Volume xx (200y), Number z, pp. 1–15

A Framework for Interactive Hypertexture Modeling

G. Gilet and J.M. Dischler

LSIIT UMR CNRS-UDS 7005, Université de Strasbourg, Strasbourg I, France

Abstract

Hypertexturing can be a powerful way of adding rich geometric details to surfaces at low memory cost by using a procedural 3D space distortion. However, this special kind of texturing technique still raises a major problem: the efficient control of the visual result. In this paper, we introduce a framework for interactive hypertexture modeling. This framework is based on two contributions. Firstly, we propose a reformulation of the density modulation function. Our density modulation is based on the notion of shape transfer function. This function, which can be easily edited by users, allows us to control in an intuitive way the visual appearance of the geometric details resulting from the space distortion. Secondly, we propose to use a hybrid surface and volume-point-based representation in order to be able to dynamically hypertexture arbitrary objects at interactive frame rates. The rendering consists in a combined splat- and raycasting-based direct volume rendering technique. The splats are used to model the volumetric object while raycasting allows us to add the details. An experimental study on users shows that our approach improves the design of hypertextures and yet preserves their procedural nature.

1. INTRODUCTION

Hypertextures were introduced about two decades ago by Perlin and Hoffert [PH89]. They represent a special kind of volumetric texturing technique that allows one to add true geometric details to surfaces. Unlike volumetric texels or displacement maps (see related works), both consisting in explicitly texture mapping details with a certain depth over surfaces, hypertextures are not based on texture mapping principles. They subsequently necessitate no surface parameterization. Instead, they require a volumetric density-based representation of the entire object. Hypertexturing basically consists in distorting the 3D space of the object. The space distortion, based on one or multiple density modulation functions (DMF) (these can be, in turn, based on a procedural noise function) induces a distortion of the object. The latter can completely alter the object space (as for melting or dripping effects). Or it can result in a texture-like appearance, as for fur or moss. Hypertextures may also be used to create turbulent gazes, smoke, clouds or fire balls by adding “meso-scale” details to coarsely defined volumetric models. Some examples of complex real-world effects, that potentially can be modeled with hypertextures are shown on figure 1. None of these examples can be obtained with simple techniques such as bump mapping [Bli78] or displacement mapping [Coo84]. In turn, 3D texture mapping techniques like volumetric texels [KK89] would require an excessive amount of memory to store the full 3D data or are not suitable for some effects that affect the entire object, such as the cloud or the grass on this figure. Because of the procedural definition of the DMF, hypertexturing requires nearly no texture memory at all. In this sense, hypertexturing remains a very interesting approach for creating some volumetric details on objects that would be too memory consuming if represented as explicit triangular meshes or volumetric data (regular voxel grids, tetrahedral meshes, etc.). However being able to hypertexture an arbitrary surface in a controlled way still remains a challenging issue, because it is usually hard to correlate a given DMF with a certain visual result. Hypertextures usually require many experiments. In addition, procedural methods require a good programming experience, that artists or users might not have. Our aim in this paper is to address this issue. We propose a new framework for an interactive modeling of hypertextures, which requires no programming knowledge and yet preserves the procedural nature of these textures. Our work is based on two contributions.

Firstly, we introduce a combined splat-based and raycasting based approach for interactive rendering purposes. Our motivation is to allow users to get an instant feedback of

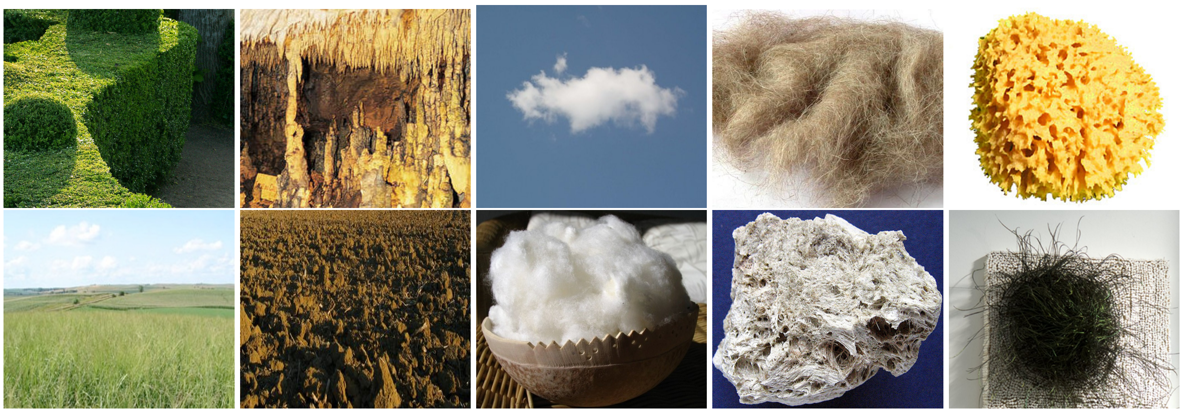


Figure 1: *Photographs of complex natural phenomena. From left to right, top to down: hedge, cave, cloud, wool, sponge, lawn, clods on a field, cotton, porous stone and grass.*

their manipulations. Unlike previous interactive hypertexturing techniques (see related works), we do not represent the object by a full regular grid of voxels (usually stored as 3D texture on the GPU). The motivation is that with splats we adaptively fit better shapes. With a regular grid of voxels, many voxels might be empty depending on how the shape of the object fits into a box. Unstructured volumes that would result in many empty voxels, like the Utah teapot for instance (because of its handle and its spout), can be more compactly approximated by an unstructured set of splats, especially with an adaptive kernel size (surface splats for the border and volume splats of different sizes for the interior). It furthermore allows us to naturally skip empty spaces, by focusing computations on useful object parts. Our representation is bi-scaled. The splats are used to model the coarse shape of the object. Then, we use a hardware-based raycasting to add hypertexture details. We propose two different methods for rendering the splats. The first one aims at obtaining "good quality"results and is based on EWA-splatting with overlapping kernels in screen space. This results in antialiasing but increases computations because multiple rays are cast on each pixel for the details. Conversely, the second one aims at obtaining fast results by casting a single ray per pixel.

Secondly, we propose a reformulation of the DMF, based on a set of three transfer functions (a shape transfer function, as well as color and transparency transfer functions). 1D, 2D or even 3D transfer functions have been used a lot to render volumetric data sets by associating color and opacity to numerical scalar values, especially in the field of scientific visualization (as for medical MRI data). Here, we propose to add a new transfer function (in addition to the classical RGBA one), which allows users to control the appearance of the hypertexture details. This function, when edited in real-time, allows users an easy exploration of various results. We note that in order to keep the procedural nature of hypertextures, we still use a noise function, but it will be "filtered" by the shape transfer function. Experimental studies that we conducted show that such a function can significantly help users to reproduce complex natural effects such as the ones illustrated in figure 1.

The remaining parts of the paper are organized as follows. The next section presents an overview of related works. Then, we recall the basic principles of hypertextures using DMFs. Section 4 describes our new framework. We show how the use of shape transfer functions can help to control some visual aspects. Section 5 describes the two interactive rendering techniques, both using a combined splat- and raycasting-based approach. Finally, before concluding, we present some graphical results, as well as an experimental study.

2. Related works

In computer graphics, it has been early understood that some types of complex textures characterized by a pronounced geometry cannot be rendered by bump mapping [Bli78], or more generally normal mapping. Conversely, an explicit modeling of such details in the form of triangles seems not reasonable because of huge memory requirements and strong aliasing artifacts. Therefore, "true 3D" textures were introduced. These are generally based on a volumetric representation of the natural structures. In the past, mainly two different approaches (see [DG01] for more details on 3D texturing) were proposed:

Firstly, 2D texture mapping has been generalized to 3D texel mapping in [KK89]. This approach consists in mapping repetitive semi-transparent volumetric elements, called texels, onto surfaces. Besides efficiently modeling such textures [Ney98], recent work [CTW*04] also tackled the problem of further illuminating correctly volumetric materials by introducing shell texture functions that include subsurface scattering. 3D texturing techniques raise several problems such as the need for a parameterization of the surface (not only in 2D but also in 3D), which is not always easy to obtain with arbitrary shapes. Furthermore, important distortions (due to both, parameterization and curvature) as well as self-intersection problems might arise, especially if the surface has concavities with important curvatures and if the texture depth is important compared to the size of the object. Therefore, these textures are often limited to create thin layers upon surfaces. This excludes "deeper" effects like the cloud or grass of figure 1 which affects the entire object. Volumetric textures are often explicitly represented as discrete 3D arrays (or even higher dimensioned if some visibility issues are pre-computed and stored explicitly), so that texture compression is nearly always necessary. Despite compression, only small samples can be stored and, thus, must be repeated. So, especially for non-periodic textures, more or less important repetition artifacts may also become visible. Moreover, because of explicit storage, dynamic editing of such textures seems difficult. Generally, the creation of 3D texture samples is decoupled from rendering.

Secondly, Bump mapping has been generalized to displacement mapping [Coo84]. Displacement mapping consists in truly deforming the surface along its normal by using an elevation map. Since it is based on texture mapping, it raises problems similar to the previously described 3D texturing techniques, namely need of surface parameterization, problems of self-intersections and distortions. In addition, it can only be used to model very simple structures: height fields. This excludes more complex effects as shown in figure 1. Actually, displacement mapping can be considered as a special case of more general surface and object deformation techniques, which is illustrated in figure 2. Instead of displacing surface points along the normal, one may displace them along any arbitrary direction (shown on the middle of the figure) for instance using a vector field [Lew89, Ped94]. Such a more general deformation implicitly includes displacement mapping as a particular case for which the vector field is simply matching the normal distribution on the surface. Using an arbitrary vector field allows one to recover more complex and also more natural structures including concavities, which are frequent in natural processes like erosion for example. The right-most picture of figure 2 represents the most general case of deformation. In this case, the full volume is distorted according to a 3D vector field, e.g. all points, including the interior of the object are displaced. This allows one to further introduce changes of topology, such as for example holes, which may also happen with natural processes (see the cloud, the sponge or the stone of figure 1).

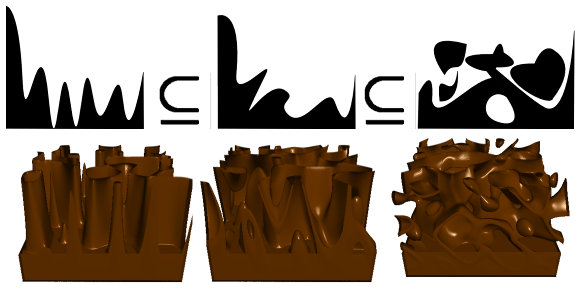


Figure 2: *Three classes of deformation: simple displacement mapping (left), surface deformation by an arbitrary vector field (middle) and "full" volumetric deformation by a 3D vector field with change of topology (right).*

Hypertexturing [PH89] belongs to the second type of texturing technique. It is the most general form of deformation and based on a density modulation function (DMF) directly defined throughout 3D space. The full volume is deformed, which allows one to model concavities and holes. The 3D definition of the DMF makes that no surface parameterization is required. So, no self-intersection problems or texture distortions due to surface parameterization or surface curvature arise. Because the DMF is procedurally defined (generally based on 3D noise or turbulence), hypertextures are also extremely compact. Finally, the volumetric representation allows for semi-transparent effects to be rendered such as clouds, fire or fur for instance.

Because of the volumetric representation, hypertexturing was mainly applied in conjunction with implicit surfaces, as in [WH96]. But hypertextures may also be used to create complex planet terrains with caves [GM08] or trees [SG04]. In [DG95], polygonal surfaces are converted into pseudo-implicit representations using an additional skeletal structure in order to be able to hypertexture polygonal objects by defining distance fields. Other approaches consist in converting objects into a 3D regular grid of voxels and then apply a thinning algorithm to build a density variation according to the distance to the completed skeleton [SJ02]. All volumetric texturing techniques require complex rendering systems, like ray marching, a method for the direct ray tracing of volumetric objects. This rendering technique is extremely time consuming. Hence, recent work was also concerned with the interactive rendering of hypertextures. [MJ05] uses a voxel-based representation of the object, stored as 3D texture on the GPU, and then applies a hardware-based direct volume rendering technique (either slice-based or, with increasing GPU power, pixel-shader raycasting-based). The pixel-shader is in charge of applying the procedural DMF directly uploaded as source code into the GPU. As for [PH89], the DMF is defined procedurally in shader language, so programming knowledge is required.

A lot of past research has also focused on defining noise functions with various statistical, spectral and visual properties [Per85, Lew87, Lew89, Wor96, Per02]. However the correlation with hypertextures and DMFs is not always obvious. Most hypertextures are still obtained "by chance". In [DG97], profile curves are analysed in order to build turbulence-like functions with some given visual characteristics. Such functions can then be extended to 3D space and can be used as underlying DMF for hypertexturing purposes. However, the correlation with distortion fields is not obvious and the control of the resulting visual effect on an arbitrary object remains difficult.

In this paper, we propose a new framework for an interactive modeling of hypertextures, e.g. a method for easily and interactively creating such textures. We do not address issues like very efficient real-time rendering (our rendering is limited to providing an interactive feedback to users) or the problem of computing realistic illuminations and lighting effects.

3. Density modulation functions and hypertextures

In this section, we briefly recall the principles of traditional hypertexturing as introduced by Perlin and Hoffert [PH89]. With hypertextures, an object $O(x,y,z)$ needs to be defined as a density variation throughout 3D space. Generally, a bounding volume is set. Beyond this bounding volume the density is considered as null. Inside the object the density smoothly varies between 0 and 1, where 1 means totally opaque and 0 totally transparent (e.g. no matter is present). The in-between region is called the soft region of the object. The key point consists in applying to this soft region one (or multiple) *density modulation functions* (DMFs). The DMFs are based on noise, turbulence, bias and gain. We do not further describe these four basis functions here, but refer the reader to [PH89] for more details. Note that any procedural definition may be used as DMF. In [PH89] no limitations are given. A common example is a soft sphere of radius R centered on the origin:

$$O(x,y,z) = \begin{cases} 1 - \frac{d}{R} & \text{if } (d < R) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

d is the Euclidean distance $d = \sqrt{x^2+y^2+z^2}$. On the origin the density is 1 while it decreases towards 0 when moving away and approaching the distance R. The soft sphere can now be distorted by a DMF for instance based on noise in the following way:

$$O(kx,ky,kz)\ ,\ \textit{with}\ k = 0.5 + 0.5 \cdot \textit{noise}(fx,fy,fz). \tag{2}$$

noise is a 3D noise function returning a pseudo-random value between -1 and 1, and f represents its "frequency". Rendering of such a distorted object can be done using software raymarching or by using the GPU in the form of one of the existing hardware-assisted direct volume rendering techniques. Direct volume rendering can be based for instance on a 3D texture rendered with slicing or raycasting. Or it can be based on a splatting / EWA splatting technique. Programmable graphics hardware allows us to evaluate noise functions directly within shader programs so that the DMF can be applied straightforwardly during the pixel shading computation. In order to apply a lighting model, the gradient $\nabla O(kx,ky,kz)$ can be used as normal vector with any reflectance model (for example the Phong model). A phase function requiring no normal vector may also be used instead.

As outlined by this example, hypertexturing is a very general approach for adding procedural details to soft objects. However, as can be seen as well from this example, hypertexturing is not always easy to bring into play with arbitrary objects, since it requires both: a volumetric functional definition of the object and a density modulation function. Concerning the first point, implicit surfaces are very close to such a representation (e.g. a distance field, where the skeleton represents the innermost high density part). Therefore, implicit surfaces can be generally straightforwardly used with hypertextures. Other objects may also be converted into soft objects, for example using a volumetric distance transform.

The second point is more intricate. Firstly, the distortion, if not carefully defined, might modify the actual bounding volume of the object, e.g. the boundary beyond which the density is sure to be always null. Knowing well the bounding volume is however important for rendering efficiency. The ray marching must start at some location and a distant starting point would considerably increase the rendering time (because the density might be null on many locations before entering the actual object). In the previous case for example, a point P outside the initial radius R might have a density greater than 0 after distortion by the DMF although the initial bounding volume was a sphere with radius R. Indeed, take a point $P = (2R, 2R, 2R)$ (e.g. outside the bounding volume) and $f = 1$. If by chance $noise(2R,2R,2R) = -1$, then $k = 0$, hence $d = 0$, so that $O(k2R,k2R,k2R) = O(0,0,0) = 1$. So, because it depends on noise, which is a random function, it seems difficult to determine the new boundary of the distorted sphere. Techniques, like for example Heidrich et al., proposed [HpS98] to use affine arithmetics to approximate the bounding boxes of the deformation for procedural displacement mapping. But determining the new boundary of a distorted model remains a difficult issue. One simple solution would consist in choosing a DMF that does not expand the initial object.

Secondly, there is no intuitive way for correlating a DMF with a certain visual result. With a lot of experience, it is naturally possible to guess, especially as experience grows with time, better and better what DMF will approximately result in what visual effect. Following question is even more problematic: given a certain visual effect in mind, what DMF should I program to obtain that exact result? The global "lack of intuitive control" is in our opinion one of the major difficulties.

In the following subsection, we describe our model. It is a specialization and thus less generic than the original hypertexture model that we just described. But by trading generality for better usability, we intend to globally improve the control of visual effects on arbitrary objects. Our interactive rendering further allows users to dynamically explore solutions.

4. Shape transfer functions and hypertextures

This section describes our new framework for modeling hypertextures. In our framework, as for traditional hypertextures, an object $O(x,y,z)$ is defined in the form of a density variation throughout 3D space, e.g. $O(x,y,z) = d$, where d represents a scalar density between 0 and 1. Our space distortion model is defined as follows:

$$O(x - k \times \frac{V_x}{||V||}, y - k \times \frac{V_y}{||V||}, z - k \times \frac{V_z}{||V||}) \qquad (3)$$

where $V = (V_x, V_y, V_z)$ represents a 3D vector field and $||\,||$ the norm of a vector. k allows us to apply the actual distortion by shifting points along V. k, which can be considered as a distortion factor, is defined as follows:

$$k = aT\left[\frac{1 + noise(\frac{x+rV_x}{f}, \frac{y+rV_y}{f}, \frac{z+rV_z}{f})}{2}\right] \qquad (4)$$

a and f respectively represent an amplitude and a frequency. $T[]$ is the shape transfer function. This function, returning a value between 0 and 1 is a table stored as texture on the GPU. The formula is for the 1D case. A more complex 2D function will be shown in the next subsection. *noise* represents any 3D noise function returning a pseudo-random value between -1 and 1. Note that in our model, the noise is not evaluated on (x,y,z) but on a shifted point, according to the vector field V and a factor r (between 0 and 1). Together, the vector field V and r allow users to control the class and type of deformation as illustrated in figure 2. With classical hypertextures the deformation is generally applied along the gradient of density of the object or along a noise-perturbed gradient (in the previous case of the soft sphere the deformation is applied along the radius since the sphere is centered on the origin, e.g. along the gradient). But this somewhats limits the possibilities. Using in our case an explicit vector field improves the design possibilities while results remain intuitive (see subsection concerning vector field).

For shading purposes, a color and transparency on $O(x,y,z)$ is given by another table $RGBA[d]$. This is a classical color and opacity transfer function. In our case, we do not directly use the density d of the soft object as opacity coefficient, but propose to use a transfer function. This allows us to decouple the actual transparency from the density value defining the volumetric object, for instance to apply hypertextures also on completely opaque objects (e.g. objects with no semi-transparency at all). Such objects will be defined by a density variation between 0 and 1, but have an opacity of 1 for all densities greater than 0. The opacity remains 0 where the density is 0. The gradient ∇O is computed by discrete differentiation for shading purposes.

We now describe and explain the influence of the parameters of this model, especially the motivation of using a vector field and a shape transfer function, which at first glance seem to restrict the variety of DMFs compared to a full procedural definition. a and f are classical parameters. The former coefficient a represents the depth of the hypertexture. With $a = 0$, the depth is null, that is, the amplitude of distortion is null. The latter coefficient f allows one to determine the *size* of hypertexture features. The greater f, the more *features* are present per volume element unit and the smaller are these features.

The shape transfer function represents the core of our model. It is represented as a 1D or 2D table $T[]$ and allows us to control the actual shape of the distortion, and thus the resulting visual features added to the object. It is indexed by noise in order to maintain the procedural nature of the 3D distortion. 1D or 2D shape transfer functions can be edited in real time using adequate tools. But in this paper, our aim is not to focus on human-computer interaction ergonomics, e.g. to show how such functions can be most efficiently edited. Our aim is to show that the use of such functions (however they do have been obtained) can help to improve the control of visual results. For our examples, we used simple spline curves editors, as well as image painting tools (paint brushes). Indeed, a 1D function can be easily edited like a 1D spline curve along the x axis with some control points and tangents. A 2D function can be either edited like a grid of surface patches using the tensor product of 1D curves or it can be painted as a grey scale image using photo-editing techniques and tools. Once it has been painted by the user, the transfer function (1D or 2D) can be instantly uploaded into the GPU, thus showing in real-time the effect on the hypertextured object. In the following subsections we respectively describe the influence of the shape transfer function in 1D and 2D cases as well as the influence of the vector field.

4.1. 1D shape transfer functions

As introduced in the previous formula 4, noise is used as index for the shape transfer function. So, the underlying aspect of the noise function will play an important role concerning the visual effect of the transfer function, and more specifically the probability and spatial distribution of the noise values. The basic shape of a classical Perlin noise function as well as its probability distribution histogram are shown on figure 3 respectively on the left and the middle. We can see that values close to 0 are more probable than the extremities -1 and 1. The right part of this figure shows the same noise, but instead of using the actual noise value, we draw its corresponding probability. On this image the 0 isoline is well highlighted (highest probability). Our shape transfer function will allow us to modify the value distributions, but without modifying the actual shape of the noise.

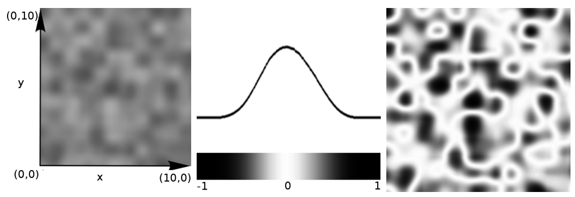


Figure 3: *A 2D crop of a classical 3D Perlin noise (left), its histogram of probability distribution (middle) and the same 2D crop of noise as on the left, but drawn with a grey intensity related to the probability (right). On the latter the extreme values -1 and 1 appear darker (less probable) and values around 0 brighter (highly probable).*

Figure 4 shows results obtained for different 1D transfer functions, respectively depicted on the first column. We applied the functions to a simple soft sphere with density decreasing according to the increasing radius. For a better visual comprehension, we used a step-function for the opacity which results in extracting an iso-surface without any semi-transparency effect. The second column shows the result obtained by indexing the shape transfer function with the noise function of figure 3 (we show a 2D crop of a 3D noise function). The next columns show the result when the shape transfer function is applied to the soft sphere with a factor $r = 0.0$, $r = 0.5$ and $r = 1.0$ according to the previously described model (the second picture is obtained by using a clipping plane, which allows us to see the interior of the perturbed sphere). For these examples, we used a vector field projecting points onto the closest sphere boundary (other vector fields will be shown later, since the latter has a significant visual effect). The case $r = 0.0$ means that the 3D noise is evaluated exactly on the location of the considered point without any shift. Depending on the vector field and on the amplitude of deformation, this can create complex effects, including cavities and holes. Figure 3 helps us to understand the results of figure 4 and especially to predict what effect a given 1D transfer function might have. The first example of figure 4, on the first row, shows an identity transfer function, e.g. $T[x] = x$. The result is a classical noise-based distortion. This image can be used as reference, showing what happens when no transfer function is used at all. On the second example, second row, the transfer function has a value of 1 everywhere, except on the two extremities, where the value is decreasing to form holes. This intuitively means that when the noise value reaches these extremities (-1 or 1), holes will be formed in the direction of the vector field. The cross section of the holes is related to the noise shape (see right of figure 3: black zones corresponding to -1 and 1 values). The obtained result on the hypertextured sphere actually matches well this intuition for all values of r. The next example, third row, shows a transfer function that creates bumps with a crater-like profile as the noise value reaches 1. The result obtained on the sphere for all values of r is again intuitive. The last example shows a transfer function that results in bumps when the noise value approaches 0. The effect is that isolines are created corresponding to the noise 0 isovalues.

The last column of figure 4 illustrates the extreme case $r = 1.0$ and needs some more comments. Since the vector field we used in this case projects towards the closest point on the sphere, all points laying on a radius of the sphere are given the same displacement factor. In addition, in a sphere, the radius matches the normal direction of its surface (e.g. the gradient). All of this leads to a final visual result that looks like classical displacement-mapping. But compared to displacement mapping, our approach differs on several important points. Firstly, instead of using a surface parameterization, difficult to obtain for arbitrary surfaces, our approach is based on a vector field that, in this particular case, points towards the surface. Such a vector field is not only easier to compute compared to a surface parameterization, but by modifying this field, as will be shown in the next subsection, we give users the possibility to obtain more complex visual results.

As illustrated by these examples, 1D transfer functions represent an easy and straightforward tool to control visual aspects. However, the possibilities for designing more complex hypertextures are still limited. It actually allows us only creating very simple and symmetric structures such as bubbles, bumps, peaks or holes, and not more complex structures like flames. In order to create more complex details on surfaces, we can use a combination of multiple noises with individual transfer functions at different amplitudes and scales.

Figure 5 shows three examples of complex structures obtained by using a combination of multiple transfer functions and noises. To do so, we replace the previous formula 4 by

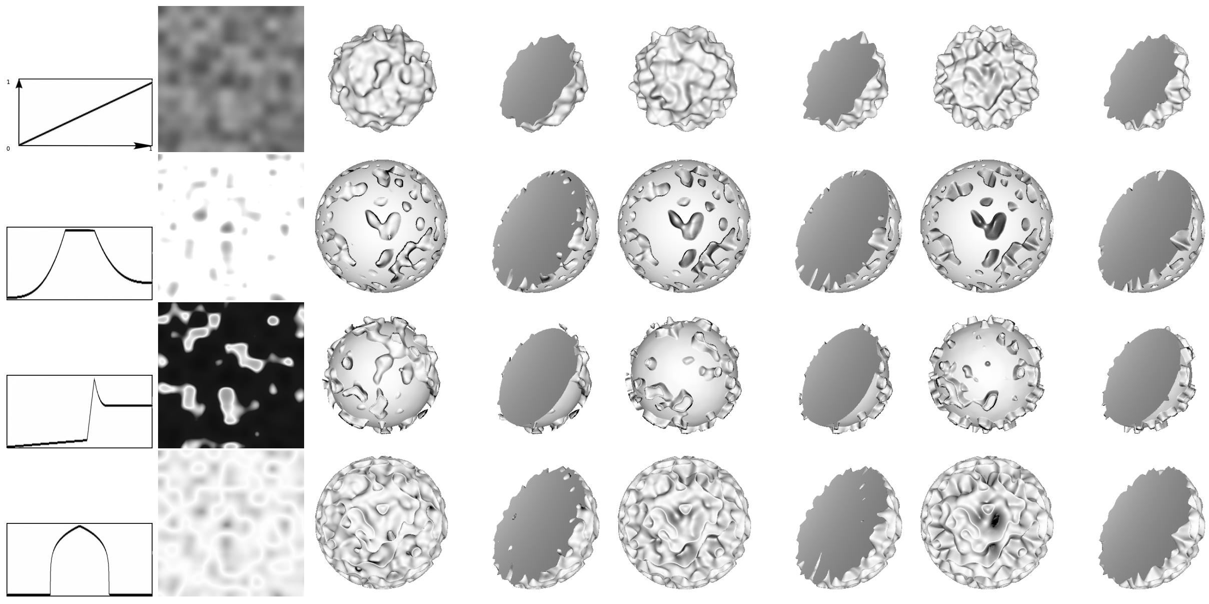


Figure 4: *Four examples of 1D shape transfer functions (left) and the corresponding results on soft spheres for values* $r = 0$, 0.5 *and* 1 *from left to right.*

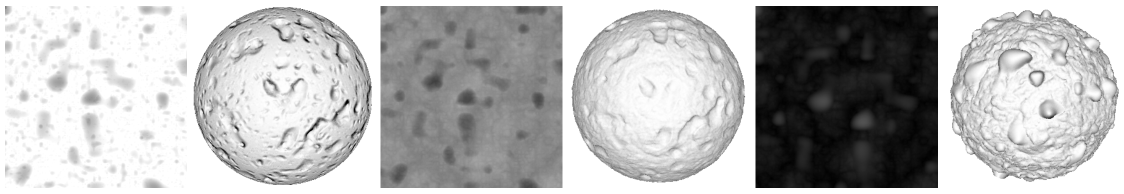


Figure 5: *Combining multiple shape transfer functions with multiple noises at different scales (*$r = 0.5$*). Left and middle show a straight sum. Right shows blending.*

following one:

$$k = \oplus_{i=1}^{n} a_i v_i,\ with\ v_i = T_i\left[\frac{1+noise(\frac{x+r_iV_x}{f_i},\frac{y+r_iV_y}{f_i},\frac{z+r_iV_z}{f_i})}{2}\right] \quad (5)$$

$\oplus$ represents a given operator chosen among a predefined set of operators (in our case we defined only two operators: sum and blending). For the first example on the left, we used a sum of three noises (e.g. $\oplus$ is simply $\sum$) with the same transfer function for each (the second shape transfer function of figure 4, e.g. representing holes). Only amplitudes and frequencies (scales) were changed, thus creating holes of different sizes. On the second example (middle), we used a sum of five noises, with two different transfer functions. The first noise used again the holes-like transfer function and the four following ones a V-shape $T[x] = abs(x)$ with increasing frequency (decreasing scale), thus creating a turbulence. The last example (right) shows again a combination of five noises, the four-last used a V-shape transfer function and the first one an inverted holes-like transfer function, thus creating bumps instead of holes. In this case, however, we did not use a sum for the first noise, but blending, which explains that the bumps are smooth and not affected by the turbulence. Blending between two values v_1 and v_2 is obtained using a traditional blending formula $v_1 \oplus v_2 = v_1 * A + v_2 * B$, where in our case $A = 1 - v_2$ and $B = 1$.

Figure 6 shows another example using multiple noises (in this case, three noises). It illustrates how meso-scale structures evolve when progressively modifying the shape transfer function. It also shows a simple example of experimental editor. The latter is based on spline curves. Tangents used for editing are highlighted on the figure. For this example, the basis object is an ellipse. We used a linear opacity transfer function to account for semi-transparency. The amplitude was set important with respect to the ellipse size (third of its size) to create deep deformations and the coefficient r was set to 0 to create important topological changes. The first column shows the result when only the first noise is activated. The second column shows the result when all three noises are activated. Note that because additionnal details are added in this case, the clouds appear bigger.

From a user point of view, multiple transfer functions can be entered using a simple standard GUI without requiring any shader code (as shown here). We adopted a multi-texture pipeline by setting the maximum number of transfer functions and noises to a fixed constant n (here $n = 3$). Each noise can be individually activated by the user by checking / unchecking a button as shown on figure 6. The formula is then evaluated step by step according to whether or not a noise is active. Each step consists in combining the previous result (initially 0) in the pipeline with the next one according to the selected operator. The amplitude and frequency parameters can be entered with sliders (see right part of figure 6), and the operator with a multiple choice button (here $\oplus$). For more flexibility, additional simple conditions could be added such as doing the combination (sum / blending) only if the incoming value is within a given range, where the extremities have been entered by the user. Once the data has been entered by the user, a corresponding pixel-shader code can be generated automatically and then uploaded to the GPU in real-time. For experienced users, it is naturally still possible to propose a simple parser and shader language, that allows him or her to enter an own combination formula instead of choosing among some predefined operators.

Combining different noises and transfer functions allows us to increase the variety of details on surfaces, but the precise control is still restricted. One way to further improve the designing possibilities is to increase the dimension of the transfer function by adding for example a second parameter, which does not depend on noise but on the object itself.

4.2. 2D shape transfer functions

2D shape transfer functions consist in using a 2D table instead of a 1D table. For a 2D table we need a second entry. This entry can be for example related to the object itself, such as its density. We replace the previous computation of k given by formula 4 by the following new formula:

$$k = aT_{2D}\left[\frac{1+noise(\frac{x+rV_x}{f},\frac{y+rV_y}{f},\frac{z+rV_z}{f})}{2},O(x,y,z)\right] \quad (6)$$

Figure 7 shows some 2D shape transfer function examples applied to the soft sphere. In this case, the second entry of the shape transfer function is the density of the soft sphere, as described in formula 6. This density increases as we move towards the center of the sphere. The first row of the figure can be used as reference image: we used a constant 2D transfer function that does not vary with the density of the object. It is thus equivalent to a 1D transfer function. The coefficient r has been set to 1 in order to highlight the variation along the depth of the object (with a 1D transfer function and $r = 1$, there is no variation along the depth). Complex structures can be defined, like mushroom-similar structures for the last example, but yet in an intuitive way (the correlation with the 2D transfer function is straightforward).

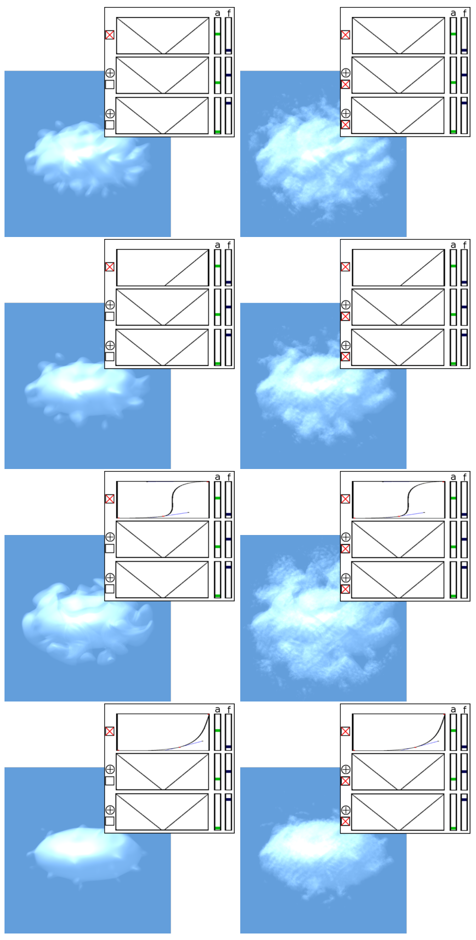


Figure 6: *An example of interactive editing. The four pictures illustrate the effect of editing the shape transfer function using spline curves. For each picture, the left column shows the model deformed using a single noise and the corresponding transfer function (18 fps). The right shows the result obtained by activating two more noises at finer scale and higher frequency (7.1 fps).*

.

As for the 1D transfer function, combinations of multiple noises can be used to furthermore increase the possibilities. Higher dimensioned transfer functions may also be possible, but such functions would be certainly less intuitive. Meaningful additional parameters must first be found. In addition,

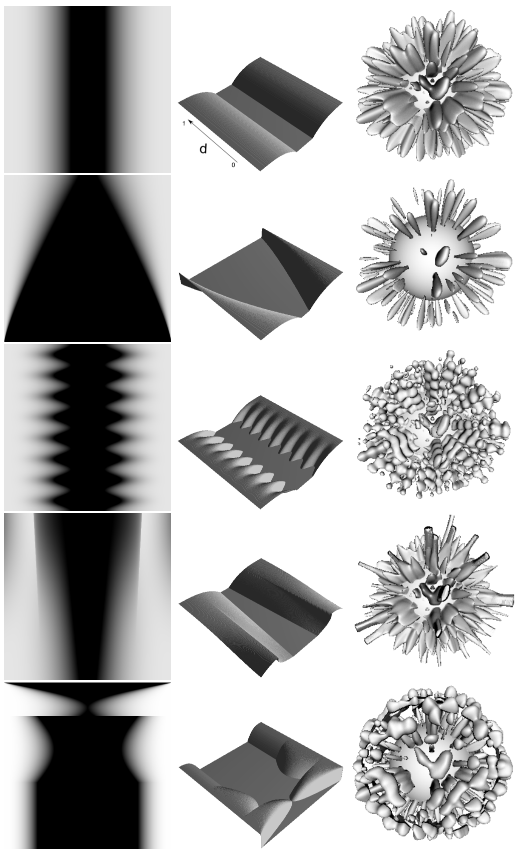


Figure 7: *Examples of 2D shape transfer functions to obtain complex variations along the depth.*

designing a 3D-transfer function in real time and uploading it to the GPU seems to be a complex issue.

4.3. Vector field

With usual hypertextures, the deformation is generally applied along the gradient direction of the density variation or along a noise-perturbed gradient (the latter has been used by Perlin and Hoffert [PH89] to obtain curled hairs for instance). In our case, we use an explicit vector field V. In our framework, this field has a twofold purpose.

Firstly, as for usual hypertextures, it defines a direction by means of a unit vector $\frac{V}{\|V\|}$, which is used to distort the space by shifting points along this direction. The vector field allows one to control the orientation of the distortion. It further allows us to predict how the boundary of the object could be maximally expanded. Indeed, since the shape transfer function $T[]$ returns a value between 0 and 1, the amplitude factor a will represent a sort of a maximal shift value. That is, a point P is maximally shifted by: $P + a\frac{V}{\|V\|}$. If necessary, this allows us to compute a new object boundary, e.g. a frontier beyond which the density is always null, thus improving ray casting. We note, however, that in our case we always oriented the distortion towards the interior of the object, so as to avoid any expansion (this is why we used a minus in formula 3).

Secondly, the vector field is used to determine the actual location at which the noise function is applied, namely on $P + rV$. In the case of the previous soft sphere, we have for example defined a field that shifts all space points exactly on the sphere boundary along the line passing through its center (closest point on the surface). This case is again illustrated on the top of figure 8. The shape of the resulting deformation is correlated to the shape transfer function (in this case it creates peaks). The factor r controls how far points are shifted to evaluate the noise function. When r equals 0 no shift at all is applied. If $r = 1$ the maximal shift is applied.

We note that in formula 3, we used a normalized vector field to shift the object points. This can potentially be problematic for locations where the vector field is null (no normalization is possible). For example, if the field points towards the surface, all points on the surface have a null vector value. In this case, we use a null vector, which has no graphical influence at all, since the points on the surface have a 0 density value (in turn an opacity of 0), and are consequently invisible.

Unlike traditional displacement mapping, where only points on the surface are shifted along the normal direction, hypertextures provide more flexibility as shown on figure 8. The first example (top, left) shows the previously described case: a field that projects points onto the sphere boundary along the radii. The second example (second row, left) shows a field projecting points from a upward-shifted center. The next example (last row, left) illustrates a field with constant vertical direction. Then, we show an example of a noise perturbed field. For this fourth example (first row, right) we used a field pointing towards the border but perturbed by a noise-based random vector field. This makes the peaks trajectories become more random, and even creates branching structures. The fifth example (second row, right) illustrates a curvilinear field making the peaks rotate around a vertical axis. Finally, the last example illustrates the case of the previous field, but with orientation and magnitude perturbed by noise. As outlined by this example, allowing the field not only to follow the gradient and surface normal direction significantly increases the design possibilities. Random perturbations of fields can be furthermore straightforwardly integrated into the shader and do not need to be stored explicitly since they can be based on noise.

5. Interactive Rendering of hypertextured objects

In this section we describe our two interactive splat-based rendering techniques. We recall that splats are used to model the global shape of the object. Both of these methods then

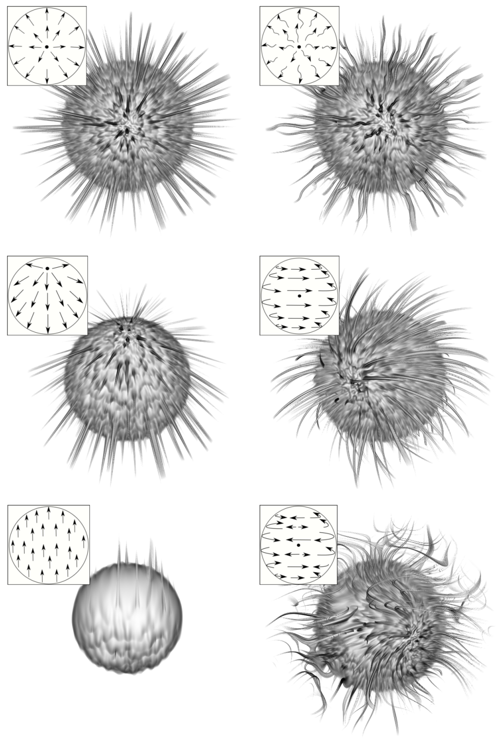


Figure 8: *Influence of the vector field. The latter can be further perturbed by a vector noise to get more complex effects.*

further use hardware-accelerated raycasting to add some procedural volumetric details. The principle of both methods is to compute the contribution of viewing rays emanating from the viewpoint and intersecting the object deformed by the hypertexture. To this end, these methods consider one or several segments along the viewing rays. Along these segments, evenly spaced sampling points are placed and evaluated using our deformation formula, which yields for each sampling point a color and an opacity. These are then combined using a traditional raymarching scheme to compute the contribution of the segment to the final image. The generation and the number of segments are different for both methods, as described in the following two subsections.

We note that we use an illumination based on Phong shading. The latter requires a normal, e.g. the gradient of the density. Since the gradient of the actual deformation induced by the procedural hypertexture is not known beforehand (and cannot be pre-computed), we have to compute it through discrete differentiation during the shading stage. Using central differences, this requires six more evaluations of the hypertexture model and has therefore an impact on the rendering performance for both rendering techniques.

5.1. Using an extended EWA splatting rendering method

The first splat-based rendering method is straightforwardly inspired by the well-known EWA splatting framework [CRZP04], which is naturally suited to render volumetric point-based datasets. EWA splatting consists in associating a 3D interpolation kernel (usually a truncated 3D Gaussian) to each point sample defining the volumetric object. Theses overlapping kernels are used to express the density on any location inside the volume as a weighted sum of the values of neighboring point samples. The resulting image is obtained by computing a weighted combination of the projection of these kernels (2D footprints) onto the screen, e.g. the overlapping footprints are blended in screen space. Because of blending, and in order to maintain a coherent visibility, the point samples must be depth-sorted. To do so in an efficient way, they are usually processed sequentially using a slice-based clipping approach. The choice of the slice depth has an influence on visibility accuracy and on rendering speed.

We can extend this framework to render hypertextured splat-based objects (figure 9) as follows. For each pixel of the 2D footprint, we compute a segment (defined by the entry and exit points of the kernel) and apply a local raymarching along this segment to determine an individual color and opacity, while using the previously presented hypertexture formula. Concretely, we consider for each pixel of each 2D footprint the viewing ray going from the camera center of projection and passing through the pixel. Since this pixel is part of the footprint, the corresponding viewing ray intersects a reconstruction kernel in object space. We consider the intersection of the viewing ray with a sphere B_i centered at the sample location. This yields a segment $p_a p_b$ in object space, with p_a (resp. p_b) being the entry (resp. exit) of the sphere. The subdivision of the segment $p_a p_b$ allows us to apply raymarching and to evaluate the hypertexture. The number of samples n is dependent on the radius of the sphere and a global *quality* setting s defined by the user. This setting has an impact on the quality/performance tradeoff. As of now, s is set to be constant for the whole object and is defined as the *sampling step*, e.g. the distance between two consecutive samples along the segment. s is chosen empirically and can be modified interactively according to the need of the application (from fast experimentation to good quality rendering). We note that the spheres are overlapping in screen space, so more than one segment per pixel might be computed. But the corresponding samples are different for each sphere. This causes an overdrawing, but since all contributions are blended by the Gaussian kernel in screen space, we obtain oversampling with Gaussian reconstruction kernel rather than straightforward redrawing. This yields a natural anti-aliasing and better signal reconstruction along the viewing rays. Therefore visual results are generally of good quality with this approach. Of course, the amount of oversampling depends on how much the spheres are overlapping.

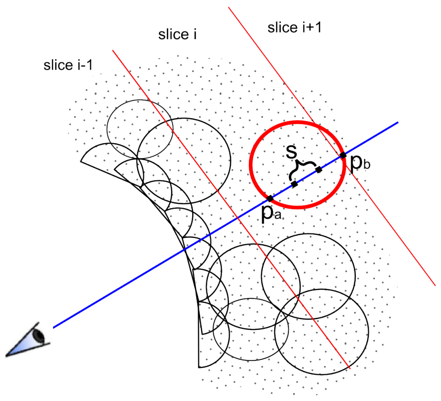


Figure 9: *Principle of the hybrid splat-based volumetric rendering method. Hypertexture details are added with a local hardware ray marching.*

5.2. Using depth peeling for rendering

Our second splat-based rendering method aims at avoiding overdrawing by generating only one sampled segment per viewing ray and pixel. This method relies on the hardware accelerated raycasting method proposed by Scharsach [Sch05]. The principle is to compute the segment using a rasterization process. All splats are sequentially projected to render the global shape of the object, while writing for each pixel into an intermediate buffer two depth values: one for the closest entry and one for the farthest exit point. Both depths then allow us to reconstruct the corresponding segment. But, obviously this technique works only for convex models. For concave objects, we must consider multiple segments along one viewing ray. Indeed, a ray can enter and leave the model several times. We thus propose to use a depth-peeling similar principle [Mam89] as in the recent work of Kanamori et al. [KSN08]. This multi-pass approach processes sequentially each intersection between the viewing ray and the object to obtain sets of consecutive entry and exit points. In our case, this consists in rasterizing the surface splats and storing for each pixel the depths of the n^{th} front-faced and back-faced splat,i.e. during the n^{th} rendering pass, the depths necessary to compute the coordinates of the n^{th} segment. Since our surface splats are oriented (as described in figure 9), we can thus easily determine if the viewing ray is entering or leaving the model by comparing the normal of the surface splat with the viewing direction. To efficiently detect the current segment to process, we maintain a depth buffer storing for each pixel the depth of the last processed segment. Thus, we are able to discard during the rasterization fragments with a depth less than the stored value for this pixel. This naturally discards segments processed during the previous passes. The algorithm stops when no new segment is created, which can be automatically detected using occlusion queries. In practice, we define a fixed number of passes (usually 4), which is sufficient for most models. This second method does not suffer from overdraw and is thus inherently faster than the previous rendering method for a similar sampling step along the ray (e.g. a similar factor s).

5.3. Implementation details

This section describes the GPU-compatible implementation of the rendering methods and some technical details necessary to gain benefits from latest hardware acceleration. For both algorithms, the hypertexture is evaluated at each sample using a GPU implementation of the simplex noise described in Perlin [Gus05, Per01]. Due to the compact definition of hypertextures (related to the procedural nature of noise), the GPU memory usage remains globally low. The model in itself is defined as a collection of 3D vertices and corresponding radius values, and thus can be efficiently stored into a *Vertex Buffer Object*. Furthermore, we need to store onto the graphical memory the vector field and the various transfer functions (1D-2D textures). The vector field is used to compute a coarse displacement direction. Therefore, this vector field does not need to be of high resolution. In practice, it is defined as the RGB channels of a low resolution 3D texture (for all examples presented here, a 64^3 texture). Finally, the density, which varies also only coarsely, is further stored in the alpha channel of this 3D texture.

Each interpolation kernel is represented by a simple OpenGL point primitive. At rendering time, the vertex shader conservatively estimates the size d_i of the projected splat, based on a perspective division of the larger of the ellipse radii by the eye-space depth value of the splat center followed by a window-to-viewport scaling. This causes the single OpenGL vertex to be rasterized as a $d_i \times d_i$ image space square. Each pixel $f_{i,x,y}$ of this square is then tested by a pixel shader to lie either inside or outside of the projected elliptical splat contour. Each pixel inside the splat contour is then, in turn, associated with a color and opacity using a fragment program that is implementing raymarching. As we use a front to back sequential process, segments generated for a pixel with a current opacity of 1 should be discarded. To achieve this, we write into the depth buffer an arbitrary maximal value for completely opaque pixels. Since a depth comparison is made during the process, it automatically discards subsequent (occluded) fragments, thus saving computation time.

6. Results

Our test setup consisted of an Intel Core 2 Quad and a NVIDIA GeForce GTX280 on a 512×512 window. All

		Bunny (7608 splats)		Teapot (4483 splats)		Cube (8000 splats)	
		HQ	IE	HQ	IE	HQ	IE
EWA splatting	No Shading	12.8 fps	30.0 fps	15.3 fps	35.0 fps	13.1 fps	30.0 fps
	Phong Shading	8.5 fps	20.8 fps	9.5 fps	20.0 fps	7.4 fps	13.9 fps
	Phong Shading w/ Shadows	6.3 fps	14.5 fps	8.7 fps	19.1 fps	6.2 fps	13.0 fps
Depth Peeling	No Shading	12.8 fps	> 60.0 fps	17.0 fps	> 60.0 fps	20.2 fps	> 60.0 fps
	Phong Shading	12.2 fps	51.0 fps	15.1 fps	60.0 fps	14.8 fps	48.0 fps
	Phong Shading w/ Shadows	9.0 fps	32.4 fps	12.1 fps	58.1 fps	11.8 fps	20.9 fps

Figure 10: *Comparing both rendering methods. The framerates depend on the sampling step and vary with the desired quality, e.g. between quality images (HQ) and fast interactive exploration (IE). Corresponding visual differences are presented in figure 13.*

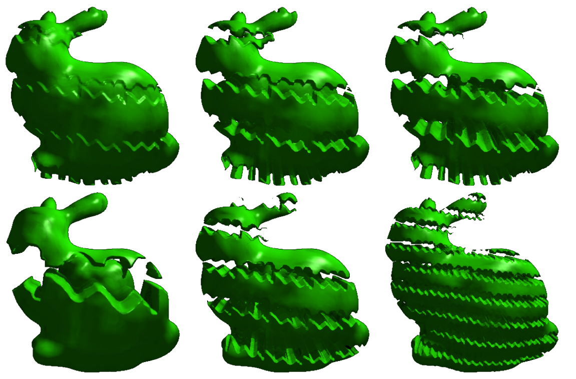


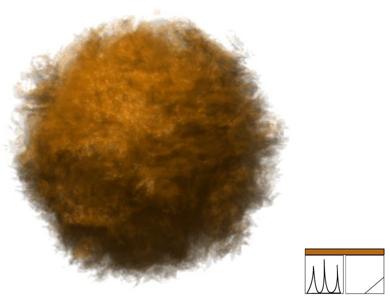
Figure 11: *Influence of amplitude (top) and frequency (bottom) parameters.*

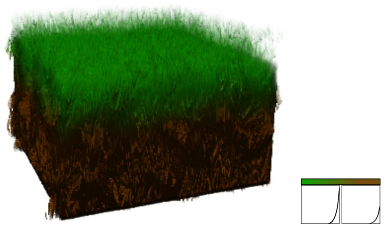
framerates have been measured with the model covering approximately 75 % of the window. Our method uses several parameters allowing a precise and intuitive control of the final result. Firstly, the effect of the amplitude a and frequency f parameters are illustrated in figure 11.

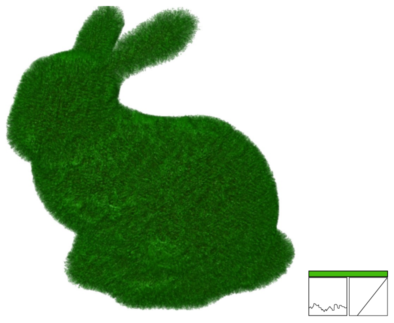
The sampling rate s of the raycasting affects the quality and performance. It allows one to modulate between interactive exploration or better-quality rendering. A table on figure 10 shows the frame rates obtained for different parameters on three objects with the same hypertexture (shown on figure 13), using the two proposed rendering methods and two different sampling rates for different shading techniques with and without shadowing (shadows have been obtained by casting a secondary ray into the light source direction). As illustrated by this table, the framerates decrease with the introduction of a Phong shading process. This is due to the estimation of the gradient, computed by central differences, thus requiring 6 more evaluations of our hypertexture formula on each sample. In the same way, this table shows that the framerates are more strongly dependent on the value of the sampling step s than on the number of splats composing each object. Actually, the main bottleneck of our application lies in the local raycasting technique, and in particular, in the evaluation of noise. The drop in performance due to noise has been evaluated (independently of other parameters) at roughly 65 %. When using a combination of several noises (three), even more than 85 % of the computation time is spent on the simplex noise computation. Our measurements confirm that the main computational cost lies in per fragment processing. These high computations are however necessary to guarantee that the textures can be edited dynamically by users.

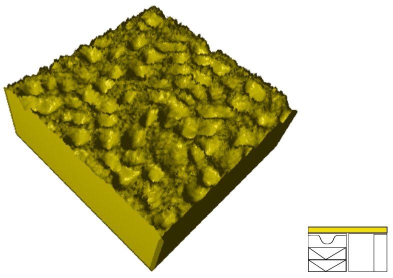
The first and second columns of figure 13 show for both methods a visual estimation of the differences between good quality (HQ) settings ($s = 0.002$) and fast interactive exploration (IE) settings ($s = 0.015$). As can be noticed, the EWA splatting method (top row) is generally slower but tends to produce smoother results. This is confirmed by the last column of figure 13, where we used an identical sampling rate s along the casted rays for both methods. Due to overalpping kernels, the EWA splatting method results in much overdrawing. The framerate is consequently much slower compared to depth peeling although the sampling step is the same. But the kernel convolution introduces a natural antialiasing and smoothing. Even when the sampling step s is decreased with the depth peeling method (e.g. more samples are taken), the result yet remains different from splatting, since no convolution (blending of values) is performed. So beyond oversampling, splatting also acts like a Gaussian filtering and signal reconstruction kernel. We note that since we use for both rendering methods an opacity-based ray termination acceleration technique, highly transparent transfer functions are usually characterized by much lower framerates.

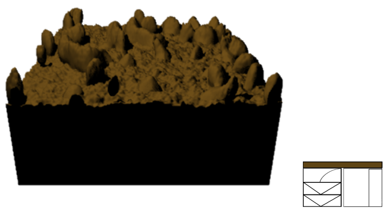
In order to evaluate how well transfer functions, especially the opacity and shape transfer functions, can help to model and reproduce natural effects, we conducted a test series on users previously familiarized with our modeling system. The users have been given the possibility to edit and modify the transfer functions and to change the parameters amplitude a, frequency f and shift r. The 1D transfer

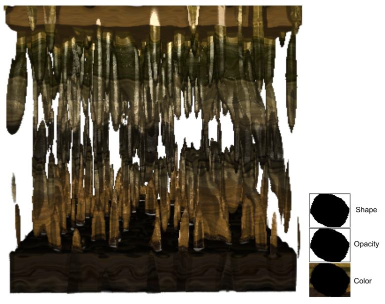

(a) wool, $r = 0.2$, 6.2 fps


(b) lawn, $r = 0.9$, 9.4 fps


(c) hedge, $r = 0.2$, 19.9 fps


(d) sponge, $r = 0.5$, 8.3 fps


(e) clods, $r = 0.6$, 8.0 fps



(f) cave, $r = 0.85$, 6.8 fps

Figure 12: *Result of an experiment concerning the capacity of reproducing natural phenomena by letting users edit transfer functions during about 20mn.*

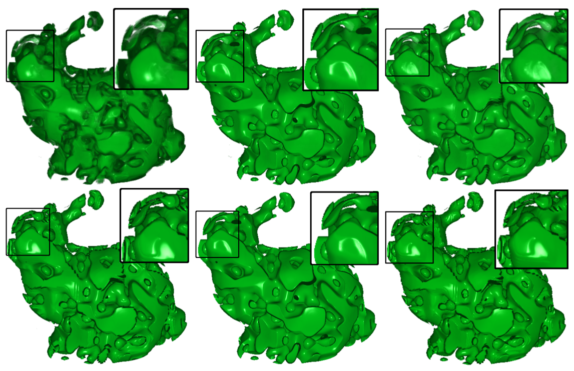
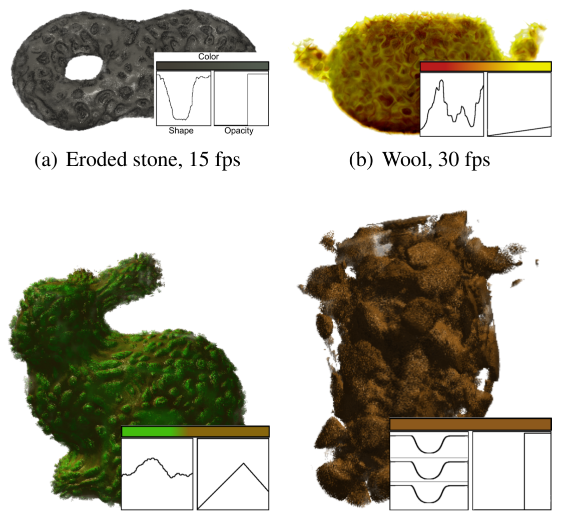


Figure 13: *Visual comparison between EWA splatting (top row) and Depth-Peeling (bottom row). The first and second column correspond respectively to Interactive Exploration (IE, top 20 fps ,bottom 50 fps) and High Quality (HQ, top 8 fps, bottom 12 fps) settings. The last column shows results with the same sampling step used along a ray for both methods (top, 12 fps, bottom 60 fps).*

function editor is based on spline curves and the 2D one on simple paint brushes. The goal of the experiment was to ask them to reproduce some of the natural effects shown in figure 1. Concretely, we asked to reproduce something that looks like wool, a lawn, a hedge, a sponge, clods and a cave. The obtained results after about 20 minutes of free manipulation are shown on figure 12. As outlined by this figure, this experiment demonstrates that transfer functions can indeed help to reproduce natural phenomena without using any programming language. The wool, lawn and hedge have been obtained with a 1D shape transfer function. The sponge and clods were obtained with three noises. Finally, for the cave we specifically asked to use a 2D transfer function, which is well adapted to such a situation. However, 2D functions have turned out to be globally less intuitive than 1D functions. In fact, we experienced that 2D transfer functions are only intuitive when the value r is close to 1. We have also experienced that a shift value of r close to one (at least greater than 0.5) generally allows one to avoid that the object breaks into disconnected pieces. With a longer manipulation, more advanced shading, and additional 3D color textures, these examples could be probably further improved to make them better match the examples of figure 1, but obtaining an exact one-to-one match was not the goal of our experiment.

More results are shown on figure 14. These are purely synthetic. We show them on different shapes to illustrate that our hypertextures can be applied to arbitrary objects. Figure 15 shows that once a hypertexture has been edited, it can be used in software rendering systems. Here, we show an example of the clods of figure 12 applied to a field and rendered using Monte Carlo ray tracing with a synthetic environment map including global illumination effects. Note that because of the procedural nature of the hypertexture, objects of arbitrary size can be textured without repetition effects.



(a) Eroded stone, 15 fps (b) Wool, 30 fps

(c) Moss, 11 fps (d) Mud/Soil, 9.3 fps

Figure 14: *Examples of purely synthetic effects.*

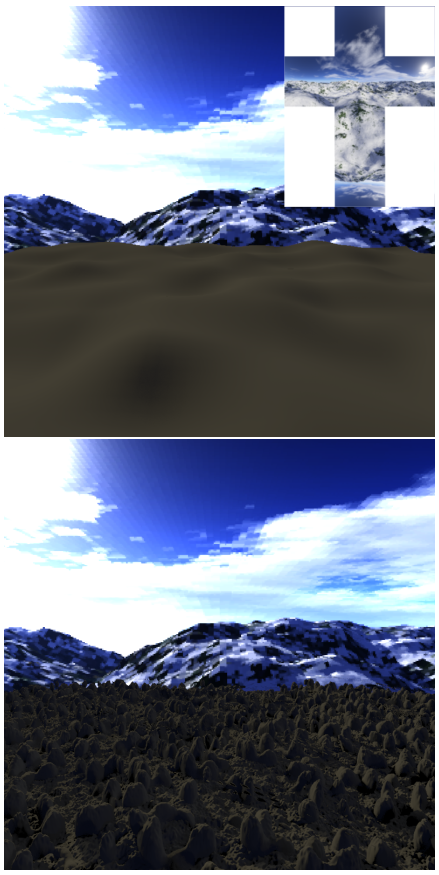


Figure 15: *Once hypertextures have been edited, they can be used in software raytracing with global illumination systems to texture objects that can be very large.*

7. Conclusions

We have introduced a new framework for the interactive exploration, creation and design of hypertextures applied to arbitrary objects. This framework increases the control of results as it has been outlined by our experimental user-study. In addition, rendering is sufficiently fast to allow for interactive manipulations. Our approach is based on a reformulation of the density modulation function, especially the real-time edition of the shape and opacity transfer functions, allows a user to control, along with other parameters (amplitude, frequency, vector field, etc.) the final appearance of hypertextures in an intuitive and straightforward way. One specificity of our approach is that no particular shader programming knowledge is required. Simple curve, patch or image editing tools are sufficient to create the most various effects (wool, grass, clouds, etc.). Because of dynamic editing, it should be possible to further obtain hypertexture animations, for example by making the transfer functions time-dependent, or by using a 4D-noise instead of a 3D one or even by making the vector field dynamically change. This is one of the future works that we would like to explore.

Several other problems need to be tackled in order to further improve the global efficiency of our framework. Firstly, the quality setting of the user, namely the sampling step s defined as the distance between two consecutive samples along a ray, could be better exploited to obtain improved performances. As for traditional volume rendering, an adaptive sampling step would be highly valuable in order to concentrate computations an useful parts. By defining for example this parameter on a per reconstruction kernel basis, it would allow us to increase the quality of hypertexture details on some important parts (parts of the object relevant to its shape, facing the user, or on the silhouette of the object) while rendering other parts with a coarser sampling resolution, thus increasing performance without losing visual quality. Once a given hypertexture has been modeled by users on an object, and it remains "static", we could further imagine pre-computing some data to accelerate rendering.

Some visibility information could be pre-computed to improve empty space skipping, or even the gradient and noise values to avoid their explicit computation. Secondly, concerning shape transfer functions, it would be certainly highly valuable to propose in future a method that would automatically build such a function, by using examples. That is, we could imagine a system where the user designs a sample of the details with traditional 3D modeling tools, and supplies it to the system. The latter would, in turn, analyze this geometry and construct automatically an adequate shape transfer function to recover a similar visual effect. But, as of now, this seems to be an extremely difficult and non-trivial problem, because of the random (e.g. highly non deterministic) nature of noise.

References

[Bli78] BLINN J.: Simulation of wrinkled surfaces. In *Computer Graphics ACM Siggraph annual Conference Series* (1978), vol. 11, pp. 286–292.

[Coo84] COOK R.: Shade trees. In *Computer Graphics ACM Siggraph annual Conference Series* (1984), vol. 18, pp. 223–231.

[CRZP04] CHEN W., REN L., ZWICKER M., PFISTER H.: Hardware-accelerated adaptive ewa volume splatting. In *VIS '04: Proceedings of the conference on Visualization '04* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 67–74.

[CTW*04] CHEN Y., TONG X., WANG J., LIN S., GUO B., SHUM H.-Y.: Shell texture functions. *ACM Trans. Graph. 23*, 3 (2004), 343–353.

[DG95] DISCHLER J., GHAZANFARPOUR D.: A geometrical based method for highly complex structured textures generation. *Computer Graphics forum 14*, 4 (1995), 203–215.

[DG97] DISCHLER J., GHAZANFARPOUR D.: A procedural description of geometric textures by spectral and spatial analysis of profiles. *Computer Graphics forum (proc. of Eurographics 97) 16*, 3 (1997), 129–139.

[DG01] DISCHLER J.-M., GHAZANFARPOUR D.: A survey of 3d texturing. *Computers & Graphics 25*, 1 (2001), 135–151.

[GM08] GAMITO M., MADDOCK S.: Topological correction of hypertextured implicit surfaces for ray casting. *The Visual Computer 24* (2008), 397–409.

[Gus05] GUSTAVSON S.: Simplex noise demystified. In *http://www.itn.liu.se/stegu/simplexnoise/* (March 2005).

[HpS98] HEIDRICH W., PETER SEIDEL H.: Ray-tracing procedural displacement shaders. In *In Graphics Interface* (1998), pp. 8–16.

[KK89] KAJIYA J., KAY T.: Rendering fur with three dimensional textures. In *Computer Graphics ACM Siggraph annual Conference Series* (1989), vol. 23, pp. 271–298.

[KSN08] KANAMORI Y., SZEGO Z., NISHITA T.: GPU-based fast ray casting for a large number of metaballs. *Computer Graphics Forum (Proc. of Eurographics 2008) 27*, 3 (2008), 351–360.

[Lew87] LEWIS J.: Generalized stochastic subdivision. *ACM Trans. Graph. 6*, 3 (1987), 167–190.

[Lew89] LEWIS J.: Algorithms for solid noise synthesis. In *Computer Graphics ACM Siggraph annual Conference Series* (1989), pp. 263–270.

[Mam89] MAMMEN A.: Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Comput. Graph. Appl. 9*, 4 (1989), 43–55.

[MJ05] MILLER C., JONES M.: Texturing and hypertexturing of volumetric objects. In *Volume Graphics* (2005), pp. 117–125.

[Ney98] NEYRET F.: Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (1998), 55–70.

[Ped94] PEDERSEN H. K.: Displacement mapping using flow fields. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), ACM, pp. 279–286.

[Per85] PERLIN K.: An image synthesizer. In *Computer Graphics ACM Siggraph annual Conference Series* (1985), pp. 287–296.

[Per01] PERLIN K.: Noise hardware. In *Real-Time Shading SIGGRAPH Course Notes* ((Ed.), 2001), Olano M.

[Per02] PERLIN K.: Improving noise. *ACM Trans. Graph. 21*, 3 (2002), 681–682.

[PH89] PERLIN K., HOFFERT E.: Hypertextures. In *Computer Graphics ACM Siggraph annual Conference Series* (1989), vol. 23, pp. 253–262.

[Sch05] SCHARSACH H.: Advanced gpu raycasting. In *CESCG '05* (2005), pp. 67–76.

[SG04] STRNAD D., GUID N.: Modeling trees with hypertextures. *Computer Graphics Forum 23*, 2 (2004), 173–187.

[SJ02] SATHERLEY R., JONES M.: Hypertexturing complex volume objects. *The Visual Computer 18* (2002), 226–235.

[WH96] WORLEY S., HART J.: Hyper-rendering of hyper-textured surfaces. In *Implicit Surfaces 96 Eindhoven (The Netherlands)* (1996).

[Wor96] WORLEY S.: A cellular texturing basis function. In *Computer Graphics ACM Siggraph annual Conference Series* (1996), pp. 291–294.