Texture Based Volume Rendering of Hexagonal Data Sets

Niklas Röber *
Otto-von-Guericke Universität

Markus Hadwiger †
VRVis Research

Alireza Entezari ‡
Simon-Fraser University

Torsten Möller §
Simon-Fraser University

Graphics, Usability, and Visualization (GrUVi) Lab Simon-Fraser University

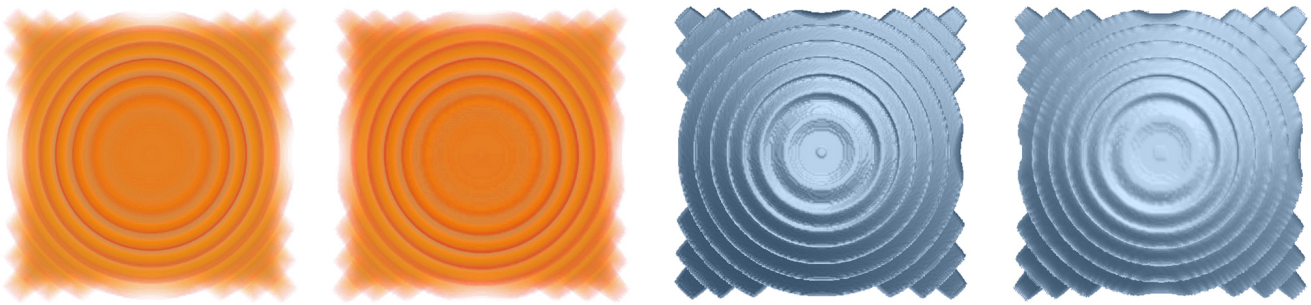

Figure 1: Marschner-Lobb (150^3) with post-classification and shading rendered as Cartesian and BCC (left and right image)

ABSTRACT

We present a new method for rendering hexagonal based data sets using common available graphics hardware. Our approach combines the efficiency of hexagonal sampling with hardware accelerated, texture based volume rendering techniques. Pre- and post-classification as well as simple shading are possible within one rendering pass. Our methods are applicable for both, static and time-varying data sets.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: hexagonal data, texture based volume rendering, pre/post-classification, shading, 3D, 4D

1 INTRODUCTION

When discretizing data either by a physical acquisition process (e.g. medical imaging such as Computed Tomography or Magnetic Resonance Imaging and others) or by numerical techniques (e.g. Computational Fluid Dynamics), we typically assume, that there is no principal axis in the frequency spectrum of the data. In other words, there is no information on weather a certain direction is preferable for the sampling and discretization process. Hence the frequency domain can be considered spherically bandlimited. During the sampling process the main spectrum of the signal is in effect replicated on a new lattice in the frequency domain [20]. A lattice is described by a matrix called the *sampling matrix*. For so-called *regular sampling* the sampling matrix is simply formed by the basis vectors of the sampling operation as its columns. If the sampling matrix of a sampling scheme is a diagonal matrix, that scheme results in the defacto standard of sampling on a Cartesian lattice. However, it has been shown, that the most efficient lattice is the one characterized as hexagonal [4].

While most of the applications of sampling theory on regular lattices have been focused on the Cartesian lattice, a mainstream use of hexagonally sampled data depends on the availability of the appropriate tools and algorithms for its manipulation and rendering. The major contribution of this paper is to enhance the suite of tools available for the rendering of a given hexagonal lattice. The focus of recent research on optimal sampling structures has been on the so-called BCC lattice (one of many hexagonal lattice structures in 3D) due to its simplicity [19] [3] [14].

One of the most promising rendering algorithms to this date is texture slicing. This is the fastest rendering algorithm and with the recent work by Engel et al [5] it has obtained further popularity due to its high-quality results.

Assuming that we are given hexagonally sampled data on a BCC lattice, we introduce an efficient rendering algorithm by using texture based graphics hardware. In particular, we develop a technique which allows one to visualize a shaded three-dimensional (3D) BCC lattice and time-varying 3D data (sampled on a so called $D4^*$ lattice [4] [14]) in a single rendering pass.

While the theory shows that hexagonally sampled data (although being represented with fewer number of samples) retains the identical information as comparable Cartesian lattices, we will investigate how well our rendering algorithm for hexagonally sampled data competes with the traditional algorithm operating on an equivalent Cartesian data set. We will show that the performance is comparable and at times even improved. Additionally, an advantage of data sets sampled on the BCC lattice is that one half of the data can be easily taken out (without any additional subsampling) and rendered with good quality as a preview image.

The structure of the paper is as follows. After the discussion of related and similar work in section 2 we will first describe the idea of hexagonal sampling for three- and four-dimensional data sets in section 3. In section 4 we explain our new algorithm on how to use commodity graphics hardware to visualize data sampled on BCC and $D4^*$ lattices. Section 5 presents qualitative and quantitative results and compares our achievements with respect to the Cartesian lattice. Finally, section 6 summarizes the paper, draws conclusions

*nroeber@cs.uni-magdeburg.de
†markus@vrvis.at
‡aentezar@cs.sfu.ca
§torsten@cs.sfu.ca

and presents possible ways for future investigations.

2 PREVIOUS WORK

A major challenge for Volume Graphics is the large amount of data that needs to be processed in order to visualize the data. Research in the past 15 years has focused to solve this problem. One successful avenue that has been pursued is the development of dedicated hardware specialized in volume rendering. The early architecture of SGI [1] has been used by Cabral et al [2] for interactive volume rendering using a 3D texture buffer. While this is bending the purpose of the hardware that has been originally designed to do surface graphics really well, special hardware for volume rendering has been developed [15]. As this hardware can be used to create stunning results, its disadvantage is the price.

With the advent of the line of commodity graphics hardware by *n*Vidia and ATi Cabral's algorithm [2] has been adapted and evolved [16]. One drawback of texture based volume rendering algorithms has been their poor quality [10]. With the availability of cheap, programmable graphics hardware the renditions have seen a huge improvement. Post-classification and shading got possible with great results at interactive rates [13] [16] [12]. Hadwiger et al [7] have used multi-texturing in order to implement tri-cubic filtering during the sampling of the 3D texture to increase the accuracy of the renditions. The new line of graphics boards offer full 128bit pipelines, opposed to the 32bit on current hardware, and hence we should see huge improvements in the rendering quality in the coming month. Engel et al [5] introduced an algorithm, that uses depended textures to store the result of an analytic integration between two rendering slices which can be used to create high-quality renditions.

Optimal sampling has recently received a lot of attention in the scientific community. Ibáñez [8] adapted a ray casting algorithm to visualize medical data sets. Later Theußl et al [19] discussed the use of the Body Centered Cubic (BCC) lattice for splatting. They illustrate the efficiency of this sampling lattice by comparing the quality and the performance of images rendered from data sampled on the Cartesian lattice with resampled data on the BCC lattice. Müller et al later described the use of the BCC lattice for Shear-Warp volume rendering [18]. Additionally they extended splatting for the time-varying hexagonal data sets with great results [14]. Carr [3] developed techniques for an efficient iso-surfaces generation for the BCC lattice and demonstrated that his results were comparable to the Cartesian lattice.

3 OPTIMAL SAMPLING

As hexagonal sampling has been discussed throughout the literature, we will give a brief introduction with the focus on the later visualization using texture based hardware.

Sampling a signal corresponds to replicating the spectra in the frequency domain. In order to avoid artifacts, these aliased spectra have to be far enough from each other to avoid overlapping in between. From correspondence between spatial and frequency domain we know that moving spectra further apart in the frequency domain results in a decrease of the sampling distance in the spatial domain. To move the samples in the spatial domain further apart, i.e. to decrease the sampling distance, we have to pack the spectra in the frequency space as dense as possible, under the assumption that the spectra of the signal are spherically bandlimited. This refers to the sphere packing problem [17]. As requirement for hexagonal sampling we assume an isotropic and hyperspherically bandlimited signal. This guarantees that the frequency responses are restricted to hyperspheres.

A signal can be sampled in an unlimited number of different ways. Due to simplicity, the most often sampling matrix used is the Cartesian lattice. However, the Cartesian lattice is not the most efficient lattice in terms of the Shannon theorem. In 3D several different hexagonal sampling grids exist which have a similar packing density, all better than the Cartesian lattice. However, only a few can be described as a lattice. In this paper for static volumetric data sets the BCC lattice is used and for the time-varying case the $D4^*$ lattice [4].

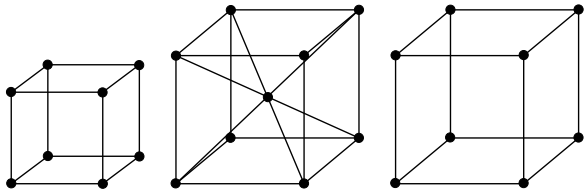

Figure 2: Comparison of lattices (CC, BCC, $D4^*$)

Figure 2 shows a direct comparison in size and topology of the different lattices used. As can be seen, the main differences between the Cartesian and the BCC lattice are the additional sampling point in the centre of the cell and a by $\sqrt{2}$ increased sampling distance. For the time-varying hexagonal lattice, only one cell of a time frame is shown.

Generally, for the n-dimensional case, the hexagonal lattice can be reconstructed by taking the $n-1$ Cartesian lattice and by offsetting it along the n-th dimension. A hexagonal lattice can also be described as being composed out of two interleaving n-dimensional Cartesian grids which are offset by 0.5 in all n dimensions.

For the one-dimensional case, the Cartesian and the hexagonal lattice are identical. However, the efficiency of the Cartesian over the hexagonal lattice decreases with increasing dimensionality. A two-dimensional hexagonal lattice already needs only 0.86 of the samples which are necessary for the appendant Cartesian lattice (3D 0.707, 4D 0.5).

3.1 3D

For static volumetric data sets the BCC lattice is used. As can be seen from Figure 3 the BCC lattice can be thought of being composed out of two interleaving Cartesian lattices. These two grids have an increased sampling distance in x, y and z by $\sqrt{2}$. The corner of each cell describes the centre of a cell in the other lattice, refer Figure 3.

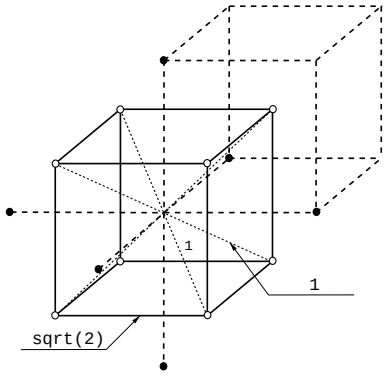

Figure 3: BCC lattice

Here for simplicity only two cells are visualized. With an increased number of samples along z and a decreased number of sam-

ples along x and y, this results in a total number of $1/\sqrt{2}$ samples, or roughly 30 percent less samples. The quality that the BCC lattice can be decomposed into two Cartesian lattices is exploited later for rendering such hexagonal data sets using multi-texturing.

Gradients have to be computed in order to perform volumetric shading. The gradient information can be determined by using central differences. A sampling point in the BCC lattice has eight closest neighbours (weight) and six additional neighbours along the axes (black), Figure 3. The distance to the eight closest neighbours is 1.0, while the distance to the next samples along the axes is $\sqrt{2}$. The simplest way is to only consider the six axes aligned neighbours and compute the gradient information using straightforward central differencing. Additionally, the eight closest neighbours can be used to refine the gradient information. This can be achieved in two ways. The first one is to compute the central differences from these eight neighbours along the axes where the results are linearly interpolated and added to the existing gradient. Alternatively, four additional non-axis aligned central differences can be determined and added to the existing gradient. Care has to be taken to consider the different sampling distances between the eight and the six neighbours.

To enhance the interactivity with the visualization for preview or Level-of-Detail purposes, one of the two textures can be easily taken out without resampling and rendered alone without blending with the second BCC texture. Here the opacity transfer function has to be adjusted as only half the number of samples are used. This *half BCC lattice* can be easily rendered using existing Cartesian based volume rendering systems. Even though, only minor details are missing, to visualize the complete signal both textures have to be used.

3.2 4D

The interaction with and the rendering of time-varying data sets from numerical simulations or medical acquisition processes is a challenging task to this date. The optimality of hexagonal sampling comes in very handy since using a four-dimensional hexagonal lattice ($D4^*$) we save 50% of the data.

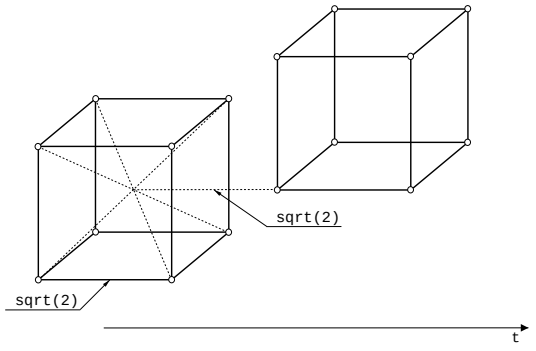

Figure 4: $D4^*$ lattice [14]

As for the BCC lattice, the $D4^*$ lattice can be thought of being build by several $n-1$ base lattices which are offset along the n-th dimension. Or in other words, several 3D Cartesian grids which are offset along the time axis by 0.5 in x, y and z. The sampling distance along t is increased by $\sqrt{2}$ and decreased by the same amount in all other dimensions, refer Figures 2 and 4. This results in a total amount of 50 percent less samples than the Cartesian lattice. Figure 4 shows how the $D4^*$ lattice is offset over t. Here two cells of two neighbouring time frames are displayed. These time frames are offset by $1/\sqrt{2}$ along t. Additionally, there are also offset by 0.5 in x, y and z which has to be considered for the rendering to accommodate for the lattice topology.

To load and render a time frame, three different methods are applicable. The simplest way is to load and visualize only one $D4^*$ lattice time frame. Each time frame has $1/\sqrt{2}^3$ less samples than the appendant Cartesian time frame. To increase the quality, two, or correctly three, time frames can be used additionally and rendered as a BCC volume. Here the second texture results from either loading the previous or the next time frame, or is determined by the interpolation of both of them.

Gradient information for shading the data set can be computed straightforwardly. If the data set is rendered using one time frame only, simple central differences are applicable. If the time frames are rendered as BCC volume, the gradient information can be computed by using the methods discussed in the last section.

4 RENDERING

The adaptation of existing texture based volume rendering techniques to the BCC lattice is straightforward and will be discussed in detail in this section. For the implementation we are targeting on the widely available graphics accelerators from *n*Vidia. The rendering application was developed under Linux and the graphics API used was OpenGL.

Unfortunately, current OpenGL implementations do not support hexagonal textures in hardware. An advantage of the BCC lattice is, that it can be decomposed into two interleaving cubic Cartesian grids which are offset by 0.5 in all three dimensions in the spatial domain. In order to render the BCC volume correctly, one texture has to be translated by $1/2 \cdot tex\_size$ in x, y and z. These two textures can be loaded separately into two texture units. Each texture can now be sliced using a standard texture based volume renderer and the result from both texture units is blended together using register combiners in a final compositing step.

As discussed in the last section, time-varying optimal sampling results in two Cartesian based volumes which are offset in the same way as the two textures in the BCC lattice. However, these time frames are also offset over time which needs to be considered for the rendering. Each time frame can be rendered independently from each other using existing texture based volume rendering techniques [14]. Care has to be taken to adjust the texture coordinates by 0.5 in x, y and z for each odd time frame. Two, or correctly three, time frames can be used to render the data set as a BCC volume. Here the two additional time frames ($t-0.5$ and $t+0.5$) have to be interpolated in order to be at the same point in time as t. Alternatively, if the changes over time are minor, just one additional time frame is used and the interpolation over time is discarded.

In this section we explain how static and time-varying hexagonal data sets can be rendered using pre- or post-classification and simple shading.

Different blending schemes to achieve either direct volume rendering, non-polygonal iso-surfaces, MIP, or X-Ray renditions can be applied straightforwardly and are the same as for Cartesian based rendering systems.

Some characteristics of hexagonal lattices can be exploited to increase the interactivity with BCC and $D4^*$ based data sets. For instance, both data sets can be rendered using two interwoven textures to increase the quality. But they can also be rendered by only using one texture for Level-of-Detail or preview purposes to increase the interactivity.

In time-varying data sets, the selection of a specific time frame can be very tedious, especially for large data sets as when changing the current time frame the appropriate volume has to be loaded and rendered. Here, a simple yet very efficient approach can be used to interpolate between several time frames in hardware and after the

selection of the demanded time frame is made load and render this one only.

In general, the rendering of BCC and $D4^*$ data sets is more efficient than for the appendant Cartesian based data sets. However, due to the OpenGL requirement of textures being the size of 2^n, some hexagonal data sets (as texture) are larger than the Cartesian ones, which can result in an increase in the rendering time. Section 5 shows some examples and explains the problem in more detail.

4.1 Classification

All data sets, rendered either as BCC volume or as Cartesian volume can be classified using pre- or post-classification in a single rendering pass.

For pre-classification generally a luminance volume is loaded into the texture memory together with a colour lookup table. For rendering BCC based data sets, simply two luminance textures are loaded into texture units 0 and 1. The second texture has to be offset by $1/2 \cdot tex\_size$ in x, y and z to accommodate for the BCC lattice topology, compare with Figure 3. Now both textures are sampled at view-aligned quads and the result of both texture units is blended together using register combiners and output to the screen. Figure 5 shows the rendering pipeline used for pre-classification.

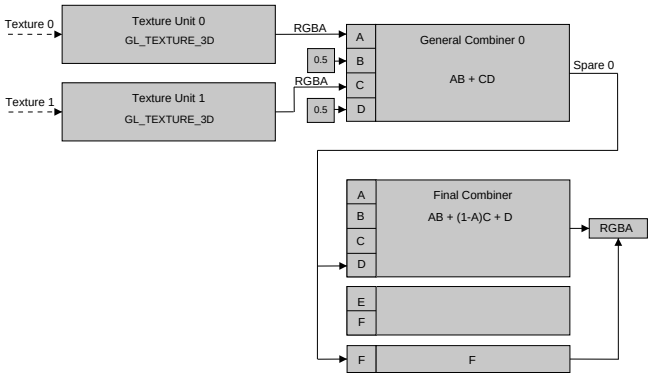

Figure 5: Pre-classification for BCC rendering

Post-classification [13] can also be adapted straightforwardly. As for pre-classification, BCC data sets are loaded into texture memory as luminance volumes and assigned to either texture unit 0 or 1. The colour and opacity lookup table is loaded as dependent texture into the texture units 2 and 3. Now both luminance textures are sampled at view-aligned quads and the result is used for a dependent texture lookup in the texture units 2 and 3. Finally, the output is blended in the same way as for pre-classification. The pipeline can be seen in Figure 6.

However, even though we are rendering with less samples, the frame rate can be lower for BCC volumes with post-classification than for the rendering of cubic Cartesian data sets. This is due to the additional use of texture units 2 and 3.

For the rendering of $D4^*$ data sets, or when only one BCC texture is used, pre- and post-classification can be accomplished as usual.

4.2 Shading

Shading is very important for volume rendered images as it uses additional visual cues which improves the perception of the data. Several shading techniques have been proposed for texture based volume rendering [13] [12]. In this section we are demonstrating how simple shading using texture shaders and register combiners

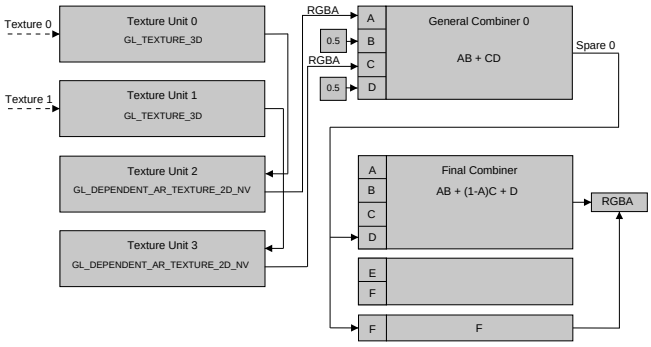

Figure 6: Post-classification for BCC rendering

can be accomplished for hexagonal data sets in a single rendering pass.

The gradient information can be pre-computed using one of the techniques which were discussed in section 3. The luminance volume is loaded together with the gradients as RGBA volume where the gradient information is stored in the RGB part and the data as luminance volume in the A section. For BCC rendering, two data sets are loaded into either texture unit 0 or 1. The texture is sampled as usual at view-aligned quads and the output of both texture units is shaded using register combiners. The final result is blended together in the final compositing step. The light colour and direction can be changed using the OpenGL constant colours. Figure 7 visualizes the rendering pipeline.

For the rendering of $D4^*$ data sets, or when only one BCC texture is used, shading can be performed as usual for cubic Cartesian data sets.

4.3 Hardware based interpolation of time frames

The selection of a specific time frame in a time-varying volumetric data set can be very tedious. Especially for large data sets, this can be very time consuming as every frame has to be loaded and rendered while seeking for specific features. Additionally, we are proposing a simple yet efficient technique which allows one to navigate easily through many time frames at interactive rates.

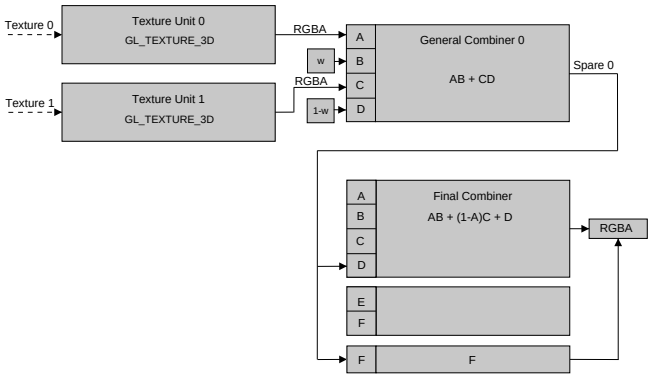

Figure 8: Hardware based blending of time frames

In a pre-processing step every 5th or 10th time frame is downsampled and stored in an additional 4D matrix. Usually 64^3 volumes are sufficient enough to detect most features while quickly skimming through the data. From a time-varying data set with 100 time steps, this results in 20 volumes with an entire size of just

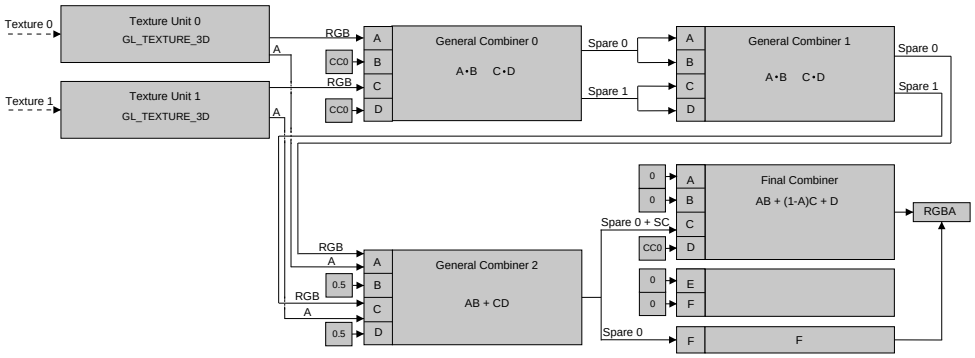

Figure 7: Shading for BCC rendering

about 5 MB. These volumes can be loaded resident into the texture memory. When searching for a specific time frame, these volumes are interpolated to match the current point in time for visualizing a preview volume. If the demanded time frame is selected, the full resolution volume is loaded from disk or memory and rendered as usual.

As the small preview volumes stay resident in texture memory they can be used immediately without reloading. Figure 8 visualizes the pipeline for hardware based blending of time frames. This pipeline is similar to the ones used for blending BCC volumes except that two different weights are applied to interpolate the two textures.

As this feature is used for preview purposes only, no BCC rendering is necessary, but can be implemented as well.

5 RESULTS

This section is used to present and discuss the results achieved. The section is divided into two parts, one focuses more on the visual quality of the rendering, while the other one examines the quantitative results.

The system which was used to render the images and benchmarks is a single processor Athlon 600 with 256 MB of Ram and a GeForce4 Ti 4200 graphics accelerator running Linux. The images were obtained in a 512×512 visual.

5.1 Quantitative Results

First, quantitative results like data size or rendering performance are discussed. Seven different data sets have been used to compare the quality and the rendering performance of hexagonally sampled data sets with cubic Cartesian ones. The data sets used along their size for the Cartesian and the hexagonal lattice can be found in Table 1. The first five data sets are static, while the last two vary over time.

Data Set	Data Size CC	Data Size Hex
Lobster	$301 \times 324 \times 56$	$213 \times 229 \times 79$
Static ML	$41 \times 41 \times 41$	$29 \times 29 \times 58$
Statue Leg	$341 \times 341 \times 93$	$241 \times 241 \times 131$
UNC Brain	$256 \times 256 \times 145$	$181 \times 181 \times 205$
Engine	$256 \times 256 \times 128$	$181 \times 181 \times 181$
Animated ML	$150 \times 150 \times 150 \times 50$	$106 \times 106 \times 106 \times 70$
Kidney	$90 \times 90 \times 80 \times 64$	$63 \times 63 \times 56 \times 90$

Table 1: Data sets, size and dimensions

As expected, the hexagonal data sets are by a factor of $\sqrt{2}$ for static and 0.5 for time-varying data sets smaller than the appendant Cartesian data sets. However, due to the OpenGL requirement of texture sizes of 2^n, in some cases this can result in bigger textures sizes as the next larger texture has to be chosen. Examples for this are the Engine data, the UNC Brain and the Statue Leg. The Statue Leg misses the next smaller texture size in the z-axis by three voxels. This can result in an increase in the rendering time.

Rendering	Lobster	ML 3D	Statue	unc Brain	Engine
CC Pre	58	54	31	20	27
CC Post	46	34	24	15	18
CC Shad.	56	55	-	19	26
BCC Pre	64	45	23	25	27
BCC Post	42	30	16	16	18
BCC Shad.	62	45	23	25	27
Half BCC Pre	113	81	41	43	50
Half BCC Post	77	56	28	30	32
Half BCC Shad.	112	80	39	45	49

Table 2: Rendering performance 3D (fps)

Lobster, Statue Leg, Engine and the UNC Brain are commonly used sample data sets from volvis.org [11]. Static and Animated ML are a static as well as a time-varying version of the Marschner-Lobb data set [9]. As the original Marschner-Lobb is 41^3 in size, a larger example was chosen for the animated version as well as for the teaser figures on the front page for better comparison, both in quality and quantity. The Kidney data is a time-varying data set from nuclear medicine which visualizes the washout of radioactive pharmaceuticals in the kidneys.

Rendering	Animated ML	Kidney
CC Pre	22	49
CC Post	17	29
CC Shad.	21	50
D4 (BCC) Pre	30	41
D4 (BCC) Post	18	28
D4 (BCC) Shad.	30	41
D4 Pre	53	89
D4 Post	35	50
D4 Shad.	48	85

Table 3: Rendering performance 4D (fps)

All data sets, except Marschner-Lobb, had to be resampled to the hexagonal lattice. For better quality comparison of our rendering algorithm, also the Cartesian data sets were resampled and offset by 0.5 in x, y and z. For the resampling, a tri-cubic interpolation was used. A similar version of the Marschner-Lobb data set was

used to directly create data sets sampled to the Cartesian, BCC or $D4^*$ lattice. The qualitative comparisons can be seen in the next section.

Table 2 compares the rendering performance of the static data sets, while Table 3 shows the performance results for the two time-varying data sets. The rendering performance is measured in frames per second (fps). Each data set was benchmarked using pre- and post-classification as well as shading (first column). Half BCC in Table 2 refers to the performance obtained when only one of the two BCC textures was used for the rendering. Additionally, the time-varying data sets were rendered as BCC volume which are either directly loaded trough an additional time frame, or interpolated over time, Table 3. See also section 3 and 4.

No results were obtained for the shaded Statue Leg, as this data set is too large to be rendered in a single pass on the system used. This would require bricking and a two pass operation and hence would not be a fair comparison with the BCC data set.

Generally, as can be seen in Table 2 and 3 the rendering of BCC and $D4^*$ data sets is more efficient than the rendering of the Cartesian data sets. However, if the resampling results in a larger texture, the rendering of the hexagonal data set can be more time-consuming. Additionally, the render performance decreases slightly with post-classification, due to the additional use of the texture units 2 and 3.

Very important for time-varying volumes is a quick navigation through the entire data set. Especially for larger volumes, this can be a difficult and very time consuming task. Table 4 shows the time which is needed to navigate trough the data and to load another time frame. This includes the time which is needed for loading the next volume and to render it. The next frame number can be arbitrary and does not need to be the next neighbouring time frame.

Rendering	Animated ML	Kidney
CC	≈ 700	≈ 100
D4	≈ 90	≈ 25
D4 (BCC)	≈ 175	≈ 35
D4 (BCC time)	≈ 370	≈ 60
Hardware CC	≈ 8	≈ 8
Hardware D4	≈ 8	≈ 7

Table 4: Frame loading (ms)

The first row shows the results for loading the next volume from the Cartesian lattice. The time is measured in milliseconds (ms). The second row displays the results for the $D4^*$ lattice while the next two following rows load two, respective three time frames and render the data as BCC volume. BCC time loads three time frames and interpolates the second texture from the previous and the following volume in time. The last two rows show the results for interpolating the next time fame in hardware and displaying a lower resolution version for preview.

5.2 Qualitative Results

While the last section focused on quantitative results, this one compares the quality of the renditions between the Cartesian and the hexagonal lattices.

The first part of this section compares the engine data set sampled on the Cartesian and the BCC lattice. It further shows the results when only one half of the BCC data is used for rendering. Figure 18 displays the engine rendered using pre-classification. It clearly shows that the image quality of the BCC rendering is comparable to the one obtained from the Cartesian sampled data set. However, due to the multitexture based rendering, the BCC rendition seems to be a bit smoother.

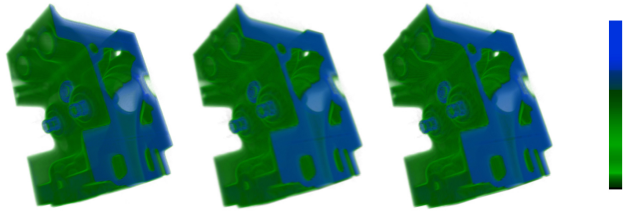

Figure 9: Engine pre-classification (CC, BCC, BCC Half)

The most right image, where only one half of the BCC data set is visualized, misses only minor details. This is an advantage of the BCC lattice and the presented rendering algorithm, which allows a quick change of the resolution and a first Level-of-Detail for faster preview.

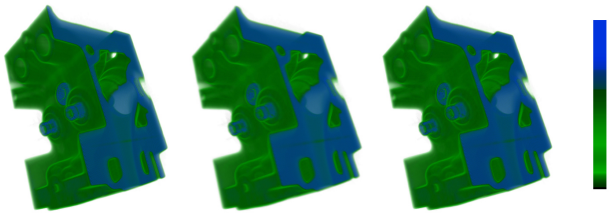

Figure 10: Engine post-classification (CC, BCC, BCC Half)

Figure 19 shows the engine rendered using post-classification, while Figure 20 uses simple shading. Both images exhibit the same qualities as the first one using pre-classification. Especially the shaded visualization shows that the BCC volume is rendered smoother than the Cartesian version. This is due to the final blending step where both parts of the BCC volume are composed together.

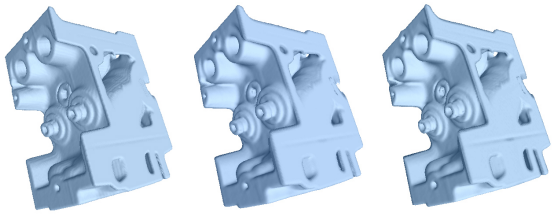

Figure 11: Engine shading (CC, BCC, BCC Half)

Figures 12 and 13 display the Marschner-Lobb data set [9] sampled on the Cartesian and the BCC lattice and rendered using post-classification and shading. Figure 12 compares the rendering quality of the Cartesian sampled data (left image) with the BCC data set (middle and right image).

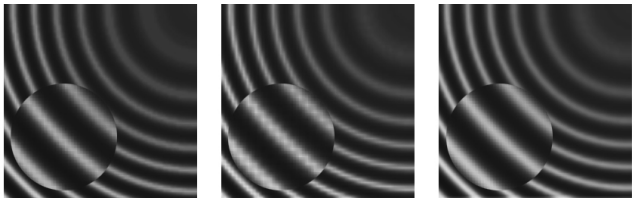

Figure 12: Marschner-Lobb post-classification (CC, BCC Half, BCC)

In these images only a small part of the data is rendered using a ramp transfer function. The middle image is rendered using only one of the two BCC textures and the artifacts due to missing data can be seen in the fuzzy rings. When both textures are used (right image) the signal is rendered correctly, resulting in complete and smooth rings.

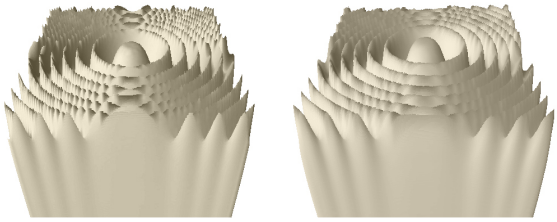

Figure 13: Marschner-Lobb shading (CC, BCC)

Figure 13 shows two shaded renditions of the Marschner-Lobb data. The BCC volume on the right was created from a downsampled larger version using non-separable wavelet filters for the BCC lattice [6]. The Cartesian data was resampled from a similar Cartesian volume.

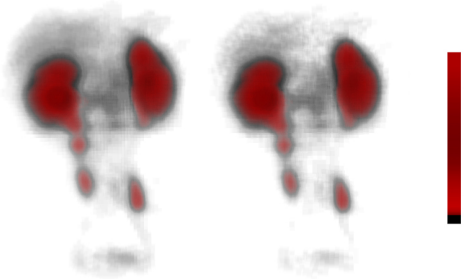

Figure 14: Kidney post-classification (CC, D4)

The last part of this section compares the rendering quality for the time-varying data sets. Figures 14 and 15 show one time frame of the kidney data set. Here, time frames 37 (Cartesian), or 52 (hexagonal), are rendered using post-classification. In Figure 14 the left image shows the Cartesian sampled data set and the right one displays the belonging time frame from the hexagonal lattice.

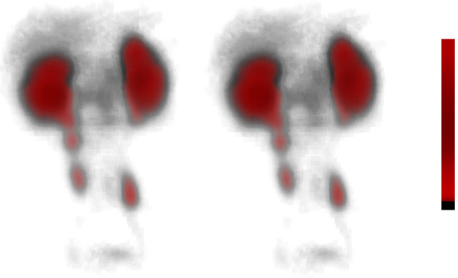

Figure 15: Kidney post-classification (BCC, BCC time)

Even though the right images uses only 35% of the original samples, all vital information is present. Using two or three additional time frames, the four-dimensional hexagonal lattice can also be rendered as a BCC volume as can be seen in Figure 15. Here, the left image shows a BCC rendition of the same time frame where the additional BCC texture is directly loaded from the next $D4^*$ volume. The right image interpolates the additional texture from the previous and the following time frame. Due to the slow changes in the animation, no real difference can be seen.

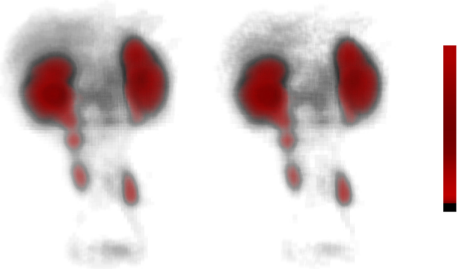

Figure 16: Kidney time frames (CC, CC Hardware)

Figures 16 and 17 show the results when a new time frame is selected and rendered. A new time frame can be loaded and rendered from disk, or interpolated in hardware. For the hardware based interpolation scheme, in a pre-processing step every 5th time frame of the data set was downsampled and loaded resident as low resolution volume into the texture memory. Here it can be used for fast preview when quickly skimming trough a time-varying volume. Two of these time frames are interpolated and weighted by the demanded time.

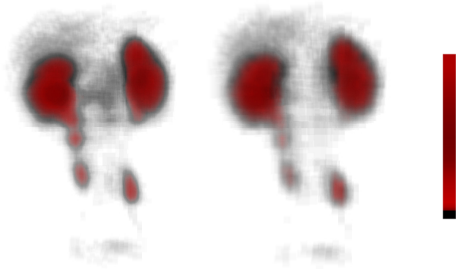

Figure 17: Kidney time frames (D4, D4 Hardware)

The rendering quality of the hardware interpolated images are not superior, but sufficient enough for preview purposes and to detect certain features in the data. The benefits are a by a large factor faster navigation through time-varying data sets (Table4).

6 Conclusions and Future Work

We have presented a hardware accelerated rendering scheme for hexagonal based volumetric data sets. The data sets used are sampled to the BCC or when varying over time to the $D4^*$ lattice. The data sets can be classified using pre- or post-classification and also be shaded using simple shading. The renditions can be created in a single rendering pass. All data sets can be rendered as BCC volume or as first Level-of-Detail version using one half of the BCC data set. We have also presented a small optimization which allows one to interactively select a specific time frame from time-varying data sets.

We have shown that the rendering performance is comparable and sometimes even improved with respect to the Cartesian sampled data sets. The quality of the renditions is comparable, but sometimes the BCC volumes are smoother than the Cartesian ones, which is due to the final blending step.

Future investigations will include an adaptation of pre-integrated volume rendering [5] for hexagonal data sets to further increase the rendering quality. Also, higher order filtering [7] can be applied to achieve high-quality renditions. Additionally, more research has to be spend to investigate proper interpolation schemes within the hexagonal lattices as this is not a trivial task, due to the lattice topology. Also other hardware accelerated rendering algorithms might be possible which are able to interpolate within the entire BCC volume and do not need to rely on the fact that the data can be decomposed into two Cartesian grids. Interesting would be also to further explore the possibilities for bricking large BCC volumes and to use linear separable or non-separable wavelet filters [6] for a progressive LoD based volume rendering technique. As less samples are needed for rendering BCC volumes, with respect to Cartesian sampled data sets, it can be assumed that less bricks have to be used for rendering the same amount of data. This would increase the rendering performance.

7 ACKNOWLEDGMENTS

The authors would like to thank Troy Farncombe for allowing us to use his dSPECT kidney data sets and the GrUVi lab for the invaluable support.

REFERENCES

[1] Kurt Akeley. Realityengine graphics. In *Proceedings of SIGGRAPH 93*, Computer Graphics Proceedings, Annual Conference Series, pages 109–116, August 1993.

[2] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Symposium on Volume Visualization and Graphics 1994*, pages 91–98, 1994.

[3] Hamish Carr, Thomas Theußl, and Torsten Möller. Isosurfaces on optimal regular samples. To be published, 2002.

[4] J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices and Groups*. Springer, 2nd edition, 1976.

[5] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Siggraph/Eurographics Workshop on Graphics Hardware 2001*, 2001.

[6] Alireza Entezari, Niklas Röber, and Torsten Möller. Wavelets on optimal sampling lattices for volumetric data. Technical Report SFU CMPT TR 2003-01, GrUVi Lab, School of Computing Science, Simon-Fraser University, March 2003.

[7] Markus Hadwiger, Helwig Hauser, and Möller Torsten. Quality issues of hardware-accelerated high-quality filtering on pc graphics hardware. In *11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*. University of West Bohemia, Pilsen, Czech Republic, February 2003.

[8] Luis Ibáñez, Chafiaâ Hamitouche, and Christian Roux. Ray casting in the bcc grid applied to 3d medical image visualization. In *Proceedings of the 20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 20, pages 548–551, 1998.

[9] Stephen Marschner and Richard Lobb. An evaluation of reconstruction filters for volume rendering. In *Proceedings of IEEE Visualization 94*, pages 100–107, 1994.

[10] M. Meissner, J. Huang, D. Bartz, K. Müller, and R. Crawfis. A practical comparison of popular volume rendering algorithms. In *Symposium on Volume Visualization and Graphics*, pages 81–90, Salt-Lake City, October 2000.

[11] Michael Meissner. Volvis website, 2000. `http://www.volvis.org`.

[12] Michael Meißner, Stefan Guthe, and Straßer Wolfgang. Interactive lighting models and pre-integration for volume rendering on pc graphics accelerators. In *Proceedings of Graphics Interface 2002*, 2002. to appear.

[13] Michael Meißner, Ulrich Hoffmann, and Straßer Wolfgang. Enabling classification and shading for 3d texture mapping based volume rendering using opengl and extensions. In *Proceedings of IEEE Visualization 1999*, pages 207–214, October 1999.

[14] Neophytos Neophytou and Klaus Müller. Space-time points: 4d splatting on efficient grids. In *Symposium on Volume Visualization and Graphics 2002*, pages 97–106, October 2002.

[15] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler. The volumepro real-time ray-casting system. In *SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 251–260, August 1999.

[16] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard pc graphics hardware using multi-textures and multi-stage rasterization. In *2000 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 109–118, August 2000.

[17] Neil J. A. Sloane. The sphere packing problem. In *Proceedings of the International Congress of Mathematicians*, pages 387–396, Berlin, 1998. Doc.Math.J.DMV Extra Volume ICM III.

[18] Jon Sweeney and Klaus Müller. Shear-warp deluxe: The shear-warp algorithm revisited. In *Joint Eurographics, IEEE TCVC Symposium on Visualization 2002*, pages 95–104, May 2002. Barcelona, Spain.

[19] Thomas Theußl, Torsten Möller, and Eduard Gröller. Optimal regular volume sampling. In *Proceedings IEEE Visualization 2001*, pages 91–98, Oct 2001.

[20] P. P. Vaidyanathan. *Multirate Systems and Filter Banks*. Prentice Hall, New Jersy, 1993.

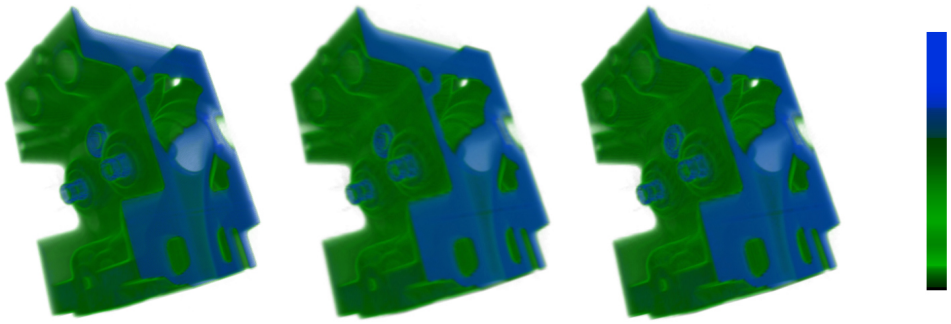

Figure 18: Engine pre-classification (CC, BCC, BCC Half)

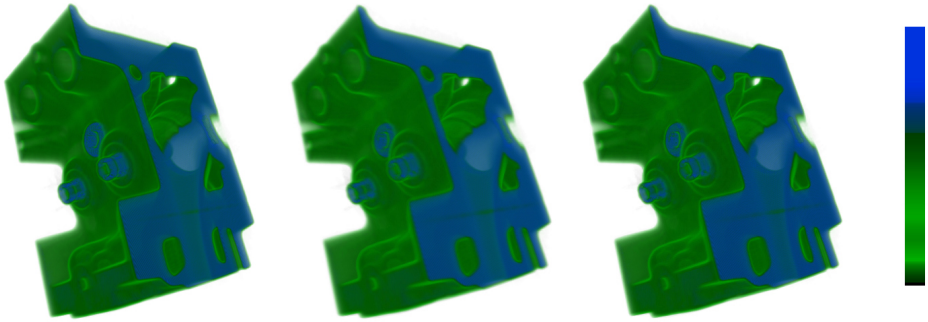

Figure 19: Engine post-classification (CC, BCC, BCC Half)

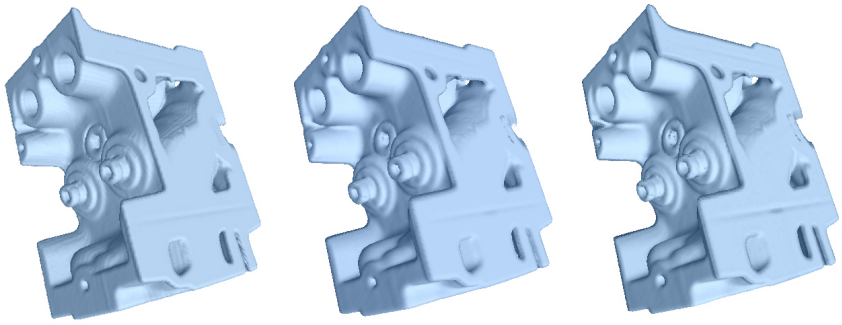

Figure 20: Engine shading (CC, BCC, BCC Half)