Volumetric Textures

Esteban W. Gonzalez Clua[1]
Marcelo Dreux[2]

[1]PUC-Rio - Department of Computer Science
Rua Marquês de São Vicente, 255, 22443-900, Rio de Janeiro, RJ, Brazil
esteban @inf.puc-rio.br

[2]PUC-RIO – Mechanical Engineering Department
Rua Marquês de São Vicente, 255, 22443-900, Rio de Janeiro, RJ, Brazil
dreux @mec.puc-rio.br

Abstract

There are some types of nature elements that are adequately represented in Computer Graphics only through volumes. In order to visualize scenes with volumes, together with geometrical objects, it is necessary to make use of a hybrid rendering algorithm. However, the presence of volumetric elements heavily increases the visualization processing time, independently of the technique being used. This article presents a method that is being developed in order to reduce the volumetric rendering to a projection of a special texture on a geometric surface that surrounds the volume. This technique seems to be particularly efficient to volumes that represent nature elements, such as gases, clouds and smoke. It is possible, however, to extend the method to other types of volumetric objects.

Keywords: *Volume Rendering, Nature Elements, Textures.*

1. Introduction

Traditional Volume Rendering techniques perform a density/transparency evaluation, along each visited voxel, relative to each pixel that contains part of the volumetric object. Consider, for instance, a volumetric object in a scene with 1000x1000x1000 voxels. In this case, for each pixel of the image that contains part of the projected object, on average 1000 voxels will be visited. This number varies according to the angle and position of the ray that comes from the observer and reaches the volume. If Ray-tracing is being used, such as it is described by Sobierajski et al [9], the situation is even worse since, due to the recursions, the volume is traversed more than once.

A number of volume visualization optimizations have been proposed [3, 7, 8]. These methods, however, try to minimize the number of visited voxels along the ray path, inside the volume, by avoiding unnecessary voxels or by grouping voxels with low density contribution. Even with those optimizations, it is necessary, in many situations, to traverse a long voxel list.

Gardner[4] makes use of procedural textures in order to distribute density onto an object surface, when dealing with *nature elements*. However, in that method, the texture is only applied on the surface, and thus information from the interior of the volume is not stored. They are not real volumes such as the ones modeled by Ebert [1, 2].

Volumetric Textures, as they are called in this research, try to model a volume through a set of special textures applied to simple geometrical surfaces. Those textures are built pre-processing the volume, to avoid traversing its interior during the visualization. That approach enormously accelerates the rendering process.

It becomes a simple texture mapping with a value that corresponds to the density accumulation along the path traversed by the ray inside the volume.

2. Texture Evaluation

The volumetric texture pre-processing consists basically of storing the density accumulation of a ray that reaches the volume at a certain point, considering a variety of possible angles. Therefore, volumetric textures are matrices, where each element maps a part of the geometric surface that surrounds the volume. In this work, these elements are called input polygons. Each input polygon has an associated table that stores the results obtained by a ray intersecting the volume at that point with different input angles.

So, the resolution of the volumetric texture object depends on the input polygon quantity and on the resolution of the table of input angles (see figure 1).

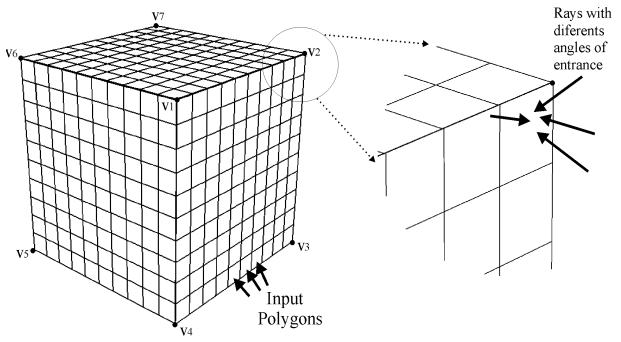

Figure 1 – The volume is represented by a geometric object (a box in this figure),which is divided in input polygons. Each of those polygons have a table with different input angles. It stores the density accumulation along the ray path relative to each input angle.

Consider the volume represented by a box defined by the vertices *v1, v2,..., v8* and by the faces *F1, F2, ..., F6*. In this case there will be 6 matrices, one for each face.

Each volumetric texture (VX_i) will also need to store the following vectors, in order to speed up the visualization process:

$$Up_i = \frac{v_k - v_j}{\| v_k - v_j \|}$$

And

$$\vec{S}_i = \vec{Up}_i \times \vec{N}_i \quad (1)$$

Where *i* is the face to be mapped by the volumetric texture VX_i, $\vec{Up}_i$ is face *i* up vector, $\vec{N}_i$ is face *i* normal and $\vec{S}_i$ is a vector perpendicular to both $\vec{Up}_i$ and $\vec{N}_i$.

As the face is square the three vectors form a local coordinate system to face *i* (see figure 2). These base vectors must be normalized in order to simplify future evaluations and the origin of this local coordinate system will be located at the center of the top left input polygon of the face *i*. It is also necessary to set up a base change matrix T_i, that transforms a point from this local coordinate system to the global coordinate system.

Each polygon is, initially, a square of size m. The location of the center of each input polygon is determined by (*j.m, k.m, 0*) and *J.m* and *K.m* are the horizontal and vertical face *i* sizes, respectively.

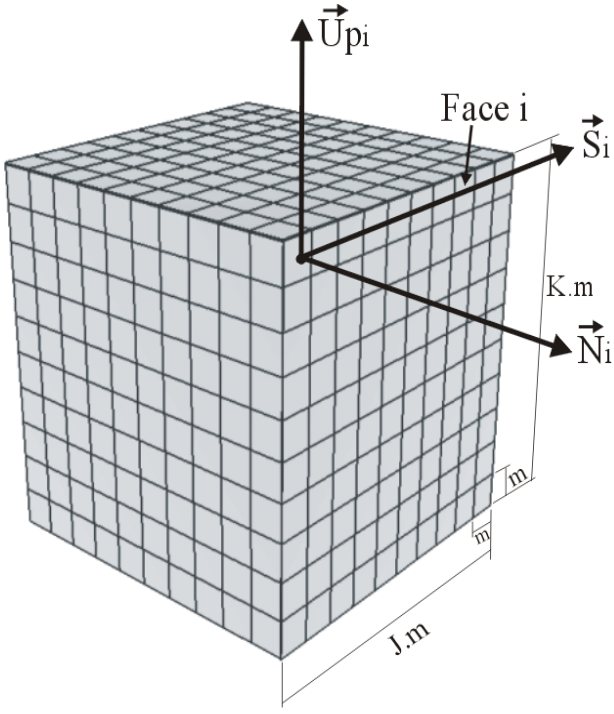

Figure 2 – Each face *i* has a local coordinate system, formed by the vectors $(\vec{S}_i, \vec{Up}_i, \vec{N}_i)$. Each face must have its own base transformation matrix T_i, which converts a local coordinate to the global coordinate system.

The pre-processing, then, consists in evaluating the density to a set of different input angles to each face polygon. That set of angles is a sampling of all possible

input angles. Each polygon table of angles is a matrix addressed by the pair of angles α e β formed between the input vector and the vectors $\vec{S}_i$ e $\vec{Up}_i$, respectively (figure 3). The input vector can be expressed, in the local coordinate system of face *i*, by:

$$\vec{R}' = (j.m, k.m, 0) - (j.m + tg^{-1}\alpha, k.m + tg^{-1}\beta, 1) \qquad (2)$$

However, as this vector will traverse the whole volume, which is described in the global coordinate system, it is necessary to apply the transformation matrix T_i to obtain the correct input vector (eq. 3).

$$\vec{R} = T_i.(j.m, k.m, 0) - T_i.(j.m + tg^{-1}\alpha, k.m + tg^{-1}\beta, 1) \qquad (3)$$

The following algorithm builds the volumetric texture for the face F_i, using regular increments to obtain the conjunct of input angles. The accumulated density for each ray passing by the volume is stored in $VX_{i,j,k}$ (VX_i is the volumetric texture of F_i and *j*, *k* are the input polygon coordinates of the point of the volume being evaluated).

For each input polygon of face F_i do

Δ_α = *180 / vertical angular resolution*
Δ_β = *180 / horizontal angular resolution*

For $\alpha = (-90 + \Delta_\alpha)$ *to* $(90 - \Delta_\alpha)$ *do*
For $\beta = -90$ *to* $(90 - \Delta_\beta)$ *do*

$$\vec{R} = T_i.(j.m, k.m, 0) - T_i.(j.m.tg^{-1}\alpha, k.m.tg^{-1}\beta, 1)$$

Evaluate the points t_{in} and t_{out}. They are the intersection points between vector $\vec{R}$ and the box.

$$VX_{i,j,k}[\alpha, \beta] = \int_{t_{in}}^{t_{out}} f\rho(t)\,dt$$

$\beta = \beta + \Delta_\beta$
$\alpha = \alpha + \Delta_\alpha$

Where *i* is one of the 6 faces, (*j*, *k*) is the coordinate of the current polygon, given in the local system of face *i* and *fp* is the function that describes the density inside the volume as a function of space.

When α or β are -90^o or 90^o, it is not necessary to evaluate or to store the accumulated density value since, for a vector parallel to the face, the accumulated density value is 0. The rays in that situation are tangent to the face and, therefore, do not penetrate the volume.

Note that the *fp* integral can be evaluated based on the density function description. Therefore, the volume to be visualized does not need to exist at any time. This fact can bring, in some cases, an enormous optimization. However, if only the volume is known, and not its density function, then the integral evaluation is analogous to the incremental density accumulation of the volume rendering algorithm [6].

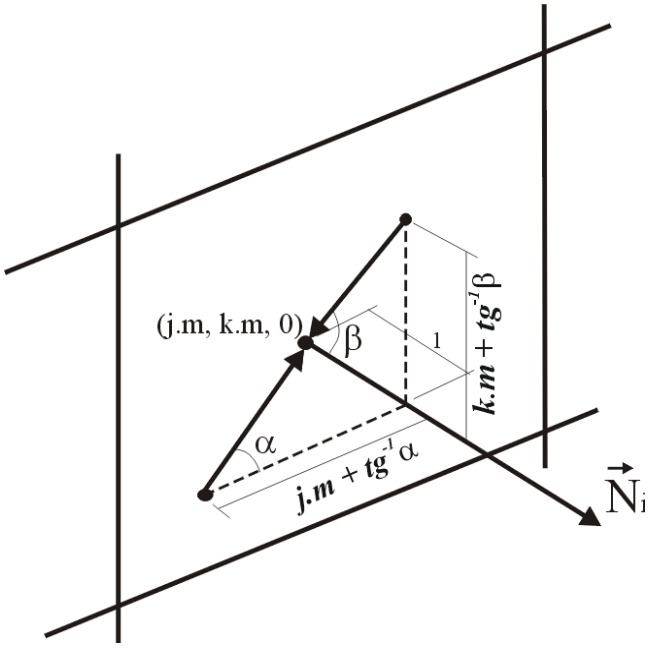

Figure 3 – In order to obtain the angles where the accumulated density is evaluated it is necessary to apply regular increments to α and β.

3. Visualization

The volume visualization consists in evaluating, for each input polygon, which are the α and β angles formed between the observer vector $\vec{O}$ (vector from the observer to the input polygon being visualized) and the local coordinate system of the face the polygon belongs to. So, α is the angle formed between the projection of $\vec{O}$ onto the plane formed by the vectors $\vec{N}_i$ and $\vec{S}_i$ ($\vec{P}_\alpha$) and β is the angle formed between the projection of $\vec{O}$ onto the plane formed by the vectors $\vec{N}_i$ and $\vec{Up}_i$ ($\vec{P}_\beta$) (figure 4).

$\vec{P}_\alpha$ and $\vec{P}_\beta$ can be evaluated by the following formulas:

$$\vec{P}_\alpha = (T^{-1}(\vec{O}) \cdot \vec{S}_i)\vec{S}_i + (T^{-1}(\vec{O}) \cdot \vec{N}_i)\vec{N}_i \qquad (4)$$

and

$$\vec{P}_\beta = (T^{-1}(\vec{O}) \cdot \vec{S}_i)\vec{S}_i + (T^{-1}(\vec{O}) \cdot \vec{Up}_i)\vec{Up}_i \qquad (5)$$

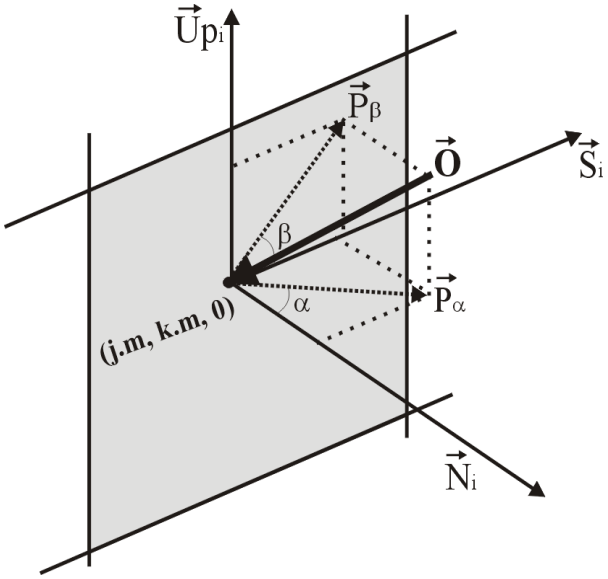

Figure 4 – Evaluation of angles α and β which are formed between vector $\vec{O}$ and the local coordinate system of the face the current polygon belongs to.

It is necessary to apply the inverse of the transformation matrix *T*, over vector $\vec{O}$, since it is described in terms of the global coordinate system and the other vectors are in terms of the face coordinate system. It is convenient to evaluate T^{-1} during the pre-processing phase and to store it with the volumetric texture.

Once the α and β angles have been evaluated, it should be only necessary to search the volumetric texture $VX_{i,j,k}[\alpha, \beta]$ in order to obtain the accumulated density value of polygon *j*, *k* from volume face *i*. However, α and β are real values while the table of angles has a discrete angle sampling. It is necessary to perform a linear interpolation of the density values to avoid an undesirable flickering effect during the visualization.

That interpolation slightly increases the computational processing time, however it generates images with greater quality, when compared to images with no interpolation. Volumes with few geometric details, such as gases, clouds and leaves, do not suffer a significant loss with the interpolation. When dealing with models with fine details, such as in medical images, the interpolation only works if the table of angles has a high resolution.

4. Optimization

The size of the volumetric textures is proportional to the table of angles resolution and to the size of the input polygons. For instance, if the table considers variation of 30^o to both α and β, there will be 25 density values to each input polygon. Considering a 1000 x 1000 x 1000 volume resolution, then the size of each matrix will be 25.10^6, in case each input polygon is the size of a voxel. As it is necessary to use 6 matrices to represent a complete volume, then the total size will be 150.10^6, while the volume, not considering possible compressions, has 10^9 voxels. In that case, the volumetric textures not only heavily accelerate the visualization process, but also require less storage space, when compared to volume rendering.

If a quadtree [5] is applied to the table of angles representation it is possible to obtain a more accurate and precise result. The matrix is, then, sub-divided each time the density variation from one angle to its neighbor is high. This method could also be used to perform table compression. The matrix could be built by joining parts of the table with similar values.

Another possible optimization is to make use of graphical acceleration (OpenGL, for example). For this case it is possible to plot an entire input polygon by applying the appropriate transparency, which was obtained by its volumetric texture.

5. Conclusion and Further Research

The proposed technique seems to be extremely efficient to visualize volumetric objects with a diffuse density distribution, as in the case of nature elements. Moreover, the technique allows the presence of volumes in scenes with surface rendering (ray-tracing, scan-line, etc.), since it is not necessary to perform the traditional volume rendering. The volume processing is treated as a kind of texture mapping and the real volume does not have to be present in the scene.

This work applies the textures in volumes being represented by 3D boxes. A possible extension could be the creation of other types of bounding volumes, such as spheres, cylinders or implicit surfaces, since for other

volumetric objects those bounding volumes would be more appropriate.

Other compression data structures could also be studied, since in certain situations the use of quadtrees could prove to be not so efficient.

This work only stores, for each input angle, the accumulated density for a ray in certain directions. However, to generate a more realistic image, it could be necessary to store extra data, such as the gradient or normal of the object inside the volume for each one of this input angles.

The volumetric textures still have limitations and it needs a thorough investigation to pursue possible solutions. The main drawback concerns the volumes with high details that could not be well represented, as mentioned before. Other limitation is that the observer cannot penetrate the volume, because the textures store accumulated density from the input point until the end of the volume. Furthermore, if there is an alteration of a voxel density the volumetric textures will have to be reconstructed, which is an expensive task.

Acknowledgement

The first author is grateful to CAPES, a brazilian government research council, and to TeCGraf, Technological Group in Computer Graphics of PUC-Rio, which sponsored this research.

References

1. David Ebert. Procedural Volumetric Modeling and Texturing, *SIGGRAPH 97*, course 14, notes, chapter 6, August 1997.

2. David Ebert. Procedural Volumetric Cloud Modeling and Animation. *SIGGRAPH 99*, *course notes*. August 1999.

3. John Danskin and Pat Hanrahan. Fast Algorithms for Volume Ray-tracing. *Proceedings of the 1992 Workshop on Volume Visualization*, *The Association for Computing Machinery, Boston, MA*, pp. 91-98. Special issue of Computer Graphics, ACM SIGGRAPH, New York.

4. Geoffrey Y. Gardner. Simulation of natural scenes using textured quadric surfaces. In Hank Christiansen, editor, *Computer Graphics* (*SIGGRAPH'84 Proceedings)*, volume 18, pp 11-20, July 1984.

5. C. L. Jacklins and S. L. Tanimoto. Octrees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing*, 14, pp. 249-270, 1980.

6. Marc Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Aplications* 8 (5) pp. 29-37. May 1988.

7. Marc Levoy. Volume Rendering by Adaptive Refinement. *The Visual Computer 6 (1), pp. 2-7*. 1990.

8. Renben Shu and Alain Liu. A Fast Ray Casting Algorithm Using Adaptive Isotrilinear Subdivision. *Proceedings Visualization*, *IEEE Computer Society Press*, pp. 232-238, 1991.

9. L. M. Sobierajski and A. E. Kaufman. Volumetric Ray Tracing. Proceedings of the Symposium on Volume Visualization, pp.11-18, 1994.