Volumetric Spot Noise for Procedural 3D Shell Texture Synthesis

Nicolas Pavie, Guillaume Gilet, Jean-Michel Dischler, Eric Galin, Djamchid Ghazanfarpour

EG UK Computer Graphics & Visual Computing (2016)
Cagatay Turkay and Tao R. Wan (Editors)

Volumetric Spot Noise for Procedural 3D Shell Texture Synthesis

Nicolas Pavie[1], Guillaume Gilet[1], Jean-Michel Dischler[2], Eric Galin[3] and Djamchid Ghazanfarpour[1]

[1] XLIM - UMR CNRS 7252, University of Limoges, France
[2] ICUBE - UMR CNRS-UDS 7357, University of Strasbourg, France
[3] LIRIS - UMR CNRS 5202, University of Lyon, France

Abstract

In this paper, we present an extension of the Locally Controlled Spot Noise and a visualization pipeline for volumetric fuzzy details synthesis. We extend the noise model to author volumetric fuzzy details using filtered 3D quadratic kernel functions convolved with a projective non-uniform 2D distribution of impulses. We propose a new method based on order independent splatting to compute a fast view dependent approximation of shell noise at interactive rates. Our method outperforms ray marching techniques and avoids aliasing artifacts, thus improving interactive content authoring feedback. Moreover, generated surface details share the same properties as procedural noise: they extend on potentially infinite surfaces, are defined in an extremely compact way, are non-repetitive, continuous (no discrete voxel-artifacts when zooming) and independent of the definition of the underlying surface (no surface parameterization is required).

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms 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—Color, shading, shadowing, and texture

Keywords *Procedural texturing, Image synthesis, volumetric visualization, procedural noise*

1. Introduction

Artists try to achieve realism by constantly increasing the visual complexity and details of objects in virtual scene. A huge amount of small scale geometric details over surfaces needs to be defined and rendered. Despite the improvement of Graphics Processing Unit (GPU), authoring and rendering large scenes with millions of micro and meso scale details remain a challenging problem, in particular for fuzzy details over surfaces such as fur or grass. Simple color and bump map texturing techniques do not allow for realistic rendering as important parallax effects are missing and artifacts are visible at close range. Mass-instancing of small geometric primitives is also ill-suited because of storage limitations, prohibitive rendering time and important aliasing artifacts.

Shell texturing avoids most of these limitations. It consists in mapping a thick semi-transparent layer upon the surface [PBFJ05]. The corresponding volumetric texture can be represented either by explicit data arrays (voxels) or by continuous analytic functions. Analytic representations based on procedural noise [LLC*10] benefit from several important properties which makes it particularly attractive for modeling fuzzy 3D details. With a low memory overhead, it provides continuous details allowing close-ups without voxelization artifacts. Moreover, it is generic and local variations can be defined by modifying the noise parameters.

Several important problems still limit the use of procedural noise for shell texturing in computer graphics applications: Noise is hard to control and displaying shell textures requires computationally demanding direct volume rendering. *Locally Controlled Spot Noise* (LCSN) [PGDG16] proposes a spatially controlled procedural noise, with non-uniform distribution of impulses, to ease the authoring process. But this noise is parameterization dependant as it relies on a distribution in texture space. Although GPU ray casting [KW03] is tractable for voxel data, it is computationally demanding for processing procedural noise. Hundreds of evaluations per pixel might be required along a single ray, which limit the interactive visualization of volumetric models even with the fastest existing procedural noises. This paper addresses those limitations and aims at improving the authoring of fuzzy surface details using procedural noise.

We extend the *Locally Controlled Spot Noise* to model volumetric surface details composed of many similar elements. We use a projective 2D sparse convolution with filtered 3D kernel functions, represented as sums of anisotropic Gaussians with arbitrary orientations to model shell textures. To interactively visualize the results of our noise, we propose a per-pixel projective volume rendering approach to obtain a view-dependent fast approximation of accumulated densities along viewing directions. We exploit the fact that

the Locally Controlled Spot Noise is based on sparse convolution with spatially bound kernel functions, the latter being directly projected, i.e. splatted [CRZP04], thanks to their analytic representation.

Our method provides better interactivity for authoring procedural shell textures with noise in a parameterization free definition. We illustrate this through different case studies showing that such details can be edited at interactive frame rate with a very low memory footprint.

2. Related work

In this section we review previous work on the modeling of volumetric fuzzy details such as fur, hair or grass and on the rendering of shell textures.

Volumetric fuzzy details Semi-transparent volume representations are often preferred to triangles meshes or relief textures [PO06] to model thick fuzzy surface structures such as grass [BH02] or hair [LPF01].

Volumetric textures provide a generic and compact alternative to representations based on individual discrete elements: tiles can be repeated to cover arbitrarily large surfaces. Such textures can be generated either by converting explicit geometry [DN09], by using tomography scanners [ZJMB11], by using reconstruction [MK06] or by using synthesis based on multiple 2D exemplars [KFCO*07]. Two different types of 3D textures should be distinguished: solid textures, which are surface independent, and shell-textures [JMW07] which are linked to the surface geometry. We are concerned with the second category. To avoid periodicity, 3D extensions of 2D tiling techniques, like shell-Wang cubes [LEQ*07], have been proposed. The discrete nature of voxels yields sampling artifacts when the view point is close. Local variations (i.e. non-stationary surface details) are hard to manage and interactive authoring is almost impossible. Analytic function-based textures avoid these shortcomings and have additional advantages related to continuity, as discussed in [NL13]. An analytic representation based on procedural noise [EWM*98] is particularly interesting since it allows one modeling various different types of fuzzy details, including fur [PH89], cotton, fire, etc. in a extremely compact way without visual repetition. Sparse convolution noises like Gaussian- [Lew89], Spot- [vW91] and Gabor- [LLDD09] noise furthermore allow to get rid of surface parameterization by using direct kernel projection [LHN05]. The recently introduced Locally controlled Spot Noise [PGDG16] extends Spot noise to improve the spatial control and to produce structured patterns in texture space, but their distribution depends on parameterization. We extend this work by proposing a method for distortion-free shell-texturing without the need for parameterization.

Volume rendering of shell textures and GPU-based volume rendering have been widely explored. A complete overview is beyond the scope of the paper. Projection of slices orthogonal to the viewing direction [WE98] has been used for a long time. It is also suitable for real-time shell-texturing provided the slices can be easily computed, i.e. the intersection of the shell with a plane, as done in [DN09]. But slice-based volume rendering is known to produce numerous visual artifacts [KW03]. Ray marching [PH89] provides better quality results. Thanks to GPU improvements, it can be now computed in real-time. Ray marching consists in casting rays throughout the shell (one ray per pixel) and sampling each ray. Because a ray might exit and then re-enter the shell, depth peeling [NK03] must be applied. When the shell is defined procedurally, especially using noise, computations rapidly become prohibitive because of the high cost of noise evaluations. Elliptical Weighted Average (EWA) volume splatting [ZPvBG01] avoids depth peeling and permits high quality anti-aliasing. It requires time consuming back-to-front ordering however, and introduces a rasterization overhead, negatively impacting frame rate because ellipses overlap in screen space. Even adaptive techniques [CRZP04] still suffer from rasterization overhead.

Order independent (OI) volume rendering techniques [ME04, MDM10] trade realism for speed by getting rid of ordering. Inspired by these works, we propose a novel OI EWA volume splatting. It unifies their advantages and provides an efficient high speed approximation, that exploits the analytical expression of the elliptical Gaussian kernels that we use for our noise.

3. Procedural details using noise

In this section, we first recall the principles of structured texture synthesis using Locally Controlled Spot Noise.

Spot Noise [vW91, dLvL97] is a sparse convolution noise [Lew89] with a flexible kernel formulation: a uniform distribution of impulses is convolved with an arbitrary kernel spatially defined. For impulses generation on the fly, jittering and pseudo-random number generation are used. Using a Gabor filter, Gabor-noise [LLDD09] trades spectral control for spatial control. But It can only produced Gaussian patterns (i.e. fully characterized by their power spectrum). In contrast, LCSN extends the original spot noise formulation to improve spatial control and to produce structured textures. A non-uniform distribution of impulses is convolved with a kernel composed of multiple ellipsoidal Gaussian functions. The kernel formulation is given as:

$$k(\mathbf{p}) = \sum_{V_i} g_{V_i} \qquad g_V(\mathbf{p}) = Ae^{-\frac{1}{2}\mathbf{p}^T\mathbf{V}^{-1}\mathbf{p}} \tag{1}$$

$\mathbf{p}$ is a point in dimension $N+1$ with last coordinate set to 1, N refers to the evaluation space dimension. $\mathbf{V}$ is a $(N+1)$ semi-positive square matrix that describe the ellipsoid iso-contour and A is the Gaussian envelop magnitude. Still using a random distribution, non-uniformity is obtained by the use of a Kroenecker delta (δ): Impulses failing a density test between a random density value generated per impulse and a local density factor are rejected. The noise formulation is the following:

$$n_s(\mathbf{p}) = \sum_j \delta\left(\xi(\mathbf{p}_j) < d(\mathbf{p}_j)\right) |w_j(\mathbf{p}_j)| K_j(\mathbf{p}) \tag{2}$$

with $K_j(\mathbf{p}) = k_s\left((\mathbf{p}-\mathbf{p}_j)\mathbf{R}_s(\mathbf{p}_j)\mathbf{S}_s(\mathbf{p}_j)\right)$. Orientation $\mathbf{R}_s$ and scale $\mathbf{S}_s$ are related to underlying data fields. d is a scalar field and represents a probability. $||$ denotes the absolute value. ξ is a random variable selected independently of w_j. d allows to control the density of impulses in given regions.

Two points are not addressed by LCSN. Filtering is not further explored in the noise model, but the use of a sum of Gaussian function allows for analytic filtering if splatting is used for individual gaussian evaluation [ZPvBG01]. The non-uniform distribution proposed depends on parameterization as impulses are generated in texture space.

We propose to extend the kernel definition of LCSN to model shell textures, i.e. volumetric textures. We do this by defining filtered 3D kernels instead of 2D kernels and by projecting random impulse positions over surfaces. A kernel k is defined by a sum of ellipsoidal Gaussian functions with arbitrary orientation. Kernels can be modeled by editing the parameters of the ellipsoidal Gaussian functions while impulse density, kernel orientation, kernel color and kernel scale can be directly painted over the surface, for instance using octree textures [BD02] if no surface parameterization is wanted. We consider now the Gaussian formulation of [ZPvBG01] instead of the one defined in (1):

$$g_V(\mathbf{p}) = \frac{||\mathbf{V}^{-1'}||^{1/2}}{(2\pi)^{3/2}} e^{-\frac{1}{2}\mathbf{p}^T\mathbf{V}^{-1}\mathbf{p}} \tag{3}$$

Here $\mathbf{V}$ is a 4×4 matrix, such that $\mathbf{V}^{-1} = (\mathbf{M}\,\mathbf{R}\,\mathbf{S})^{-T}(\mathbf{M}\,\mathbf{R}\,\mathbf{S})^{-1}$ and $\mathbf{V}^{-1'}$ is the matrix $\mathbf{V}^{-1}$ reduced to a 3×3 matrix (i. e. last row and column encoding translations are ignored). $||\mathbf{V}^{-1'}||$ is the determinant of the matrix. $\mathbf{M}$,$\mathbf{R}$ and $\mathbf{S}$ respectively correspond to shift, rotation and scaling matrices. The iso-contour of the Gaussian is given by $\mathbf{p}^T\mathbf{V}^{-1}\mathbf{p}$, which describes an implicit quadratic surface given that $\mathbf{V}$ is a semi-positive quadratic matrix ($\mathbf{p}$ is a 3D point in homogeneous coordinates). From (3), we can obtain a reconstruction filter using the Jacobian matrix of the pixel into kernel space for each Gaussian function, as in [RPZ02]. The kernel formulation becomes:

$$k(\mathbf{p}) = \sum_{V_i} \rho'_{k,V_i} \qquad \rho'_{k,V}(\mathbf{p}) = g_{V+J_k^{-1}J_k^{-1^T}}(\mathbf{p}) \tag{4}$$

J_k is the Jacobian projected in kernel space and $\rho'_{V,k}$ is the reconstruction filter for a Gaussian g_V in kernel space.

This 3D kernel definition is versatile and allows us to model very thin surfaces such as grass blades by using a single truncated flat Gaussian kernel. Complex semi-transparent structures like fur or cotton can be obtained by using multiple Gaussians as demonstrated in shell textures (Section 5).

3.1. On-the-fly impulse generation over surfaces

To achieve on-the-fly generation of impulses over an arbitrary surface, the latter is partitioned using a regular 3D grid. The size of each grid cell is fixed according to the size of the kernels, so that no kernel spans more than two cells (i.e. a kernel with center in one cell can only contribute to directly neighboring cells). Similar to [LHN05] and [LLDD09], our cells are voxels straddling the surface. Kernels are distributed in three-dimensional space using a PRNG initialized inside the voxel. Randomly generated 3D impulses are then projected on a plan approximating the overlapped surface (see figure 1). To save space, an octree data structure is used. The leaves of the octree are voxels, on which we store a vector $\mathbf{N}(u,v)$ to the closest surface point $P(u,v)$, as well as a tangent

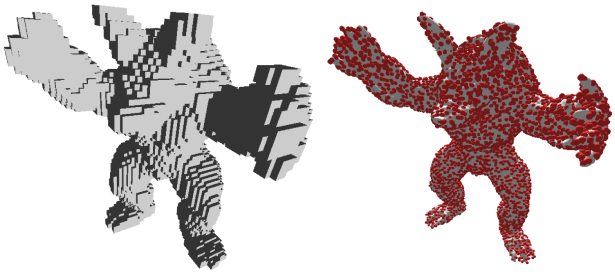

Figure 1: *Generating impulse distributions over arbitrary surfaces using an octree (left), with leaves of equally-sized voxels. distributions.*

$T(u,v)$. $-\mathbf{N}/||\mathbf{N}||$ is the normal on P. The kernel K is then oriented according to these surface properties (see figure 2). Note that the impulse position might also be shifted away from the surface along the normal, which is denoted as $P(u,v,h)$ in the figure 2.

The number of impulses in one cell is randomly defined according to the local impulse density property of the cell. That is, each cell contains the density information d. As pointed out in [LLDD09], such set-up free surface noise definition assumes that texture detail is small compared to geometric detail, which is a common assumption in texturing.

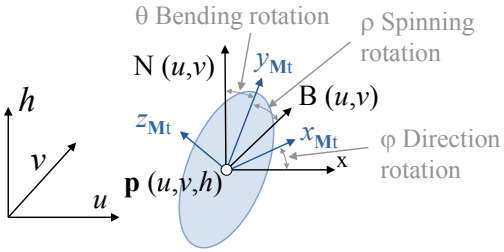

Figure 2: *Transformed kernel evaluation space in shell space* (u,v,h).

4. Shell spot noise evaluation on GPU

Shell spot noise defines volumetric color and transparency fields, n_{rgb} and n_a. It requires volume rendering. The volume rendering equation describes the light intensity L that reaches the center of projection along a ray of length l. For interactive display, a common approximation consists in neglecting scattering effects, thus considering only emission and absorption. In this case, the classical formulation is:

$$L = \int_0^l (\mathbf{E}\circ n)(\mathbf{p}(s)) e^{-\mathbf{A}_0(s)} \delta s \qquad \mathbf{A}_0(s) = \int_0^s n_a(\mathbf{p}(t))\delta t \tag{5}$$

where the exponential term can be interpreted as an attenuation factor, and $(\mathbf{E}\circ n)(\mathbf{p}(s))$ the term describing both the emission and extinction in the direction of the ray at point $\mathbf{p}(s)$. When no lighting is considered this term corresponds in our case to:

$$(\mathbf{E}\circ n)(\mathbf{p}(s)) = n_{rgb}(\mathbf{p}(s))\, n_a(\mathbf{p}(s)) \tag{6}$$

where $n_{rgb}(\mathbf{p}(s))$ is the color of the noise function and $n_a(\mathbf{p}(s))$ its transparency at point $\mathbf{p}(s)$. For clarity, we will use from now on the function $\mathbf{E}$ to denote the multiplication of the color component by the extinction component of an expression. When shading is used, the emission term n_{rgb} has to be modified by multiplying with an additional shading factor usually computed by the dot product between the gradient of $n_a(\mathbf{p}(s))$ and the light direction [GAMD10]. For the sake of simplicity, we present our formula without shading.

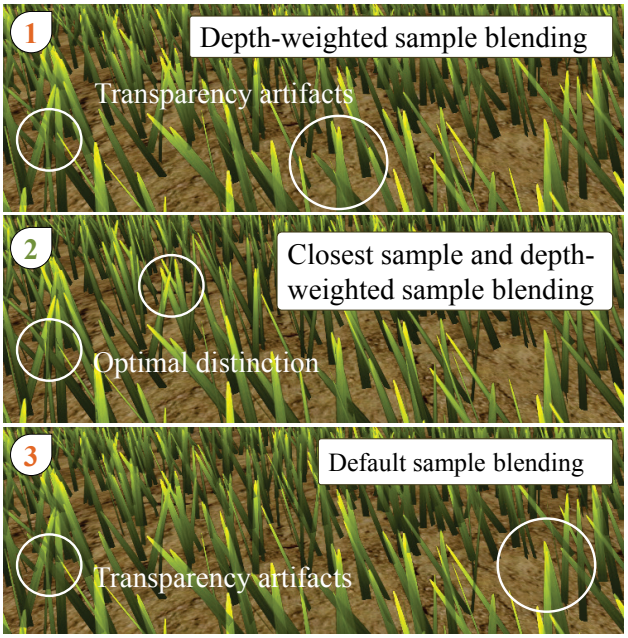

Figure 3: *Impact of OIT. 1st row: Using a depth-related weight function as described in [MB13]. 2nd row: Blending the closest contributive sample with the depth-weighted samples accumulator. 3rd row: No OIT (all kernels are blended together, which is equivalent to the use of a constant function T).*

Evaluating shell spot noise is optimized by using a volumetric regular grid, so that the search of closest kernels is accelerated. In addition, a simple extruded version of the underlying surface can be rasterized to produce viewing rays for each corresponding fragment covered by our shell. By considering that a ray traverses individual cells, we can write the previous equation by decomposing the domain into a succession of disjoint cells $\mathscr{C}_i$. By further introducing the term $1-\alpha_j$, commonly used in volume rendering to approximate the corresponding exponential attenuation, Equation 5 becomes:

$$L \approx \sum_{i=0}^{N-1} L_{\mathscr{C}_i} \prod_{j=0}^{i-1} (1-\alpha_j) \tag{7}$$

with N the number of cells traversed by the ray. $L_{\mathscr{C}_i}$ is the light scattered in the direction of the ray by cell $\mathscr{C}_i$. It is expressed as:

$$L_{\mathscr{C}_i} = \int_{l_i}^{l_{i+1}} (\mathbf{E} \circ n)(\mathbf{p}(s)) e^{-\mathbf{A}_{l_i}(s)} \delta s \tag{8}$$

For the next part, we only consider the case of uniform distribution (i.e. with $\delta = 1$). By substituting (2) into (8) and by using the standard 0-order approximation [HMDM08] of the attenuation factor, the contribution of one cell $\mathscr{C}_i$ can be rewritten as:

$$L_{\mathscr{C}_i} \approx \sum_{u=0}^{U} \left(\mathbf{E} \circ \sum_{\mathbf{p}_j \in \mathscr{C}_i^*} w_j K(\mathbf{p}(u) - \mathbf{p}_j) \right) (\mathbf{p}(u)) \prod_{v=0}^{u-1} (1-\alpha_v) \tag{9}$$

where U denotes the number of samples taken along the ray traversing the cell and $\mathbf{p}_j \in \mathscr{C}_i^*$ the impulses *contributing* to the cell $\mathscr{C}_i$ (i.e. the impulses of the cell and those of the directly neighboring cells).

Computing $L_{\mathscr{C}_i}$ using a straightforward ray marching algorithm introduces an important computational overhead. There is a double sum on samples and on kernels. Hence, each cell kernel (itself composed of several elliptical Gaussians) is evaluated multiple times (i.e. at each ray sample). Moreover, due to the Nyquist limit, in the extreme case of perfectly flat Gaussian kernels, as for grass blades, the step between successive samples must be infinitely small in order to achieve an accurate rendering.

EWA volume splatting [ZPvBG01] provides an alternative computation scheme by relying on the rasterization of individual kernels K, thus getting rid of sampling along rays: each kernel becomes evaluated only once. In practice, the Gaussians composing the kernels are first sliced into several flat Gaussians facing the camera and then rasterized directly into screen space. But such a technique is suitable for interactive rendering with modern GPUs only if few Gaussians are rendered or if correct attenuation is not mandatory: volume splatting algorithms require the flat elliptical Gaussian slices to be depth-sorted to achieve correct attenuation. This drastically reduces performance in the case of large numbers of Gaussians, which is our case. Depth-peeling [NK03] can be used to some extend to accelerate sorting, but it is still too time consuming for very large numbers of Gaussians. Indeed, such method would require one depth-peeling pass for each overlapping kernel on screen. To approximate attenuation at much lower cost, we propose a novel approach: it consists in computing an unordered accumulation of depth-weighted view-dependent contributions of kernels, renormalized in a final composition pass. The contribution of each volumetric kernel is approximated per pixel by a ray-based depth-ordered slicing. Kernel slicing is performed by casting rays throughout the kernel K according to the viewing direction (i.e. slicing is implicit through ray sampling). For each individual kernel, we consider the classical ray marching approximation consisting of a discrete sum on samples d_i taken along the viewing ray:

$$\tilde{K}(\mathbf{d}, \mathbf{p}_j) \approx \sum_{u=0}^{N_s} \left(\mathbf{E} \circ w_j K(\mathbf{d}_u - \mathbf{p}_j) \right) \prod_{v=0}^{u-1} (1-\alpha_v) \tag{10}$$

$K(\mathbf{d}_u - \mathbf{p}_j)$ is the evaluation of kernel K on impulse $\mathbf{p}_j$ at a sample $\mathbf{d}_u$ inside the volume delimited by K. The number N_s of samples per kernel is directly related to the thickness of K. In practice, the bounding box of each kernel is rasterized to cast rays. The opacity and color are evaluated on samples $\mathbf{d}_u$ and accumulated until a maximum opacity value is reached or all samples have been evaluated. In case of infinitely flat Gaussian kernels, a single sample matching the intersection between the ray and the kernel is used instead of the sum.

As opposed to equation 9, the previous ray marching is limited

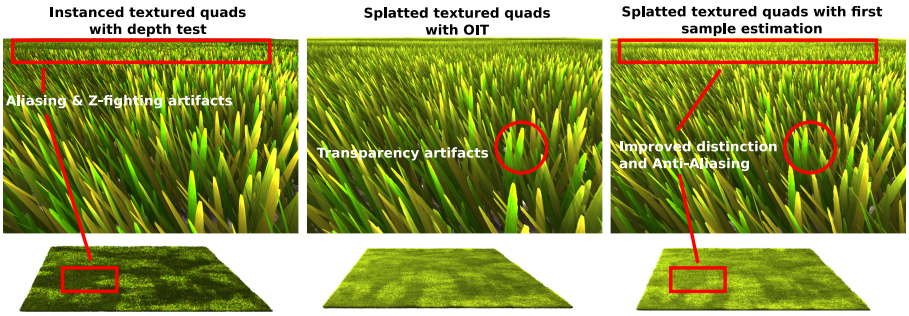

Figure 4: *Comparison of rendering techniques: the result computed by the instancing of opaque kernels with depth test (1st column) is efficient both in performance and quality of the opaque parts at close up view, but cannot handle transparency. Furthermore flickering and aliasing artifacts appears when viewed from far away. This could be solved by using a complex geometry filtering scheme. Splatting of kernels with weighted order independent transparency (2nd column) shows none of this aliasing artifacts, but introduces transparency artifacts in close up views. Our rendering pipeline (3rd column) correct this issue by using the depth information provided by the instancing technique.*

to individual kernels: it allows us to evaluate the opacity of a given kernel according to a given viewing direction $\mathbf{d}$. The contributions of all kernels must be now accumulated, without explicit depth-sorting or depth-peeling to avoid the aforementioned computation overhead. Since depth cues are crucial to get visually convincing results, we use an order independent transparency (OIT) function. The core of OIT is to define a weight, given in our case by a function $T(\mathbf{p}_j)$, related to the depth of the kernel.

Using our combination of EWA splatting with OIT, we replace the previous equation 9 by:

$$L_{\mathscr{C}_i} \approx \sum_{\mathbf{p}_j \in \mathscr{C}_i^*} T(\mathbf{p}_j)\tilde{K}(\mathbf{d},\mathbf{p}_j) \tag{11}$$

A normalization is needed to get an approximation of L and because the result of accumulating depth-weights $T(\mathbf{p}_j)$ depends on the number of samples and cells:

$$L \approx \frac{\sum_{i=0}^{N-1} L_{\mathscr{C}_i}}{\sum_{i=0}^{N-1} \sum_{\mathbf{p}_j \in \mathscr{C}_i^*} T(\mathbf{p}_j)} \tag{12}$$

$T(\mathbf{p}_j)$ can be defined in different ways. We have first experimented the simple and generic formulations of [MB13]. Using [MB13], a slightly improved distinction of kernels can be noticed (see fig. 3.1 and fig. 4 center) compared to the use of no OIT at all (see fig.3.3). The latter amounts in setting $T(\mathbf{p}_j) = 1, \forall j$. Some opaque kernels, especially the ones that are very close to each other, are not processed correctly which motivated the following original solution.

For a spot noise mostly composed of opaque kernels, visual results are significantly improved by maximizing the contribution of the closest kernel to the final color L. In any rendering pipeline, this kernel can be trivially determined by using a depth test. L is then obtained by combining the previous OIT-based EWA splatting approximation with the contribution of the closest sample:

$$L \approx \tilde{K}(\mathbf{d},\mathbf{p}_f)\alpha_f + \left(\frac{\sum_{i=0}^{N-1} L_{\mathscr{C}_i}}{\sum_{i=0}^{N-1} \sum_{\mathbf{p}_j \in \mathscr{C}_i^*} T(\mathbf{p}_j)}\right)(1-\alpha_f) \tag{13}$$

$\mathbf{p}_f$ denotes the impulse corresponding to the closest kernel, and α_f its transparency. In practice, our experiments show that results computed using this improved OIT are almost identical to standard OIT for highly transparent kernels, but show significant improvements when opaque kernels are rendered (fig. 3.2 and fig. 4 right).

5. Results

Figures 6 and 7 show examples of opaque flat kernels on 3D objects. These images are interactively rendered using a GeForce 980, an OpenGL 4.3 framework and a 1920×1080 rendering window.

Speed The performance of our rendering method is strongly related to the density and complexity of the kernels composing the shell spot noise. All interactive examples using flat Gaussian kernels run between 5 and 60+ fps, depending on the kernel density on screen and the viewpoint. As shown by figure 4, several method can be used to render our noise model. Rendering opaque kernels by geometry instancing offers the best performance (from 60fps for 800k kernels to 3 fps for 10M kernels) but introduces many aliasing artifacts (fig 4 left) as the size of the kernels decrease, due to heavy z-fighting and limited depth precision. Furthermore, such method is unable to handle semi-transparency. Aliasing artifacts can be removed and semi-transparency processed by using Volume EWA Splatting. However, as stated in the previous section, traditional EWA Splatting is not tractable in practice as depth-sorting introduces a severe performance and storage overhead. OIT offers a very fast alternative with a performance close to geometry instancing but introduces noticeable transparency artifacts in close up views (fig 4, center). Basic OIT is not precise enough to distinguish pairs of neighbouring kernels (with performances to instancing). By re-introducing some depth information using a geometry instancing pass combined with OIT EWA splatting, we are able to avoid these artifacts (fig 4, right) at the expense of some performance loss (roughly half the framerate of previous methods).

Using a 3D kernel definition based on sums of Gaussians allows us to model both thin surfaces like grass blades and more complex semi-transparent structures. Figure 5 illustrates examples of procedural shell textures obtained using multiple Gaussians. We show

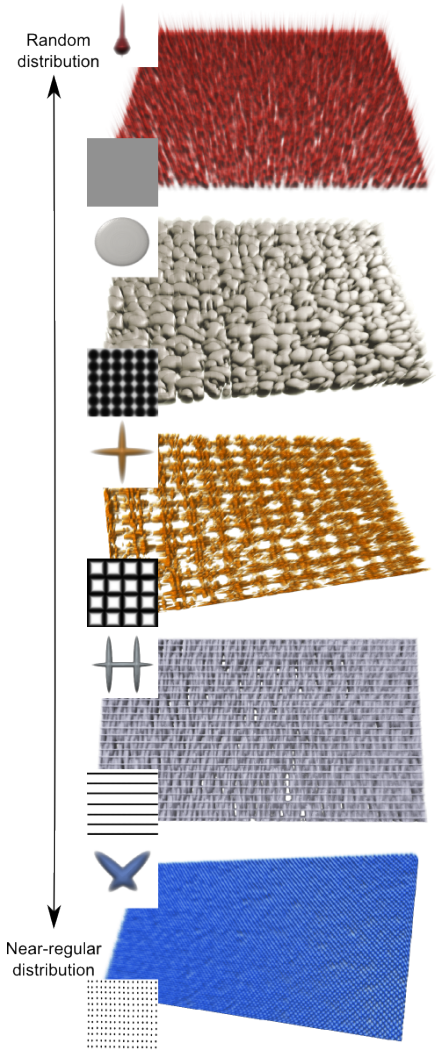

Figure 5: *Volumetric textures defined by non-uniform shell spot noise. Top left: a sum of elliptical Gaussians is used to define kernel functions k (here, up to three Gaussians). Bottom left: the field that controls impulse distributions. The darker the grey value, the more impulses. All textures are extremely compact (only the noise function, one* 4×4 *matrix per Gaussian and a density field are stored), continuous and aperiodic.*

from top to bottom different impulse distributions ranging from uniformly random to semi-regular. These textures were modeled interactively by editing the Gaussians of k (anisotropy, position, orientation) and by providing different periodic fields defined in the form of 2D intensity images (lower left images). Note that a very low number of Gaussians already allows one to model complex fuzzy textures. Even more complex shell textures are obtained by furthermore controlling the bending and spinning (rotation), size and color of kernels, as well as by using different types of kernels and different types of distributions. Finally, figure 8 shows examples of 3D volumetric kernels over 3D objects. All details are represented as a combination of semi-transparent volumetric kernels over a surface. As for previous examples, kernels can be edited interactively and controlled using low-resolution discrete maps or functions. The performance of our method in the case of semi-transparent 3D volumetric kernels is strongly related to both the number of kernels and the number of slices N_s in equation 10 (i.e. the precision of the kernel evaluation) and performs in range from 55 fps for 2.5k volumetric kernel to 1.5 fps for 300k kernels.

Memory The memory footprint of our model is extremely low, since the kernel and distribution definition itself only need a few kilobytes. In other words, the memory does not depend on the amount of rendered kernels (linked to the amount of impulses), but only on the definition of the parameters of the kernels. Parameters governing the kernel orientations, distributions, densities and scales can be defined in the form of painted low-resolution textures or they can be procedurally computed. Figure 6 illustrates the visual effect of interactive user control over noise. The noise can be edited and animated interactively by modifying density and orientation of kernels, This is related to the fact that all parameters are evaluated on-the-fly, without any need of pre-computation (unlike voxel based pre-filtering techniques). The parameter modification can be expressed through a function such as wind or a combination of low resolution textures. The latter gives the user an efficient way to control kernels and facilitates authoring of local features.

As shown in figure 7, our method allows close zooms and the complex micro-geometry becomes more visible (individual hairs) with optimized distinction. At far distance (top) individual hairs are rendered with no aliasing and improved distinction, thanks to the EWA volume splatting with OIT.

6. Conclusion

We have presented an extension of Locally Controlled Random Spot Noise for interactive authoring of volumetric shell textures. This extension can produce a wide range of procedural 3D patterns from near-regular to stochastic for a very low memory footprint. The hybrid rendering pipeline can render patterns at interactive frame rate. Interactivity is guaranteed even for millions of kernels, thus significantly improving the authoring and design of such textures.

A promising extension would consist in enhancing the performances by evaluating our continuous noise model function using a multi-scale rendering scheme. This would enable us to adapt the pattern evaluation to the viewing conditions (such as tessellation/instancing for close-ups and statistically- based filtering approximation for far distance views) to provide a method targeted at real-time realistic rendering of volumetric procedural shell textures

References

[BD02] BENSON D., DAVIS J.: Octree textures. *ACM Trans. Graph. 21*, 3 (July 2002), 785–790. URL: http://doi.acm.org/10.1145/566654.566652, doi:10.1145/566654.566652. 3

[BH02] BAKAY B., HEIDRICH W.: Real-time animated grass. In *Proceedings of Eurographics (short paper)* (2002). 2

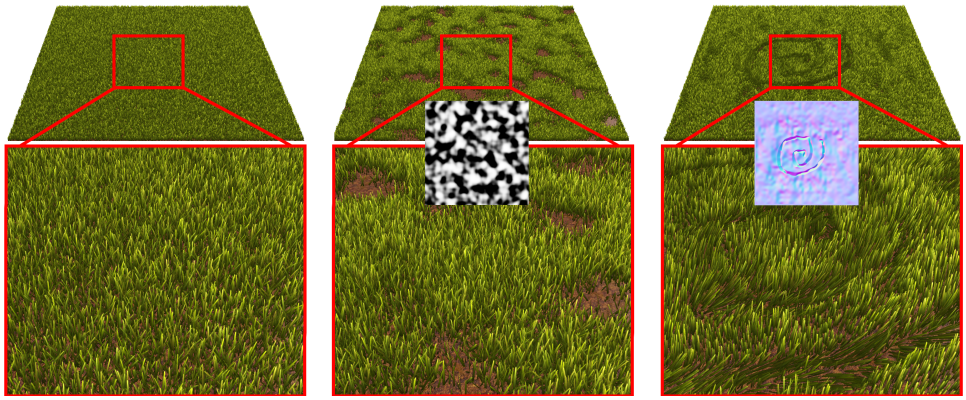

Figure 6: *(left) uniform density, (middle) user controls density, (right) user controls orientation. In all cases, control maps can be painted interactively.*

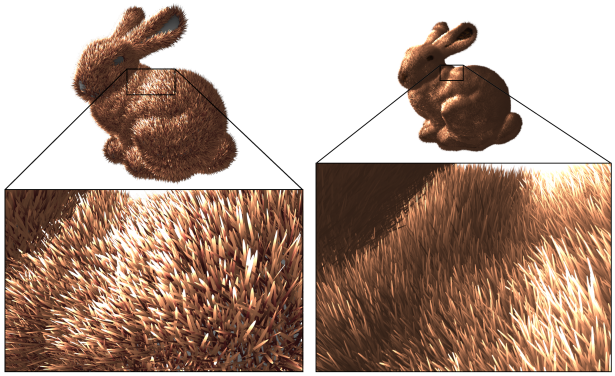

Figure 7: *The appearance of these models are defined using* $\approx$ $400k$ *small kernels for the left bunny (10 fps) and* $\approx$ $1.6M$ *small kernels for the right bunny (3 fps). Each of these kernels can be interactively edited.*

[CRZP04] CHEN W., REN L., ZWICKER M., PFISTER H.: Hardware-accelerated adaptive ewa volume splatting. In *Proceedings of the Conference on Visualization '04* (Washington, DC, USA, 2004), VIS '04, IEEE Computer Society, pp. 67–74. URL: http://dx.doi.org/10.1109/VISUAL.2004.38, doi:10.1109/VISUAL.2004.38. 2

[dLvL97] DE LEEUW W., VAN LIERE R.: Divide and conquer spot noise. In *Proceedings of the 1997 ACM/IEEE Conference on Supercomputing* (New York, NY, USA, 1997), SC '97, ACM, pp. 1–13. URL: http://doi.acm.org/10.1145/509593.509612, doi:10.1145/509593.509612. 2

[DN09] DECAUDIN P., NEYRET F.: Volumetric billboards. *Computer Graphics Forum 28*, 8 (2009), 2079–2089. URL: www.antisphere.com/Research/VolBlbCGF09.php. 2

[EWM*98] EBERT D. S., WORLEY S., MUSGRAVE F. K., PEACHEY D., PERLIN K., MUSGRAVE K. F.: *Texturing and Modeling : A Procedural Approach*, 2nd ed. Academic Press, Inc., Orlando, FL, USA, 1998. 2

[GAMD10] GUETAT A., ANCEL A., MARCHESIN S., DISCHLER J.-M.: Pre-integrated volume rendering with non-linear gradient interpolation. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (Nov 2010), 1487–1494. doi:10.1109/TVCG.2010.187. 4

[HMDM08] HAJJAR J.-F. E., MARCHESIN S., DISCHLER J.-M., MONGENET C.: Second order pre-integrated volume rendering. In *Visualization Symposium, 2008. PacificVIS '08. IEEE Pacific* (2008), pp. 9–16. 4

[JMW07] JESCHKE S., MANTLER S., WIMMER M.: Interactive smooth and curved shell mapping. In *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)* (6 2007), Kautz J., Pattanaik S., (Eds.), Eurographics, Eurographics Association, pp. 351–360. 2

[KFCO*07] KOPF J., FU C.-W., COHEN-OR D., DEUSSEN O., LISCHINSKI D., WONG T.-T.: Solid texture synthesis from 2d exemplars. *ACM Trans. Graph. 26*, 3 (July 2007). URL: http://doi.acm.org/10.1145/1276377.1276380, doi:10.1145/1276377.1276380. 2

[KW03] KRÜGER J., WESTERMANN R.: Acceleration Techniques for GPU-based Volume Rendering. In *Proceedings IEEE Visualization 2003* (2003). 1, 2

[LEQ*07] LU A., EBERT D. S., QIAO W., KRAUS M., MORA B.: Volume illustration using wang cubes. *ACM Transactions on Graphics (TOG) 26*, 2 (2007), 11. 2

[Lew89] LEWIS J. P.: Algorithms for solid noise synthesis. *SIGGRAPH Comput. Graph. 23*, 3 (July 1989), 263–270. URL: http://doi.acm.org/10.1145/74334.74360, doi:10.1145/74334.74360. 2

[LHN05] LEFEBVRE S., HORNUS S., NEYRET F.: Texture Sprites: Texture Elements Splatted on Surfaces. In *Symposium on Interactive 3D Graphics and Games* (Washington, États-Unis, 2005), ACM SIGGRAPH, ACM Press. URL: http://hal.inria.fr/inria-00510158. 2, 3

[LLC*10] LAGAE A., LEFEBVRE S., COOK R., DEROSE T., DRETTAKIS G., EBERT D., LEWIS J., PERLIN K., ZWICKER M.: A survey of procedural noise functions. *Computer Graphics Forum 29*, 8 (December 2010), 2579–2600. doi:10.1111/j.1467-8659.2010.01827.x. 1

[LLDD09] LAGAE A., LEFEBVRE S., DRETTAKIS G., DUTRÉ P.: Procedural noise using sparse gabor convolution. In *ACM SIGGRAPH 2009 papers* (New York, NY, USA, 2009), SIGGRAPH '09, ACM, pp. 54:1–54:10. URL: http://doi.acm.org/10.1145/1576246.1531360, doi:10.1145/1576246.1531360. 2, 3

[LPF01] LENGYEL J., PRAUN E., FINKELSTEIN A.: Real-time fur over arbitrary surfaces. In *2001 ACM Symposium on Interactive 3D Graphics* (2001), pp. 227–232. 2

[MB13] MCGUIRE M., BAVOIL L.: Weighted blended order-independent transparency. *Journal of Computer Graphics Techniques*

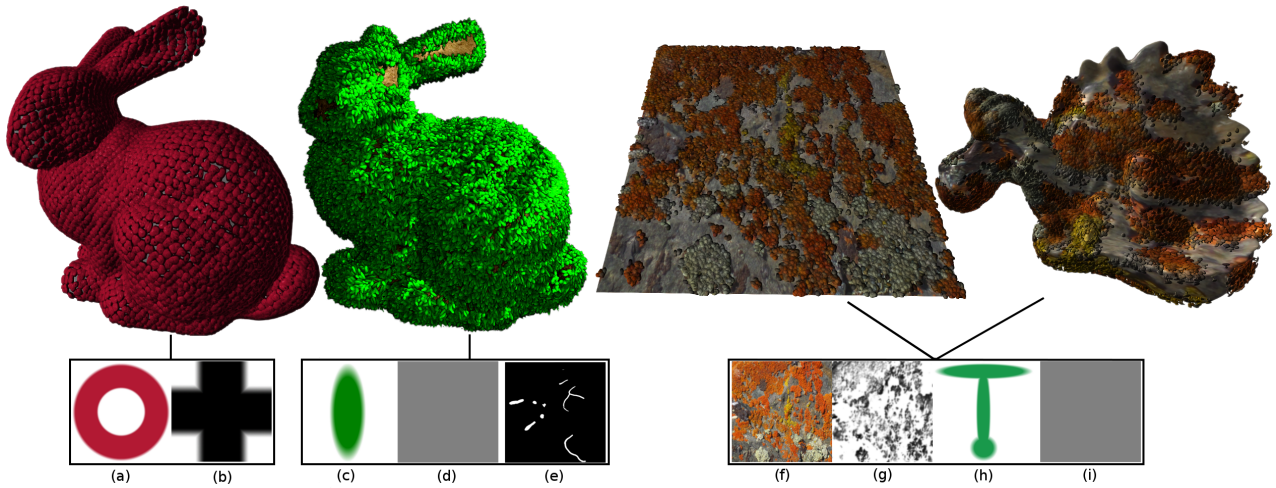

Figure 8: *Volumetric patterns applied on several model. Pattern applied on the first bunny from the left uses the "ring" kernel profile (a) composed of 2 Gaussians and a semi regular distribution profile (b). The second bunny uses a single Gaussian kernel profile (c) with a random distribution profile (d), and a density map (e) is added to limit distribution over specific area. Both bunnies use* $\approx 50k$ *impulses and run at* ≈ 2 *frames per second. The lichen pattern applied on the dragon and the plan is created using a color map (f), a density map (g), a kernel composed of 3 Gaussians (h) and a random distribution (i). It uses* $\approx 400k$ *impulses runs at* ≈ 1.5 *frames per second. All results were rendered using 32 samples per ray per evaluated kernel.*

(JCGT) 2, 2 (December 2013), 122–141. URL: http://jcgt.org/published/0002/02/09/. 4, 5

[MDM10] MARCHESIN S., DISCHLER J.-M., MONGENET C.: Per-pixel opacity modulation for feature enhancement in volume rendering. *Visualization and Computer Graphics, IEEE Transactions on 16*, 4 (July 2010), 560–570. doi:10.1109/TVCG.2010.30. 2

[ME04] MORA B., EVERT D. S.: Instant volumetric understanding with order-independent volume rendering. *Computer Graphics Forum 23*, 3 (2004), 489–497. URL: http://dx.doi.org/10.1111/j.1467-8659.2004.00780.x, doi:10.1111/j.1467-8659.2004.00780.x. 2

[MK06] MAGDA S., KRIEGMAN D.: Reconstruction of volumetric surface textures for real-time rendering. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques* (Aire-la-Ville, Switzerland, Switzerland, 2006), EGSR 06, Eurographics Association, pp. 19–29. URL: http://dx.doi.org/10.2312/EGWR/EGSR06/019-029, doi:10.2312/EGWR/EGSR06/019-029. 2

[NK03] NAGY Z., KLEIN R.: Depth-peeling for texture-based volume rendering. In *Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on* (2003), IEEE, pp. 429–433. 2, 4

[NL13] NIESSNER M., LOOP C.: Analytic displacement mapping using hardware tessellation. *ACM Trans. Graph. 32*, 3 (July 2013). URL: http://doi.acm.org/10.1145/2487228.2487234, doi:10.1145/2487228.2487234. 2

[PBFJ05] PORUMBESCU S. D., BUDGE B., FENG L., JOY K. I.: Shell maps. *ACM SIGGRAPH 2005 Papers* (2005), 626–633. URL: http://doi.acm.org/10.1145/1186822.1073239, doi:10.1145/1186822.1073239. 1

[PGDG16] PAVIE N., GILET G., DISCHLER J.-M., GHAZANFARPOUR D.: Procedural texture synthesis by locally controlled spot noise. In *Proceedings of WSCG* (August 2016). To appear. URL: http://wscg.zcu.cz/wscg2016/full/F67-full.pdf. 1, 2

[PH89] PERLIN K., HOFFERT E. M.: Hypertexture. *SIGGRAPH Comput. Graph. 23*, 3 (July 1989), 253–262. URL: http://doi.acm.org/10.1145/74334.74359, doi:10.1145/74334.74359. 2

[PO06] POLICARPO F., OLIVEIRA M. M.: Relief mapping of non-height-field surface details. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2006), I3D '06, ACM, pp. 55–62. URL: http://doi.acm.org/10.1145/1111411.1111422, doi:10.1145/1111411.1111422. 2

[RPZ02] REN L., PFISTER H., ZWICKER M.: Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Computer Graphics Forum (Eurographics 2002)* (Sept. 2002), pp. 461–470. 3

[vW91] VAN WIJK J. J.: Spot noise texture synthesis for data visualization. *SIGGRAPH Comput. Graph. 25*, 4 (July 1991), 309–318. URL: http://doi.acm.org/10.1145/127719.122751, doi:10.1145/127719.122751. 2

[WE98] WESTERMANN R., ERTL T.: Efficiently using graphics hardware in volume rendering applications. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 169–177. URL: http://doi.acm.org/10.1145/280814.280860, doi:10.1145/280814.280860. 2

[ZJMB11] ZHAO S., JAKOB W., MARSCHNER S., BALA K.: Building volumetric appearance models of fabric using micro ct imaging. *ACM Trans. Graph. 30*, 4 (July 2011), 44:1–44:10. URL: http://doi.acm.org/10.1145/2010324.1964939, doi:10.1145/2010324.1964939. 2

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Ewa volume splatting. In *Visualization, 2001. VIS '01. Proceedings* (Oct 2001), pp. 29–538. doi:10.1109/VISUAL.2001.964490. 2, 3, 4