HAL
archives-ouvertes.fr

Rendering Forest Scenes in Real-Time

Philippe Decaudin, Fabrice Neyret

▶ **To cite this version:**

Philippe Decaudin, Fabrice Neyret. Rendering Forest Scenes in Real-Time. EGSR04: 15th Eurographics Symposium on Rendering, Mark Ollila, Jun 2004, Norrköping, Sweden. pp.93–102. inria-00537458

HAL Id: inria-00537458
https://hal.inria.fr/inria-00537458

Submitted on 18 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

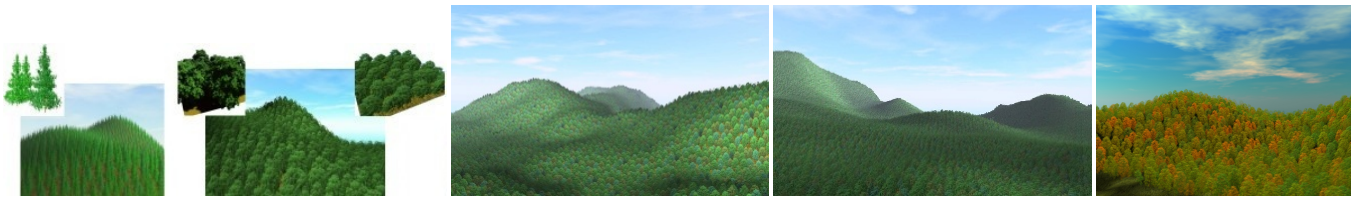
L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Eurographics Symposium on Rendering (2004)
H. W. Jensen, A. Keller (Editors)

Rendering Forest Scenes in Real-Time

Philippe Decaudin* Fabrice Neyret*

GRAVIR/IMAG-INRIA, Grenoble, France



Abstract

Forests are crucial for scene realism in applications such as flight simulators. This paper proposes a new representation allowing for the real-time rendering of realistic forests covering an arbitrary terrain. It lets us produce dense forests corresponding to continuous non-repetitive fields made of thousands of trees with full parallax.
Our representation draws on volumetric textures and aperiodic tiling: the forest consists of a set of edge-compatible prisms containing forest samples which are aperiodically mapped onto the ground. The representation allows for quality rendering, thanks to appropriate 3D non-linear filtering. It relies on LODs and on a GPU-friendly structure to achieve real-time performance.
Dynamic lighting and shadowing are beyond the scope of this paper. On the other hand, we require no advanced graphics feature except 3D textures and decent fill and vertex transform rates. However we can take advantage of vertex shaders so that the slicing of the volumetric texture is entirely done on the GPU.

Keywords

real-time rendering, natural scenes, 3D textures, aperiodic tiling, volumetric rendering, slicing, texcells.

1. Introduction

Forests seen from an airplane appear to be a strange material: while the vegetable cover looks like a quasi-continuous surface, one can also recognize individual trees, and sometimes see deep between them. The volumetric nature of the forest 'matter' is especially obvious when the viewer moves, due to omnipresent parallax and visibility effects. There is far too much foliage data to expect a direct real-time rendering of the leaves (to say nothing of aliasing), and classical mesh decimation does not apply to sparse data.

Applications requiring real-time rendering of natural scenes – *e.g.,* flight simulators and more recently video-games – are of such importance that early solutions had to be found to get *some sort* of forest populating landscapes: various alternate representations have been introduced to mimic trees and are still used nowadays.

Our purpose is to represent and render high quality dense forests in real-time. Our method is based on real-time volumetric textures. Our contributions include:

- The combination of two slicing methods to render volumetric textures efficiently: a simple one used for most locations, and a more complex one used at silhouettes.
- A novel camera-facing scheme for the slices of silhouette cells such that no vertex needs to be created by the CPU.

* e-mails: *Firstname.Name*@imag.fr
web: http://www-evasion.imag.fr/Publications/2004/DN04/
http://www.antisphere.com/Research/Forest.php

© The Eurographics Association 2004.

This permits a full GPU handling of the slices via vertex shaders.
- A new pre-filtering scheme allowing to achieve a correct non-linear 3D MIP-map in relation with our LOD adaptation.
- A LOD scheme for our volumetric textures, in relation with the pre-filtering scheme.
- A new aperiodic tiling scheme avoiding interpolation artifacts at tiles borders.

In this paper we do not deal with dynamic lighting and shadowing. Shadowing is an important and difficult topic in and of itself, and various existing shadowing algorithms could be adapted to our representation. Moreover, our implementation do not rely on any advanced pixel shader programmability. Using them would permit to get a per-voxel dynamic lighting. Our contribution concentrates on a representation allowing efficiency and quality (especially in terms of filtering, parallax richness and aperiodicity).

2. Previous work

In this section we review techniques allowing the real-time or interactive rendering of forests.

Billboards: Billboards are the most common tool for real-time rendering of forests. Thanks to their low cost, they are still considered the best choice in many recent industrial simulators. They are used in two different ways: classical billboards are small images that always face the camera [OGL98] (possibly with an axis constrained to be vertical) while cross billboards consist of two to four regular textured quads crossing each other (see fig. 1).The first method shows no parallax when the camera moves, which is only acceptable for axis-symmetrical objects and grazing view angles. Moreover, a tree slightly behind another is likely to pop in front at a given angle (fig. 1 right), so billboard-based forests are usually sparse. The second method shows artifacts whenever one of the quads is seen at a grazing angle.

To render a full dense forest would mean rendering millions of textured quads, which is very expensive even with modern graphics hardware. Moreover, there is no simple LOD scheme to gather individual billboards.

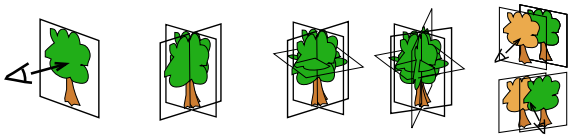


Figure 1: From left to right*: billboards, cross billboard, 3-cross billboard, 4-cross billboard, popping of billboard for a small camera motion.*

Other image based methods: One way to avoid the obvious artifact of cross billboards at grazing angles is to fade the billboards that are not facing the camera enough [Jak00]. This introduces two new artifacts which show up when one moves around a tree (see fig. 2,left): a ghosting effect (duplicated features) and transparency variation (blending two half-faded textures is not equivalent to a single opaque texture).

The same idea can be used with a whole set of images taken from various view angles [PCD*97, MNP01]. This reduces the artifacts because the difference between nearby images is smaller. But selecting and blending the images gets costly and could not be done in real-time for a whole forest: at least 3 textures have to be combined, and simply fetching 3 textures instead of a single one is about 2 to 3 times slower on recent GPUs [3DM].

Relying on even more images would correspond to bidirectional textures [SvBLD03] and light-fields [LH96, GGSC96]. But the huge amount of image data may not fit the graphics memory.

[Max96, MDK99] combine a hierarchical model with multi-layered depth images. Rendering is not real-time.

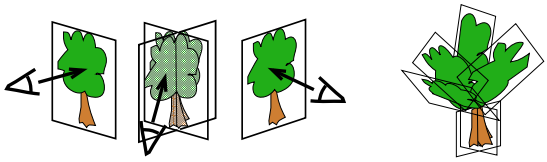


Figure 2: Left*: image fading with the incidence angle: the intermediate level is half-transparent and features are doubled (ghosting effect).* Right: *Simplified textured tree.*

Simplified textured trees: Another way to rely on textures is to build manually or using dedicated tools an extremely simplified tree with a few dozen to a few hundred polygons approximating the foliage distribution (see fig. 2,right). This has been used in video games and simulators for isolated trees close to the viewer but cannot be used for representing dense forests in real-time.

Lines and points: Reeves introduced particle systems in 1983 [RB85]. Even though it was not real-time, the idea of procedurally drawing many simple primitives has often been used to render vegetation. The number of primitives drawn can be adapted to the distance to ensure interactive frame rate [WP95].

In the same spirit, point-based approaches have been recently introduced [PZvBG00, SD01, DVS03, DCSD02]: objects are rendered using a set of points having roughly the size of a pixel.

These primitives are very convenient. But an important issue is that they rely on lots of vertices, and each vertex requires a geometric transform (projection, clipping) plus possibly a bus transfer. The vertex transform cost limits the performance in the case of huge sparse geometry such as forests

© The Eurographics Association 2004.

since the transform rate[1] is about 20 times slower than the fill rate[2] on recent graphics boards[3DM]. E.g., if the target frame rate is $60Hz$ at 1280×1024 resolution, one can redraw each pixel 20 times if textured polygons are used, while one can only draw each pixel once using points. The issue is that sparse geometry such as trees behaves like a globally transparent volume: a lot data projects to the same pixel, and there is no easy way to cull occluded data in advance.

Beside this performance issue, the main drawback of point-based (and line-based) methods is that distance adaptation is done by suppressing elements, which can induce popping. Moreover, the primitives are generally made opaque to avoid sorting and blending costs, which prevents proper antialiasing.

Volumetric textures: The volumetric textures approach consists of mapping a 3D layer on a surface using a 3D data set as texture pattern. This is especially adapted to a layer of continuous vegetation covering a landscape. It was first introduced in ray-tracing [KK89, Ney98] and later adapted to hardware rendering [MN98, LPFH01].

To rely on hardware acceleration the volume is rendered using textured slices. This has nice properties: the parallax is perfect because the generated fragments are really 3D (*i.e.,* at their proper depth), and the filtering is smoothly managed by the texture hardware (which has no equivalent for polygonal data). A lot fewer polygons are required than with the other approaches since each instance of the pattern corresponds to a portion of forest which can contain several trees.

The slicing must be adapted to the viewpoint otherwise one could see between the slices at grazing angles.

- [MN98] switch between 3 slicing directions: one 'horizontal' (parallel to the ground) and two 'vertical' (orthogonal to the ground and following the u or v texture parameterization).
- [LPFH01] use only slicing parallel to the surface and adds *fins* (edges extruded along the normal) near the silhouette. Note that this works well for fur (the goal of the paper) because fur is very homogeneous, but would fail for heterogeneous data like trees.
- Real-time volume rendering tools such as Volumizer [Ope] generally prefer to rely on slices that are facing the view direction. This way, one can never see between slices. Moreover, better antialiasing can be done (because slices not facing the camera are over-filtered by isotropic MIP-mapping). Unfortunately, it has several drawbacks in our context (see section 3). In particular, the computation of the adaptive slicing of a 3D layer mapped on a surface can get complicated. Moreover, one must recompute the geometry (*i.e.,* slices) at every frame, which implies that all this data must be transferred to the GPU at every frame.

[1] Number of transformed (i.e. projected) vertices per second.
[2] Number of textured pixel fragments rasterized per second.

Aperiodic tiling: Recently, several methods for mapping textures aperiodically have been introduced [Sta97, NC99, CSHD03]. They consist of preparing a set of compatible square or triangular patterns with appropriately chosen boundary conditions (*i.e.,* matching edges) when tiling the surface (see fig. 3). The tiles might also be rotated depending on the method. This also lets us represent textured area edges [NC99] which is important since textures rarely cover the whole surface. These methods are totally compatible with 3D textures.

Figure 3: Left*: Four edge-compatible triangular tiles.* Right: *Set able to represent textured area boundaries.*

3. Issues of 3D textures

3D textures are now a standard feature of graphics APIs. They are very convenient for implementing volumetric textures or volume rendering: the quadrilinear MIP-map is managed by the hardware and slices can have arbitrary orientation (*e.g.,* facing the camera). However, current implementations suffer numerous drawbacks:

- The mag and min[3] filtering are linear. This makes sense in 2D when four texture pixels lying on a face have to be averaged, but not for two texture voxels aligned along the view-ray direction: occlusion effects make the filtering non-symmetrical (see fig. 4). Filtering 3D data should separate the plane normal to the view direction – where linear filtering applies – and the view direction where the filter should use the transparency blending equation. Implementing a correct mag filter in the general case is not obvious, and implementing a correct min filter is impossible because occlusion is a view-dependent phenomena.

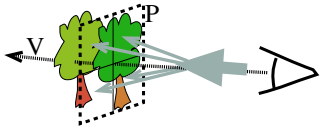


Figure 4: *Voxels that are on the same plane P orthogonal to the view direction V can be filtered linearly. Voxels along V should be blended using the transparency formula. Thus the filtering of a 3D neighborhood should not be linear.*

- Despite the 3D storage, the rendering primitive is still 2D: one must slice the volume to render it. The issue is that GPUs set the MIP-map level based only on the parameterization of the 2D slices regardless the *slicing rate* (*i.e.,* the sampling orthogonal to slices). So the programmer must ensure than the slicing rate fits the MIP-map level chosen by the GPU (which is not known to the CPU).

[3] Mag filtering corresponds to the interpolation between pixels. Min filtering corresponds to MIP-mapping.

© The Eurographics Association 2004.

• On current GPUs the MIP-mapping of 3D textures is isotropic. This means that the sampling density in depth must be equal to the sampling density parallel to the screen. This is wasteful since an error in depth might have little consequence on screen.
• This is even worse for slices non-parallel to the screen: with isotropic filtering a slanted slice is MIP-mapped relying on the most compressed projected parameterization. But the resulting averaging applies to the two other directions as well. At grazing angles the volume data will be totally blurred in the slicing direction regardless of the slicing rate.
• Volume MIP-mapping is rarely used for volumetric rendering applications, and it appears that there are several bugs in the implementation of 3D textures even on recent boards.[4]

4. Our approach

Our representation relies on real-time volumetric textures [MN98, LPFH01]. We increase both their quality and performance and we handle richer forest attributes.

As illustrated in section 2 and 3 numerous slicing algorithms are possible, with various consequences on properties, quality and efficiency. The case of volumetric texture rendering is quite different than classical volume rending applications (*e.g.,* medical) in that:

• The overall number of voxels in the scene is enormous, and will generally be undersampled. In contrast, traditional volumetric applications highly oversample the data at rendering.

• The volumes are mapped on a terrain and therefore are distorted, so that the overall volume to be sliced has a complicated shape (*i.e.,* it is not a simple cube).

• The overall number of slices' polygons is so large that generating them from the CPU at each frame cannot be accomplished in real-time[5].

• The volume layer thickness is small compared to its horizontal width implying that each slice has a small surface on screen.

Using slices parallel to the surface is efficient since the mesh vertices already stored on the GPU can be simply offseted. But it is inappropriate for grazing view angles because one would see between slices. Introducing vertical plates (or *fins*) as proposed in [LPFH01] only works well for fuzzy data such as fur, while for contrasted data such as trees, the grazing slices will show-up clearly and the features will not map onto their counterparts in fins.

Generating slices facing the viewer can get complicated for a complicated volume shape. Moreover, it is not GPU-friendly since all vertices would have to be generated on the CPU at each frame, and their transfer to the GPU would be the bottleneck.

Moreover, the forest covers the landscape on a very wide range of distances, so levels of detail and filtering (MIP-mapping) must be addressed. Note that the 3D quality filtering of volumes is quite different from the classical 2D filtering as explained in section 3. In particular, occlusions make it highly non-linear, and the default isotropic filtering for MIP-map is not acceptable.

We propose a solution to these problems by combining two slicing methods. The first one uses slices parallel to the terrain and is adapted for most parts of the scenes (we also introduce a new quality filtering and LOD scheme for it). The second is a slicing scheme that uses slices nearly facing the viewer (fig. 6,middle-right), using a new GPU-friendly algorithm: the slices are an offset of the tilted base triangle (fig. 9).

For both slicing schemes, we define a LOD-scheme in order to adapt the cost to the distance, using a new non-linear pre-filtering scheme. This is especially important for natural scenes since the amount of data projecting to a given pixel grows quadratically with the distance. The best way to adapt in terms of quality is to average – i.e. to filter – the 3D data. We describe how the volumetric texture data can be correctly filtered. To enrich the appearance of forests, we want to handle natural variations: we draw on aperiodic tiling [NC99] to avoid texture repetitiveness.

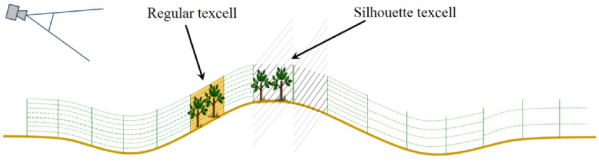


Figure 5: *Our slicing scheme.*

The characteristics of our method are the following:

• We represent forest coverage by a volumetric texture layer.

• Repeated instances of the pattern (which we call *texcells*[6]) have a prism shape: a base triangle on the ground vertically extruded.

• We implement two different kinds of texcells: A simple one (*regular texcell*) with good quality and efficiency except at the silhouette and another (more expensive) one to be used near the landscape silhouette (*silhouette texcell*), illustrated in fig. 5.

• Regular texcells are sliced parallel to the ground (fig. 6,left). We introduce a new representation with which we can filter a 3D texture non-linearly (thus addressing the

[4] Several have been confirmed by both nVIDIA and ATI, and should be corrected soon. However, they limit the performance of our current implementation.
[5] It is important to note that a real-time forest renderer must keep high performance even when processing *thousands* of trees.
[6] After [KK89] who introduced the term *texels*. Our term is phonetically similar but it avoids the confusion with texture pixels that are also called *texels* in numerous documents.

© The Eurographics Association 2004.

issues mentioned in section 3). This lets us manage levels of detail by adapting the number of slices to the distance.

• Silhouette texcells are sliced parallel to the screen (fig. 6,middle,right). The slices are an offset of the tilted base triangle (see fig. 9). This avoids creating and sending new polygons from the CPU: a simple vertex shader can translate the vertices of a GPU-resident mesh. We use hardware 3D textures to store the volumetric pattern[7]. The levels of detail are created by adapting the slicing rate to the 3D MIP-map level.

• We define an aperiodic mapping of the texcells in the spirit of [NC99] by creating several edge-matching patterns. In our current implementation we rely on patterns containing precomputed shading and shadows, so the orientation is constrained: the patterns cannot be rotated as in [CSHD03]. We propose a scheme in section 7 showing that at least 8 patterns are required.

Creating the volume data

Like [MN98] we create our volume data from standard geometry (in our case, a piece of polygonal forest) using an off-line renderer. The camera is orthographic and is looking down. We obtain each slice by setting the near and far clipping planes at distance $\frac{\delta}{2}$ above and below the slice location (δ is the distance between two slices), then rendering the data. In our experiments we used a commercial package and activated shadows (clipping planes should not prevent objects out of the clip volume to cast shadows).

All our textures are alpha-premultiplied to avoid texture interpolation artifacts [PD84]: the color component $C \in \{R,G,B\}$ stores the C value times the opacity A. We denote premultiplied colors $\bar{C}$ to avoid confusion.

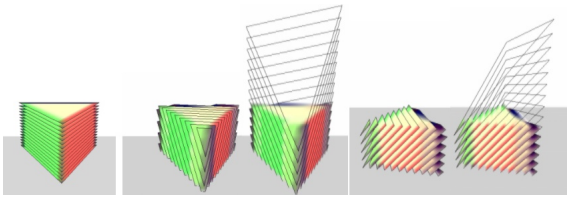


Figure 6: Left*: a regular texcell.* Middle*: a silhouette texcell (facing the camera) with and without clipping of the empty top.* Right: *a side view of it.*

5. Regular texcells

Following [LPFH01] we prefer using whenever possible the canonical slicing parallel to the landscape because it can be obtained simply by offsetting the base surface. This lets us rely on a set of 2D textures instead of 3D textures, which avoids the problems described in section 3, and is efficient since the geometry does not need to be rebuilt and transferred from the CPU at each frame (bus transfer can be an important bottleneck in applications using complex geometry). Our goal is to define a multiscale model of volumetric texture. This requires correct filtering. We show in this section that since the slicing direction is constant, we can improve upon linear interpolation by efficiently emulating non-linear 3D filtering. This provides us a degree of freedom to control levels of detail.

Our representation

As explained in section 3 and fig. 4, 3D filtering should not be linear: the data along the slicing plane and the slicing direction must be treated separately. Instead of 3D textures, we rely on a set of 2D textures corresponding to the slices. Each 2D texture has an associated MIP-map pyramid (since linear filtering *in* the slice plane is justified). Moreover, we construct a level of detail pyramid of this set (see fig. 7), relying on non-linear filtering: At the finest level, we have $n = 2^N$ slices whose textures are $L \times L$; call S_N this level. At level S_i we have 2^i slices (representing an aggregation of 2^{N-i} slices from the finest level). The size of textures at level S_i is $\frac{L}{2^{N-i}} \times \frac{L}{2^{N-i}}$. Each texture, at every level, is MIP-mapped.

Creating the texture set

As explained in the previous section the base textures of S_n correspond to the original unfiltered 3D volume (created offline).

A set S_i could be generated the same way as S_N by rendering thicker slices of geometry, but we can obtain the same result with no extra rendering by blending the textures in S_N using the transparency equation $\bar{C}_{front} + (1 - A_{front})\bar{C}_{back}$: the 2D texture $S_i[j]$ (i.e. the j^{th} slice of the set S_i) is obtained by blending $S_{i+1}[2j]$ and $S_{i+1}[2j+1]$. Then each resulting 2D texture is MIP-mapped.

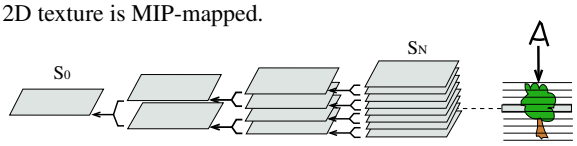


Figure 7: Right*: slicing real geometry using clipping planes.* Left*: Pyramid of 2D texture sets $S_i, i = 0..n$.*

Rendering

We can now freely tune the number of slices to implement levels of detail: We set the slicing rate according to the distance to the observer and to the incidence angle. If best quality is required we activate the hardware anisotropic filtering (very efficient on recent hardware, but available only for 2D textures), which makes the mag and min filtering really correct in the three directions.

A last issue arises because the distance between slices along a ray depends on the ray (see fig. 8): at grazing angles the sampling is coarse. Since the same number of slices

[7] As with all 3D textures, the 2D slices simply rely on (u,v,w) texture coordinates at their vertices pointing at the correct 3D location. So rotating slices and updating the (u,v,w) accordingly does not rotate the represented features.

© The Eurographics Association 2004.

are traversed as for normal incidence, the same total opacity is obtained despite the longer path within the volumetric texture. So the opacity of slices must be adjusted to account for the real sampling rate: $A(dl) = 1-(1-A(\delta))^{\frac{dl}{\delta}}$ where $A(dl)$ is the opacity along a length dl. We approximate this using the first terms of the Taylor series: $A(dl) \approx \frac{dl}{\delta} A(\delta)$. Values greater than 1 are clamped, so opaque voxels are handled correctly. The distance δ_v between two slices along a ray (see fig. 8) is $s_z|\frac{\vec{N}.\vec{z}}{\vec{N}.\vec{V}}|$

with $\vec{N}$ the normal to the terrain, $\vec{V}$ the view direction, $\vec{z}$ the vertical and s_z the vertical scaling (in case the height of the mapped forest differs from the original forest sample). So we have to set an opacity multiply factor at each vertex $\boxed{\pi_A = \frac{s_z}{\delta}|\frac{\vec{N}.\vec{z}}{\vec{N}.\vec{V}}|}$ that will be interpolated on the face and multiplied by the texture (`modulate` mode) during rasterization. Note that this can easily be done by a vertex shader.

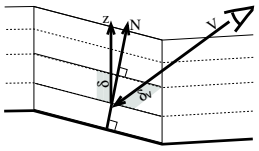


Figure 8: *Opacity factor.*

6. Silhouette texcells

The previous model is efficient and accurate as long as the incidence is not grazing: in such case the parallax error gets high and one might even see between the slices. This situation corresponds to landscape silhouettes. In this situation we switch to our second model, which relies on orientable slicing. In this section we explain how to compute each slice's location and how to deal with levels of detail.

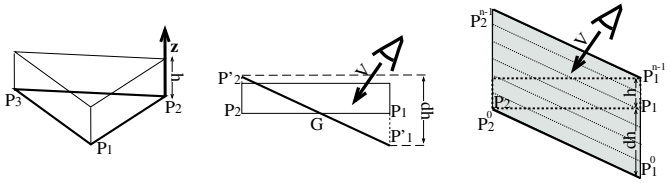


Figure 9: Left*: The prism shape texcell.* Middle*: Tilting the base triangle to face the camera.* Right*: The sliced volume.*

Tilting the slices

We tilt the slices (see fig. 9) by moving vertices vertically. This allows us to *transform* polygons pre-stored on the GPU instead of *creating* them from the CPU, avoiding a bus transfer bottleneck.

Given a prism of height $h\vec{z}$ and base triangle $(P_1P_2P_3)$, let G be the center of the base triangle and $\vec{V}$ the view direction. Assuming the viewpoint is distant, the n slices are all parallel. So we just have to determine the tilting of $(P_1P_2P_3)$ so that it faces the camera: $P'_i = P_i + k_i\vec{z}, i = 1..3$. Letting $(P'_1P'_2P'_3)$ facing the camera means that $\overrightarrow{GP'_i}\cdot\vec{V} = 0$, *i.e.*, $\boxed{k_i = -\frac{\overrightarrow{GP_i}.\vec{V}}{\vec{z}.\vec{V}}}$. Let $k_{min} = min(k_i)$ and $k_{max} = max(k_i)$ (note that $k_{min} \le 0$ and $k_{max} \ge 0$).

We have to set the first and last slices $(P_1^0P_2^0P_3^0)$ and $(P_1^{n-1}P_2^{n-1}P_3^{n-1})$ so as to insure that the whole volume of the prism is sampled (fig. 9,right). Since the triangle slanting adds a slope $d_h = k_{max} - k_{min}$ (fig. 9,middle), the sliced volume is now a prism of height $(h+d_h)\vec{z}$ and base triangle $(P_1^0P_2^0P_3^0)$. $(P_1^0P_2^0P_3^0)$ is a vertical translation of $(P'_1P'_2P'_3)$ so that the triangle is just below the ground: $P_i^0 = P_i + (k_i - k_{max})\vec{z}$. The other slices are simply obtained by offsetting the base triangle: $P_i^j = P_i^0 + \frac{j}{n-1}(h+d_h)\vec{z}$, $j = 0..n-1$.

However, for grazing view angles the slant goes to infinity, so we want to limit the slant (at the price of an approximate facing). Let k_M be the maximum slant allowed and $d'_h = min(d_h, k_M)$. We now have $\boxed{P_i^j = P_i + \left(\frac{d'_h}{d_h}(k_i - k_{max}) + \frac{j}{n-1}(h+d'_h)\right)\vec{z}}$, which is easily implemented on a vertex shader. In our implementation, we chose $k_M = 2h$.

Note that a part of the sliced volume is out of the 3D texture space (the `border color` attribute let us set this region transparent). We do not want these top and bottom empty regions to cost. Setting the `alpha test` culling discards the transparent fragments, but these are still rasterized. The new coming *clip registers* feature will soon permit to clip the slice polygons in the vertex shaders.

Slicing rate and filtering

The slices sample a MIP-mapped 3D texture. This 3D linear filtering is justified assuming the silhouettes are distant: when the resolution of the 3D texture is small the opacity of the voxels is low, thus the effect of occlusion is weak.

As stated in section 3 GPUs set the MIP-map level independent of the *slicing rate*. We must ensure instead that the slicing rate fits the MIP-map level. 3D MIP-mapping is isotropic so the distance δ between slices should not be less than the 'voxel' size or some information will be lost. Volume rendering applications tend to choose higher sampling rates to improve the reconstruction quality. But we prefer to improve rendering time, so we choose a rate as close as possible to the voxel size. We have $\delta = \frac{1}{n-1}(h+d'_h)\vec{z}.\vec{V}$. For a texture resolution L^3 and a MIP-map level l, we want $\delta = \frac{2^l}{L}$, which implies that $\boxed{n = 1 + \frac{L}{2^l}(h+d'_h)\vec{z}.\vec{V}}$.

7. Aperiodic tiling

To avoid repetitivity when mapping texcells onto a terrain we rely on the [NC99] aperiodic scheme (see section 2). Since our implementation assumes that the textures are preshaded we cannot rotate the patterns, which is the same situation as in [CSHD03]. We, too, consider separate boundary conditions in each direction (see fig. 10). In our case we

© The Eurographics Association 2004.

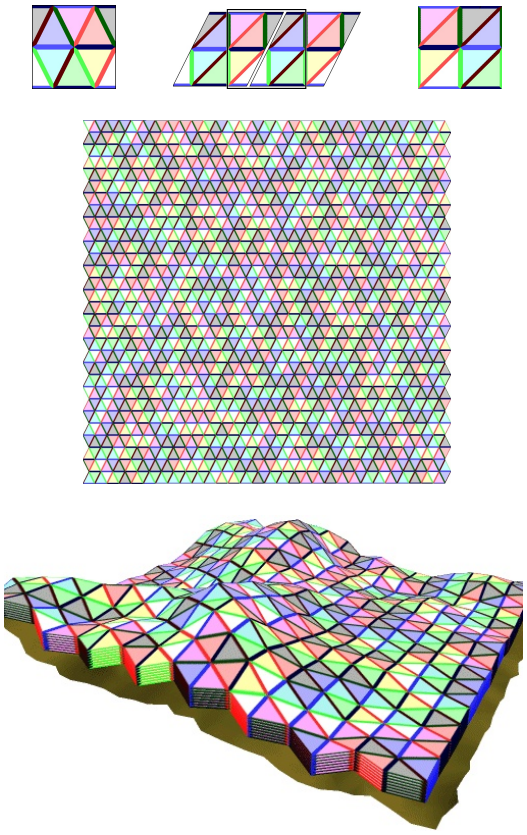


Figure 10: Top*: The reference texture* (left) *containing the 8 edge-matching triangular patterns. The wrapping is slanted (the bottom red and top yellow triangles share an edge). We shear and wrap the pattern* (middle) *to obtain a square wrapping* (right). Bottom*: aperiodic tilings showing the choices.*

have 3 directions (triangular tiling). If we assume 2 different boundary conditions in each direction (indicated by colors) we have 8 possible triplets, and thus 16 possible triangles (pointing north or south). To save memory we choose a subset of 8 triangles (4 pointing north and 4 pointing south), so as to use each triplet once. This neither breaks aperiodicity nor generates artifacts, as illustrated in fig. 10. If higher variability were required we could of course keep the 16 tiles. Moreover it is also possible to build more than one tile for one given triplet+direction combination: this lets us to create several kind of forest patches and to account for rare events (e.g. a higher tree).

Storing the patterns

Storing the patterns is an issue that has not been discussed in previous papers to our knowledge: when rendering a pixel close to a tile border, the hardware will linearly interpolate for both mag and min filters. The color stored *outside* the tile (taking the wrapping of the texture space into account) will be blended with the inside color and thus must be coherent to avoid artifacts on edges. We manage this by arranging the 8 tiles (shown on fig. 10,top-left) in the *reference texture* so as to fit their edge constraints[8]. Since each kind of edge is used by 4 tiles it appears twice: we call the two edges sharing the same boundary condition *twin edges*. In fact our wrapping is slanted: the bottom green and top grey triangles share an edge, but only square wrapping is tractable by hardware textures. To obtain a square wrapping, we shear and wrap the pattern (fig. 10,top-middle). The result texture is shown in fig. 10,top-right. Two aperiodic tilings showing the choices are shown on fig. 10,bottom.

Building the patterns

The boundary conditions (figured by edges colors in fig. 10) yield constraints when drawing the content of the reference texture:

• an object crossing an edge axis must be replicated on the twin edge. This means that the neighborhoods of twin edges are correlated, but not identical (only the content sampled exactly on their axis is identical). The smaller the objects, the less correlated the twin edges.

• Moreover, an object must never cross two edges (e.g. at corners), or it would correlate them and would be replicated on half of the cells.

Whatever their content (2D, 3D, color, bump...), the patterns have to be built to respect the constraints above. For representing forest we rely on geometrical models; numerous tree models are available (online or through commercial packages). Each tree can be represented by a bounding disk. We first place the trees that cross the boundaries, which must be replicated on the twin edges. (It is a good idea to start by placing one tree quite close to a corner – without crossing 2 edges – to avoid having a vicinity bias). Then we can freely place the other trees (with only the constraint of not crossing the triangle tile boundary). We did not implement an automatic placement of the objects populating the patterns. [CSHD03] describes how to produce unbiased Poisson-disk distributions.

8. Results

Implementation

We have implemented regular texcells, silhouette texcells and aperiodic mapping described in this paper, comprising the adaptive slicing and the non-linear filtering. Nevertheless, our implementation is limited:

- Scenes are lit by fixed directional lights (in our images a single light is used). Shading is precomputed and stored in the textures.

[8] Here 8 triangle tiles are used. If one wants to use square tiles, [DN04] addresses the problem of packing an arbitrary number of them.

© The Eurographics Association 2004.

- Our slices cannot be rotated (because our texture includes the lighting), which increases the number of tiles to create to match the combinatorics of edge boundary conditions.
- Our levels of detail are discrete (which induces some popping in the video): as in MIP-mapping, lod should be a float and two levels $\lfloor lod \rfloor, \lfloor lod+1 \rfloor$ should be combined at any time.
- We did not use any programmable shader: the regular texcells' vertices are stored on the GPU in a vertex buffer, and the silhouette texcells are generated by the CPU. However, this did not slow down the performance since the bottleneck is in the pixel fill rate.
- We used 3DS-Max for setting the reference forest sample and pre-rendering the textures. But the amount of geometry for only 30 trees (300k polygons each) made the interactive placement of trees painful, and the ray-tracing shadowing (compulsory with clipping planes) no longer worked, which explains the lack of shadows in our texcells.
- The lack of real scene-wise shading results in a constant illumination of the landscape. To compensate this, we estimate a pseudo-shading corresponding to the Lambert lighting of the terrain.

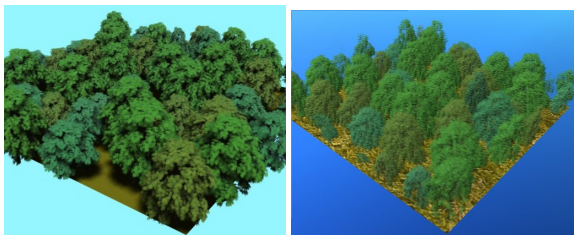


Figure 11: Left*: real geometry used for the reference texture.* Right*: corresponding texcells (without shadows).*

Results and performance

All the computations are done on a P4 at 1.7 GHz with a GeForceFX 5800 graphic card. Resolution was 640×480. The size of volumetric data was 128 slices of 256×256 32bits textures yielding 12 Mb of compressed texture storage on the GPU (3D texture + the 2D texture sets). Using our implementation, we were able to map terrains with a dense forest. We show our results in fig. 12. In fig. 11,left we can see the original forest pattern rendered by the offline renderer. Fig. 11,right shows the same pattern displayed in real-time by our program. Fig. 12,top (seq #1 on the video) shows a terrain mapped with this pattern. There are 576 tiles containing 4 trees each, so the scene includes roughly 2300 trees. The frame rate is 25 to 40 fps depending on the view and the tuning of parameters. Silhouette texcells are about 4 times more expensive to render than regular ones, therefore frame rate depends on the number of rendered silhouette texcells which varies with the view direction and the threshold used to switch between the two kind of texcells.

Fig. 12,bottom (seq #2 on the video) shows a more complex terrain with more texcells mapped: 9212 tiles representing a total of 37000 trees. The frame rate is 20 to 30 fps depending on the view and the tuning of parameters. The frame rate is inversely proportionnal to resolution, which means that the bottleneck lies in the fill rate. As one can see on the video, transitions between LODs of the same type of slices are quite unnoticeable. Transitions between the two different types of slices (regular and silhouette) can be noticeable if one focuses on it, but the popping is weak and quite acceptable for a real-time application.

Limitations

- The main assumption for volumetric textures is that the vegetable cover has *textural* characteristics: one cannot expect to model a scene with precise control of given trees. Conversely, in the sequence 2 on the video we unfortunately made a reference texture including one remarkable tree (high, narrow and dark). As a result one can see the correlated location of the rare instances of this tree which are aligned with the tiling. However, we can increase the variability by creating more tiles than the minimum set. Moreover, we can mix individual key-elements within the forest texture: usual geometric objects will be inserted coherently since 3D fragments are generated.
- Another characteristic of textures is that they are meant to cover everything continuously. E.g. if the terrain includes cliffs, the mapping has to be cut accordingly.
- Some characteristics of the hardware (and drivers) were disappointing, such as various problems with the 3D textures and not so high performance in situations supposed to be accelerated. But boards and drivers are evolving fast, so these issues should be fixed soon.

9. Conclusion and future work

We have described a multiscale scheme for real-time volumetric textures by defining new ways to represent, filter, and render them in real-time. We also introduced a new aperiodic scheme which avoids interpolation artifacts at tiles borders.

This lets us efficiently render large forest scenes. Since we compute a correct filtering – in contrast to previous volumetric texture methods – we get high quality rendered images, which is strengthened by the fact that we generate non-repetitive forests. Compare to previous real-time methods for rendering landscapes which treat individual trees represented by a few polygons, we show for the first time a wide and dense forest in real-time with high quality standards.

Despite our limited implementation we have shown that our results are already convincing: we can move interactively above a large forest which shows a continuous range of tree size. The tree appears really 3D, in contrast to billboards: we can move around them and show full parallax effects in real-time, and a tree never pops in front of another.

© The Eurographics Association 2004.

Figure 12: Top*: 576 tiles, 2300 trees in real-time (seq #1 on the video).* Bottom*: 9212 tiles, 37000 trees in real-time (seq #2 on the video).*

Therefore there is room to improve the features and test enormous scenes. Some limitations are linked to the evolution of graphics hardware: we expect 3D texture will be soon totally usable. Better hardware culling (especially for transparent polygons) would increase the performance a lot: we could draw the slices front to back and thus avoid useless overdrawing.

For future work, the most important point to improve is the dynamic lighting and shadowing: not only it is a useful feature per se but also it would let us rotate the patterns during the mapping. This would decrease the number of necessary 3D patterns to store. Some existing real-time shadow-mapping schemes might be adapted to our representation. In addition, dynamic lighting could be handled by using pixel shaders to compute per-voxel lighting [CDN04].

Furthermore, since our volumetric textures produce fragments at the correct 3D location, texcells can be mixed with other 3D objects such as peculiar trees, grass or small bushes.

Acknowledgments

We wish to thank John F. Hughes and Laks Raghupathi for rereading a previous version of this paper, and Adrien Treuille for rereading this version.

Figure 13: *A forest made of about 30000 trees in real-time with fog effect.*

References

[3DM] 3DMARK: XbitLabs benches and comments. http://www.xbitlabs.com/articles/video/display/geforcefx-5900ultra.html. 2, 3

[CDN04] COHEN F., DECAUDIN P., NEYRET F.: GPU-based lighting and shadowing of complex natural scenes. *Siggraph'04 Poster (conf. select CD-ROM and conf. DVD-ROM).* Los Angeles, USA (2004). www-evasion.imag.fr/Publications/2004/CDN04/. 9

[CSHD03] COHEN M. F., SHADE J., HILLER S., DEUSSEN O.: Wang tiles for image and texture generation. *ACM Transactions on Graphics (Siggraph'03 conference proceedings) 22*, 3 (July 2003), 287–294. 3, 5, 6, 7

[DCSD02] DEUSSEN O., COLDITZ C., STAMMINGER M., DRETTAKIS G.: Interactive visualization of complex plant ecosystems. In *Proceedings of IEEE Visualization '02* (2002), pp. 219–226. 2

[DN04] DECAUDIN P., NEYRET F.: Packing square tiles into one texture. In *Proceedings of Eurographics '04 (short papers)* (2004). www-evasion.imag.fr/Publications/2004/DN04b/. 7

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential point trees. *ACM Transactions on Graphics (Siggraph'03*

© The Eurographics Association 2004.

conference proceedings) 22, 3 (July 2003), 657–662. 2

[GGSC96] GORTLER S., GRZESZCZUK R., SZELISKI R., COHEN M.: The lumigraph. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), Annual Conference Series, pp. 43–54. 2

[Jak00] JAKULIN A.: Interactive vegetation rendering with slicing and blending. In *Proc. Eurographics 2000 (short papers)* (Aug. 2000), Eurographics. 2

[KK89] KAJIYA J., KAY T.: Rendering fur with three dimensional textures. *Computer Graphics (Proceedings of SIGGRAPH 89) 23*, 3 (July 1989), 271–280. 3, 4

[LH96] LEVOY M., HANRAHAN P.: Light field rendering. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), Annual Conference Series, pp. 31–42. 2

[LPFH01] LENGYEL J. E., PRAUN E., FINKELSTEIN A., HOPPE H.: Real-time fur over arbitrary surfaces. *2001 ACM Symposium on Interactive 3D Graphics* (March 2001), 227–232. 3, 4, 5

[Max96] MAX N.: Hierarchical rendering of trees from precomputed multi-layer z-buffers. In *Eurographics Workshop on Rendering '96* (1996), Springer-Verlag, pp. 165–174. 2

[MDK99] MAX N., DEUSSEN O., KEATING B.: Hierarchical image-based rendering using texture mapping hardware. In *Eurographics Workshop on Rendering '99* (1999), pp. 57–62. 2

[MN98] MEYER A., NEYRET F.: Interactive volumetric textures. In *Eurographics Rendering Workshop 1998* (July 1998), pp. 157–168. 3, 4, 5

[MNP01] MEYER A., NEYRET F., POULIN P.: Interactive rendering of trees with shading and shadows. In *Eurographics Workshop on Rendering* (Jul 2001), pp. 183–196. 2

[NC99] NEYRET F., CANI M.-P.: Pattern-based texturing revisited. In *SIGGRAPH 99 Conference Proceedings* (Aug. 1999), ACM SIGGRAPH, Addison Wesley, pp. 235–242. 3, 4, 5, 6

[Ney98] NEYRET F.: Modeling animating and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (Jan.–Mar. 1998). 3

[OGL98] *Siggraph Course Notes CD-ROM. Advanced Graphics Programming Techniques Using OpenGL*. Addison-Wesley, 1998. http://www.sgi.com/software/opengl/advanced98/notes/. 2

[Ope] OPENGL VOLUMIZER: SGI. http://www.sgi.com/software/volumizer/whitepaper.pdf . 3

[PCD*97] PULLI K., COHEN M., DUCHAMP T., HOPPE H., SHAPIRO L., STUETZLE W.: View-based rendering: Visualizing real objects from scanned range and color data. In *Eurographics Rendering Workshop 1997* (June 1997), pp. 23–34. 2

[PD84] PORTER T., DUFF T.: Compositing digital images. In *Computer Graphics (Proceedings of SIGGRAPH 84)* (July 1984), vol. 18, pp. 253–259. 5

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. *Proceedings of SIGGRAPH 2000* (July 2000), 335–342. ISBN 1-58113-208-5. 2

[RB85] REEVES W., BLAU R.: Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *Computer Graphics (SIGGRAPH '85 Proceedings)* (July 1985), vol. 19(3), pp. 313–322. 2

[SD01] STAMMINGER M., DRETTAKIS G.: Interactive sampling and rendering for complex and procedural geometry. In *Rendering Techniques 2001 (Proceedings of the Eurographics Workshop on Rendering 01)* (2001), Eurographics. 2

[Sta97] STAM J.: *Aperiodic Texture Mapping*. Tech. Rep. R046, European Research Consortium for Informatics and Mathematics (ERCIM), Jan. 1997. http://www.ercim.org/publication/technical_reports/046-abstract.html. 3

[SvBLD03] SUYKENS F., VOM BERGE K., LAGAE A., DUTRÉ P.: Interactive rendering with bidirectional texture functions. *Computer Graphics Forum 22*, 3 (Sept. 2003). 2

[WP95] WEBER J., PENN J.: Creation and rendering of realistic trees. In *Computer Graphics (SIGGRAPH '95 Proceedings)* (Aug. 1995), pp. 119–128. 2

© The Eurographics Association 2004.