Particle Tracing on Sparse Grids

Christian Teitzel, Roberto Grosso, and Thomas Ertl

Computer Graphics Group, University of Erlangen
Am Weichselgarten 9, 91058 Erlangen, Germany

Abstract. These days sparse grids are of increasing interest in numerical simulations. 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 differential equations, for instance in numerical fluid mechanics. The particle tracing algorithms that are available so far cannot cope with sparse grids. Now we present an approach that directly works on sparse grids. As a second aspect in this paper, we suggest to use sparse grids as a data compression method in order to visualize huge data sets even on small workstations. Because the size of data sets used in numerical simulations is still growing, this feature makes it possible that workstations can continue to handle these data sets.

1 Introduction

In 1990 sparse grids were introduced by Zenger [10]. With their help it is possible to reduce the total amount of data points or the number of unknowns in discrete partial differential equations. Due to these benefits, sparse grids are more and more used in numerical simulations nowadays [1–4].

On the other hand, it is rather difficult to visualize the results of the simulation process directly on sparse grids, since evaluation and interpolation of function values is quite complicated on such grids. Because of this, up to the present the results of numerical simulations on sparse grids are extrapolated to the associated full grid. Then, all known visualization algorithms on full grids can be performed, e.g. particle tracing, iso-surface extraction, volume rendering, etc.. However, a major drawback of this procedure is the fact that the advantage of low memory consumption of sparse grids comes to nothing using the associated full grid for the visualization step.

Therefore, visualization tools working directly on sparse grids are going to be an important topic of research. Heußer and Rumpf already started working on iso-surface extraction on sparse grids [7]. The first aim of our work is to introduce particle tracing directly on sparse grids (Section 3). Furthermore, a second aspect of this work is the idea that sparse grids can be used for data compression in order to visualize huge data sets on small workstations (Section 4). Additionally, the results of error, time, and memory analyses are listed in Section 4. In order to introduce particle tracing on sparse grids, new methods and classes had to

be developed. This special class hierarchy is described in Subsection 3.1. In Subsection 3.2 we describe the implementation of our sparse grid classes as modules within the framework of the IRIS Explorer visualization environment.

2 Basics of Sparse Grids

In this section a brief summary of the basics of sparse grids is given. For a detailed survey of sparse grids we refer to [1, 10]. In order to make this overview easy to understand and to reduce the number of indices, we describe only three-dimensional grids, whereas the sketches reveal the one- and two-dimensional situations.

Let $f : [0,1]^3 \longrightarrow \mathbf{R}$ be a smooth function defined on the unit cube in $\mathbf{R}^3$ with values in $\mathbf{R}$. Furthermore, f should vanish on the boundary of the cube. This condition is not a strong restriction but is just helpful for an elegant description. Of course, our program can handle three-dimensional functions and even vector fields without zero boundary conditions. If such a function f is stored in the computer memory, then function values at certain positions on a spatial grid are stored in an array. The simplest mesh is a uniform one. Now let G_{i_1,i_2,i_3} be a uniform grid with respective mesh widths $h_{i_j} = 2^{-i_j}$, $j = 1, 2, 3$. On these grids we can introduce the following partial ordering relation: G_{i_1,i_2,i_3} is a refinement of G_{k_1,k_2,k_3} if and only if $k_j \leq i_j$, $j = 1, 2, 3.$ and $k_1 + k_2 + k_3 < i_1 + i_2 + i_3$. Thus we obtain a hierarchy of meshes.

Now 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. Apparently, the hierarchy of grids naturally introduces a hierarchy of subspaces and it follows:

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

Hence, we have found a hierarchical basis decomposition of the function space $\hat{L}_n$. Piecewise tri-linear finite elements are used as basis functions in each subspace S_{i_1,i_2,i_3}. We define the basis functions (Figure 1) of the subspace S_{i_1,i_2,i_3} of $\hat{L}_n$:

$$b_{k_1,k_2,k_3}^{(i_1,i_2,i_3)}(x_1,x_2,x_3) := \prod_{j=1}^{3} w_{i_j}(x_j - m_{k_j}^{(i_j)}) \quad \text{with} \quad m_{k_j}^{(i_j)} = (2k_j - 1) \cdot h_{i_j} \quad ,$$

$$1 \leq k_j \leq 2^{i_j - 1} \quad , \quad \text{and} \quad w_i(x) := \begin{cases} \frac{h_i + x}{h_i} & : \quad -h_i \leq x \leq 0 \\ \frac{h_i - x}{h_i} & : \quad 0 \leq x \leq h_i \\ 0 & : \quad \text{else} \end{cases} \quad .$$

Now we are interested in some estimations of the interpolation error. Hence, let

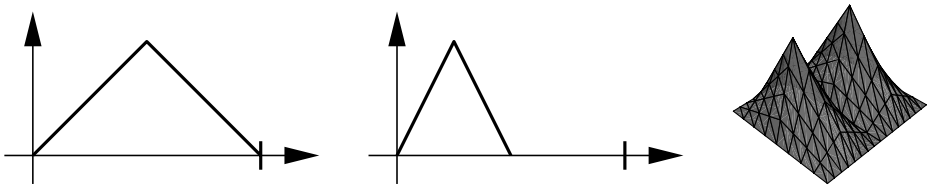


Fig. 1. Examples of basis functions, $b_1^{(1)}$ and $b_1^{(2)}$ on the left and $b_{1,1}^{(1,2)}$ and $b_{1,2}^{(1,2)}$ on the right hand side.

$\hat{f}_n \in \hat{L}_n$ be the interpolated function on the grid $G_{n,\ldots,n}$. Then, $\hat{f}_n$ is given by

$$\hat{f}_n = \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \sum_{i_3=1}^{n} f_{i_1,i_2,i_3} \quad \text{where} \quad f_{i_1,i_2,i_3} = \sum_{k_1=1}^{2^{i_1-1}} \sum_{k_2=1}^{2^{i_2-1}} \sum_{k_3=1}^{2^{i_3-1}} c_{k_1,k_2,k_3}^{(i_1,i_2,i_3)} \cdot b_{k_1,k_2,k_3}^{(i_1,i_2,i_3)}.$$

The values $c_{k_1,k_2,k_3}^{(i_1,i_2,i_3)}$ are called contribution coefficients and $f_{i_1,i_2,i_3} \in S_{i_1,i_2,i_3}$ is a linear combination of the basis functions of the appropriate subspace. It can be shown that the following estimations hold with regard to the L^2 and L^∞ norms (compare [1, pp. 13]):

$$\|f_{i_1,i_2,i_3}\|_2 \leq \frac{1}{27} \left\| \frac{\partial^6 f}{\partial x_1^2 \partial x_2^2 \partial x_3^2} \right\|_2 \cdot h_{i_1}^2 h_{i_2}^2 h_{i_3}^2 \quad , \tag{1}$$

$$\|f_{i_1,i_2,i_3}\|_\infty \leq \frac{1}{8} \left\| \frac{\partial^6 f}{\partial x_1^2 \partial x_2^2 \partial x_3^2} \right\|_\infty \cdot h_{i_1}^2 h_{i_2}^2 h_{i_3}^2 \quad , \tag{2}$$

$$\left\| f - \hat{f}_n \right\|_2 \leq \frac{1}{243} \left\| \frac{\partial^6 f}{\partial x_1^2 \partial x_2^2 \partial x_3^2} \right\|_2 \cdot h_n^2 = O\left(h_n^2\right) \quad , \tag{3}$$

$$\left\| f - \hat{f}_n \right\|_\infty \leq \frac{1}{72} \left\| \frac{\partial^6 f}{\partial x_1^2 \partial x_2^2 \partial x_3^2} \right\|_\infty \cdot h_n^2 = O\left(h_n^2\right) \quad . \tag{4}$$

So far we have just dealt with regular uniform meshes, which are named full grids. Now let us turn to sparse grids. Consider the subspaces S_{i_1,i_2,i_3} with $i_1 + i_2 + i_3 = const$. Equations (1) and (2) show that $\|f_{i_1,i_2,i_3}\|_\infty$ and $\|f_{i_1,i_2,i_3}\|_2$ have 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:

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

Now the interpolated function $\tilde{f}_n \in \tilde{L}_n$ is given by

$$\tilde{f}_n = \sum_{i_1+i_2+i_3 \leq n+2} f_{i_1,i_2,i_3} \tag{5}$$

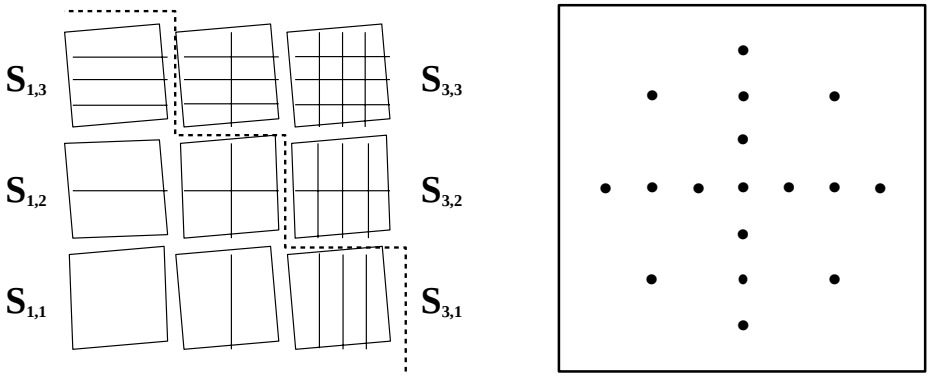


Fig. 2. On the left hand side a two-dimensional hierarchical subspace decomposition is shown and on the right hand side you can see the respective sparse grid.

and the interpolation errors with regard to the L^2 and L^∞ norms are given by (compare [1, pp. 23])

$$\left\| f - \tilde{f}_n \right\|_2 \leq \left(1 + \frac{3}{4}n + \frac{9}{16}\binom{n+1}{2}\right) \frac{h_n^2}{9^3} \left\| \frac{\partial^6 f}{\partial x_1^2 \partial x_2^2 \partial x_3^2} \right\|_2 \tag{6}$$

$$= O\left(h_n^2 \left(\log_2\left(h_n^{-1}\right)\right)^2\right), \tag{7}$$

$$\left\| f - \tilde{f}_n \right\|_\infty \leq \left(1 + \frac{3}{4}n + \frac{9}{16}\binom{n+1}{2}\right) \frac{h_n^2}{6^3} \left\| \frac{\partial^6 f}{\partial x_1^2 \partial x_2^2 \partial x_3^2} \right\|_\infty \tag{8}$$

$$= O\left(h_n^2 \left(\log_2\left(h_n^{-1}\right)\right)^2\right). \tag{9}$$

These estimations show 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

$$dim\left(\hat{L}_n\right) = O\left(2^{3n}\right) = O\left(h_n^{-3}\right). \tag{10}$$

For the sparse grid the following equation holds:

$$dim\left(\tilde{L}_n\right) = O\left(2^n \cdot n^2\right) = O\left(h_n^{-1}\left(\log_2\left(h_n^{-1}\right)\right)^2\right). \tag{11}$$

Therefore, a tremendous amount of memory is saved if sparse grids are used instead of full grids (see Section 4).

If the function f is given and a certain accuracy is required, then it is possible to use $\hat{f}_n \in \hat{L}_n$ or $\tilde{f}_m \in \tilde{L}_m$ where m is just slightly greater than n. Due to the very low memory consumption of sparse grids, it is better to use the function $\tilde{f}_m$. On the other hand the function f is often given in discrete form as data set

on a full grid. In this case it is not possible to reach a better accuracy with the sparse grid approach than with the original full grid data. However, equations (7), (9), and (11) show that a very small loss of accuracy is rewarded with a huge amount of saved storage.

3 Particle Tracing on Sparse Grids

Flow visualization tools based upon particle methods continue to be an important utility of flow simulation. Additionally, the importance of sparse grids in numerical simulations is still growing. However, so far particle tracing algorithms could only handle data sets given on full grids. Now we present a particle tracer that can cope with sparse grids. Our new particle tracing module supplies the same features, e.g. colored streak lines, ribbons, tubes, balls, and tetrahedra (see Figure 3), as our previous full grid particle tracing tool, which is partially described in [5] and [6].

3.1 Class Hierarchy for Efficient Interpolation on Sparse Grids

Lagrange visualization techniques of a vector field v are based upon the numerical solution of an initial value problem for the differential equation: $dx/dt = v(x,t)$. Usually, a numerical integration method is used to obtain a solution. All such methods have in common that they must evaluate the vector field v at certain positions, which are in general not at grid points. Therefore, the value of v at such a position has to be interpolated. As mentioned in Section 2, this interpolation on sparse grids is different from that one on full grids, whereas the other parts of the particle tracing algorithm can remain unchanged.

In contrast to the tri-linear full grid interpolation, the sparse grid interpolation does not operate locally, because one basis function in every subspace contributes to the function value. Since the tri-linear interpolation is one of the most time consuming operations during the particle tracing process on full grids [8], the complicated sparse grid interpolation is all the more time consuming. Therefore, it is important to execute the interpolation as fast as possible.

Normally, the contribution coefficients of the sparse grid are stored in a binary tree [1, 2, 7]. Then, a recursive tree traversal has to be performed in order to interpolate the function value. This tree traversal is very slow. Although caching strategies can increase the efficiency of the traversal [7], the computation of the values remains rather time consuming.

Hence, the contribution values are not stored in a binary tree but in arrays. Then, it is not necessary to traverse a tree but the required contribution coefficient can be accessed directly. Therefore, we have implemented a particular C++ class hierarchy. Due to the limited amount of space, we can just give a very brief idea of the classes.

Initially, recall that the sparse grid space $\tilde{L}_n$ is the direct sum of all subspaces $S_{i,j,k}$ with $i+j+k \leq n+2$. Now we define the *level of a subspace* as the number $n = i + j + k - 2$. Moreover, we define a *level of the sparse grid space* as the

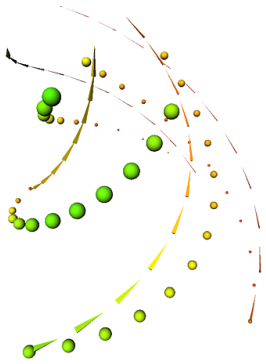


Fig. 3. Colored streak balls and tetrahedra in a vortex flow given on a sparse grid.

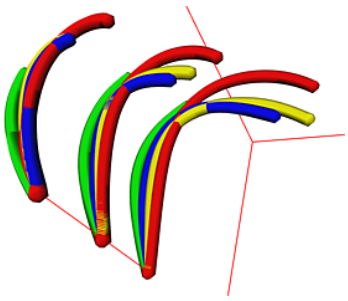


Fig. 4. Streak tubes in a cavity flow; the red tubes are computed on a full grid of level 7, the other tubes are created on sparse grids of level 7 (yellow), 5 (blue), and 3 (green).

direct sum of all subspaces of the same level of subspaces. Therefore, $\tilde{L}_n$ is the direct sum of its first n levels and is called a *sparse grid of level n*.

Besides abstract base classes, classes for input, and other auxiliary classes, the classes of interest are named `hbSparseGrid`, `hbLevel`, and `hbSubspace`. The class `hbSparseGrid` contains a stack of n levels of class `hbLevel`. Furthermore, `hbLevel` comprises the respective number of subspaces ($(n+1)n/2$), denoted `hbSubspace`. The class `hbSubspace` contains an array of the size 2^{n-1} times data dimension, where the contribution coefficients are stored. The function value at an arbitrary position is computed by means of formula (5). In order to compute a function value, the class `hbSparseGrid` contains a method `calcValue(...)`. This method sends a '`calcValue()`' to each `hbLevel` to accumulate the contributions to the resulting value. Then, the method `hbLevel::calcValue(...)` performs a loop over all subspaces of the current level. In this loop, the required basis function is determined by means of the coordinates of the current position. Recall that only one basis function per subspace is unequal to zero at a certain position because all basis functions are hat-functions. Hence, we know the required contribution value. Now the 'height' over the current position in the tri-linear hat-function is determined and multiplied with the contribution value. Thus, we obtain the total contribution of this subspace to the function value. Additionally, we compute the Jacobian, which is needed to compute the local rotation of the flow for displaying bands and tetrahedra, in this loop by looking up the correct 'height' of the derivative of the hat-function, a simple box-function. The efficiency of this implementation is shown in Section 4.

3.2 Implementation as IRIS Explorer Module

Our new particle tracer, which works on data sets given on sparse grids, is implemented as an IRIS Explorer module and named `StreakbandHB`. As integration

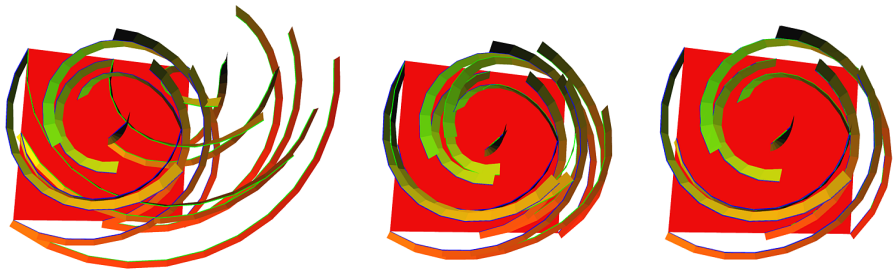


Fig. 5. Streak bands in a vortex flow; ribbons containing blue edges display the flow on a full grid of level 7, bands with green edges the flow on sparse grids of level 0 (left), 1 (middle), and 4 (right); the ribbons computed on full and sparse grids coincide on screen for levels greater than 3.

methods for the particle tracing algorithm of `StreakbandHB`, we use the integration schemes that we have already implemented in our full grid particle tracer, called `Streakband`. A comparison of these schemes can be found in [9]. An adaptive Runge-Kutta method of order 3 (RK3(2)) is used for the tests described in Section 4.

In order to visualize the particles, we have chosen the same geometrical primitives as in our full grid particle tracing module, namely lines, bands, tubes, balls, and tetrahedra. Of course, all kinds of traces can visualize an additional scalar value by means of color coding. Moreover, balls and tetrahedra can reveal another scalar value by their size. Besides that, bands and tetrahedra display the local vorticity of the flow via rotating around the actual streak line. Since both modules, `Streakband` and `StreakbandHB`, are provided with the same functionality, their results can be compared easily (see Section 4).

Besides the actual particle tracer, some additional modules had to be implemented in order to handle sparse grids properly. First of all, a module, called `DemoSparseGridHB`, is needed to create an analytical demo vector field on a sparse grid of a certain level. Secondly, a function, denoted `LatToSparseGridHB`, is used in order to transfer a full grid given as Explorer `cxLattice` data type to a sparse grid. Finally, `PrintSparseGridHB` is a helpful tool for debugging sparse grid routines.

In order to allow these new modules sending and receiving sparse grid data via the Explorer network, a new Explorer data type has been created, named `HBSparseGrid3D`.

4 Results

In order to compare our sparse grid particle tracing module with full grid particle tracers, two data sets were used. The first one, which was provided by S. H. Enger from the Lehrstuhl für Strömungsmechanik of the University of Erlangen, is a cavity flow data set on a full grid of level 7, i.e. 129^3 nodes (see Figure 4). The

data set contains the velocity, pressure, and temperature at each vertex. Hence, it consumes more than 40 MB. Notice that the same data set with a resolution of 8 levels would need more than 320 MB, that is too much for most workstations. On the other hand, this data set stored on a sparse grid of level 7 consumes only 175 kB.

The second data set is an analytic one. It is a vortex flow (compare Figures 3 and 5). Since the data set is analytical, we are able to create sparse and full grids in any resolution only limited by the main memory of the used machine. Therefore, we chose the analytic vector field for our quantitative efficiency tests. Nevertheless, the performance of the compared modules was nearly the same while testing on the cavity data set.

All tests were performed on a Silicon Graphics computer with a 196 MHZ R10000 processor. For testing, at each time nine streak ribbons were computed consisting of about 500 particles (see Figure 5). The computing time of `StreakbandHB` is compared with that of our full grid `Streakband` module and of the `NAG-Advect` module, which is provided together with the IRIS Explorer. The CPU-times were measured in seconds and are listed in the following table.

level	2	3	4	5	6	7
points of full grid	5^3	9^3	17^3	33^3	65^3	129^3
StreakbandHB (sparse grid)	0.15 s	0.28 s	0.47 s	0.95 s	1.65 s	4.61 s
Streakband (full grid)	0.42 s	0.87 s	1.60 s	3.22 s	6.51 s	13.26 s
NAG-Advect (full grid)	1.09 s	1.33 s	1.61 s	1.89 s	2.28 s	2.66 s

Table 1. Computing times in CPU-seconds using an analytic vortex flow.

The used integration methods were an adaptive Runge-Kutta scheme RK3(2) in case of our `Streakband` modules and an adaptive Runge-Kutta scheme RK4(5) in case of the `NAG-Advect` program. See [9] for a discussion of different integration algorithms for particle tracing.

At first glance, it is astonishing that the full grid `Streakband` module is slower than our sparse grid `StreakbandHB` module. This is due to the fact that `Streakband` is adjusted to multi-block curvilinear grids. In order to cope with such grids, the stencil walk algorithm is performed during the particle tracing. This algorithm is unnecessary on uniform grids and therefore not performed by `StreakbandHB`. Thus, it is not fair to compare the computational times of those modules, but anyway the full grid `Streakband` is needed for the comparison of the actual particle traces.

The measured times show that interactive particle tracing is possible even on sparse grids of level 7. Secondly, the table reveals the drawback of sparse grid interpolation that the computing time exponentially rises if the level of the grid is increased. In contrast to this, the computing time of the `NAG-Advect` module is growing slowly. In theory, the time for particle tracing on full grids is independent of the grid size.

Now the accuracy of sparse grid particle tracing is considered. Therefore, the traces computed by `StreakbandHB` are compared with their counterparts resulting from `Streakband`. Recall that the error of full grid interpolation can be estimated at $O(h^2)$ and that of sparse grid interpolation at $O((h \cdot \log_2(h^{-1}))^2)$. This is a rather small difference. Moreover, the integration error of RK3(2) is of order $O(\tau^3)$ where τ denotes the current time step [9]. From this point of view, it does not seems to be too bad using sparse instead of full grid particle tracing. In fact, the results of particle tracing on the analytic data set confirm these estimations because the ribbons computed on full and sparse grids coincide on screen for levels greater than 3 (compare Figure 5).

However, during the deduction of the mentioned upper bounds of the interpolation errors, the smoothness of the data was needed (compare equations (3), (4), (6), and (8)). Since discrete data sets are not smooth at all, these estimations do not hold in case of discrete data. Indeed, Figure 4 reveals that the particle traces computed on sparse grids converge rather slowly to the full grid solution. Nevertheless, due to the great advantage of low memory consumption, it is possible to use a sparse grid of quite a high level to overcome this problem.

The great benefit of the sparse grid technique is the low number of required grid points. The next table shows the memory consumption of a typical data set resulting from a numerical flow simulation. Assume that five floating point values, namely three velocity components, pressure, and temperature, are given at each grid node. Then, these floating point values add up to 20 bytes per node. Thus, we obtain the following results:

level	5	6	7	8	9	10
points of full grid	33^3	65^3	129^3	257^3	513^3	1025^3
sparse grid	29 kB	73 kB	175 kB	415 kB	970 kB	2.2 MB
full grid	640 kB	5 MB	40 MB	320 MB	2.5 GB	20 GB

Table 2. Memory consumption of a typical data set.

This table shows that sparse grids are very suitable for compressing huge data sets. By dint of this, it is possible to visualize such data even on small workstations.

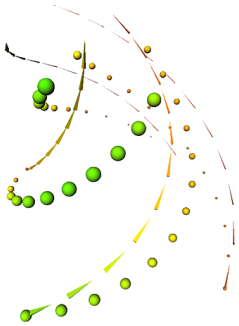
5 Conclusion

We have introduced particle tracing on sparse grids. This allows to carry out flow visualization directly on sparse grids without transforming the results of numerical simulations on sparse grids to the associated full grids. Secondly, the sparse grid approach can be used as a compression method in order to realize particle tracing in huge data sets on small workstations.

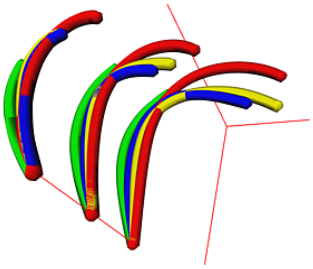
There are several directions of future work. The first aim is to introduce further visualization techniques on sparse grids. First of all, we are going to introduce volume rendering on these grids. A second goal is to enlarge the field of applications. At the moment, we are thinking about a particle tracing algorithm on curvilinear sparse grids. Furthermore, we intend to implement adaptive sparse grids with error monitoring. Last but not least, there are possibilities to accelerate the sparse grid interpolation by sophisticated caching strategies. On the one hand, a pre-computing mechanism of a certain number of levels could be implemented. On the other hand, pre-computing a certain number of cells could be advantageous.

References

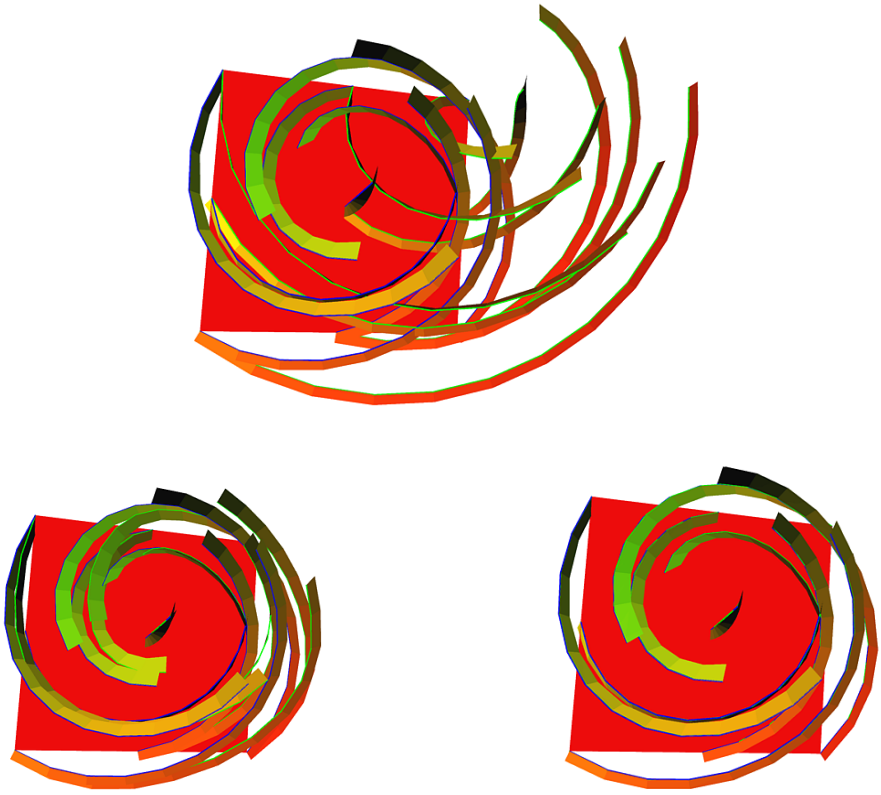
1. H.-J. Bungartz. *Dünne Gitter und deren Anwendung bei der adaptiven Lösung der dreidimensionalen Poisson-Gleichung.* PhD thesis, TU Munich, 1992.
2. H.-J. Bungartz and T. Dornseifer. Sparse grids: Recent developments for elliptic partial differential equations. Technical report, TU Munich, 1997.
3. 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.
4. 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.
5. R. Grosso, M. Schulz, and T. Ertl. Fast and accurate visualization of steady and unsteady flows. Technical Report 3, University of Erlangen, 1996.
6. R. Grosso, M. Schulz, J. Kraheberger, and T. Ertl. Flow visualization for multiblock multigrid simulations. In *Virtual Environments and Scientific Visualization '96*, Heidelberg, 1996. Springer-Verlag.
7. N. Heußer and M. Rumpf. Efficient visualization of data on sparse grids. In H.-C. Hege and K. Polthier, editors, *Visualization and Mathematics*, Berlin. Springer-Verlag. In preparation.
8. D. N. Kenwright and D. A. Lane. Optimization of Time-Dependent Particle Tracing Using Tetrahedral decomposition. In G. M. Nielson and Silver D., editors, *Visualization '95*, pages 321–328, Los Alamitos, CA, 1995. IEEE Computer Society, IEEE Computer Society Press.
9. C. Teitzel, R. Grosso, and T. Ertl. Efficient and Reliable Integration Methods for Particle Tracing in Unsteady Flows on Discrete Meshes. In W. Lefer and M. Grave, editors, *Visualization in Scientific Computing '97*, pages 31–41, Wien, April 1997. Springer-Verlag. Proceedings of the Eurographics Workshop in Boulogne-sur-Mer, France.
10. C. Zenger. Sparse grids. In *Parallel Algorithms for Partial Differential Equations: Proceedings of the Sixth GAMM-Seminar*, Kiel, 1990.



Colored streak balls and tetrahedra in a vortex flow given on a sparse grid (Teitzel et al., Fig. 3)



Streak tubes in a cavity flow; the red tubes are computed on a full grid of level 7, the other tubes are created on sparse grids of level 7 (yellow), 5 (blue), and 3 (green) (Teitzel et al., Fig. 4)



Streak bands in a vortex flow; ribbons containing blue edges display the flow on a full grid of level 7, bands with green edges the flow on sparse grids of level 0 (top), 1 (left), and 4 (right); the ribbons computed on full and sparse grids coincide on screen for levels greater than 3 (Teitzel et al., Fig. 5)