Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2004)
O. Deussen, C. Hansen, D.A. Keim, D. Saupe (Editors)

Visualization of Cardio-CT Data on Standard PC Hardware

Michael Bauer[1]

[1] Computer Graphics Group, University of Erlangen-Nuremberg, Erlangen, Germany

Abstract

In the last years, texture based volume rendering on the PC platform has proven very successful. Many methods that could formerly only be done with software renderers can now be done completely by the graphics hardware. In this paper we present first results of our ongoing work that deals with the visualization of time dependent CT data of the human heart. We compare the drawbacks and benefits of 3D and 2D texture based methods. We also show that a high quality shaded rendering can considerably improve the visual quality. Then we present our results in the area of classification methods, especially using two-dimensional classification. Finally we demonstrate that it is possible to visualize a beating heart using moderately sized time dependent Cardio-CT data.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Picture and Image Generation, Graphics Processors I.3.3 [Computer Graphics]: Picture and Image Generation, Viewing algorithms

1. Introduction

Texture based volume rendering has gained significant importance in the last years. It benefits above all from the growing rasterization power of modern GPUs (Graphics Processing Units). When using 3D textures the volume has to be decomposed into small bricks of data that have to be sorted and composited according to their distance to the camera.

For moderate sized datasets the bottleneck of the method is usually located in the fragment processor of the GPU. On the other hand, for very large volumes, e.g. time dependent data, the amount of dedicated graphics memory limits the performance. When the size of a volume exceeds the available amount of free graphics memory, texture data has to be fetched via the AGP Port from main memory.

Driven by the programmable vertex and fragment processors of the GPUs, techniques like early ray termination or empty space skipping which originate from software based ray casting algorithms can now be implemented using current graphics hardware.

In this work we present our current results, especially considering time dependent volume data. We achieve efficient empty space skipping by using small bricks with additional storage of the corresponding minimal and maximal data values. This reduces the rasterization load, the memory footprint and the data transfer by culling bricks according to the current transfer function. To further reduce the amount of necessary texture memory, we show that gradient data needed for local illumination can be computed on the fly. We also show that 2D textures are optimal when the data cannot be stored completely within the graphics memory.

2. Previous Work

Texture based volume rendering has been used the first time in 1993 by Cullip and Neumann [CN93]. They developed the ideas of axis- or viewing-plane aligned sampling planes.

In the area of medical imaging, the method was first applied by Cabral et al. in 1994 [BCF94] who demonstrated that interactive volume rendering can be done when 3D texture acceleration is available. In the last years, many works have dealt with the topic of using consumer graphics cards for volume rendering purposes. We can cite only a fraction of them, e.g. Rezk-Salama [RSEB*00], Engel [EKE01], Kniss [KMM*01] [Kni03], Guthe [GRS*02] and Roettger [RGW*03]. The topic of time dependent rendering has been adressed in many works, e.g. Lum [LMC02] or Schneider [SW03]. Krüger and Westermann [KW03b] showed that it is possible on recent graphics hardware to do hardware accelerated ray casting with early ray termination and empty

© The Eurographics Association 2004.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

space leaping. Li et al. [LK03a] [LK03b] also address these topics in their current work.

3. Texture based Volume Visualization

3.1. The 3D Texture Approach

Texture based volume rendering has established itself as a standard method for the visualization of moderate sized volume data. Basically, the algorithm renders semitransparent textured slices that are sorted in back-to-front manner. The slices result from intersecting equidistant planes that are parallel to the viewing plane with the bounding box of the volume. The texture coordinates can be generated automatically by using the glTexGen facility of OpenGL.

The emission and absorption is derived from a transfer function that maps the volume data to color and opacity. Using modern graphics hardware this can be implemented using a small *fragment program*. The inputs of a fragment program can be some user defined values that are constant over a primitive (e.g. quadrangle or triangle), some interpolated color parameters and texture coordinates for every texture and the textures themselves. Figure 1 shows a simple example that does a 3D texture lookup and uses the result to do a post-interpolative transfer function lookup. The presented code is written in Cg (C for Graphics). This example shows how easily the fundamental components of a volume renderer can be implemented using current techniques.

```
0  void main(in float4 texCoord:TEXCOORD0,
1            out float4 color0:COLOR0,
2            uniform in float baseOpacity,
3            uniform sampler3D baseTexture,
4            uniform sampler1D colorTable)
5  {
6    float4 base = tex3D(baseTexture,
7                        texCoord.xyz);
8    color0 = tex1D(colorTable, base.x);
9    color0.a *= baseOpacity;
10 }
```

Figure 1: *A simple pixel shader computing a post-interpolative texture lookup.*

The advantages of the Cg shading language are manifold. It is highly standardized, so it can be used over a wide range of hardware. The shaders are easier to develop and to debug and there is no unpleasant assembly code to cope with. Unfortunately the current Cg compiler often generates suboptimal code that has to be improved manually.

3.2. Volume Rendering using 2D Textures

Recently 2D texture based volume rendering has gained a lot of importance. Since it only allows to render slices parallel to the axes of the volume, three orthogonal stacks of slices are used. A stack of slices is selected if it is most perpendicular to the current viewing vector.

The 2D texture based approach generally delivers higher performance compared to 3D textures. We achieve a higher fill rate even when doing trilinear interpolation using multitextures. The main reason seems to be that the drivers are extremely optimized. This fact also explains the better AGP transfer rates we experienced for datasets that do not fit entirely into the graphics memory. In our tests with time dependent data we found out that the performance of transferring 3D texture bricks via AGP is about a factor of 10 lower compared to 2D texture slices containing the same amount of data. As a consequence, 2D textures should always be used for large datasets.

3.3. Shading Methods

Adding shading to volume rendering algorithms has often been used to improve image quality. We are using a simple local model here. Former works [RSEB*00] stored the precomputed gradients in the *RGB* channels of an additional texture and computed the dot product using a small pixel shader which increased the memory usage by a factor of 5, because *RGB* textures internally use four channels. Since modern fragment programs are able to do dependent texture lookups using arbitrary inputs, we are now able to put the gradient into *RGB* and the data into *A* of an *RGBA* texture. Then, we can use the *A* channel as input of the transfer function via a dependent texture lookup.

To save memory, an alternative to precomputed gradients is to compute the gradient directly within a fragment program. For this, the 3D texture bricks have to overlap by two voxels instead of one. For a forward difference scheme, the fragment program then does additional three texture lookups to get the values of neighboring voxels. After normalizing the gradient vector the local illumination is computed. Unfortunately, the performance of this approach is not as high as we expected. The approach using precomputed gradients is about a factor of four faster. The Cg code for this approach is shown in figure 2.

For 2D textures the same methods can be applied. To compute gradients on the fly within the fragment pipeline, the neighboring slices have to be used as additional parameters to the fragment program. For trilinear interpolation of the data itself we need texture slice i and $i+1$. In addition, we need the slices $i-1$ and $i+2$ for computing the gradients, for example by a central difference operator.

4. Classification Methods

4.1. Classical Transfer Functions

Usually a 1D transfer function is used to map gray values to color and opacity which can be implemented as a lookup

© The Eurographics Association 2004.

```
0  void main(in float4 texCoord:TEXCOORD0,
1      out float4 colorO : COLOR0,
2      uniform sampler3D BaseTexture,
3      uniform sampler1D colorTable,
4      uniform float3 voxelSize,
5      uniform float3 lightVector)
6  {
7    float4 base = tex3D(BaseTexture,
8                        texCoord.xyz);
9    colorO = tex1D(colorTable, base.x);
10
11   float bx = tex3D(BaseTexture,
12               texCoord.xyz +
13               float3(voxelSize.x,0,0)).x;
14   float by = tex3D(BaseTexture,
15               texCoord.xyz +
16               float3(0,voxelSize.y,0)).x;
17   float bz = tex3D(BaseTexture,
18               texCoord.xyz +
19               float3(0,0,voxelSize.z)).x;
20
21   float3 gradient = float3(bx,by,bz) -
22                     base.xxx;
23   gradient = normalize(gradient);
24
25   float ndotl = max(dot(gradient,
26                         lightVector),0);
27
28   colorO.rgb += ndotl * float3(1,1,1);
29 }
```

Figure 2: *A more complex pixel shader computing gradients and local diffuse illumination.*

table. We use a 1D-dependent texture within a fragment program (see figure 1) where the tri-linearly interpolated gray values are used as texture coordinates.

Transfer functions are usually edited by drawing curves for the R, G, B and A channels. Instead, we propose to always use the *HSVA* color space for their manipulation which is much more intuitive. Before uploading the 1D-Texture containing the transfer function, we convert the *HSVA*-values to *RGBA* of course, since the hardware does not directly support *HSVA*.

When necessary it is possible on current hardware to use 12 bit precision or more for the transfer function. This can be achieved by using the texture format `GL_LUMINANCE_16`. Since the texture dimension is usually limited to 2048 or 4096 for a 1D texture we have to map the 16 bit values into corresponding (s,t) texture coordinates for a 2D texture lookup containing the 1D transfer function.

4.2. Two-dimensional Transfer Functions

As stated earlier, there are more powerful methods for classification than simple 1D transfer functions. Using 2D transfer functions is a rather canonical extension of the 1D case. For every voxel we store the scalar data value and an additional value that is either derived by the data itself or could even be another volume of other modality. Usually the gradient's magnitude is computed and used as the second parameter of the 2D transfer function. In the corresponding fragment program the two data channels are used directly as a pair of texture coordinates for a lookup into a 2D *RGBA* texture.

There are multiple advantages of two-dimensional classification. By nature it is more powerful, since its domain is two-dimensional. As a consequence, we can distinguish identical data values during classification when their gradient magnitude differs. Gradient information is especially useful for rendering material boundaries, since the gradient is the canonical means of detecting edges. Conversely, we could display or hide all the parts of the volume that have a homogeneous data value, depending on the aim of the visualization.

One of the crucial points of 2D transfer functions is their manipulation. It should be intuitive, but the degrees of freedom should not be reduced too much. The most promising approach to the problem seems to be the method of Kniss [KMM*01] who has implemented a very remarkable editor for the manipulation of 2D transfer functions and applied it very successfully, e.g. for medical applications.

4.3. Manipulation of 2D Transfer Functions

Our implementation of a 2D transfer function editor was largely influenced by ideas of J. Kniss. The user places bounding boxes of different shapes in the domain of the transfer function. These boxes can contain any kind of color and alpha gradient. The color gradients are defined by two inputs editable by respective sliders controlling hue, saturation, value and opacity. The interpolation of the color is done within the HSV model.

We have found using the HSV model very intuitive. For example, the user can easily get feedback by simply setting up a hue gradient over the complete domain of the transfer function. In the resulting image he can quickly spot the color of the interesting part and put another shape onto this part of the transfer function's domain.

In order to have a useful default setting of the used color gradient, we decided to use a full hue cycle combined with full saturation and a value-gradient ranging from 0 to 1. The latter enables a built-in faux shading by surrounding each feature by a thin dark "aura" and nicely enhances the contours of features. The edges of a shape's bounding box can be picked and dragged with the mouse to change the area they cover. The edges between two adjacent boxes can also be simultaneously moved which enables to place a set of boxes next to each other and manipulate the positions of the respective boundaries.

The basic built in shapes are tensor products of a ramp,

© The Eurographics Association 2004.

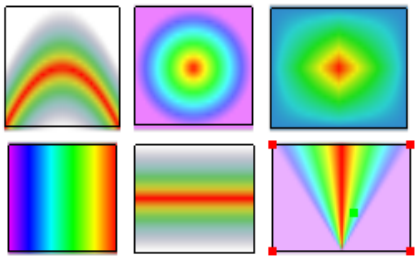

Figure 3: *2D transfer functions are constructed by combining different predefined shapes. Some examples are shown here, e.g. a parabola, a radial gradient, three tensor product based shapes and a triangular shape.*

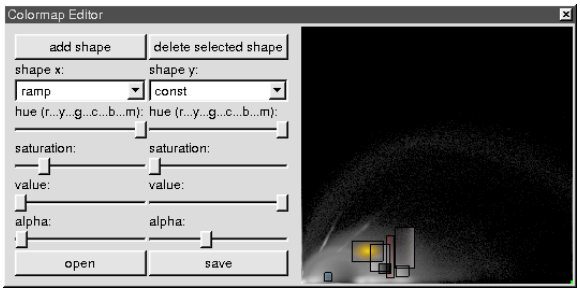

Figure 4: *The manipulation of 2D Transfer functions is done via an editor show here. In the background a 2D histogram of a CT dataset is shown. It contains typical parabola-shapes which arise from boundaries with partial volume effects between different tissues.*

hat, exponential, logarithm etc. Some examples are given in figure 3. In addition there is a radial gradient, a triangle-shape and a parabola-shaped gradient, inspired by my colleague Fernando Vega. Any other shape can easily be added to the set of built in shapes.

In order to find the right position of the shapes a 2D histogram is displayed on the domain of the transfer function. A snapshot of the current version of the transfer function editor is shown in figure 4.

5. Visualization of a beating Heart

In this section the described methods are applied and their advantages are shown. As an example we have chosen time dependent Cardio-CT data that have been produced at the institute of medical physics located in Erlangen.

The basic assumption of our approach is that we have the complete sequence in-core. The second assumption we made is that the data can be fetched fast enough over the the AGP port from main memory into graphics memory. This approach will work even better when the upcoming PCI-Express standard will be available. Conversely, a rendering of out-of-core data would only be possible with rather small datasets for each time step.

The first approach for time dependent data was to use 3D textures. We first preferred them to 2D textures since the memory consumption is by a factor of three lower. We considered the lower fill rate not to be harmful for this application since it would not be the bottleneck. Unfortunately we had to realize that 3D textures have a very bad behavior during AGP texturing. So, they are not suited for time dependent data that don't fit entirely into the graphics memory. In consequence, we make use of 2D textures to get optimal performance. The explanation seems to be a non optimal texture management for 3D texture objects of the current graphics drivers.

Throughout our experiments the relevant data values were located in rather small subintervals of the data domain. As a consequence, we propose the spatial resolution should not always be preferred to intensity resolution. For example a lower spatial resolution should be considered to be able to upload $16-$bit data instead of lossy conversion to 8 bit. When a conversion is necessary, the 16-bit data are converted to 8 bits per voxel after having found an optimal transfer function. This is done by quantizing the data based on the information delivered by the chosen transfer function.

Using an NVIDIA FX5900 graphics card we have measured an AGP texture transfer rate of 245 MB/s which corresponds to 15.3 time steps per second each containing 16 MB. Using an ATI Radeon 9800XT we are able to transfer 736 MB/s corresponding to 46 time steps per second. This shows that a real time playback is possible on the ATI hardware.

6. Results

In this section we present current results of the described algorithms. We here present renderings of Cardio-CT data that were generated by a ECG-based reconstruction of a heart patient. The ASSRCI algorithm (Advanced Single Slice Rebinning Cardio Interpolation) has been used to reconstruct 5% steps relative to the R-peak of the ECG and the Siemens "B35f" convolution kernel was used. The data were acquired with a Siemens Sensation 16 scanner where the gantry has a rotation time of 0.42 seconds. A complete scan took 19.65 seconds. Unfortunately, the data are rather noisy, since only a range of 180° has been used for the reconstruction.

In figure 5 (left) we present a volume rendered image of a heart using phong illumination. The advantage of a shaded display can be clearly seen, for example at the coronary arteries. During our experiments with the Cardio-CT data, we found that using 1D transfer functions only allows to visualize the surface of the heart. When trying to reveal inner structures by using high transparency, the 2D transfer

© The Eurographics Association 2004.

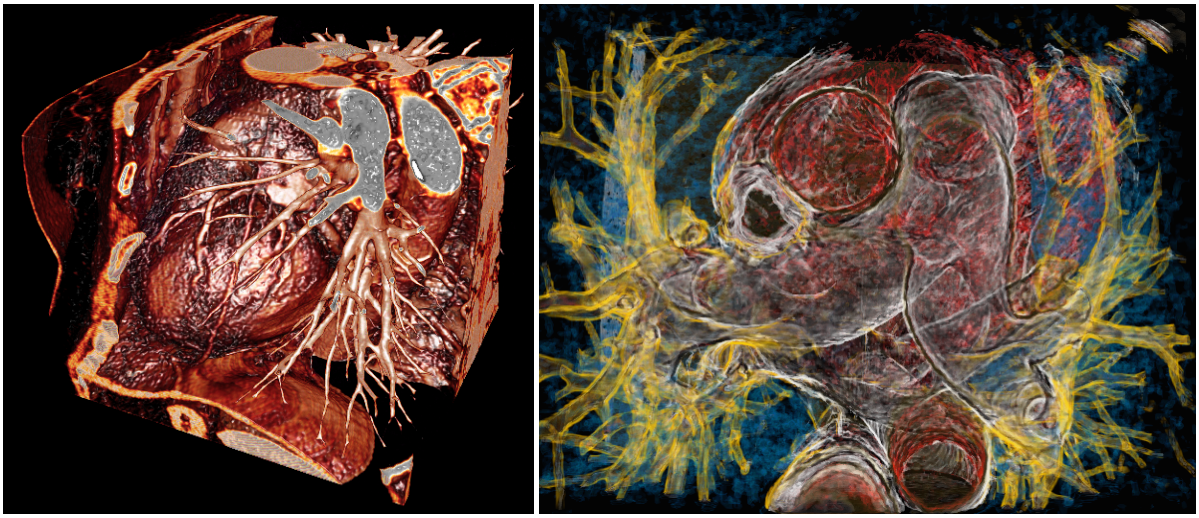

Figure 5: *Left: High quality shading is very important for Cardio-CT data. Right: Inner structures of the heart can only be revealed with 2D classification.*

functions show a significant advantage. The result using 2D transfer functions can be seen in figure 5 (right) where areas of low gradient were made transparent. (The images can also be found in the color section.)

texture type	transfer function	shading	frames per second
3D	1D	n	15
3D	2D	n	15
3D	1D	y	1.4
3D	2D	y	1.7
2D	1D	n	63
2D	2D	n	64
2D	1D	y	5.3
2D	2D	y	5.2

Table 1: *Performance of the implemented rendering methods. All measurements were done with a volume of* 256^3 *cubic voxels with 8 bit precision per voxel using a* 512^2 *viewport. The data have been sampled with a slice distance of one voxel and shading was done on the fly with a forward difference scheme for gradient estimation. The graphics board was an NVIDIA GeForce FX 5900 ultra.*

7. Conclusions

We have shown that high quality visualization of time dependent data is possible on modern PC hardware. All the implementations have been done on a Linux PC with an NVIDIA GeforceFX 5900 graphics card but also do work on current graphics cards of other manufacturers. Even for moderately sized time dependent data, e.g. 20 time steps of 16 MB we have achieved a real time playback. Our tests with a ATI Radeon 9800XT have shown that over 700 MB/s can be transferred to the graphics card.

To achieve an optimal image quality we first showed that high quality shading can be done by on the fly computation of the gradients. High quality shading is very useful for CT data of the heart, since the vessel structures are nicely emphasized. Our work with 16 bit data has shown that higher precision is often useful for medical data, and even the quality of the shading is significantly improved when using higher precision.

During our experiments we found out that 1D classification does not allow a good visualization of the inner structures of the heart. The solution for this problem was to use 2D transfer functions. They allow to hide the homogeneous parts of a volume and reveal features that allow a better interpretation of inner structures of the data. For a more powerful classification they should often be considered since the results are usually superior to a 1D classification.

8. Future Work

The first task will be to initiate a closer cooperation with cardiologists. The clinical evaluation will be a very important part of this project. The presented methods are currently under development and there are many ideas to realize for the future. One of the most interesting parts will be to implement volumetric compression methods that both reduce the amount of data and unfortunately could also negatively affect the rendering quality and performance. In the medical field, lossy compression methods are usually not accepted,

© The Eurographics Association 2004.

since no relevant information should be lost. On the other hand, lossless compression does not bring high compression rates and possibly decreases the rendering performance even more.

Another task we are currently working on is to implement a pre-integrated volume renderer that can cope with 2D transfer functions, since pre-integration often gives better results than slice based volume rendering. Furthermore, we will implement a more intuitive version of the 2D transfer function editor. We also have high expectations to the upcoming PCI-Express standard, since the limited AGP performance is one of the main bottlenecks of the presented approach.

Acknowledgments

The research was supported by the German Science Foundation DFG, which funds the Graduate Research Center 244 3D Image Analysis and Synthesis. Many thanks to Dirk-Alexander Sennst and the Institute for Medical Physics headed by Prof. Kalender. They provided many datasets and helped with fruitful discussions and ideas.

References

[BCF94] B. CABRAL N. C., FORAN J.: Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings ACM Symposium on Volume Visualization* (1994). 1

[CN93] CULLIP T., NEUMANN U.: Accelerating volume reconstruction with 3d texture mapping hardware. In *Technical Report TR93-027, Department of Computer Science at the University of North Carolina, Chapel Hill* (1993). 1

[EKE01] ENGEL K., KRAUS M., ERTL T.: High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In *Eurographics / SIGGRAPH Workshop on Graphics Hardware '01* (2001), Annual Conference Series, Addison-Wesley Publishing Company, Inc., pp. 9–16. 1

[GRS*02] GUTHE S., ROETTGER S., SCHIEBER A., STRASSER W., ERTL T.: High-quality unstructured volume rendering on the pc platform. In *ACM Siggraph/Eurographics Hardware Workshop* (2002). 1

[KMM*01] KNISS J., MCCORMICK P., MCPHERSON A., AHRENS J., PAINTER J., KEAHEY A., HANSEN C.: Interactive texture-based volume rendering for large data sets. *IEEE Computer Graphics and Applications 21*, 4 (/2001), 52–61. 1, 3

[Kni03] KNISS J. M.: Gaussian transfer functions for multi-field volume visualization. In *Proceedings IEEE Visualization 2003* (2003). 1

[KW03a] KRÜGER J., WESTERMANN R.: Linear algebra operators for gpu implementation of numerical algorithms. In *Proc. SIGGRAPH* (New York, NY, USA, July 2003), vol. 22 of *Annual Conference Series*, ACM Press, pp. 908 – 916.

[KW03b] KRUEGER J., WESTERMANN R.: Acceleration techniques for gpu-based volume rendering. In *Proceedings IEEE Visualization 2003* (2003). 1

[LK03a] LI W., KAUFMAN A.: Empty-space skipping and occlusion clipping for texture-based volume rendering. In *Proceedings IEEE Visualization 2003* (2003). 2

[LK03b] LI W., KAUFMAN A.: Texture partitioning and packing for accelerating texture-based volume rendering. In *Graphics Interface 2003* (2003). 2

[LMC02] LUM E. B., MA K.-L., CLYNE J.: A hardware-assisted scalable solution for interactive volume rendering of time-varying data. *Transactions on Visualization and Computer Graphics 8*, 3 (July/Spetember 2002), 286–301. 1

[MHS99] MEISSNER M., HOFFMANN U., STRASSER W.: Enabling Classification and Shading for 3D Texture Mapping based Volume Rendering using OpenGL and Extensions. In *IEEE Visualization '99 Proc.* (Oct. 1999).

[RGW*03] ROETTGER S., GUTHE S., WEISKOPF D., ERTL T., STRASSER W.: Smart hardware-accelerated volume rendering. In *Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization* (2003). 1

[RSEB*00] REZK-SALAMA C., ENGEL K., BAUER M., GREINER G., ERTL T.: Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage-Rasterization. In *Eurographics / SIGGRAPH Workshop on Graphics Hardware '00* (2000), Addison-Wesley Publishing Company, Inc., pp. 109–118,147. 1, 2

[SW03] SCHNEIDER J., WESTERMANN R.: Compression domain volume rendering. In *Proceedings IEEE Visualization 2003* (2003). 1

© The Eurographics Association 2004.