Efficient modeling of entangled details for natural scenes

Eric Guérin, Eric Galin, François Grosbellet, Adrien Peytavie, Jean-David Genevaux

Pacific Graphics 2016
E. Grinspun, B. Bickel, and Y. Dobashi
(Guest Editors)

Volume 35 (2016), Number 7

Efficient modeling of entangled details for natural scenes

Eric Guérin[1,2] Eric Galin[1,3] François Grosbellet[1,4] Adrien Peytavie[1,4] Jean-David Génevaux

[1] Université de Lyon, CNRS
[2] INSA-Lyon, LIRIS, UMR5205, F-69621, France
[3] Université Lyon 2, LIRIS, UMR5205, F-69676, France
[4] Université Lyon 1, LIRIS, UMR5205, F-69622, France

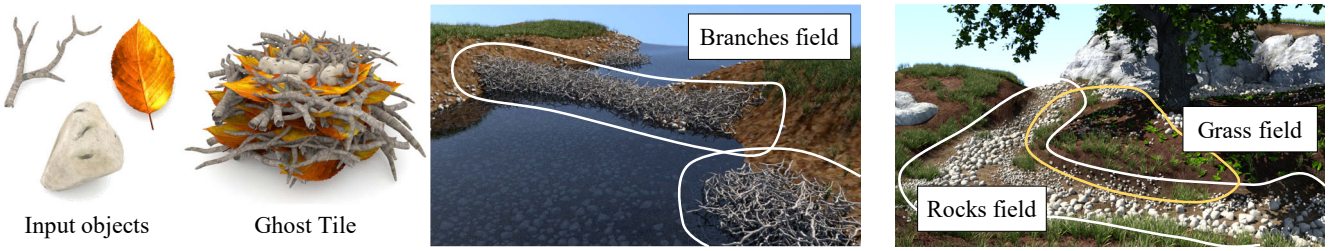

Figure 1: *Our method allows for the automatic generation of ground details such as grass tufts, rock piles, fallen leaves or twigs in natural landscapes. Given a set of geometric models, our algorithm automatically creates a Ghost Tile structure that stores overlapping candidate objects and a graph representing collisions between them. Given user-defined control density fields, our method instantiates candidates according to collision constraints stored in the graph. This control enables us to sculpt complex piles and thick layers of entangled objects.*

Abstract

Digital landscape realism often comes from the multitude of details that are hard to model such as fallen leaves, rock piles or entangled fallen branches. In this article, we present a method for augmenting natural scenes with a huge amount of details such as grass tufts, stones, leaves or twigs. Our approach takes advantage of the observation that those details can be approximated by replications of a few similar objects and therefore relies on mass-instancing. We propose an original structure, the Ghost Tile, that stores a huge number of overlapping candidate objects in a tile, along with a pre-computed collision graph. Details are created by traversing the scene with the Ghost Tile and generating instances according to user-defined density fields that allow to sculpt layers and piles of entangled objects while providing control over their density and distribution.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid and object representations I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—

1. Introduction

Modeling realistic landscapes covered with vegetation, rock piles, fallen leaves and branches is a perennial and important problem in computer graphics. The challenge stems not only from the complexity and diversity of shapes that need to be modeled, but also from the huge amount of small entangled objects that exist in a natural scene. Carefully authoring every detail by hand is a time-consuming editing task for the artists. Interactive editing tools provide artists with high level brushes to distribute objects over the surface of the terrain. To the best of our knowledge none of these tools allow the user to generate volumetric clusters of entangled objects. While several specific techniques have been proposed for generating stones and rocks piles or fallen leaves, those approaches do not allow the simultaneous placement of different types of objects in the same scene. Finally, physically-based collision detection or biologically-inspired ecosystem simulations such as lichen growth simulation methods can guarantee collision-free placement of objects, but at the cost of expensive computations.

In this paper, we introduce a unified framework for augmenting scenes with thick layers and piles of different types of entangled objects with mass-instancing. Our approach builds from the observation that these details can be approximated as replications of a few reference objects.

Our method consists in computing an original generic structure,

the *Ghost Tile*, that stores a dense collection of partially overlapping candidate geometric objects in a tile and a collision graph between those objects. The Ghost Tile structure is computed once and for all as a pre-processing step. The different details are then distributed and controlled by a set of field functions that define the relative density of details in regions of space. Geometric models are finally created by tiling the scene with the Ghost Tile and instantiating objects according to the density fields and to the constraints of the collision graph (Figure 1). Moreover, our method allows the user to use control fields to constrain the instantiation process and specify the size or the orientation of generated instances. More precisely, our contributions are as follows:

- We present a generic, simple and efficient model, the Ghost Tile, for creating a vast variety of entangled details. Our unified framework not only allows for the generation of entangled rocks, branches, fallen leaves or grass on the ground, but also permits to sculpt complex shapes made of clustered details.
- We propose a simple volumetric density field model that allows us to freely sculpt any kind of shape that will be then filled with entangled details. Our method provides the user with both local and global control over the distribution of the different types of objects.
- Contrary to existing methods, our tiling method is fast and allows for interactive editing and sculpting. Moreover, our instantiation scheme can generate hundreds of thousands of objects with a reduced memory footprint.

Our framework is versatile and can be used in combination with simulations, procedural generation and interactive editing. Interactive editing is achieved by controlling density fields through standard primitives, which can be performed in real time. Although our method relies on tiling, the generated arrangements of details do not show visible repetition patterns and even though collisions are not fully avoided, interpenetrations are not noticeable.

2. Related work

Several methods have been proposed to generate realistic distributions of details in a natural scene. Existing techniques can be organized into four categories: biologically-inspired and physically-based simulations, procedural and pattern-based distribution methods, texture synthesis and interactive editing.

Simulations aim at generating realistic distribution of details according to physically or biologically-inspired algorithms. General multi-body simulations [HK10] generate piles of shapes, but lack control over the distribution and are computationally demanding. Ecosystem simulations [DHL*98, AD05] have been successfully used for generating vegetation plants competing for resources such as nutriments, lights or space. Desbenoit *et al.* [DGAG06] proposed to use a wind simulation coupled with a stochastic aging process to generate leaves falling and piling onto the ground. Lichens were also addressed [DGA04] by introducing the Open Diffusion Limited Aggregation constrained to the surface of the objects and taking into account the characteristics of the environment. An important limitation of simulation approaches is that they provide a limited user-control over the placement of plants or ground details. Moreover, they do not scale well with the size of the scene and cannot be used for modeling large scenes or generating hundreds of thousands of small objects.

Procedural techniques rely on simple production rules to generate distributions of objects organized into patterns or structures. CGA shape grammars [MWH*06] have been successfully used to create buildings from a set of user-defined input geometric models. A similar rule-based approach was used to create brick walls with procedurally defined patterns [LDG01]. Those techniques are limited to regular layouts and therefore cannot be used for generating natural details.

In contrast, stochastic distributions based on tiled Poisson disk and sphere distributions [LD06] have been successfully used to create random natural distributions of plants. Aperiodic Wang tiles were also used to generate trees [AD06]. Poisson distribution [JZW*15] approaches generate non overlapping objects distributions however, and do not lend themselves for modeling dense piles or layers of entangled objects.

Impostors controlled by density maps is another general two dimensional-technique frequently used for rendering ground vegetation in real-time. While it can efficiently approximate ground details such as grass tufts as well as bushes [DN04, FUM05, BCF*05, BN12], it cannot be used to create piles of leaves or rocks. An important limitation of those approaches is that consistent contact between objects cannot be guaranteed. A specific technique was proposed in [PGMG09] for modeling rock piles by combining aperiodic tiling and an erosion-based modeling system. This approach is memory demanding as it requires the generation of hundreds of different rock models to guarantee consistent contact between piled rocks. A more general approach [SM14] consists in randomly distributing objects over a control shape and using a relaxation method to limit collisions. It becomes more computationally demanding as the amount of details increases. In contrast, our method can sculpt piles or thick layers of different kinds of entangled objects with only a few input models, at the cost of a relaxed collision constraint.

Solid texture synthesis techniques can generate large scale structured aggregates from a small exemplars [JDR04, DHM13, ZDL*11]. Discrete element textures [MWT11] were successfully used to generate piles of objects by using a three-dimensional texture synthesis technique. They rely on computationally demanding energy minimization algorithms and are not adapted for interactive modelling.

Interactive editing tools allow the user to distribute instances by the mean of smart brushes that automatically adapt the distribution of details according to the characteristics of the terrain or the environment. Emilien *et al.* [EVC*15] proposed an original method based on a statistical analysis of input examples to reproduce the distribution patterns of large scale objects such as trees or terrain features in a scene. Grosbellet *et al.* [GPG*16] introduced environment sensitive objects that automatically instantiate details such as fallen leaves or grass according to user-defined environment variables such as humidity, temperature or accessibility. Our method also relies on control fields for sculpting the volume and controlling the relative density of entangled objects, but can be directly applied without explicitly programming the behavior of objects.

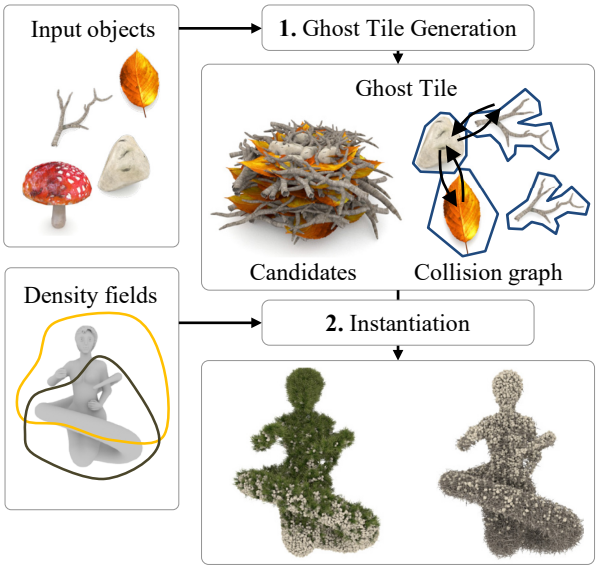

Figure 2: *Given input geometric models, the Ghost Tile structure groups a few hundreds of overlapping candidate models distributed in the tile, and stores the collision graph between objects in the reference tile. The instantiation process traverses the scene, selects and instantiates candidates on the fly from the pre-computed Ghost Tile according to user-controlled density fields.*

3. Overview

Our method is based on the definition of a Ghost Tile structure, referred to as $\mathcal{T}$, which is defined as a collection of candidate objects $\mathcal{O} = \{\mathcal{O}_i\}$ and a collision graph $\mathcal{G}$ embedded in a tile. The Ghost Tile stores the position of the different candidate objects in the tile. The collision graph encodes the intersections between the objects inside the tile and with neighboring tiles. Since the density of candidate objects is very high, we are able to produce many different collision-free configurations by picking only a subset of candidate objects that do not intersect each other. Our method can be divided into two steps (Figure 2).

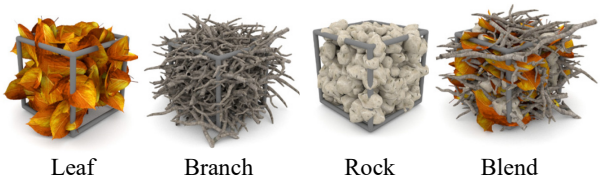

Figure 3: *Several Ghost Tile examples, with only one type of object (leaf, branch and rock) and with different types of objects.*

Ghost Tile generation is a pre-processing step that aims at generating a collection of candidate objects in a tile and a corresponding set of constraints defined as a collision graph between these candidates. Candidates are generated by distributing input objects at different positions and orientations in the tile (Figure 3). Note that candidates can partially straddle the tile boundary and overlap several tiles.

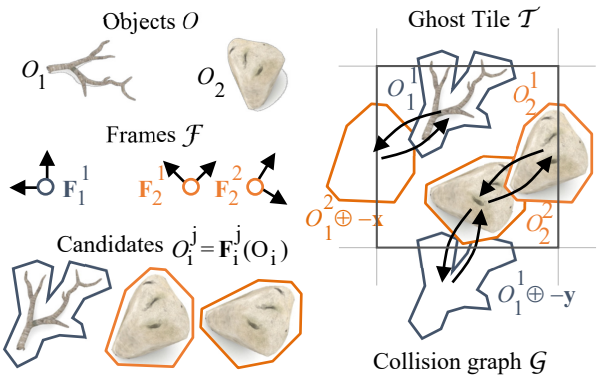

Figure 4: *The Ghost Tile structure is created from a set of input objects $\mathcal{O} = \{\mathcal{O}_i\}$. It is composed of a list of candidate objects $\mathcal{O}_i^j = \mathbf{F}_i^j \circ \mathcal{O}_i$, where $\mathbf{F}_i^j$ represent frames, and a collision graph $\mathcal{G}$ that encodes intersection relationships between candidate objects.*

Instantiation computes the details by traversing the scene with the Ghost Tile and instantiating candidate objects according to the user-prescribed control volume and densities. This step can process many different control volumes and/or densities with the same Ghost Tile. Note that we do not need to rely on aperiodic tiling schemes to avoid tiling artifacts: whenever several candidates have the same or similar priorities, we randomly select a candidate so as to avoid such tiling artifacts. Instances are represented as a list of indexes, which is very efficient in terms of memory when dealing with complex scenes featuring hundreds of thousands of instances.

4. Ghost Tile generation

In this section, we present the Ghost Tile data structure, useful notations used throughout the paper and we detail the algorithm for creating it. Figure 4 shows a simplified version of the structure with only a few objects out of clarity as Ghost Tiles are composed of hundreds of thousands of candidates.

4.1. Ghost Tile structure

Given an input set of objects $\mathcal{O} = \{\mathcal{O}_i\}$ representing the different details that may be created during the instantiation step, a Ghost Tile $\mathcal{T}$ is composed of a collection of candidates $\mathcal{O}_i^j$ and a collision graph $\mathcal{G}$ connecting them. Let $\{\mathcal{O}_i\}$ denote the input set of objects that will be used in the process. We denote $\mathcal{O}_i^j = \mathbf{F}_i^j(\mathcal{O}_i)$ the j^{th} candidate created from object $\mathcal{O}_i$ with $\mathbf{F}_i^j$ referring to its associated frame. Candidate objects are linked to an axis-aligned cubic tile of size s. They can partially straddle the cubic tile boundaries however, and therefore overlap several tiles. The notation $\mathcal{T} \oplus \mathbf{t}$ will refer to a tile $\mathcal{T}$ translated by a vector $\mathbf{t} = s\,\mathbf{v}$ where s refers to the size of the tile and $\mathbf{v} \in [-1,0,1]^3$ is a vector of integers indicating the offset between two tiles.

For every input object $\mathcal{O}_i$, we define a list of anchor points (Fig-

ure 5) denoted as $\mathbf{a}_i^k$ attached to it. Anchor points will be used during the instantiation process for evaluating whether a candidate object $\mathcal{O}_i^j$ should be instantiated or not. This is performed by evaluating user-defined density functions $f(\mathbf{a}_i^k)$ at the anchor points of candidate objects (Section 5). In our system, anchor points are defined by the user in the local frame of every object $\mathcal{O}_i$. They can be also generated procedurally by sampling the volume or the surface of the objects.

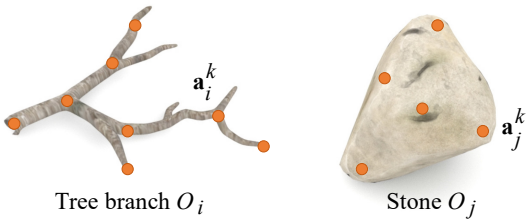

Figure 5: *Anchor points for a branch and a stone. Anchor points serve for the evaluation of the control density fields during the instantiation process.*

The collision graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is composed of a set of nodes $\mathcal{N} = \{\mathcal{N}_k\}$ that represent the set of candidate objects and of oriented edges $\{\mathcal{E}_{ij}\} = \{(\mathcal{N}_i, \mathcal{N}_j, \mathbf{v})\}$ that represent collision relationships. More precisely, an edge $(\mathcal{N}_i, \mathcal{N}_j, \mathbf{v})$ encodes that the object at node $\mathcal{N}_i$ is in collision with the object at the node $\mathcal{N}_j$ located in the tile $\mathcal{T} \oplus \mathbf{t}$.

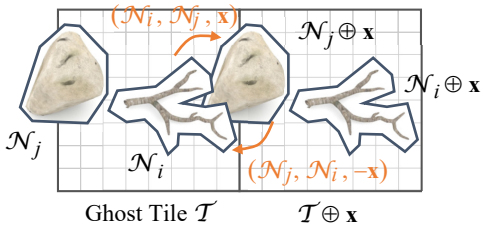

Figure 6: *Notations for nodes and arcs between candidate objects, candidate objects may intersect other objects located at a different tile position.*

Thus, $\mathbf{t} = \mathbf{0}$ when the two objects are in collision in the same Ghost Tile (Figure 6). Note that every edge $(\mathcal{N}_j, \mathcal{N}_k, \mathbf{v})$ in the graph has a symmetric pair $(\mathcal{N}_k, \mathcal{N}_j, -\mathbf{v})$ by construction.

User constraints can be used to prescribe specific orientations for some objects, which reduces the number of candidates in the Ghost Tile structure and speeds up the overall process. For instance fallen branches may have random orientations in the horizontal plane, and random tilt angle comprised in $\pm 45°$.

4.2. Ghost Tile creation

The Ghost Tile creation algorithm proceeds in two steps: candidate object distribution and collision graph computation. Given a set of input objects $\mathcal{O} = \{\mathcal{O}_i\}$, the first step creates candidates into the cubic tile by distributing objects $\mathcal{O}_i$ into the tile with different orientations and positions. The second step computes the collision graph $\mathcal{G}$ between the previously generated candidates. The overall algorithm is as follows:

1. For all objects $\mathcal{O}_i$, compute a random frame $\mathbf{F}_i^j$ according to the user constraints, and add the new candidate $\mathcal{O}_i^j = \mathbf{F}_i^j(\mathcal{O}_i)$ to the set of nodes: $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{O}_i^j$.
2. Compute the intersection between the nodes $\mathcal{N}_k$ in the tile $\mathcal{T}$ with offset tiles $\mathcal{T} \oplus \mathbf{v}$ and the new node $\mathcal{O}_i^j$. If an intersection occurs, add the two arcs $(\mathcal{N}_k, \mathcal{O}_i^j, \mathbf{v})$ and $(\mathcal{O}_i^j, \mathcal{N}_k, -\mathbf{v})$ to the graph $\mathcal{G}$.

Candidates $\mathcal{O}_i^j$ are generated by distributing objects in the tile. The user can control different types of distribution according to the type of the objects.

Figure 7: *Comparison of different results obtained by controlling the distribution of the orientation of candidate objects: left image shows random orientations and right image shows 75% almost horizontal ($0° - 30°$) and 25% with inclined orientation ($45° - 60°$).*

In our system, we implemented different strategies for computing the orientation of the distributed objects. *Random orientation* lends itself for piles or thick layer where objects can have totally different orientations, such as pile of rocks or pebbles. *Constrained orientation* strategies rely on a user-controlled distribution of orientations, and lends itself for controlling fallen leaves, branches or even grass tufts. Figure 7 shows different kinds of effects obtained by modifying the statistics of the orientation of candidate objects. Note that the orientation can also be constrained during the instantiation process (see section 5.3).

4.3. Graph computation

For every candidate $\mathcal{O}_i^j$, we add a node $\mathcal{N}_k$ into the graph. We then compute the possible intersections between pairs of nodes and store those intersections as edges in the graph.

At the end of this step, we have obtained a collision graph $\mathcal{G}$ between the candidates $\mathcal{O}_i^j$. A brute-force collision detection between all pairs of candidates would be computationally demanding because the complexity would be in $O(\#\mathcal{N}^2)$ and in our case $\#\mathcal{N}$ can be really large (up to tens of thousands of candidate objects). Therefore, we optimize collision detection by using a grid as an accelerating data structure.

Collision detection We process the intersection between candidate objects $\mathcal{O}_i^j$ by approximating the shape of input objects using sphere sets (Figure 8). This representation can be computed either automatically [WSL*06, SKS12] or optimized manually with a geometric editor.

Figure 8: *Some input objects approximated by unions of spheres.*

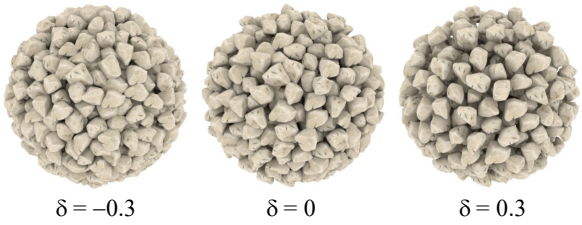

Figure 9: *The interpenetration threshold* δ *provides control over the density of instances either by allowing a slight interpenetration* ($\delta < 0$) *or forcing a minimum distance between instances* ($\delta > 0$).

The union of spheres model lends itself for our Ghost Tile generation algorithm as rotations, translation and uniform scaling used in the definition of the frames $\mathbf{F}_i^j$ of candidate objects $\mathcal{O}_i^j$ can be directly applied to union of spheres. Note that the union of spheres lends itself for the automatic generation of anchor points. The intersection test consists in computing an approximation of the interpenetration distance d between two sphere-sets and compare it to a user-defined threshold δ. Let $S(\mathbf{c},r)$ denote a sphere of center $\mathbf{c}$ and radius r. Let $\mathcal{A} = \{S(\mathbf{a}_i,r_i)\}, i \in [0,n-1]$ and $\mathcal{B} = \{S(\mathbf{b}_j,r_j)\}, j \in [0,m-1]$ two sphere-sets with their corresponding centers and radii. We define the interpenetration distance $d(\mathcal{A},\mathcal{B})$ as:

$$d(\mathcal{A},\mathcal{B}) = \min_{i,j} \|\mathbf{b}_j - \mathbf{a}_i\| - (r_i + r_j)$$

We consider that two candidates intersect if $d(\mathcal{A},\mathcal{B}) \leq \delta$. Figure 9 illustrates the influence of the threshold parameter δ: a positive value can be used to force objects to be placed at a minimum distance, whereas a negative value allows instances to interpenetrate. Small negative distance thresholds are recommended with a view to generating dense piles of objects: although instances may intersect, their interpenetration remains almost invisible providing a good approximation for objects into contact (Figure 22).

Grid-based acceleration We speed-up the overall collision graph generation process by using a grid-based decomposition of space which is adapted to our dense candidate object distributions in the Ghost Tile. Objects are first projected on the grid and collision detection is performed on the instances that have common voxels. Every voxel of the grid stores a list of identifiers that define which candidates $\mathcal{O}_i^j$ intersect the voxel. We use $(i,j,\mathbf{v})$ as the identifier, where i,j refer to candidate $\mathcal{O}_i^j$ and $\mathbf{v}$ represents the integer offset vector to a neighboring tile which is necessary to process objects straddling outside of the tile (Figure 10).

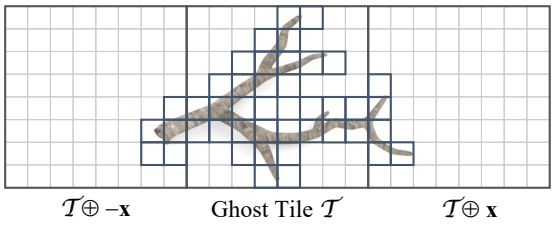

Figure 10: *Objects that are out of the reference tile boundaries are taken into account by storing an offset index.*

5. Instantiation

The instantiation is controlled by a set of density fields $f = \{f_i\}$ that prescribes the relative density and the orientation of the different details in space. The algorithm tiles space with the Ghost Tile structure and instantiates candidates according to the value of the density field f_i at their anchor points and to the collision graph. The two sets of rejected and instantiated candidates will be denoted as $\mathcal{R}$ and $\mathcal{I}$ respectively.

Density fields are mathematically defined as functions $f_i : \mathbb{R}^3 \rightarrow \mathbb{R}$. Any kind of function can be used, provided that it is continuous. In our implementation, we rely on density functions created from skeletal primitives and combination operators as in the Blob-Tree [WGG99]. This primitive-based model allows us to control the density of details in different regions of space efficiently. More precisely, our approach enables us to define volumetric densities which enables us to freely sculpt any kind of shape, piles or thick layers which will be then filled with details.

Candidates of type i should be instantiated only in regions of space where the corresponding density function f_i is strictly positive. Moreover, the values of the fields f_i define the relative density of each kind of object. If the sum of densities is less than 1, the instantiation process fills only a fraction space with details. Otherwise, we create as many details as possible according to the average density.

The overall process is performed in three steps: tiling and instance culling, priority management between candidate objects, and finally selection and propagation of constraints.

5.1. Tiling and instance culling

The first step of the instantiation process consists in tiling the entire scene volume with the Ghost Tile. Recall that anchor points are defined in the local frame of every object $\mathcal{O}_i$. Therefore, the world coordinates of the anchor point $\mathbf{a}_i^k$ in a Ghost Tile translated by a vector $\mathbf{t}$ are defined as $\mathbf{t} + \mathbf{F}_i^j\ \mathbf{a}_i^k$.

For every tile of coordinate $\mathbf{t}$, every object $\mathcal{O}_i$, every instance of this object $\mathcal{O}_i^j$, and every anchor point $\mathbf{a}_i^k$, we evaluate the density function $f_i(\mathbf{t} + \mathbf{F}_i^j\ \mathbf{a}_i^k)$. If $\exists k \,|\, f_i(\mathbf{t} + \mathbf{F}_i^j\ \mathbf{a}_i^k) = 0$, then the instance $\mathcal{O}_i^j \oplus \mathbf{t}$ is considered outside of the control volume and discarded. Otherwise, the instance is added to the prioritized list of candidates (Figure 11). Anchor points may be used to control how the details may straddle or lie strictly inside the control volume (Figure 12).

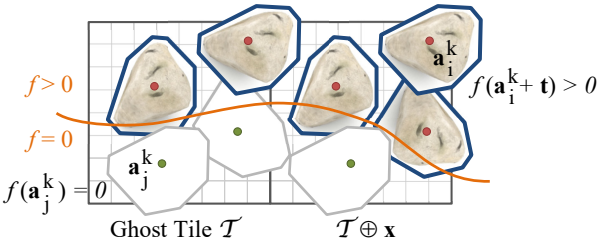

Figure 11: *Culling discards candidates $\mathcal{O}_i^j$ such that the density function at their anchor points $f(\mathbf{t} + \mathbf{F}_i^j \mathbf{a}_i^k) = 0$.*

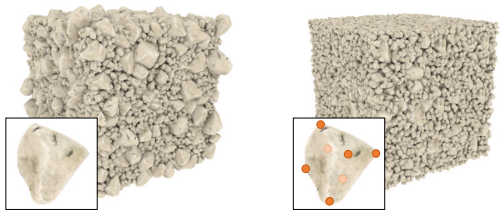

Figure 12: *Anchor points provide user-control over volume filling. When only anchor points are inside the control volume, instances may straddle out of the control volume (left). By placing a few anchor points on the surface of the object, such instances are discarded which provides a tighter volume control (right).*

When using a single anchor point located at the center of the object, it can straddle outside of the control volume. In contrast, constraints are better enforced when using several anchor points distributed inside and at the surface of the objects.

5.2. Priority strategy

When a candidate is selected, it is placed in a priority list. Every type of object stores its associated priority list. We compute the priority value for every anchor point of the candidate and apply the maximum found. The priority lists are managed according to three different user-specified strategies (Figure 13).

Random priority simply selects a random candidate from the list. Random priority lends itself for the generation of thin layers of objects or sparse objects randomly distributed over the terrain, such as a few scattered rocks or a few leaves fallen (Figure 1,19). In contrast this strategy is not appropriate for the generation of piles of objects such as rock or leaf piles as the random selection does not generate the most dense distribution of objects almost into contact.

Elevation-based priority is directly computed as the opposite of the z-component of translation associated to $\mathbf{F}_i^j$. It fills control volumes with details starting with lower altitudes candidates. In contrast, the elevation-based priority efficiently approximates compact piles of objects that are into contact by incrementally instancing successive layers.

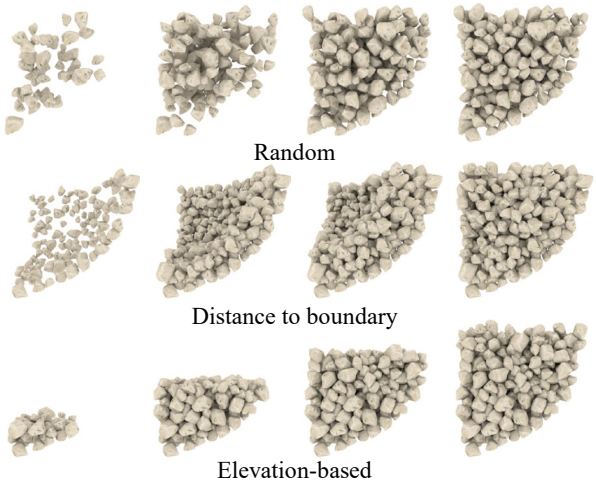

Figure 13: *Overview of different strategies for instancing candidate objects. Random priority may produce space between objects, whereas elevation-based and distance-to-boundary priority schemes always select a candidate close to already instantiated objects, which produces more compact piles.*

Distance-to-boundary priority starts placing instances on the boundary before filling the interior of the control volume. The density fields f_i that describe the control volumes provides us with an approximation of the distance to the boundary of the volume. The algorithm sorts candidates according to the distance to the boundary, and iteratively selects the nearest instance to the boundary while invalidating candidates that are in collision with it. Moreover, this strategy allows to reduce the number of instantiated objects by only generating a few layers and leaving the interior of the density field volume empty.

5.3. Candidate selection and constraint propagation

In order to respect the relative densities of each objects in the tile, we first evaluate the densities at the center of the tile and choose an object $\mathcal{O}_i$ randomly according to the probability. We select the candidate $\mathcal{O}_i^j$ with the highest priority. Because the density function is not constant inside the tile, we evaluate the density functions at the anchor positions $f(\mathbf{t} + \mathbf{F}_i^j \mathbf{a}_i^k))$. Let $\|f\|$ denote the $\mathcal{L}^1$ norm of the column vector of density function values. If $\|f\| > 1$ then instance will be selected. Otherwise, let u denote a random number uniformly generated on unit interval, the candidate will be instantiated only if $u \leq \|f\|$. This allows us to control the density of details, and create mixed objects distributions as prescribed by f (Figure 15).

We then update the list of candidates in the considered tile by querying the collision graph $\mathcal{G}$. We also update in the neighboring tiles the list of candidates that need to be discarded because they intersect the selected instance (Figure 16).

Control Some parameters such as the orientation can be constrained in order to modify the instantiation rules and achieve special effects. Given a user-prescribed direction field $\mathbf{u} : \mathbb{R}^3 \to \mathbb{R}^3$,

Figure 14: *Volumetric density fields guide the generation of heterogeneous piles of entangled details very efficiently and intuitively by controlling the relative contributions.*

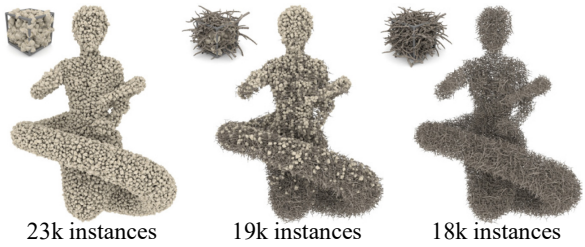

Figure 15: *Our system can fill complex volumes with many small instances. Here, the rock and branches density fields were defined using skeletal implicit field functions to represent a statue.*

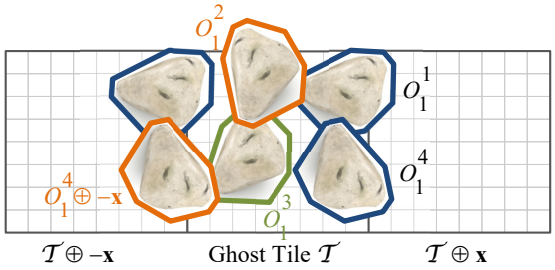

Figure 16: *The candidate with the highest priority ($\mathcal{O}_1^3$ in green) is selected and the collision graph $\mathcal{G}$ discards candidates in collision in the current tile ($\mathcal{O}_1^2$ in orange) as well as candidates in the neighboring left tile ($\mathcal{O}_1^4 \oplus -\mathbf{x}$ in orange).*

constraints are taken into account during the selection step by computing the dot product $\mathbf{z}(\mathbf{F}_i^j) \cdot \mathbf{u}(\mathbf{t} + \mathbf{F}_i^j\, \mathbf{0})$ between the z-vector of the frame $\mathbf{F}_i^j$ of the candidate and the direction field evaluated at the origin of the frame. The candidate is selected if the dot product is greater than a given threshold. Figure 17 shows two spheres covered with grass tufts instantiated from the same Ghost Tile. Left image shows a vertical constraint with a constant field $\mathbf{u}(\mathbf{p}) = \mathbf{z}$. Right image was computed using a procedural constraint by defining $\mathbf{u}$ as the gradient of the density function $\mathbf{u} = \nabla f_i$. Note that the tile contains grass tufts distributed in all directions; the selection process selects the right ones.

6. Results

Our method has been implemented in C++/OpenGL and tested on Intel Core *i*7 with 16 GB of RAM. High quality renderings were produced by directly ray tracing mass-instantiated details.

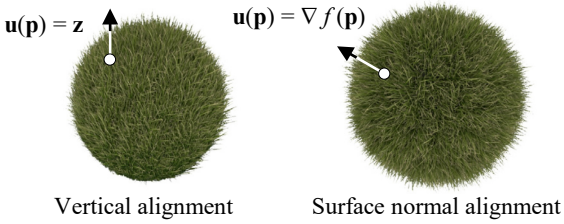

Figure 17: *Normal constraints allows the user to control details orientations. Instances can be selected either with a constant orientation prescribed by the user (left) or with arbitrary rotations around the gradient of the density field (right).*

6.1. Creation and control

Ghost Tiles can be used as a detail generation tool in combination with simulations, procedural generation and interactive editing. Moreover, our method provides both local and global control over the generation of details. Figure 14 shows that our method can be used to generate piles of different materials efficiently, including with different entangled objects. Density fields allow us to sculpt the shape of complex piles and control the relative density of entangled objects easily.

Parameters are set so that the size of the Ghost Tile ranges from 1 to 3 times the size of the largest objects in it. In several examples shown throughout the paper, grass tufts and leaves were set to $\sim 10\,\text{cm}$, rocks and stones were set to $\sim 10-20\,\text{cm}$ and branches to $\sim 20-30\,\text{cm}$. The Ghost Tile size was set to $\sim 50\,\text{cm}$.

Figure 18: *Undergrowth scene showing very complex stacking and entangling between grass tufts, fallen leaves and mushrooms. Complex layers featuring different types of objects were created by combining several partially overlapping density fields.*

Figure 18 shows a closeup of an undergrowth scenery. Our

Figure 19: *The grass and fallen leaves layers and the rock piles of the meadow were authored in a few minutes using our interactive editor.* $\sim 50\,\text{k}$ *objects were instantiated.*

Name	Figure	$\#\mathcal{O}$	$\#\mathcal{N}$	$\#\mathcal{E}/2$	$\#\mathcal{E}/(2\#\mathcal{N})$	Time (s)	Storage (kB)
Rocks	22	1	$0.32k$	$14k$	44	0.17	80
Branches, rocks	21	3	$1.12k$	$990k$	901	93.0	4770
Autumn leaves	14	4	$0.32k$	$102k$	319	31.0	454
Mushrooms	23	2	$0.17k$	$137k$	1370	87.5	559
Pine needles, leaves	18	2	$0.85k$	$941k$	1107	138	4290

Table 1: *Statistics for the Ghost Tile creation step, reported time is in seconds and memory in kilobytes. The tile size was* $\sim 50\,\text{cm}$ *whereas the size of the objects was* $\sim 10\,\text{cm}$ *for rocks and leaves,* $\sim 15\,\text{cm}$ *for mushrooms and* $\sim 30\,\text{cm}$ *for branches.*

Figure 20: *Haystacks were created using cylinder-shaped density fields. Orientation constraints were prescribed by using a rotational field. The scene features* $\sim 4260\,\text{k}$ *instances.*

method takes into account the complex shape of input objects. The ground is covered with different small objects, including leaves, pine needles and grass tufts that do not intersect each other. Small mushrooms were automatically instantiated below the cap of larger ones and are partially covered by leaves and needles in very complex entanglements which would be difficult to obtain with other existing techniques.

Interactive painting and sculpting is made possible by the performance of the instantiation process, which allows to use the Ghost Tile in an interactive editing environment. We have implemented a simple painting and sculpting tool to edit the density function primitives (see accompanying video). The Ghost Tile is pre-computed once and for all and stored in memory. During the interactive editing session, density fields are edited in real time according to the user input strokes. Details are generated only inside the tiles where the density fields were modified. Primitives are combined together using blending and Boolean operators as described in [WGG99]. Finally, density fields are modified by computing the difference with the other solid objects in the scene such as the tree and the terrain in order to avoid interpenetration with details.

Figure 19 shows a meadow where the terrain is entirely covered with stones, grass and fallen leaves instantiated from a single Ghost Tile. The scene was authored in a few minutes, and features $\sim 50\,\text{k}$ instances.

Procedural generation is another approach to generate the density functions that are used to define details. Figure 20 shows a procedurally generated terrain with haystacks. The dried grass density field was automatically computed from the input height field, whereas the haystacks were sculpted by randomly distributing cylinder primitives over the terrain.

Figure 21 shows a river scenery with a beaver dam that was sculpted using generalized cylinder implicit primitives. Entangled branches were automatically created from this very simple control volume authored by the designer. In contrast, the density fields for the grass tufts and rocks were automatically generated from the river skeleton.

6.2. Performance

Recall that our method performs in two separated steps. First, the Ghost Tile is pre-computed once and for all and stored in memory. Then the details are generated by traversing the scene with the Ghost Tile and instantiating candidates according to the density fields.

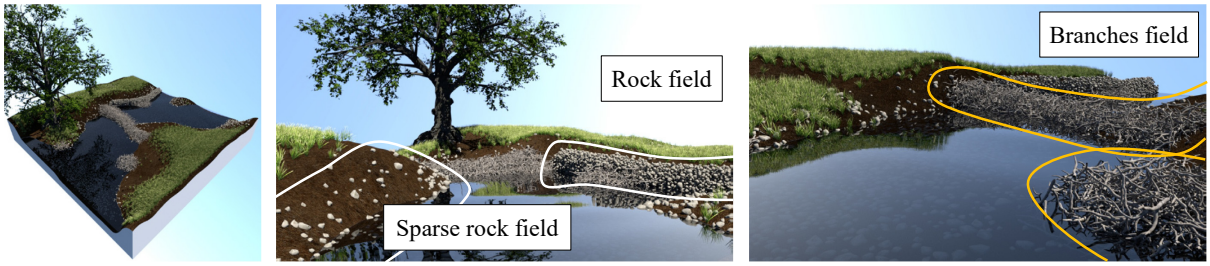

Figure 21: *The beaver dam was authored in less than one minute by sculpting a density field combined with a Ghost Tile full of branches of different sizes and orientations. The shape of the dam was obtained by assembling sphere and generalized cylinder implicit primitives. The control fields for grass tufts and rocks were procedurally generated according to the distance to the water body.*

	Name	Figure	$\#\mathcal{I}$	$\#\mathcal{R}$	Time (s)
Local	Meadow	19	0.11 k	58 k	0.3
	Meadow	19	0.75 k	320 k	1.2
	Meadow	19	2.0 k	577 k	3.4
	Mushrooms	23	48	219 k	0.9
Global	Field	20	1 990 k	35 000 k	54.6
	Beaver	21	36 k	26 300 k	10.7
	Haystack	20	81 k	10 400 k	6.8
	Borie	22	63 k	33 800 k	16.8

Table 2: *Instantiation statistics: reported time is given in seconds. Recall that $\#\mathcal{R}$ and $\#\mathcal{I}$ represents the number of candidates that have been rejected and instantiated respectively during the collision detection step of the instantiation process.*

Ghost Tile generation The performance of this step depends on the number of candidates $\#\mathcal{N}$ in the tile and on the number of collision arcs $\#\mathcal{E}$. An important statistic is the average number of collisions per candidate denoted as $\rho = \#\mathcal{E}/2\#\mathcal{N}$. Note that $0 \leq \#\mathcal{E} \leq \#\mathcal{N}(\#\mathcal{N}-1)$ where the upper bound represents the case when all candidates are intersecting each other (complete graph). Table 1 reports statistics and timings for computing different Ghost Tile setups. Note that the memory footprint of the collision graph $\mathcal{G}$ is small, ranging from less than one hundred kilobytes for simple arrangements with few collisions between objects, to a few megabytes for complex Ghost Tiles.

Given a few input objects ($\#\mathcal{O} \leq 4$), we create Ghost Tiles with hundreds of candidates ($300 \leq \#\mathcal{N} \leq 1200$) that partially overlap and intersect each other (average number of intersections per candidate $35 \leq \rho \leq 1\,400$). Generation is the more computationally demanding as ρ is high and as objects have complex shapes. This explains that the pre-processing time is high for leaves and branches as those objects are approximated by many spheres.

Instantiation cost is directly related to the size of the scene. Table 2 reports statistics for the instantiation process. Timings show that local editing tools that only operate on a few tiles in the scene perform at interactive rates as demonstrated in the meadow (Figure 19) and for a small scene such as Figure 18. In contrast, processing time increases when generating details over an entire scene, as the number of rejected candidates $\#\mathcal{R}$ increases as demonstrated in the *Beaver* dam and *Hut* scenes (Figure 21,22). Timings show that even for a large number of instances, the generation time is less than a minute.

6.3. Comparison with other techniques

Our method resembles in spirit the aperiodic tiling [PGMG09] proposed for generating rock piles. While our method does not guarantee consistent contact points between objects, our experiments show that visually plausible piles can be created by tuning the interpenetration distance parameter δ. Although both approaches rely on similar instantiation algorithms, our method compares favorably in terms of versatility, control and efficiency. Our framework allows us to create piles with different types of objects such as grass tufts of branches or rocks or even mushrooms. Moreover, the combination of several density fields allows for a better control over the relative density and orientation of details.

Finally, aperiodic tiling needs to generate thousands of different geometric models to guarantee consistent contact between rocks. In contrast, our framework allows the mass-instantiation of a few geometric models, which is considerably less memory demanding. Figure 22 shows an ancient dry stone hut and stone wall modeled with only two 2 reference rock models ($\sim$ 63 k instances). In contrast, the same scene created with aperiodic tiling required 7 355 different rock models ($\sim$ 28 k instances).

Our method also resembles to recent procedural approaches such as discrete element textures [MWT11] and non-periodic aggregates generation methods [SM14]. Both techniques rely on a computationally demanding energy-minimization step, which prevents interactive editing and limits the number of generated instances to a few thousand. In contrast, in our method, the collision detection is performed during the Ghost Tile generation pre-processing step, which enables us to speed up the instantiation process and generate complex scenes featuring tens of thousands of entangled objects.

Our method also compares favorably to wind-based simulation techniques for modeling fallen leaves. The leaf distribution ap-

Figure 22: *An ancient dry stone hut generated with our Ghost Tile technique (∼ 63 k flat stones). The shape of the hut was sculpted by defining a density field for rocks as the difference between a hollow sphere and several smaller spheres. The user defined another field to constrain the size of the stones around the openings.*

Figure 23: *The 3D shape of objects is taken into account in our collision solver leading to this automatic placement of mushrooms.*

proach presented in [DGAG06] relies on computationally demanding trajectory and collision computations for every leaf. In contrast, our method can generate visually plausible leaf piles and distributions. Although our method cannot guarantee precise contact between objects, results remain visually plausible.

Environmental objects [GPG*16] is another approach for generating details by augmenting a database of procedural objects with environment sensitive details such as grass, moss or leaves. The user needs to define the behavior of every object with respect to the different environment variables prior to using it. In contrast, our framework can be directly combined with other editing, simulation of procedural techniques to generate details on the surface or sculpt any kind of volumetric object filled with details.

6.4. Limitations

Because our model does not aim at generating structured arrangements, it generally fails at assembling details organized into regular patterns such as bricks in a wall. Since collisions between candidates are computed during a pre-processing step and stored in a static graph, our method cannot handle animated objects. Still it should be possible to handle small deformations on instantiated objects like grass tufts blowing in the wind. In this case, the collision volume should embed the animated shape.

7. Conclusion

In this paper, we have introduced a novel tile-based approach for modeling details. Our approach relies on pre-computed *Ghost Tiles* that store a set of overlapping candidates and a graph that represents collisions between them. The originality of our method is that it provides the designer with a unified framework for creating a vast variety of entangled details such as fallen leaves, grass tufts, rocks and pebbles or tree branches. The distribution of details can be efficiently controlled through density fields which can be either obtained by interactive editing, simulations or procedural techniques. Moreover, our method allows to freely sculpt piles of entangled objects and control the relative density of the different objects.

Acknowledgments

This work is part of the project PAPAYA funded by the *Fonds National pour la Société Numérique*. The algorithms were implemented in the *Arches* framework supported by the LIRIS/CNRS. We would like to credit Lise Baron, computer artist, for providing the assets.

References

[AD05] ALSWEIS M., DEUSSEN O.: Modeling and Visualization of symmetric and asymmetric plant competition. In *Eurographics Workshop on Natural Phenomena* (2005), pp. 83–88. 2

[AD06] ALSWEIS M., DEUSSEN O.: Wang-tiles for the simulation and visualization of plant competition. In *24th Computer Graphics International Conference* (2006), pp. 1–11. 2

[BCF*05] BEHRENDT S., COLDITZ C., FRANZKE O., KOPF J., DEUSSEN O.: Realistic real-time rendering of landscapes using billboard clouds. *Computer Graphics Forum 24*, 3 (2005), 507–516. 2

[BN12] BRUNETON E., NEYRET F.: Real-time Realistic Rendering and Lighting of Forests. *Computer Graphics Forum 31*, 2 (2012), 373–382. 2

[DGA04] DESBENOIT B., GALIN E., AKKOUCHE S.: Simulating and Modeling Lichen Growth. *Computer Graphics Forum 23*, 3 (2004), 341–350. 2

[DGAG06] DESBENOIT B., GALIN E., AKKOUCHE S., GROSJEAN J.: Modeling Autumn Sceneries. In *Eurographics Short Papers* (2006), pp. 107–110. 2, 10

[DHL*98] DEUSSEN O., HANRAHAN P., LINTERMANN B., MĚCH R., PHARR M., PRUSINKIEWICZ P.: Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (1998), SIGGRAPH '98, pp. 275–286. 2

[DHM13] DU S. P., HU S. M., MARTIN R. R.: Semiregular solid texturing from 2d image exemplars. *IEEE Transactions on Visualization and Computer Graphics 19*, 3 (2013), 460–469. 2

[DN04] DECAUDIN P., NEYRET F.: Rendering forest scenes in real-time. In *Eurographics Symposium on Rendering* (2004), pp. 93–102. 2

[EVC*15] EMILIEN A., VIMONT U., CANI M.-P., POULIN P., BENES B.: WorldBrush: Interactive Example-based Synthesis of Procedural Virtual Worlds. *ACM transactions on Graphics, Proceedings of ACM SIGGRAPH 34*, 4 (2015), 106:1–106:11. 2

[FUM05] FUHRMANN A., UMLAUF E., MANTLER S.: Extreme Model Simplification for Forest Rendering. In *Eurographics Workshop on Natural Phenomena* (2005), pp. 57–67. 2

[GPG*16] GROSBELLET F., PEYTAVIE A., GUÉRIN E., GALIN E., MÉRILLOU S., BENES B.: Environmental objects for authoring procedural scenes. *Computer Graphics Forum 35*, 1 (2016), 296–308. 2, 10

[HK10] HSU S., KEYSER J.: Piles of objects. *ACM Transactions on Graphics 29*, 6 (2010), 155:1–155:6. 2

[JDR04] JAGNOW R., DORSEY J., RUSHMEIER H.: Stereological techniques for solid textures. *ACM Transactions on Graphics 23*, 3 (2004), 329–335. 2

[JZW*15] JIANG M., ZHOU Y., WANG R., SOUTHERN R., ZHANG J. J.: Blue noise sampling using an SPH-based method. *ACM Transactions on Graphics 34*, 6 (2015), 211:1–211:11. 2

[LD06] LAGAE A., DUTRÉ P.: Poisson sphere distributions. In *Vision, Modeling, and Visualization* (2006), pp. 373–379. 2

[LDG01] LEGAKIS J., DORSEY J., GORTLER S.: Feature-based cellular texturing for architectural models. In *The 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)* (2001), pp. 309–316. 2

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural modeling of buildings. *ACM Transactions on Graphics 25*, 3 (2006), 614–623. 2

[MWT11] MA C., WEI L.-Y., TONG X.: Discrete element textures. *ACM Transactions on Graphics 30*, 4 (2011), 62:1–62:10. 2, 9

[PGMG09] PEYTAVIE A., GALIN E., MERILLOU S., GROSJEAN J.: Procedural Generation of Rock Piles Using Aperiodic Tiling. *Computer Graphics Forum (Proceedings of Pacific Graphics) 28*, 7 (2009), 1801–1810. 2, 9

[SKS12] STOLPNER S., KRY P., SIDDIQI K.: Medial spheres for shape approximation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 34*, 6 (2012), 1234–1240. 4

[SM14] SAKURAI K., MIYATA K.: Modelling of non-periodic aggregates having a pile structure. *Computer Graphics Forum 33*, 1 (2014), 190–198. 2, 9

[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. 5, 8

[WSL*06] WANG R., SNYDER J., LIU X., BAO H., PENG Q., GUO B.: Variational sphere set approximation for solid objects. *The Visual Computer 22*, 9–11 (2006), 612–621. 4

[ZDL*11] ZHANG G.-X., DU S.-P., LAI Y.-K., NI T., HU S.-M.: Sketch guided solid texturing. *Graphical Models 73*, 3 (2011), 59 – 73. 2