Interactive Visualization of Very Large Medical Datasets using Point-based Rendering

Christof Nuber, Ralph W. Bruckschen, Bernd Hamann and Kenneth I. Joy

Center for Image Processing and Integrated Computing, Department of Computer Science, University of California, Davis, One Shields Avenue, Davis, CA 95616, U.S.A.

ABSTRACT

Visualizing high-resolution volumetric medical datasets is a challenging task. Current off-the-shelf graphics hardware supports interactive texture-based volume-rendering of volumetric datasets up to a resolution of 512^3 data points only. We present a method that allows us to visualize higher-resolution datasets, providing images similar to texture-based volume-rendering techniques at interactive frame-rates and full resolution.

Our approach is based on an out-of-core point-based rendering approach. We pre-process the data by grouping points in the given dataset according to their value on disk and read them, when needed, from disk to immediately stream data to the rendering hardware. The high resolution of the dataset and the density of the data points allows us to use a pure point-based rendering approach. The density of points with equal or similar values within the dataset can be considered as being high enough to display regions and contours using points only.

With our data-stream based approach we achieve interactive frame-rates for volumes even exceeding 512^3 resolution. Interactivity is not restricted to navigation through the dataset itself. It is also possible to change the values of interest to be displayed in real-time, enabling us to change display-parameters and thus looking for interesting and important features and contours interactively. For a human brain extracted from a 753x1050x910 colored dataset we achieved frame-rates of 20 frames/second and more, depending on the values selected.

We describe a new way to interactively display high-resolution datasets without any loss of detail. By using points instead of textured volumes we reduce the amount of data to be transferred to the graphics hardware when compared to hardware-supported texture-based volume rendering. Using a data-organization optimized for reading from disk, we reduce the number of disk-seeks, and thus the overall update-time for a change of parameter-values.

Keywords: Point-based Rendering, Volume Rendering, Interaction, High-resolution datasets, Medical Imaging

1. INTRODUCTION

Well established 3D volumetric data visualization techniques, like isosurface extraction and volume rendering, usually cannot support real-time interaction with massive medical datasets. Thus, new ways of accessing and displaying large voxel datasets need to be investigated.

Isosurface extraction is the process of computing a surface $C(v) = \{x|F(x) = v\}$, where F is a scalar function defined over a 3D domain and v the function value defining the isosurface. Depending on the value, this process can result in a surface representing the surface of a structure within the voxel dataset, *i.e.*, organs or blood vessels. Interactive isosurface extraction is a two-step approach. First the cells containing the surface need to be identified, before the surface can be extracted from these active cells. We perform the search phase in a pre-processing step and reduce the contour generation phase to the task of loading and displaying the active cells using voxels.

Further author information: (corresponding author Christof Nuber)
Christof Nuber: E-mail: cnuber@ucdavis.edu
Ralph W. Bruckschen: E-mail: rwbruckschen@ucdavis.gov
Bernd Hamann: E-mail: hamann@cs.ucdavis.edu
Kenneth I. Joy: E-mail: joy@cs.ucdavis.edu

Isosurface extraction methods generate large numbers of triangles. Storing these triangles requires to store not only the vertex-coordinates, but also the connectivity information describing the triangles. For real-time rendering applications, the number of triangles becomes a limiting factor.

Multi-resolution representations, mesh reduction and view-dependent rendering are techniques used to cope with massive triangle-based surfaces originating from medical datasets. These approaches fail to provide interactive rendering at the highest original resolution for very large datasets. If translucency, for example, is required, to display occluded isosurfaces, artefacts can result due to lack of sufficient rendering-hardware support. Using mesh-based contour visualization methods is not suitable when a surface degenerates to a volumetric isoregion, like a contiguous set of neighboring voxels with equal values. Volumetric regions of equal value can thus not be displayed.

Hardware-supported volume rendering approaches using textures have become very popular, exploiting the capabilities of current hardware. Using hardware-supported algorithms, manipulations of volumes can be performed in real time, and both isosurfaces and isoclouds can be visualized. The volume to be displayed is restricted by the available amount of texture memory and the transfer rate between main and graphic memory. Large datasets must be rendered either at lower resolutions or with distributed renderers, where interactive frame rates are required. Affordable graphics boards are capable of rendering an eight-bit 512^3 voxel dataset at interactive rates and full resolution. When rendering medical datasets exceeding a size of 512^3 voxels, different approaches must be applied.

Our approach is based on the principle of reducing the amount of data that needs to be transferred to the graphics hardware for rendering. The general idea is to visualize contours using point primitives, which we will simply refer to as points. Due to the usage of points and the assumption that the point density is "high enough" (in the order of the screen resolution), we do not rely upon additional information, like connectivity and topology, to render a contour. This approach allows us to render contours and regions of very large voxel datasets at the highest possible resolution in real time. Using out-of-core techniques and storing the pre-processed data on disk, the limiting factor for interactive rendering of datasets, namely available memory, becomes the size of the hard-drive, allowing us to handle larger datasets than other technologies.

In our method, a given voxel dataset is transformed from grid space to a color-space representation, sorting voxels according to their colors. For each color, a list of coordinates for all points is generated. When a specific value is selected, we access the list of points stored on disk. Thus, for each value only one seek/read operation is required, which allows us to access data in real time.

2. RELATED WORK

Point-based rendering is used in several applications to display both large point datasets and surfaces. For complex surfaces, point-based rendering using splatting techniques has become a powerful alternative. Cline et al.[1] were among the first to apply point-based rendering to render complex surfaces inherent in medical datasets, using their "dividing-cubes" algorithm. They discovered that, with decreasing triangle size, it is more efficient with respect to memory and time to render point primitives when triangle size approaches pixel size. Later, Peng et al.[2] used point-based rendering to render complex surfaces at different levels of resolution by sampling a given surface and splatting only points. Zwicker et al.[3] used a texture mapping approach for splats to render point-based datasets acquired by laser-range and optical scanners. They also developed a technique called "EWA volume splatting,"[4] using splatting for direct volume rendering.

Out-of-core approaches have been used to speed up and simplify the generation of isosurfaces, e.g., by optimizing the selection of active cells. Livnat et al.[5] introduced a near-optimal isosurface extraction approach (NOISE), using a *kd*-tree-based search method to identify active cells. Shen et al.[6] extended the NOISE approach for both sequential and parallel isosurface extraction by using a regular 2D lattice instead of a *kd*-tree. Cignoni et al.[7] accelerated isosurface extraction by using interval trees to locate active cells for a given isovalue. Chiang and Silva[8] introduced an I/O-optimal approach of isosurface extraction to speed up the search for active cells. Chiang et al.[9] improved this approach by using a two-level indexing scheme, which further reduces main memory and disk space requirements. Bajaj et al.[10] developed a parallel accelerated contouring approach that partitions large datasets into multiple levels of granularity to achieve load balancing and disk-access optimization

for isosurface extraction. All these approaches have in common that they attempt to optimize the generation of a mesh-based isosurface representation, which can result in large numbers of triangles to be displayed. Bruckschen et al.[11] and Kuester et al.[12] used an out-of-core, point-based approach to interactively render time-varying vector field data in a virtual environment. Pascucci et al.[13] re-ordered a voxel dataset based on a z-ordered space-filling curve, allowing them to extract arbitrary slices with increasing resolution from a previously re-ordered voxel dataset at interactive frame rates, using a homogeneous PC-cluster.

While standard point-based volume-rendering approaches lack interactivity when it comes to select arbitrary values, as accessibility of the complete volume for image-generation is assumed, point-based surface rendering approaches assume that a surface is already given. No approach is applicable for real-time access and rendering of arbitrary values within very large voxel datasets.

3. APPROACH

The objective for our out-of-core approach is to generate visualizations that ideally bring out features in very large datasets and to do this at interactive frame rates, supporting real-time interaction capabilities. Using a point-based approach, we convert the colored voxel dataset from grid space to color space, sorting points by their color for faster isovalue-access. Assuming an underlying Cartesian grid and an RGB-color space, we perform a mapping

$$f_v : (x, y, z) \rightarrow (r, g, b)$$

to group points according to color.

In RGB-color space, a voxelset $V(r, g, b) = \{v_i | f_v(v_i) = (r, g, b)\}$ is accessed using the corresponding color (r, g, b) via the mapping

$$f_c : (r, g, b) \rightarrow \{(x, y, z)\}.$$

This transformation allows us to easily identify and access a set of values when the corresponding color is defined, without the necessity of searching the complete voxel dataset.

The color of a point can be determined by the set it belongs to. Retrieving the color at a given coordinate is a very expensive operation. As we are displaying point-sets associated with a known color value, we are interested in all points with a given value, not the color of the volume at a given coordinate.

All voxelsets $\{V\}$ are stored in one file, together with a look-up table containing position and length of each set within the file. As the colors are stored in a linear order on disk, an indexing scheme for RGB values is needed to convert a given color to an index. To determine the index of a given RGB value (encoded by three bytes) we use a Morton order scheme[14] that places colors close in RGB space close to each other on disk. The indexing scheme can be varied, depending on the underlying dataset (eight-bit greyscale datasets, for example, do not require any indexing scheme as the greyscale values can be used as indices). The only restriction is that the index for the background color (which depicts points that will be neither encoded nor rendered) equals zero.

Data processing is separated into two stages: pre-processing and data-retrieval. We use a two-pass approach to pre-process the data; during the first pass we set up the look-up table for offset-calculation, during the second pass we use the look-up table to resort and store the voxels in the data file. The look-up table is later used to locate and read data from disk for rendering.

3.1. Morton Order

Morton ordering is a mapping $\{M : N^m \rightarrow N\}$ that allows us to convert the three-tuple index of any RGB color to a single index within a linear array. It defines a linear index structure of the leaves of a multidimensional Cartesian grid in such a way that it optimizes the number of seeks needed to select an arbitrary subset. In the worst case, only eight seek-operations are required to access any rectangular $n \times n$ subset in a 3D octree, and usually less seek-operations are needed. This fact reduces the number of disk accesses based on the assumption that multiple voxelsets have similar colors. Using a Morton order for indexing reduces the number of seek-operations when reading voxelsets of multiple or similar color/function values when the indices are sorted by value.

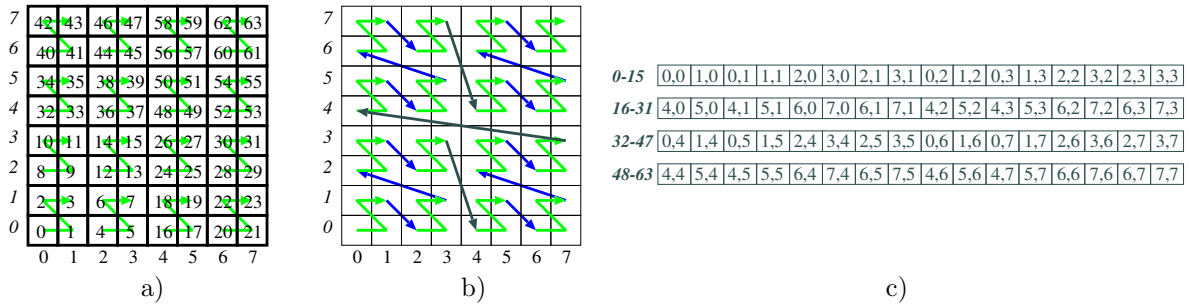


Figure 1. Bottom-up construction of a Morton-order for 2D-arrays: a) lowest-level traversal, b) complete traversal, c) resulting order

Constructing a Morton order is done by applying a basic pattern recursively to an m-dimensional structure with $\{2^n\}^m$ elements, where 2^n is the length of an edge and m denotes the dimension. Considering a 2D example (see Fig. 1), a $2^n \times 2^n$ dataset is traversed by following a z-shaped curve within each 2×2 square. The traversal scheme starts at the lowest level of resolution, continuing to the next-higher level until the complete grid has been traversed. This ordering scheme allows us to compute the index of any given n-dimensional vector (here the RGB color) using simple bit-shift operations. When using a Morton order for greyscale images, the resulting order depends on the level of grey of the voxels.

3.2. Pre-processing

For our out-of-core approach the data is pre-processed using a two-pass algorithm. In the first pass, a histogram h is generated, which serves as a base for a look-up table l that is used for retrieving the voxelsets $V(r, g, b)$ during rendering. During the second pass, the data values are stored at the pre-calculated offsets in the file.

Given the number of occurrences for each contour value, as contained within the histogram h, we can calculate the starting point $s_{V(r,g,b)}$ of each voxelset $V(r, g, b)$ in the file and store it in a second table, the look-up table l, which is later used for index-based data access. The look-up table l is generated using the histogram h by successively adding the number of bytes $h(r, g, b)$ needed to store a voxelset on disk. With $M(r, g, b)$ giving the Morton index and $M^{-1}(i)$ as the inverse, the starting position $s_{V(r,g,b)}$ of any voxelset $V(r, g, b)$ is defined by

$$s_{V(r,g,b)} = \sum_{i=0}^{M(r,g,b)-1} h\big(M^{-1}(i)\big).$$

The number of bytes to be read is defined by $h(r, g, b)$.

3.3. Data Encoding

For each valid voxel, we store its coordinates together with a gradient that can be used as a surface normal during rendering. Assuming that we have three four-byte values for the position and three bytes for the normal-information, 15 bytes are required initially to store each data-point.

We decided to use four bytes for position data and three bytes for normal information, enabling us to do lossless compression. The normal-information (n_x, n_y, n_z) is stored using one byte per component. Using four bytes for position data, we can handle volumes up to 2^{i+k+j}, with $i + k + j \leq 32$, *i.e.*, $2^{11} \times 2^{11} \times 2^{10}$, using basic shift-operations to eliminate bits not used for position information. Assuming that we have a 2^{32} dataset, the worst-case scenario for storage for this scheme is 28 GB, as compared to 60 GB when storing the data uncompressed and 12 GB when storing raw data. The expected file size for such a volume is usually smaller, as we store no voxels with "background"-value.

This encoding scheme is simple, and it provides sufficient data reduction together with efficient decoding and predictable storage space, which is necessary for look-up table initialization. The size of datasets we can handle is restricted only by the file-size supported by the operating-system, and the encoding-scheme used. The overall file-size depends mainly on the effective number of points to encode and the number of bits/bytes used to store them.

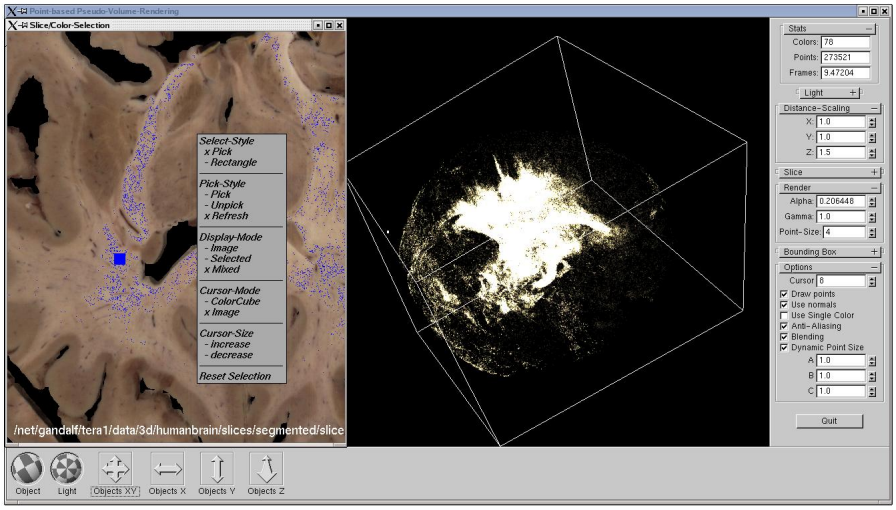


Figure 2. Interactive selection of values using an image slice: slice, menu and rendering result

3.4. Data Retrieval

Whenever a new set of voxelsets is selected, we must retrieve the voxels from disk in order to render them. As the voxelsets are stored in a defined order on disk, we access the voxelsets according to this order. This access can easily be executed by calculating the Morton index of every color selected and sorting the codes in ascending order. By accessing the datasets in ascending order we exploit the following facts:

- Similar selected colors are stored next to each other on disk.
- We avoid unnecessary seek operations by accessing datasets in a consecutive manner.
- If n datasets are located next to each other in a file, we can read by performing just one disk-read operation, reducing the number of seek/read operations and avoiding unnecessary disk rotations.

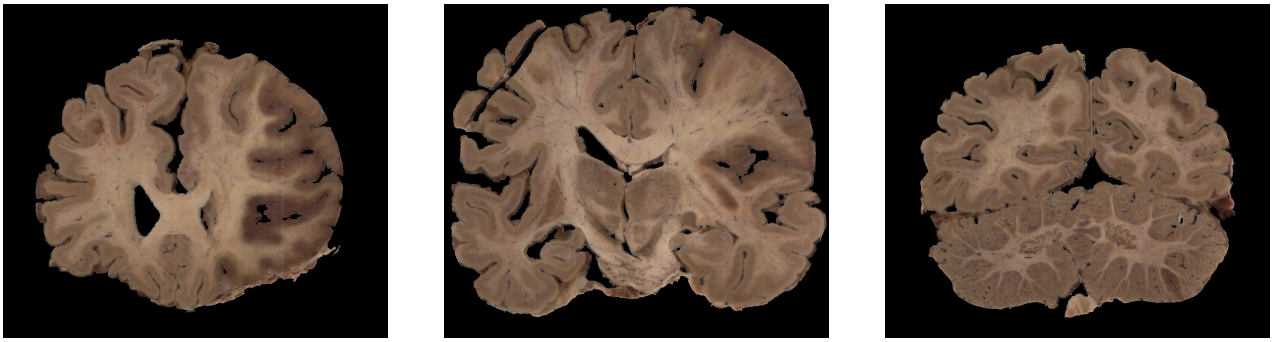
This type of data retrieval allows us to read the information from disk in real time. When data has been read from disk, we can immediately decode it and render it by the hardware.

3.5. Interactive Data Selection

We perform value selection using an image slice of the voxel dataset (see Figure 2). A slice-based selection is a highly intuitive approach, as most data is either stored as a volume and looked at by using slices within a plane (*i.e.*, MRI, CT), or is available as a set of slices (*i.e.*, the human brain dataset, see Fig. 3).

By picking pixels with a variable cursor-size or by selecting a rectangular region, the user defines which values should be rendered. We provide two different pick-modes, an image-mode and a color-space mode. In image-mode the selection is done in the image, resulting in all the colors that are currently covered by the cursor. In color space, the color of the image that has been picked together with its surrounding colors in RGB color space are selected. While the image-mode is useful when selecting tissue, the color space mode is useful when looking for similar values. Especially when point density is sparse, increasing the cursor enhances features (see Fig. 4). The mode can be changed interactively, allowing the user to switch whenever necessary.

By providing incremental selection of values, with only a limited number of voxelsets to be loaded at a time, interactive selection of values can be performed. Loading an already defined set of colors can take relatively more time, depending on the number of colors/voxelsets to load, their size and their location within the data file. De-selection of colors can be done the same way, giving the user full control over the point sets to be rendered.



Slice 200 Slice 400 Slice 600

Figure 3. Slices of the brain dataset (data courtesy of A. W. Toga, UC Los Angeles)

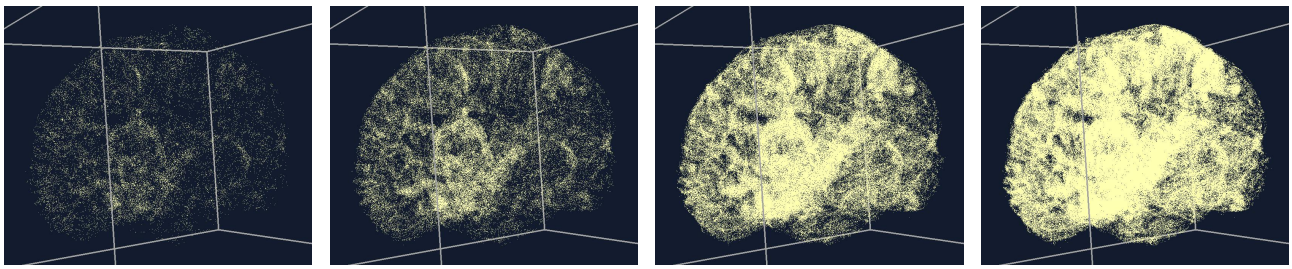
3.6. Adjusting Rendering Parameters

Not all datasets have the same resolution in every direction, so that the volume needs to be scaled accordingly. As the resolution is known beforehand, scaling can be adjusted correspondingly. For example, the brain dataset required a scaling of 1.5 in z-direction. When the values of interest are selected, adjusting the rendering parameters is the next important step to achieve a good visual representation.

The selected voxelsets can be rendered with or without alpha-blending, providing either a more surface-oriented or a volume-oriented view of the dataset. For fine-tuning, parameters like point-size, alpha- and gamma-value, and anti-aliasing can be changed interactively. Using GL commands and GL-point parameter extensions, we render each voxel as a square or circular splat, with a fixed or distance-dependent diameter. Depending on the distance from the dataset, the point-size needs to be changed to achieve sufficient point density.

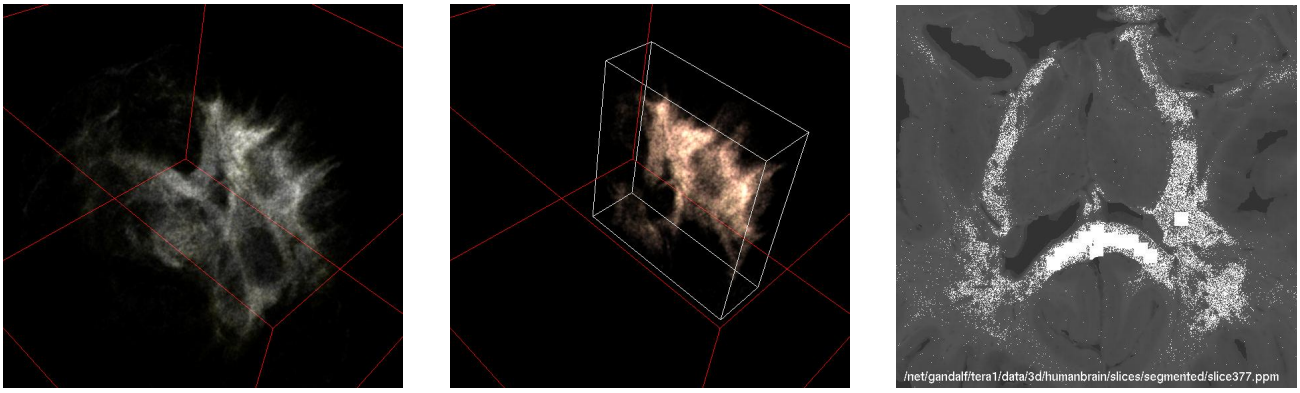
Some datasets are colored voxel datasets obtained from pictures. Colors can slightly change during data acquisition, so that several colors need to be selected in order to emphasize desired features in a visualization. Especially within the human body this can be an obstacle, as different tissue types can have the same color. If the density of color values between tissues differs, alpha-blending and gamma-correction operations can be used to reduce the contribution of those points to the resulting image. While gamma-correction changes the image, alpha-blending shows the accumulated values along the line of sight, generating an image resembling an X-ray. When rendering without alpha-blending, the stored normals can be used to adjust the contribution of voxels to the resulting image, providing us with a closed shaded surface if the point-density is high enough.

Depending on the number of voxel sets selected, the amount of data can become too large to be rendered at high frame rates. This situation typically occurs when interesting features have been detected in a smaller volume of the dataset. By defining a bounding box, we allow the user to reduce the number of pixels to be rendered, sending only those values to the rendering hardware that are within the given boundary. Also, surrounding material obstructing the view on features can be clipped. Figure 5 shows images of the brain with and without



a) b) c) d)

Figure 4. Point density with different image-cursor sizes: a) 3x3, b) 6x6, c) 9x9, d) 12x12; images rendered without normal information and single color (data courtesy of A. W. Toga, UC Los Angeles)



a) b) c)

Figure 5. Visualization of human brain dataset: a) no bounding box, b) bounding box, c) slice showing selected area (data courtesy of A. W. Toga, UC Los Angeles)

a bounding box, using the same parameter settings. We still must retrieve all data from disk, but only select those voxels for rendering which lie within the bounding box. As selection can be done during decoding, this can be performed at no relevant additional cost.

3.7. Strengths of Our Approach

Our approach leads to a simple and efficient algorithm for displaying contours and features of very large voxelized scalar fields. The approach is efficient for volumes exceeding resolutions of 512^3 voxels, and can support also volumes exceeding 2^{32} voxels using more bits for encoding. By using point primitives surfaces as well as local features defined by a fixed function value can be visualized. We can also visualize the original color of the voxels.

4. RESULTS

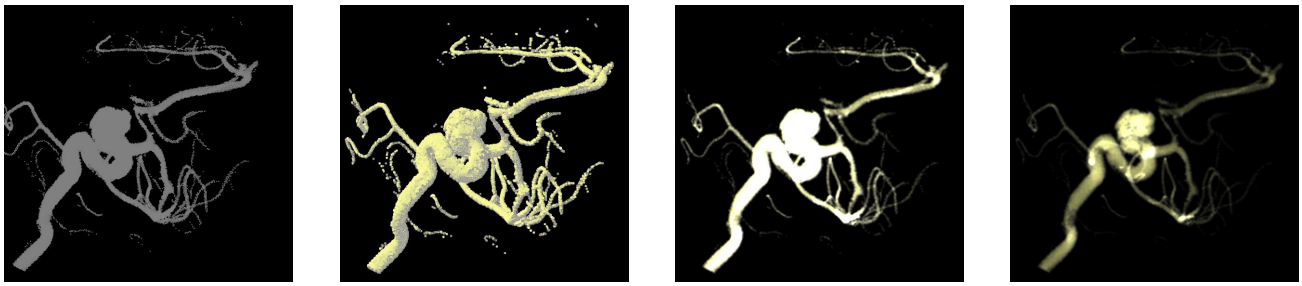
We have applied our approach to a aneurysm-dataset and a human brain dataset. All examples were generated by a Linux-PC with an Intel Pentium4 2GHz processor, two GB memory, a GeForce4 Ti4600 and a one-terabyte RAID-system for data storage.

4.1. Aneurysm Dataset

The aneurysm dataset is a 256^3 greyscale dataset. Figure 6 shows the aneurysm dataset rendered with several options. As can be seen, the results are similar to those using regular volume-rendering. With a window of approximately 600×600 we achieved frame-rates of 350 frames/second and above, with approximately 37000 points rendered. It can be seen in Figure 6 that our approach is capable of rendering surfaces (see Fig. 6b) and generating images similar to volume rendering approaches (see Fig. 6c and d). The frame rates achieved for this dataset are much higher than the rates for hardware-based volume rendering, as the amount of data to be displayed is much lower when compared to the size of the complete dataset used for hardware-based volume rendering.

4.2. Human Brain Dataset

The human brain dataset was generated by cutting slices off a frozen brain and taking photographs of the remaining top layers. The slices were segmented to remove regions showing ice and deeper layers.[15] The original dataset consists of 753 slices in RGB format, each slice having a resolution of 1050 by 910 pixels, with 24 bits each. We use the segmented slices stored in PPM-format, requiring 2.1 Gigabytes of storage space. The generated datafile has a size of 5.7 Gigabytes. Figure 5 shows how bounding boxes can be used to concentrate on smaller areas and reduce the amount of points to be rendered. Figure 7 shows images of the brain with different options. Using blending helps to eliminate "noise" in the picture and makes inner structures visible, structures that would otherwise be hidden.



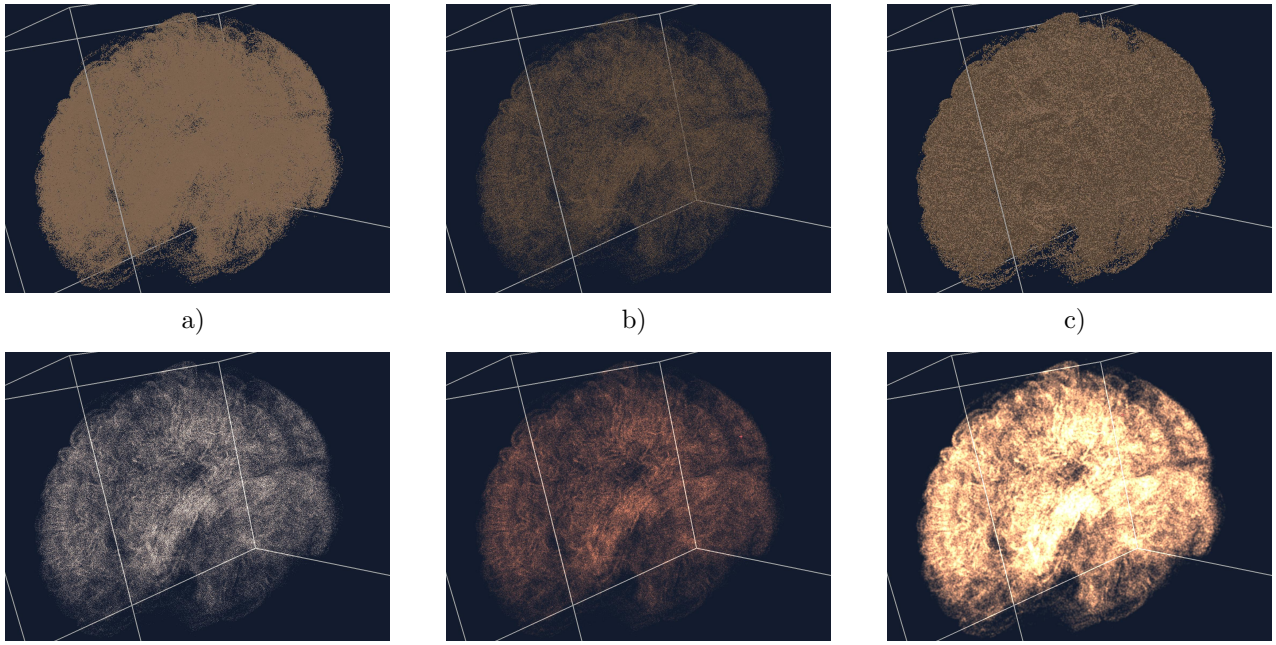
a) b) c) d)

Figure 6. Aneurism dataset: one value selected, using different rendering options; a) no normals, b) colored, no blending, c) blending with high alpha value, d) blending with low alpha value (Data courtesy of Philips Research, Hamburg, Germany)

Table 1 lists loading times and frame rates for the pictures shown. Clearly, the method can provide interactive visualization at reasonable frame rates for very large datasets.

5. CONCLUSIONS AND FUTURE WORK

We have introduced an interactive technique to render large discretized volumetric scalar field datasets in real time, allowing a user to interactively change the data being displayed. This technique supports fast and efficient means of accessing and displaying contours and bringing out features generated by several values, or value ranges, which is especially advantageous for datasets with no "inherent" dedicated isosurfaces like colored medical datasets.



a) b) c)

d) e) f)

Figure 7. Image sequence of brain dataset with different options; a) pixel-size 1, no normals, b) normals activated, c) pixel-size set to 3, d) blending activated, e) gamma-value changed, f) transparency changed (data courtesy of A. W. Toga, UC Los Angeles)

Dataset	Aneurysm (greyscale)		Human Brain RGB
screen resolution	600×600		(1600×1200)
original size	16.4 MB		2.1 GB
size of datafile	1 MB		5.7 GB
number of colors	173		330460
colors selected	1	173	2409
points rendered	37154	146574	114103
initial load time	$0.015s$	$0.5s$	$0.8s$
frames/second	380	50	80

Table 1. Load and rendering times for datasets

Our approach supports the generation of images of large datasets rendered at full resolution. We have shown that, with a sufficient density of points, structures and regions can be detected easily in rendered images. Our method can display contours using multiple isovalues. It supports features like color preservation and translucency, and it does not depend on high-performance system hardware; it does not suffer from limited buffer-fill rates, as only points, and not polygons, are drawn.

The basic principle of our out-of-core approach is not restricted to point-based rendering. It can be adapted to any technique that supports on-line re-loading of data. Point-based rendering provides several advantages with respect to performance due to the simplicity of the primitives used for rendering. The size of the used datasets are large, but the rendered data only represents a fraction of this. Random file access is important for this out-of-core-approach, so any arbitrary compression will be reduced to the single voxelsets on disk. If the dataset is sparse with respect to the number of non-background values, the generated datafile can become very small, as we store only information about voxels with non-background-values.

For better surface visualization, we plan to use both the direction and the size of the gradient, allowing us to emphasize high-gradient regions. Especially for datasets with low values (and thus dark colors), or greyscale/single-valued datasets, an extended color-transfer function needs to be added, allowing a user to brighten and change the color of the displayed contours. We intend to optimize the file structure by using incremental encoding for position information and angle-based quantification of normal information. Using differential encoding for the positional dimension with the largest span, we expect to achieve a reduction of about 33% for the position-values, storing quantified angles and a quantified length instead of components for the normals will give us a 33% reduction for the normal information.

Acknowledgments

This work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award) and ACI 0222909, through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the National Institute of Mental Health and the National Science Foundation under contract NIMH 2 P20 MH60975-06A2 and the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159 We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) and the Center for Neuroscience at the University of California, Davis.

REFERENCES

1. H. E. Cline, W. E. Lorensen, S. Ludke, C. R. Crawford, and B. C. Teeter, "Two algorithms for the three-dimensional construction of tomograms," *Medical Physics* **15**, pp. 320–327, June 1988.
2. Q. Peng, W. Hua, and X. Yang, "A new approach of point-based rendering," in *Proceedings of the Computer Graphics International 2001*, pp. 275–282, IEEE, Computer Society Press, July 2001.
3. M. Zwicker, H. Pfister, J. van Baar, and M. Gross, "Surface splatting," in *Proceedings of SIGGRAPH 2001*, pp. 275–282, ACM Press, (New York), Aug. 12–17 2001.

4. M. Zwicker, H. Pfister, J. van Baar, and M. Gross, "EWA volume splatting," in *Proceedings of Visualization 2001*, IEEE, Computer Society Press, Oct. 12–17 2001.
5. Y. Livnat, H. W. Shen, and C. R. Johnson, "A near optimal isosurface extraction algorithm using the span space," *IEEE Transactions on Visualization and Computer Graphics* **2**, pp. 73–84, Mar. 1996.
6. H.-W. Shen, C. D. Hansen, U. Livnat, and C. R. Johnson, "Isosurfacing in span space with utmost efficiency (ISSUE)," in *IEEE Visualization '96*, pp. 287–294, 496, IEEE, Computer Society Press, Oct. 1996.
7. P. Cignoni, P.Marino, C. Montani, E. Puppo, and R. Scopigno, "Speeding up isosurface extraction using interval trees," *IEEE Transactions on Visualization and Computer Graphics* **3**, pp. 158–170, Apr./June 1997.
8. Y.-J. Chiang and C. T. Silva, "I/O optimal isosurface extraction," in *IEEE Visualization '97*, pp. 293–300, 554, IEEE, Computer Society Press, Oct. 1997.
9. Y.-J. Chiang, C. T. Silva, and W. J. Schroeder, "Interactive out-of-core isosurface extraction," in *IEEE Visualization '98*, pp. 167–174, 530, IEEE, Computer Society Press, Oct. 1998.
10. C. L. Bajaj, V. Pascucci, D. Thompson, and X. Y. Zhang, "Parallel accelerated isocontouring for out-of-core visualization," in *Proceedings of Parallel Visualization and Graphics Symposium 1999*, pp. 97–122, IEEE, Oct. 25–26 1999.
11. R. W. Bruckschen, F. Kuester, B. Hamann, and K. I. Joy, "Real-time out-of-core visualization of particle traces," in *IEEE Symposium on Parallel and Large-data Visualization and Graphics (PVG 2001)*, A. H. D. E. Breen and A. Koning, eds., pp. 45–49, IEEE, IEEE Computer Society Press, (Los Alamitos, California), 2001.
12. F. Kuester, R. W. Bruckschen, B. Hamann, and K. I. Joy, "Visualization of particle traces in virtual environments," in *ACM Symposium on Virtual Reality Software & Technology 2001*, pp. 151–157, ACM, ACM Press, New York, NY, USA, November 2001.
13. V. Pascucci and R. J. Frank, "Global static indexing for real-time exploration of very large regular grids," in *Proceedings of Supercomputing 2001*, ACM, 2001.
14. H. J. Samet, *Design and Analysis of Spatial Data Structures: Quadtrees, Octrees, and other Hierarchical Methods*. Addison-Wesley, 1989.
15. I. Takanashi, E. B. Lum, K.-L. Ma, J. Meyer, B. Hamann, and A. J. Olson, "Segmentation and 3d visualization of high-resolution human brain cryosections," in *Visualization and Data Analysis 2002*, R. F. Erbacher, P. C. Chen, M. Groehn, J. C. Roberts, and C. M. Wittenbrink, eds., **4665**, pp. 55–61, SPIE - The International Society for Optical Engineering, (Bellingham, Washington), 2002.