PREPRINT

Parallelizing Sparse Grid Volume Visualization with Implicit Preview and Load Balancing

Matthias Hopf Thomas Ertl

Visualization and Interactive Systems Group, IfI, University of Stuttgart

Abstract

New algorithms that work entirely on sparse grids can create data sets that cannot be handled on uniform grids any more due to their size. On the other hand, most visualization techniques are only capable of handling uniform grids. As the interpolation on sparse grids is a complicated and time consuming process, direct volume visualization is unthinkable for bigger data sets until the underlying interpolation is accelerated by some orders of magnitude.

On the other hand, quite a number of super computers and PC clusters exist nowadays, providing MPI as the primary communication API. By streaming the data sets and the resulting images from and to the end user's workstation, their processing power can be utilized without leaving the office.

Parallelizing visualization techniques rises the necessity to balance the computational load, and for time consuming rendering methods previews are useful for the user. Both, generating preview images and load balancing, is performed explicitly in most cases. In this case study, we approach these problems by applying a special pixel rendering sequence, which achieves superb results implicitly without generating communication overhead.

Keywords: sparse grids, parallel, volume visualization, preview, load balancing, MPI

1 Introduction

The ever growing size of data sets resulting from industrial and scientific simulations and measurements have created the need to employ multi-resolution techniques for both analysis speedup and data reduction. Among the most sophisticated approaches are wavelets and sparse grids. Based upon hierarchical tensor product bases, the sparse grid approach is a very efficient one improving the ratio of invested storage and computing time to the achieved accuracy for many problems in the area of numerical solution of partial differential equations, for instance in numerical fluid mechanics. Recently, the best of both worlds have been merged by using wavelet bases in the sparse grid representation of multi-resolution data sets [4]. We are studying how these grids can be visualized interactively.

There are already several algorithms that are working entirely on sparse grids [1, 2, 5, 6, 17], creating data sets that cannot be handled on uniform grids in full resolution any more due to their size. Many of these systems work on three dimensional data. One of the best understood volume visualization techniques for scalar data is direct volume rendering [3, 7, 8, 16]. Velocity information and 1-forms, on the other hand, can be preprocessed so that direct volume visualization can be used for this data type as well, e.g. with three dimensional LIC [11]. The visualization of 2-forms is yet an unsettled question. As the underlying interpolation is the computation time dominating algorithmic part, we cannot benefit from hardware accelerated volume visualization techniques. Thus we use ray casting, which guarantees for the best image quality at little or no additional cost.

{hopf,ertl}@informatik.uni-stuttgart.de
Institut für Informatik
Breitwiesenstr. 20 – 22
70565 Stuttgart
Germany

We have already introduced visualization toolkits that are working directly on sparse grids [12, 13, 14]. By accelerating the sparse grid interpolation using special graphics hardware we were able to perform direct volume rendering interactively. However, the graphics hardware acceleration approach is limited to high end graphics systems with a high pixel depth and to sparse grids of level 10-11 (which resemble uniform grids of size 1025^3-2049^3) and below. Low end graphics systems have only a pixel depth of 8 bits per channel, which is far too less for sufficient accuracy. Sparse grids of level 12 and above have limited accuracy due to a high component scaling factor — for details see [13]. Additionally, the graphics hardware approach does not scale with current hardware, as the days of most high end graphics suppliers seem to be counted, and no really new systems are available.

2 Sparse Grids

In this section a brief summary of the basic ideas of sparse grids is given. For a detailed survey of sparse grids we refer to [1, 17].

When talking about volume visualization, the data is usually given on a uniform grid with trilinear basis functions. Interpolation on these grids is computationally cheap, as one has only to locate and evaluate the surrounding $2^3 = 8$ basis functions for one interpolation value. This number does not change with respect to the grid size.

Now let G_{i_1,i_2,i_3} be a uniform grid with respective mesh widths $h_{i_j} = 2^{-i_j}$, $j = 1,2,3$ and basis functions $b_k^{i_j}$. Let $\hat{L}_n$ be the function space of the piecewise tri-linear functions defined on $G_{n,n,n}$ and vanishing on the boundary. Additionally, consider the subspaces S_{i_1,i_2,i_3} of $\hat{L}_n$ with $1 \leq i_j \leq n$, $j = 1,2,3$, which consist of the piecewise tri-linear functions defined on G_{i_1,i_2,i_3} and vanishing on the grid points of all coarser grids, with

$$\hat{L}_n = \bigoplus_{i_1=1}^{n} \bigoplus_{i_2=1}^{n} \bigoplus_{i_3=1}^{n} S_{i_1,i_2,i_3} \quad . \tag{1}$$

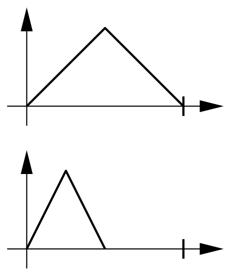

Figure 1: Examples of basis functions b_1^1 and b_1^2

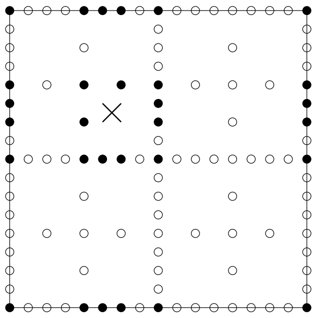

Figure 2: Interpolation on a sparse grid of level 4

This forms a hierarchical basis decomposition of the function space $\hat{L}_n$ where piecewise tri-linear finite elements are used as basis functions in each subspace S_{i_1,i_2,i_3} (compare Fig. 1 for one-dimensional examples). From now on we will deal with the interpolated function f_{i_1,i_2,i_3} on the grids of the above mentioned subspaces. Please note that the basis functions of these subspaces do not overlap, compared to the basis functions of $\hat{L}_n$.

When looking at the interpolation error, one finds that $||f_{i_1,i_2,i_3}||$ has a contribution of the same order of magnitude, namely $O(2^{-2\cdot const})$ for all subspaces with $i_1+i_2+i_3 = const$.

Additionally, these subspaces have the same number of basis functions, namely $2^{const-3}$. Since the number of basis functions is equivalent to the number of stored grid points and because of the contribution argument as well, it seems to be a good idea to define a sparse grid space $\tilde{L}_n$ as follows (compare also Fig. 3):

$$\tilde{L}_n := \bigoplus_{i_1+i_2+i_3 \leq n+2} S_{i_1,i_2,i_3}. \tag{2}$$

An estimation of the interpolation error with regard to the L^2 or L^∞ norm (compare [1, pp. 23]) shows that the sparse grid interpolated function $\tilde{f}_n$ is nearly as good as the full grid interpolated function $\hat{f}_n$.

Now we consider the dimensions of the function spaces $\hat{L}_n$ and $\tilde{L}_n$, which correspond to the number of nodes of the underlying grids. Obviously, the dimension of the full grid space is given by $\dim(\hat{L}_n) = O\left(2^{3n}\right) = O\left(h_n^{-3}\right)$. For the sparse grid the following relation holds: $\dim(\tilde{L}_n) = O\left(2^n n^2\right) = O\left(h_n^{-1}(\log_2(h_n^{-1}))^2\right)$. Therefore, a tremendous amount of memory can be saved if sparse grids are used instead of full grids.

Considering the number of basis functions that contribute to the interpolated function, sparse grids are much more computational intensive than uniform grids. Fig. 2 shows in an example which basis functions have to be evaluated to interpolate the sparse grid at the marked position.

3 Parallelization

A lot of work has been done to implement parallel volume raycasting on PC clusters [9, 10]. By using MPI the parallelization process itself is relatively straight-forward, spreading the rays across the available processors. Memory management is not really an issue, as sparse grids need only very little data space and can thus be replicated throughout the cluster.

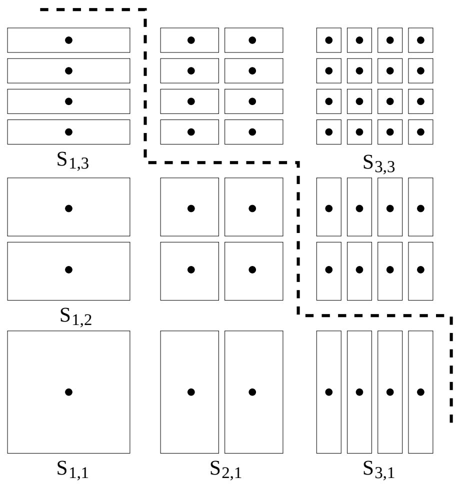

Figure 3: Two-dimensional hierarchical subspace decomposition

A key problem that is noteworthy is that scientists are often unable to work at the front-end nodes of the cluster directly. Thus, the rendered data has to be streamed to the users' workstations. This is done by a dedicated communication node (typically not all nodes have direct internet connection), that collects incoming ray data and serves the TCP stream. As the pixels delivered by the render nodes may come in any order, the communication node sends both pixel position and RGBA values to the workstation, making a total package size of 8 bytes per pixel. With this information the visualization process can continuously generate preview images from early rendered rays.

As the clusters are often shielded by firewalls, ssh tunneling may be required. This seems to be a horrible bottleneck, but in fact the interpolation process on sparse grids is so computational intensive, that slow communication is not hindering the visualization process.

4 Implicit Preview and Load Balancing

The remaining freedom in this process is the distribution of rays among the nodes. Usually, the 'master' node selects by some scheme which node shall render which ray and sends new orders, when a job has been done. However, when several nodes finish their job at the same time, the lag between delivering rays and getting new job data can be quite annoying.

Implicit assignment of rays as a function of the images size $s = s_x \cdot s_y$, the number of processors n, and the rank r (the index of the current processor) prevents any additionally communication overhead and reduces the lag between rendered rays to the time needed to calculate the next ray assignment.

The quality of the ray assignment function has great impact on equalizing the rendering time of the processors as well as on the possibility to generate previews from early rendered rays. By providing previews the user can very often make decisions about the significance of the rendered images, when only a very small fraction of the rays have been computed. So the quality of the ray assignment is reflected in three properties:

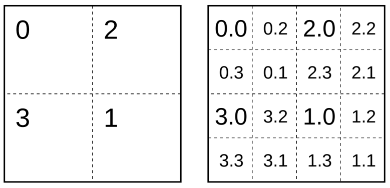

Figure 4: Recursive Pattern for total ordering scheme

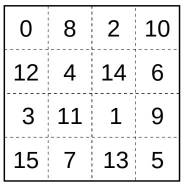

Figure 5: Final index pattern

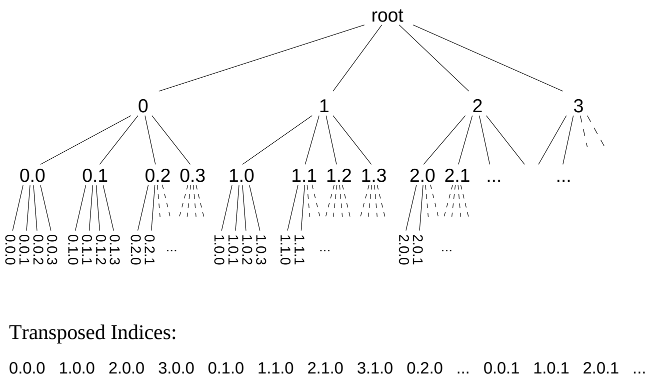

Figure 6: Transposed indexing of the index tree

a The distribution of rays for one processor should be evenly spread in space.

b The distribution of rays should be evenly spread in time.

c Rays that fall in slots of coarser grids should be rendered first.

a) ensures that the load between the processors is stochastically implicitly balanced, while b) and c) ensure that early rendered rays can be combined to form a preview image.

A very simple scheme assigns the rays $\left\{p : r\frac{s}{n} \leq p < (r+1)\frac{s}{n}\right\}$ to rank r. In order to be able to render early previews, one could index the rays in both image dimensions. However, the processors will typically be active for very different times, and the process is only finished after the last processor is done.

By assigning every n$^{\text{th}}$ ray with $\{p : p \bmod n = r\}$ in an interleaving pattern to the nodes one can overcome this problem. However, with this pattern no previews can be generated from early rendered rays.

For ensuring properties b) and c) one can use the pattern in Fig. 4 to generate a total ordering for all rendered rays. The pattern itself is applied recursively, generating a tree of pixel index leafs (Fig. 6). The indices are now transposed so that the highest node index runs fastest. Then continuous numbers are assigned to the leafs, which results in an pixel ordering scheme that can be seen in Fig. 5. In practical implementations the pixel order indices can be generated by a recursive function without explicitly building the tree. For arbitrary image sizes $s_{x,y} \neq 2^m$ the pattern has to be cropped and the indices have to be reordered.

We can now divide the index list into $\frac{s}{n}$ slots S_t of size n:

$$S_t := \{x : tn \leq x < (t+1)n\}$$

We will assign the rays of each slot individually to the processor nodes, so that each processor gets exactly one ray from each slot, which it will render in the order of increasing t.

One can clearly see, that the first $n := 2^i$ rays fill exactly the pixel slots of coarser grid resolutions, as can be seen in Fig. 7. This way we get perfect preview images that can be computed from the rays rendered on coarser grids. As the fastest running index in this scheme corresponds to the lowest resolution grid, adjacent indices are usually not close-by in image space. Thus the temporal distribution of rays is perfectly balanced as well.

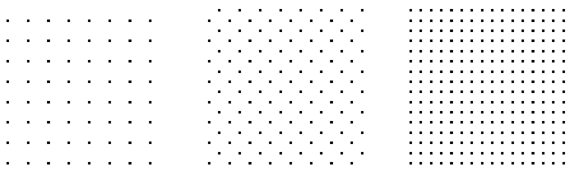

Figure 7: The first 64, 128, 256 rendered rays on an 64^2 image

The selection of ray indices for one processor

$$x_t := tn + i(t,r) \quad \text{with} \quad x_t \in S_t\ ,\ 0 \leq i(t,r) < n$$

has one more freedom to investigate, the index selection function $i(t,r)$. As the index selection should make no differences for the individual processors, we can set

$$i(t,r) := (\tilde{\imath}(t) + r) \bmod n \quad .$$

This way we easily ensure that the set of rays do not intersect.

In order to spread the rays for one processor evenly in space, $\tilde{\imath}$ has to be selected carefully. The very first thought would be to use $\tilde{\imath}(t) = \text{const}$ or $\tilde{\imath}(t) = t$. But both trivial functions do not spread well for all combinations of s and n. Especially when n divides s_x or s_y, the rays cluster on one part of the image. For values that are prime, however, good results can be achieved (see Fig. 10 on the color plate).

Heuristically, we have found an index distribution function that creates very evenly spaced ray selections for almost all combinations of s and n, and the worst cases encountered so far are not as problematic as the ones described above.

Although we do not currently know whether the quality of this index selection function can be proven somehow, it is still a very usable approach in practical implementations as it can be computed iteratively with very low computational costs. Figure 11 shows several distributions created with this approach.

By subdividing the image plane into several tiles, each one a bit larger than n, one can use an iterative algorithm, that counts how many times a ray has already been assigned to the processor on a particular tile, thus ensuring that every tile is at least addressed once by each processor. The most important aspect of this idea is that the algorithm can, again, run separately on each processor without additional communication.

The suggested ray selection method has one drawback that may make it useless for some types of scenes. Cache coherency will not be employed at all using this kind of selection pattern, so memory bound problems may be slower than with tile based approaches. But this is a problem shared by most non-explicit load balancing algorithms.

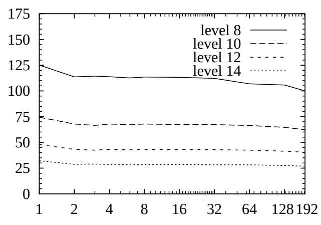

Figure 8: Rendering speed in rays per second and processor vs. number of processors.

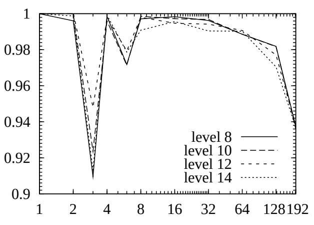

Figure 9: Load balancing quality.

5 Results

The parallelization version has been tested both on a set of workstations with a TCP/IP implementation of MPI (LAM) and on the new PC cluster 'Kepler' [15] of the University of Tübingen. This cluster consists of 96 dual PIII nodes connected with Myrinet, and two additional front-end nodes. The results were streamed to the University of Stuttgart. All rendering times presented here include the communication lag, which off course affects the rendering speedup significantly. The visualization of the incoming ray data is performed in a sparse grid visualization toolkit that effectively hides the parallelization technique from the user.

First, we were interested in the scalability and load balancing quality of our approach. As one can see in Fig. 8, the system scales almost perfectly with the number of processors, as long as the problem is computational bound, and the final TCP streaming is not hindering the rendering process. Load balancing works also extremely well for a system that does not require any additional communication at all. The load balancing presented in Fig. 9 is expressed as the quotient of the rendering time of the fastest and the slowest processor. Note that a bad load balancing has immediate influence on the scaling properties as well.

We found that being able to generate previews completely eliminates the need to reduce the image resolution e.g. for finding good views of the volume. As soon as one is satisfied with image precision, the rendering process is interrupted and a new view can be set. Figure 12 on the color plate shows different stages of this process.

Figures 13 and 14 on the color plate show views of a data set that we were able to render in interactive rates for the very first time. This data set and other following data sets have been computed directly on sparse grids and cannot be expanded to uniform grids due to their size.

6 Conclusion

Parallelization is *the* key feature to create high quality volume visualization images at interactive rates. Due to the nature of sparse grids (small data size, high computational complexity), the parallelization itself is relatively straight-forward, and neither memory consumption nor access times are problematic. In this context implicit load balancing works terrific, and does not imply any additional communication overhead. By using a specialized ray distribution pattern we can additionally create early preview images with no additional cost. We are now able to visualize huge data sets computed directly on sparse grids in high quality in acceptable time for interactive visualization sessions, which was not possible before.

7 Acknowledgements

I would like to thank Vasile Gradinaru from the mathematical institute of Tübingen for his help and the supplied data sets. Peter Leinen from the University of Tübingen made the timing measurements possible by helping us with the new PC cluster 'Kepler', on which our system has been tested.

References

[1] H.-J. Bungartz. *Dünne Gitter und deren Anwendung bei der adaptiven Lösung der dreidimensionalen Poisson-Gleichung*. PhD thesis, Technische Universität München, 1992.

[2] H.-J. Bungartz and T. Dornseifer. Sparse grids: Recent developments for elliptic partial differential equations. Technical report, TU Munich, 1997.

[3] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *Symposium on Volume Visualization*, pages 91–98, October 1994.

[4] V. Gradinaru and R. Hiptmair. Whitney Forms on Sparse Grids. Technical Report 153, University of Tübingen, Department of Mathematics, Tübingen, April 2000. *submitted to Numerische Mathematik.*

[5] M. Griebel, W. Huber, U. Rüde, and T. Störtkuhl. The combination technique for parallel sparse-grid-preconditioning or -solution of pde's on multiprocessor machines and workstation networks. In L. Bougé, M. Cosnard, Y. Robert, and D. Trystram, editors, *Second Joint International Conference on Vector and Parallel Processing*, pages 217–228, Berlin, 1992. CONPAR/VAPP, Springer-Verlag.

[6] M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems. In P. de Groen and R. Beauwens, editors, *International Symposium on Iterative Methods in Linear Algebra*, pages 263–281, Amsterdam, 1992. IMACS, Elsevier.

[7] A. Kaufman. *Volume Visualization*. IEEE Computer Society Press, 1991.

[8] P. Lacroute and M. Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In *Computer Graphics Proceedings*, Annual Conference Series, pages 451–457, Los Angeles, California, July 1994. ACM SIGGRAPH, Addison-Wesley Publishing Company, Inc.

[9] K. L. Ma, J. S. Painter, C. D. Hansen, and M. F. Krogh. A Data Distributed Parallel Algorithm for Ray-Traced Volume Rendering. In *Parallel Rendering Symposium*, pages 15–22, New York, 1993. ACM SIGGRAPH.

[10] M. E. Palmer, S. Taylor, and B. Totty. Exploiting Deep Parallel Memory Hierarchies for Ray Casting Volume Rendering. In *Parallel Rendering Symposium*, pages 15–22, New York, 1997. ACM SIGGRAPH.

[11] C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl. Interactive Exploration of Volume Line Integral Convolution Based on 3D–Texture Mapping. In *Proc. Visualization '99*, pages 233–240. IEEE, 1999.

[12] C. Teitzel, R. Grosso, and T. Ertl. Particle Tracing on Sparse Grids. In D. Bartz, editor, *Proc. 9th Eurographics Workshop on Visualization in Scientific Computing*, pages 132–142, 1998.

[13] C. Teitzel, M. Hopf, and T. Ertl. Volume Visualization on Sparse Grids. *Computing and Visualization in Science*, (2):47–59, 1999.

[14] C. Teitzel, M. Hopf, and T. Ertl. Scientific Visualization on Sparse Grids. In H. Hagen, G. M. Nielson, and F. Post, editors, *Proceedings of Scientific Visualization - Dagstuhl '97*, pages 284–295, Heidelberg, 2000. IEEE Computer Society, IEEE Computer Society Press.

[15] University of Tübingen. *http://kepler.sfb382-zdv.uni-tuebingen.de/*, 2000.

[16] R. Westermann and T. Ertl. Efficiently Using Graphics Hardware in Volume Rendering Applications. *Computer Graphics SIGGRAPH '98*, 32(4):169–179, 1998.

[17] C. Zenger. Sparse grids. In *Parallel Algorithms for Partial Differential Equations: Proceedings of the Sixth GAMM-Seminar*, Kiel, 1990.

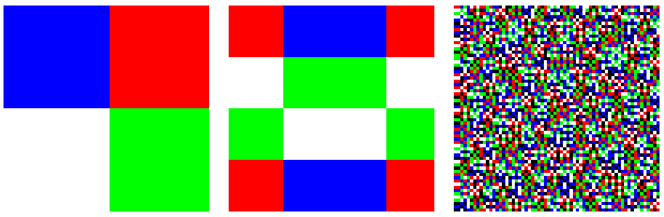

Figure 10: Ray distribution for trivial index selection functions, different processors are encoded with different colors. 64^2 rays, $\tilde{\imath}(t) = const$ on 4 processors, $\tilde{\imath}(t) = t$ on 4 and 5 processors.

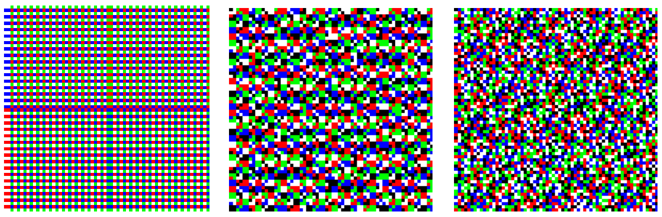

Figure 11: Ray distribution for the heuristic index selection function, 64^2 rays on 4 and 5 processors, 65^2 rays on 5 processors.

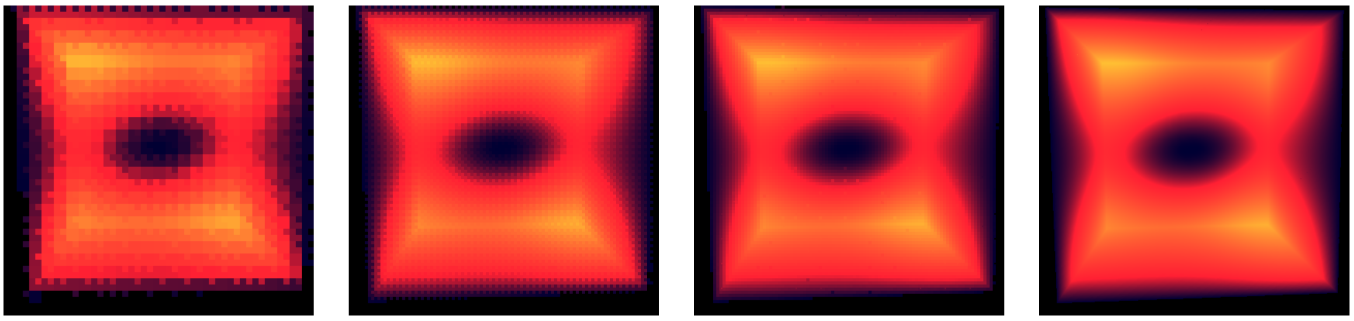

Figure 12: Preview images after 0.625%, 3.125%, 6.25%, and 100% of 160000 rays have been rendered.

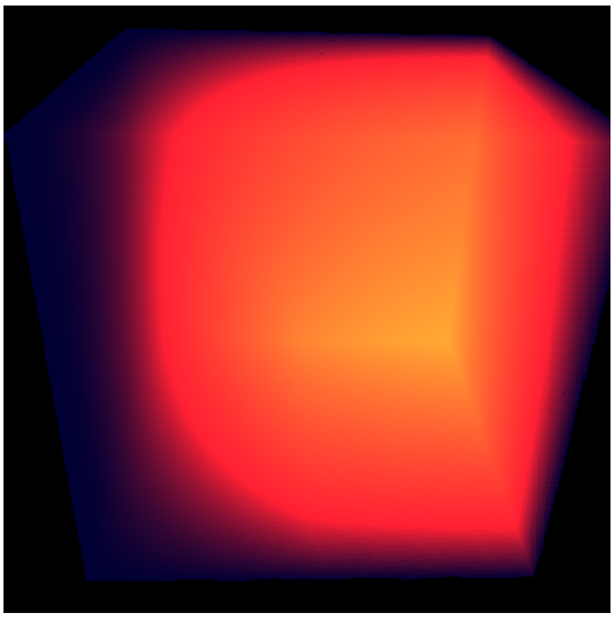

Figure 13: A sparse grid data set of level 12 (corresponding to a full grid of size 2047^3) rendered with an X-ray shading method

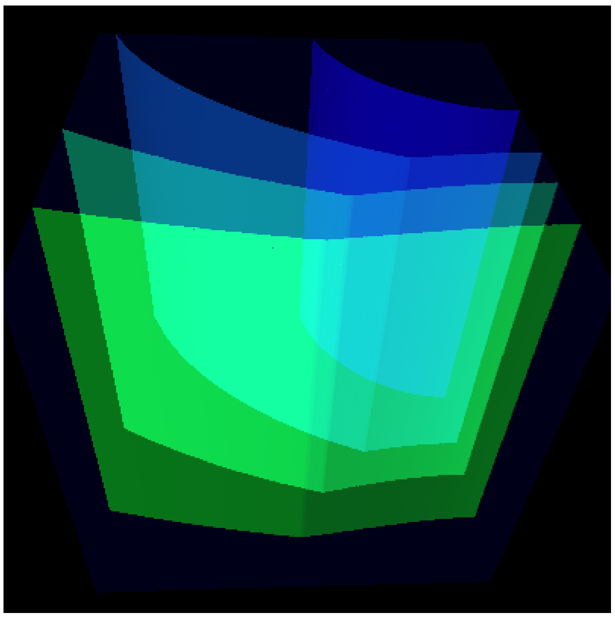

Figure 14: The same data set rendered with multiple semitransparent shaded ISO-surfaces