UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2569

Implementing and Applying Direct Volume Rendering with Textures

Igor Prata Soares
Rosane Minghim
Maria Cristina Ferreira de Oliveira
Luis Gustavo Nonato

Nº 128

RELATÓRIOS TÉCNICOS

São Carlos – SP
Dez./2000

Implementing and Applying Direct Volume Rendering with Textures

IGOR PRATA SOARES
ROSANE MINGHIM
MARIA CRISTINA FERREIRA DE OLIVEIRA
LUIS GUSTAVO NONATO
Departamento de Ciências de Computação e Estatística
ICMC/USP - São Carlos
Caixa Postal 668
13560-970, São Carlos - SP, Brazil
{igor, rminghim, cristina, gnonato}@icmc.sc.usp.br

Abstract This paper reports on the implementation of a Direct Volumetric Rendering (DVR) and Visualization technique in the context of a general-purpose, low cost visualization software with available source code. The technique employs a texture mapping strategy to combine scalar values of volumetric data into three-dimensional images. Relevant implementation issues of this technique are presented, and the results are discussed in the light of a target application, namely visualization in dentistry. The technique presented here, DVRT, has demonstrated to behave better than the ray casting DVR technique for the data sets of interest, and also to produce very good results under shading.

1 Introduction

The construction of three-dimensional (3D) models from two-dimensional (2D) information collected from a number of acquisition devices is a major step in many visualization applications. The field of Medicine has been particularly successful in pursuing such a visualization approach to obtain 3D models depicting internal body structures. Resulting models can play a major role in a number of tasks, from educational and training systems to diagnosis and technologically advanced cooperative surgery.

For a number of reasons, the use of graphical and visualization algorithms has been much less noticeable in Dentistry than in the related field of Medicine. However, reconstruction is also potentially useful for a number of tasks in Dentistry. For example, training of dentistry professionals could certainly benefit from the visualization of realistic 3D models of teeth that could support, for example, the simulation of dental procedures. Some activities in this field are being reported in the literature, as in mandible reconstruction [1] and chewing simulation [2]. A project on Virtual Dentistry is being conducted at the Computer Science Department of ICMC/USP, in collaboration with the School of Information Systems of the University of East Anglia, and with the Dentistry School of UNESP at Araraquara. A major goal of this project is the building of a system to support education, training and assistance for dentistry professionals and students. A virtual environment shall provide a set of 3D models of teeth for user interaction.

Initial reconstruction tasks that generated realistic 3D teeth models have been performed using general-purpose packages [3] and building extensions to others, such as the work of Shimabukuro et al. [4,5]. The work of Nonato [6], which is an improvement of classical volume reconstruction methods based on Delaunay Triangulations (DT), has proven to be more adequate to our reconstruction needs [5]. In these works, surface or volume reconstruction is performed from 2D contours extracted from images. In such approaches the resulting model can be rendered using conventional surface rendering algorithms.

As opposed to the surface fitting approach above, DVR generates images straight from scalar data sets, without creating intermediate geometrical structures [7,8]. Popular approaches to DVR are the well-known ray casting algorithm and the splatting algorithm. We are interested in investigating DVR in the scope the Virtual Dentistry project, because it presents some advantages when compared to a surface fitting approach. One of them is that it preserves more information from the data set, rather than displaying only user selected values. This should enable, for example, easier observation of tissue. A major problem, however, is that it is difficult to achieve real time user interaction in DVR, as most algorithms are computationally expensive.

Dedicated texture mapping hardware, usually targeted at adding realism to scenes created with conventional surface graphics techniques, is being investigated as a feasible alternative for implementing fast DVR of scalar data. Several strategies that carry out such an idea are reported in the literature [9-12]. These solutions may ensure considerable speedups in the costly volume rendering processes by means of graphical hardware [12,10]. The same literature describes the advantages and disadvantages of using a texture mapping approach with graphics hardware for speeding up the rendering process. Instead of handling this issue, here we discuss the visual differences of a 3D texturing DVR approach against another, classical, DVR method.

We present a particular implementation of a DVR algorithm with texturing (DVRT) to generate images from volumetric scalar data sets. We show and compare visual results obtained from applying this solution, illustrating its use to generate images in our target application domain (dentistry). In the following section we present the DVRT technique and describe its implementation within the VTK environment. In Section 3 we discuss some results from the use of this approach in the dentistry case, analyzing its potential for further visualization, exploration and virtual manipulation of teeth and related structures.

2 DVR with texture (DVRT)

In this section we describe a VTK implementation of a DVRT 3D algorithm – Direct Volume Rendering with 3D Textures. In Section 2.1 we present an overview of the DVRT algorithm introduced by Gelder et al. [12], and in Section 2.2 we describe our VTK implementation of a particular version of that approach.

2.1 The DVRT Approach

In short, the DVRT approach constructs a volumetric texture map (*vtex*) from a data volume that is then sampled through parallel planes. The final image is obtained by blending the planes together in back-to-front order. The procedure for implementing a DVRT may, therefore, be split into two stages. The first one builds the *vtex*, whereas the second one samples the *vtex* in parallel planed and displays the blended planes, generating a 3D image of the data. We shall describe both stages in the sequence.

The data volume consists of volume units called voxels, and the texture map consists of 3D units of texture named texels. The texture map is a volumetric mesh whose data attributes are given by four values defining a color (R,G,B,A). The alpha-values A encode the opacity of a particular pixel or texel. Dimensions are the same for volume and texture, ensuring a one-to-one correspondence between the sets of voxels and texels. Building the texture map involves collecting the scalar values from the data volume and associating a color to their corresponding positions in the texture map. To build this mapping the user must define a set of transfer functions that associate scalar ranges to colors and opacities, an activity known as *data classification*.

Once the 3D texture map is available, the second stage of DVRT is ready to start, that is, the sampling and blending of the volume. The steps involved in this stage may be considerably complex if shading is required. For shaded rendering, data classification requires, in addition to the transfer functions, the definition of the materials contained in the volume. A material is defined by a scalar range plus a set of parameters that control its appearance under direct light. These are the reflection coefficients *kd* for diffuse light, *ks* for specular light, *sp* for specular power, and *ka* for ambient light.

The shading equation is used to compute the interaction between the directional lights in the scene and the volume data, normally employing the normal vectors at points of the surface being shaded. In this case, normal vectors are approximated by gradients computed from the scalar values placed at all volume voxels [13]. This may be done using central differences [7,14]. Gradient values are also used to classify voxels as ambient or reflectin. Reflectin voxels are those in the boundary of a particular material, and ambient voxels are considered to be inside the object defined by a given material. Classification is obtained by considering the voxel's scalar value (d), its gradient (γ), and the lower end of the scalar range of the voxel's value (m). If Δ is the spacing between voxels, one of the 26 neighboring voxels contains a scalar smaller than b if and only if $(m - b) < s$, where:

$$s = \Delta_x |\gamma_x| + \Delta_y |\gamma_y| + \Delta_z |\gamma_z|$$

s is called the cell-diagonal data shift. When s is close to the value $(d - m)$ one can find the probability of its host voxel being at the boundary of the material, rather than taking a binary decision. When $s \leq 0.5(d - m)$ this probability is zero, and it increases linearly to 1 when $s \geq 1.5(d - m)$. It is possible to fine-tune the voxel classification by adjusting the classification scale factors (constants 0.5 and 1.5 above) to increase or

decrease the 'thickness' (in number of voxels) of the object boundary.

Only ambient voxels receive the ambient light component of the illumination equation, and the reflection components are added to the reflecting voxels. Such components can be tabled to speed up the process of building the texture map. A Gradient Index can be used to access unit gradients, generated by a process of quantization over a sphere, that approximates a set of voxel gradients. A Reflection Table stores the reflection factors for each combination of material and gradient present in the volume. The size of this table is, therefore, given by the number of quantized gradients multiplied by the number of different user defined materials. Each voxel in the data volume has an index in the Reflection Table. As the Reflection Table depends on the direction of the lights relative to the volume, it must be updated after any movement in the scene. For further details the reader is referred to [15] and [16].

The voxel classification stage depends on the transfer functions and the scale factors mentioned previously. Classification must take place after any change on any of these. The updating of the Reflection Table occurs after every change in the volume position or in the illumination parameters (including scene position). All such changes cause the texture map to be fully rebuilt.

Figure 1 provides an overview of the DVRT process, with and without shading. Rendering of the texture map is carried out in the second stage by sampling and blending, as discussed in the following.

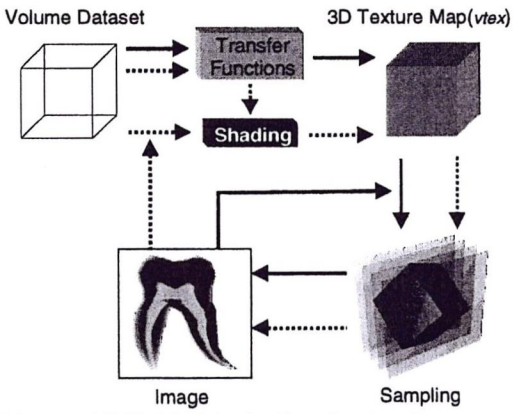

Figure 1 DVRT with shading (broken lines) and without shading (full lines).

Rendering in DVRT is performed by applying texture on consecutive sampling planes, all parallel to the viewing vector and placed at equally spaced intervals according to a user defined distance. To allow volume visualization from any viewpoint, a bounding cube is defined for the volume whose edges have the same size as the diagonal of the volumetric texture. When the volume is rotated, the bounding cube is positioned so that one of its faces is parallel to the projection plane. The direction of that face drives the planar samplings of the volume within the bounding cube.

The sampling process requires the texturing of the sampling planes. Since these are parallel to the viewing plane, they cover a rectangular region of the screen. Sampling points are defined at the planes that correspond to pixels in this region. For each pixel, a sample collects an RGBA color in the texture map. The intersection between a sampling plane and the texture map is named a texture polygon. Such a polygon bounds the region of the sampling plane where color samplings must occur, avoiding unnecessary calculations over the whole plane. A blending operator that processes all sampling planes in back-to-front order is used to combine all samples relative to a single pixel.

Shading in DVRT is very time consuming, since every user interaction causes the texture map to be rebuilt, which in turn affects the whole process. Its use with DVRT is justified in some situations because it improves 3D perception and helps to resolve ambiguity. The DVRT algorithm has been implemented within the VTK visualization library [7], thus making its code available to the whole community of VTK users. Details of the implementation are given in the next section.

2.2 VTK implementation of the DVRT algorithm with shading

VTK is an object-oriented visualization library with available source code. To implement the DVRT approach, VTK classes were analysed, particularly those responsible for Ray Casting. Two new classes were added, named *vtkVolumeTexture* and *vtkVolumeTextureMapper*, which implement each of the two stages of DVRT. VTK implementation standards were followed to maintain consistency for potential users of the new classes.

The *vtkVolumeTexture* class creates the texture map. Both the data volume and the texture map are regular grids, represented as instances of the VTK class *vtkStructuredPoints*. The data volume is loaded using VTK's reader objects and is connected to *vtkVolumeTexture*, in a pipeline fashion, through its *SetDataset* method. VTK supplies two classes that implement transfer functions for defining RGB color

and opacity mappings, *vtkColorTransferFunction* and *vtkPiecewiseFunction*, respectively. Two methods of *vtkVolumeTexture* are used to set these parameters: *SetColor* and *SetOpacity*, respectively.

Definition of materials is done using the method *AddMaterial*, which takes two scalars that delimit an interval, plus the four illumination coefficients *kd*, *ka*, *ks*, *sp*. Two classes were coded for handling materials: *vtkMaterialCollection* and *vtkMaterial*. The first class manages a list of the latter, which actually contains the material information. In fact, class *vtkVolumeTexture* contains an object of the *vtkMaterialCollection* class. A reference to that instance can be obtained, if necessary, by use of the method *GetMaterials*.

Classes *vtkVectorIndex* and *vtkReflectionTable* realize the Gradient Index and the Reflection Table, respectively. *vtkVolumeTexture* handles those in a way that is transparent to the user. However, methods for their manipulation by the user are also supplied. The *vtkVectorIndex* class generates a table of vectors that recursively refines a unit sphere. The starting point of the refinement is a regular icosahedron that has its triangles further subdivided at each step, with the new points being displaced by a one unit distance of the center. The user sets the refinement level through the method *SetLevelOfRefinement*. A larger refinement level improves image quality at an extra computational cost. To speed up calculations, class *vtkVectorIndex* handles the first octant of the sphere and gets the others from signal permutation. Taking a generalized vector as input, method *GetVector* returns an index to the closest approximation to it in the vector table.

Class *vtkReflectionTable* stores the reflective light component for each combination of gradient and user defined material. Table building is managed by the method *BuildReflectionTable*. The method *GetReflectionComponent* returns a reflection component tuple from the table, taking as input a gradient vector index. The methods *SetShadingOn* and *SetShadingOff* can be used to control shading, and the methods *SetUseOfReflectionTableOn* and *SetUseOfReflectionTableOff* to choose from using the reflection table (faster) or calculating all the illumination values on the fly (storage saving).

Gradient sense (inward or outward to the object being rendering) bears importance in the illumination effects, and as a consequence sometimes the "shaded surface" turns out to be inside the object, rather than outside. A method *InvertShadingHemisphere* can be used to toggle illumination sides. Scale factors are set by two methods: *SetLowScale* (more voxels in the boundary of the material) and *SetHighScale* (less voxels in the boundary).

The main method of *vtkVolumeTexture* is *MakeVolumeTexture*, which creates the texture map. It generates the map according to user settings, and whenever necessary triggers gradient calculations, voxel classification and Reflection Table upgrades. The texture map created can be read using the method *GetVolumeTexture*.

The *vtkVolumeTextureMapper* class samples the texture map and renders its image. This class is directly related to VTK's standard DVR visualization pipeline. To render an image, the user calls the method *Render* of class *vtkRenderWindow*, which chains a series of calls. In DVRT, *vtkVolumeTextureMapper* has its own method *Render*, where everything is set, including the decision of actually building the texture map. In this method, two floating-point arrays are filled up (*RGBAImage* and *Zimage*), which store the image produced by DVRT with the information on color and depth, respectively. From those arrays, VTK itself takes charge of presenting the image on a rendering window, together with other possible objects in the scene. Such image arrays can be read from the class *vtkVolumeTextureMapper* using methods *GetRGBAPixelData* and *GetZBufferData*. Only parallel projection is available for DVRT at this point.

In a user program for viewing a data set using DVRT, an instance of the *vtkVolumeTexture* class must be connected to an instance of *vtkVolumeTextureMapper* using the method *SetVolumeTexture*. The *Render* method calls *MakeVolumeTexture* to create or update the texture map. Such a design choice was made to allow checking of the scene configuration to decide whether an interaction has occurred and, therefore, there are things to be changed. That information belongs to *vtkVolumeTextureMapper*.

The sampling process is controlled by the user with methods *SetSampleDistance*, *SetSampleInterpolationOn* and *SetSampleInterpolationOff*. The first one sets the distance between consecutive sampling planes. *SetSampleInterpolationOn* allows for interpolation in the sampling process, so that each texel is influenced by the several color in its vertices. *SetSampleInterpolationOff* causes the texel to be viewed with uniform color in the whole of its interior, thus avoiding interpolation. Because DVRT is not yet implemented using texture hardware, sampling with interpolation is 8 times slower than the voxel type of

sampling. Use of proper 3D texture hardware will improve this discrepancy.

Arrays *RGBAImage* and *Zimage* are initialized with transparent black and maximum depth, respectively, and updated by the sampling procedure. The screen region covered by the sampling planes is delimited, and the numbers of pixels in the horizontal and vertical directions are computed. These are required to map points in the sampling plane to screen pixels. Textures are mapped to the sampling planes in a back-to-front order, and only points within the texture polygon are sampled. For each sample, a color is collected from the texture map, with or without interpolation. Nothing is done if the opacity is 0.0; otherwise, the associated pixel position in the *RGBAImage* is updated with blending. The corresponding position in the *Zimage* array is also updated to the depth of the plane. When the sampling ends, the arrays contain the resulting DVRT image.

Figure 2 shows the outline of a C++ program that uses the VTK DVRT. The code uses DVRT with shading (line 5), using Reflection Table (line 6) and an interpolated sampling process (line 10). Variables *VolTexMapper* and *VolTex* are instances of *vtkVolumeTextureMapper* and *vtkVolumeTexture*, respectively.

In line 1 we observe the connection between *vtkVolumeTexture* and the data volume (*Dataset* is any *vtkStructuredPoints*). Lines 2 and 3 show the assignment of color and opacity transfer functions. In line 4, a material is defined (*Min* and *Max* define its scalar range, and the illumination constants are given by *ka*, *kd*, *ks* and *sp*). The refinement level for gradient quantization is set in line 7. Line 8 shows the connection between *vtkVolumeTexture* and *vtkVolumeTextureMapper*. Configuration of sampling plane distance is given by the command on line 9. Line 11 configures VTK for parallel projection. Lines 12 and 13 set the VTK objects required for the rendering process.

```
1  VolTex->SetDataset(Dataset);
2  VolTex->SetColor(ColorTransferFunction);
3  VolTex->SetOpacity(OpacityTransferFunction);
4  VolTex->AddMaterial(Min,Max,ka,kd,ks,sp);
5  VolTex->SetShadingOn();
6  VolTex->SetUseOfReflectionTableOn();
7  VolTex-
>SetLevelOfRefinamentToGradientIndex(Level);

8  VolTexMapper->SetVolumeTexture(VolTex);
9  VolTexMapper-
>SetSampleDistance(SampleDistance);
10 VolTexMapper->SetTexelAsCell();

11 Renderer->GetActiveCamera()
->ParallelProjectionOn();

12 Volume->SetVolumeMapper(VolTexMapper);
13 Renderer->AddVolume(Volume);
14 RenderWindow->AddRenderer(Renderer);
15 RenderWindow->Render();
```

Figure 2 Part of code for a DVRT program.

Every parameter that can be set in a DVRT program has a default value. If nothing is set by the user, shading and interpolation are off, for instance. Tables 1 and 2 show some of the default parameters of DVRT. In the following section we present various results obtained from use of DVRT.

vtkVolumeTexture class	**Default Value**
SetDataset	None
SetColor	None
SetOpacity	None
AddMaterial	None
SetLevelOfRefinament	1
SetShadingOn / Off	Off
SetLowScale	0.5
SetHighScale	1.5

Table 1 Default values for *vtkVolumeTexture* class

VtkVolumeTextureMapper class	**Default value**
SetVolumeTexture	None
SetSampleDistance	1.0
SetSampleInterpolationOn /Off	Off

Table 2 Default values for *vtkVolumeTextureMapper* class

3 Results and application

The implementation of DVRT was tested in three different ways. First, observation of the behavior of the resulting images against changes in the various parameters of the methods was carried out. This we call 'fine tuning' of the technique, that is, balancing the compromise between image quality and computing time for a particular application. The method behaved very well as far as image quality is concerned, and was very stable, in that noise generation and errors were very rare. Specifically, we tested:

- spacing between sampling planes: in this case image quality varies and, for larger distances, image continuity can be broken;
- resolution of gradient quantization: patterns appear on the picture for smaller amounts of gradients. Smoothness of visualization is

obtained by increasing the number of gradients, or by calculating them on the fly (without using reflection tables).

- use of interpolation between neighboring voxels. For coarser grids, jagged surfaces occur. Darkening of the surface may also appear, considering that empty spaces outside the data volume are filled up with black.
- illumination parameters: (diffuse and specular coefficients, as well as specular power) they control shading quite well, with similar behavior to that of surface rendering. Even when internal surfaces are shown through external, transparent surfaces, the effect can be clearly noted.
- light colors, hemisphere choice, and effect of interpolation on execution time were tested and confirmed to follow the statements previously presented in this text.

Figure 3 shows the effect of interpolation in the sampling process, for a data set with three color levels, and no shading. Figure 4 shows the effects of increasing the number of sampling planes. Figure 5 shows a shaded example of DVRT, with two different materials defined.

The other two classes of tests realized with DVRT were concerned with verifying its effectiveness for visualization of dentistry data, which is a problem our team has been working with; and comparing the final images with the most common DVR method, Ray Casting, also for dentistry data. The Ray Casting implementation used is the one provided by VTK itself.

A major argument for using DVRT instead of ray casting has been the possibility of obtaining images of similar quality with the speed up conferred by graphics hardware. That given, although we have not tried DVRT with any particular graphics hardware up to this point, we also identified some visual differences between both methods that tend to point towards DVRT, at least for some of the applications we have been handling.

In figure 6 there is a set of direct volume rendering of pictures of the same data set, using similar color definitions, without shading. This particular data set was generated by distance sampling of a set of tooth planar contours on a regular 3D grid. It can be seen from these pictures that the images from both methods are very similar (see, for example, fig 6 (a) to (d)). The main differences lye on treatment of transparency for some of the views. See, for example, the bottom (fig. 6(e) and 6(f)) and top views (fig. 6(g) and 6(h)). In that case, the appearance the DVRT results are more convincing. The external object actually covers the whole of the internal portion of the volume, and that situation is clearly identified for DVRT, while in Ray Casting, the internal portion misleadingly shows up in front. Also, some of the depth is lost.

Figure 7 shows similar views of the same data set as figure 6, but in this case, shading was added, both in Ray Casting and DVRT. Side views, although slightly different, give out similar effects as far as interpretation is concerned (fig. 7(a) to (d)). However, they present an interesting aspect. For the same visual attributes specified for both techniques, Ray Casting presents brighter images, as if reflection were stronger than in DVRT (although DVRT reflection also looks good). In that case the interpretation is subjective as to how two transparent reflecting surfaces should look like when subject to light. DVRT images, though, look smoother in that particular case. The same problem of transparency mentioned above can be once again noticed in figures 7(f) and 7(g).

An additional point is that, in some cases, as the case where the gradient changes are sharper in a particular view direction, the Ray Casting method tends to generate line patterns, similar to contours, while in DVRT this effect is not noticeable (see Figures 7(e) to 7(h)). Although this is a feature typical of the data set at hand (and similar ones), it is important to our application that the DVR methods can act in standard ways, as it happens with DVRT. DVRT can, though, produce spurious 'light points' when, together with this gradient problem, the resolution of the data set is lower, as in figure 8.

Smoothness of object surface is very nicely obtained using DVRT, particularly when less transparency is used. For similar visual attributes, figure 9 shows this aspect of the technique. In the following section we present some of the conclusions and further developments of the project.

4 Conclusions

The importance of having volumetric rendering in a visualization system has been put forward by almost the whole of the visualization community. In the virtual Dentistry project where the work reported here

takes part, an integrated system for visualization and interaction for training is sought. In this case, volumetric information is of utmost importance, and one of our goals is to discuss strategies for combining surface and DVR displays in a virtual environment.

Much of the information presented by DVR is missed in surface displays, so that multiple visualization techniques play an important role in data analysis. In the case of DVRT, the possibility of speeding up the process by means of graphics hardware (a follow up of this project), opens the possibility of doing joint visualization in real time with less expensive equipment.

The comparison between Ray Casting and DVRT shows that the latter by no means looses in quality, and in some cases, produces more stable and manageable behavior, for the particular application under study.

Acknowledgements

We wish to acknowledge Mike Goetz and Andy Day, from the School of Information Systems, UEA, UK, for the data used to generate most of the images presented here. We also wish to acknowledge the funding of FAPESP and CNPq, Brazil, and the good work of our undergraduate student Carlos Frederico Rocha, of ICMC-USP.

References

[1] S. Seipel, I. Wagner, S. Koch, W. Scheneider, "Three-dimensional visualization of the mandible: A new method for presenting the periodontal status and diseases", *Comput. Meth. Programs Biomed.* (46), 51-57 (1995).

[2] K. Myszkowski, G. Okuneva, J. Herder, T.L. Kunii, T.L, M. Ibusuki, "Visual simulation of the chewing process for dentistry", *Visualization and Modelling*, Academic Press, 419-438 (1997).

[3] M.R. Goetz, A.M. Day, "Surface reconstruction for teeth", *Proc.16th EUROGRAPHICS UK*, 25-27 March, Leeds , UK, (1998).

[4] M.H. Shimabukuro, R. Minghim; P. Licciardi, "Visualisation and reconstruction in dentistry", in *Proc. Int. Conf. on Information Visualisation IV'98*, London, UK, IEEE CS Press, 25-31 (1998).

[5] L.G. Nonato, R. Minghim, M.H. Shimabukuro, "Qualitative Analysis of Reconstruction Techniques for Dentistry", accepted for publication in the *J. of Electronic Imaging* (2000).

[6] L.G. Nonato, *Volumetric Manifold Reconstruction from Planar Sections*, Ph.D. Thesis (in Portuguese), Mathematics Dep., Pontifical Catholic University, Rio de Janeiro, Brazil, (1998).

[7] W.J. Schröeder, K. Martin, W. Lorensen, *The Visualization Toolkit, an object-oriented approach to 3D graphics*, 2nd ed., Prentice-Hall, (1998).

[8] A. Kaufman, "Advances in volume visualization", *SIGGRAPH'98 Course Notes no. 24*, Orlando, USA, 1998.

[9] B. Cabral, N. Cam, J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware", *Proc. 1994 Symp. on Volume Visualization*; 91-98.

[10] R. Grzeszczuk, C. Henn, R. Yagel, "Advanced geometric techniques for ray casting volumes", *ACM SIGGRAPH'98; Course Notes 04*; 1998.

[11] M. Teschner; C. Henn, "Texture mapping in technical, scientific and engineering visualization"; SGI, Aug. 4, 1995; also in *ACM SIGGRAPH'98 Course Notes 17*; Apr. 20, 1998.

[12] A.V. Gelder, K. Kim, "Direct volume rendering with shading via three-dimensional textures", University of California, Santa Cruz, CA 95064 USA; *Technical Report UCSC-CRL-96-16* (1996).

[13] D. Hearn, M. P. Baker, *Computer Graphics – C Version*; Prentice Hall, 1997.

[14] M. Levoy, "Display of surfaces from volumetric data", *IEEE Computer Graphics & Applications*, 8(3), 29-37, May 1988.

[15] J. Wilhelms and A. Van Gelder, "A coherent projecion approach for direct volume rendering", *Computer Graphics* 25(4); Aug. 1991; 275-284

[16] A.V. Gelder and K. Kim, "Direct volume rendering with shading via three-dimensional textures", University of California, Santa Cruz, CA 95064 USA; *Technical Report UCSC-CRL-96-16*; Jul. 19, 1996.

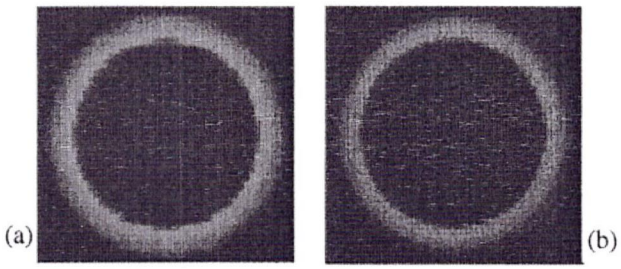

Figure 3 Effect of using interpolation between voxels during the sampling process. (a) without interpolation. (b) with interpolation (smoother). No shading is used.

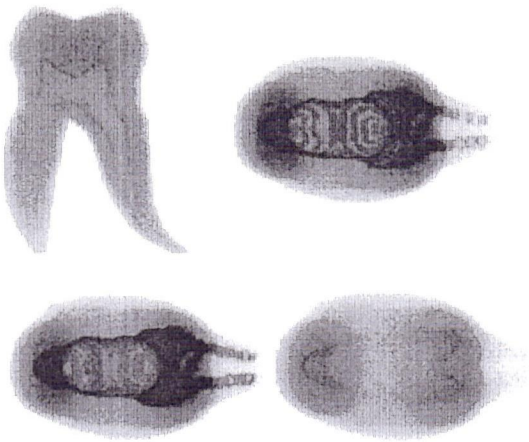

Figure 4 DVRT without shading for the dentistry data.

(a) side view of a tooth with definition of two different scalar ranges (one internal, another external).

(b) top view, with 40 sampling planes.

(c) top view, with 60 sampling planes.

(d) top view, with 80 sampling planes.

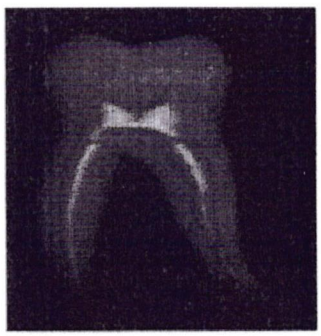

Figure 5 DVRT with shading, two materials defined.

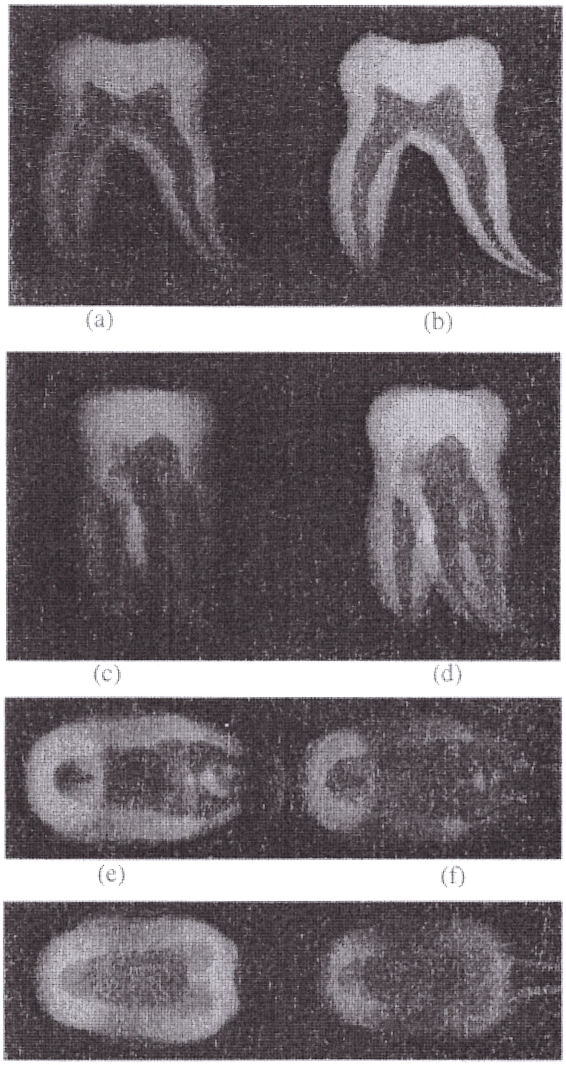

Figure 6 DVRT and Ray Casting of dentistry scalar data, without shading. Data Set is organized in a regular grid (306x174x83). Left Column is DVRT, right is Ray Casting

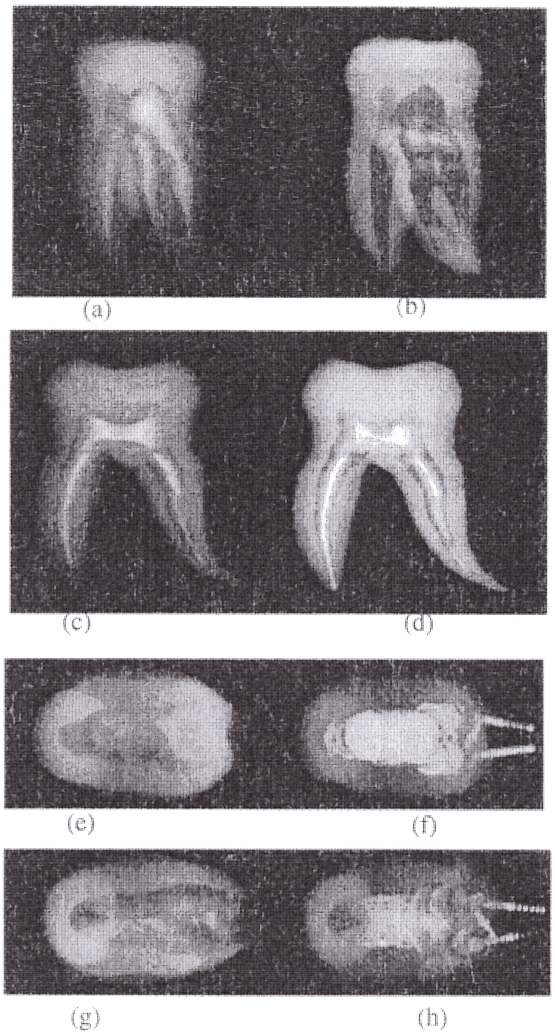

Figure 7 DVRT and Ray casting with shading for the same data as figure 6, in the same views. Illumination coefficients are: *ka*=1.0, *kd*=*ks*=0.5, *sp*=10.

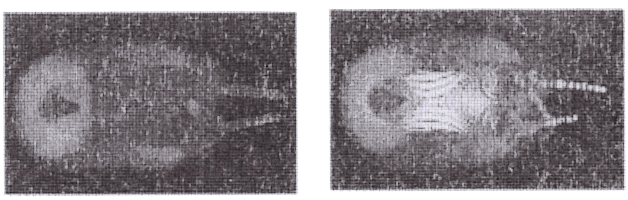

Figure 8 DVRT and Ray Casting of data with lower resolution (158x88x83), in a view direction with ill-defined gradients.

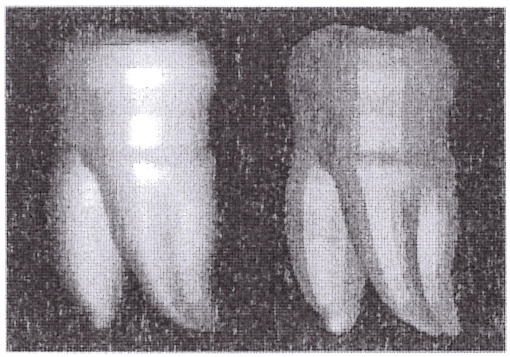

Figure 9 External surface of the tooth, rendered with shading for both DVRT (left) and Ray Casting (right).