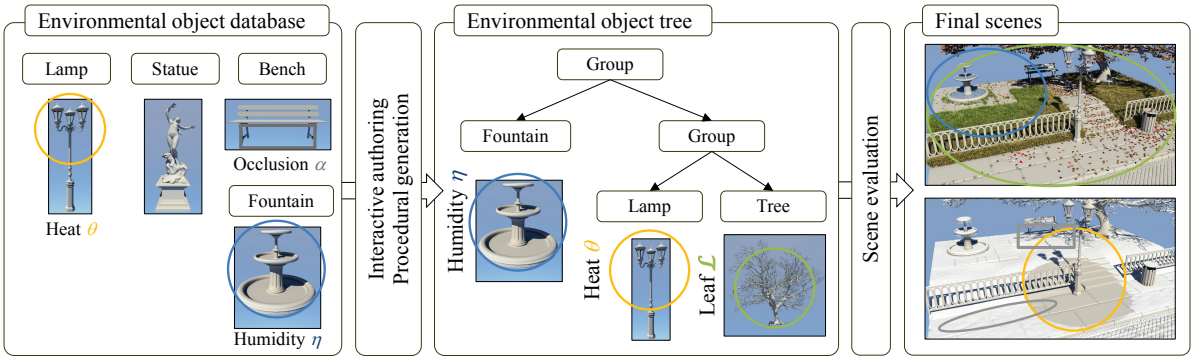

Figure 2: Overview of our method: given a set of environmental objects in a database, we create a rich construction tree that hierarchically combines geometric objects with scalar fields and effects (snow, icicles, leaves, grass, and trash) so that the objects can affect the environment and react to it. The environmental object tree can be either authored by the user or generated procedurally. During the scene creation step, new scalar fields can be added to the scene graph to allow authoring and special effects. The objects automatically decorate themselves with details according to the prescribed parameters of the fields.

Leaf modeling was addressed by [MMPP03] who presented an automatic technique for creating lobed leaves from images and by Peyrat *et al.* [PTMG08] who extended this approach for an aging leaves simulation. Jeong *et al.* [JPK13] introduced a method to produce realistic autumn leaves. Physically-based techniques have been used for animating leaves in the wind [WH91, WZF*03], the leaf distribution results from complex dynamics by combining leaves falling and floating in the wind, tumbling, rolling, and eventually colliding and stacking to the ground. Desbenoit *et al.* presented a method to model the distribution of thousands of leaves on the ground in [DGAG06]. The trajectories were approximated by template-based movements and large piles were created by using a collision detection and stabilization technique.

Simulating *ice formation* is a challenging task, due to the wide range of involved scales. An approach by [KG93] uses a random-walk model of icicle growth, where water droplets move along the ice surface and freeze with a certain probability. Kim *et al.* presented ice formation algorithms for small-scale ice growth on objects [KL03, KHL04] and a physically-based algorithm for simulating icicle growth [KAL06] by approximating water solidification as a thin-film Stefan problem. Gagnon *et al.* [GP11] proposed a procedural approach for modeling icicles based on a water flowing and droplet dripping simulation.

Most of the previous techniques deal with a specific natural phenomena simulation either on a global or a local scale. Simulation approaches are computationally expensive. Specific methods that can capture the small geometric details do not scale for modeling of large scenes with many details such as hundreds of thousands of leaves, lichens and grass tufts, or highly detailed snow cover. They can still be used in our framework as a pre-processing step for obtaining a particular effect for a given kind of environmental object.

Our work presents a unified approach that defines interactions of various objects and the environment through object-specific behaviors description. While objects react to the environment, our system does not support interaction between effects. Similar to our approach is the method for wind simulation of [WH91] that composes the scene from simple interacting elements. Various approaches for multi-procedural modeling exist [KPK10]. Close to our method is the guided procedural modeling [BSMM11] that generalizes the concept of environment and allows for parallel execution and communication of multiple procedural models. However, it allows only a limited inter-influence between elements and is specific to L-systems. In contrast, our approach encapsulates a virtually arbitrary procedural model. Moreover, our technique addresses the scalability problem by creating environmental procedural objects whose geometry and details in appearance adapt to their environment.

3. Overview and notations

In this section, we present the environmental object tree that defines the enhanced scene graph managing objects influencing and reacting to the environment.

3.1. Scene

Our framework defines the scene as a construction tree as shown in Fig. 2. Leaves of the tree store environmental objects whereas inner nodes are grouping operators that combine them. An environmental object $\mathcal{O}$ (See Fig. 4) consists of a base geometry $\mathcal{B}$, associated effects whose geometry will be denoted as $\mathcal{A}(e)$, and three-dimensional scalar fields $\mathcal{F}$:

$$\mathcal{O}(e) = (\mathcal{B}, \mathcal{A}(e), \mathcal{F})$$

In our implementation, the particular effects $\mathcal{A}(e)$ are snow, fallen leaves, grass tuft, trash, and icicles models, which will be referred to as $\mathcal{S}(e)$, $\mathcal{L}(e)$, $\mathcal{G}(e)$, $\mathcal{T}(e)$, and $\mathcal{I}(e)$ respectively. Scalar fields $\mathcal{F}$ control different effects; for example the snow thickness is related to the temperature and the occlusion fields.

The scene is computed by hierarchically traversing the scene tree and combining the base geometry with all the produced effects. We denote e the *environment* that corresponds to the evaluation of the fields on the scene root node. The base geometry does not depend on fields and thus does not have to query the environment. On the contrary, the effects computing process is allowed to query on the fly the environment e in order to quantify the considered effect (snow, ice, leaves, grass) and to adapt it according to temperature, humidity, and occlusion fields. The scene can be controlled *per-object*, by defining the object behavior, *per-group*, by defining the behavior of a sub-tree, or *globally*, by inserting global scalar fields embedding the whole scene. The object hierarchies, and the option of replacement of complex effects by simpler and less computationally expensive ones, provide a means for speeding up the computations if necessary (Section 6).

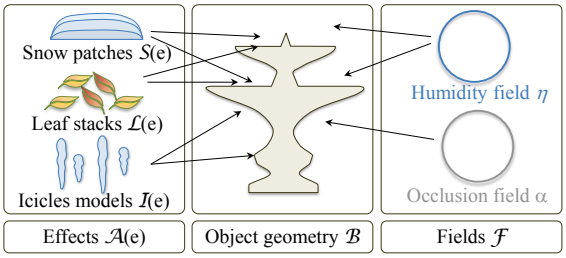

Figure 4: An environmental object $\mathcal{O}(e)$ is defined by its base geometry $\mathcal{B}$ (center), procedural effects $\mathcal{A}(e)$ (left) and scalar fields $\mathcal{F}$ (right).

Grouping nodes are internal nodes that combine several sub-trees into the hierarchy. The base geometry, as well as effects geometries, are combined by a union operation. Scalar fields are mostly combined additively except for the occlusion field which is computed by multiplying each sub-tree contribution. Grouping nodes are used to structure the scene tree and allow a bounding volume hierarchy to be constructed. Different kinds of internal nodes with a modified behavior can be used to perform level of details as explained in Section 6.

3.2. Workflow

The overall workflow consists of two main steps: environmental object database creation and scene construction.

Environmental object database creation is performed by extending an input geometric object $\mathcal{B}$ with procedural *effects* $\mathcal{A}(e)$ that define the behavior of the object i.e., its change of appearance, and *fields* $\mathcal{F}$ which specify the impact of the object on the environment. Fields are inspired by [KL05] and they define scalar fields that are combined together to modify the characteristics of the environment around the objects. The effects implement the parameterized details that are added to objects, such as leaves, grass, snow, icicles, or trash. The particular effects can be defined either manually by attaching them to the objects, or generated automatically by augmenting an existing generation process. For example environmental buildings can be generated with the CGA Shape Grammars [MWHG06] by adding the definition of environment scalar fields such as temperature emitters and parameterized effects in the grammar. Once defined, these objects can either be immediately used in the second step or stored for later reuse.

Scene creation can be performed either by interactive editing or procedural generation. Authoring is achieved by selecting the environmental objects from the database, and arranging them in a scene graph. Environmental objects can be added, removed, and modified, and the scene can be enriched at any time. Moreover, objects can be grouped to create hierarchies and a group may have a behavior that overrides or combines behavior of the objects in the group. During the scene authoring, objects and groups can be instanced for further reuse. Global scalar fields can be added in order to produce the desired effects. This step can be done once the scene is completely defined but also during the authoring process to have a preview of the effects.

One example of the scene creation and control is depicted in Fig. 3. Temperature and occlusion fields allow for a high level of control over the final scene by locally decreasing the snow thickness. Occlusion fields can be used to create details or add features, such as footprints and tire tracks.

4. Environment evaluation

Environmental objects are enhanced with scalar fields that have an important impact over the environment. In this section, we review the definition and the evaluation of the parameters of the environment. Formally, the environment is defined as a function $e : \mathbb{R}^3 \rightarrow \mathbb{R}^n$ that computes a set of scalar values $e(\mathbf{p})$ at every point in space.

$$e(\mathbf{p}) = \Big(s(\mathbf{p}), i(\mathbf{p}), l(\mathbf{p}), g(\mathbf{p}), t(\mathbf{p}), \alpha(\mathbf{p}), \eta(\mathbf{p}), \theta(\mathbf{p})\Big)$$

In our system, the environment includes the amount of fallen snow $s(\mathbf{p})$, the density of icicles $i(\mathbf{p})$, leaves $l(\mathbf{p})$, grass $g(\mathbf{p})$, and trash $t(\mathbf{p})$. It also includes the *occlusion* $\alpha(\mathbf{p})$, the *humidity* $\eta(\mathbf{p})$, and the *temperature* $\theta(\mathbf{p})$ which affect the resulting shape and distribution of snow cover, fallen leaves, grass, trash, and icicles.

The environment is computed by recursively traversing the construction tree and combining the scalar fields of the environmental objects. Therefore, e can be defined as the

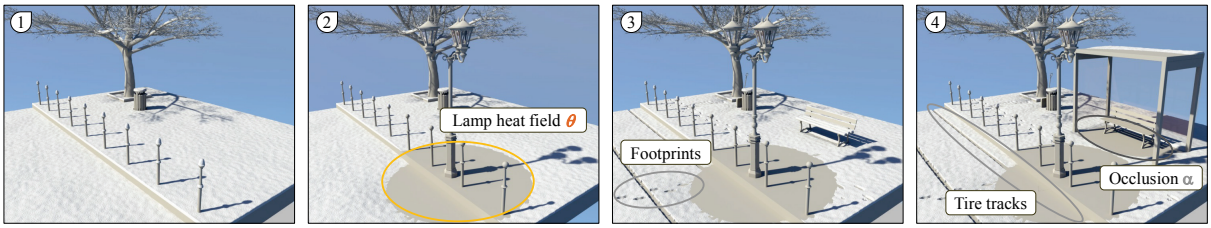

Figure 3: A four step creation of a winter scene. The user defined the initial scene by assembling pavement and road pieces and adding a tree, and a trash can ①. The scene was embedded in a *global* snow field to get a winter scenery. In the next steps the user added a lamppost ②, a bench ③ and a bus shelter ④ that prevents the accumulation of snow on the bench. Tracks and footprints in the snow were produced by a set of user controlled *local* environment fields.

evaluation of the fields $\mathcal{F}$ of the root node of the construction tree. Depending on the queried point $\mathbf{p}$, branches of the tree may be pruned during the traversal due to to the bounding volume hierarchy.

Modeling with fields is the heart of our system because it reflects the degree of approximation of the real physics. While it would be possible in our general framework to compute temperature and occlusion using heat transfer simulations and calculations based on the geometry of the objects in the entire scene, this evaluation would be costly. In our implementation, we use a less precise, but faster approach by using skeleton-based implicit primitives. Although this approach is not physically correct, it provides the user with an efficient and intuitive tool that allows precise control for authoring complex scenes (Fig. 5) and allows the creation of scenes that automatically and interactively adapt to the parameters of the environment.

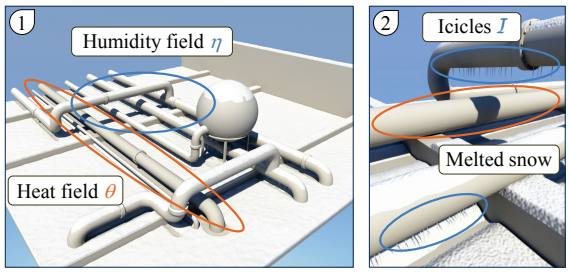

Figure 5: Example of a combination of hot and cold temperature fields emanating from pipes and producing complex snow and icicles distribution by using per-object snow and icicles effects.

Our approach is inspired by the BlobTree model [WGG99]. We rely on skeletal primitives organized into the hierarchical environmental object construction tree to define the parameters of the environment by combining the influence of the different objects in the scene. The environmental objects located at the leaves of the tree define compactly supported scalar fields, which allow for local field evaluations and for an efficient evaluation of bounding volume hierarchies.

Scalar fields such as snow, icicles, leaves, grass, trash, temperature, and humidity are combined by using standard blending as described in [WGG99]. For instance the temperature $\theta(\mathbf{p})$ is defined as:

$$\theta(\mathbf{p}) = \sum_{i=1}^{n} \theta_i(\mathbf{p}).$$

The occlusion scalar field is computed differently. Recall that the $\alpha(\mathbf{p})$ is defined as a function mapping onto unit interval. Combination is obtained by multiplying the influence of occlusion fields α_i:

$$\alpha(\mathbf{p}) = \prod_{i=1}^{n} \alpha_i(\mathbf{p}).$$

Multiplying values, instead of using a sum as in traditional implicit surface models, allows us to combine the relative influence of different occlusion fields while preserving mapping onto unit interval.

5. Environmental Objects

Environmental objects are geometric objects enhanced with two aspects: scalar fields that define the impact of the object on the environment, and effects defining the way the objects react to the environment.

Below we review the definition of different effects produced by environmental objects. We detail our techniques for snow, icicles, leaves, and grass. It is important to note that the choice of our methods is in respect to speed and controllability. In theory, any effect implementation that allows control with scalar fields could be used.

5.1. Snow

Although any method for snow cover generation from Section 2 could be used in our framework, we have developed a novel and fast procedural snow generation technique that

adapts the resolution of the snow mesh to the supporting object, which enables us to process objects at a high level of detail.

Our approach for generating snow layers consists of covering the input geometric object $\mathcal{B}$ with *snow effects*. A snow effect, denoted by $\mathcal{S}(e)$, is a parameterized textured mesh whose geometry and discretization adapt to the scale and geometry of its supporting object $\mathcal{B}$. The mesh vertices $\mathbf{p}$ are displaced from the object's surface by an offset value that varies according to the amount of fallen snow $s(\mathbf{p})$ and the geometry of the object. This offset is a function of the parameters of the environment: occlusion $\alpha(\mathbf{p})$ and temperature $\theta(\mathbf{p})$.

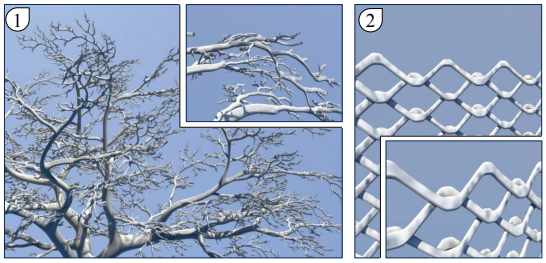

Figure 6: By targeting the snow cover for specific objects, we can generate snow with a high level of detail and capture small geometric features.

Snow mesh construction We define two different categories of *snow effects*: geometry-controlled and procedural. *Geometry-controlled snow effects* are automatically derived from the geometry $\mathcal{B}$ and are used for the generation of snow covering objects with complex geometries, such as trees (Fig. 6.1) or fences (Fig. 6.2). The vertices of the border of the snow effects automatically snap to the vertices of the underlying object, preventing cracks that could appear between the snow patch and the object geometry. Moreover, snow effects can be progressively refined to adapt the snow mesh to the desired precision. This enables us to process large scenes efficiently or use it for level of detail generation.

In contrast, *procedural snow effects* are black box functions that generate meshes according to a set of input parameters that define their shape and subdivision level. *Procedural snow effects* can be optimized for simple geometric objects, such as boxes, cylinder, spheres, or surfaces of revolution, that are common in architecture design. Such effects dynamically generate an adaptively refined snow mesh on the fly.

Snow elevation computation Let $\mathbf{p}_k$ denote the vertices of the mesh of the snow surface. The vertices have a snow direction vector $\mathbf{v}_k$ associated to them. In general, the vector $\mathbf{v}_k$ points in the vertical direction, but the direction vectors at the border of the snow mesh may be edited to allow the creation of overhangs.

A snow effect defines a displacement function $\delta(\mathbf{p}_k)$ that characterizes the displacement of the mesh vertices according to the amount of fallen snow $s(\mathbf{p}_k)$. The positions of the vertices of the displaced mesh are calculated as:

$$\mathbf{p}_k(e) = \mathbf{p}_k + \delta(\mathbf{p}_k)\,\mathbf{v}_k.$$

The elevation function is $\delta(\mathbf{p}) = s(\mathbf{p})\,g(d(\mathbf{p}))$ where $d(\mathbf{p})$ denotes a distance function between the point and the border of the mesh and g refers to the snow elevation function:

$$g(d(\mathbf{p})) = \begin{cases} \left(1-(1-d(\mathbf{p})/d_0)^2\right)^4 & d(\mathbf{p}) < d_0 \\ 1 & \text{otherwise,} \end{cases}$$

where d_0 controls the snow slope near the object's border.

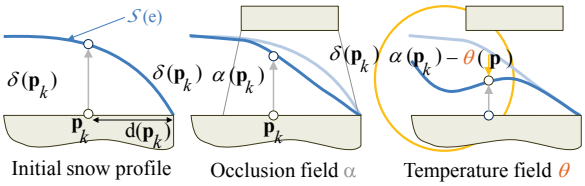

Figure 7: The influence of the environment is captured by the occlusion (center) and temperature fields (right) that are applied to the initial displacement $\delta(\mathbf{p}_k)$.

Fig. 7 illustrates how the lower object snow mesh is influenced by an occlusion field and a temperature field. The final position of the snow vertex accounts for the snow thickness $s(\mathbf{p}_k)$, occlusion field value $\alpha(\mathbf{p}_k)$, and temperature field value $\theta(\mathbf{p}_k)$ obtained by querying the environment:

$$\mathbf{p}'_k(e) = \mathbf{p}_k + (\alpha(\mathbf{p}_k)\,\delta(\mathbf{p}_k) - \theta(\mathbf{p}_k))\mathbf{v}_k.$$

The term $(\alpha(\mathbf{p}_k)\,\delta(\mathbf{p}_k) - \theta(\mathbf{p}_k))$ is clamped to zero to avoid negative snow displacements. The occlusion field $0 \leq \alpha \leq 1$ acts as a multiplicative factor, whereas the temperature field θ diminishes the snow height in an absolute way.

5.2. Icicles

Icicles are point-based skeletal implicit surfaces whose size and shape are computed according to the parameters of the environment. Given an input object $\mathcal{B}$, we create a collection of m candidate icicles $\mathcal{I} = \{\mathcal{I}_j,\, j \in [0, m-1]\}$ that is attached to the object. Each icicle $\mathcal{I}_j = (\mathbf{p}_j, \mathcal{S}_j(\theta,\eta))$ has an anchor position $\mathbf{p}_j$ and a parameterized shape $\mathcal{S}_j(\theta,\eta)$ where θ is the temperature field and η is the humidity field. The icicle positions can be set manually along object edges, or computed automatically by using a water flow and droplet dripping simulation as described in [GP11].

We use a particle-based representation generated by BlobTrees [WGG99]. A list of sphere primitives is generated

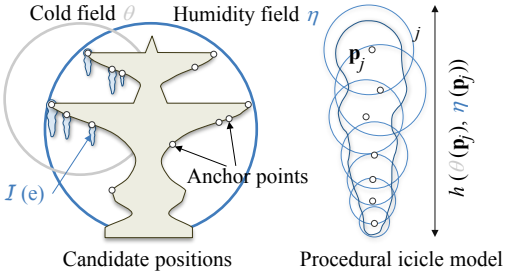

Figure 8: Icicle anchor positions $\mathbf{p}_j$ are attached to the objects (left). The dimension of the icicles depend on the value of $\eta(\mathbf{p}_j)$ and $\theta(\mathbf{p}_j)$ (right).

along the vertical axis of the icicle (Fig. 8) from the upper part of the icicle downward. The sphere radius is continuously decreasing as well as the spacing between them. A horizontal perturbation is also applied to introduce randomness.

The presence of an icicle is determined by evaluating the ice field $i(\mathbf{p}_j)$ and temperature field $\theta(\mathbf{p}_j)$ at its anchor position $\mathbf{p}_j$. If the ice value $i(\mathbf{p}_j)$ is positive and the temperature value $\theta(\mathbf{p}_j)$ is below freezing point then an icicle will be generated:

$$\mathcal{I}(e) = \{\mathcal{I}_j | \theta(\mathbf{p}_j) < 0 \wedge i(\mathbf{p}_j) > 0\}.$$

The height $h(\theta(\mathbf{p}), \eta(\mathbf{p}))$ of the icicles varies according to the humidity and temperature fields. To speed up the icicle geometry calculation we generate a reduced set of icicles models during an offline pre-processing step, and we store them and their properties. Icicles are then instantiated from the precomputed model whose parameters are the closest (Fig. 9).

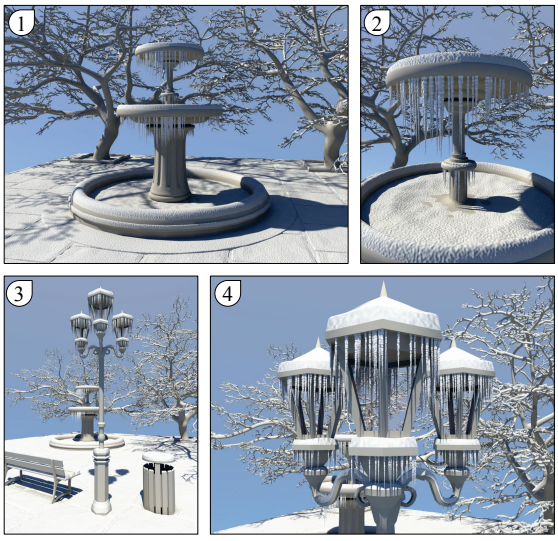

Figure 9: Example of procedural icicles formed on a fountain (①, ②) and on a lamp post (③, ④).

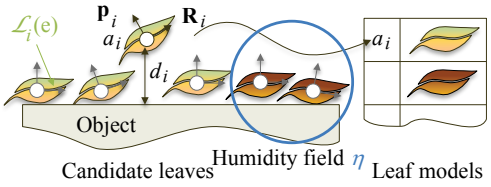

Figure 10: A leaf effect uses predefined leaf models that are aperiodically tiled. The texture and deformation are parametrized by the humidity field η and leaf field l.

5.3. Leaves, petals, and trash

Leaves can be distributed by using a simulation as presented in [DGAG06] and by recording the final positions and orientation of the leaves as well as their number in the obtained leaf pile structure. This approach becomes more computationally demanding as the size of the supporting object $\mathcal{O}$, such as terrain patches, increases.

We introduce a two-step procedural method for generating fallen leaves. First, we construct a collection of candidate leaves organized and stored into a leaf pile structure. Leaf effects are attached to their supporting objects $\mathcal{B}$ and define a set of virtual leaves instances for the corresponding object. The final distribution of leaves $\mathcal{L}(e)$ for a given object is generated by selecting and instancing some of its candidates leaves according to the field value of the leaf field $l(\mathbf{p})$ and the environment as described below.

We define *a leaf effect* as a set of n leaf models $\{\mathcal{L}_i, i \in [0, n-1]\}$. Every leaf model $\mathcal{L}_i = \{\mathbf{p}_i, \mathbf{R}_i, d_i, a_i\}$ is defined by its position $\mathbf{p}_i$, its rotation matrix $\mathbf{R}_i$, its distance d_i to the underlying object geometry, and its index a_i referring to a textured geometric leaf model (Fig. 10) whose characteristics depend on the leaf type and humidity field η.

The construction of the candidate leaves distribution is performed during a pre-processing step that defines the positions $\mathbf{p}_i$, the orientations $\mathbf{R}_i$ and the distances d_i. We use two techniques to generate candidate leaves information: a procedural 3D tiling approach for fast and automatic generation of large leaf piles, and manual editing for fine tuning of the leaf distribution.

Our 3D aperiodic tiling approach is inspired by the rock pile generation algorithm of [PGGM09]. First, a set of cubic tiles that contain layers of leaves that aperiodically tile the space is generated. The cells are created by incrementally filling tiles with different layers of leaves. Every layer is then created by using Poisson-disk distribution of points with a radius proportional to the size of leaf models to avoid intersections. This approach is used to distribute leaves on

large surfaces and to model leaves piles, as it can be seen in Fig. 20 and 21.

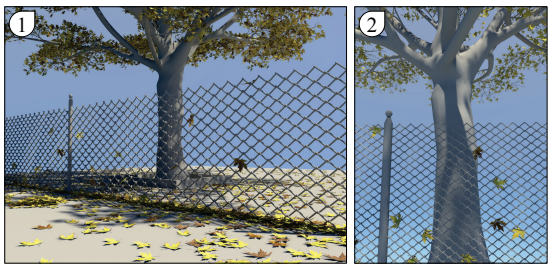

Figure 11: Procedural leaf effects generated the leaves on the ground ① whereas some specific positions were edited to allow the instantiation of leaves pinned on the fence ②.

Manual editing allows the user to store candidates leaves information corresponding to a specific leaf distribution. This approach allows for the tuning of leaves effects, which is a mandatory step for the fine tuning of the final leaves distribution. It made specific distribution patterns possible, including distributions extremely difficult to achieve with physical simulations, such as leaves pinned on the fence in Fig. 11.

During the scene effects computation, a candidate leaf $\mathcal{L}_i$ is instantiated if the field value $l(\mathbf{p}_i)$ is greater than the distance d_i from the object (Fig. 10). The set of instantiated leaves is defined as:

$$\mathcal{L}(e) = \{\mathcal{L}_i \,|\, \alpha(\mathbf{p}_i)\, l(\mathbf{p}_i) > d_i\}.$$

The occlusion field is taken into account as a multiplicative factor, so that neighboring objects may prevent the accumulation of leaves. In practice, only the upper layer of leaf piles needs to be generated as the other leaves will not be visible, which can be easily obtained by checking $d_i > \alpha(\mathbf{p}_i)\, l(\mathbf{p}_i) - \varepsilon$.

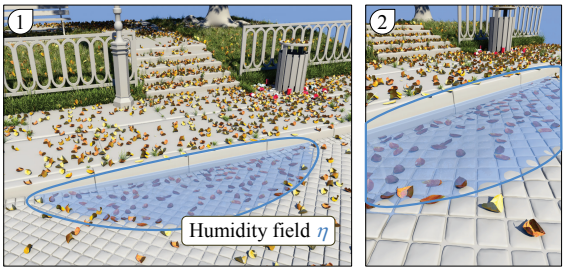

Figure 12: Leaves on a partially wet sidewalk. Leaves outside the water puddle are dry and wrinkled, whereas the leaves inside and near the puddle are wet and flat.

By taking humidity fields η into account at the instantiation of the candidates leaves, we determine the leaf model associated to a given candidate leaf $\mathcal{L}_i$. If $\mathcal{L}_i$ doesn't lie in a humidity field, so if $\eta(\mathbf{p}_i) < 0$, $\mathcal{L}_i$ will be instantiated using a wrinkled version of its geometric model a_i. Otherwise, when $\mathcal{L}_i$ lies in a humidity field, we then instantiate a decayed version of a_i (see Fig. 12).

The tiling method to distributes leaves is easily generalizable to a wide range of other objects, and we use it for petals and trash generation. It also could be used for various kinds of other objects such as twigs, rocks, sea shells, etc.

5.4. Grass

Another procedural effect used in our framework allows for covering objects with grass. Grass could be computed using ecosystem simulation like the one presented in [DHL*98], but this kind of approach is computationally demanding, and is not well suited for the fast generation and easily controllable generation process we are aiming at.

A *grass effect* is a collection of n grass tufts $\mathcal{G} = \{\mathcal{G}_j,\ j \in [0, n-1]\}$. A grass tuft $\mathcal{G}_j = (\mathbf{p}_j, a_j)$ is defined by its anchor position $\mathbf{p}_j$ and its index a_j referring to a textured geometric grass model whose characteristics depend on the grass field g, the humidity and temperature field η and θ respectively. Grass tuft anchors on their supporting objects are computed using a Poisson-disk sampling with the radius of the grass tuft (Fig. 13).

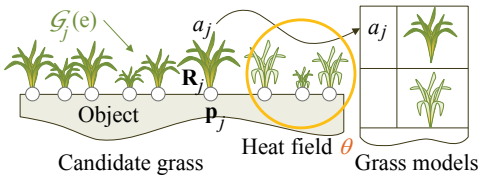

Figure 13: Grass tufts distribution on the surface of an object, tiled with a Poisson-disk sampling. The texture of the grass tufts are parametrized by the humidity field η and temperature field θ.

Occlusion fields are taken into account as a multiplicative factor during the computation of grass tufts, allowing us to create features such as trampled trails in parks. Temperature fields are taken into account as shape modification: grass tufts under the influence of heat will be represented with a dried model.

The grass effect is shown in Fig. 14. A simple scene is enhanced by procedurally generated grass. The dry grass growing on the sidewalk in Fig. 14.1 is created by a combination of grass fields and temperature fields. The appearance of the half-dried grass that can be seen in Fig. 14.2 is achieved by combining grass fields with temperature fields and occlusion fields.

6. Scene evaluation

Recall that a scene is defined as a construction tree combining environmental objects with effects and scalar fields

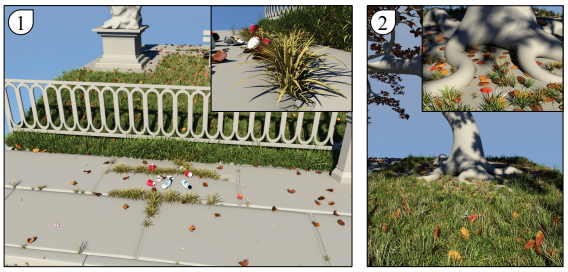

Figure 14: Procedural grass grows on soil, but also in the cracks on and between tiles ①. The occlusion field created by the tree decrease the density of grass between its roots ②.

(Section 3). Environmental objects can be grouped into different kinds of operators to optimize the evaluation of the scene. In this section we present two techniques: *level of detail nodes* that modify and optimize the traversal of the tree during the scene evaluation in order to speed-up its computation, and *effects instancing* which generates a more compact representation of the final scene.

6.1. Level of detail nodes

Level of details (LOD) nodes are used to speed-up the evaluation of the environment. Let $\mathcal{N}$ denote a node and $\mathcal{F}$ its corresponding scalar fields. The children of $\mathcal{N}$ will be denoted as $\mathcal{N}_i$ and their scalar fields as $\mathcal{F}_i$ respectively.

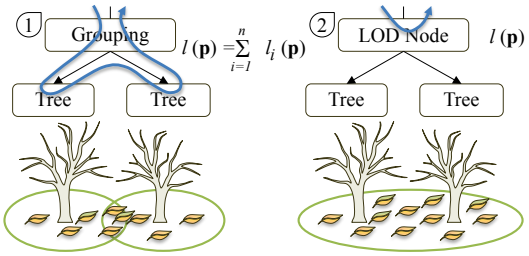

Figure 15: Comparison of the evaluation of the leaf field $l(\mathbf{p})$ between a grouping node ① and level of detail node ② which simplifies the recursive evaluation and speeds up queries.

The recursive evaluation of $\mathcal{F}$ is more computationally demanding as $\mathcal{N}$ has more children, particularly if the set of $\mathcal{F}_i$ are computationally demanding. Therefore, we introduce LOD nodes that are specific nodes whose set of scalar fields are computed locally. Instead of recursively traversing its sub-tree, the node defines its own $\mathcal{F}$ (Fig. 15). When carefully set, the visual difference produced by the original node and the LOD node is negligible, while evaluation speed can increase by an order of magnitude in complex cases.

The higher level hierarchy LOD should be designed such that it approximates the collection of the lower LODs. Although it could be done automatically by approximating the bounding volumes of the fields, in our implementation it is done manually by the designer. An example in Fig. 16.1 shows three trees shedding leaves and Fig. 16.2 shows the same scene, where the trees are grouped in a level of detail node. There is a negligible visual difference between the two images, but the second scene is easier to control and is evaluated three times as fast as the first one.

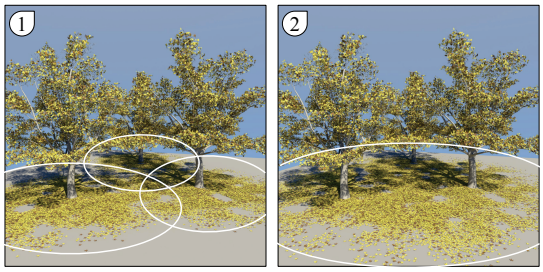

Figure 16: Leaf distribution by using per-object leaf field produced by three trees ① and the per-group ②.

6.2. Effects instancing

Multiple occurrences of a same type of environmental object often exist multiple times in a given scene. During the scene evaluation, effects of this type of object are computed once for each occurrence. To optimize the evaluation, our framework traverses the scene graph and detects environmental objects of the same type that have the same environmental parameters. Effect generation is called only once for all objects sharing the same environmental parameters, and the resulting effects are then instantiated for each of those objects (Fig. 17).

There are two ways the environmental objects with the same parameters can appear. They can be generated procedurally, in which case the system automatically keeps track of each instance, as is the case of the windows in Fig. 18. If the field is a combination of other fields, we regularly sample them and compare if their L^2 distance is under a user-defined threshold value.

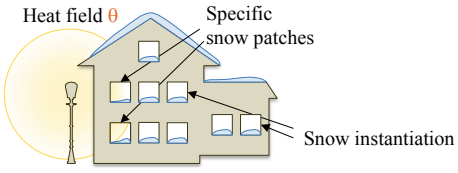

Figure 17: Window instances that have the same environmental parameters call the generation of snow once and the resulting snow effects are duplicated.

An example of instantiated windows and balcony on a

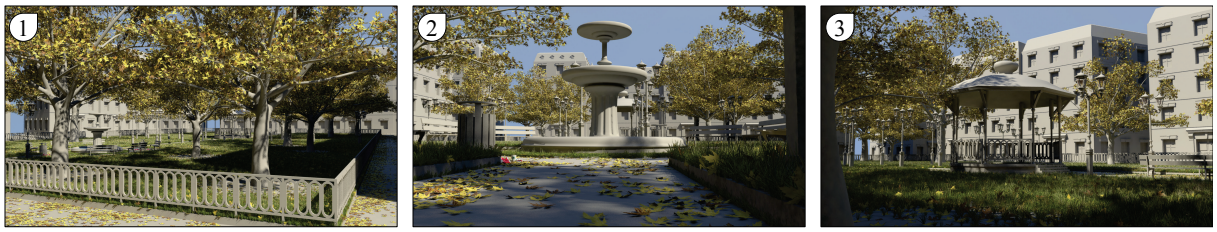

Figure 19: A North American inspired square park. Approximately 980k leaves were generated on the ground, the benches and fountain from the 18 trees.

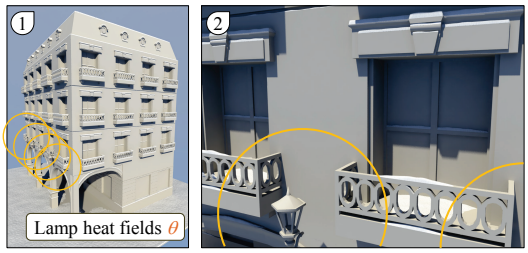

Figure 18: Multiple occurrences of a single object, such as the lamp post and the windows ① and the balconies ② are represented by their instances. Moreover, the snow effects with similar environmental conditions are instantiated.

building produced by enriched a CGA shape grammar is shown in Fig. 18. The complete building features 70 windows and balcony, as well as four lampposts on its front facade. The 70 windows and balcony are instantiated, and their effects only need to be computed six times: once for each of the five lower windows of the front facade that are under the influence of the temperature fields created by the lampposts, and only once for all the others.

7. Results and discussion

Our framework has been implemented in C++ and MentalRay® was used to produce photo-realistic images. Results were calculated on a desktop computer equipped with Intel Core i7, clocked at 3 GHz, with 16GB of RAM, and NVIDIA GTX 670 with 2GB of RAM.

7.1. Procedural generation

The scene hierarchy lends itself for procedurally generated models such as buildings or trees. By replacing the standard geometric models used as terminal leaves in the generation process by our environmental objects, we create *environmental procedural objects* that react and participate in the environment. The CGA shape grammars and L-systems can naturally be enriched with our environment objects in order to produce the buildings and trees shown in Fig. 19. It took us one hour to completely parameterize the city building generation framework, including the time to augment the different assets that were used to produce Fig. 22. The scene consists of $4,110$ groups and over $19,000$ environmental objects. The Paris Montmartre-inspired scene (Fig. 22) and the park (Fig. 19) were created using CGA shape grammars combined with our environmental object framework.

7.2. Authoring and control

We claim a contribution in control of the procedural scenes. This control is simple, efficient, and allows for a fast and intuitive creation and editing of complex scenes. Our models allow the user to control the placement and density of features such as grass between pavement blocks, trash around trash cans, leaf and snow piles, or areas where snow melts or turns into ice. In Fig. 20 *global* leaf fields have been placed to produce a fall scenery. The tree has associated a large spherical leaf field which modifies the appearance of the road and pavement blocks. Those environmental objects automatically generate leaves over their geometry. Leaf piles are controlled in a similar fashion and it is possible to add piles of leaves or accumulations produced by wind. These sequences demonstrate that scene authoring is easy and once objects are aware of the environment, the manipulation of the fields that affect the scene is predictable and intuitive.

7.3. Scalability

Our procedural approach can generate geometric details for large scenes with a high level of detail and allows local editing at interactive rates. In our implementation, our per-object approach allows the maximum scale factor between the smallest details and the size of scene to reach more than 10^4 (Table 1). Because our fields are defined on continuous domains, virtually infinite precision on effects could even be reached.

Our description of the generated geometry has native multi-resolution support. We can visualize the scenes at multiple scales, and we can use view-dependent clipping algorithms or resource-dependent strategies. Examples in Fig. 19 and Fig. 22 show different city scenes composed from objects with details varying from 5 cm (bumps on road and

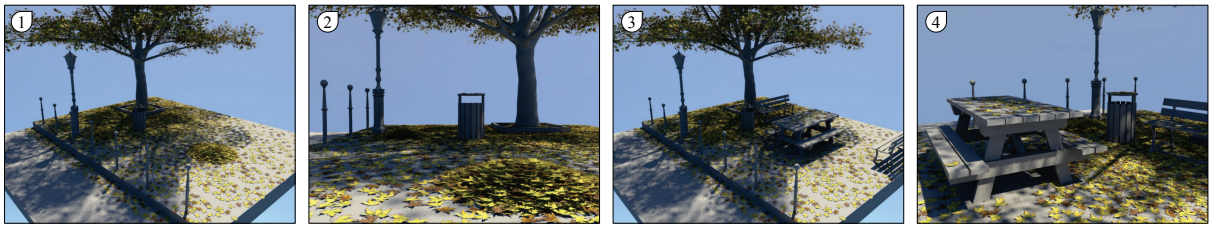

Figure 20: By changing the global environment parameters, the road, the pavement, and the tree generated their own leaves. Images ① and ② demonstrate user control when two small leaf fields were specified to create two leaf piles on the pavement. Images ③ and ④ show a table and bench which cast occlusion fields that limit the number of leaves on the ground.

Scene	Size	Objects		Snow		Leaves		Grass	
		Models	Instances	Time	Triangles	Time	Instances	Time	Instances
Street (Fig. 20)	10×10	13	191	0.13	0.5M	0.12	36k	0.01	1k
Road (Fig. 21)	70×70	5	2563	4.01	20.2M	6.73	1461k	4.76	3020k
Montmartre (Fig. 22)	33×100	85	10519	2.09	78.8M	0.54	221k	0.18	78k
Park (Fig. 19)	115×90	68	19855	1.60	22.0M	4.53	839k	1.86	1050k

Table 1: Statistics for different scenes presented throughout the paper. Each row reports the size of the scene (in meters), the number of environmental objects and their instances, the snow generation time (in seconds), the number of triangles of the snow cover, the leaves and grass generation time (in seconds) and the number of instances for the corresponding effect.

pavement) to 1 cm (carvings at window corners, and details on stairs and the lamp post model).

7.4. Performance

Our parametrized *per-object* effect description provides a scalable representation of large scenes with many details. Table 1 reports detailed statistics of the number of environmental objects in each scene, number of instances, and groups. A scene of several thousands of square meters, such as Fig. 19 and Fig. 22, can be generated in a few seconds. Our phenomenological approach for controlling the distributions of snow, leaves, grass, and icicles does not rely on computationally demanding physically-based simulations and allows for interactive editing.

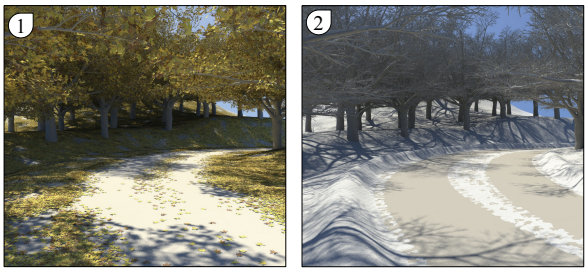

Figure 21: A countryside road in fall ① and in winter ②.

The road scene (Fig. 21) shows another example of a hierarchical grouping and instancing applied to a large scene. It was created by combining a terrain and 24 trees, which are instances of two groups of approximately 110 environmental objects yielding a total equivalent of 2, 653 environmental objects.

Authoring time for the most complicated scene (Fig. 19) consisting of almost 20k generated instances took approximately one hour. The interaction with the scenes for fine-tuning environmental and ambient fields for special effects such as large leaf piles or pedestrian steps in the snow was performed at interactive rates (less 0.1 second per local update). The time needed to create a new environmental object is related to the overall object complexity in terms of size (number of polygons) and expected behavior. The amount of time needed varied from a few minutes for simple objects, such as the lamppost or the bench, to approximately fifteen minutes for balconies, which contain many complex parts.

The entire scene recalculation for all examples was in general under 7.0 seconds. The most time-demanding part was the generation of large leaf piles. The road scene (Fig.21) has more than 1.4 millions leaves selected among more than 20 million virtual candidate leaves. By using level of details nodes and instancing, we were able to reduce the computation time from 24.0 to 4.5 seconds for Fig. 19 (see Table 1).

There is a negligible memory overhead for enhancing ob-

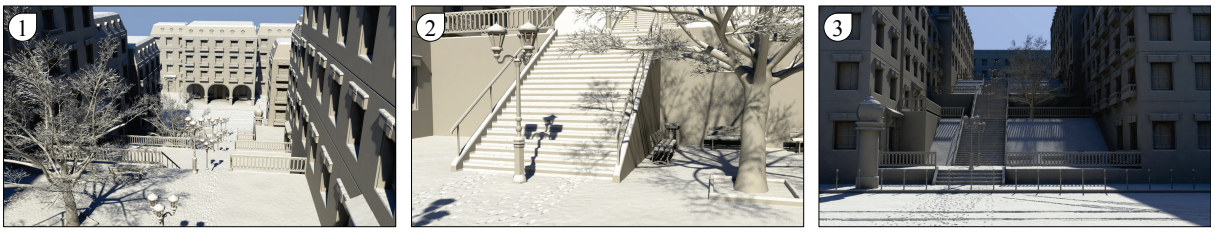

Figure 22: View of a Montmartre-like street in winter, covered by snow. Event though the scene is bigger than 3, 000 square meters, it contains a lot of small details, such as footprints ① ② or wheel tracks ③.

jects with environmental properties. In our implementation, the scalar fields are defined as skeletal implicit primitives which are a compact representation characterized by only a few parameters. In contrast, each effect can potentially generate additional geometry such as a snow mesh or leaves. For complex meshes, such as trees, it is beneficial to cache the snow geometry and only define its changes when necessary. For simpler objects, it is more efficient to generate the procedural geometry on the fly. Our instancing technique for representing some effects allows us to reduce the size of the generated detailed models.

7.5. Discussion

Our framework is focused on fast generation of various effects with many small scale details. Scalar fields and procedural effects provide a very efficient and intuitive framework to control and author complex scenes. A limitation of this approach lies in the lack of physical accuracy of the different effects produced. Unlike biologically-inspired or physically-based natural phenomena simulations in which features naturally emerge, complex features like snow bridges or ice column formation between different objects can only occur if they have been anticipated in the effects' algorithms.

Another limitation of our model is its inability to simulate interaction between effects. While the environmental objects react to the environment, the interaction between effects is not supported. Although different effects can be generated at the same time, e.g., fallen leaves on dry grass, or icicles and snow, those effects do not influence each other.

Our per-object representation allows us to optimize the generation of details. New effects can be easily added in our framework by implementing the corresponding effects parameterized by scalar fields. In practice, this means adding a new type into the list of fields of influence, which is rather straightforward, and defining for every kind of environmental object the corresponding effect. For example, rust could be integrated in our system in two steps. First, a rust field combined with the humidity field and accessibility fields would control the rust generation. Then, metallic environmental objects would implement the rust effect, whereas the appearance of other environmental objects would not be modified. Contrary to the global simulation methods, a limitation of the per-object approach is that effects should be optimized and implemented for different kinds of objects.

8. Conclusion

We have presented a novel model for the representation of large scenes with geometric details, guided by environmental factors. Objects are extended by effects and fields, making them environmental objects. Environmental objects react to the environment and change their appearance accordingly. Our approach has several advantages over existing appearance modeling techniques: it is intuitive, predictable, highly controllable, easy to implement, and fast. Our per-object approach allows for the creation of large scenes with a high level of details. Our framework allows for quick local and global changes that permits the interactive design of scenes. We have shown a variety of examples and provided several novel algorithms for generation of snow, icicles, leaf piles, grass, and trash; and new effects can be easily added in our framework.

There are several weaknesses of our method and possible avenues for future work. Our scenes are static and they are defined in a state of equilibrium. An improvement would be to combine our approach with techniques for dynamic simulations, such as those that can account for leaf movement and snowdrift in the wind. The environmental objects creation process could be partially or entirely automatized, based on an analysis of the object's geometry. Because our model addresses changes per object with techniques optimized for different effects, various algorithms such as aging, cracks and fractures or corrosion can also be implemented. Using a combination of effects, such as leaves falling on snow, or snow depositing on leaves, would also be an interesting way to enhance the framework. Another disadvantage to our method is that the effects are executed at the same time. However, there may be further hidden dependencies, for example snow could melt and produce water. This would require an entire simulation loop with a timeline that is not currently considered. Our environmental fields are approximations and it would be possible to calculate them precisely, for example shadows and other effects could be provided by global illumination algorithms.

References

[BSMM11] BENES B., STAVA O., MĚCH R., MILLER G.: Guided procedural modeling. *Comp. Graph. Forum 21*, 3 (2011), 325–334.

[DEJ*99] DORSEY J., EDELMAN A., JENSEN H. W., LEGAKIS J., PEDERSEN H. K.: Modeling and rendering of weathered stone. In *Proceedings Siggraph* (1999), SIGGRAPH '99, pp. 225–234.

[DGAG06] DESBENOIT B., GALIN E., AKKOUCHE S., GROSJEAN J.: Modeling autumn sceneries. In *Eurographics Short Papers* (2006), pp. 107–110.

[DH96] DORSEY J., HANRAHAN P.: Modeling and rendering of metallic patinas. In *Proceedings SIGGRAPH '96* (1996), pp. 387–396.

[DHL*98] DEUSSEN O., HANRAHAN P., LINTERMANN B., MĚCH R., PHARR M., PRUSINKIEWICZ P.: Realistic modeling and rendering of plant ecosystems. *Conference on Computer Graphics and Interactive Techniques* (1998), 275–286.

[DPH96] DORSEY J., PEDERSEN H. K., HANRAHAN P.: Flow and changes in appearance. In *Proceedings Siggraph* (1996), SIGGRAPH '96, pp. 411–420.

[DRS08] DORSEY J., RUSHMEIER H., SILLION F.: *Digital Modeling of Material Appearance*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[FB07] FOLDES D., BENES B.: Occlusion-based snow accumulation simulation. In *Workshop on Virtual Reality Interactions and Physical Simulation* (2007), pp. 35–41.

[Fea00] FEARING P.: Computer modelling of fallen snow. In *Proceedings of the 27th Annual Conference on Computer Graphics andInteractive Techniques* (2000), SIGGRAPH '00, pp. 37–46.

[FO02] FELDMAN B., O'BRIEN J.: Modeling the accumulation of wind-driven snow. In *Proceedings of Siggraph 2002, Technical Sketch* (2002), p. 218.

[GP11] GAGNON J., PAQUETTE E.: Procedural and interactive icicle modeling. *The Visual Computer 27*, 6-8 (2011), 451–461.

[JPK13] JEONG S., PARK S.-H., KIM C.-H.: Simulation of morphology changes in drying leaves. *Comp. Graph. Forum 32*, 1 (2013), 204–215.

[KAL06] KIM T., ADALSTEINSSON D., LIN M.: Modeling ice dynamics as a thin-film Stefan problem. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer animation* (2006), pp. 167–176.

[KG93] KHARITONSKY D., GONCZAROWSK J.: A physically based model for icicle growth. *The Visual Computer 32*, 1 (1993), 88–100.

[KHL04] KIM T., HENSON M., LIN M.: A hybrid algorithm for modeling ice formation. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (2004), pp. 305–314.

[KL03] KIM T., LIN M.: Visual simulation of ice crystal growth. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (2003), pp. 86–97.

[KL05] KONTKANEN J., LAINE S.: Ambient occlusion fields. *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games - SI3D '05* (2005), 41.

[KPK10] KRECKLAU L., PAVIC D., KOBBELT L.: Generalized use of non-terminal symbols for procedural modeling. *Computer Graphics Forum 29*, 8 (2010), 2291–2303.

[LZK*04] LANGER M., ZHANG L., KLEIN A., BHATIA A., PEREIR J., REKHI D.: A spectral-particle hybrid method for rendering falling snow. In *Proceedings of Eurographics Symposium on Rendering* (2004), pp. 217–226.

[MC00] MURAOKA K., CHIBA N.: Visual simulation of snowfall, snow cover and snowmelt. In *Proceedings Seventh International Conference on Parallel and Distributed Systems: Workshops* (2000), pp. 187–194.

[MG08] MÉRILLOU S., GHAZANFARPOUR D.: A survey of aging and weathering phenomena in computer graphics. *Computer and Graphics 32*, 2 (2008), 159–174.

[MGG*10] MARÉCHAL N., GUÉRIN E., GALIN E., MÉRILLOU S., MÉRILLOU N.: Heat transfer simulation for modeling realistic winter sceneries. *Comp. Graph. Forum 29*, 2 (2010), 449–458.

[MMPP03] MUNDERMANN L., MACMURCHY P., PIVOVAROV J., PRUSINKIEWICZ P.: Modeling lobed leaves. In *Proceedings of the Computer Graphics International Conference* (2003), pp. 60–65.

[MWHG06] MUELLER P., WONKA P., HAEGLER S., GOOL A. U. L. V.: Procedural modeling of buildings. In *ACM Transactions on Graphics* (2006), vol. 25, pp. 614–623.

[NIDN97] NISHITA T., IWASAKI H., DOBASHI Y., NAKAMAE E.: A modeling and rendering method for snow by using metaballs. *Comp. Graph. Forum 16*, 3 (1997), 357–364.

[PGGM09] PEYTAVIE A., GALIN E., GROSJEAN J., MERILLOU S.: Procedural generation of rock piles using aperiodic tiling. *Comp. Graph. Forum 28*, 3 (2009), 1801–1809.

[PTMG08] PEYRAT A., TERRAZ O., MÉRILLOU S., GALIN E.: Generating vast varieties of realistic leaves with parametric 2Gmap L-Systems. *The Visual Computer 24*, 7-9 (2008), 807–816.

[PTS99] PREMOZE S., THOMPSON W., SHIRLEY P.: Geospecific rendering of alpine terrain. In *Eurographics Workshop on Rendering* (1999), pp. 107–118.

[SPK*14] STAVA O., PIRK S., KRATT J., CHEN B., MĚCH R., DEUSSEN O., BENES B.: Inverse Procedural Modelling of Trees. *Computer Graphics Forum 33*, 6 (2014), 118–131.

[STBB14] SMELIK R. M., TUTENEL T., BIDARRA R., BENES B.: A survey on procedural modelling for virtual worlds. *Computer Graphics Forum* (2014), 31–50.

[vFG09] VON FESTENBERG N., GUMHOLD S.: A geometric algorithm for snow distribution in virtual scenes. In *Proc. of the Fifth Eurographics conference on Natural Phenomena* (2009), pp. 17–25.

[vFG11] VON FESTENBERG N., GUMHOLD S.: Diffusion based snow cover generation. *Comp. Graph. Forum 30*, 6 (2011), 1837–1849.

[WGG99] WYVILL B., GUY A., GALIN E.: Extending the CSG Tree. Warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2 (1999), 149–158.

[WH91] WEJCHERT J., HAUMANN D.: Animation aerodynamics. *ACM SIGGRAPH Computer Graphics 25*, 4 (1991), 19–22.

[WZF*03] WEI X., ZHAO Y., FAN Z., WEI L., YOAKUM-STOVER S., KAUFMAN A.: Blowing in the wind. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (2003), pp. 75–85.