A 3D TEXTURE-BASED OCTREE VOLUME VISUALIZATION ALGORITHM

Imma Boada[⋆]

Isabel Navazo[◇]

Roberto Scopigno[‡]

[⋆] Institut d'Informàtica i Aplicacions,Universitat de Girona, Spain
[◇] Departament LSI,Universitat Politècnica de Catalunya, Spain
[‡] CNUCE - National Research Council (C.N.R.),Italy
imma@ima.udg.es,
isabel@lsi.upc.es
r.scopigno@cnuce.cnr.it

ABSTRACT

We present a 3D texture based Octree Volume Visualization algorithm that combines 3D texture hardware and hierarchical representation of data sets to obtain multiresolution renderings of very large data sets. The algortihm exploits advantages of both octree representation and 3D texture hardware. The basis of the algorithm is to take advantage of nearly iso-valued areas of the volume and regions of no interest to compute a more synthetical volume texture representation. A new volume-texture assignation policy allows to exploit 3D texture volume visualization technique on large data sets. The algorithm guarantees hight quality image for regions of maximal interest.

Keywords: 3D Textures,Volume Visualization, Octrees

1 INTRODUCTION

During the last years a lot of efforts have been undertaken to obtain real-time exploration of volume data sets. Although the adoption of hardware-enhanced approaches, such as 3D texture mapping, guarantees to obtain image quality almost interactively, it seems that is difficult to exploit interactive rendering on very large data sets.

3D texture mapping hardware has became the fastest volume rendering method available on high-end workstations. Based on the volumetric data, loaded into the texture memory, the data is rendered by compositing back to front 3D textured data polygons which slice and sample the volumetric texture space. The approach is simple and yield high quality results, but is limited by the amount of texture memory. The required texture memory is proportional to the volume resolution and greater is the volume more difficult is to represent it. Some techniques have been proposed to deal this situation. [Wilso94][Grzes98], for example propose to partitionate the volume into several appropiately sized bricks. Each brick is treated independently, and the entire volume rendering is obtained by the composition of all the bricks contributions; [Srini97],[Tong99] propose to use a pre-processing in which empty regions of the volume are identified to avoid its representation in the texture space.

However, despite all the improvements to exploit the technique on large data sets, none of the related techniques can always guarantee the enterely volume representation in the texture memory. The assignation policy of one texel per voxel, applied in all these methods, forces the volume data to be smaller or of the same size of the available texture memory.

To overcome this problem (i.e. to guarantee the enterely volume representation in texture space), we propose to modify the common assignation, texel-voxel, for a more synthethical one (i.e. less than one texel per voxel), that benefits of nearly homogeneous regions and areas of no interest of the volume. Based on this new assignation rule, we present a 3D texture based Octree Volume Visualization algorithm that combines 3D texture hardware and hierarchical representation of volume data to obtain multiresolution renderings of very large data sets. The algorithm exploits advantages of both octree representation and 3D texture hardware.

The paper is structured as follows. In Section 2 a brief description of background and related work in 3D texture volume rendering is given. After a description of the octree volume representation in Section 3, we introduce in Section 4 the proposed 3D texture based volume visualization algorithm. In Section 5 some results are given. Finally in section 6 conclusions and future work are described.

2 BACKGROUND

The 3D texture based approach is a direct data visualization technique that takes advantage of spatial coherence, being much faster than ray casting and obtaining *almost* the same quality images. Unlike ray casting, where each image pixel is built up ray by ray, the 3D texture based approach processes a set of rays simultanouesly, using the 3D texture as a voxel cache to store intermediate results.

This volumetric rendering technique can be decomposed in two steps [Grzes98]:

1. Texture Definition. The volume geometry is sampled to determine the set of voxels that will be cast to generate the texture. The texture is stored in texture memory and represents a $RGB\alpha$-encoded view of the 3D volume.
2. Volume Rendering. The rendering of the volume is obtained through the composition of texture-mapped polygons into the frame buffer. This set of polygons slices and samples the volumetric texture space. Before the composition a depth sorting step determines the order in which they will be composited.

The 3D texture approach is simple and yield high quality results when the volume data set can be enterely represented in texture memory. But, in situations where the volume is larger than the texture memory the volume data has to be broken up into smaller chunks called bricks [Grzes98]. Each brick is processed independently. The entire volume visualization requires binding , loading and swapping of all different textures, affecting interactivity and image quality. This is the main drawback of the technique.

The use of 3D texture-mapping hardware for direct volume rendering was first mentioned by Akeley for the SGI Reality Engine [Akele93]. Cullip and Neumann sketch two approaches and apply them to CT data obtaining the first pictures based on this technique [Culli93]. Guan and Lipes discuss hardware issues [Guan94]. Cabral, Cam and Foran describe how to use texture mapping hardware to accelerate numerical Radom transforms [Cabra94]. Wilson et al in [Wilso94] present an easy to implement method for direct volume rendering that uses 3D texture maps. The paper describes how most of programming of previous methods can be eliminated by the use of graphics library procedures to perform texture space transformations. In [Gelde96] a shading model is incorporated to this method.

[Srini97], [Tong99] presents some improvements to deal large data sets. In [Srini97] an octree is proposed to skip empty regions of the volume. In [Tong99] a preprocessing step is presented to select the volume data that contains object voxels to ensure that only these regions will be loaded in texture memory. In these methods the final texture representation of the volume assign one texel per voxel, thus for large data sets texture memory limitation is still a problem.

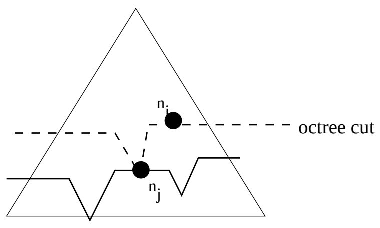

Figure 1: The triangle represents the pyramid that indicates a complete resolution set. The solid line represents leaves of the octree. The octree is a subset of the pyramid that completely spans the volume.

3 THE OCTREE REPRESENTATION

An octree is a multi-resolution data representation structure used to reorganize volume data for easy identification of *interesting* regions. Octrees have been used previously for volume rendering, e.g. [Levoy90] [Laur91][Wilhe92], often with the objective to avoid the evaluation of non-interesting regions and to optimize rendering speed.

The octree construction always requires a criteria to determine when a region of the volume is important or not. This criteria goes from the simplest that evalutes the presence or absence of data, to the more sofisticated in which the variation of internal samples values with respect to the samples of the vertices is evaluated to detect homogeneous zones. Once this criteria has been defined, leaf nodes are identified as those with maximal degree of satisfaction of the evaluated criteria. We call this degree of satisfaction the *error* of the node. In this paper, we consider that the octree (OT) representation of a given volume dataset has been obtained in a pre-processing step in which homogeneous zones have been identified, and the error of each one of the nodes is known [Boada99].

A **cut on the octree** is a set of nodes such that, for each possible root-leaf path, one and only one of its nodes is contained in the cut (see figure 1). The cut gives a multi-resolution representation of the volume as a set of nodes of different sizes and with different associated errors upper to the one considered in the leaf OT nodes. No redundant data is stored in the cut, and all the voxels are represented with more or less accuracy by one of its nodes.

In [Boada99] we present an algorithm to select an optimal OT cut. It is optimal in the sense that the enterely cut can be represented in the available texture memory. The user fixes the desired degree of interactivity and the regions of maximal interest. The algorithm provides the nodes of the cut and the level of resolution that must be used for their representation in order to optimize the image quality.

To describe the 3D texture based octree visualization algorithm we consider that all the required information of the cut is known.

4 THE 3D-TEXTURE BASED OCTREE VISUALIZATION

The visualization algorithm can be applied for the rendering of any cut of the octree. We consider that it is applied to a C cut of the octree, where $C = \{n_0, ..., n_l\}$. The rendering process, following the description of the technique given in section 2, is decomposed in two steps: the representation of the cut in texture memory and the cut rendering. Next subsections gives all the details of these steps.

4.1 OT Cut Representation in Texture Space

The OT cut representation in texture space is obtained from the representation of each one of its nodes in an independent texture. Given a node of C two situations are possible :

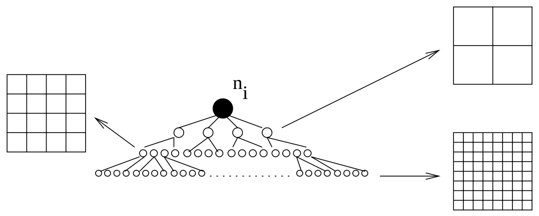

Figure 2: Given a node n_i of one of the possible cuts on the OT of figure 1, we show the corresponding subtree; because n_i is not a leaf node, different texture representations can be chosed according to the depth of the subtree rooted in n_i

- *The node is a leaf* (see node n_j in figure 1). In this case, we consider that all relevant information of the subvolume represented by this node can be obtained from the samples distributed over the vertices of the node (*external samples*). This is the information we will represent in texture memory.

- *The node is not a leaf* (see node n_i in figure 1). In this situation, the volume information of the node is represented by the subtree rooted in n_i. To store this information in the texture several options are possible. These options vary from the finest representation, that is a grid $2^k x 2^k x 2^k$ with k the depth of the deepest descendent of node n_i, to the coarser, obtained from the children of the node (see figure 2). The selected representation determines the samples that will be considered for the representation of the node in the texture space. To select which is the best representation of the node a user fixed criteria based on the node's degree of interest, or node's degree of homogeneity, or node's error, ... could be applied.

  In particular, if the selected level for the representation is q the region covered by n_i is represented by the set of all external samples of descendent nodes of n_i that are at level q.

It could be concluded that the assignation of one texel per voxel, will be applied only when the node of the cut is represented by minimal division nodes (i.e. voxels). In all other situations, the assignation policy will assign a more synthethical data size to texture size correspondence. Based on this new assignation rule, the representation of the cut in texture memory can be selected in such a way that the enterely cut could be represented in texture memory. This situation is considered in the optimal cut, optimal because all the volume is represented in the available texture space.

4.1.1 From Samples to Texels

Once the set of samples that will represent the node's volume are selected we have to represent them in texture memory. Differently of classical methods, in which the application of a transfer function and/or a look-up-table maps voxel field values to the RGBα texel values, we apply an opacity correction when a subsampled representation of the node is stored. In the following we describe how this opacity correction factor is obtained.

Following [Levoy90], the accumulated opacity α_{out} of a volume element can be expressed as

$$\alpha_{out} = \alpha + \alpha_{in}(1-\alpha) \qquad (1)$$

where α is the opacity of the volume element and α_{in} is the entering opacity to this element. The α_{out} value depends directly on the number of samples that are considered. The higher is the resolution of the representation the better are the results. Thus, the best representation will be obtained when the node is represented by minimal division nodes (i.e. the classical assignation of one texel per voxel).

In figure 3, the (a) representation is the best, it is defined at voxel level. But, if (b), or (c) has to represent the same area of the volume it should be desiderable the difference $\alpha_a - \alpha_b$, or $\alpha_a - \alpha_c$, to be minimal. To reduce this difference (i.e. to obtain images very similar to the one we obtain using the higher resolution representation), we propose to apply an *opacity correction* factor represented as α_c to the samples of the subsampled representation.

To compute the opacity correction factor we analise the two situations of figure 4 in which, for simplicity, only the first samples of a ray from (a) and another from (b) representations of figure 3 are evaluated. These samples are composited in back-to-front order, where α_0, α_1 and α_2

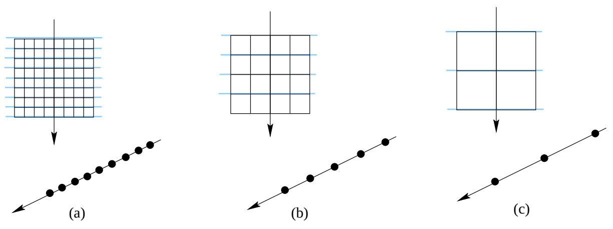

Figure 3: Possible representations of a volume node. The higher is the resolution the better are the results.

represents opacities of s_0, s_1 and s_2 respectively. Applying equation (1),we obtain that the accumulated opacity in each situation is

$$\alpha_a = \alpha_2 + \alpha_1(1-\alpha_2) + \alpha_0(1-\alpha_1)(1-\alpha_2) \quad (2)$$

and,

$$\alpha_b = \alpha_2 + \alpha_0(1-\alpha_2) \qquad (3)$$

respectively. As both situations represent the same area of the volume, the $\alpha_a - \alpha_b$ difference has to be minimal. To reduce this difference we apply the opacity correction at α_0 in the coarser representation. To determine the α_c value an approximation of α_1 for the coarser representation is required. As this approximation depends on the criteria applied for the octree construction we consider two possible situations, and we describe how the α_c is obtained in each one.

(i) The first situation consider that the applied criteria for the octree construction considers a node *homogeneuous when all the samples are equal.* Under this criteria we can substitute in equation (2) α_1 for α_0, and we obtain that α_0 has to be substituted for $\alpha_0+\alpha_0(1-\alpha_0)$ in equation (3) to obtain the minimal difference between representations $(\alpha_a - \alpha_b)$. In this situation

$$\alpha_c = \alpha_0(1-\alpha_0)$$

In general, if the difference between the higher resolution representation and the selected one is n (i.e. there are $n-1$ samples between the two represented samples) and thus the represented samples are α_0 and $\alpha_n + 1$ the α_0 has to be substituted for

$$\alpha_0 - (1-\alpha_0)((1-\alpha_0)^n)$$

(ii) The second situation is when the applied criteria for the octree construction considers a node *homogeneous when internal samples can be obtained by interpolation of the external ones.* Under this criteria we can substitute in equation (2) α_1 for $\frac{\alpha_2-\alpha_0}{2}$, and we obtain that the minimal $\alpha_a - \alpha_b$ is obtained when α_0 is substituted for $\alpha_0(1+\alpha_2)$ in equation (3).

    In general if the difference between the higher resolution representation and the selected one is n, to determine the α_0 value we have to consider that $\alpha_i = \frac{(n-i)\alpha_0+\alpha_n}{n}$. The accumulated opacity on a sample can be obtained as $T_{\alpha_i} = \alpha_i + (1-\alpha_i)T_{\alpha_{i-1}}$, thus the opacity correction in this situation will be obtained from the difference between T_n-T_0.

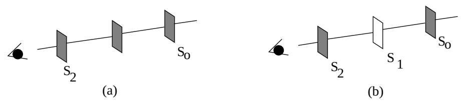

Figure 4: The composition of the first samples of the ray. In (a) three samples are composited and in (b) only two

The complexity of the computation of the α_c value depends on the criteria applied for the octree construction.

4.1.2 Texture Restrictions

Two restrictions have to be considered. The first one, is imposed by hardware and forces texture data loaded in texture memory to be packed into a rectangular parallelepiped, each dimension constrained to be a power of two [1].The second one has to be accomplished to guarantee image quality. To avoid artifacts between adjacent nodes in the final image, each texture node has to share its boundary with its neighboring texture nodes.

Driven by these restrictions, if n_i is a k level node of the OT cut and q is the level we will use to represented it, $k \leq q$ we consider that the region covered by n_i can be represent by a voxel set composed of all external samples which delimits q level cells in the subtree of n_i. Thus, $(2^{(q-k)}+1)^3$ texels are required to represent the node. As we have to store neighbor information we define a $(2^{(q-k)}+2)^3$ texture, using $(2^{(q-k)}+2)^3 - (2^{(q-k)}+1)^3$ texels to store it. If $q = k$ the texture is $(2^1+2)^3$.

[1] OpenGL supports a border option that allows textures to be (2^n+2)

4.2 OT Cut Rendering

The OT cut rendering, which only considers orthographic projections, is based on an iterative application of the node visualization function which computes the contribution of each node to the final image. The composition order of the nodes is obtained directly from a back-to-front octree traversal, driven by the viewing direction.

4.2.1 Node Visualization Function

The function is based on the [Gelde96] algorithm; all texture space transformations and clipping plane performace are implemented using graphics library procedures

To describe the function we consider that it is applied to the node n_i of the cut. The texture representation of this node is T_i, and the number of polygons np that will be composited to obtain the final node's contribution to the image depends on the resolution of the texture. The number of planes used for the rendering coincides with the number of planes used to obtain the sampled representation of the node.

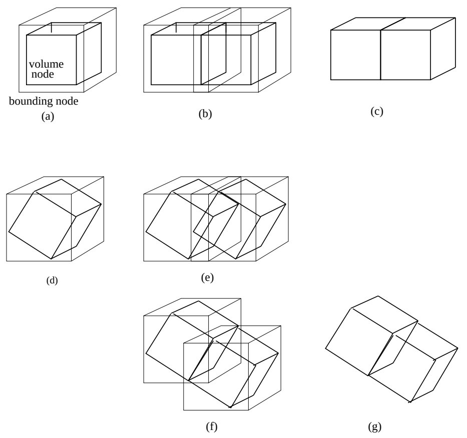

Figure 5: This illustrates an orthographic projection of two 3D bounding-node, the first in the original orientation (a) and the second one rotated (d). Neighbor node is correct positioned by a set of transformations, (b)(e) and (f).

The function can be decomposed in the following steps:

1. Bounding node definition. A bounding cube centered on the center of the node n_i is defined. We call it bn_i, *bounding node.* The corresponding polygons np_i to be rendered form series of slices through bn_i, parallel to the xy face. Texture coordinates will be

assigned to corners of these polygons in such a way that they interpolate into the texture-space T_i of this node when they are within the node.

If l_n is the lenght of the node's edge, the corners of the bn are $(\pm\frac{l_{bn}}{2}, \pm\frac{l_{bn}}{2}, \pm\frac{l_{bn}}{2})$, where $l_{bn} = l_n * \sqrt{3}$

2. Clipping Planes activation.The set of np_i polygons to be rendered always remains parallel to the projection plane, and extend to the bounding node. The node's volume is represented in T_i and rotates into bn_i according to the viewing direction (see figure 5(a)(d)) . To guarantee that only the volume information represented in the texture will contribute to the final image (i.e. external areas of the node contained in bn_i are eliminated) a set of clipping planes is defined. The clipping planes are positioned according the viewing direction at each one of the node's volume faces.

3. Node's Positionment. Once the *clipped polygons* that represent the volume node are computed, we determine where they have to be projected. The position of these polygons is fixed by the node's octree position and by the viewing direction. These parameters determine the set of geometrical transformations to be done.

   A first geometrical transformation determines the position of the bn_i according to the OT position (see figure 5(b)(e)). A new orientation requires a rotation of the texture space, keeping bn_i stationary. This rotation is done independently for each node according to the viewing direction. To maintain continuity between nodes, a second transformation is applied (see figure 5(f)). This transformation translates the bounding node.

4. Texture Mapping. Finally, texture coordinates are assigned to the *transformed and clipped polygons* in such a way that they interpolate into the texture-space when they are within the node. All mathematics to generate texture coordinates are based on [Gelde96].

Once these steps are applied we obtain the final I_i contribution to the image.(see figure 5(c)(g)).

5 RESULTS

The algorithm has been implemented and applied to several cuts of an octree representation of a CT head volume data model of $128 * 128 * 128$. The cuts were rendered on a SGI 02 computer system.

The first rendering, see figure 7, corresponds to a cut composed by 8 nodes of 64*64*64. All the nodes have been represented at the higher resolution and have been rendered with the same number of polygons. In this situation the image quality offered by the proposed algorithm is the same as the one obtained when the volume is rendered with the [Gelde96] algorithm where a unique texture of $128 * 128 * 128$ has been defined, see figure 6.

This first rendering (figure 7) shows that precission errors that could appear in node's positionment (see 4.2.1 (3)) are not perceptible if the neighbor bounding nodes are represented at the same resolution, and rendered with the same number of polygons.

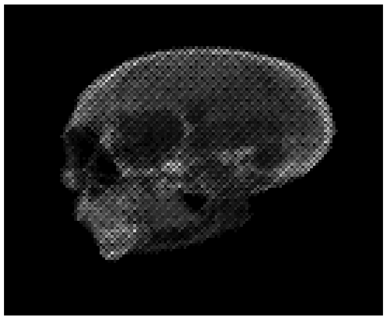

Figure 6: Image of $128x128x128$ CT model rendered by [Gelde96] algorithm.

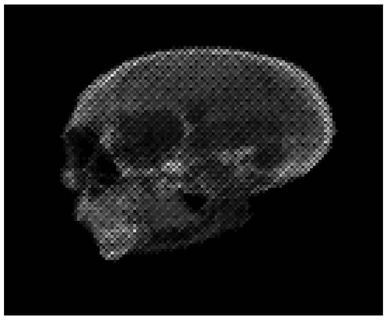

Figure 7: A cut of the CT head octree, composed by 8 nodes of $64x64x64$ rendered by the proposed algorithm

To evaluate the image quality offered by the algorithm when it is applied on a cut in which the nodes have been represented in texture space at different resolutions, we have selected a region of maximal interest, see figure 8(a). For this particular situation the rendered cut is composed by 6 nodes of 64*64*64 voxels and 16 nodes of 32*32*32 (see figure 8(b)). The texture resolution of each node is set according the degree of interest of the node. In this example, only the marked nodes of figure 8(b) are represented at higher resolution (i.e. $32x32x32$ texels). While textures that represent all other nodes are obtained from a subsampling of the node. In next tables we represent the resolution of the volume covered by the node (*voxel resolution*) and the resolution of the texture representation (*texture resolution*) that have been defined to obtain the renderings of figure 10 and figure 11 respectevely. To compute the required texture memory it is necessary to add to the node's texture space the boundary texels.

number nodes	*voxel resolution*	*texture resolution*
6	$64x64x64$	$32x32x32$
8	$32x32x32$	$32x32x32$
8	$32x32x32$	$16x16x16$
Figure 10	Texture space	491.520
	Boundary texels	105.392

number nodes	*voxel resolution*	*texture resolution*
6	$64x64x64$	$16x16x16$
8	$32x32x32$	$32x32x32$
8	$32x32x32$	$8x8x8$
Figure 11	Texture space	290.816
	Boundary texels	66.608

Of course, the image quality of regions of no interest is not comparable to the rendering that we obtain when these regions are rendered at higher resolution (see figure 9). But we have to consider the benefits of this new representation in terms of texture space. For figure 9 2.097.152 texels are required, in front of the 592.912 texels required for figure10 and the 357.424 texels required for figure11. It can be seen that savings are about 70% per cent and thus interactive rendering times are possible to obtain images of accepted quality.

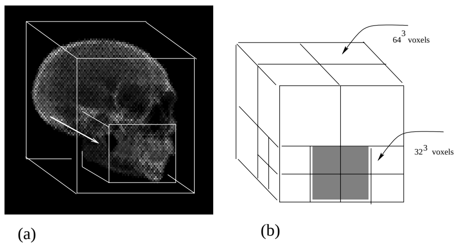

Figure 8: (a)The region of maximal interest is selected.(b) The cut of the octree is selected according the region of interest.

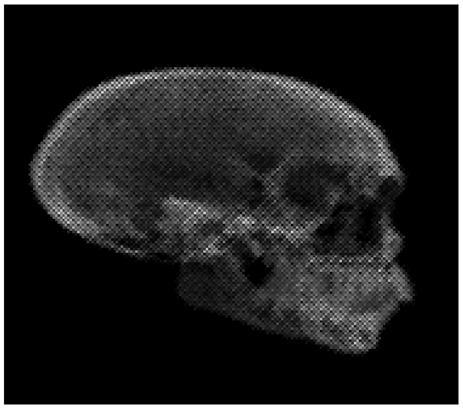

Figure 9: The CT head rendered with a $128 * 128 * 128$ texture. (2.097.152 texels are required)

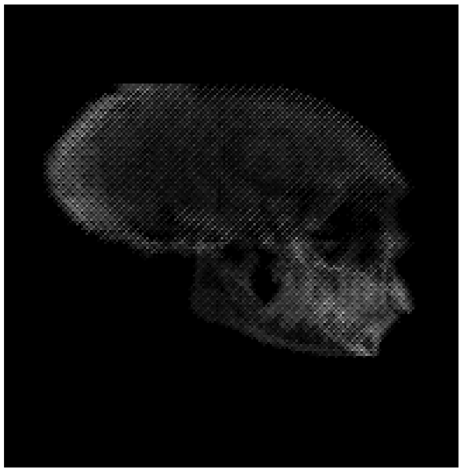

Figure 10: A cut of the CT head octree in which only the region of maximal interest is rendered at higher resolution. The representation of no interesting nodes is obtained from a subsampling. (592.912 texels are required)

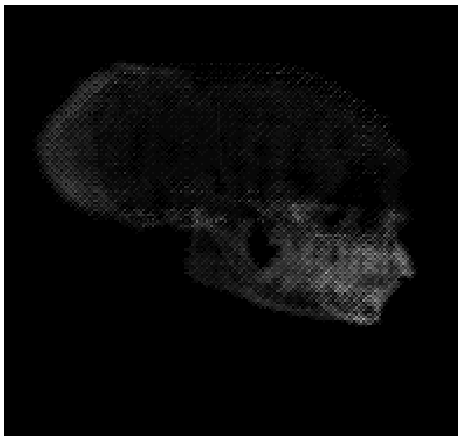

Figure 11: A cut of the CT head octree in which only the region of maximal interest is rendered at higher resolution. The representation of no interesting nodes is obtained from a subsampling. (357.424 texels are required)

6 CONCLUSIONS AND FUTURE WORK

We have presented a 3D texture based Octree Volume Visualization algorithm that combines 3D texture hardware and hierarchical representation of data sets to obtain multiresolution renderings. The basis of the algorithm is a new rule that takes advantage of iso-valued areas of the volume or regions of no interest, and modifies the classical texture volume representation for a more synthetical one. This new assignation policy improve the use of 3D texture memory and optimize the explotation of 3D texture based visualization on large data sets.

Given an octree representation of the original data set, one cut of this octree is selected. Each node of the cut is represented in texture memory at one determined accuracy according the degree of homogeneity and the degree of interest of the node. Once all the nodes of the cut are represented in texture space a rendering function is applied to each one to obtain the final image. The algorithm guarantees at regions of maximal interest the image quality offered by classical 3D texture algorithm [Gelde96].

In this first stage of our work we have prioritized the evalution of image quality in front of rendering speed, nevertheless, interactive times are obtained. In our future work some other strategies to improve image quality will be investigated, different policies to speed up the rendering time. We will also work on the extension of the algorithm for perspective projections.

REFERENCES

[Akele93] Kurt Akeley, Reality Engine Graphics, Computer Graphics (ACM Siggraph Proceedings), 27:109-116,1993.

[Boada99] I.Boada,I.Navazo,R.Scopigno, *An Interactive 3D Texture Based Volume Visualization Using an Octree based Multiresolution Representation.*Technical Report(IIiA99-16-RR). Girona University (Institut d'Informatica i Aplicacions)

[Cabra94] Brian Cabral, Nancy Cam and Jim Foran, *Accelerated Volume Rendering and Tomographic Reconstruction using Texture Mapping Hardware*, In ACM Symposium on Volume Visualization,pp. 91- 98 Washington, D.C. October 1994

[Culli93] T.J.Cullip, U. Neumman. *Accelerating volume Reconstruction with 3D texture hardware.* Technical Report TR93-027, University of North Carolina, Chapell Hill, 1993.

[Guan94] Sheng-Yih Guan and Richard Lipes. *Innovative Volume rendering using 3D Texture Mapping.* In Image Capture, Formatting and Display.SPIE 2164, 1994

[Grzes98] R. Grzeszczuk, C.Henn, and R.Yagel.*Advanced Geometric Techniques for Ray Casting Volumes.* Course Notes. SIGGRAPH '98. ACM July 1998.

[Levoy90] Marc Levoy.*A hybrid ray-tracer for rendering polygon and volume data.* IEEE Computer Graphics and Applications,10 (2):33-40, March 1990.

[Laur91] David Laur and Pat Hanrahan. *Hierarchical Splatting: A progressive refinement algorithm for volume rendering.* Computer Graphics (ACM Siggraph Proceedings), 25 (4):285-288, July 1991.

[Srini97] Rajagopalan Srinivasan, Shiaofen Fang, Su Huang. *Volume Rendering by Template-Based Octree Projection.* Workshop Eurographics 97. Visualization in Scientific Computing.

[Tong99] Xin Tong, Wenping Wang, Waiwan Tsang, Zesheng Tang. *Efficiently Rendering Large Volume Data Using Texture mapping Hardware.* Data Visualization '99 (in press).

[Wilhe92] Jane Wilhems, Allen Van Gelder. *Octrees for Faster Isosurface generattion.* ACM Transactions on Graphics, 11(3):201-297, July 1992.

[Wilso94] Orion Wilson, Allen Van Gelder, Jane Wilhems.*Direct Volume rendering via 3D textures.* Technical Report UCSC-CRL-94-19, University of California, Santa Cruz, June 1994

[Gelde96] Allen Van Gelder, Kwansik Kim. *Direct Volume rendering with shading via 3D textures.* Symposium on Volume Visualization 1996, pp 23-30.