COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds (2016)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cav.1719

RESEARCH ARTICLE

Realistic modeling of porous materials

Rodrigo Baravalle[1,2]*, Leonardo Scandolo[2,5], Claudio Delrieux[3], Cristian García Bauza[4] and Elmar Eisemann[5]

[1] CIFASIS-CONICET, Ocampo y Esmeralda, S2000EZP Rosario, Argentina
[2] Universidad Nacional de Rosario, Rosario, Argentina
[3] DIEC, Universidad Nacional del Sur, Bahia Blanca, Argentina
[4] CONICET; PLADEMA, Universidad Nacional del Centro de la Pcia. de Bs. As., Tandil, Argentina
[5] Delft University of Technology, Delft, The Netherlands

ABSTRACT

Photorealistic modeling and rendering of materials with complex internal mesostructure is a hard challenge in Computer Graphics. In particular, macroscopic porous materials consist of complex translucent substances that exhibit different details and light interaction at several different scales. State-of-the-art techniques for modeling porous materials manage the material either as a surface and set up complex capture procedures or as a volume by employing different instances of procedural noise models for its representation. While the surface solution achieves several desired material properties, it still presents drawbacks in practical applications—high computational costs, complex capture procedures, and poor image variability, among others. Volumetric solutions are more flexible, but the final structure and appearance are difficult to control. To overcome these drawbacks, we propose an algorithm for the procedural generation of porous materials. The method is based on an artistic and physically inspired simulation of the growth of self-avoiding bubbles inside a volume, by means of dynamical systems. The patterns induced by the bubbles can be easily and intuitively controlled. The bubbles adapt to any given shape and have convincing global and local fluid-like patterns as seen in bread and sponges. Our method generates 3D textures that adequately represent porous materials, which can be used as input for creating realistic renderings of different porous objects. As a case study, we present the results of using these 3D textures as input to a direct volume renderer and show that they compare favorably with standard 3D texture synthesis methods. Copyright © 2016 John Wiley & Sons, Ltd.

KEYWORDS
porous materials; dynamical systems; photorealism

***Correspondence**
Rodrigo Baravalle, CIFASIS-CONICET, Ocampo y Esmeralda, S2000EZP Rosario, Argentina.
E-mail: baravalle@cifasis-conicet.gov.ar

1. INTRODUCTION

The overwhelming variety of materials and complex light transport phenomena they induce have been a central research topic in computer graphics for decades. Physics-based modeling may achieve realistic light transport phenomena simulations, and, for this reason, feasible computational approximations are a mainstream topic in rendering research. In complex materials, geometric modeling undergoes a similar research strategy, starting from a physics-based understanding of the material that leads to an adequate computational approximation. Realistic material rendering should take geometrical modeling and light interaction simultaneously into consideration.

The choice of a geometrical model depends on the material and the intended representation scale, which, in turn, highly influences the design of a rendering algorithm. In particular, the main approaches can be categorized as either surface or volume modeling. Mathematical models of the light transport phenomena in these two approaches have been proposed by J. Kajiya; the rendering equation [1] for modeling the geometrical optic behavior in surfaces and the volume rendering equation [2] for the representation of scattering phenomena in volume densities.

Typically, a surface representation is preferred for macroscopically homogeneous materials (metals, plastics, and similar materials) [3]. This method models microscopic surface details using statistical assumptions. If mesoscopic details are modeled (as required for instance

in wood or bricks) [4], a usual technique is to precompute these details in textures that can be mapped onto surfaces. Locally flat surfaces are fast to render, but surface rendering imposes limitations to the features that can be realistically modeled, in particular the ones that arise because of the mesoscopic structure and the related light transport phenomena in the materials.

Bread crumbs, sponges, stones, and other porous materials are extremely complex to model and render because of their intricate geometry and the light transport phenomena arising within them. In this particular case, the naïve surface approach flatly fails, because mesoscopic details, clearly visible on their surfaces, are essential for a faithful representation of the material. More sophisticated surface modeling techniques, such as bidirectional texture functions [5], do not always handle light transport phenomena adequately. To overcome some of these limitations, a complex material model was proposed [6]. This method produces more realistic results, but the capturing and rendering procedures are extremely complex, and the geometric modeling is somewhat inflexible (e.g., the surface representation does not allow arbitrary cuts).

There is a vast body of literature on volumetric representation of materials (e.g., smoke and clouds) [7,8]. Volumes, while usually computationally more expensive to render, do not have the aforementioned drawbacks that surfaces have [9]. The characteristics of surface and volume rendering imply that there is a trade-off between geometrical representations for complex materials. Where simplicity and real-time rendering are required, simplified geometric models and surface-based rendering approaches are used, and when photorealism is the strongest requirement, complex geometric models and volume-based rendering are better suited for the task. However, the development of Graphics Processing Unit (GPU) has an influence on this trade-off, since recent hardware allows us to rely on volume representations to be rendered at interactive rates, as we will show in this work.

In [10], the authors presented a bread crumb modeling algorithm that was able to emulate most of the mesoscopic features of the material. However, the procedure was independent from the external shape. In other words, the mesoscopic structure (the gas bubbles in the crumb) does not follow the boundaries of the object and does not have a well-defined center. In [11], the procedural model was extended to adequately accommodate the mesoscopic structure of the material with respect to the original dough shape (before bread baking), and other enhancements were also proposed, including a simple but effective model of the crust formation process. The phenomenological aspects of the model were validated using a multifractal characterization. In both papers, the focus was on a realistic model of bread crumb, without concerning on computational limitations.

In this work, we propose a non-physically based method, obtained as a simplified relaxation of physically based models, that is nevertheless able to represent and render porous materials with adequate accuracy. This simplification extends the modeling capabilities to include arbitrary geometries into the process.

Instead of computing voxel values using algebraic functions [12], the geometry is modeled using particle systems [13] in a 3D cube that evolves by means of dynamical systems (DSs) [14]. This allows us to emulate several processes that achieve mesostructural distributions with controllable geometric and statistical properties.

To demonstrate the capabilities of our method, we use its output in a direct volume rendering (DVR) [15–17] application. We show that using our method results in realistic-looking porous objects. Furthermore, the final shape of the rendered object can be modified interactively, allowing, among other things, to perform arbitrary cuts and slices in real time. Finally, we compare our results with images generated with standard texture synthesis algorithms used for creating porous or cell-based structures.

This work is organized as follows. First, we present an overview of the underlying ideas used in our modeling, including particle systems and DSs. Then, we present a detailed description of the modeling workflow. Following that, we discuss the results of the modeling step and show that they can be used to create realistic renderings with DVR. Finally, we summarize the conclusions, as well as the future work enabled by our results.

2. PREVIOUS WORK

The topic of photorealistic material modeling and rendering recently attracted a significant amount of research. Much work has focused on frequently appearing complex materials such as water [18,19], skin [20], metals, and plastics [21]. Still, the computer graphics community has struggled to emulate the appearance of other materials adequately, for instance, baked materials (e.g., pizza and cookies), but because of their complex geometry and the complex underlying light transport phenomena, this continues to be an open problem [22]. Until recently, the computational cost of rendering physically based models of these materials was impractical when real time was a requirement. However, the remarkable growth in computing power due to the massively parallel design of modern graphics cards [23,24] is enabling the simulation of complex light transport phenomena at acceptable refresh rates.

The presence of mesostructures (bubbles and alveoli of complex shapes) makes porous materials quasi-homogeneous [6]. For this reason, adequate surface representations of these materials are not possible. Typical techniques such as bidirectional reflectance distribution functions [25] and bidirectional surface scattering reflectance distribution functions [26] are not satisfactory. A material model [6] solves these limitations, but the associated drawbacks (complex capture procedure involved, computational costs, poor geometry variability) makes the method far from practical.

DOI: 10.1002/cav

Simulating acceptable materials with rich and complex mesostructure represents an additional challenge. Porous structures are the result of complex mechanisms involving physical deformations, heat and mass transport (if baked), and several chemical reactions. In the past, there have been few attempts to synthesize or simulate the resulting geometries. One approach uses artistic considerations [27], but details were not published because of copyright issues. Recent studies employ phenomenological considerations in actual specific materials, for example, bread, but the modeling procedures produce fixed geometries (i.e., non-procedural) [28]. This is a serious drawback because of several reasons. A fixed geometry cannot be used to obtain different material types easily, because it requires a capture procedure for each of them. In addition, the mesostructural distribution (bubbles, alveoli, etc.) can be neither controlled nor changed. Finally, this method of modeling does not allow an interaction with the geometry of the material (i.e., slicing).

Nonetheless, research on physically inspired mathematical models of materials such as bread crumbs is not uncommon in the food industry-related literature. These models aim to adequately simulate heat and mass transfer in dough during baking, as well as other properties. Recent results suggest that 1D models could suffice, for instance, modeling the geometry as an infinite cylinder, or assuming only one radial coordinate [29,30]. These and other results in the food industry have some significance in bread crumb modeling and rendering and may be used as a further basis for computational baking models.

3. MODELING POROUS MATERIALS

In this section, we provide the required background and the algorithms involved in the modeling workflow.

3.1. Particle Systems

Particle systems [13] deal with phenomena that have no well-defined geometric interface, like water [18], smoke [31], and fireworks. They are composed of entities called *particles* whose properties evolve over time. For instance, the method can easily model fireworks by simply defining a common space position for all particles and, after each time step, modifying each particle position following a slightly different trajectory for each particle. Particle systems are used to model fire and other effects by modifying properties such as color, size, and direction and they can affect each other. We emulate a porous formation process using a self-avoiding particle system in 3D space, combined with a system of differential equations to control the particle growth. This algorithm produces a 3D texture representing the geometry of the material.

3.2. Proposed Modeling Algorithm

This algorithm produces the underlying mesoscopic geometry description. Instead of delivering the color of a particular space position, it will generate a volumetric texture composed of 0's and 1's (0: air, 1: mass). The system consists of a set of particles P,

$$P = \{p_1, \ldots, p_n\}, n \in \mathbb{N} \tag{1}$$

a lattice $L_{N\times N\times N}, N \in \mathbb{N}$ (initially $L_{xyz} = 1$) of mass and air, and a lattice $L^2_{N\times N\times N}$, (initially $L^2_{xyz} = -1$), of positions and particle ownership (i if the lattice element belongs to the interior or contour - immediate lattice neighbors to interior positions - of the particle i). Each element in P has the following properties:

$$p_i = \{O_i, C_i\}, 1 \leq i \leq n \tag{2}$$

where

- $O_i = \{o_1, \ldots, o_{n_i}\}$: (occupied) vector (set) of occupied positions by the particle in L.
- $C_i = \{c_1, \ldots, c_{m_i}\}$: (contour) vector (set) of positions representing the particle *contour* in L.

The vector O represents the positions affected by the particle, and the contour C warrant avoidance with other particles. We describe the procedure in more detail in Algorithm 1. When $t = 0$, a set of particles takes random lattice positions. Each particle adds its position to O and the surrounding positions to C. In each iteration, each particle chooses a position on its contour. If the position lies in the contour of any other particle ($L^2_{position} <> i$ and $L^2_{position} > -1$), the process discards it and selects another contour position. If that position is free ($L^2_{position} = i$ or $L^2_{position} = -1$), the particle adds the position to its O and updates its contour C and the lattices. If the contour vector is empty, the particle *dies*, because it cannot grow anymore in the simulation. Termination of the algorithm is possible at any t. The user can stop the simulation at a particular event, for instance, when the L^2 lattice is full ($L^2_{xyz} <> -1$ in any lattice position), because no progress can be made.

The method produces different structures varying the contour size (*separation* in Algorithm 1; Figure 1). Figure 1 shows 2D output examples (for better understanding) of random growing particles. The contour size determines the white region among particles (mass) (the *width* of the white area). The output of the algorithm is the lattice L. The resulting images resembles Voronoi-like patterns.

3.3. Particle Evolution Using Dynamical Systems

Human perception is very sensitive to patterns, particularly in porous structures. For instance, in bread, the eye immediately recognizes that the bubble shapes tend to follow the crust shape. This effect is due to the fact that the temperature during baking affects the shape of the bubbles [32], stretching them to follow the crust. Also, the entire structure has a fluid-like appearance. Indeed, this is

Algorithm 1 Modeling Algorithm

```
t = 0, P = []                                              ▷ time (iteration), particles
L = matrix(MxMxM).init_values(1)                           ▷ Geometry - initiated to 1 (mass)
L^2 = matrix(MxMxM).init_values(−1)                        ▷ Particle Domain
for i ∈ [1, particles_count] do                            ▷ random position in L for each particle
    x ← random(), y ← random()
    O[i] ← [[x, y]], C[i] ← []
    for v ∈ neighborhood(x, y) do
        C[i].add(v)
    end for
    P.add([O[i], C[i]])
end for
for t ∈ [0, max_time] do
    for i ∈ [1, particles_count] do
        if empty? C[i] then
            die()
        end if
        for h ∈ C[i] do
            C[i].delete(h)                                 ▷ the position was explored
            if !(L^2[h] > 0 && L^2[h]! = i && free_boundary(separation)) then
                L[h] ← 0, O[i].add(h)                      ▷ mass − > air **
                C[i].add(neighborhood(h)), L^2.set(neighborhood(h), i)
                                                           ▷ Mark positions in L^2 as i
                L^2.set(h, i)
            end if
        end for
    end for
end for
```

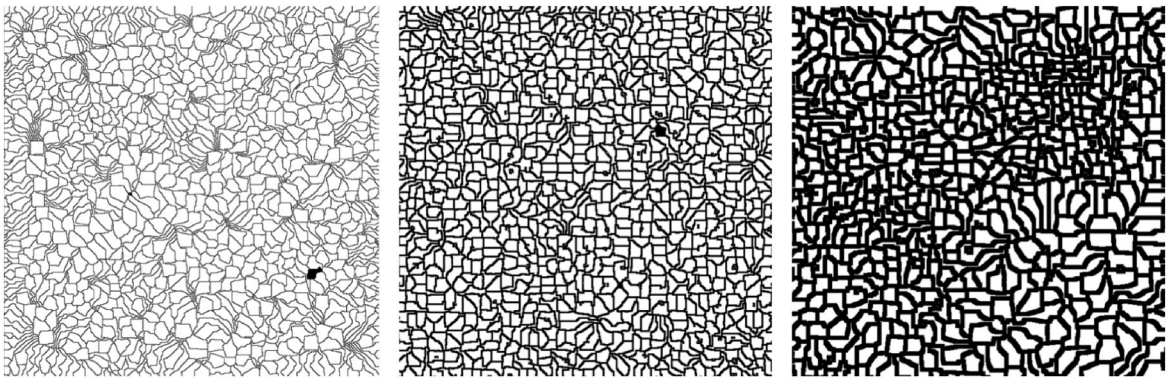

Figure 1. Modeling algorithm, different values for the separation parameter. Left: separation = 1, center: separation = 2, right: separation = 4.

the case in early stages of the baking process. At some point, the viscosity of the dough decreases, and the bubbles stop growing and merging with each other.

Dynamical systems are a mathematical framework for understanding the dynamic behavior of differential equations in dynamical processes [14]. Many dynamical phenomena can be described by defining differential equations that represent their behavior in time, simulating the evolution of the system and deriving approximate albeit useful solutions. DS have been used in several domains for generating structures that resemble natural shapes. A numeric integration of the equations starting from several initial conditions (or *seeds*) gives rise to a set of streamlines. For instance, the following equations produce Figure 2(a):

$$\begin{aligned} \dot{x} &= x^2 - y^2 + 1 \\ \dot{y} &= 2xy + 1 \end{aligned} \tag{3}$$

Figure 2(b) and (c) is an example of other equations sets, where the resulting streamlines show different geometrical patterns

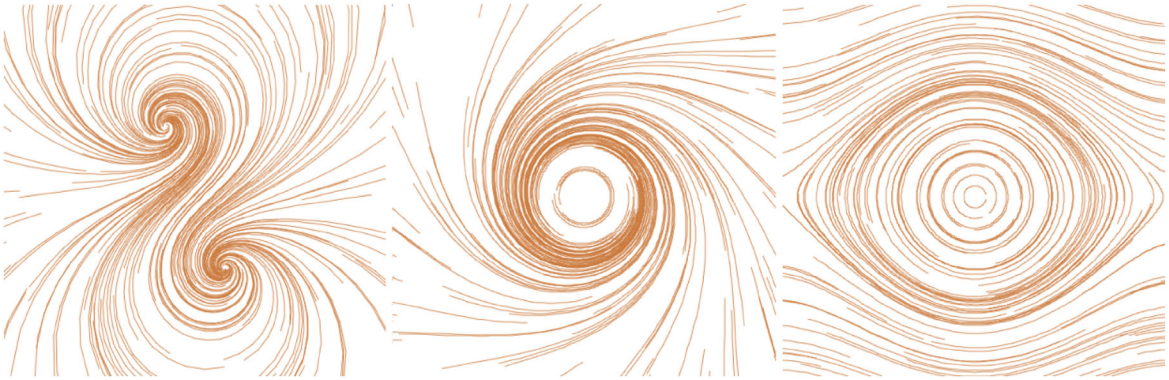

Figure 2. Examples of dynamical systems in the plane used to simulate fluid-like patterns.

A DS can be used to *drag* information along the streamlines. In particular, for our modeling algorithm, we will use a DS to modify the underlying geometry of the particle system shown previously. The application of our proposed DS will modify the system by moving the particles along the streamlines. This will alter the geometrical structure of the volume in a way that mimics the deformation of the bubbles described previously. Algorithm 2 shows the modification that we introduced to the original modeling algorithm.

Algorithm 2 Dynamical Systems Modification for the modeling algorithm

```
L[h] ← 0                                   ▷ mass − > air **
O[i].add(h)
solution ← Runge_Kutta(h)   ▷ Computes next position
neigh = neighborhood(h)
best = abs(neigh[0] − solution)
chosen = h
neigh.delete(h)
for w ∈ neigh do
    // Best neighborhood solution that approximates the system
    if abs(neigh[w] − solution) < best then
        best = abs(neigh[w] − solution)
        chosen = w
    end if
    if random() > 1 − randomness then
                                   ▷ 0 <= random() <= 1
        C[i].add(w)
    end if
end for
// The best approximation is added
C[i].add(chosen)
```

Figure 3 shows particles following the DS streamlines, where the DS is the same as in Figure 3(c). From left to right, we decrement the randomness of the trajectories, which is a parameter of our model. A randomness value of r ($0 \leq r \leq 1$) forces the particles to follow the DS trajectories with a probability of $1 - r$. The resulting patterns are now more similar to those arising in real bread and other baked materials. Different mesostructures can be produced employing a different set of equations and different parameters for the particle systems (lifetime, randomness).

3.4. System Setup

In order to represent several porous structures such as sponges, bread, cakes, and pizzas, the growing particles should adapt to arbitrary material silhouettes. Some materials, for instance, bread or bones, have elongated bubbles oriented parallel to the object boundaries, because the bubbles are stretched and pushed towards the boundaries during baking or growing. In other porous structures as sponges, the cells seem to be homogeneously orientated. Typical bones show mixed porous and non-porous structures, and regions where the trabecular tissue is predominantly oriented in some directions, depending on bone health, age, and other factors. Our modeling algorithm can cope with all these situations by defining several DS with different behaviors for different regions.

The procedure starts by voxelizing an arbitrary input geometry, defining a region of interest where the particle system can grow (setting 1 if the voxel belongs to the geometry, and 0 otherwise). Then, we define boundary conditions on the resulting volumetric texture. We employ a slice-based approach, meaning the boundary condition and the dynamic system are defined on a 2D volume slice. Albeit not a complete 3D parametrization, this choice produces a modeling algorithm that is more intuitive to define and visualize, because the DS only needs to be specified in 2D coordinates.

To do this, we define a principal axis on the volume among the three Cartesian axes (the axis is a parameter of our model), based on which each slice is defined. From each slice, we obtain its center of mass, defining a circular pattern around it using a DS. The exact dynamic system can be tuned to obtain slightly different appearances (roll, spiral, etc.). Typical systems to obtain this behavior are depicted in the center and right image of Figure 3. After a principal axis is chosen, the procedure computes boundary conditions on each voxel. For this, the method blurs

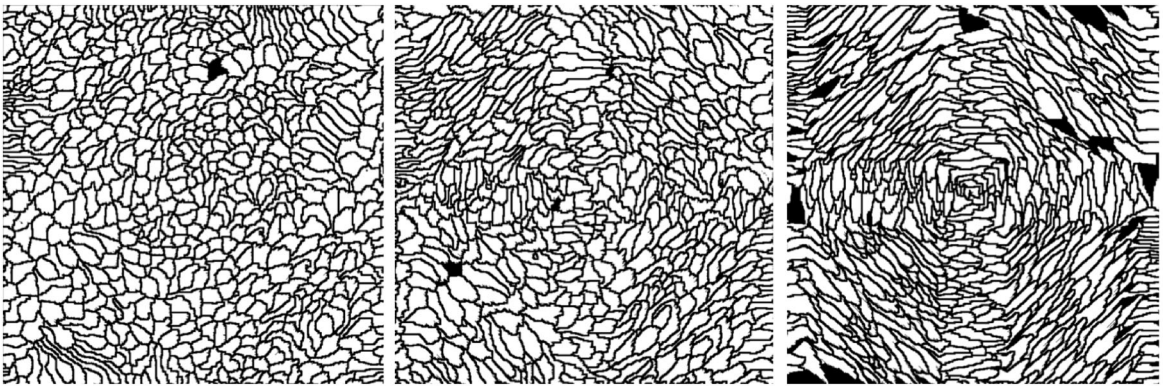

Figure 3. Particle systems following dynamical systems. Effect of the randomness parameter. From left to right, randomness: 0.3, 0.2, and 0.1, respectively.

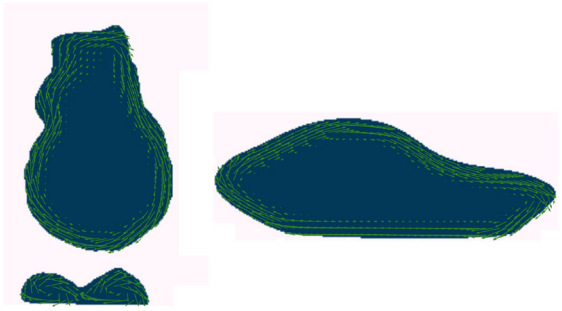

Figure 4. Discrete gradient vector field. The image shows that the computed vectors follow the exterior silhouette of arbitrary figures. For clarity, the image shows only some vectors in the field.

each resulting volume slice s_i and computes the gradient of the 2D blurred slice at each voxel, obtaining a vector at each position:

$$(v_x, v_y) = g_{s_i}[x, y],$$

where v_x and v_y are the gradient vector component in x and y, respectively. The vector that is orthogonal to the gradient can be used to follow the exterior silhouette. An orthogonal vector to (v_x, v_y) is $(v_y, -v_x)$. Figure 4 shows the result of these computations on a sliced model. The resulting discrete vector field adequately follows the exterior silhouette. The image also shows that the method works on slices with disconnected regions.

The boundary conditions are chosen if v_x or v_y are > 0, while a DS, centered at the center of mass, is employed on the interior of the slice. With this procedure, it is even possible to control the boundary region's thickness, to represent outer shapes and crusts. This is achieved by varying the kernel size in the blurring operation.

The process creates a realistic global deformation pattern for the slice (Figure 5). The bubbles will predominantly follow the 2D slice, but because of the inclusion of random numbers in the growth process, the bubbles are not constrained to the actual slice and may switch among slices and DSs.

The *randomness* parameter can be used to determine the homogeneity of the bubbles and orientations. For instance, a *randomness* value close to 0 forces bubbles to tightly follow the DS. This can be used for bread crumbs. A higher *randomness* value is suitable for sponges.

Figure 5. Porous slices generated with our method. The image shows that the bubbles follow the boundary conditions on edges, and the DS in the center, producing global and local natural patterns. The left, top image uses a higher randomness value than other slices, producing a sponge-like pattern. The left, bottom image was produced with the lowest randomness value.

DOI: 10.1002/cav

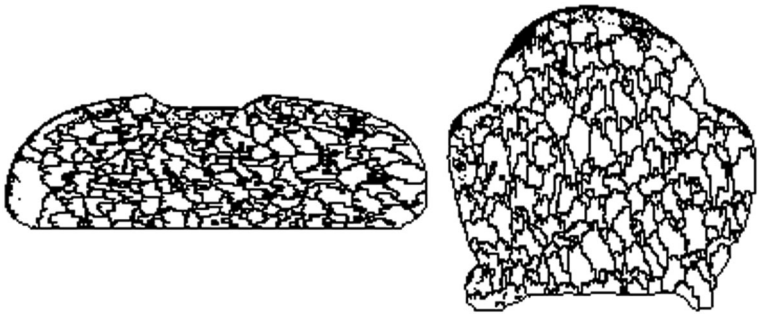

Figure 6. 2D slices with predominantly horizontal (left) and vertical (right) DSs. The image shows that the resulting bubbles produces patterns adapted to the ratio of each slice.

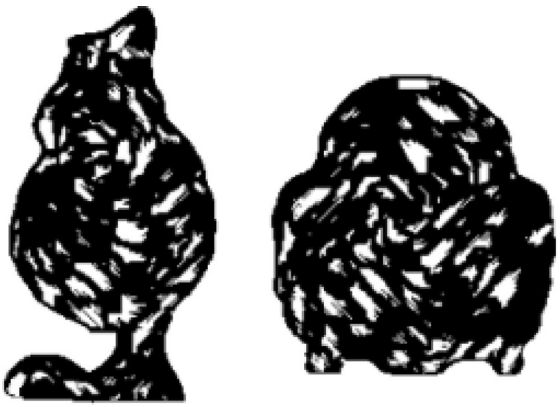

Figure 7. Bunny-shaped slices showing bubbles following the center of mass of the slice. Additionally, the separation parameter for bubbles is set to 2.

In addition, this method can be used to model different behavior on different regions by defining a different DS for different slice sets. The resulting porous structures may show different orientations depending on voxel and/or slice position.

If desired, the system can be configured to change the DS depending on the shape of the slice. For instance, if the width is much larger than the height, a horizontal predominant DS could be used. The same idea applies if the height surpasses the width. A threshold ratio value can be used to set these behaviors. Figure 6 shows the effect of setting horizontal and vertical predominantly DS.

As a final example, in Figure 7, we show bunny-shaped slices with bubbles following its center of mass, with separation among bubbles set to 2. This example demonstrates that the system can be configured to match different observable properties of materials (in this case, the visible distance between bubbles).

4. RESULTS

For all our tests, we employed an Intel(R) Core(TM) i5-2300 CPU (quad core) with an nVIDIA GTX 480 (480 shader units) GPU, which is a typical standard configuration. In Table I, we show computing times and memory usage of our modeling procedure. The table presents times for the different key steps of the algorithm (geometry loading, voxelization, and particle growing).

Table I. Modeling times expressed in seconds (mean of 10 runs), where *main algorithm* denotes the particles growing and avoiding each other.

Texture resolution	128^3	256^3	384^3
Voxelize and geometry loading	14.69	38.83	85.82
Main algorithm	34.61	155.99	356.35
Total time	49.3	194.82	442.17
Output texture size (MB)	2	16	54
Process memory usage (MB)	∼150	∼400	∼1800

Memory usage during the modeling algorithm run and final output size are also shown.

Given the computation times for high-resolution 3D textures it would be impractical to try different parameters at this resolution. The usual approach, then, consists in computing and selecting different parameter sets at low resolutions, and computing only one high-resolution texture.

The table shows that the procedure has low memory requirements, given that no optimizations are applied (one texture is used for each purpose, textures are uncompressed, etc.). Future versions of this software could drastically reduce memory consumption.

Currently, the code is written in Python and runs on a single CPU core. A GPU version of this software may achieve interactive or real-time performance by parallelizing the particle growing step.

5. APPLICATION: DIRECT VOLUME RENDERING

We created a demo application[†] that uses the particle system described in the previous sections to render bread and other porous objects using a standard DVR approach. Figure 8 shows an example of a porous structure, generated

[†]Available at https://www.github.com/rbaravalle/Pysys.

Figure 8. Our demo application showing a porous structure rendered in real time.

with our modeling algorithm. The image shows that the bubbles follow the center of mass of the visible slice.

This application uses a volume texture that is the result of our modeling method. This texture defines the structure of the object that is rendered, and a DVR algorithm along with a simple physically based shading model [33] is used to create the images. Physically based shading parameters are intuitive and, as shown in the example figures, can be used to easily represent a wide variety of porous objects.

To illustrate the variability, we created realistic-looking breads (Figure 13, Figure 14), sponges (Figure 9) and stones (Figure 10) by simply adjusting the modeling and shading model parameters. Because the sponge porous structure is more homogeneous, we set *randomness* to 0.5. The stones use fewer particles than the other materials, and the renderer uses a higher reflectance and index of absorption (light does not propagate as in the other two materials).

5.1. Comparison with Other Models

Artistic generation of porous structures are usually performed using different noise models, such as white noise, Voronoi noise, Perlin noise [34], and Worley noise [35]. These models generate pores by random value variations and further smoothing of the resulting volume. This gives acceptable results when the final pattern does not need to be fully controlled, given the randomness nature of the methods.

In particular, white noise does not capture the mesostructure required for an adequate porous material representation, because the generated textures are quite unrealistic. In a similar fashion, Perlin noise produces results that are too smooth, with statistical patterns unable to form pores in a flexible manner. These noise models make it

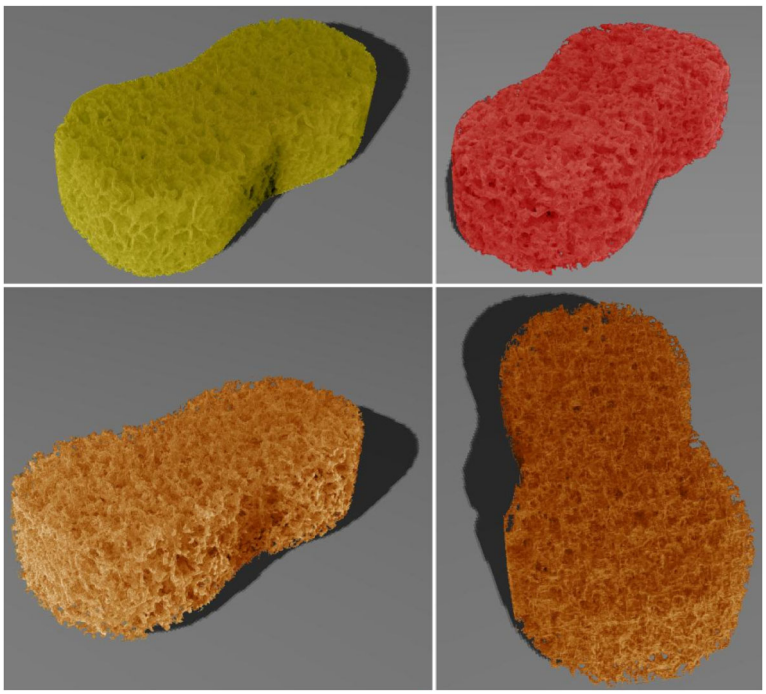

Figure 9. Rendered sponges. For this material, we generated a volumetric texture with a higher randomness value than in the bread case. The resulting geometry has homogeneous bubble sizes and distributions.

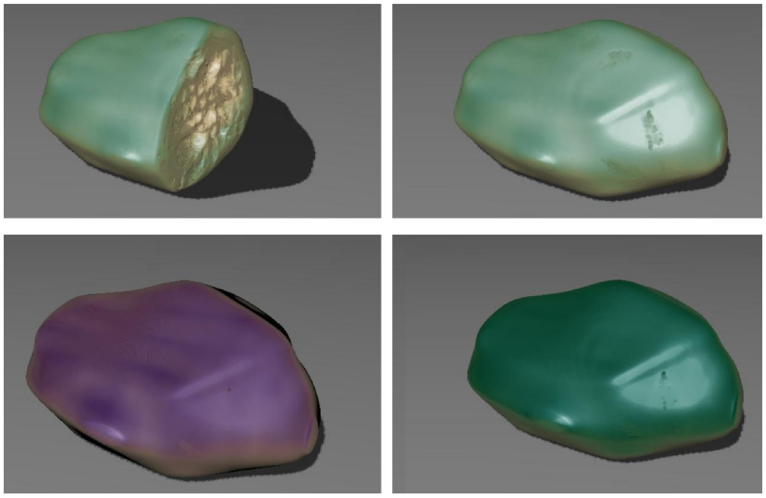

Figure 10. Rendered stones. For this material, we used less bubbles in the generation. When rendering, the index of absorption was incremented to prevent excessive subsurface scattering.

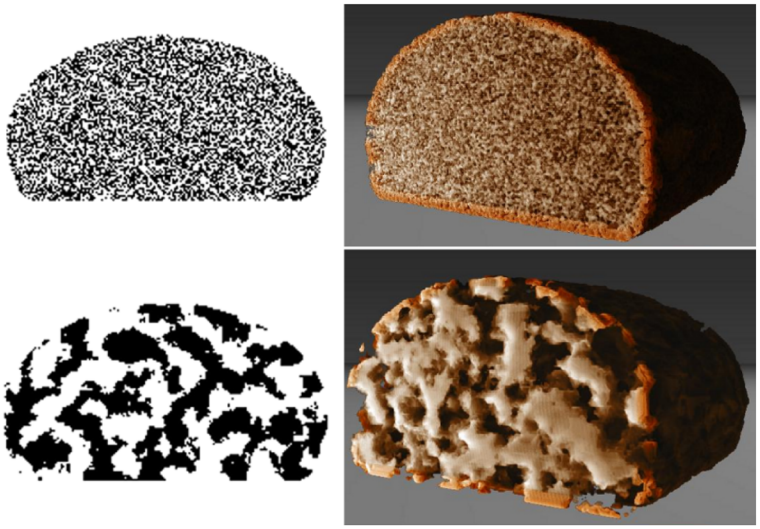

Figure 11. Porous structure simulated with white noise (top) and Perlin noise (bottom). The image shows that the material is somewhat non-porous (top) or smooth (bottom), making it difficult to control pore size distributions.

difficult to control the size and distribution of the resulting pores. Figure 11 shows rendering results using different noise models to generate a bread model. We could find no parameters that would generate realistic-looking results.

Voronoi and Worley noise are able to represent pore sizes and distributions at randomly positioned points in 3D space. Distance functions defined over these points are used to construct volume densities, generating air/mass volumes. This may result in acceptable representations for sponges and other homogeneous or randomly featured materials, such as stones and cells. Nevertheless, it is a difficult and an ad hoc task to obtain specific orientations on the resulting patterns (Figure 12). Other ad hoc techniques such as texture maps[‡] would be required for bubble deformation.

More advanced texture synthesis frameworks, such as Hypertexture [12], are overly general and use a combination of random noise and user-defined mathematical formulas to create 3D textures. Therefore, they require trial-and-error tuning functions and parameters that do not have an intuitive meaning in order to achieve realistically looking porous structures.

Our procedure does not require ancillary techniques to obtain convincing patterns, because the underlying DS

[‡]https://graphics.stanford.edu/wikis/cs348b-11/aharaux/FinalProject.

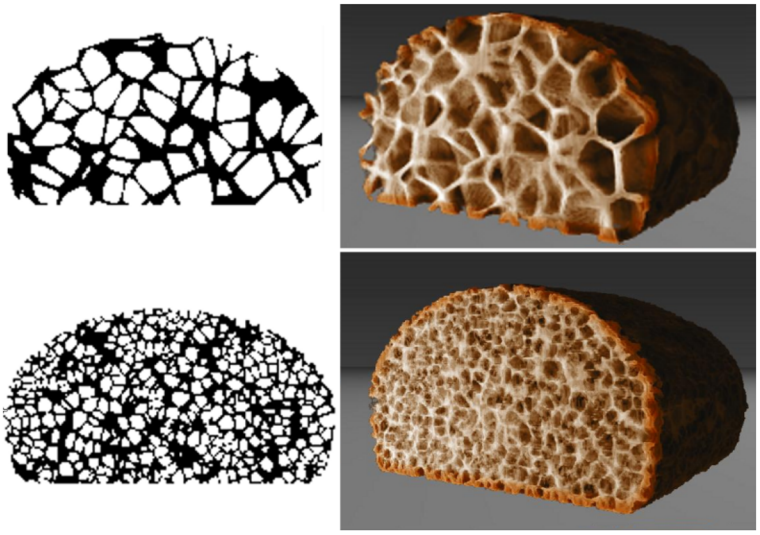

Figure 12. Porous structure simulated with Voronoi noise (top) and Worley noise (bottom). The image shows that, while the methods generates convincing porous structures, users should consider other ad hoc mechanisms to gain flexibility in the resulting patterns (for instance, to control pore orientations and shapes).

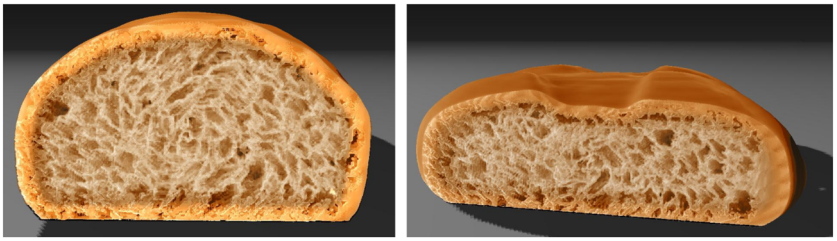

Figure 13. Rendered breads highlighting concentric bubble patterns obtained with our modeling algorithm.

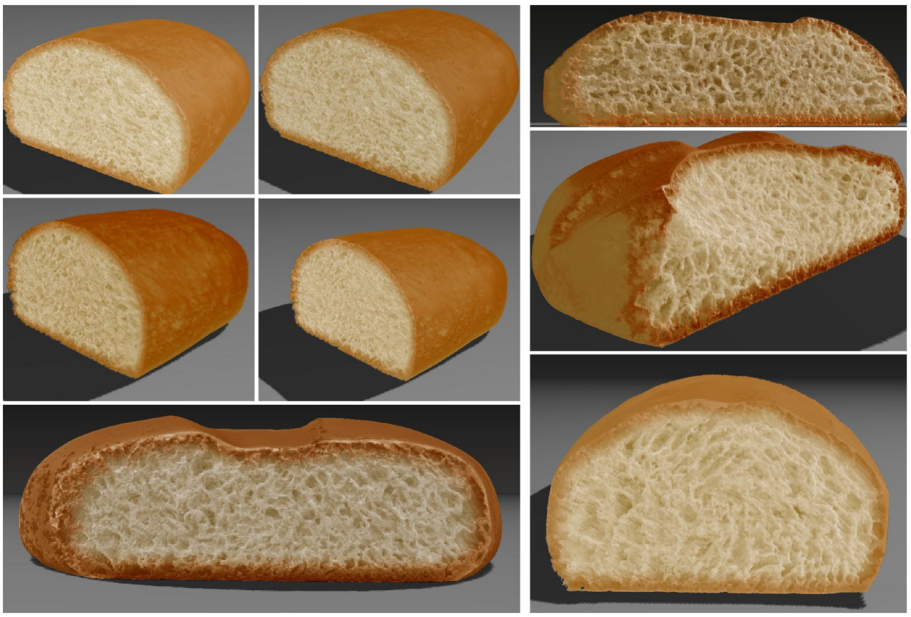

Figure 14. Rendered breads with different geometries and rendering parameters. The image shows that the method targets different bread appearances.

DOI: 10.1002/cav

captures these behaviors. Moreover, our technique naturally represents and controls the material outer shells. To reproduce this with any other method discussed here, it will be necessary to consider further ad hoc mechanisms, requiring extra computing resources. This implies that our method reduces modeling times and complexity. The resulting method is easy to parametrize and fully automatic, allowing not only programmers or computer graphics experts but also artists to employ the technique.

6. DISCUSSION

Our modeling algorithm convincingly represents the geometry of porous materials without introducing complex intermediate processes (capture, mesh generation, precomputation, post-processing). There are few previous approaches to porous modeling in computer graphics. An example is [27], but comparisons with this technique could not be established because key details are not explained (e.g., computing times and artist input).

Computing times for modeling are in the order of minutes (less than a minute for low-resolution textures). This is acceptable because the process is fully automatic, and the user only needs to care about selecting a very small set of parameters such as the outer 3D structure and the global appearance (amount of particles, texture size, etc.). The method takes care of all the modeling process details (global and local positioning, etc.).

Most of the computations occur during the particle growth phase. A parallel version of this algorithm could parallelize this computation, greatly reducing the computation times. Also, the method has low memory requirements, which can be further reduced with careful algorithmic considerations.

The modeling algorithm is flexible enough to create different materials like bread, sponges, and stones, by changing only a few parameters. Other materials such as pizzas and cheeses can be implemented in the same way, allowing to manage multiple materials using the same method. We choose to keep the DS set as simple as possible, representing the basic shapes observed in porous materials. Other complex patterns may be obtained using more involved DS equations. Nevertheless, given the DS and the boundary conditions defined, it was possible for us to obtain the most common natural shapes of porous materials.

Furthermore, the volume representation method allows real-time cuts in the volumes. This feature is clearly useful in many real-time applications, for instance, in video games. We also showed that the resulting model is suited to create realistic renderings of these materials in a real-time application using commodity hardware.

So far, with current off-the-shelf hardware, our method is limited in the final 3D resolution of the material when real-time applications are required. In other words, drastic close-ups to the structure could lead to a frame-rate drop. This limitation is tied to the GPU texture size and will be eventually circumvented with the next generations of graphic hardware.

7. CONCLUSIONS

We proposed a procedural generation model for porous materials. The method is based on a particle system that emulates a self-avoiding growth process within a volume, by means of DSs. A numerical simulation is computed to solve the resulting set of equations that represent the behavior of the DS. The particles avoid each other and grow by following the streamlines and the exterior silhouette of the material. The modeling algorithm is simple and crisp. The resulting 3D textures adequately represent porous materials, which can be used directly in a DVR renderer. We showed that we can create realistic renderings of porous materials like bread, sponges, and translucent stones in real time using standard off-the-shelf hardware. The results are suitable for application in several areas, such as medicine, video games, and photorealistic rendering. Our proposed technique is simple and flexible and does not share the drawbacks of current state-of-the-art methods, such as complex capture processes or mesh generation. We are currently developing methods for simulating other materials with different mesostructures, such as cheeses and bones. A main continuation of this work will be the parallelization of the modeling algorithm, (e.g., by employing OpenCL). Also, we will validate our method phenomenologically using measurements from actual porous materials. Lastly, we plan to explore other applicable rendering methods (such as surface representations or bidirectional texture functions [5]) and the requirements they may impose on the modeling output.

ACKNOWLEDGEMENTS

The authors wish to thank the anonymous reviewers, who helped to improve the quality of this manuscript. This work was supported by a doctoral scholarship of the National Science and Technology Council of Argentina (CONICET). This work was partially supported by research grant PGI 24/K060 of the SECyT-UNS. The bunny 3D model is courtesy of Stanford University.

REFERENCES

1. Kajiya JT. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques,* SIGGRAPH '86, New York, NY, USA, 1986; 143–150, ACM.
2. Kajiya JT, Von Herzen BP. Ray tracing volume densities. *SIGGRAPH Computer Graphics* 1984; **18**(3): 165–174.
3. Neumann L, Neumann A, Szirmay-Kalos L. Compact metallic reflectance models. *Computer Graphics Forum* 1999; **18**: 161–172.
4. Lefebvre L, Poulin P. Analysis and synthesis of structural textures. In *Proceedings of Graphics Interface '00*, Montréal, Québec, Canada, 2000; 77–86.

5. Tong X, Zhang J, Liu L, Wang X, Guo B, Shum H-Y. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Transactions on Graphics* 2002; **21**(3): 665–672.
6. Tong X, Wang J, Lin S, Guo B, Shum H-Y. Modeling and rendering of quasi-homogeneous materials. *ACM Transactions on Graphics* 2005; **24**(3): 1054–1061.
7. Chentanez N, Müller M. Real-time Eulerian water simulation using a restricted tall cell grid. In *ACM SIGGRAPH 2011 Papers,* SIGGRAPH '11, New York, NY, USA, 2011; 82:1–82:10, ACM.
8. Zhou K, Ren Z, Lin S, Bao H, Guo B, Shum H-Y. Real-time smoke rendering using compensated ray marching. In *ACM SIGGRAPH 2008 Papers,* SIGGRAPH '08, New York, NY, USA, 2008; 36:1–36:12, ACM.
9. Li S, Zhao Q, Wang S, Hao A, Qin H. Interactive deformation and cutting simulation directly using patient-specific volumetric images. *Computer Animation and Virtual Worlds* 2014; **25**(2): 155–169.
10. Baravalle R, Scandolo L, Delrieux C, García Bauza C. Modelado para la renderización foto-realista de pan. In *Mecánica Computacional*, Vol. XXXIII, Bertolino G, Cantero M, Storti M, Teruel F (eds). AMCA, 2014; 1695–1709.
11. Baravalle R, Patow GA, Delrieux C. Procedural bread making. *Computers & Graphics* 2015; **50**(2015): 13–24.
12. Perlin K, Hoffert EM. Hypertexture. *SIGGRAPH Computer Graphics* 1989; **23**(3): 253–262.
13. Reeves WT. Particle systems—a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics* 1983; **2**: 91–108.
14. Strogatz SH. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering (Studies in Nonlinearity)* 1st ed., Studies in Nonlinearity. Westview Press: Boulder, USA, 2001.
15. Levoy M. Display of surfaces from volume data. *IEEE Computer Graphics and Applications* 1988; **8**(3): 29–37.
16. Kruger J, Westermann R. Acceleration techniques for GPU-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03),* VIS '03, Washington, DC, USA, 2003; 38, IEEE Computer Society.
17. Kratz A. Advanced illumination techniques for GPU-based direct volume rendering. *Master's Thesis*, Koblenz - Landau, Wien, Germany, 2006.
18. Schechter H, Bridson R. Ghost SPH for animating water. *ACM Transactions on Graphics* 2012; **31**(4): 61:1–61:8.
19. Shao X, Zhou Z, Wu W. Particle-based simulation of bubbles in water solid interaction. *Computer Animation and Virtual Worlds* 2012; **23**(5): 477–487.
20. Donner C, Jensen HW. A spectral BSSRDF for shading human skin. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques,* EGSR'06, Aire-la-Ville, Switzerland, Switzerland, 2006; 409–417, Eurographics Association.
21. Kurt M, Szirmay-Kalos L, Křivánek J. An anisotropic BRDF model for fitting and Monte Carlo rendering. *SIGGRAPH Computer Graphics* 2010; **44**(1): 3:1–3:15.
22. Voglsam G. Real-time ray tracing on the GPU—ray tracing using CUDA and kD-trees. *Master's Thesis*, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 2013.
23. Mišić MJ, Đurđević DM, Tomašević MV. Evolution and trends in GPU computing. In *Proceedings of the 35th International Convention*. MIPRO, 2012; 289–294.
24. Owens JD, Houston M, Luebke D, Green S, Stone JE, Phillips JC. GPU Computing. In *Proceedings of the IEEE* 2008; **96**(5): 879–899.
25. Kurt M, Edwards D. A survey of BRDF models for computer graphics. *SIGGRAPH Computer Graphics* 2009; **43**(2): 4:1–4:7.
26. Donner C, Lawrence J, Ramamoorthi R, Hachisuka T, Jensen HW, Nayar S. An empirical BSSRDF model. *ACM Transactions on Graphics* 2009; **28**(3): 30:1–30:10.
27. Cho JH, Xenakis A, Gronsky S, Shah A. Anyone can cook—inside Ratatouille's kitchen. In *ACM SIGGRAPH 2007 Courses*. ACM, New York, NY, USA, 2007; 1–58.
28. Van Dyck T, Verboven P, Herremans E, *et al.* Characterisation of structural patterns in bread as evaluated by X-ray computer tomography. *Journal of Food Engineering* 2014; **123**(0): 67–77.
29. Purlis E. Bread baking: technological considerations based on process modelling and simulation. *Journal of Food Engineering* 2011; **103**(1): 92–102.
30. Thorvaldsson K, Janestad H. A model for simultaneous heat, water and vapour diffusion. *Journal of Food Engineering* 1999; **40**(3): 167–172.
31. Rasmussen N, Nguyen DQ, Geiger W, Fedkiw R. Smoke simulation for large scale phenomena. *ACM Transactions on Graphics* 2003; **22**(3): 703–707.

32. Scanlon MG, Zghal MC. Bread properties and crumb structure. *Food Research International* 2001; **34**(10): 841–864.
33. Brent Burley and Walt Disney Animation Studios. Physically-based shading at Disney. In *ACM SIGGRAPH*, 2012; 1–7.
34. Perlin K. An image synthesizer. *SIGGRAPH Computer Graphics* 1985; **19**(3): 287–296.
35. Worley S. A cellular texture basis function. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques,* SIGGRAPH '96, New York, NY, USA, 1996; 291–294, ACM.

AUTHORS' BIOGRAPHIES

Rodrigo Baravalle received the licenciate degree in Computer Science (2010) and PhD in Informatics (2016) at National University of Rosario, Argentina. He is a PostDoc Fellow at the National Science and Technology Council of Argentina. His research interests include procedural models, fractals, and photo-realistic rendering.

Leonardo Scandolo is a PhD candidate at Delft University of Technology in the Computer Graphics and Visualization group. He received his Licenciate degree in Computer Science from the National University of Rosario in 2013. His research interests include real-time rendering, visualization, and high-performance computing.

Claudio Delrieux is a full professor and PI at the Electrical and Computer Engineering Department, Universidad Nacional del Sur, Bahia Blanca, Argentina. He holds a BSEE (1990), MSCS (1996), and PhD in CS (2004), all from the Unversidad Nacional del Sur. He was Fulbright Posdoctoral Fellow at the University of Denver during 2006 and 2007. His research interests include image processing, scientific visualization, and computational intelligence.

Cristian García Bauza received the Systems Engineering Degree (2006) at University of Central Buenos Aires and PhD in Computer Science (2013) at National Southern University, Argentina. He is a Lecturer in Computer Science at University of Central Buenos Aires and Assistant Researcher at the National Science and Technology Council of Argentina. His research interests are visualization and virtual reality and graphic engines architectures.

Elmar Eisemann is a professor at Delft University of Technology, heading the Computer Graphics and Visualization group. Before, he was an associate professor at Télécom Paris-Tech and senior researcher in the Cluster of Excellence at MPII/Saarland University. His interests include real-time and perceptual rendering, alternative representations, shadow algorithms, global illumination, and GPU acceleration techniques. In 2011, he was honored with the Eurographics Young Researcher Award.