A General and Multiscale Model for Volumetric Textures

Fabrice Neyret
INRIA
B.P. 105
78153 Le Chesnay Cedex
France

Fabrice.Neyret@inria.fr
http://www-rocq.inria.fr/syntim/research/neyret

Abstract

Volumetric texturing is a method dedicated to modeling complex repetitive geometries, such as grass, fur or foliage, by storing a density and a local reflectance in voxels. It was introduced by Kajiya in 1989 [KK89], who applied it to fur rendering.

In this paper, we propose to extend the method to a more general and effective tool for complex repetitive geometry modeling and rendering. Local reflectance is modeled by a sufficiently generic and compact representation. Then, for efficiency and image quality, a multiscale representation of the volumetric texture is used, which requires one to 'filter' the geometry represented by the texture.

A wide class of complex objects can be represented, and rendered efficiently with a cost linked to apparent complexity instead of data complexity. This is done with low aliasing, in a usual ray-tracing environment.

Keywords: volumetric textures, complex geometry, rendering.

1 Introduction

Complex repetitive geometries such as grass, hairs, foliage, fur, forest and so on are a major component of the natural world, very important for the realism of synthetic images, but hard to deal with in terms of explicit geometry. Many methods exist to model them, introducing more or less 3D effects depending on user needs and means, such as color or transparency mapping, mip-mapping [Wil83], bump mapping [Bli78, Per85, Lew89], displacement mapping [Coo84, MKM89], inverse displacement mapping [PHL91, Tai92], filterable reflectance mapping [Fou92], particle systems [Ree83, RB85], hypertextures [PH89], volumetric textures [KK89, Shi92].

To replace geometry by a representation more adapted to the complexity, a method has to deal with two main problems:

- generating most of the 3D effects: parallax, local illumination and blocking effects, [1]
- producing representative results at various scales while avoiding the aliasing due to the great amount of information visible in a pixel area.

Two more general problems are:

- to design a sufficiently general model: the user should not have to write additional code for each new kind of data.
- to build the representation from the classical geometric data.

This paper extends Kajiya's volumetric texture method in order to satisfy the four points listed above, in the following way:

• Volumetric texture is a textural approach to simulate geometry. This means that a sample of 'material' is built, and mapped upon a surface. The sample, called a *texel*, is a volume, which guarantees parallax and blocking 3D effects. A reflectance is stored in the volume's voxels, producing the local illumination 3D effect. Thus, volumetric texture is able to fully simulate geometry, generating all the 3D effects. This solves the first problem explained above.

• We describe in this paper how to get a multi-scale representation of a texel, inspired by mip-mapping techniques, with an octree encoding of the volume. This both avoids aliasing and saves a large amount

[1] That is to say apparent motion depending of the distance of the different parts of the objects, light reflection depending on the local surfaces orientation, occlusion and shadowing depending of relative positions compared to light sources and observer.

of computation time, hence addressing to the second problem. [2]

• By using a sufficiently generic reflectance model, we can adapt the parameters to represent almost any sample of complex geometry, which is the third problem.

• The fourth problem is a vast subject, which is independent from texel representation and rendering. We focus here only on these two latter aspects.

We present the volumetric texture method in subsection 2.1. Some reflectance models are considered in subsection 2.2.

We describe our extension in section 3 : first, we present the multiscale data structure and algorithm. Then, we justify the choice of ellipsoid to model reflectance in subsection 3.1, we explain how to compute the local reflectance using this primitive in subsection 3.2, and finally we describe in subsection 3.3 how to 'filter' it in order to achieve the multiscale representation.

2 Previous Work

2.1 Volumetric Textures

As mentioned in section 1, Kajiya's volumetric texture method is a textural approach : the complex geometry that has to be modeled covers a wide surface, as a thick skin (e.g. a lawn covering a hill, or fur on an animal). A sample of this geometry is built in a cubic *reference volume*, which is mapped over the surface similarly to a 2D texture (allowing deformations). The copies of the reference volume, named *texels*, are also deformed in height in order to stick to each other.

The reference volume represents an area of space sampled into *voxels*, at a given resolution. Three kinds of information are stored in each voxel : a density (which is in fact an isotropic probability of occlusion), and a 'memory' of the reflectance behavior of the geometry represented in this subarea, which can be separated into a generic reflectance model and a local orientation (a basis).

In the implementation described in [KK89] for the purpose of rendering fur, the reflectance model is the same for the whole texel (it is an ad-hoc cylinder reflection model), and the local orientation (cylinder axis) is equal to the height vector (hairs are straight in the reference volume, and the texels are 'combed'). Therefore only the density has to be stored. Texels fit exactly with the bilinear patches of the underlying surface, and the vertical edges follow the surface's normals (which can be jittered or 'combed'). Thus, texels are a trilinear deformation of the reference volume.

The rendering is done by ray-tracing, which becomes a volumetric ray-tracer when a texel is crossed. The light and opacity accumulations are achieved by stratified sampling. The local illumination causes the geometric illusion : the light will be reflected in a voxel as if a cylinder layed inside.

Here are the principles of the rendering task :

• A ray-tracer scans the scene with a succession of rays, which hit the surfaces.

• When a surface is covered by a texel, the ray has to switch in 'volumetric mode'. This is done in the reference volume, after having computed the corresponding in and out points. The path in the reference volume is approximated by a straight trajectory (the curvature of the rays in the texture space is neglected), thus the volume can be crossed as usual.

• The volumetric ray-tracer crosses the volume from front to back, multiplying transparencies and adding intensities (weighted by cumulated transparency). An area with density ρ crossed on a length L has a transparency $e^{-\tau\rho.L}$, where the optical depth τ converts density into attenuation (see [KK89]). The final intensity collected through a wide inhomogenous area is so $I = \sum_{near}^{far}(I_{loc}.\prod_{near}^{cur.} e^{-\tau\rho.dL})$.

• The local illumination I_{loc} is the product of the received light, the reflectance and the density. The reflectance indicates the amount of energy scattered from the light to the viewing direction, and is obtained from the reflectance model.

• The received light is estimated by casting a shadow-ray towards the source, which only takes into account attenuation from the light source to the point (low-albedo hypothesis neglects multiple light reflections) [3].

It may be suggested that very efficient algorithms like [LL94] already exist for volumetric rendering, and are able to render tomographic data in a few seconds. In fact, they cannot be used here, because the volumetric rendering is done in a particular context : the volume is small but repeated and deformed, computations have to be done at each voxel to obtain

[2]Multiscaling requires the ability to 'filter' the data. One has to consider that 'filtering' geometric data is an outstanding point, connected to the level of detail problem : color elements of a 2D texture can be smoothed, because the apparent color is a linear function of color parameters, which is not the case for parameters like normals, positions, etc.

[3]See [AW87] for volumetric ray-tracing, and [RT87] for all 'interbleeding' volumetric effects in general case.

local illumination, and texels are just an element among a larger ray-traced scene. Anyway, the cost is not concentrated on the volume parsing itself.

Shinya introduces in [Shi92] some ideas and extensions (correlation between voxels content, cone-tracing to go through the volume), and especially the need of filtering the data to pre-compute information at any scale.

Filtering is a matter of efficiency in two ways: it saves time by avoiding the oversampling needed to prevent aliasing, and it increases the quality by generating adapted pre-computed data at the needed scale according to the apparent size.

2.2 Reflectance Modeling

Isolating and filtering the photometric aspect of a shape is linked to the problem of reflectance encoding, and therefore to anisotropy modeling: at low scale, the photometric behavior is much more important than shape itself, so that a local surface can be simulated by a reflectance function.

On the other hand, an arbitrary reflectance behavior can be modeled by a 'micro-geometry', which can be figured as a kind of 'crystallization' (the shape is too small to be seen except by its photometric behavior), where shapes could be spheres [Bli82], cylinders [PF90], or less restrictive primitives.

There are many other ways to model reflectance, of varying generality. The more general representation is a full *BDRF* (Bi-Directional Reflectance Function), indicating how much light coming from a given direction reflects towards another given direction [CMS87], that can be tabulated, or encoded with harmonic functions.

When reflectance is caused by microsurfaces, it can be represented by the local normal repartition function. Fournier approximates it by a set of Phong-peaks. In [Fou92], he explains how to 'filter the geometry' encoded by bump maps, and finally how to encode, map and filter reflectance on a surface.

To choose a model, a compromise has to be found between generality requirement and memory constraint.

3 Multiscale Extension of Volumetric Textures

In this section, we develop our extension to volumetric textures: we first present the multiscale texel data structure, and the changes to the rendering process. Then we present the local reflectance model, and we discuss how to render it, and how to filter it in order to build the multiscale representation.

Regarding 2D textures, multiscaling is achieved by methods such as mip-mapping [Wil83], which stores successively smoothed and reduced images of the texture, and chooses, while rendering, the one adapted to the apparent size (so that a pixel on screen corresponds roughly to an image pixel). Thus, 2D textures can be rendered with low aliasing and at low cost.

To adapt this idea to 3D data, we use an octree to store the reference volume. Octrees are already used by volumetric algorithms as a compact representation, constant areas being representable by few nodes (see [Sam90b, Sam90a]). We exploit them here for their multiscale ability (all nodes from root to locally useful size are kept), in addition to their compactness.

Any voxel in the octree simulates the photometric local behavior (i.e. the 3D effects) of the object represented in the texel: the voxel position gives the parallax, but the occlusion and the reflectance in each direction have to be encoded. We store the local density and a micro-primitive (described in 3.1) modeling the local reflectance. This primitive also modulates the opacity according to the direction.

'Painting' in 3D the texel with the complex object sample fills the leaves of the octree. Higher levels are built by successive filterings (we will see in 3.3 how to sum the micro-primitives).

Like in [KK89], our texels are positionned on bilinear faces, with vertical edges following the normals at the four vertices. They are thus a trilinear deformation of the reference volume. We associate Phong color coefficients to each texel.

The texel rendering described in 2.1 is modified according to the multiscale data structure and to the reflectance encoding:

• The ray-tracer is extended to become a minimal cone-tracer, able to indicate roughly the size of the pixel inverse projection on the intersected surface. This size is used to choose the right level in the octree (the chosen voxel is a cubic area approximating a part of the ray conic area, so the filtering is not exact: a few aliasing remains or a blur appears, like for mip-mapping).
• The octree is parsed recursively along the ray: the segment of ray inside a node is split until the required or the minimal level is reached.
• The local illumination is computed using the local

reflectance and density. The reflectivity is obtained from the micro-primitive and the directions of light and view (see 3.2); the local occlusion is obtained from the density, modulated by the micro-primitive.

In our approach, cone tracing, texel mapping, trilinear deformation, octree crossing and volume rendering are implemented in a usual way. We focus here only on the reflectance model, which allows one to implement the whole multiscale approach.

Our requirements are therefore:

- Specify a good primitive in terms of generality, computability, and filtering ability. This is the subject of section 3.1.
- Render the primitive, i.e. evaluating the total amount of light reflected towards the eye by the primitive. This is detailed in section 3.2.
- 'Filter' the primitive: the purpose is to build coarser representations of the geometry, obtained by summing the reflectance functions as explained in section 3.3.

3.1 Choosing a Micro-Primitive

Keeping a full BDRF is not useful here: anisotropy can be represented by the normal repartition function since it is due to the microgeometry inside a voxel area. Such a function has to be itself described by only a few parameters, because of its storage in each voxel. Fournier [Fou92] Phong-peak decomposition is a good one, but is still too expensive for volumetric data.

On the other hand, geometric primitives like sphere or cylindrical models are too specific. Moreover, the light reflectance of a set of two cylinders has to be represented at coarser resolution by a single primitive, which cannot be a cylinder. (We have to keep in mind that although geometric, these primitives are used to define a normal repartition, more than a shape by itself, thus representing a class of shapes).

As a compromise, we have chosen our geometric primitives to be ellipsoids. This is less general than a least-square approximation of normal repartition, but it has enough degrees of freedom and is quite compact: with six parameters it can approximate at least a sphere, a cylinder (a long thin ellipsoid), a plane element (a very flat ellipsoid), and all other intermediary shapes.

There are several ways of choosing these parameters: two useful representations are a basis with three lengths, or the coefficients of a quadratic form. We chose to store the former, which is easier to use at construction time, and can quickly be converted to the latter representation.

In fact, we need less than six parameters: as said before, the 'micro-primitive' is a kind of 'crystallization', a shape without size and position, which only purpose is to reflect the light. So a normalisation process is necessary to treat different shapes with the same weight during the rendering and filtering stages. We choose the mean apparent surface (the context is visibility), the mean being approximated by evaluating the apparent surface in the three axis directions.

We show in section 3.3 that the photometric behavior of a set of ellipsoids can reasonably be approximated by an ellipsoid.

As a remark, we have to keep in mind that we manipulate two different levels of shapes when drawing in the reference volume: the geometrical global shapes, which occupy many voxels (they can be themselve cylinders or ellipsoids), and the local micro-primitives stored in each voxel, which model microgeometry. For usual objects, the micro-primitive represents the geometrical contribution of the shape clipped in a voxel.

Anisotropic objects can be modeled as well, by using a micro-primitive (the 'crystallization') more or less independent of the global shape, as shown in figure 1. In the same way, roughness can be represented by increasing the variations of the local geometric contribution.

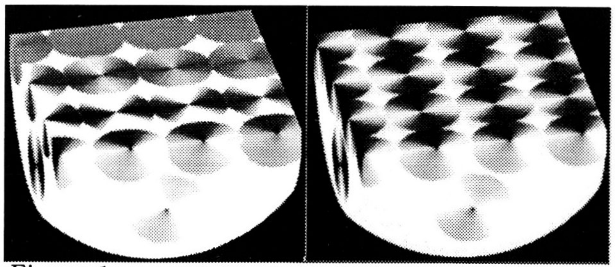
Figure 1: *scratched aluminium* (single texel). The local primitives are concentric cylinders progressively smoothed to spheres (left), or radial cylinders (right). 'Cylinders' are modeled by thin long ellipsoids.

3.2 Rendering the Primitives

We have to compute the reflectance of a micro-primitive, that is to say the ratio of energy received from a light source and scattered towards the eye by the whole ellipsoid, according to the Phong local reflection model.

The environmental interactions are solved by the volumetric ray-tracing, which evaluates the amount of incoming light, and accumulates illumination and opacity along each ray. The local illumination emitted is light×density×reflectance. The density is also modulated by the micro-primitive, according to its apparent surface in the ray direction.

Evaluating the global reflection consists in summing the BDRF, i.e. integrating the Phong reflection model over the micro-primitive apparent surface.

Integrating over conics seldom has an exact formulation. So we use a numerical scheme, sampling uniformly the apparent surface.

The rectangle which bounds the apparent ellipse is obtained from the quadratic form of the ellipsoid. Only half of the ellipse surface needs to be sampled, since symmetrical points can be constructed at the same time.

To evaluate the incoming light, a shadow-ray has to be casted towards light sources to test occultations and collect attenuation. We consider the low-albedo hypothesis, so we just have to take care of the opacity on this path, the secondary reflections being omitted.

One has to note that complicated opaque shapes are welcome: the more occluding the shapes, the quicker the computation. For instance only the surface of a solid object is visible, so even a fractal solid object looks like a surface (possibly discontinuous) to the eye. On the other hand, rays fully go through in an almost empty volume, and numerous shadow-rays will be launched from a spread material.

3.3 Filtering the Primitives

3.3.1 Intuitive Filtering

Considerations on ellipsoids have to be done in the normal distribution dual space rather than in the geometric space, because local primitives are visible by their reflectance behavior rather than by their geometry. One has to remember that behaviors that seem approximately correct geometrywise often lead to incorrect behaviors in the normals space.

The sum of the normal repartition of two ellipsoids *looks* not too different from an ellipsoid normal repartition, the approximation being the worst for orthogonal ellipsoids, but not too bad for primitives similar to each over. This is easy to illustrate in 2D (see figure 2).

For repeatedly filled volumes, the procedure is straight forward, e.g. grass can be modeled by cylinders, with fluctuating orientations. Successive filterings of a lawn produces at the coarsest level (when the whole lawn is included in a single voxel) an ellipsoid oriented to the mean direction, and elongated according to the weakness of the variation. A wide cylinder is locally almost flat, so local primitives are planar pieces. Successive filterings produce a single quasi-cylinder local primitive at the coarsest level, as shown in figure 3.

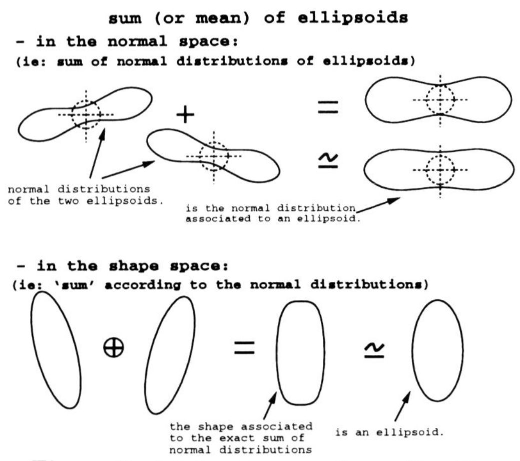
sum (or mean) of ellipsoids
- in the normal space:
(ie: sum of normal distributions of ellipsoids)
+
=
≃
normal distributions of the two ellipsoids.
is the normal distribution associated to an ellipsoid.
- in the shape space:
(ie: 'sum' according to the normal distributions)
⊕
=
≃
the shape associated to the exact sum of normal distributions
is an ellipsoid.

Figure 2: defining the *sum* of two ellipsoids.

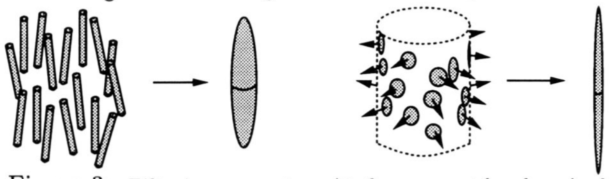

Figure 3: *Filtering geometry*: At the coarsest level, a single primitive result of the successive filterings, and has to represent the global reflectance of the whole texel. This can be verified with two examples: a lawn composed of repeated elements, and a wide cylinder composed of continous surface elements.

For non-repeatedly filled volumes, there is really an approximation: in 2D, filtering two orthogonal similar primitives by this method produces an isotropic reflectance.

Blocking effects (i.e. occultation and shadowing), which are lost with simple mapping techniques, are correctly dealt with by texels, in that a real 3D information is kept. But things degenerate under the voxel size, where photometry replaces geometry: while filtering, the sum is purely geometric, without the notion of hidden objects, which become partly visible through the resulting reflectance function. Again, things go well with repeatedly filled volumes, as auto-similarity persists while filtering (if there is no correlation between positions). Otherwise, there is a significant approximation, as no blocking effect occurs below the voxel size. This is connected to the fact that a normal repartition function loses the position information, and thus forgets that a single normal may correspond to two different areas of a non-convex object, hiding each other.

Another point concerns shadows: during successive filterings, the density difference decreases between voxels 'inside' or 'outside' objects. Thus density tend to become uniformly low, producing the impression of diffuse shadows in haze more than one of shadows over solid objects. Although we have to be aware of these effects, in practice things behave relatively well.

3.3.2 Defining Filtering

As suggested before, *adding* ellipsoids consists in choosing the shape whose normal repartition function is the closest to the sum of the normal repartition functions of the ellipsoids to be merged. In order to avoid a long least-square optimisation, we choose a direct way of computing this 'sum', which is reasonable as far as the primitive is simple. Nevertheless, there is no analytical solution when integrating upon ellipsoids, so an approximation has to be found.

With the quadratic form $M^t.Q.M = 1$ associated to the ellipsoids, it is possible to write the expression of the normal repartition function, which can be interpreted as the probability density f of having a normal N in a given direction. It is equal to the Jacobian of the bijection from the ellipsoid surface to the dual normal space on the Gaussian sphere: $f_Q(N) = det(Q^{-1})/(N^t.Q^{-1}.N)^2$
The problem is then to find Q with f_Q closest to $f_{Q_1} + f_{Q_2}$. A study of f_Q shows that things are 'almost-additive' with Q^{-1}: with $g_{Q^{-1}} = f_Q$, we have $g_{\lambda.Q^{-1}} = \lambda.g_{Q^{-1}}$, so the sum of identic shapes is conservative. The Taylor series development of $g_{Q_1^{-1}} + g_{Q_2^{-1}}$ for two similar ellipsoids rotated of θ relative to one another is $g_{Q_1^{-1}+Q_2^{-1}} + O(sin^2(\theta))$. This confirms that we can obtain a good approximation for low angles between ellipsoids.

Thus, we define as the 'mean ellipsoid' the one which inverse quadratic form is the mean of inverse quadratic forms of the ellipsoids to merge. Of course, summed ellipsoids have to be weighted by their associated density: $Q^{-1} = \sum \rho_i.Q_i^{-1} / \sum \rho_i$. Quadratic forms are easily obtained from the geometric basis and lengths, corresponding to matrices R (orthogonal) and L (diagonal): $Q_i = R_i.L_i^{-2}.R_i^t$. The basis and lengths of the resulting ellipsoid are recovered from the eigenvalues ($r_i = \lambda_i^{-2}$) and eigenvectors of Q.

Figure 4 illustrates the successive filterings steps of the voxels: the series of pictures looks like the result of successive smoothing operations on the original image, while they are the rendering of successive *geometry filterings*, these filterings being precomputed at modeling time.

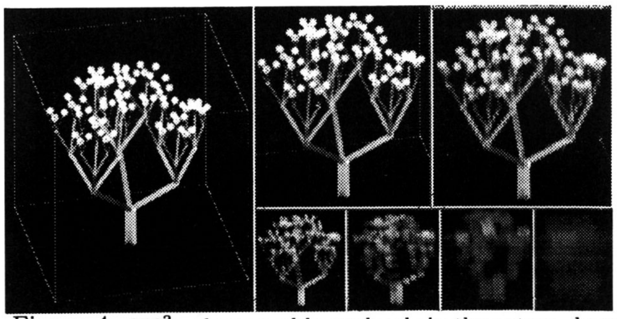
Figure 4: 256^3 volume and lower levels in the octree, down to 4^3. On an Indigo^2, the building costs 8 sec, and the rendering 20 sec for the first picture, 12 and 10 for the two next at resolution 444x444.

4 Results

To illustrate the 'geometric illusion' obtained with texels, we first show in figure 5 an example of complex data in a wide single texel (a typical 3D sample does not need to be so complicated!).

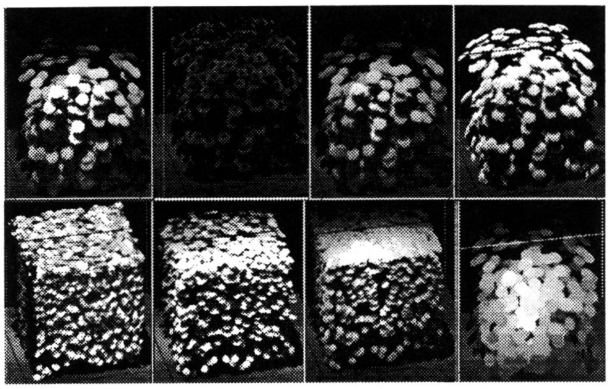
Figure 5: garden-bushes, with various local primitives. (The texel is bounded by a frame, whose shadow is visible on the floor, itself slightly anisotropic.)

The strange bushes-and-spheres image of figure 6 illustrates the progressive filtering: bush texels at 256^3 resolution contain 2000 leaves; 50x500 texels are mapped on a plane. This is equivalent to a geometric database of 50 million flattened spheres, or at least 400 million triangular facets. Despite only one ray per pixel, no aliasing appears. Computation takes 14 minutes on an *Indigo*^{2} at resolution 444x444.

With the velvet image of figure 7, we can see anisotropic effects obtained by small repetitive geometry: each hair locally obeys the Phong model, but depending on orientations we see accumulations of illuminated top of hairs (on the top of the hump), or shadow at the base of hairs (on the right of the hump), in addition to the self-occlusion effect (equivalent to the phases of the moon) for each hair. The

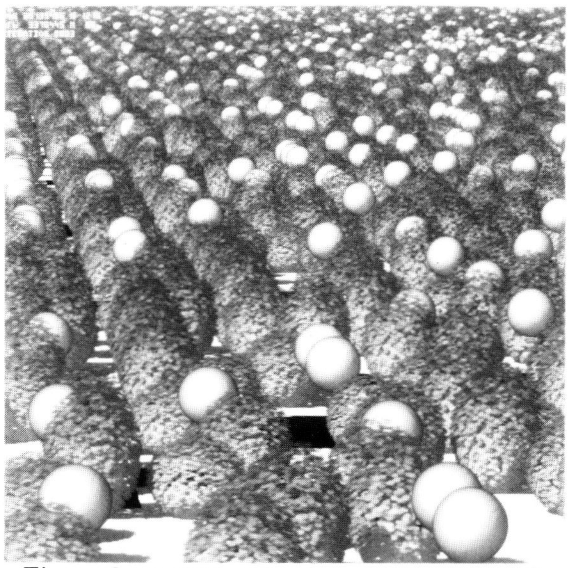

Figure 6: 50x500 bushes and spheres (14 minutes).

shadow on the left is a classical one, the light being on the right.

The last three image pairs show various kinds of texels at resolution 128^3, and their mapping on a geometry composed of 100 to 1000 bilinear patches (when the texels are too near, voxels are sometimes individually visible). At video size, the computation takes between 5 and 20 minutes, a large part being taken by the computation of the intersection of the rays with the geometric patches.
In the forest image (figure 8), we can see that the representation can be coarse: the level of details painted in the texel just has to fit the minimal distance the user wants to assume. On the building image (figure 10), the reference volume shows that reflectance information can produce an illusion of high resolution (however, details finer than a voxel become transparent). The furry torus of figure 9 illustrates a cyclic texel (like the lawn on figure 7). On the mapped image, some remaining aliasing is visible: the texels are deformed a lot in order to 'comb' the hairs in a given direction.

Comparatively, the famous Teddy Bear image illustrating Kajiya's paper needed the equivalent of a dozen CPU hours on an IBM 3090, at resolution 1280x1024.

Another method used for simulating repetitive geometry are particle systems ([Ree83, RB85]). The realism induced by the geometric complexity is also impressive, but rendering an image takes many hours, shading and shapes are specified by coding, and the rendering is hard to mix into a classical ray-traced scene.

Color images and mpeg animations can be seen at our WWW address.

5 Conclusion

In this paper, we focused on modeling complex repetitive geometry efficiently, by extending Kajiya's approach of volumetric textures. We introduce a compact model which is able to generalize the local reflectance, and encode texels with octrees. Then we describe how to adapt the mip-mapping multiscale approach to volumes containing reflectance information, which requires one to filter it.
This greatly extends the scope of this technique:

- With adaptative rendering, and thus low cost, texels are able to produce information at many scales: from the level where the whole texel is seen in a single pixel to the level where a single voxel appears as a visible cube. Thus, texels can be used everywhere as a saving to repetitive geometry. Beyond acceleration, this makes affordable the computation of animation on very complicated scenes.
- Using a quite generic reflectance primitive, a wide range of data can be modeled.
- Avoiding most of the aliasing, texels can be seen as a way to correctly render small geometries. Moreover, this is done at low cost.
- The concept of mapping 3D geometry on 3D geometry is itself a useful way of designing complex scenes.
- Anisotropy is available, as a simple side effect of the method.

There are nevertheless some limits and drawbacks:

- As a textural approach, volumetric texturing applies to repetitive patterns, and to surfaces on which the mapping can be performed with reasonable deformations.
- It may seem natural to imbricate two 'geometric textures', or to put a real object and a texel on the same place, but this turns out to be hard to implement.
- Mapping supposes no motion inside the pattern itself, otherwise the reference pattern has to be recomputed each time.
- There is no perfectly precomputed data, as for classical 2D mip-map: a choice has to be made between blur and aliasing (or oversampling) in bad cases. However things are far better than with real repetitive small geometry!
- Filtering does not handle blocking effects perfectly, which may limit the scale of applicability in some cases.

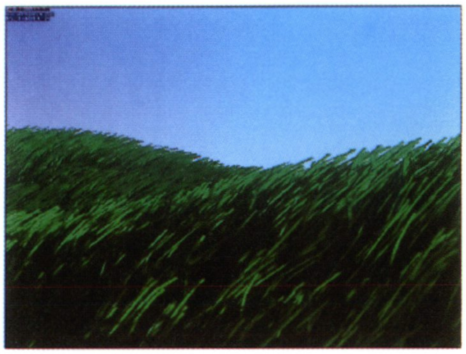
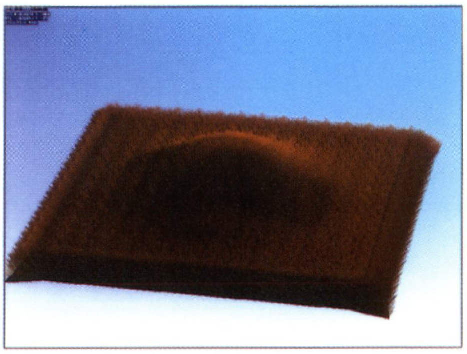

Figure 7: *left*: a lawn on 1404 bilinear patches. The texel contains 16 blades of grass with a 'V' section; its resolution is 128x128x128 (compression 91%). *right*: velvet. Cylinders all orthogonal to the surface cause a global anisotropy.

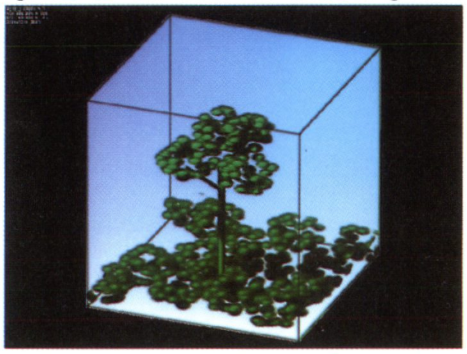
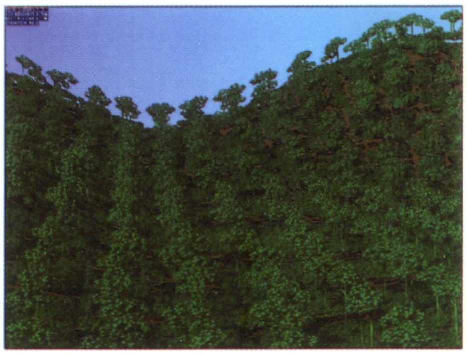

Figure 8: *left*: a single texel at 128x128x128 (compression 92%), designed to be seen from far away. *right*: mapping on a hill with 578 bilinear patches (23 minutes for rendering, including 12 for patches intersection).

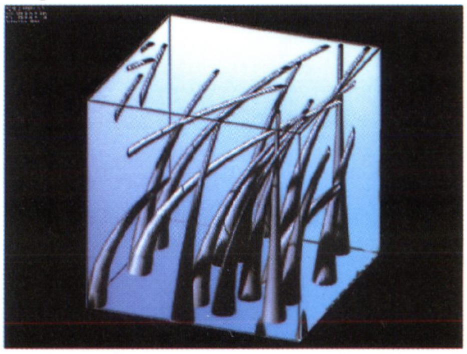
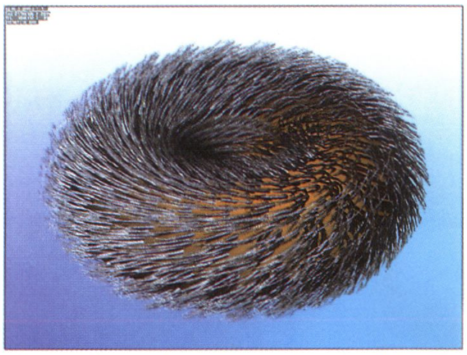

Figure 9: *left*: hairs cyclically drawn in a 128x128x128 volume (compressed at 93%). The single texel rendering needs 3.5 minutes. *right*: mapping on a torus with 240 bilinear patches (12 minutes).

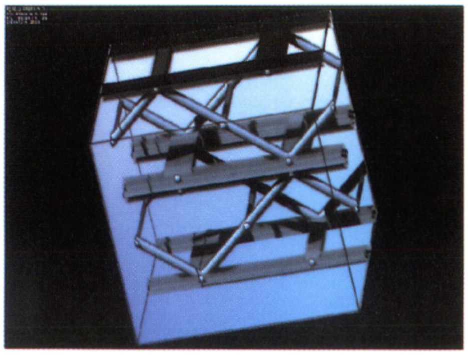
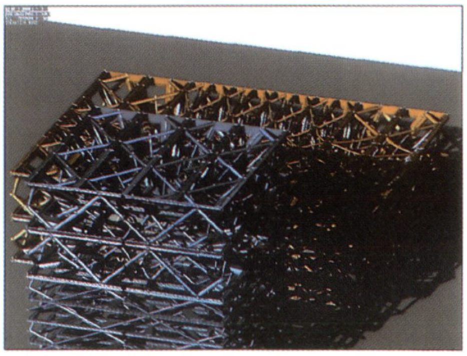

Figure 10: *left*: texel with building elements at 128x128x128 resolution compressed at 92% (elements are sometimes smaller than a voxel, becoming transparent). *right*: mapping on cubic shapes (81 bilinear patches, 14 minutes).

Future work will address improving the filtering techniques, and accelerating the rendering by estimating reflection without numerical integration. Fine estimation of the correct voxel size to use at rendering time has to be done, in order to suppress the remaining aliasing without blurring the data. Moreover, the mapping refers to bilinear faces, which is quite limiting. The usual parametrizations used with other texture methods have to be adapted.

And, of course, to obtain a useful productive tool, the implementation has to be completed with several volume initialisation methods, such as tomographic 3D images, particles systems, script primitives description, geometry sampling, procedural noise functions [Lew89] or hypertextures [PH89].

6 Acknowledgements

I would like to thank Sabine Coquillart, Anne Verroust, Jean-Marc Vezien, Philippe Decaudin and Xavier Provot for their helpful discussions, and their patient corrections of this paper.

References

[AW87] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In G. Marechal, editor, *Eurographics '87*, pages 3–10. North-Holland, August 1987.

[Bli78] James F. Blinn. Simulation of wrinkled surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, volume 12(3), pages 286–292, August 1978.

[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.

[CMS87] Brian Cabral, Nelson Max, and Rebecca Springmeyer. Bidirectional reflection functions from surface bump maps. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21(4), pages 273–281, July 1987.

[Coo84] Robert L. Cook. Shade trees. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 223–231, July 1984.

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

[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.

[Lew89] John-Peter Lewis. Algorithms for solid noise synthesis. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23(3), pages 263–270, July 1989.

[LL94] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear–warp factorization of the viewing transformation. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 451–458. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[Mil88] Gavin S. P. Miller. From wire-frames to furry animals. In *Proceedings of Graphics Interface '88*, pages 138–145, June 1988.

[MKM89] F. Kenton Musgrave, Craig E. Kolb, and Robert S. Mace. The synthesis and rendering of eroded fractal terrains. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23(3), pages 41–50, July 1989.

[Per85] Ken Perlin. An image synthesizer. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19(3), pages 287–296, July 1985.

[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.

[PHL91] J. W. Patterson, S. G. Hoggar, and J. R. Logie. Inverse displacement mapping. *Computer Graphics Forum*, 10(2):129–139, June 1991.

[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.

[RT87] Holly E. Rushmeier and Kenneth E. Torrance. The zonal method for calculating light intensities in the presence of a participating medium. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21(4), pages 293–302, July 1987.

[Sam90a] Hanan Samet. *Applications of Spatial Data Structures*. Addison-Wesley, Reading, Massachusetts, 1990.

[Sam90b] Hanan Samet. *Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, Massachusetts, 1990.

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

[Tai92] Frédéric Taillefer. Fast inverse displacement mapping and shading in shadow. In *Graphics Interface '92 Workshop on Local Illumination*, pages 53–60, May 1992.

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