Synthesizing Verdant Landscapes using Volumetric Textures

Fabrice Neyret

INRIA Rocquencourt
BP 105, 78153 Le Chesnay Cedex, FRANCE
Fabrice.Neyret@inria.fr   http://www-rocq.inria.fr/syntim/research/neyret

Abstract: Volumetric textures are able to represent complex repetitive data such as foliage, fur and forests by storing one sample of geometry in a volumetric *texel* to be mapped onto a surface. This volume consists in samples of densities and reflectances stored in voxels. The texel can be prefiltered similarly to the mip-mapping algorithm, giving an efficient rendering in ray-tracing with low aliasing, using a single ray per pixel.
Our general purpose is to extend the volumetric texture method in order to provide a convenient and efficient tool for modeling, animating and rendering highly complex scenes in ray-tracing. In this paper, we show how to convert usual 3D models into texels, and how to render texels mapped onto any mesh type. We illustrate our method with verdant landscapes such as forests and lawns.

1 Introduction

Geometric complexity is an important aspect of realism in synthetic scenes. This is particularly true for natural scenes such as forests, lawns, fur, etc. In order to represent and render scenes, one has to choose between using a general synthesis process (consisting of interactively modeling meshes, specifying the animation, and launching a renderer), and using a data-specific tool for either the modeling stage or eventually the whole synthesis process. In this paper, we show that for complex scenes, volumetric textures are a good trade off between generality and efficiency.

Classical synthesis process
It is convenient to use general tools and representations, since many products are available and existing data can be re-used. The high geometric complexity makes this theoretical solution usually impractical in each of its stages :

- modeling requires a lot of detailed knowledge from the user, and is very repetitive.
- the generated database is highly memory-consuming.
- similarly, the animation needs the user to specify the motion of many components.
- the rendering has to deal with billions of facets, which is costly in time and results in a lot of aliasing.

Although clever optimisations and database structurations exist [FvDFH90], antialiased ray-tracing of such scenes is still very slow. An important aspect lies in the determination of levels of details, which is currently an active research topic. Scanline-renderers with shadow capabilities can also be used, but one has to deal with aliasing, and detailed shadows can hardly be obtained with a wide scene containing fine details.

Specific approaches
These approaches solve the problem for particular kinds of scenes, either by procedurally describing the objects, or by providing a specific representation and the mean to render it. The first category concerns L-systems, fractals, botanical and physical models

[Pa88, dRa88, FR86] which generate polygons. The second category mostly concerns particle systems and some landscape-specialized models [RB85, WP95]. which use specific representations and rendering techniques. Procedural geometric models do not address the rendering aspect of the problem, and specific representations forbid the use of general modeling tools and existing databases. The integration of different kinds of objects in a single scene can also be limited.

Volumetric textures
In our previous work, we have shown how to make volumetric textures an efficient way of representing and rendering complex repetitive data : the precomputed multiscale representation provides the right look of the data at the right scale. The textural approach is limited to scenes which contain some repetitiveness. However it allows us to separate the scales of specification into a local 3D aspect (the texel pattern) and a wide aspect (the surface on which texels are mapped, and the deformation of these texels). Therefore, volumetric textures provide a relatively general representation of complex scenes, and enable an efficient unaliased ray-traced rendering (see section 2). Thus, volumetric textures are a good trade off between generality and efficiency.

But the texel representation described in previous work is mostly specific, as the texel content has to be specified 'manually' by the user, and the surface has to be discretized by bilinear patches in such a way that one texel corresponds to one patch.
In order to make the volumetric texture model usable, we describe in this paper :

- A way to convert usual object descriptions into volumetric textures (section 3), thus allowing the use of existing modeling tools and databases ;
- A method of rendering texels mapped on any mesh type (section 4), using texture coordinates such as those for usual 2D textures.

Having solved these two major issues, we illustrate the model by synthesizing verdant landscapes (section 5) using an existing representation of trees and a free mapping on surfaces.

2 Volumetric Textures

Kajiya and Kay first introduced texels in 1989 [KK89] mainly to render fur. Despite the fact that the paper contained the basic ideas, a lot of coding is still required to simulate other materials, and rendering is very slow since no multiscale scheme is proposed. Moreover, the surface has to be subdivided into bilinear patches so that one texel corresponds to one patch. However, the look of the resulting teddy-bear is simply marvelous. The authors use a full volumetric storage as a 3D texture pattern (the *reference volume* stored once) to encode the geometric data, and an underlying surface meshed with bilinear patches (see figure 1). Each texel is mapped exactly upon a bilinear patch and deformed in order to stick to the neighboring texels, thus forming a thick layer upon the surface. The vertical edges (common with adjacent texels) can be 'combed' by the user.

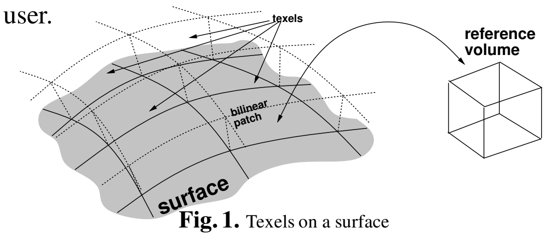

Fig. 1. Texels on a surface

The voxel content is not really a density but rather an non-directional probability of occlusion. Another crucial piece of data is stored in the voxels : a reflection model and a local frame, which makes the volume reflect light just like a real object would. The fur rendering implementation models the reflection from a cylinder, and the frame is limited to the cylinder axis. These parameters are not really stored in the voxels since they are constant in the volume (the variation in the combing is obtained at the mapping level).

Thus the volumetric texture is defined by a reference volume, an underlying surface made of bilinear patches, and a thickness vector at each vertex of the surface.

Shinya proposed some improvements in 1992 [Shi92], by storing the occlusion for each spatial direction in order to make the occlusion direction-dependent, and by traversing the volume by steps in each spatial direction. He listed unsolved problems (texel construction from polygonal description, voxels prefiltering), and recommended the use of a hierarchical approach.

In a previous paper, we proposed an extension to the texel representation [Ney95b], for generality and efficiency : various data can be represented without extra coding, and ray-tracing is achieved efficiently with low aliasing, using a single ray per pixel. Efficiency is obtained with a multiscale scheme similar to the mip-mapping [Wil83]. The key idea lies in the encoding of the local geometry included in a voxel volume, characterized by a Normal Distribution Function (NDF), in such a way that it can be filtered to provide a lower resolution. Thus the volume can be precomputed at each resolution at modeling time, as in mip-mapping. The generality is obtained by keeping enough degrees of freedom in the NDF encoding. We have chosen to approximate this NDF by finding the 'closest' ellipsoid, which needs few parameters.

This ellipsoid plays the role of the reflectance model and the local frame of Kajiya's model. It is determined at modeling time while discretizing a shape into the reference volume. Local illumination is computed at rendering time by numerically integrating the Phong model on the NDF. As for 2D mip-mapping, the two scales closest to the ray thickness are used and interpolated, thus providing filtered data with only one ray per pixel. Therefore, the features of this representation are a probability of occlusion, six ellipsoid parameters, and a pointer to the eight children (if any), stored in each voxel of an octree.

We have demonstrated in another paper [Ney95a] how to animate the representation in the same spirit, separating the scales of control. Two important unsolved problems were texel construction and texel mapping. Noma [Nom95] proposed a solution to the problem of constructing texels from a particular kind of geometry : sparse little facets (given by the AMAP tree modeling software).

In the next section, we present methods to convert most of the classical geometric representations (CSG, polygonal meshes, L-systems, particle systems, hypertextures,...) into volumetric textures. In section 4, we deal with texel mapping.

3 Texel Building

Many representations and modeling tools already exist to specify shapes, either general or specific, that users know well and like to use. A convenient way to specify the reference volume content is thus to convert existing representations into volumes. 3D scan-conversion still exists to convert some representations into volumes, but this method parses the whole volume systematically, and is quite costly. Moreover, not only do we need to know the voxel occupation, but also the local reflectance, which implies that the method has to be extended.

CSG, polygonal meshes, implicit functions, particle systems and L-systems [FvDFH90, RB85, Pa88] are primitive-based techniques: these constructions design a shape by combining simple shapes (the primitives being respectively a solid, a facet, a skeleton element, a trajectory segment, a terminal symbol). The simplicity of these primitives allows more efficient and accurate conversion than straight 3D scan-conversion: such a primitive can be defined by a distance function, or more conveniently by the distance to its surface, made negative if the tested point is inside the shape. The normals are given by the gradient of the distance function. The conversion of these models is described in section 3.1.

On the contrary, scanner data and hypertextures [PH89] correspond to full volumetric representations, for which data is given at each space location. Local density orientation and octree compression are then determined by scanning the volume. We detail these models in section 3.2.

We use the texel representation presented in our previous work [Ney95b], in which the data is stored in an octree. To sum up the process, the construction corresponds to the thinnest scale; then a propagation is achieved in the upper stages of the octree in order to precompute the filtered data at all resolutions; finally a compression pass is achieved to suppress unvarying information.

Note that by using a volumetric storage, building a shape looks like painting it into a 3D drawing. This allows many operations while designing a shape: instead of just setting the voxel data, one can add, substract, set only the NDF... This is used to implement CSG operators as we will see hereafter.

3.1 Primitive-based models

Multiscale data structures like octrees are well-adapted to recursive construction: recursive construction supposes us to be able to determine if a given space area is inside, outside or on the frontier of a shape, in the same spirit of the Warnock algorithm in 2D [FvDFH90]. The function which gives the distance from a tested point to the surface of the shape to be built, conveniently made negative inside the shape, provides a way to determine the status of a voxel regarding the shape (see figure 2):

- if the distance from the center of the current voxel (initially the root of the octree) is greater than the radius of the sphere bounding the voxel, the voxel is outside the shape.
- if this distance is negative, its absolute value being greater than the radius, the voxel is inside the shape.
- otherwise the voxel is probably on the frontier, so we split the voxel into eight children, and apply this algorithm to each child. If the minimum size is reached, we evaluate and store the geometric information corresponding to this area of space.

Note that getting the distance function is easy for simple primitives such as sphere or cylinder, and that normals can directly be obtained by computing the gradient.

We have implemented a large amount of primitives, that can be directly used in a declarative script. For instance, a sphere $\{C, r\}$ is obtained with the distance function $d(M, sphere) = |\overrightarrow{MC}| - r$, which gives the criterion function $\frac{1}{2}(1 - \frac{d}{r_{voxel}})$ that is greater than 1 if the voxel is outside the shape, less than 0 if the voxel is inside, and in [0, 1] if the voxel contains the frontier.

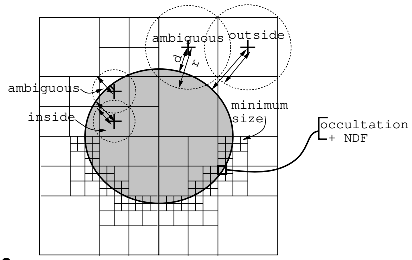

Fig. 2. Recursive volumetric construction of a shape (figured in 2D)

CSG operators like union or intersection are easy to implement: a new shape is combined with the previous ones by using a special drawing mode which adds values to existing ones in the voxels instead of setting the new value. Evaluating a whole CSG tree requires computing each sub-tree in separated volumes.

Primitives are also used to paint 'skeleton' representations such as L-systems and particle systems in the reference volume: successive segments are drawn in the volume using cone or cylinder primitives.

Triangular meshes are represented in the same way using the triangle primitive: the Euclidean distance from a point to a triangle can be easily obtained. The NDF is constant and is equal to the triangle normal. (Note that we draw the mesh surface and not the enclosed volume.)

We have also implemented implicit functions in the same spirit: a distance function can be approximated using the potential, since the only necessary properties for the function are that it be zero on the surface and to not increase any faster than the Euclidean distance (otherwise a voxel can be declared outside instead of ambiguous). Our approximation of the distance is $\frac{1}{\sqrt{pot}} - 1$, where $pot(M) = \sum \pi_i / d_i^2$, with π_i and $d_i(M)$ being respectively the weight and the Euclidean distance of a skeleton element (point, segment or triangular face).

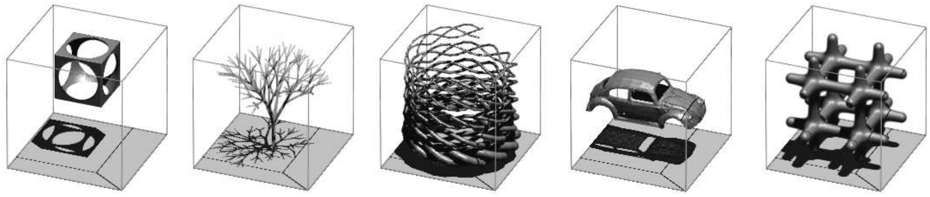

Fig. 3. Texel conversion of: CSG, L-system, particle system, mesh, implicit surface.

3.2 Volumetric models

Volumes are another kind of data: medical CT-scan data (explicit field) or hypertexture obtained from Perlin noise [Per85] (procedural field). Distances cannot easily be obtained since the data does not really represent a surface. We have to use a volumetric scan-conversion: we scan the data at the thinnest resolution, and we create the octree voxel hierarchy "on the fly" only where non empty space is found.

With Perlin noise, the normals are obtained by gradient computation. For pure volumetric density data such as CT-scan data, the density gradient has to be numerically estimated from a neighborhood (this gives poorer results, which shows that direct knowledge of the NDF is more important at this scale than the density distribution).

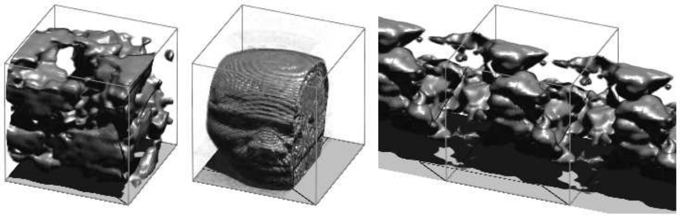

Fig. 4. Texel conversion of hypertexture (left) and tomographic image (middle). *right:* cyclic hypertexture pattern.

3.3 Discussion of the model

• Voxelizing a shape brings of course some approximations : from a short range, shapes seem made of bricks and are a bit blurred. To avoid such effects, the volume resolution has to fit the closest point of view requirements. This can be memory-consuming. But one has to keep in mind that texels are built to represent small details in complex scenes ; they are not supposed to handle short-range viewpoints. In such cases, a transition with polyhedral geometry can be made [Nom95].
• If duplicated without any variation, the texture will look very repetitive. This problem is dealt with in section 4.3.
• Color representation is not handled by the volumetric model which is a pure geometric representation (it is not easy to filter colored geometry). In previous papers [KK89, Shi92, Ney95b, Nom95], a color material is associated with a whole pattern. We address this problem in two ways : by merging separated texels associated to different materials, which is limited but easy to implement in volume rendering, and by implementing classical textures (picture or procedural) that are used inside texels.

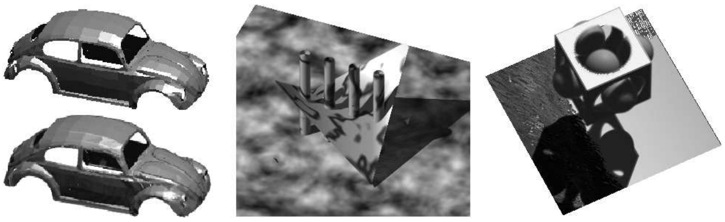

Fig. 5. *left:* real (top) and texelized (bottom) meshes. *middle:* textured texel. *right:* merging of two colored texels.

4 Mapping Texels

Previous publications on texels use a very simple mapping, requiring an underlying surface made of bilinear patches on which texels are placed. The base of each texel is fitted to the patch and their vertical sides are deformed in order to stick to adjacent texels. This constrains the surface mesh, and supposes that the orientation and the size of the texels correspond to the ones of the patches.

We specify in section 4.1 a general texel mapping in the same way that the 2D mapping is defined, offering new degrees of freedom. The idea is to introduce 3D texture coordinates inside the volumetric layer to specify the texel mapping. Note that unlike 2D textures where the color computation is separated from the ray scene traversal, in volumes rays have to traverse the texture. We describe this ray traversal in section 4.2. We explain in section 4.3 how to decrease the repetitive appearance of the mapping.

4.1 Mapping specification

We use surfaces consisting of triangles and bilinear patches. This allows the use of almost any mesh type, since polygons with more points can be decomposed into triangles and quadrilaterals. The size and the orientation of the polygons are independent of their texel counterparts.

We associate to each vertex a vector $\mathbf{H}$ called a *height vector* which controls the texel 'combing', and a vector $\mathbf{u} = (u, v, w)$ which gives the texture coordinate at the point. Also we define a scalar dw such that the vertex at the top of $\mathbf{H}$ has a texture coordinate $(u, v, w + dw)$. In most cases $w = 0$ and $dw = 1$. Kajiya's mapping corresponds to surfaces composed only of bilinear patches, and successive integer values for u and v at vertices (the texel bounds coincide to the patch bounds). Note that we use three different coordinate systems : the spatial one $\mathcal{S}$ where the coordinates are (x, y, z), the thick skin space $\mathcal{F}$ attached to the faces where the coordinates are (U, V, W) ($W = 0$ on the face, $W = 1$ on the top of the layer, and (U, V) are barycentric coordinates of the face), and the texture space $\mathcal{T}$ where the coordinates are (u, v, w) (see figure 6).

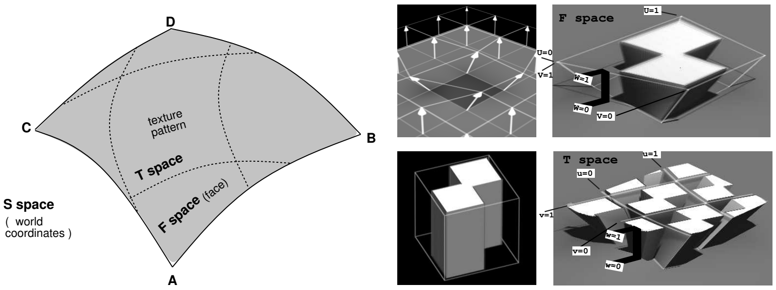

Fig. 6. *draft*: The parameterization $\mathcal{F}$ is normalized within the face. The parameterizations $\mathcal{S}$ and $\mathcal{T}$ are interpolated using $\mathcal{F}$ and knowing the values at the vertices. *top left*: the boxes corresponding to the faces. *bottom left*: the reference volume. *top right*: Kajiya's mapping : one texel fits one face (for clarity we have only mapped the central box). *bottom right*: general mapping defined by the (u, v, w) at the eight vertices (central box only).

Let us call the thick skin area corresponding to the faces *boxes* (in Kajiya's mapping boxes and texels are equivalent, i.e. $\mathcal{T} = \mathcal{F}$). Assuming that a triangle is a degenerated bilinear patch, the transformations $\mathcal{F} \rightarrow \mathcal{S}$ and $\mathcal{F} \rightarrow \mathcal{T}$ are trilinear interpolations (spatial coordinates and texture coordinates at any point are obtained from the values at the eight vertices).

As in previous work [KK89], a material pointer is associated to each face, which indicates the reference volume to use at this place, and we use classical Phong material description, such as ambient diffuse and specular color, and roughness. The material specification is slightly more complicated in our implementation since we allow the merging of texels in the same volume, the superposing of texel layers, or the achievement of operations such as scaling, permuting orientation, and so on. Moreover, Phong parameters can be defined by 2D texture functions such as an image map or a Perlin 3D noise. This is achieved by adding some items and linking the material descriptors used for the same face.

4.2 Texture traversal while rendering

We precompute a bounding sphere and a grid around the surface, and a bounding sphere for each face including the texel thickness (see figure 7). During the tracing of a ray, this leads efficiently to the first face intersected by the ray. Texels stick to each other so that in most cases only the upper and lower surfaces have to be tested for intersection.

When entering inside the skin space, we leave classical ray-tracing. The ray traversal inside this space uses the neighborhood coherence of faces, so that we just have to

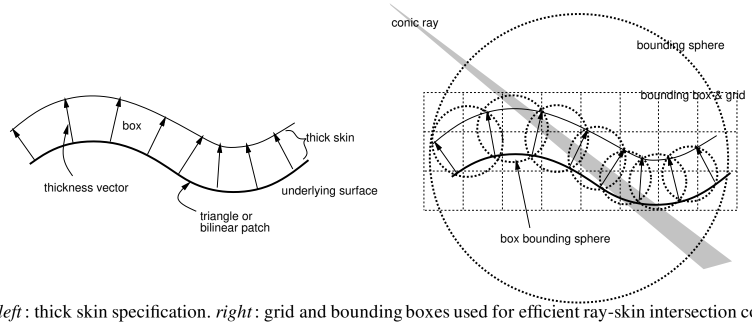

Fig. 7. *left* : thick skin specification. *right* : grid and bounding boxes used for efficient ray-skin intersection computation.

compute the point where the ray leaves the box, thus providing the next crossed box. This part of the traversal is described in previous articles [KK89, Ney95b]. Note that we only have to compute intersection between a ray and bilinear patch, which leads to a simple 2×2 Q_1 system[1] to solve.

At this stage we can switch to the $\mathcal{F}$ coordinates which are associated to the skin space. For Kajiya's simple mapping, $\mathcal{T}$ and $\mathcal{F}$ are identical, otherwise we have to cross also this $\mathcal{F}$ space before accessing to a single texel area. This stage (see figure 8) does not exist in previous methods. Note that in the skin coordinate system, the ray no longer propagates in a straight line. The trilinear deformation being small, we can either use a simple iterative scheme that gives correct intersections on the (u, v, w) grid[2], or approximate the path by a straight line, which is generally sufficient. In the last case, this stage of the traversal is a regular grid crossing, which is easy to implement.

Between two slices of the (u, v, w) grid we are in a texel, so we switch to the reference volume (i.e. the $\mathcal{T}$ space) that we cross linearly. This part of the volume traversal is described in our previous paper [Ney95b], and is similar to classical volumetric rendering (excepted the voxel content, and the local illumination computation). We use a kind of cone tracing to allow adaptative rendering : we know the thickness of the ray at this place, and we estimate the voxel size to be used in the octree regarding this aperture (this is exactly a 3D mip-mapping).

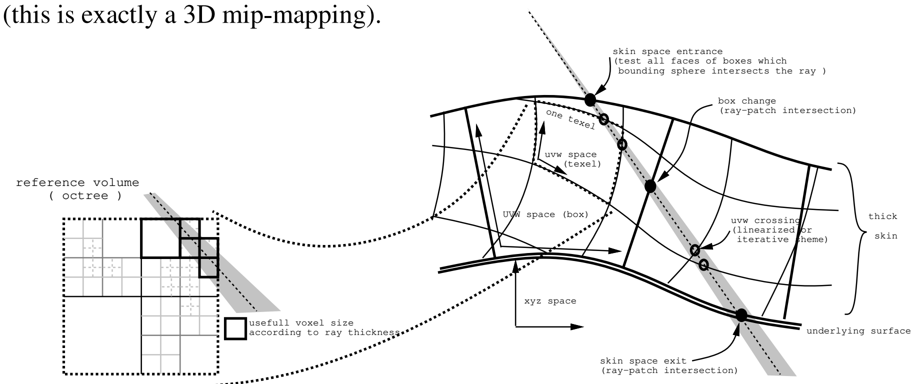

Fig. 8. Ray traversal inside the thick skin, crossing the boxes, then the texels, then the voxels at the adapted size.

[1] A Q_1 polynomial is of degree one in each variable, i.e. the highest degree term is $U.V.W$.

[2] The ray direction is expressed in $\mathcal{T}$ space. Assuming it is straight allows to obtain an approximation of the distance to the grid, that we used to follow the path in $\mathcal{S}$ space. We iterate while the point is not close enough to the (u, v, w) grid of $\mathcal{T}$ space, on the wanted slice.

4.3 Continuous and discrete jittering

A simple mapping induces a very regular aspect. This aspect can be improved by jittering the diverse parameters.

When the texture is continuous, the pattern has to be cyclic so that no frontier is visible. Continuous perturbations can be obtained by jittering texture coordinates, texel thickness, or thick skin orientation (by modifying height vectors). A Perlin noise is well-suited to this purpose.

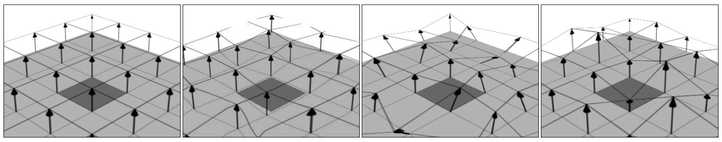

Fig. 9. Simple mapping ; jittering of texture coordinates, vectors direction, vectors length.

When the pattern consists of an isolated object, there is no longer a continuity constraint. Then, more manipulations can be used : alternating reference volume and material, displacing and scaling the texel content, applying symmetries or $\pi/2$ rotations (i.e. either axis permutation).

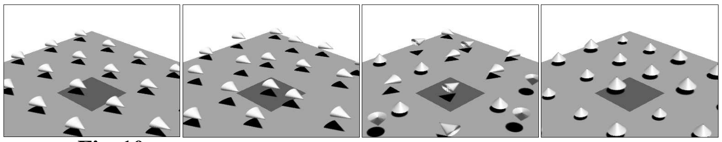

Fig. 10. Simple mapping ; perturbation due to displacing, rotating, scaling.

5 Results

We have applied our technique to generate verdant landscapes (see Appendix for the color images). The computations are done on an SGI Indy having a 200Mhz R4400 processor. The rendering is achieved with a single ray per pixel, at video size (768×576), and requires 20 minutes on average.

The first scene is a lawn, covering a hill made of 1400 bilinear patches. The mapping is jittered by modifying the height vectors. Each texel contains 16 grass blades and sometimes a flower. In total 22000 blades and 700 flowers are present. The blades are generated using parabolic trajectories, similar to particle systems, with the section having a ‘V’ shape. A 128^3 resolution is used for the volume.

The second scene represents 512 spaced trees on a flat land made of 1024 bilinear patches. The trees are modeled using 6 iterations of a L-system, yielding 2154 branches and 6336 leaves. The reference volume contains one isolated tree. Since the camera is very close to the trees, we have taken a 512^3 volumetric resolution (the volume is compressed more than 99.9%). A single tree model is used, and is modified along the texels by changing the size, the orientation, the position and the material. Note the continuous transition between further and closer trees.

The last example is forest covered mountains, using 25000 trees mapped on a 1404 bilinear patches surface. The texture is continuous so that the reference volume has a cyclic content, consisting of two trees clipped on the edges of the volume. Texture coordinates and height are jittered. The trees are seen from a far point of view most of the time, but the camera sometimes gets closer : a 256^3 resolution is used. Note that the scene contains around 200 million primitives (branches and leaves), reproduces fine shadows, gives smooth transitions while zooming, with very little aliasing, using a single ray per pixel.

6 Conclusion

Two important problems of the volumetric texture representation were texel construction and mapping. We have presented here how to convert various usual representations such as meshes or L-systems into texels and how to define and render mapped texels.

Volumetric texture is now a complete, convenient and efficient tool: a scene is conveniently built and animated using the textural aspect, the pattern can be designed using usual modeling tools, the rendering is done efficiently with low aliasing in ray-tracing, with the cost depending more on visual complexity than on data complexity.

A lot more can still be brought to the texel world. One can imagine specialized interactive tools to specify or manipulate the reference volume. On the other hand, texels are a new approach to the level-of-detail problem. Some studies may be conducted in order to use texels outside the scope of textures, as an alternate geometric representation to be used for distant viewpoints.

References

[dRa88] Phillippe de Reffye and al. Plant models faithful to botanical structure and development. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22(4), pages 151–158, August 1988.

[FR86] Alain Fournier and William T. Reeves. A simple model of ocean waves. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 75–84, August 1986.

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

[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.

[Ney96] Fabrice Neyret. *Textures Volumiques pour la Synthèse d'Images*. PhD thesis, Université Paris-XI - INRIA, 1996.

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

[Pa88] Przemyslaw Prusinkiewicz and al. 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.

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

[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.

[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.

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

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

[WP95] Jason Weber and Joseph Penn. Creation and rendering of realistic trees. In Robert Cook, editor, *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 119–128, August 1995.