Interactive Visualization of Very Large Datasets Using an Out-of-Core Point-based Approach

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

Center for Image Processing and Integrated Computing (CIPIC)
Department of Computer Science
University of California
One Shields Avenue
Davis, CA 95616-8562

Abstract

We present an out-of-core, point-based approach for interactive rendering of very large volumetric datasets. Our approach is based on the assumption that the density of voxels with the same function-value in large discretized volumetric scalar fields is high enough to be used to render contour and volume approximations using points to represent the voxels. This approach allows us to visualize isovalue-structures in high-resolution datasets at full resolution and interactive frame rates.

In a pre-processing step, we sort the voxels by function-value and store them in a file together with a look-up table for later interactive retrieval. The displayed voxelsets can then be changed in real time by determining their locations in the file and loading them into memory. As we store position, and not function-value, the volumetric dimension of a dataset to be handled by our approach is limited by three factors: the number of points that can be rendered to achieve a sufficient frame rate, the number of bits used to store the position data, and the maximum file-size supported by the operating system. Depending on the spatial distribution of the voxels among the function-values selected, the result is either one or multiple contours or "isoclouds".

Keywords: Point-based Rendering, Isovalue, Volume Rendering, Out of Core, Data Exploration

1 Introduction

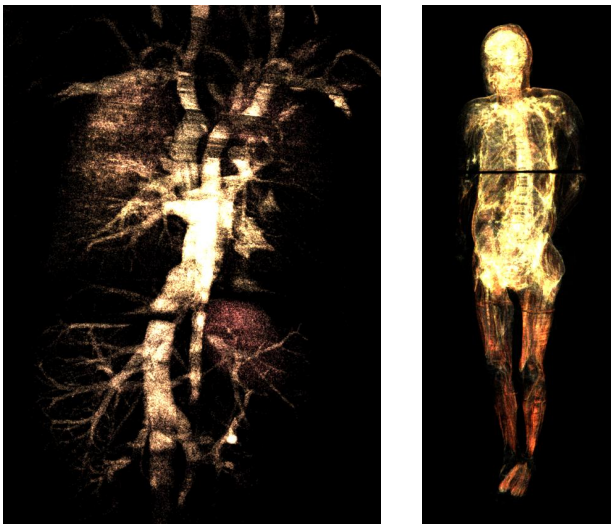
Well established 3D volumetric data visualization techniques, like isosurface extraction and volume rendering, usually cannot support real-time interaction with massive datasets. New ways of accessing and displaying large volumetric datasets need to be investigated.

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. This process is done in two phases, a search phase to identify the cells of a dataset that defines the isosurface (active cells) and a generation phase, where the isosurface is extracted using the active cells. In our approach, 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.

Isosurface extraction algorithms usually assume that a trivariate scalar field is represented on a grid in an at least C^0-continuous fashion. Isosurface extraction methods generate large numbers of triangles, and one must store positional and connectivity information for the resulting triangle-meshes. 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 contour visualization. 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. Mesh-based contour visualization methods are not suitable when a contour (locally) degenerates to an actual volumetric isoregion, like a contiguous set of neighboring voxels. Changes in density 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, like the GeForce4, provide up to 128MB of on-board memory, sufficiently enough memory to render an eight-bit 512^3 volumetric dataset at interactive rates at full resolution. The available on-board memory becomes the limiting factor for hardware-accelerated volume rendering methods. When rendering datasets exceeding a size of

*E-mail: {cnuber,rwbruckschen,bhamann,kijoy}@ucdavis.edu



a) b)

Figure 1: Visible female dataset: a) upper-body blood system, b) bone-marrow color selection.

512^3 voxels, different approaches need to 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 use a discontinuous, point-based approach to visualize contours using point (= small volumetric) 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," we do not rely upon additional information, like connectivity and topology, to render a contour. This approach allows us to render contours of very large volumetric datasets at the highest possible resolution in real time (Fig. 1 shows pictures generated for the visible female dataset). By using out-of-core techniques the limiting factor for interactive rendering of datasets, namely available memory, becomes the size of the hard-drive, which allows us to handle datasets that are much too large to fit into graphic or main memory.

In our out-of-core method, a given gridded dataset is transformed from grid space to a color-space representation by sorting voxels according to their colors (which, in turn, are defined by the original function values). For each color, a list of voxels containing the coordinates of all voxels is generated. When a value is selected, we access the list of voxels already computed and stored on disk. Thus, for each value only one seek/read operation is required, which allows us to access data in real time. Selecting consecutively stored values reduces the number of seek/read operations further.

2 Related Work

Point-based rendering is used in several applications to display both large point-based datasets and surfaces. Especially for complex surfaces that would require a large number of triangles for surface-based discretization, point-based rendering using splatting techniques has become a powerful alternative. Cline et al. [1] applied 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. 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 for an isosurface. 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. Bruckschen et al. [11] and Kuester et al. [12] used an out-of-core, point-based approach to interactively render time-varying vector fields in a virtual environment. Pascucci et al. [13] presented an approach based on a z-ordered space-filling curve, allowing them to extract arbitrary slices from a previously re-ordered volumetric dataset at interactive frame rates, using a homogeneous PC-cluster.

While 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, and out-of-core approaches for surfaces are limited to mesh-based isosurface-generation. No approach is applicable for real-time access and rendering of arbitrary values within very large volumetric 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 volumetric colored 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 the 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 volumetric 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 are using 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: preprocessing and data-retrieval. During the preprocessing the data-file is generated using a two-pass algorithm. For data retrieval the look-up table is used to 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

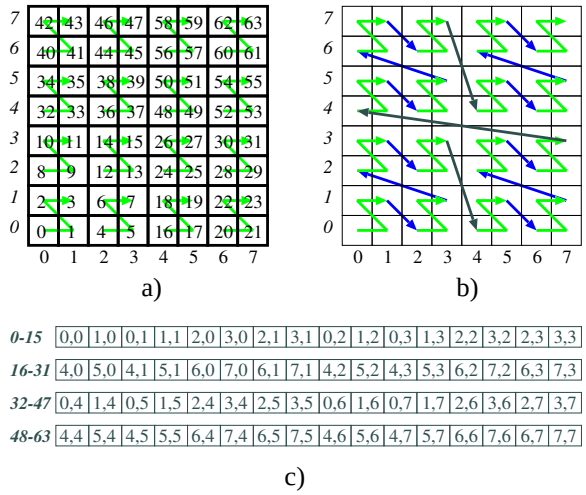


Figure 2: Bottom-up construction of a Morton-order for 2D-arrays: a) lowest level traversal, b) complete traversal, c) resulting order.

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.

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. 2), 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.

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 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 of each voxelset V 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 of any voxelset V is defined

by

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

Some datasets contain background noise that cannot be eliminated easily. To avoid storing unnecessary background noise data, we use "background-images" to clean up the initial histogram before setting up the look-up table. The histogram entries of all the colors found in the background-images are set to zero, excluding them from further processing during the encoding/storing step.

3.3 Data Encoding

For each valid voxel, we store its coordinates together with a normalized gradient that can be used as a surface normal during rendering. Assuming that we have three four-byte-values for position and three bytes for normal-information, 15 bytes are required for each data-point. In order to keep file-size small, the number of bits/bytes per voxel on disk needs to be reduced.

We are quantizing normals using 240 points distributed nearly uniformly over the sphere. We consider these 240 normals to be sufficient for rendering. When alpha-blending is disabled, we can use the normals to shade points, providing us with the "illusion" of a shaded surface when point-density is high enough. The use of normals improves the perception of 3-dimensional features. The 240 points are generated from a dodecahedron by triangulating each pentagon uniformly and performing a four-split operation on the resulting triangles. This approach results in 240 triangles, of which we use the centers. The angle between two adjacent normals is approximately 23 degrees. The normals can be stored using one byte only.

The points are stored with their position relative to the first valid point, so that the number of bytes needed for each dimension is determined by the size of the bounding box of all valid data-points. This allows us to optimize the number of bits needed for each data-point. Further, we only store the number of points for each z-plane. The z-value is given implicitly, followed by the x- and y-coordinates of the corresponding points. Assuming that we have 255 different colors in a 4096^3 volume, the worst-case scenario for storage for this encoding-scheme is $255 * 4096 * 3$ (#colors*#slices*sizeof(counter)) + $4096^3 * 1$ (normal) + $4096^3 * 3$ (xy) bytes ≈ 256 Gigabytes as compared to 448 Gigabytes when storing the data uncompressed.

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. The file-size depends mainly on the number of points to encode and the number of bits/bytes used to store them.

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. After the data has been read from disk, we immediately decode it for rendering.

3.5 Rendering Parameters

The selected voxelsets can be rendered either with or without alpha-blending, providing either a more surface-oriented view of the dataset or a volume-oriented view. 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.

The datasets used are colored volumetric datasets based upon pictures, i.e., 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 tissues can have the same color. It is possible that points are selected and rendered that are not of any interest. Decoding and rendering only points within a pre-defined region (bounding box) helps us in dealing with this situation and allows us to select more values at a time, as not all points are decoded and rendered. 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, see Fig. 5.

3.6 Interactive Value Selection

For interactive value selection, we use a slice of the dataset, in which colors can be selected and unselected by simply clicking on the pixels in the slice. Whenever the user selects one or several pixels/colors, the corresponding voxelsets are

retrieved from disk, decoded and rendered. Using this incremental approach, where only some voxelsets are loaded at a time, we can provide interactive selection of values. 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.

3.7 Strengths of Our Approach

Our approach provides a simple and efficient method for displaying contours and features of very large volumetric scalar fields. The approach is efficient for volumes exceeding resolutions of 512^3 voxels, and supports also volumes exceeding 4096^3 voxels. 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. By using indices for data access we can use any data tuple as a sorting/accessing criteria, provided that the indexing scheme is given. Our approach can also be applied to non-Cartesian, irregular grids by sampling the given grid onto a Cartesian grid.

4 Results

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

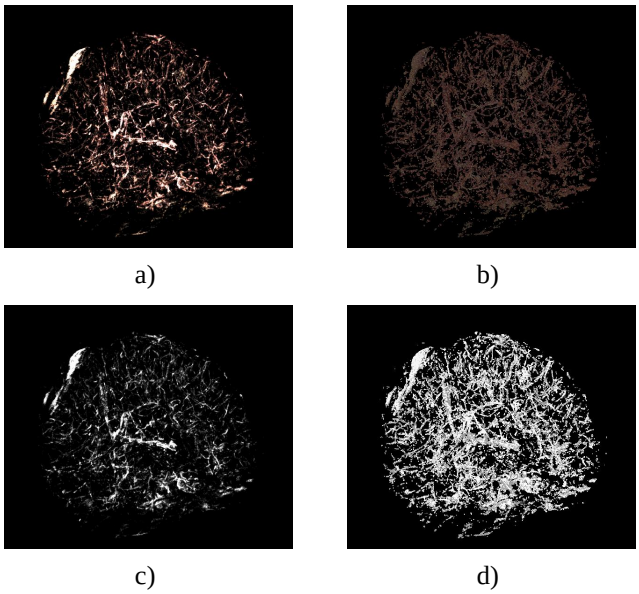
4.1 Human Brain Dataset

The human brain dataset was generated by cutting slices off a frozen brain and taking pictures of the remaining top layers. The slices were segmented to remove regions showing ice and deeper layers [15]. The original dataset consisted of 753 slices in RGB format, each slice having a resolution of 1050 by 910 pixels with 24 bits each. We are using the segmented slices stored in PPM-format, requiring 2.1 Gigabytes of storage space. The generated datafile has a size of 1.7 Gigabytes.

Figure 3 shows several snapshots of the human brain. The goal was to visualize blood vessels within the brain. Loading the complete data from a RAID requires about 0.8 seconds for the 2409 colors and 114103 points used; the frame rates achieved are about $80fps$, at a resolution of 1600×1200 pixels.

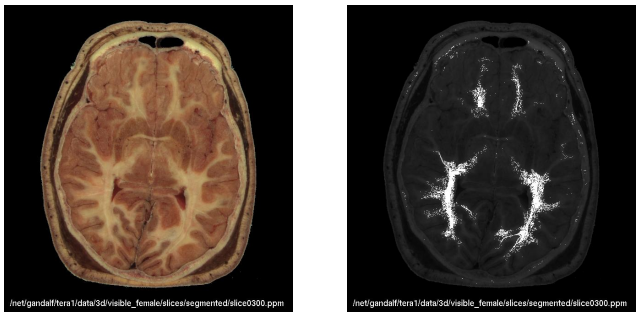
4.2 Visible Female Dataset

The visible female dataset, provided by the National Library of Medicine, contains 5189 slices taken at 0.33mm distance consisting of 2048×1216 pixels each (36.1 Gigabyte). The original images were cropped to 1640×940 pixels, eliminating outer areas containing no data and reducing the size



a) b) c) d)

Figure 3: Human brain dataset using different rendering options: a) colored, with blending, b) colored, no blending, c) unicolor, with blending, d) unicolor, no blending. (Dataset courtesy of A. W. Toga, UC Los Angeles)



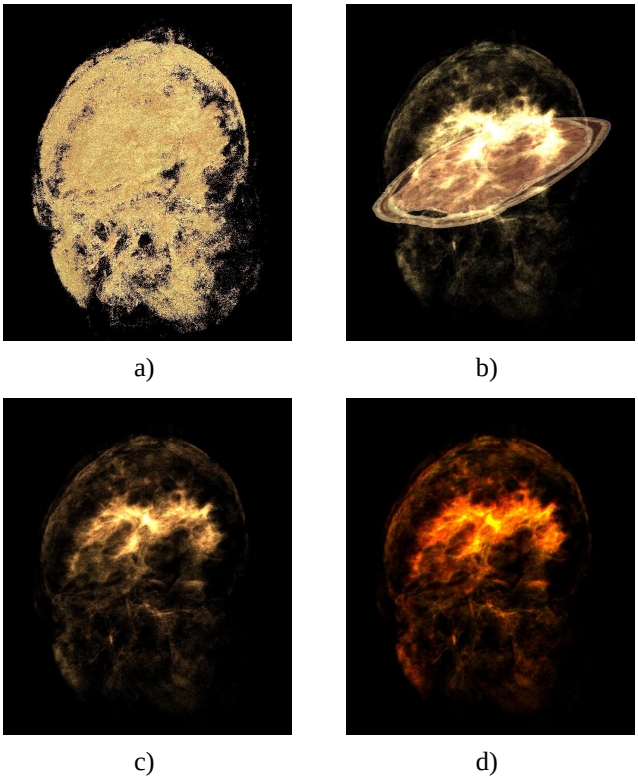
a) b)

Figure 4: Slice used for visualizing brain-dataset: a) original slice, b) selected pixels shown in white color.

of the dataset to 22.4 Gigabytes. The generated datafile has a size of 12.7 Gigabytes. The surrounding ice was removed by converting the RGB values to the CMY color model and setting all colors with a "color-angle" between $0.8 \times \pi$ and $1.5 \times \pi$ to background color.

Figure 4 shows a slice used for interactive selection of datasets, and Figure 5 shows the resulting images. The number of selected colors is 2299, resulting in 1239811 points to be used for rendering; the displayed volume has been reduced to the head-section. Loading and decoding for rendering required about 21 seconds, the selection-process could be performed at interactive frame-rates, as the number of colors added was well below 100 colors at a time. The average frame-rate during rendering at 1600×1200 pixels was about $17fps$.

Although point selection seems to be relatively sparse



a) b) c) d)

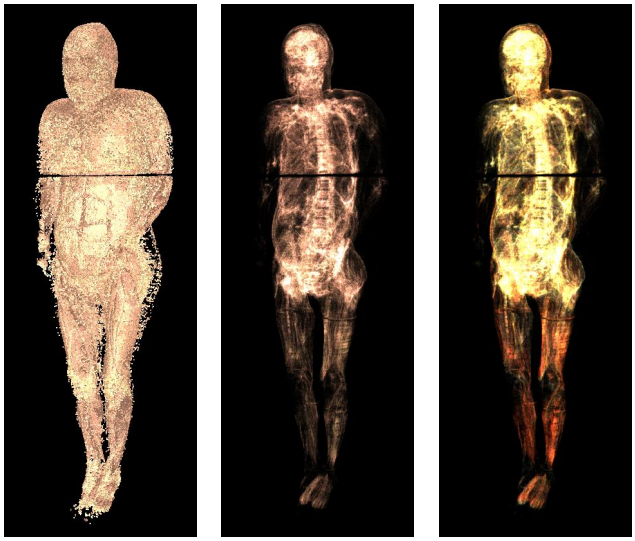
Figure 5: Renderings of the volume (head section only) for the selection shown in Fig. 4 b): a) points rendered without blending, b) alpha-blending applied, all voxels rendered in same color, slice shown, c) alpha blending applied, voxels rendered in original colors, d) alpha blending applied, voxels rendered in original colors with gamma-modification applied.

when looking at neighboring slides, it can be seen that our approach can visualize features and structures. Whereas in Fig. 5 a) no structure or feature can be detected, applying alpha-blending makes features behind the outer (thinner) layer of points visible. Fine-tuning using alpha-blending and alpha-correction provides us with even more information about the point density for the selected values, see Fig. 5 b) - d).

Dataset	Human Brain	Visible Female (Brain)	Visible Female (Bone Marrow)
original size	2.1 GB	22.4 GB	22.4 GB
size of datafile	1.7 GB	13.6 GB	13.6 GB
number of colors	330460	1333041	1333041
colors selected	2409	2299	897
points rendered	114103	1239811	7057825
initial load time	$0.8s$	$21s$	$7s$
fps (1600×1200)	80	17	3

Table 1: Load- and rendering times for datasets.

Figure 6 shows pictures of the complete visible female dataset with bone-marrow structure selected for rendering. In Figure 6 a) it can be seen that the selection also includes



a) b) c)

Figure 6: Visible female dataset (bone-marrow selected): a) points rendered without blending, b) alpha-blending applied, c) alpha-blending applied, gamma-value changed.

points of the skin and muscular tissue; applying alpha-blending makes the internal bone structure visible, see Figure 6 b) and c).

Table 1 provides an overview of loading times and frame rates for the pictures shown. It can be seen that we can provide interactive visualization at reasonable frame rates for even the largest 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. Using our scheme, 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.

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. Our technique supports features like color preserving 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.

Our technique can also be applied to more general multivariate scalar datasets, either using Morton-ordering in n dimensions or another appropriate ordering scheme that sorts

similar vectors (or vectors that are likely to be selected together) in a linear order so that they are close to each other. Our approach can also be used for time-varying datasets. Change of time is just one more parameter used to select an isosurface. Adding a temporal dimension does not affect preprocessing of the data. Changes in isovalue and change of time-step require us to only reload new information, which can be done in real time.

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. Nevertheless, 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 random file access is important for this out-of-core-approach, so any arbitrary compression will be reduced to the single voxelsets on disk. As no ordering of the voxels within a set is needed, different storage/compression approaches could be investigated exploiting re-ordering of the voxels in a set.

In order to improve visual perception of 3D structures, a shadow model should be added that supports simple and efficient shadow calculations for point-based contour renderings. 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. A value-transfer function that changes a value before loading the data from disk, and thus changes the criterion for selecting and grouping contour values, might also be beneficial in this context.

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*, vol. 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*, (New York), pp. 275–282, ACM Press, 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. Shen, and C. Johnson, "A near optimal isosurface extraction algorithm using the span space," *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, pp. 73–84, Mar. 1996.

[6] H.-W. Shen, C. Hansen, U. Livnat, and C. 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*, vol. 3, pp. 158–170, Apr./June 1997.

[8] Y.-J. Chiang and C. 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. Bajaj, V. Pascucci, D. Thompson, and X. 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.), (Los Alamitos, California), pp. 45–49, IEEE, IEEE Computer Society Press, 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. 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. Erbacher, P. Chen, M. Groehn, J. Roberts, and C. Wittenbrink, eds.), vol. 4665, (Bellingham, Washington), pp. 55–61, SPIE - The International Society for Optical Engineering, 2002.

Christof Nuber received his Diploma degree in computer science and engineering (Dipl. Inf.) from the University of Kaiserslautern, Germany, in 1995. Between 1995 and 2001 he was working as a computer scientist at Daimler Chrysler Aerospace Germany, focusing on virtual reality techniques and their application during the product development life cycle. He received his Doctorate degree (Dr. rer. nat.) in 2001 from the University of Kaiserslautern. He is currently working at the "Center for Image Processing and Integrated Computing" (CIPIC) at UC Davis. His special interests are virtual reality and visualization of very large datasets.

Ralph W. Bruckschen received his degree in mathematics (Dipl. Math.) at the University of Paderborn, Germany, in 1999. During his studies he was working on large scale CFD visualization in virtual environments. Between 1999 and 2000 he was working as a software developer for the Vircinity IT Consulting GmbH in Stuttgart, Germany. He is currently working at the "Center for Image Processing and Integrated Computing" (CIPIC) at UC Davis. His special interests are out-of-core algorithms and interactive visualization of very large datasets.

Bernd Hamann serves as co-director of a UC Davis Organized Research Unit, the "Center for Image Processing and Integrated Computing" (CIPIC), University of California, Davis. He is a full professor of computer science at the University of California, Davis. His main research interests are visualization, geometric modeling, computer graphics, and virtual reality. His current research focuses on hierarchical representations and visualization methods for very large scientific data sets. Bernd Hamann received a B.S. in computer science, a B.S. in mathematics, and an M.S. in computer science from the Technical University of Braunschweig, Germany. He received his Ph.D. in computer science from Arizona State University in 1991. Hamann is also a Faculty Computer Scientist at Lawrence Berkeley National Laboratory, a Participating Guest researcher at Lawrence Livermore National Laboratory, and an adjunct faculty in the Department of Computer Science at Mississippi State University.

Kenneth I. Joy is a Professor in the Computer Science Department at the University of California at Davis. He came to UC Davis in 1980 in the Department of Mathematics and was a founding member of the Computer Science Department in 1983. Professor Joy's research areas are visualization, geometric modeling, and computer graphics. Professor Joy received a B.A. (1968) and M.A. (1972) in Mathematics from UCLA, and a Ph.D. (1976) from the University of Colorado, Boulder. He has worked a number of years in the computer industry, and consults regularly on computer graphics, massive data visualization and geometric modeling. He is a participating guest researcher at the Center for Applied Scientific Computing and serves on the board of advisors of the Center for Computational Engineering at Lawrence Livermore National Laboratory.