Interactive Rendering of Trees with Shading and Shadows

Alexandre Meyer Fabrice Neyret Pierre Poulin
{Alexandre.Meyer|Fabrice.Neyret}@imag.fr poulin@iro.umontreal.ca
iMAGIS-GRAVIR/IMAG-INRIA LIGUM

Abstract. The goal of this paper is the interactive rendering of 3D trees covering a landscape, with shading and shadows consistent with the lighting conditions. We propose a new IBR representation, consisting of a hierarchy of Bidirectional Textures, which resemble 6D lightfields. A hierarchy of visibility cube-maps is associated to this representation to improve the performance of shadow calculations.
An example of hierarchy for a given tree can be a small branch plus its leaves (or needles), a larger branch, and the entire tree. A Bidirectional Texture (BT) provides a billboard image of a shaded object for each pair of view and light directions. We associate a BT for each level of the hierarchy. When rendering, the appropriate level of detail is selected depending on the distance of the tree from the viewpoint. The illumination reaching each level is evaluated using a visibility cube-map. Thus, we very efficiently obtain the shaded rendering of a tree with shadows without loosing details, contrary to mesh simplification methods. We achieved 7 to 20 fps fly-throughs of a scene with 1000 trees.

Keywords: Real-time rendering, natural scenes, forests, IBR, levels of detail, billboards

URL: http://www-imagis.imag.fr/Publications/2001/MNP01/index.html

1 Introduction

Walk- and fly-throughs of natural landscapes with the best possible visual quality have been a continuous challenge since the beginning of Computer Graphics. In real-time applications such as simulators and games, users want ever more convincing realism (*i.e.,* more trees, better looking trees). In off-line applications such as impact studies and special effects, users want the rendering software to compute more rapidly.

The rendering of tree covered landscapes is especially demanding, because of the total amount of details, the complex lighting situation (shading of small shapes, shadows, sky illumination), and the complex visibility situation (there are no large blockers that can be efficiently exploited).

Fortunately, trees also have interesting properties. Their complexity of appearance is counterbalanced by significant redundancy of elements, and their structure is naturally hierarchical. Trees are composed of a trunk and main branches, which group small branches together. Boughs are made of leaves or needles; and leaves, needles, boughs, branches, and even trees resemble each other. These properties allow many modeling simplifications: elements of a family can be represented as instances of a few base models, and complex objects as a collection of similar, simpler, objects.

In this paper, we exploit this notion by considering alternate and hierarchical representations and techniques to efficiently render trees, including shading and shadows.

Our goal is to render large landscapes covered with dense forests of trees with good image quality.

To achieve this level of efficiency and quality, we introduce a hierarchical bidirectional texture (HBT), inspired by recent image-based rendering (IBR) representations. It encodes the appearance of a tree model using several levels of detail relying on the hierarchy described above, for many view and illumination directions. This allows for fast rendering adapted to the amount of detail required.

The same hierarchical organization of a tree is also exploited for the visibility computation occuring within the elements of a hierarchical level. Combining the occlusions by traversing the shadowing structure upwards provides efficient and appropriate soft shadows within the tree itself, and also between different trees. This allows for the real-time calculation of shadows while the light source is moving.

All these techniques have been implemented in hardware rendering using OpenGL 1.2, and results are presented and discussed.

This paper is organized as follows. In Section 2 we review several approaches related to the rendering of natural scenes. In Section 3 we describe our representation, the *Hierarchy of Bidirectional Textures* (HBT). We explain in Section 4 how to render an object using this representation, then we provide in Section 5 the global algorithm for the rendering of the scene. Once the use of the representation is clear, we detail in Section 6 how to built it from an existing object (*e.g.,* a polygonal tree model). We describe our implementation, show our results, and give performance measures in Section 7, before concluding.

2 Previous Work

In early fight simulators, trees and other objects (*e.g.,* buildings) were painted on the floor, represented by very simple polyhedrons (pyramids, cubes), or using the very first IBR representation, the sprite. Billboards, another early IBR representation, can be seen as an extension to sprites, since they always face the observer but are attached to a 3D frame. These are still used in current simulators [17] and off-line video productions on a short schedule. Games in the mid-80's to the mid-90's, *i.e.,* before the arrival of 3D graphics boards, extensively used sprites and billboards to produce interactive fake 3D environments, *e.g.,* for car races. With the generalization of hardware-accelerated 3D and textures, workstations and home PC started using *cross-trees*, made of 2 to 3 orthogonal faces textured with a transparent image, thus behaving like 3D objects. This has been generalized in applications up to a dozen textured faces or more, used to represent foliage.

Several early papers have described methodologies to model the shape of trees. We do not discuss these here, since we concentrate on efficient rendering. Numerous methods have been progressively introduced to precompute polygonal levels of detail or simplify meshes on the fly. These are also beyond the scope of this discussion, as they do not behave very well for trees, suffering numerous artifacts during a geometric simplification. Nonetheless, an hybrid solution dedicated to trees [32] achieved nice results, but is still not sufficient for our quality and real-time goals.

The most interesting solutions result from the use of representations other than meshes, proposing a way to efficiently achieve both quality and quantity. An early example was particle systems [23, 24], in which geometry is replaced by strokes, thus producing the first dense forest images. Although originally too slow, the approach is now used in real-time applications.

Since 1995, the creativity in the field of alternate representations has exploded, following different tracks:

- IBR reuses real or synthetic views of objects [10, 27, 29, 8, 1], and in particular lightfields [9, 6], storing the color for the various possible rays hitting an object. While it can capture and redisplay most views of a tree, including under different views, the shading and shadowing remain fixed, and lightfields are very memory intensive. They have been applied to surfaces [16, 34] to incorporate some shading effects, however memory requirements are still too large to represent good quality trees.
- In a dual manner, textures have been extended to reproduce the view angle dependency with directional texture functions or relief textures [4, 20], or even to simulate objects in volumes with layered impostors, volumetric textures, and LDI [25, 15, 28]. Between these last techniques and the early cross-trees, Max *et al.* have combined Z-buffers used as a representation in various ways [13, 11, 12], with different orientations, organized in a level of detail hierarchy, or more recently with multilayers. Some of these methods have been used explicitly for trees: volumetric textures [15, 19], and the work of Max *et al.*
- A totally different approach lies in point-based rendering, such as surfels [21]. To date, it is not adapted to sparse geometry like foliage, but we think it might be a promising technique.

Bidirectional reflection distribution functions (BRDF) [18] encode the proportion of reflected light given a pair of incident and reflected (view) directions. They are thus used in high quality rendering, and represent precisely the aspect lacking in usual IBR and textural encodings. BTFs, and to a certain extent surface lightfields, are a way to introduce this missing dimension.

Our representation shares various aspects with the BTFs [4, 3], the hierarchical IBR of Max *et al.* [12], billboards, and lightfields. Like Max *et al.* [12], we manage levels of detail using a hierarchy of alternate representations: an alternate object type encodes a set of smaller objects, which can be themselves standard geometry or alternate representation. Depending on the apparent screen size, we use either the large object or the collection of smaller ones, depending on the level. Our alternate representation is a BTF, instead of layered Z-buffer as for Max *et al.* [11, 12]. These BTFs are precomputed using multiple local pre-renderings as [4], but using the 6 degrees of freedom of [3, 2]: position on texture, view direction, and light direction. Indeed, we sample all these directions because we want to encode 3D objects, while [3, 2] sample only 3 of the 4 degrees of freedom, since their purpose is the encoding of surface materials, not 3D objects; in case of anisotropy, they simply add a second sample set. As for lightfield and BRDF representations, we have to manage the sampled representation of a 4D direction space. However, our structure is much more compact despite its 6D nature, as we use it only for distant objects that appear small on the screen. Using billboards, we share the representation of 3D objects with textures which adapt to the point of view.

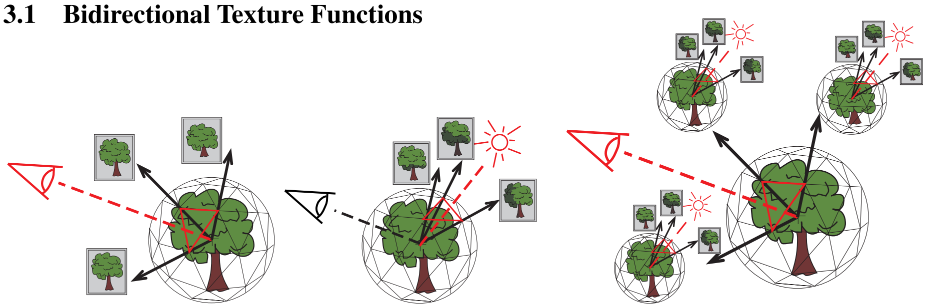
The auxiliary representation we introduce to accelerate the shadows was inspired by horizon maps [14, 31] and visibility precomputed for cells [5, 26], in a more approximate fashion. On the other hand it is 3D since all directions are valid, and also hierarchical, since it is connected to the HBT representation. We detail this representation in the next section.

3 Our Representation

There are 3 elements in our representation:

- The **BTF encoding** of an object, which is an alternate representation that produces similar images of this object even when varying the observer and light locations. This is explained in Section 3.1 and illustrated by Fig. 1.
- The **HBT structure** associated to a complex object, treated in Section 3.2, which extends the existing hierarchical scene graph of this object with level of detail information (see Fig. 5): a BTF is associated to each level. Reciprocally, most of the objects contained in a given level have an associated BTF. Since this work focuses on trees, these objects are basically organized as multiple instances of one similar base object (*e.g.,* a branch), plus possibly one mesh object (*e.g.,* a piece of trunk). This greatly reduces the total number of different BTFs, we use 3 per each kind of tree in our implementation.
- The structure for shadows, *i.e.,* the **hierarchy of visibility cube-maps**, that is described in Section 3.3 and illustrated by Fig. 2.2.

3.1 Bidirectional Texture Functions



1.1: A billboard is reconstructed from a given view direction by combining the 3 closest images stored in the sampled sphere of view directions.

1.2: A billboard is reconstructed from a given light direction by combining the 3 closest images stored in the sampled sphere of light directions.

1.3: The complete BTF allows the reconstruction of a billboard for given view and light directions by combining up to 9 stored images (in our implementation).

Fig. 1. The Bidirectional Texture Function representing a tree.

The principle of the representation is to associate a billboard representing an object with each pair of view and light directions (see Fig. 1 and Fig. 6), much like a BRDF associates a color with a pair of such directions. If we limit ourselves to objects that are not pathological shading-wise, only a small set of directions will be necessary. Moreover, these billboards are targeted to replace the detailed 3D representation of objects appearing small on the screen, and therefore their texture resolution can be very coarse (*e.g.,* 32×32 or 64×64 in our implementation). At rendering time, we reconstruct an image for given view and light directions by interpolating between the closest precomputed billboards. We detail the rendering in Section 4 and the BTF construction in Section 6.

It is better to avoid mixing in the same billboards the effects of sun illumination and ambient (sky) light, otherwise it would not be possible to separately tune their colors and intensities, or to consider several light sources. As an example, we would like to deal with the weather becoming overcast (showing only ambient term), the sun turning red while the sky remains light blue, or extra light sources such as helicopter headlights. Thus we store separately "ambient" billboards associated with each view direction. Note that this "ambient light" differs from classical Phong ambient since it takes occlusions into account.

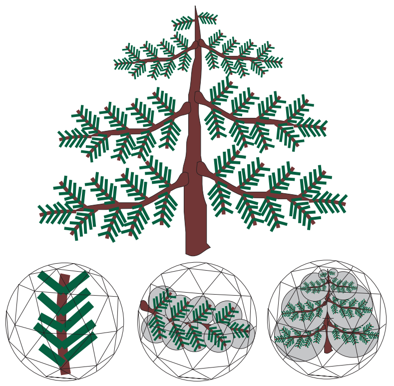
In practice we use either 6, 18, 66, or 258 samples for the view directions, evenly distributed on a sphere. For each sample direction, we associate a similar sampled sphere for light directions.[1] A small color texture with transparencies is stored for each pair of directions. To efficiently determine the closest 3 samples at rendering time, a precomputed table gives the 3 closest sampled directions for any given vector.

The instances of the obtained BTFs are associated with a local frame of reference and a scale factor, much like standard objects.

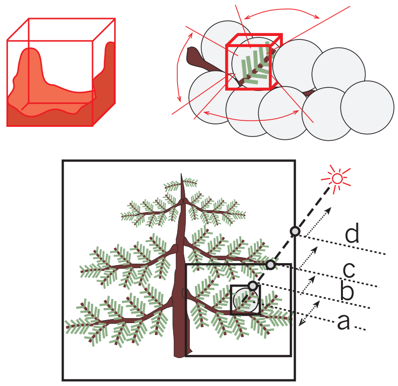
3.2 Hierarchy of BTFs (or HBTs)

We rely on the existing scene graph hierarchy, knowing that for trees this hierarchy typically contains several levels and massive instancing.

A BTF is associated to each level in the hierarchy (Fig. 2.1 and Fig. 5), representing its coarse level of detail. A level consists of a set of objects. Each of them can be unique or an instance of a given base object, located and oriented using its frame of reference. Most base objects should have an associated BTF, except if their shape is very simple. At rendering time, either the level's BTF or the set of objects will be used, depending on the distance.
Please note that in this paper, we call "highest" the coarsest level, representing the entire object.



2.1: Hierarchy of BTFs. *Top:* a typical tree model, showing a natural hierarchy (leaves or needles, small branches, main branches, trunk). *Bottom:* 3 BTFs associated to 3 levels of the hierarchy. For instance, the main branch (middle) is composed of instances of small branches (left).



2.2: Hierarchy of visibility cube-maps. *Top:* a visibility cube-map is associated to each instance, here a small branch on a main branch. It registers the occlusion (represented by grey zones on the cube-map) generated by its environment in every direction. *Bottom:* to obtain the total visibility in a given direction, we combine the corresponding values in the cube-maps of the higher levels, *e.g.,* b, c, d for the small branch. Self-shadows (a) are already encoded in the branch BTF.

[1] The number of samples can be different on the 2 spheres. In practice, the quality of interpolation is more sensitive to the view directions than to the light directions.

3.3 Hierarchy of Visibility Cube-maps for Shadows

Since we are using hardware rendering we cannot rely on it to compute shadows, and because the scene is complex we cannot evaluate it on the fly using shadow rays. [2] Thus we use precomputation information, but we still want the light direction to change in real-time. As we cannot afford a per-pixel shadow evaluation, we will evaluate the visibility at the billboard corners, and let the hardware interpolate.

Note that self-shadows are already represented in the BTF associated with the object (*i.e.,* the current level of detail), thus we only have to account for shadowing caused by occluders outside the object (*i.e.,* at the upper levels). We store the visibility around the current object (Fig. 2.2(top)), using cube-maps. The idea is analog to the horizon maps [14, 31] (except that those are 1D while we need 2D), or to visibility precomputed per cell [5, 26]. A cube-map represents the visibility at a given location. For each light direction, we precompute the total amount of light reaching this location, which gives the amount of occlusion in this direction, stored in the corresponding pixel of the visibility cube-map.[3] For an object, we sample the visibility at several locations, and the visibility at any location is estimated by interpolating the cube-maps.[4] In our implementation, we used the 8 corners of the bounding volume. In the following, the "visibility cube-map" associated to an object refers to this set of cube-maps, and its value for a given direction refers to the interpolation of the corresponding value in the cube-maps within the set. The pixel values correspond to opacities in the range [0,1], and are used at rendering time to modulate the color of the billboard in order to get the shadows. Note that the dynamic nature of this occlusion value allows for transparent objects, antialiasing, and soft shadows.

Computing and storing a different cube-map for every instance of objects in our scene would be excessively costly, given the total number of small branches in the forest. Instead, we rely on a hierarchical structure following the HBT hierarchy, and separate the occlusions occuring inside one level from the occlusions occuring outside: for a given level in the hierarchy, a visibility cube-map is associated to each instance of the objects in the set. It only represents the occlusions caused by the objects within the set. At rendering time, the illumination received by an object from a given light direction is modulated by combining the values of the cube-maps at the current and higher levels of the hierarchy, as shown on Fig. 2.2(bottom).

The status of shadows is thus as follows:

- self-shadowing is encoded in the BTF (represented by a in Fig. 2.2);
- a visibility cube-map encodes the blocking of light between its associated object and the cube-map associated to the entire hierarchy level (corresponding to b and c in Fig. 2.2);
- the cube-map at the highest level of the hierarchy encodes the blocking of light coming from outside, within the entire scene (stored in d in Fig. 2.2).

Therefore a different cube-map has to be stored for every instance of the highest level objects in the scene. In contrast, for an intermediate hierarchy level, since the cube-maps of objects handle only the visibility within this level (and not relatively to the entire scene), we only need one cube-map per instance of an object in the level. For instance, each main branch constituting a tree has its own cube-map, but as the trees

[2] A global shadowmap cannot be used either because the objects are partly transparent.

[3] The visibility cube-map is thus the dual of a shadowmap, which encodes the visibility of every location in the scene for one single light direction.

[4] We use software interpolation when the value is needed at a given vertex, and hardware interpolation when rastering a billboard.

in the landscape are instances of one single tree, we do not need to consider a new visibility cube-map for each branch in the scene. In our implementation, the maps are $32 \times 32 \times 6$, and the highest level corresponds to a tree, so the total required memory is acceptable (see Section 7). If necessary, we could add an extra level consisting of a group of trees.

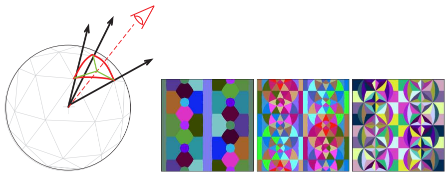
We also precompute the average value of each cube-map, that gives the ambient visibility (*i.e.,* the sky occlusion); this will be used to evaluate the ambient illumination.

4 Rendering an HBT

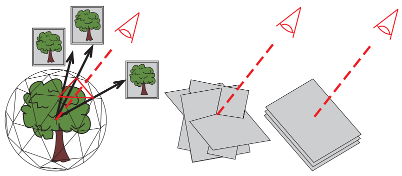
To render an HBT, first we need to determine the appropriate level of detail according to its apparent size in the image. This is done by using the level whose billboard pixels projection is close to the screen pixels size. Then all the objects in the 'opened' levels of the hierarchy, either polygonal meshes or BTF, are rendered from back to front. To render a BTF, we reconstruct a billboard for the current observer and light directions, and then compute its illumination taking into account light color, sky color, and shadows.

4.1 Reconstructing an Image from a BTF

Formally, the reconstruction problem is mainly the same as for a BRDF: we have to sum weighted values of the closest samples. Three classical issues are then: how to choose the best samples and weights, how many samples, and how to find them efficiently.



3.1: *Left:* Given the observer direction, the 3 closest sampled view directions must be found, and their 'distance' to the current direction calculated. *Right:* The 3 tables giving the precalculated 3 closest sampled directions (the colors encode the sample ids).



3.2: The two projection philosophies for the 3 images corresponding to the closest view directions: cross-tree like (middle), and billboard-like (right).

Our implementation is straightforward regarding these issues. As we separate the 4D table into a sampled sphere for view directions and another one for light directions, we have to find in which cell bounded by 3 samples a given direction falls (Fig. 3.1(left)). To be more efficient, we construct a precomputed table that directly provides the 3 sample directions for any given direction (Fig. 3.1(right)). The blending coefficients correspond to the barycentric coordinates within the cell on the sphere. Since the 3 billboards corresponding to the 3 samples from the view direction result themselves from the blend of the 3 samples from the light direction, we get 9 billboards to blend. The blending coefficients are the products of the barycentric coordinates relative to the view cell and to the light cell. However 9 can be considered a bit high.[5] To reduce this number, we do not consider samples associated to coefficients below a given threshold and we renormalize the other coefficients. It might be possible to reduce this number even further using some similarity criteria, considering only the samples that add enough information.

[5]Especially when noticing that directly considering a 4D-sphere for directions would have yielded 5-vertex cells, *i.e.,* the shape of a 4D-simplex.

As we deal with images instead of scalars in BRDFs, there are 2 extra issues: how to project each image onto the screen, and how to blend them together, knowing that we want to use the hardware Z-buffer that is not totally compliant with transparent images (see Appendix).

Two main choices exist to address the projection onto the screen, depending whether the billboard or the cross-textured philosophy is preferred for trees (Fig. 3.2):

- considering each image as a projected slice, that should then keep its original orientation;
- considering each image as a billboard, that are consequently mapped at the same location, *i.e.,* facing the eye.

The first solution provides more parallax effects especially if very few directions are available. The drawback is that the bounding volume depends on the angle of view, and that the opacity is poorly distributed, *i.e.,* we get a higher density where the images superimpose. The second solution is well adapted for objects having cylindrical or spherical symmetry: as the image is always facing the observer, the bounding volume is more stable and the opacity better distributed. Note that with more sampled directions, the two solutions are closer, so it is better to use the simplest one, *i.e.,* billboards. For a low number of samples (*e.g.,* 6 directions), it is probably better to choose the strategy depending on the look of objects: in such case, we used the standard cross-images.

4.2 Shadowing an Object

An object has to be modulated by the received illumination before it is drawn. The amount of light is simply obtained by combining the opacity values for the current light source direction in the visibility cube-maps (VCM) associated to the current object and all the levels above (Fig. 2.2). Combining two opacity values is done with the following scheme $Opacity_{combined} = 1 - (1 - Opacity_{levelB}) \times (1 - Opacity_{levelA})$. This value is multiplied by the light source intensity and color, then used as the polygon color when drawing the billboards as described above. We can also consider a cloud map to modulate the sun intensity:

$$illumination_L = C_L \prod_{level\ k} 1 - \mathtt{VCM}_k(L)$$

To handle multiple light sources, we iterate the process above, yielding a maximum of 9 new passes per light source with our implementation. We could potentially exploit the fact that any additional light source in a landscape should have a more limited visual impact on the entire forest (*e.g.,* helicopter headlights) and therefore treat it only for the closer trees.

The last component is the ambient term: to consider diffuse illumination not directly due to the sun, namely, sky illumination, we have to sum another image corresponding to the ambient contribution. Since it does not depend on light direction, we only have to combine 3 billboards, associated to the view direction sampled sphere. The "ambient shadowing" is obtained by multiplying the average visibility associated to the cube-maps and the average cloud transparency.

$$illumination_{amb} = C_{sky} \prod_{level\ k} 1 - average\_\mathtt{VCM}_k$$

So $C = \sum_{light\ s} \mathtt{HBT}(V, L_s)\ illumination_L + \mathtt{HBT}_{amb}(V)\ illumination_{amb}$ where V is the view direction and L_s the direction of the light source s.

As stated in Section 3.3, all these computations are done in software using the various precomputed tables, and evaluated at the 4 billboard corners (or possibly only once at the billboard center, for small details). The hardware interpolates these values when rasterising the billboard.

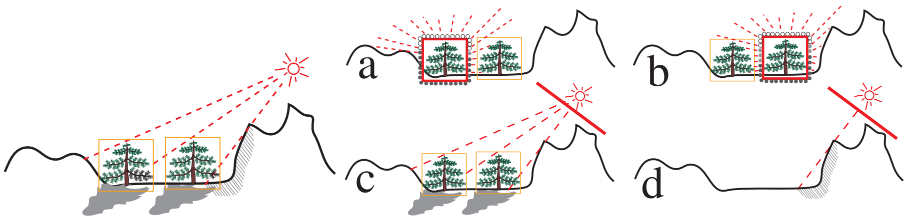


Fig. 4. The 4 kinds of shadow interaction between trees and terrain. *Left*: desired effects. *a,b*: The visibility cube-maps of each tree take into account the presence of both the mountain (and ground) and the other trees (self-shadows are included in the BTFs and in the cube-maps down in the hierarchy, see Fig. 2.2). *c*: The alpha-shadowmap accounts for the soft shadows of the trees on the ground. *d*: The Z-shadowmap (or *depth map*) accounts for the self-shadows of the mountain (any of the numerous shadowing algorithms for terrain could be used instead).

5 Rendering the Landscape

The global scene rendering algorithm is simple: first we render the terrain (and possible extra opaque objects), then we consider all the objects at their highest level (*i.e.,* trees) from back to front, culling those that are clearly not visible (outside the view frustum, and if possible behind the mountains, using a conservative occlusion technique [35, 5, 26], or extending techniques dedicated to terrains [14, 30, 31]). The rendering of HBT objects is then processed as described in Section 4.

The last issue is the shading and shadowing of the terrain. We could use the same principle described in Section 4.2, applied at the vertices of the terrain mesh. But the resolution of the shadows would probably be too coarse, and a visibility cube-map would have to be stored at each vertex which is probably too much. We prefer to simply use shadowmaps [33, 7], recomputed when the light direction changes.[6] In fact, two different kinds of shadowmaps have to be combined: one dealing with transparent objects (the trees), and the other with opaque objects (the terrain), as illustrated in Fig. 4(bottom). These shadowmaps are built by rendering the scene from the sun location.[7] For the first shadowmap, only the trees transparency is drawn (*i.e.,* we considere only the alpha value of textures), as a grey level. The terrain is drawn invisible (*i.e.,* white): it should not produce a shadow, but it should hide the trees behind the mountains. For the second shadowmap, which takes into account the terrain self-shadowing (*e.g.,* the shadow of a mountain on the valley), a depth map is used. Note that any terrain shadowing technique such as horizon map [14, 30] could be used instead, which is provided in hardware on the SGI *Infinite Reality*.

6 Building the Data

Our scenes are mainly composed of a terrain and many instances of one or more tree models. We need to encode an existing tree model into our representation, as transparently as possible for the user. Our only assumption is that instances are massively used to design a tree: *e.g.,* a tree is composed of numerous instances of a few branch models, which are composed of numerous instances of small branches and leaves (or needles).

[6] Here a shadowmap can be used because the terrain is opaque (while trees are not).

[7] Unfortunately building these 2 shadowmaps takes a significant amount of time using current SGI hardware. When the light direction moves, it is thus better not to update these at each frame.

We rely on this existing hierarchy to organize our levels of detail.

We have two hierarchical structures to built: the BTFs encoding the appearance of each subset of a tree (including the tree itself), and the visibility cube-maps encoding how the environment modulates the received illumination of each element.

The entire scheme and structure are illustrated by Fig. 5.

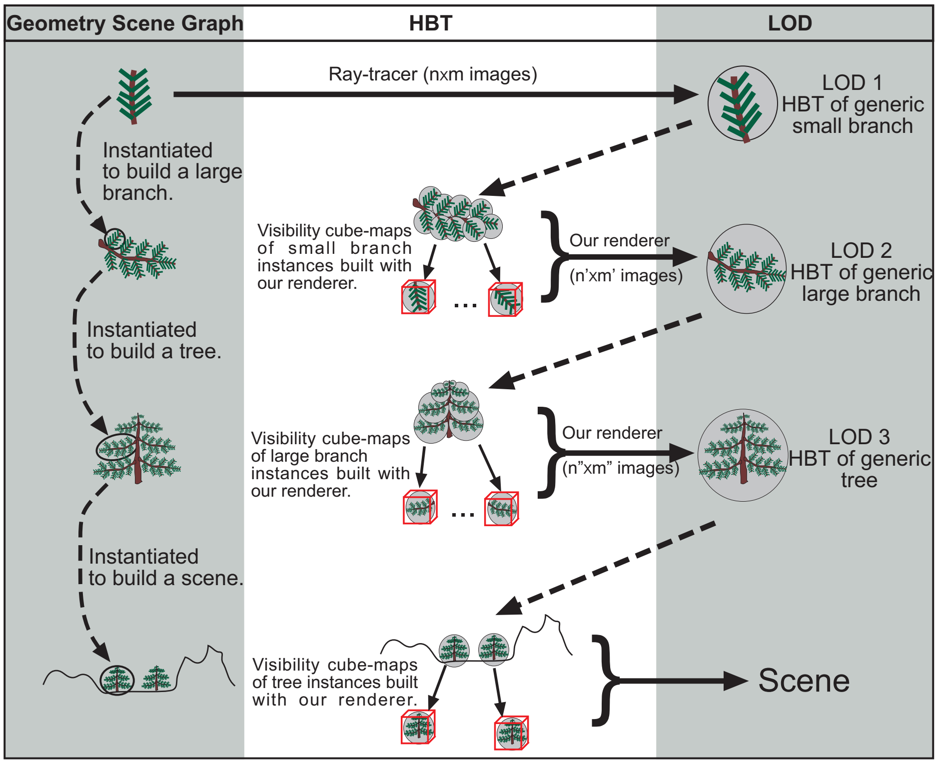


Fig. 5. Building the data.

6.1 Building an HBT

A BTF is a collection of images of a shaded object from various view directions and light directions (Fig. 6). We proceed in the same spirit as [4], adding the loop for light directions. We have two kinds of objects:

- The lowest level of the hierarchy is composed of standard objects. We use a ray-tracer to produce the views of mesh models even though final rendering is with a Z-buffer algorithm. Ray tracing allows us to easily get self-shadowing, antialiasing, transparencies, and to use complex shaders (*e.g.,* the shading of needles can be analytically integrated [**?**], while the cylinders should be discretized if a Z-buffer were used).
- The higher levels of the hierarchy are mainly composed of BTFs, plus possibly a few simple polygonal meshes (*e.g.,* for the trunk). We use the hardware Z-buffer to produce the views of these higher levels, in order to guarantee a consistent shading when either the lower or the higher level of detail is used during the

interactive rendering. Indeed, we use the same rendering algorithm as for interactive rendering, which provides us with shadows cast between the elements using the visibility cube-map structure.

In addition to the light-dependent BTFs, we also have to build the small set of light-independent BTFs corresponding to the ambient term, which are used for the sky illumination. They are currently produced by rendering the object with the ambient term only, taking into account ambient visibility. In our implementation, the views are 32×32 or 64×64 RGBA images.

6.2 Building the Visibility Hierarchy

A visibility cube-map associated to an object encodes the light occlusion in every direction due to the environment, but limited to the elements in the hierarchical level it belongs to or the entire scene for the highest level.

Thus for each pixel of the cube-map, the energy transfer between the light source (the sun) in the corresponding direction and the bounding volume of the object (taking occlusions into account) should be evaluated, and the amount of occlusion should be stored. As a cube-map has to be computed for each instance of a tree model in the scene taking into account a complex environment, this precomputation can be costly.

We implemented an approximative, yet very efficient simplification, using our OpenGL-based rendering algorithm. We place the camera at the center of the cube, and render the appropriate environment (in fact, only its transparency, as a grey level) in the 6 directions. This corresponds to the illumination received at the center, instead of the average value received by the object.

Once the cube-map is built, we need to compute its average value, that is used to evaluate the ambient (*i.e.,* sky) illumination.[8] Note that the cube-map is a generalization of the horizon map [14], while the average value is equivalent to an accessibility (or exposure) map.

7 Results

In our implementation, billboard images are 32×32 or 64×64 RGBA textures. Observer and light directions spheres are discretized with 6 or 18 evenly distributed samples. Thus regular BTFs are $32 \times 32 \times 4 \times 6 \times 6$ (144 Kb), $32 \times 32 \times 4 \times 18 \times 6$ (432 Kb), or $32 \times 32 \times 4 \times 18 \times 18$ (1.3 Mb), and ambient BTFs are $32 \times 32 \times 4 \times 6$ (24 Kb) or $32 \times 32 \times 4 \times 18$ (72 Kb). Our tree models are produced by an L-system [22]; we used 2 kinds of pine-trees and a prunus tree for our tests. We consider 3 hierarchical levels: a small branch with leaves or needles, a main branch made of instances of small branches, and a tree made of main branches. The pine-tree has 30 main branches, 300 boughs, and 30,000 needles. The whole HBT thus consists of 3 BTFs (0.5 to 4 Mb), plus the same scene graph as the tree with standard geometry.

The scene contains 1000 trees, so there are $8 \times (1000 + 30 + 10)$ visibility cube-maps. The cubes are $32 \times 32 \times 6$ luminance textures (6 Kb), thus the visibility structure represents roughly $8 \times 6 \times 1000$ Kb = 48 Mb. If this is unacceptable, [9] one could easily introduce a fourth hierarchical level consisting of a cluster of a dozen trees, which would divide the memory expense by the same amount.

Precalculation time is about 75 minutes using an $Onyx^2$ *Infinite Reality*, 2/3 for visibility and 1/3 for the HBT.

[8] See effect on Fig.7(middle).

[9] However, note that this data structure lies in main memory, and is never loaded on the graphics board.

Our global rendering algorithm is straightforward: we cull only the trees completely outside the view frustum, we sort the trees, then the parts of the trees for 'opened' levels of detail. We consider sun and sky illuminations, *i.e.,* one light source plus ambient. Our optimization for billboard combination requires on average 5 images instead of 9. In the worst case we have to draw 15 billboards per object,[10] which requires a high fill rate. The SGI we use has no multitexturing ability. However recent graphics boards such as Nvidia or ATI do have this feature, which allows us to decrease the number of passes to 5 (or 3, if the billboards of low contribution are eliminated).

We produced animations of 640×480, using an $Onyx^2$ *Infinite Reality*, showing a fly over a scenery covered by 1000 trees. The frame rate is 7 to 20 fps. We observed a 20% gain using an Nvidia board on a PC, without using multitexturing. Using this feature, we should gain another factor of 2 to 3.

A pine tree represented by a polygonal mesh is about 120k polygons. Using the lowest level of detail (*i.e.,* maximum resolution) yields a gain of 8 in the rendering time. Switching to the middle level gives an extra gain of 18. With the highest (*i.e.,* coarsest) level, we have another gain of 30.

8 Conclusion

We have introduced a new IBR representation for trees, which allows for very efficient quality rendering with complex effects such as shading, shadows, sky illumination, sun occlusion by clouds, and motion of the sun. Our naive implementation runs at 7 to 20 fps on an $Onyx^2$ *Infinite Reality* on a scenery containing 1000 trees.

The HBT representation consists of a hierarchy of bidirectional textures, which provides a billboard for each given observer and light directions, plus a visibility hierarchical structure for self- and cast-shadows. Despite the 6 degrees of freedom of the BTF, the required memory of a few tens of Mb is acceptable. Our representation is efficient in memory as long as hierarchy and instancing exist in the data, which is often the case for trees. It should thus also apply to other kind of scenes (*e.g.,* city, crowd, ...).

Specularities, BRDF, and transmission are also handled by our method, but precision is limited by the sampling density of view and light directions. The sampling of directions and of cube-map locations potentially yield artifacts, as linear interpolation does not perfectly reconstruct the data. This issue disappears with higher sampling rates, at the cost of memory. Thus there is a trade-off between visual quality *vs.* memory used. However, results show that nice real-time fly-overs can be produced with reasonable memory, even when using several kinds of trees.

A far more efficient global rendering algorithm could be implemented, using grids and smart culling methods, thus allowing the interactive visualization of much larger landscapes. Various different implementation choices could be made on the HBT rendering and construction. For instance, a more accurate visibility evaluation could be precomputed.

It would be interesting to see how the future graphics boards will include features easing the rendering process: SGI *Infinite Reality* implements 4D textures; Nvidia chips should do the same soon. Once 4D textures will be treated like any texture interpolation, filtering and compression wise, lightfields will be directly available in hardware. If 6D textures are also managed some day, BTFs will be directly available as well...

[10] 3 for darkening the background, 9 for direct illumination, 3 for ambient (*i.e.,* sky) illumination.

Appendix: Images weighted summation with hardware

We want to combine images by weighted summation, which is different than blending them together by alpha compositing which would depend on the order of operation. On the other hand, we want to blend the result with the background image already on screen by compositing.

$C = \sum_{i,j} \alpha_i \beta_j \, \mathtt{BTF}_a(i,j) \, \mathtt{BTF}_{rgb}(i,j)$,
$A = \sum_{i,j} \alpha_i \beta_j \, \mathtt{BTF}_a(i,j) \;=\; \sum_i \alpha_i \mathtt{BTF}_a(i)$,
$C_{pixel} = C + (1 - A) C_{pixel}$,
where α_i and β_j are the barycentric coordinates of the view and light directions.

Another problem is that we do not want the depth test of the Z-buffer to reject fragments that might appear behind a transparent fragment, especially when drawing intersecting billboards. A costly solution would be to process the reconstruction in a separate buffer (without a Z-test), and to use the result as a texture to be blended with the image on screen. To simulate this process, while maintaining the pipe-line, we first darken the background according to the cumulated transparency of the billboards. As the same transparency occurs for the 3 images corresponding to one view direction sample, we only have 3 transparency maps to consider. "Darkening" is done using the `ADD` blending equation of the OpenGL API with `(0,1-SRC_ALPHA)` blending coefficients. The weight α_i is encoded in the alpha of the underlying polygon.

Then we draw the weighted textures (the polygon alpha is set to $\alpha_i \beta_j$), using `(SRC_ALPHA,1)` blending coefficients. Thus we have a total of 12 passes if all the billboards are to be drawn. To avoid Z-interactions, the Z values are stored only in the last pass.[11]

This yields the following pseudo-algorithm:

```
Draw scene from back to here
Disable Z-write
BlendCoeff(0,1 - SRC_ALPHA)
Darken screen according to object transparency (3 passes)
BlendCoeff(SRC_ALPHA,1)
Draw the object (9 passes) (enable Z-write before the last pass)
```

Acknowledgments : We wish to thank Patricio Inostroza and George Drettakis for proofreading this paper. Pierre Poulin is currently at REVES/Inria Sophia-Antipolis as part of his sabbatical.

[11] As the objects are sorted, storing the Z values of trees is not very useful anyway, as long as possible extra opaque objects are drawn first (the Z-test remains enabled).

References

1. S.E. Chen. Quicktime VR - an image-based approach to virtual environment navigation. In *SIGGRAPH 95 Conference Proceedings*, pages 29–38, August 1995.
2. K. Dana, B. van Ginneken, S. Nayar, and J. Koenderink. Columbia-utrecht reflectance and texture database, http://www.cs.columbia.edu/cave/curet/.index.html.
3. K.J. Dana, B. van Ginneken, S.K. Nayar, and J.J. Koenderink. Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics*, 18(1):1–34, January 1999.
4. J.-M. Dischler. Efficiently rendering macrogeometric surface structures using bi-directional texture functions. In *Eurographics Workshop on Rendering 98*, pages 169–180, 1998.
5. F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. In *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 239–248, 2000.
6. S.J. Gortler, R. Grzeszczuk, R. Szeliski, and M.F. Cohen. The lumigraph. In *SIGGRAPH 96 Conference Proceedings*, pages 43–54, August 1996.
7. A. LeBlanc, R. Turner, and D. Thalmann. Rendering hair using pixel blending and shadow buffers. *Journal of Visualization and Computer Animation 2(3)*, pages 92–97, 1991.
8. J. Lengyel and J. Snyder. Rendering with coherent layers. *Proceedings of SIGGRAPH 97*, pages 233–242, August 1997.
9. M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH 96 Conference Proceedings*, pages 31–42, August 1996.
10. P.W.C. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. *1995 Symposium on Interactive 3D Graphics*, pages 95–102, April 1995.
11. N. Max. Hierarchical rendering of trees from precomputed multi-layer Z-buffers. In *Eurographics Workshop on Rendering 1996*, pages 165–174, June 1996.
12. N. Max, O. Deussen, and B. Keating. Hierarchical image-based rendering using texture mapping hardware. In *Eurographics Workshop on Rendering 99*, pages 57–62, 1999.
13. N. Max and K. Ohsaki. Rendering trees from precomputed Z-buffer views. In *Eurographics Workshop on Rendering 1995*, June 1995.
14. N.L. Max. Horizon mapping: shadows for bump-mapped surfaces. *The Visual Computer*, 4(2):109–117, July 1988.
15. A. Meyer and F. Neyret. Interactive volumetric textures. In *Eurographics Workshop on Rendering 1998*, pages 157–168, July 1998.
16. G.S.P. Miller, S. Rubin, and D. Ponceleon. Lazy decompression of surface light fields for precomputed global illumination. *Eurographics Workshop on Rendering 1998*, pages 281–292, June 1998.
17. J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide*. Addison-Wesley, 1993.
18. F.E. Nicodemus, J.C. Richmond, J.J. Hsia, I.W. Ginsberg, and T. Limperis. Geometric considerations and nomenclature for reflectance. October 1977.
19. T. Noma. Bridging between surface rendering and volume rendering for multi-resolution display. In *Eurographics Workshop on Rendering 1995*, pages 31–40, June 1995.
20. M.M. Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. *Proceedings of SIGGRAPH 2000*, July 2000.
21. H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. *Proceedings of SIGGRAPH 2000*, pages 335–342, July 2000.
22. P. Prusinkiewicz, A. Lindenmayer, and J. Hanan. Developmental models of herbaceous plants for computer imagery purposes. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 141–150, August 1988.
23. W.T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108, April 1983.
24. W.T. Reeves and R. Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19(3), pages 313–322, July 1985.
25. G. Schaufler. Per-object image warping with layered impostors. In *Eurographics Workshop on Rendering 98*, pages 145–156, 1998.
26. G. Schaufler, J. Dorsey, X. Decoret, and F.X. Sillion. Conservative volumetric visibility with occluder fusion. In *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 229–238, 2000.
27. G. Schaufler and W. Stürzlinger. A three dimensional image cache for virtual reality. *Computer Graphics Forum*, 15(3):227–236, August 1996.
28. J. Shade, S.J. Gortler, L. He, and R. Szeliski. Layered depth images. *Proceedings of SIGGRAPH 98*, pages 231–242, July 1998.
29. J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. *Proceedings of SIGGRAPH 96*, pages 75–82, August 1996.
30. A.J. Stewart. Hierarchical visibility in terrains. In *Eurographics Workshop on Rendering 97*, June 1997.
31. A.J. Stewart. Fast horizon computation at all points of a terrain with visibility and shading applications. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):82–93, March 1998.
32. J. Weber and J. Penn. Creation and rendering of realistic trees. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 119–128, August 1995.
33. L. Williams. Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, volume 12(3), pages 270–274, August 1978.
34. D.N. Wood, D.I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D.H. Salesin, and W. Stuetzle. Surface light fields for 3D photography. In *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 287–296, 2000.
35. H. Zhang, D. Manocha, T. Hudson, and K.E. Hoff III. Visibility culling using hierarchical occlusion maps. In *SIGGRAPH 97 Conference Proceedings*, pages 77–88, August 1997.

Fig. 6. The BTF associated to the highest level (horizontal axis: 18 view directions, vertical axis: 6 light directions, 64×64 resolution per billboard, with colors and transparencies).

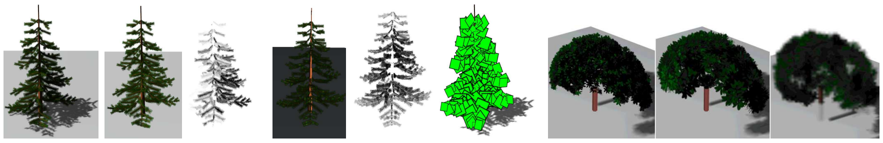


Fig. 7. *Left to right*: a pine tree with shadows, without shadows, amount of shadow. The pine with only ambient, the ambient visibility coefficient, the billboards used at the lowest level of detail. A prunus tree drawn at 3 levels of detail.

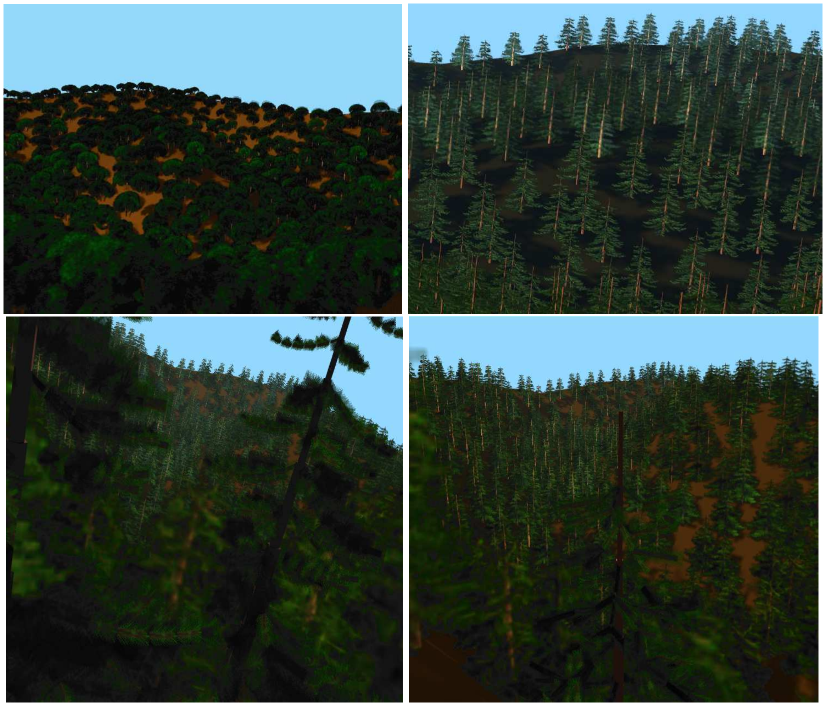


Fig. 8. 4 views in a forest (1000 trees on a landscape, with shading, shadows, and fog). The frame rate is 7 to 20 fps on our test machine. Note the detailed trees in the foreground.