Modeling Animating and Rendering Complex Scenes using Volumetric Textures

Fabrice Neyret
iMAGIS / IMAG , BP 53
38041 Grenoble Cedex 09
France
Fabrice.Neyret@imag.fr
http://w3imagis.imag.fr/Membres/Fabrice.Neyret/index.gb.html

Abstract—
Complex repetitive scenes containing forests, foliage, grass, hair or fur, are challenging for common modeling and rendering tools. The amount of data, the tediousness of modeling and animation tasks, and the cost of realistic rendering have caused such kind of scene to see only limited use even in high-end productions. We describe here how the use of *volumetric textures* is well suited to such scenes. These primitives can greatly simplify modeling and animation tasks. More importantly, they can be very efficiently rendered using ray tracing with few aliasing artifacts. The main idea, initially introduced by Kajiya and Kay [KK89], is to represent a pattern of 3D geometry in a reference volume that is tiled over an underlying surface much like a regular 2D texture. In our contribution, the mapping is independent of the mesh subdivision, the pattern can contain any kind of shape, and it is pre-filtered at different scales as for MIP-mapping. Although the model encoding is volumetric, the rendering method differs greatly from traditional volume rendering: a volumetric texture only exists in the neighborhood of a surface, and the repeated instances (called *texels*) of the reference volume are spatially deformed. Furthermore, each voxel of the reference volume contains a key feature which controls the *reflectance function*, that represents aggregate intra-voxel geometry. This allows for ray-tracing of highly complex scenes with very few aliasing artifacts, using a single ray per pixel (for the part of the scene using the volumetric texture representation). The major technical considerations of our method lie in the ray-path determination, and in the specification of the reflectance function.

Keywords— **volumetric textures, complex geometry, levels of detail.**

I. Introduction

The usual approach to geometric modeling consists of interactively specifying component object meshes or surfaces, trajectories, deformations and materials. This approach is suitable for scenes of low to moderate complexity, but it becomes problematic for complex, natural scenery due to the enormous amount of data to specify. This work is also tedious for the modeler because the data may be repetitive, and requires unnecessarily precise knowledge, such as the position of each grass blade on a hill. Special-purpose procedural tools exist to generate some families of shapes, but these are often inflexible in the control the user has over a model.

More importantly, the usual rendering methods tend to be very inefficient for highly complex scenes because rendering cost is proportional, sometimes polynomially, to the number of primitives. Furthermore, the small size of primitives, often occupying a small fraction of a pixel, creates high-frequency signals that lead to very noticeable aliasing artifacts and which are costly to avoid: there should be at least as many rays launched per pixel as the number of primitives which project onto a pixel. Some recent works propose solutions for walkthroughs at interactive rates, using image caching dynamically [SLS+96] or precomputed [SDB97]. These methods are not well adapted to the high quality rendering of complex scenes poorly structured (e.g. forest), where parallax effects and specular highlights are reluctant to be cached.

Conversely, some dedicated procedural rendering tools such as particle systems [Ree83], [RB85] can be used to efficiently render some complex shapes, at the expense of using a limited repertoire of shading or reflectance models.

Volumetric textures can provide a good trade off between generality and efficiency: the key concept, from the user's point of view, is the mapping of a replicated volumetric pattern onto an underlying surface. Such a pattern encodes a sample of the geometric material to be represented, such as a patch of grass, foliage, fur, etc. This texturing process naturally distinguishes several scales of specification: the local or fine scale is specified by the 3D pattern, the medium scale is given by the mapping function, and the smooth or coarse scale corresponds to the underlying surface shape (as shown in Fig. 1). The family of shapes handled by this approach are objects that sit within a layer in the neighborhood of a surface like a thick skin, representing a kind of repetitive complexity. The approach appears to be a good fit for a variety of natural objects, as illustrated by our results.

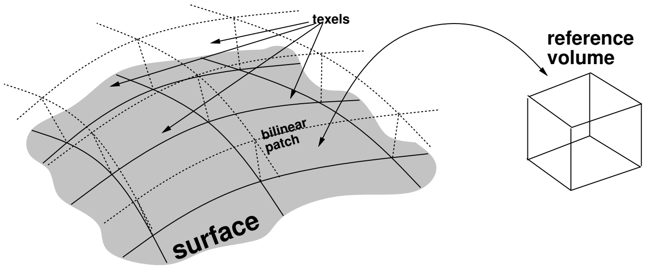


Fig. 1. The mapping of texels.

The seminal work on volumetric textures by Kajiya and Kay [KK89] was matched in their implementation to the modeling of fur. The texture mapping process was restrictive, in that each texel fits exactly on a bilinear patch of the surface. More importantly, the rendering computations were prohibitively expensive because of the volume traversal at full resolution. We review this model in Section II.

The motivations for our representation are introduced in section III. As described in section IV, our representation permits the encoding of any kind of shape in the volumetric pattern (as shown in Fig. 2), and it provides a multiscale representation akin to MIP-mapping which facilitates pre-filtering at different scales (see also [Ney95b]). As explained in Section V, it offers a much less restricted mapping from texture elements onto a surface in a manner that is independent of the mesh subdivision, just as for regular 2D textures (see also [Ney96a]). In this section, we also mention some technical issues such as how to deal with color, and how to decrease apparent periodicities in texture replication. We describe in Section VIII how to convert shapes from their usual (geometric) representation to texels (see also [Ney96a]), and in Section VII how to extend the approach to the animation of complex scenes (see also [Ney95a]). The material in this paper derives from the author's Ph.D. thesis [Ney96b] (in French), and consolidates and expands on the results presented in [Ney95a], [Ney95b], [Ney96a].

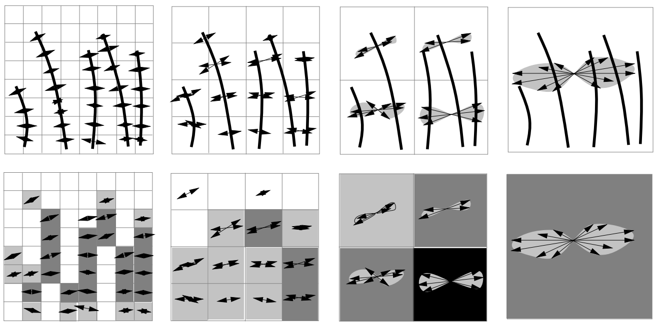


Fig. 2. Sketch of our alternate geometry encoding: 5 blades of grass, encoded at various resolutions (the rendering will choose the one to be used according to the distance). The bottom figures show what is stored: an opacity and a normals distribution function (NDF) in each voxel. The integration of the Phong illumination model on the normals distribution gives the reflectance, so that one can say that the distribution controls the way the light is reflected. For clarity, the top figures retain illustrations of the corresponding virtual geometry.

II. Previous Work

Some of the basic principles of volumetric textures, for which our work can be seen as an extension, have arisen in three lines of research:

- The texture mapping concept applied using an explicit 3D pattern was introduced by Kajiya and Kay [KK89] in 1989, and extended by various authors. The terminology *volumetric texture* and *texel* also derives from this work. We present these early representations in II-A.
- MIP-map texture multiscaling scheme has been introduced for 2D color textures by [Wil83]. We discuss the MIP-mapping approach and its limitations in II-B.
- Local reflectance modeling, especially in the scope of representing the photometric behavior of subpixel geometric scales, has been previously studied for haze [Bli82] and surfaces [PF90], [Fou92], as we will see in II-C.

A. *Early volumetric textures*

Kajiya and Kay introduced volumetric textures in 1989 [KK89] in order to model fur. The key idea was to represent the 3D material (fur) by a cubic reference volume, to be mapped onto a surface (like a thick skin). The copies of the reference volume, named *texels*, are fitted upon the bilinear patches of the surface, and are deformed in order to stick to each other (see Fig. 1). A texel is thus a trilinear deformation of the cubic reference volume. The volume itself is a 3D array of voxels. Each voxel encodes the local geometry (a hair) by specifying an opacity, a frame, and a canonical reflectance function (here the one of a cylinder). As there are nothing but parallel hairs in the volume, the frame and the reflectance function are constant, so that only the opacity is really stored in the volume. The combing of the fur is achieved using texels deformation. The authors present a seminal image of a Teddy Bear, which required a dozen of CPU hour on a IBM3090 mainframe at that time.

Shinya proposed an anisotropic opacity by storing in each voxel the opacity along the 3 canonical directions [Shi92], to be maintained on a multiscale way. However the paper does not explain explicitly how to deal with the reflectance information. Noma applies non deformed texels to encode at various scale procedurally generated trees consisting in sparse facets [Nom95], using a multiscale scheme able to switch from real geometry to texels as a function of the viewing distance.

B. *Mip-mapping and its limitation for relief data*

The MIP-mapping scheme has been introduced by Williams in 1983 [Wil83] in the purpose of filtering 2D image textures, however it can be generalized to other surface attributes. It consists in precomputing the surface attribute data (e.g. a color) at multiple resolutions. The color to be seen in a screen pixel results of the integration of whatever is visible in the cone starting from the eye and passing through this pixel. This cone covers an area on the shape to be rendered, so that the correct shading is obtained by integrating the local illumination equation on this surface. The idea of pre-filtering consists of factoring the illumination equation (usually the Phong model): assuming that only the considered parameter variates within the pixel area and that this parameter appears on a linear form in the equation, one can put the constant term out of the integral, so that one has only to integrate this parameter on the given area. For the color, this means that integrating Phong with the color varying along the surface area which corresponds to a screen pixel is (almost) equivalent to using the average color in one single Phong evaluation. Then the integrals corresponding to different scales are pre-computed, and directly used at rendering time, thus providing a quasi correct image - and consequently anti-aliased - with a single ray per pixel. For the MIP-mapping technique, the area of integration is approximated by its bounding square, and the precomputed scales are the power of two reductions of the texture resolution. When the optimal scale to be used falls between two computed scales, one has to interpolate between the two precomputed images of parameters (i.e. two smoothed color images in the original paper).

Prefiltering in the fashion described above presupposes that integrating the data to obtain a multiscale representation is easily accomplished. That was the case for color or transparency texture with which MIP-mapping is usually used, because one has just to pre-integrate the colors as explained above, as it is a linear parameter of the Phong local illumination model. But if one would want to use a map containing geometrical data such as normals or depth, this would no longer works as these parameters are not linear in the equation (averaging normals or depth would give a smoother relief, which is not the same than a smoother image of the original relief. Typically, a rough surface is darker than a smooth one).

As a remark, the MIP-mapping technique is convenient and easy to program (it is even available on some graphics hardware). It does not provide a perfect filtering however, as it assumes a square filtering kernel, whereas many situations mandate non-isotropic kernels. For instance on the border of a sphere, the texture is compressed a lot in one direction and not in the other one. This implies that more pixels of texture are covered in the compressed direction (the one that vanish behind the sphere) so the kernel should be larger in this direction, and also that the weight of the compressed texture pixels should be smaller. More complicated pre-filtering methods like SAT [Cro84] or NIL [FF88] exist to address some of the deficiencies of the MIP-map filter. Buchanan [Buc91] also specifically deals with the filtering in the 3D case (i.e. volumes of texture). We do not discuss these methods as they does not fit our needs, which are the ability to filter data that is not simply a color, and the necessity of minimal computation at rendering time.

The multiscale reference volume representation we use can be considered in some respects to be analogous to 3D MIP-mapping. However we store a kind of geometrical information in the voxels of our volumetric texture, rather than a color. Thus we are out of the trivial case, which implies that we will have to define how to filter this voxel content.

C. *Reflectance representation*

Kajiya suggests in [Kaj85] to adapt the model which represents the details according to the distance, using explicit geometry for close points of view, a texture for intermediate points of view, and a reflectance model for distant points of view. Thus, the reflectance can be used to represent subpixel geometry: a very small object, such as a single ice crystal in snow, can be observed through its reflective behavior, despite the fact that it is too small to see its shape. As a remark, this implies that polygonal decimation is not a good strategy in the scope of realistic rendering: from a distant point of view, the shape of a corrugated iron sheet looks like a flat sheet, while the illumination is still quite different than that of a flat sheet because of the normals. The same occurs for a rough surface, which geometry is close to the one of a smooth surface, while the reflectance is not (the former being darker). Becker and Max have proposed in [BM93] such kind of transition between geometry texture and reflectance for bumpy surfaces. However this cannot easily apply to free 3D details, which cannot be expressed as height-fields.

In this spirit of representing geometry by reflectance, Blinn has computed the reflectance of a haze made of microscopic spheres [Bli82], Poulin has encoded anisotropic surface with areas of microscopic parallel cylinders [PF90], and Fournier has represented the BRDF[1] caused by microgeometry using a local set of 'Phong peaks' which can be filtered according to distance [Fou92].

In the same spirit, we represent the subpixel geometry by a feature controlling the reflectance, stored in each voxel of the volume[2]. To encode this reflectance, a 4D BRDF table is too expensive in terms of memory, while microprimitives such as spheres or cylinders are too specific. The trade off will be to choose a parameterized family of functions which are generic enough, and require few parameters to specify.

III. Multiscale Generic Volumetric Textures

In this section, we present the motivations for our representation, and we precise our main choices as prepared in the previous sections.

The limitations of the early volumetric texture model [KK89] were:

- hardcoded and specific reflectance functions, e.g. for fur;
- full-resolution volume data, making rendering costly and prone to aliasing;
- strong assumptions about the mesh subdivision;
- no or few descriptions of how to specify the reference volume content;
- no specification of texel animation;
- the representations deal with shapes, but not with colors.

In order to cope with all these issues, and thus to turn the early volumetric textures into an efficient and

[1]The *Bidirectional Reflectance Distribution Function* gives the proportion of energy coming from a given direction that is reflected in another given direction, thus having 4 degrees of freedom.

[2]Kajiya and Kay's texels store an analytical reflectance function. However, this function is only used to represent the microgeometry finer than the volume resolution (while we use it at each scale of the octree), it is not really stored in the voxels as we do but rather common to the whole volume, furthermore this function is quite specific (it simulates the reflectance of a cylinder, as the microscopic data is supposed to be fur).

generic model, we have applied a MIP-map-like multiscaling scheme to this 3D data, and we have extended both the microscopic aspect (the reflectance function) and the macroscopic aspect (the mapping).

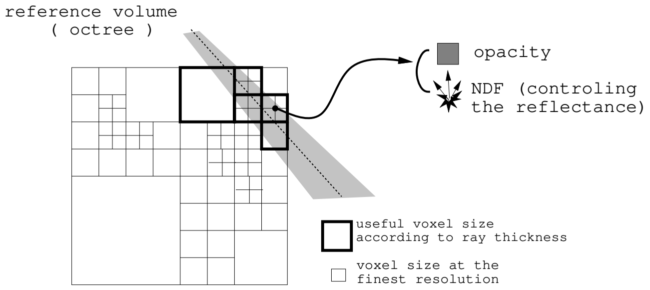


Fig. 3. Reference volume octree encoding. The space is subdivided only where the data lies, at a level depending of the data variations. Coarser resolutions of the data are also stored in the octree. The ray will consider the pyramid level for which voxels fit the pixel size once projected on screen, or coarser voxels if the data is not available at that resolution.

The MIP-mapping scheme consists of precomputing the data at multiple resolutions. For a volume, this leads to an octree structure as shown in Fig. 3. Tracing a ray through the reference volume requires accessing the voxel information at the appropriate level of detail. As for MIP-mapping, this is done by selecting the two levels corresponding to voxel sizes which bound the ray thickness, and then interpolating the two results. Thus the rendering can be compared to a cone-tracing, as all the data within the ray cone is taken into account to compute the color of a pixel. The filtering abilities of the method goes from the scale where one texture pixel projects on one screen pixel (if the viewpoint is closer, the lack of resolution of the texture will be visible), up to the scale where the whole texture pattern projects on a single screen pixel (if the viewpoint is farther, aliasing will occur, and several rays will have to be launched per pixel in such a way that each pattern receive at least one).

Each voxel stores opacity and a feature controlling the reflectance function. For each level in the octree, the information is obtained by filtering (on a way to be defined later) the equivalent information from the level below. For compacity, we want to model the reflectance by a parameterized family of functions. Such a family needs to have a group structure: the filtering of eight functions at low-scale has to be represented by one function of the same family at larger-scale. This is not directly possible for the function used in [KK89], namely the function that simulates the reflectance of a cylinder (because the sum of the reflectance of two non-parallel cylinders cannot be expressed as the reflectance of a mean cylinder). Thus, making the reflectance function more generic is also necessary in order to apply the MIP-mapping scheme.

We describe in Section IV how to choose, render and filter a good reflectance function family, which is the key point of the method, as this allows to concentrate in a voxel the average photometric behavior due to smaller scales, thus making possible the multiscale scheme.

We present in Section V the way to map texels on surfaces the same way than regular 2D textures, without constraining the mesh (thus extending the Kajiya and Kay's [KK89] method, where the mesh is made of bilinear patches). We also deal there with color encoding. Once the pattern encoding and the mapping, the kernel of the representation is complete. This will be the right place to sum-up our volumetric texture specification.

We sketch in Section VI the whole volumetric textures rendering scheme.

To turn the representation into a usable tool, we have to define ways to model the pattern content and to animate the texels. The animation specification is dealt with in Section VII. Thanks to the texturing philosophy, modeling and animation can be described at three different scales: the geometric scale (related to the underlying surface), the mapping scale (related to texels deformation), and the microscopic scale (related to the voxel content), which is the spirit of the hierarchy of details [Kaj85].

We show in Section VIII how to build a volumetric texture pattern from various kind of existing geometric description. Once again, despite the use of an octree which evokes classical volumetric representation, one has to keep in mind that the nature of the voxel content makes the problems quite different.

IV. The reflectance micro-primitive

A. Choosing a primitive

The reflectance primitive to be represented in a voxel simulates the photometric behavior of the subpixel[3] geometry which lies in this area of space, and the opacity coefficient represents the probability for a ray to be stopped. We require a family of reflectance functions generic enough to represent the geometry within a voxel, but having few parameters because they will need to be stored in each voxel. Particularly, it is convenient to define a 2D normals distribution function (NDF) rather than a 4D BRDF. Moreover, normals having a geometric nature, this form can also easily handle modifications of the shape to be represented, such as deformations (one has just to multiply the normals by the transpose of the deformation Jacobian). Using a parameterized family of distributions allows to compress even more the representation. We have chosen to encode the normals distribution by an ellipsoid, i.e. we consider the normals of a given ellipsoid [Ney95b], or to state it another way, we store the ellipsoid which NDF represents well the wanted NDF, instead that wanted NDF itself. The BRDF can be obtained back by integrating the Phong model on this normals distribution, it's what will be done at rendering time. So our 'feature' is this ellipsoid, which purpose is to encode the reflectance behavior of each voxel.

The subpixel (and subvoxel) geometry, that is the part of the scene clipped in the voxel, referred here as the 'local

[3] i.e. it represents the behavior of the voxel content, which scale is chosen to fit the size of a pixel once projected on screen.

shape', is either a small object (e.g. a section of grass) or a surface element (e.g. a facet). An oriented ellipsoid centered on origin has only 6 parameters, allowing us to encode efficiently various kind of local shapes, including planes, cylinders, spheres, etc. We choose to store as parameters the two smallest ellipsoid axis, the third one being defined as orthogonal and having a unit length (the ellipsoid scale carries no information for our purpose). A quadratic form corresponds to this ellipsoid, which can be represented by a 3×3 matrix Q (M is on the ellipsoid if $M^tQM = 1$). The value of the distribution $\mathcal{N}$ in direction N is given from Q by: $f_Q(N) = det(Q^{-1})/(N^tQ^{-1}N)^2$ (see the proof in Appendix 1). We will see that we can define the filtering of this primitive. It is important to remind that our ellipsoid is not a real object (it has no size or exact location, for instance), but only a representation for a local distribution of normals.

To be noted that an ellipsoid is symmetric, while the normal distribution of an object clipped in the voxel may not be, especially for large objects for which the voxel mainly contains a surface element. However the innexisting represented normals are oriented toward the inside of such object, so they will not be visible (excepted on the horizon of the shape at very low resolution, as one can see in Fig. 4). The main limitation of the ellipsoid is of course the few degrees of freedom it has, which limits its acuity in fitting details in distributions: if the normal distribution has two strong peaks (e.g. when summing two facets), the ellipsoid will simulate a single smoothed one in the middle direction. Thus details are blurred, this is the price for a compact representation. We present in Fig. 18 in Appendix 2 the plots of the NDF of various ellipsoids.

B. *Rendering the primitive*

At rendering time, when a ray will go through the voxels according to the global rendering scheme described in Section VI, we will have to cumulate the illumination and opacity of each visited voxel. We define here how to compute this 'atomic illumination' of an ellipsoid attached to a voxel. The illumination is the ratio of energy received from a light source and scattered towards the eye by an object. The ellipsoid we consider here encode the normal distribution $\mathcal{N}(N)$, which induced illumination is $\dfrac{\int_{\mathcal{G}auss} \psi(N,d,L)\mathcal{N}(N)(N.d)\mathbb{1}_{(N.d>0)}dN}{\int_{\mathcal{G}auss} \mathcal{N}(N)(N.d)\mathbb{1}_{(N.d>0)}dN}$ where d is the eye direction, $\psi()$ is the local illumination model (we use Phong) and $(N.d)$ is the visibility[4]. Because the visibility of a normal is the same whether it is taken on the initial shape clipped in the voxel or on the ellipsoid, this is equivalent to simply integrate the illumination of the ellipsoid as if it was the shape to be rendered: illumination $= \frac{1}{S_{ellipse}} \int_{ellipse} \psi(N(M),d,L)(N(M).d)\mathbb{1}_{(N.d>0)}dS$ so that we never need to explicitly use the expression $f_Q(N)$. Thus

[4] $\mathbb{1}_{(exp)}$ is the indicator of the expression exp which is 1 when exp is true otherwise 0.

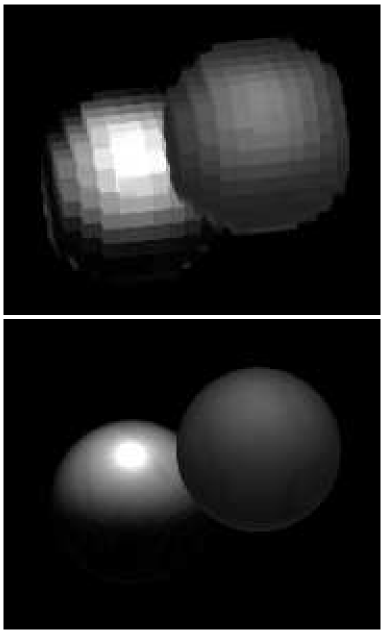


Fig. 4. Rendering of a texel containing two spheres. *left*: resolution 32^3. *right*: resolution 128^3. Note how each voxel reflects the light as if the real geometry were there, because of the reflectance encoding.

we have the illumination of a voxel, which simulates the subpixel geometry as shown in Fig. 4.

Because integrals on ellipsoids seldomly have a closed form, we use a uniformly-sampled numerical integration with 16 samples. The rectangle which bounds the apparent ellipse is easy to obtain: its dimensions correspond to the eigenvalues of the ellipse quadratic form Q', which is obtained from the quadratic form Q of the ellipsoid and the eye direction d by $Q' = Q - Qdd^tQ/d^tQd$. Each sample on the ellipsoid is obtained by solving the second degree polynomial corresponding to the ray-ellipsoid intersection (see Fig. 5) as detailed in [Ney95b].

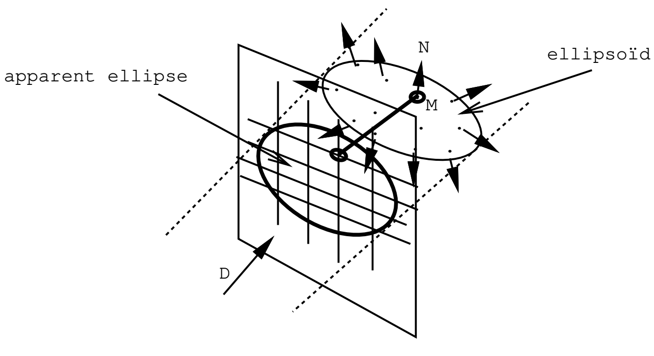


Fig. 5. Numerical integration of the ellipsoid illumination.

We also use the ellipsoid to estimate the variations of the opacity depending on the direction. This anisotropic part is obtained by dividing the apparent surface of the ellipsoid relatively with its mean apparent surface. The real opacity is then the mean opacity (stored in the voxel) times this ratio. The apparent surface is π times the surface of the bounding rectangle computed above. The mean apparent surface is approximated by the average from the 3 mains axis r_i, i.e. $\frac{\pi}{3}(r_1r_2 + r_2r_3 + r_1r_3)$.

The light arriving at the voxel where the ellipsoid stand is obtained by launching a shadow ray that is treated the same way than a regular ray (i.e. it is also a cone). We use a lower resolution, i.e. a rougher level in the MIP-map, by multiplying the computed ray aperture by a user-defined coefficient (usually 2 in our images). This allows to save computation time (shadow rays volume traversal can be very important, especially when the sun is low on the horizon), and also to get easily smoothed shadows, which is better for the image quality.

C. *Filtering the primitive*

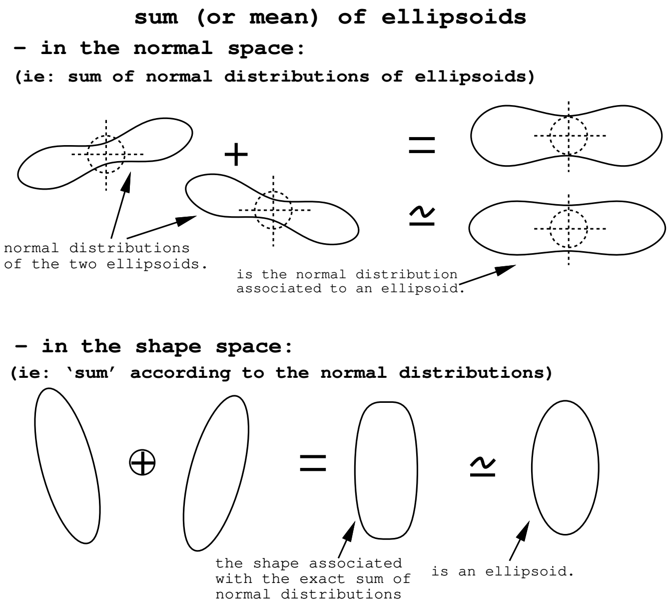


Fig. 6. 'Sum' of ellipsoids in the geometric and in the normal spaces. Maple 2D and 3D plots are available in Appendix 2.

When several child voxels are merged to obtain the average information for their parent in the octree, one has to integrate the voxels contents. This is simple in the case of opacity (we cumulates the values, that represent the mean opacity[5]), but not so for the reflectance. Ideally the normals distributions should be added, but our representation requires that the result should also be encoded by an ellipsoid. We thus need to define the 'sum' of ellipsoids in terms of their normals distributions. This correspond to finding the quadratic form matrix Q such that f_Q is closest to $f_{Q_1} + f_{Q_2}$ (see [Ney95b] and Fig. 6).

By using a first-order Taylor series approximation, we can determine that f_Q is 'almost-additive' with respect to Q^{-1} (we will see below error considerations). We thus define the 'sum' of ellipsoids as the ellipsoid for which the inverse quadratic form is the sum of the inverse quadratic forms of the contributing ellipsoids: $Q^{-1} := {Q_1}^{-1} + {Q_2}^{-1}$. More precisely, the summed ellipsoids are weighted by their opacity. The axis of the resulting ellipsoid are recovered from the eigenvectors and eigenvalues of the quadratic form matrix Q. For two similar ellipsoids separated by an angle of θ, the error obtained with this definition of the sum is $O(\theta^2)$ (i.e. we have $\frac{f_{Q_\theta}+f_{Q_{-\theta}}}{2} - f_{Q_\theta+Q_{-\theta}} = cst.sin^2(\theta)$). This implies than when there are two perpendicular shapes in an area of space, there is a significant loss of information when the whole area is reduced to a single voxel. However the approximation is quite correct when orientations are similar within a neighborhood (e.g. blades of grass in a lawn, or surface elements along a large shape). The plots of the exact and approximated sums of ellipsoids with various angles are presented in Fig. 19 in Appendix 2.

V. Modeling the scene by mapping the texels

A. *Mapping specification*

The early form of volumetric textures by Kajiya and Kay had a restricted mapping. The surface was composed of bilinear patches, which served to define the texel instances. A *height vector* defined on each mesh vertex was used to specify the vertical edges of the 3D patterns, thus allowing the texels to be aligned and 'combed' in a desired direction.

In [Ney96a], we have presented a way to map texels with the same freedom as for regular 2D textures. We attach texture coordinates (u, v, w) to each vertex and as well as to the tip of the height vector[6] (see Fig. 7). The texture space is thus independent to the mesh subdivision, although this complicates the rendering somewhat, as we will see in the next section.

Our model of volumetric texture is thus defined by:
- a cubic reference volume (the 3D pattern);
- a regular surface to be textured;
- a *material* descriptor (setting the colors, cf V-B);
- a height vector on each vertex, which the length gives the thickness of the volumetric skin;
- a mapping function that define the texture coordinates on each vertex at the bottom and at the top of the height vector.

From this point on, we shall refer to a *box* as the piece of volumetric skin above a polygon of the surface and bounded by the height vectors at the vertices. We retain the name *texel* for a copy of the reference pattern. Note that for the [KK89] model, boxes and texels are identical.

B. *Colors*

The reference volume is a purely geometrical representation, and therefore contains no color information. A material descriptor thus needs to be associated with the volumetric texture, which specifies the Phong parameters to be used. This is also what is assumed by the early texels

[5]The apparent surface of ellipsoids controls the anisotropic variation of the opacity, so we expect that the resulting ellipsoid will also represents not too bad the sum of anisotropic variations.

[6]In our implementation the u and v are constant along this vector, so that we specify a dw on the vertex rather than extra coordinates at the top of the height vector.

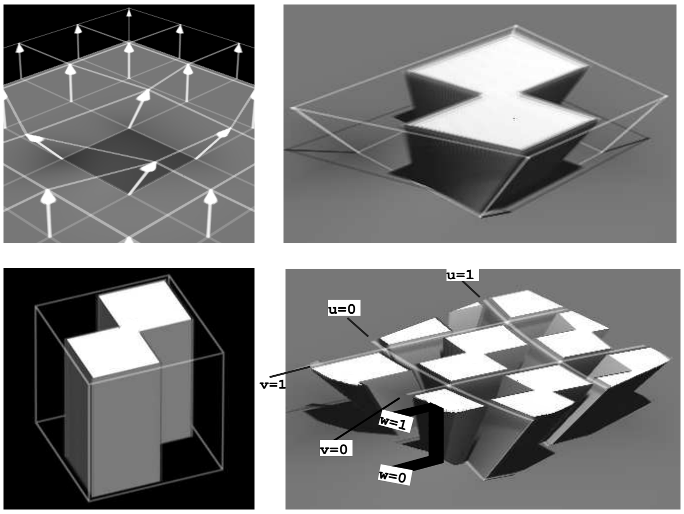


Fig. 7. *Top left*: the boxes corresponding to each faces, delimited by the height vectors on the faces vertices. *Bottom left*: the reference volume used. *Top right*: Kajiya and Kay's mapping: one texel fits exactly to one face which is a bilinear patch (for clarity we have only mapped the middle box). *Bottom right*: our general mapping defined by the (u, v, w) at the eight box vertices, or at the six ones if the face is a triangle (only the middle box is mapped). We figure isovalues of u, v and w.

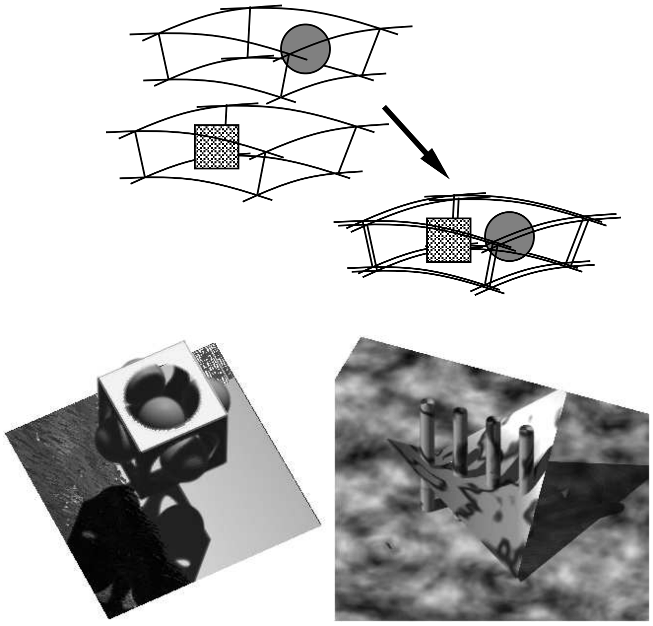


Fig. 8. *Top and left:* merging of two volumetric patterns with different colors. *Right:* one texel textured with Perlin noise (the surface is textured with another one). The texels resolution is 128^3. The surfaces are not part of the texels. As a remark, the distorted square inside the right texel is obtained by using w values that are not parallel to the underlying surface.

models [KK89], [Shi92], [Nom95]. However, it may be not sufficient to have only one material within the volume: the user may want to associate different colors to different components of the 3D pattern stored in the reference volume, or even to use some shades of color inside the volume.

We address this problem in two ways. First, we allow for the superimposition in the same space of several patterns, each associated with their own material (e.g. green grass and red flowers). Second, we can also associate classical color textures with the texels, such as a projected picture or a solid procedural noise, which define the color at any location in space. These two modalities are illustrated on Fig. 8.

C. *Mapping jittering*

Volumetric textures tend to look too much regular if mapped using a simple mapping function. In order to introduce some variations, the mapping has to be jittered. This can be done in one of several ways, depending on of the constraints occurring for the 3D pattern.

If the texture pattern is continuous (torus topology), one is restricted to applying a continuous perturbation, such as that obtained by jittering the texture coordinates, the layer thickness, or the combing (i.e. the height vectors).Perlin noise is well suited for this purpose as shown in Fig. 9 top. If the pattern contents do not cross its bounds, there are few constraints: one can change the pattern content or color, or rotate move or scale the pattern, etc. This is illustrated in Fig. 9 bottom.

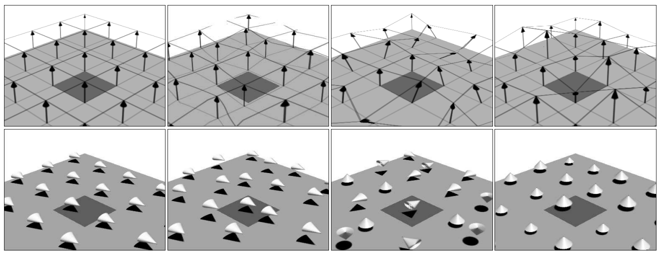


Fig. 9. *Top:* continuous perturbations: jittering of texture coordinates, vectors direction, vectors length. *Bottom:* discrete perturbations: displacing, rotating, scaling.

VI. Rendering scheme

We describe here the global rendering scheme, which computes the color or each ray, possibly traversing a volumetric texture layer mapped upon a surface of the scene.

For a classical texture, three scales are involved in the rendering process:

- the scene scale determines the surface a ray hits and its point of intersection (called *hit point*);
- the texture scale determines the texture coordinates of the hit point, from its local coordinates within the hit polygon;
- the local scale computes the color according to a local lighting model such as Phong, using the parameters obtained from the texture for this point.

For the volumetric textures, this scheme becomes a bit

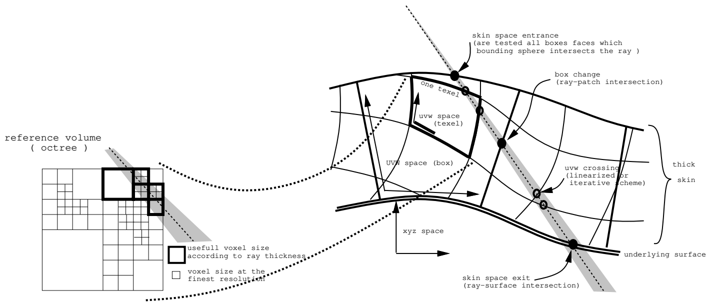


Fig. 10. Ray traversal inside the volumetric layer, traversing the boxes, then the texels, then the voxels at the adapted size. The mapping of the texture space inside the volumetric skin is controlled by the (u, v, w) values at the boxes corners.

more complicated as one obtains a 'hit segment' within the volumetric skin rather than a hit point. In Kajiya and Kay's implementation, the local coordinates of the two ends of the 'hit segment' are directly used to transpose the segment of ray inside the volumetric pattern space, along which the local illumination is collected. Because of the high number of voxels in the volume, a stochastic sampling was done along the ray.

In our implementation (detailed in [Ney96a]), the key rendering steps can be summarized as follows (see Fig. 10):

- the scene level ray-tracing gives an intersection point on the top surface of the volumetric layer, which serves as an entry point of the ray cone into a box;
- the box out point is determined (for a triangular face the associated box is a prism, which the 5 faces are tested), and will become the entry point into the next box along the ray (if there is one);
- the entry and out points local coordinates are converted into texture coordinates, and the ray aperture is used to compute the ray thickness within the texture;
- the texture space (defined by the (u, v, w) values at the corners of the box) is tiled with repeated copies of the pattern, and is thus traversed using a Bresenham-like algorithm along the (u, v, w) space[7];
- we are now in the reference pattern; the ray thickness is used to select the appropriate resolution in the octree (more exactly the two bounding resolutions, for which the results of the voxels shading are interpolated). The reference volume is traversed along the ray as for classical volumetric rendering[8] until the out point is reached or the cumulated transparency becomes opaque;
- for each voxel, the local illumination is computed using the reflectance function encoded by the local ellipsoid, on the way explained in Section IV-B. The local color (using Phong material parameters) and the local transparency are accumulated with the ray's current color and transparency.

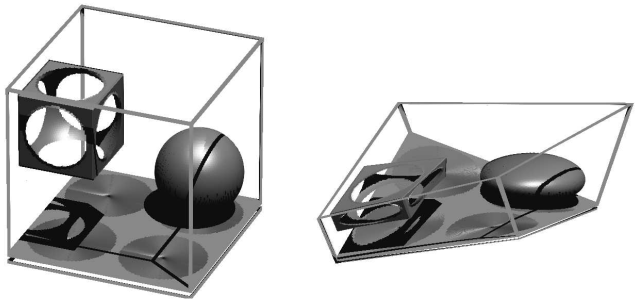


Fig. 11. The deformation affects not only the shape but also the reflective properties. The illumination is not simply deformed as if it was painted onto the surfaces, because the normals transformation is not the same as the points transformation.

Another important feature, not explicit in [Ney96a], requires careful attention. The normals distribution function which is encoded by the ellipsoid in each voxel cannot be used directly in Phong, because of space distortions due to the mapping. The illumination of a deformed shape **is not** equal to the deformation being applied after the illumination has been computed in the undeformed space. This is illustrated in Fig. 11. This means that the normals encoded by the ellipsoid in the texture space (i.e. the undeformed space) cannot be used directly to compute the illumination. We must *first* deform this local normals distribution according to the mapping and *then* integrate the Phong model over the distribution [Ney95c]. To do so, we need the Jacobian J of the mapping transformation, which consists of a transformation from world coordinates to local coordinates and a transformation from local to texture coordinates. The second deformation is trilinear, so it is easy to compute. The first is the reciprocal of a trilinear deformation, and is thus more costly (an iterative scheme is needed). To avoid an excessive computational cost, we evaluate this Jacobian only once per box. The quadratic form 3×3 matrix Q to use for the local rendering as described in Section IV-B is thus $Q = B^t B$, were $B = SFJ$, with $S = \left(\frac{1}{r_i}\delta_{ij}\right)_{ij}$, $F = \left(\frac{1}{r_i}R_{ij}\right)_{ij}$, $r_i = \|\vec{R_i}\|$, $\vec{R_i}$ being the ellipsoid axis. δ_{ij} is the Kronecker symbol which is 1 if $i = j$ otherwise 0.

VII. Animating volumetric textures

Scenes are potentially animated along the time. Specifying the animation of a complex scene may be as tedious as its modeling because of the large amount of components. In this section we propose a convenient scheme for animating complex scenes modeled using volumetric texture, following the same multiscaling approach [Ney95a].

One can imagine several ways of animating the volu-

[7]To be noted that the ray should become curved once in texture space. As long as the curvature is not too high, one can considers that the ray is still a straight line (as for all the previous volumetric texture methods). Otherwise, one can take into account this ray curvature, at least at the box level, by numerically solving the intersection between the ray and an iso-u (or iso-v or iso-w) of the texture (see [Ney95c]), which converges very quickly as the curvature is never very high. Taking also into account the ray curvature inside one texel (i.e. a tile) would increase the cost too much.

[8]The octree structure allows to proceed a recursive traversal, rather than a step-by-step one, by determining for each visited voxel which child to visit.

metric textures, as shown in Fig. 12. For a deformable surface such as cloth, the texels layer naturally follows the deformed surface (using the surface normals as height vectors for the texels). Fig. 15 shows a metallic flag animated using this method.

For a rigid surface, explicit motions can be generated by a force field acting on the height vectors, thus deforming the texels. The force field can be for instance a Laplace field [WH91], a stochastic flow [SF92], or a mass-spring [TF88] network connecting the tip of the height vectors. Fig. 16 left shows a lawn under the wind animated by this way (the force field we use is a mix of a moving periodic function and a stochastic function).

Successive steps of a simple motion, such as oscillations of leaves in foliage, can be selected and encoded in few distinct volumes, cyclically used over time on a cartoon-like way.

These three methods correspond to three different scales, much like the three scales used to model complex scenes: the geometric scale, the mapping scale and the texture pattern scale. A realistic complex object animation would probably need specifications at these three scales: the fur on the skin of a running animal follows the deformation of the animated body, it also makes waves according to acceleration and inertia, and it also likely that hairs locally oscillate inside the texels. Similarly for a tree in the wind, the surface of each main branch is geometrically deformed and consequently the associated bough, foliage waves with the wind, and leaves locally oscillate inside the texels. If the user considers that the hairs (or the leaves) should look the same all along the object (beyond combing and color that can differ), a single texel can be used per time step. The local oscillation being approximately cyclical, this can be encoded in few precomputed texels. The results of these animations are detailed in Section IX and in [Ney95a].

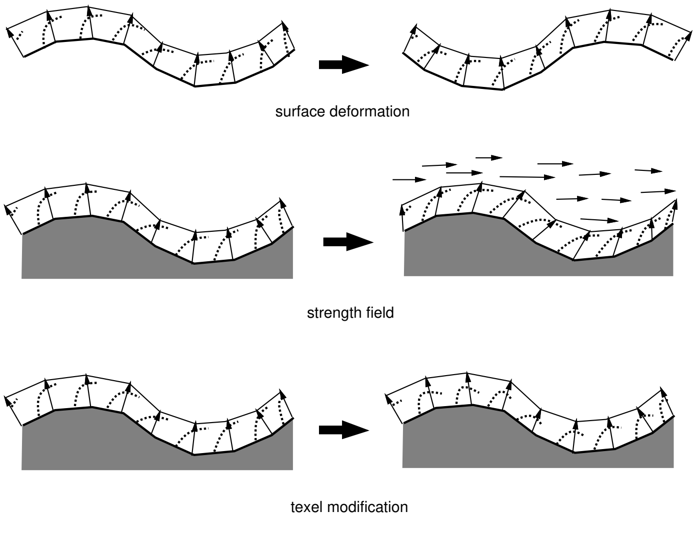


Fig. 12. The three ways of animating texels, which are also three scales of animation specification to be mixed.

VIII. Modeling a volumetric texture pattern

To design the 3D pattern, it would be convenient to use an existing modeling tool then to convert the result into a volumetric texture. However, this task is not a simple voxelization in the usual meaning of this word. Since in our case each voxel contains an opacity and a normals distribution function.

Conveniently, most existing representations can be considered primitive-based. It is quite easy to define a simple primitive by an implicit function corresponding to the distance to its surface. We can also choose this distance function such that it is negative inside the shape. Thus we can determine if a voxel is occupied (and how much) according to the value of the distance function in its area, and we can determine the normals from the gradient of this function. A Warnock-like algorithm can thus draw the shapes in the octree: if the distance function at a voxel center is more than the voxel radius, the voxel is clearly inside or outside the shape, otherwise the voxel is splitted and each child is tested. For instance, a sphere $\{C, r\}$ is obtained with the distance function $d(M, sphere) = |\overrightarrow{MC}| - r$, which gives the amount of inclusion criterion $\frac{1}{2}(1 - \frac{d}{r_{voxel}})$ that is greater than 1 if the voxel is outside the shape, less than 0 if the voxel is inside, and in the interval $[0, 1]$ if the voxel contains the frontier (see Fig. 13). If the maximum resolution is reached, this value is stored as the opacity[9]. As a remind, this is a mean opacity, since the ellipsoid that

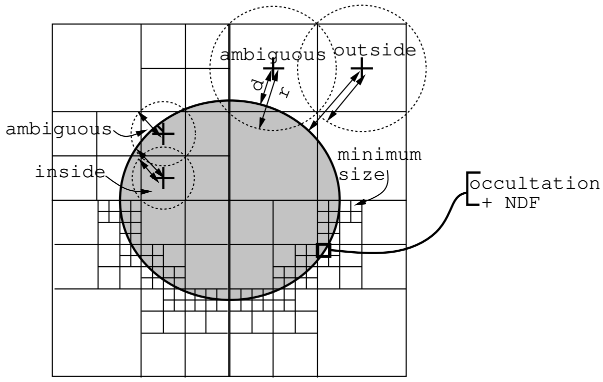


Fig. 13. Recursive volumetric construction of a shape (figured in 2D)

models the NDF will encode the variations depending on the viewpoint. Since normals are given by the gradient of the distance, the ellipsoid can be obtained by fitting its distribution of this gradient inside the voxel. For efficiency, in our implementation we only consider the value of the gradient at the voxel center, which will be used as one of the axis, and we build the two other axis from an estimated curvature of the local surface (we know what is the kind of the primitive currently drawn).

CSG, polygonal meshes, implicit functions, particles sys-

[9] Strictly speaking, this is a density of occupation. We suppose it approximates not to bad the average ray occultation. Too small values for very thin objects are avoided thanks to the bias of our sampling, such as testing only the distance to the voxel center.

tems and L-systems [FvDFH90], [RB85], [PLH88] can be considered as primitive-based techniques: these representations design a shape by combining simple shapes. Respectively, these primitive shapes are a solid, a facet, a skeleton element, a trajectory segment, and a terminal symbol. For a mesh, for instance, the distance function is simply the distance between the voxel center and a triangle. We have detailed in [Ney96a] which distance function to use in each case (see Fig. 14 top).

Some other representations are truly volumetric, such as MRI scanner data and hypertextures [PH89], for which all the voxels have really to be parsed. Fig. 14 bottom illustrates these cases.

Once a shape is drawn in the octree at the highest resolution allowed, a filtering pass is proceeded afterward in order to compute the multiple lower resolutions of the octree [Ney95b], using the filtering method described in Section IV-C. Finally, a compression pass is done in order to clean up the octree, deleting empty leaves and constant areas (that are well represented by the coarser levels). In total, the construction of the octree needs a few seconds.

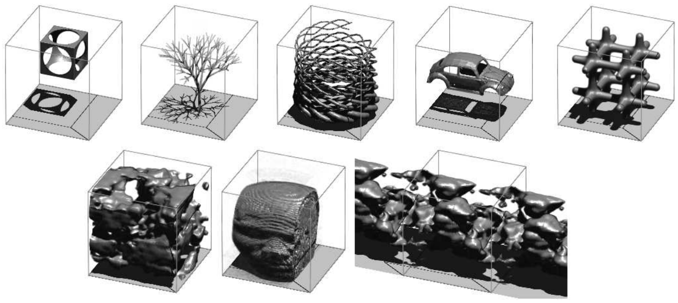


Fig. 14. *Top:* texel conversion of: CSG, L-system, particles systems, mesh, implicit surface. *Bottom:* texel conversion of hypertexture (left) and tomographic image (middle); cyclic hypertexture pattern (right). Texels resolution is 128^3, excepted for the car for which it's 256^3. The memory is 1.3 Mb for the tree, 6.6 for the rope, 6.5 for the car (it was 11 before the compression pass). The crane initial data resolution is $64 \times 64 \times 96$. The frames are part of the texels content, the floor not.

IX. Results

Our implementation has been used to implement several animations of very complex scenes having up to hundreds of millions of primitives. They were ray-traced at video resolution (768×576), with little or no apparent aliasing, despite the fact that only a single ray per pixel was used. The computation time was 20 minutes per frame on average for an SGI Indy with a 200MHz R4400 processor. It variates from 10 minutes for the flag animation up to almost 1 hour for the last forest, because of the numerous shadow rays due to the low elevation of the sun (these shadow rays that cross again the texel layer should be factored, as a lot of them have the same path). The other forests needed about 20 minutes. 60 to 70 frames were computed for the first animations which are cyclical, and respectively 409 and 323 for the two forests fly-by. The images and the mpeg animations can be viewed at the URL `http://www-rocq.inria.fr/syntim/research/neyret/index-eng.html` .

The first scene is a waving flag constructed using a fine lattice of scaffolding, as shown on Fig. 15 left. The texel containing the scaffolding pattern is shown on the right. The flag is animated using a non-linear mass-spring model[10]. The texel layer thickens the flag, and continuously follows its deformations.

The second scene is a lawn, shown in Fig. 16 left, covering a hill made of 1400 bilinear patches. The mapping is made irregular by jittering the height vectors. Each texel contains 16 blades of grass, and a few of them also contain a flower. Two texels are generated, one with the blades (5 Mb) and one with a flower (0.25 Mb). The first is mapped everywhere (the pattern is cyclical), while the second is superimposed at random locations, using different colors. In the complete scene there are 22000 blades and 700 flowers. The blades are generated using parabolic trajectories similarly to particles systems, with the cross-section having a 'V' shape. A 128^3 resolution is used for the reference volume. The motion is generated by an animated force field acting on the normals. This field $\vec{F}(M,t)$ models a gust of wind combined with a random jittering, translated by the wave propagation equation [Ney95a].

The third scene shown in Fig. 16 right represents 512 sparse trees in a flat field. The trees are modeled using 6 iterations of an L-system, yielding 2154 branches and 6336 leaves. The reference volume contains one isolated tree. Because the camera is very close to some of the trees, we have employed a 512^3 resolution. Fortunately the volume compresses by more than 99.9% thanks to octree representation that does not encode empty space. A single tree model is used, and is instantiated in different versions by changing the size, the orientation, the position and the material (i.e. the color). Two texels are generated, one for the trunk and one for the foliage. Note the continuous transition between the distant and nearby trees.

The fourth example, shown in Fig. 17 left, is a forest covering mountainous terrain. 25000 trees are mapped on a surface with 1404 bilinear patches. The forest is dense and continuous, so the 'forest texture pattern' cannot contains only a single isolated tree: the reference volume has cyclic contents, consisting of two trees clipped along the edges of the cube. Texture coordinates and height are slightly jittered (not enough, indeed). The trees are observed from a distant point of view most of the time, but the camera sometimes gets closer, thus mandating a 256^3 resolution for the reference volume. Once again two texels are generated, one for the trunk (11 Mb) and one for the foliage (18 Mb), in order to tune independently the materials (i.e. the color). The animation is a fly-by through the valley, passing very close to the trees when turning back. Note that the scene contains around 200 millions primitives (branches and leaves), reproduces fine shadows, gives smooth transitions while zooming, with very little aliasing, using a single

[10]Thanks are due to Xavier Provot [Pro95] for his flag animation model.

ray per pixel.

The last scene shown in Fig. 17 right, contains the same forest pattern applied on another terrain, using more jittering. An important issue for the resulting look is that the surface[11] is made of 12210 triangular faces on which 253 trees lie. On the previous image a patch was covered by about 18 trees, while here there is one tree per 48 triangles. Thus in the first case the perturbations defined at the patch corners were smoothly interpolated along the trees inside the patch, while in the second case the perturbations vary inside a single tree because mesh vertices are dense enough compared to the size of the pattern to precisely define the texture space. This illustrates that with enough perturbations, any visible characteristics of a regular pattern can be made to vanish.

X. Conclusion

The volumetric texture model that we have presented in this paper is an efficient representation for complex repetitive scenes. Using this representation, it is possible to ray-trace scenes containing hundreds of millions of primitives in a time (20 minutes) for which usual ray-tracing techniques achieve the rendering of simple scenes. Moreover the resulting images show very little aliasing artifacts. Compared to the ray-tracing of classical (i.e. geometrical) data, we save orders of magnitude in computation once the ray is in the skin layer, by gaining on 3 issues:

- no oversampling at all to achieve, as the data is preprocessed;
- no ray hit to test and to sort, as the data is totally orderer;
- no intersection to compute, as at the finest stage the voxel data is no longer of geometric kind.

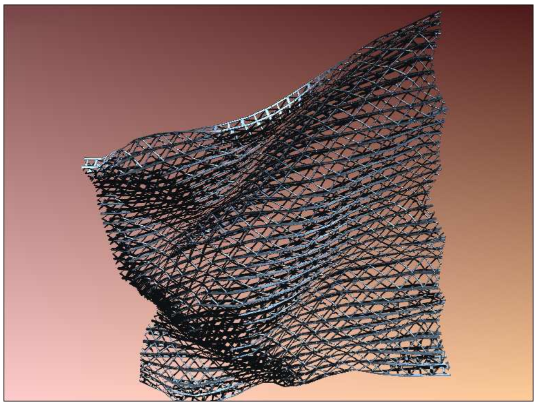
One may also mention that time can also be saved on shadows by using a coarser resolution, which moreover has the nice effect of smoothing them. On the other hand, the local shading is more costly (including 16 Phong evaluations).

The model is also a convenient tool to design and animate complex repetitive scenes, using distinct scales of specification. The modeling scheme is quite similar to regular texture mapping: the user first models an underlying surface, then builds a texture pattern that is a cubic sample of the 3D material to be mapped, and lastly defines a mapping of the texture upon the surface. The fact that the texture pattern is a 3D shape rather than a color image introduces three more issues: the design of the geometry in the reference volume, its color, and the extra degree of freedom provided by the 'combing' of the texture. The scene and animation design is thus greatly simplified, as long as the constraints inherent with mapping approaches and space deformation tools apply to the user goals.

We provide two key contributions with respect to Kajiya and Kay's early volumetric texture method. First, our reference volume representation allows for the rendering efficiency stated above. Second, our tool is complete, leading to the workflow summarized in the previous paragraph. Our representation is multiscale, and contains an opacity and a reflectance function in each voxel at each level of the octree. This reflectance function is encoded by a normals distribution function, itself encoded by an ellipsoid represented by 6 parameters. We have explained how to filter this data, and how to compute its illumination. At rendering time, the appropriate resolution of the octree is used. Our method has fewer mesh restrictions, and allows the mapping specification to use regular texture coordinates. It provides solutions for animating the complex object, to bring shapes into the reference volume, and to specify the colors.

Complex repetitive scenes are quite easy to model and relatively inexpensive to render in this framework. Animations are affordable to compute, even for a simple test. This may also alter the user behavior: complex layers can now be used quite systematically upon surfaces, and numerous trials can be performed to tune complex scenes. An interesting property is the approximately constant-time rendering of volumetric textures: distant points of view show numerous texel instances with low resolution, while near points of view show few texels in full detail. This is linked to the idea that the rendering cost now corresponds to the *visual* complexity, rather to the *geometrical* complexity.

[11] Thanks are due to Pascal Fua for his real terrain data.



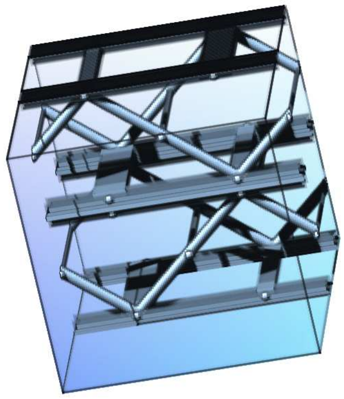
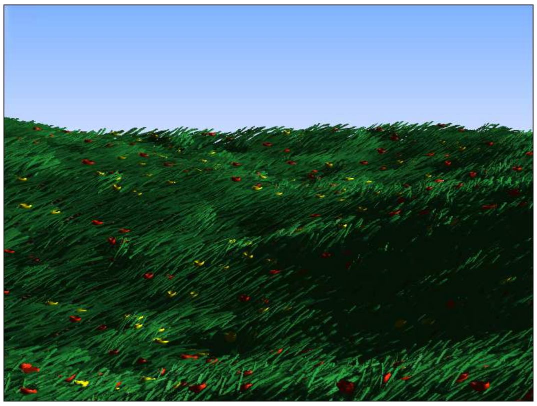


Fig. 15. A scaffolding flag. *Right:* The corresponding texel.



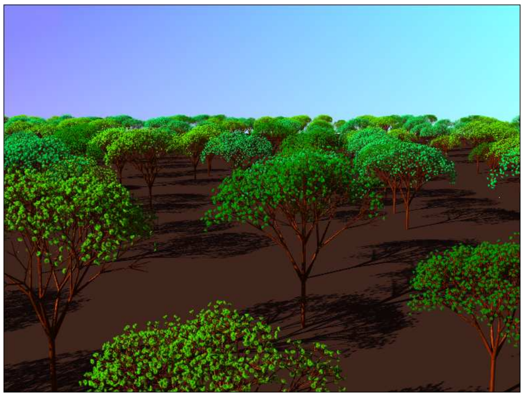


Fig. 16. *Left:* a lawn under the wind. *Right:* an orchard.

Fig. 17. Two forests.

XI. Appendix 1

We sketch here the proof that the normal distribution of an ellipsoid centered at origin and characterized by a 3×3 matrix Q is $\mathcal{N}(N) = f_Q(N) = \frac{det(Q^{-1})}{(N^tQ^{-1}N)^2}$.

By construction, a point M is on the ellipsoid if $M^tQM = 1$.
The normal to the ellipsoid at point M is got from the gradient of the bilinear form M^tQM, hence

$$N = \frac{QM}{\|QM\|}$$

So $N^tQ^{-1}N = \frac{1}{\|QM\|^2}$, which implies that $QM = \frac{N}{\sqrt{N^tQ^{-1}N}}$ and $M.N = \sqrt{N^tQ^{-1}N}$
Thus we have

$$\frac{Q^{-1}N}{\sqrt{N^tQ^{-1}N}} = M \tag{1}$$

Another remark is that for any two vectors v and w,

$$Qv \times Qw = \frac{Q^{-1}}{det(Q^{-1})}(v \times w) \tag{2}$$

where $\times$ is the cross product.

By definition $\mathcal{N}(N) = \|\frac{\partial M}{\partial S}\|$, N being associated to a surface element dS on the Gaussian sphere, and M depending of N (i.e. M is the location on the ellipsoid where the normal is N). If N is indexed by the polar coordinates θ and ϕ on the Gaussian sphere, one has

$$\frac{\partial M}{\partial S} = \frac{\frac{\partial M}{\partial \theta}d\theta \times \frac{\partial M}{\partial \phi}d\phi}{\cos\phi d\theta d\phi} \tag{3}$$

Using 1, we can observe that

$$\frac{\partial M}{\partial x_i} = \frac{1}{\sqrt{N^tQ^{-1}N}}(Q^{-1}\frac{\partial N}{\partial x_i} - (\frac{\partial N}{\partial x_i}.M)M) \tag{4}$$

Thus by expanding $\frac{\partial M}{\partial \theta}$ and $\frac{\partial M}{\partial \phi}$ in 3 using 4, then factoring Q^{-1} using 2, one get[12]

$$\frac{\partial M}{\partial S} = \frac{N}{det(Q)(N^tQ^{-1}N)^2}$$

As $\|N\| = 1$, by computing the norm $\|\frac{\partial M}{\partial S}\|$ we get the result $\mathcal{N}(N)$

XII. Appendix 2

XIII. Acknowledgements

The work described in the paper has been done during my PhD in the Syntim Project at INRIA institute, France. The paper has been written during my postdoctoral in the Dynamic Graphics Project at Toronto University, Canada. Thanks are due to Michiel Van de Panne and to Eugene Fiume for their patient rereading of the paper.

[12] At some point, one has to recognize that $(\frac{\partial N}{\partial \phi}.M)\frac{\partial N}{\partial \phi} + \frac{1}{\cos(\phi)^2}(\frac{\partial N}{\partial \theta}.M)\frac{\partial N}{\partial \theta}$ is $N - (N.M)N$

Fig. 18. Normals distribution function (NDF) of various ellipsoids. The radius are, from top to bottom: $1{\times}1{\times}2$, $1{\times}2{\times}2$, $1{\times}2{\times}4$. To be noted that the absolute size has no meaning: each distribution has to be normalized so that the integral is 1 (thus the NDF can be seen as the probability to get a normal in a given direction).

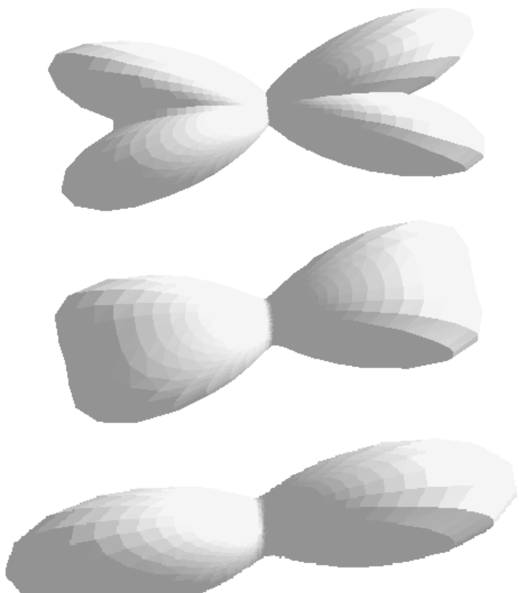


Fig. 19. Sum of ellipsoids, considering their NDF. *Top*: superimposition of the NDF of two $1 \times 2 \times 2$ ellipsoids separated by a $3\pi/16$ angle. *Middle*: sum of these two NDF. *Bottom*: our approximation of this sum, that is the NDF of the ellipsoid associated to the matrix $(Q_1^{-1} + Q_2^{-1})^{-1}$, with Q_i the 3×3 matrix associated to the ellipsoid i (if the ellipsoid is parallel to the canonical frame, the diagonal of this matrix contains the inverse of the squared radius).

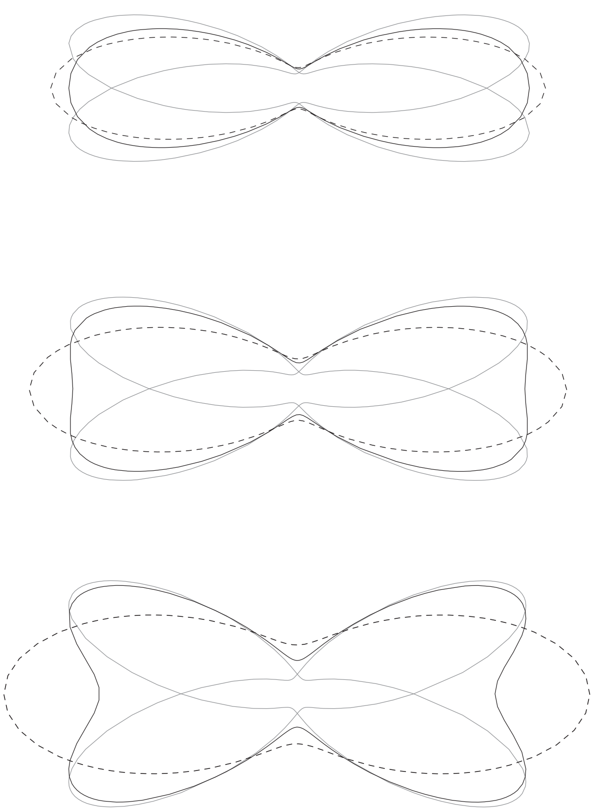


Fig. 20. xy slice of the NDF at $z = 0$ (apart the scale, this slice is the same for any value of the radius along z). The ellipsoids NDF are in grey, their sum is in black, our approximation is dashed. The angle between the ellipsoids is $\pi/8$ of top, $3\pi/16$ on middle, $\pi/4$ on bottom. The approximation is worst for a $\pi/2$ angle, as it would be a circle while the real sum is a quadripolar curve.

References

[Bli82] J. F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In *Computer Graphics (SIGGRAPH '82 Proceedings)*, volume 16(3), pages 21–29, July 1982.

[BM93] Barry G. Becker and Nelson L. Max. Smooth transitions between bump rendering algorithms. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 183–190, August 1993.

[Buc91] John Buchanan. The filtering of 3d textures. In *Proceedings of Graphics Interface '91*, pages 53–60, June 1991.

[Cro84] Franklin C. Crow. Summed-area tables for texture mapping. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 207–212, July 1984.

[FF88] Alain Fournier and Eugene Fiume. Constant-time filtering with space-variant kernels. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 229–238, August 1988.

[Fou92] Alain Fournier. Normal distribution functions and multiple surfaces. In *Graphics Interface '92 Workshop on Local Illumination*, pages 45–52, May 1992.

[FvDFH90] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practices (2nd Edition)*. Addison Wesley, 1990.

[Kaj85] James T. Kajiya. Anisotropic reflection models. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19(3), pages 15–21, July 1985.

[KK89] James T. Kajiya and Timothy L. Kay. Rendering fur with three dimensional textures. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23(3), pages 271–280, July 1989.

[Ney95a] Fabrice Neyret. Animated texels. In *Eurographics Workshop on Animation and Simulation'95*, pages 97–103, September 1995.

[Ney95b] Fabrice Neyret. A general and multiscale method for volumetric textures. In *Graphics Interface'95 Proceedings*, pages 83–91, May 1995.

[Ney95c] Fabrice Neyret. Local illumination in deformed space. Technical report, RR-2856, INRIA, 1995.

[Ney96a] Fabrice Neyret. Synthesizing verdant landscapes using volumetric textures. In *Eurographics Workshop on Rendering'96*, pages 215–224, June 1996.

[Ney96b] Fabrice Neyret. *Textures Volumiques pour la Synthèse d'images*. PhD thesis, Université Paris-XI - INRIA, 1996. http://www-rocq.inria.fr/syntim/textes/thesefabrice-eng.html.

[Nom95] Tsukasa Noma. Bridging between surface rendering and volume rendering for multi-resolution display. In *6th Eurographics Workshop on Rendering*, pages 31–40, June 1995.

[PF90] Pierre Poulin and Alain Fournier. A model for anisotropic reflection. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24(4), pages 273–282, August 1990.

[PH89] Ken Perlin and Eric M. Hoffert. Hypertexture. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23(3), pages 253–262, July 1989.

[PLH88] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, and James Hanan. Developmental models of herbaceous plants for computer imagery purposes. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 141–150, August 1988.

[Pro95] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface'95 Proceedings*, pages 147–154, May 1995.

[RB85] William T. Reeves and Ricki Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19(3), pages 313–322, July 1985.

[Ree83] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108, April 1983.

[SDB97] François Sillion, George Drettakis, and Benoit Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. In D. Fellner and L. Szirmay-Kalos, editors, *Computer Graphics Forum (Proc. of Eurographics '97)*, volume 16(3), pages 207–218, Budapest, Hungary, September 1997.

[SF92] Mikio Shinya and Alain Fournier. Stochastic motion - motion under the influence of wind. In A.C. Kilgour and L. Kjelldahl, editors, *Computer Graphics Forum (Eurographics '92)*, volume 11(3), pages 119–128, September 1992.

[Shi92] Mikio Shinya. Hierarchical 3D texture. In *Graphics Interface '92 Workshop on Local Illumination*, pages 61–67, May 1992.

[SLS+96] Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, and John Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 75–82. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[TF88] Demetri Terzopoulos and Kurt Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 269–278, August 1988.

[WH91] Jakub Wejchert and David Haumann. Animation aerodynamics. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 19–22, July 1991.

[Wil83] Lance Williams. Pyramidal parametrics. In *Computer Graphics (SIGGRAPH '83 Proceedings)*, volume 17(3), pages 1–11, July 1983.

Fabrice Neyret received his Master degree in Maths from Nancy University (France) in 1989, his ingeneering degree from Telecom Paris (France) in 1991 with a major in image processing, and his PhD in Computer Science from Orsay University & INRIA institute (France) in 1996. He did his Postdoctoral at Toronto University in 1997. He is currently a CNRS researcher in iMAGIS team (Grenoble, France). His research interests include textures, local illumination, natural phenomena (like fluids and clouds), complex scenes, etc.