Volume Ray Casting on Sparse Grids

Christian Teitzel, Matthias Hopf, Roberto Grosso, and Thomas Ertl*

Computer Graphics Group, University of Erlangen†

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 partial differential equations. The volume visualization 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 workstations with low main memory. 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. Besides the standard sparse grid interpolation algorithm and the so called combination approach, we have developed a new sparse grid interpolation method, which harnesses the texture-mapping hardware of Silicon Graphics workstations for accelerating purposes. Therefore, hardware based volume rendering becomes possible on compressed data sets at interactive frame rates.

INTRODUCTION

In 1990 sparse grids were introduced by Zenger [17]. 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, 2, 5, 6].

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. 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. Recently, Heußer and Rumpf presented an algorithm for iso-surface extraction on sparse grids [8]. In a previous work, we introduced particle tracing on uniform sparse grids [15]. The first aim of this paper is to present volume ray casting directly on sparse grids. Furthermore, a second aspect of our work is the idea that sparse grids can be used for data compression in order to visualize huge data sets even on workstations with a limited amount of main memory. Moreover, using the combination technique, which is presented below, it is possible to decompose a sparse grid into a certain number of uniform full grids of low resolution. Because of this, Silicon Graphics' texture hardware can be deployed for the necessary function interpolation. Hence, we are able to perform volume visualization methods on compressed data sets at interactive frame rates. This is not possible if other compression methods like wavelet or fractal compression are used. However, we are able to handle sparse grids of level 10, which correspond to volumes of 1025^3 voxels. Below the results of memory, time, and error analyses are listed in detail.

In order to introduce volume ray casting on sparse grids, new methods and classes had to be developed. This special class hierarchy and the implementation of our volume ray caster are described as well.

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, 17]. 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 $k_j \le i_j$, $j = 1,2,3$. 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 $1 \le i_j \le n$, $j = 1,2,3$. 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}. Then we define the basis functions of the subspace S_{i_1,i_2,i_3} of $\hat{L}_n$:

$$b^{(i_1,i_2,i_3)}_{k_1,k_2,k_3}(x_1,x_2,x_3) := \prod_{j=1}^{3} w_{i_j}(x_j - m^{(i_j)}_{k_j})$$

$$\text{with} \quad m^{(i_j)}_{k_j} = (2k_j - 1)\cdot h_{i_j} \quad , \quad 1 \le k_j \le 2^{i_j - 1} \quad ,$$

$$\text{and} \quad w_i(x) := \begin{cases} \frac{h_i + x}{h_i} & : \quad -h_i \le x \le 0 \\ \frac{h_i - x}{h_i} & : \quad 0 \le x \le h_i \\ 0 & : \quad \text{else} \end{cases} \quad .$$

*E-mail: {teitzel,mshopf,grosso,ertl}@informatik.uni-erlangen.de.
†URL: http://www9.informatik.uni-erlangen.de, Am Weichselgarten 9, 91058 Erlangen, Germany.

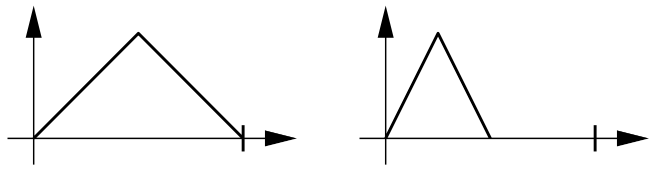

Figure 1: Examples of basis functions, $b_1^{(1)}$ on the left and $b_1^{(2)}$ on the right hand side.

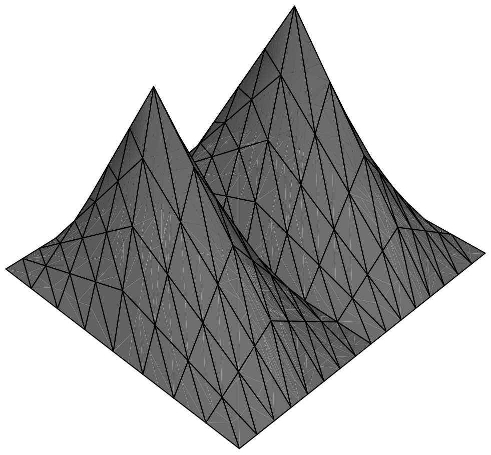

Figure 2: Examples of basis functions, $b_{1,1}^{(1,2)}$ and $b_{1,2}^{(1,2)}$.

Now we are interested in some estimations of the interpolation error. Hence, let $\hat{f}_n \in \hat{L}_n$ be the interpolated function on the grid $G_{n,\cdots,n}$. Then $\hat{f}_n$ is given by

$$\hat{f}_n = \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \sum_{i_3=1}^{n} f_{i_1,i_2,i_3}$$

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

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]):

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

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

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

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

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

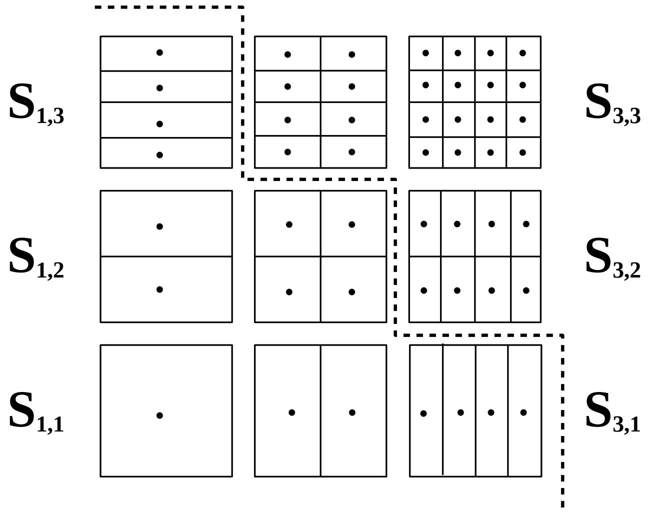

Figure 3: Two-dimensional hierarchical subspace decomposition.

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

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

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

$$\left\|f - \tilde{f}_n\right\|_2 \leq c(n) \cdot \frac{h_n^2}{9^3} \cdot \left\| \frac{\partial^6 f}{\partial x_1^2 \partial x_2^2 \partial x_3^2} \right\|_2 \quad (6)$$

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

$$\left\|f - \tilde{f}_n\right\|_\infty \leq c(n) \cdot \frac{h_n^2}{6^3} \cdot \left\| \frac{\partial^6 f}{\partial x_1^2 \partial x_2^2 \partial x_3^2} \right\|_\infty \quad (8)$$

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

$$\text{with} \quad c(n) := \left(1 + \frac{3}{4}n + \frac{9}{16}\binom{n+1}{2}\right). \quad (10)$$

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

$$dim\left(\hat{L}_n\right) = O\left(2^{3n}\right) = O\left(h_n^{-3}\right). \quad (11)$$

For the sparse grid the following equation holds:

$$dim\left(\tilde{L}_n\right) = O\left(2^n \cdot n^2\right) = O\left(h_n^{-1}\left(\log_2\left(h_n^{-1}\right)\right)^2\right). \quad (12)$$

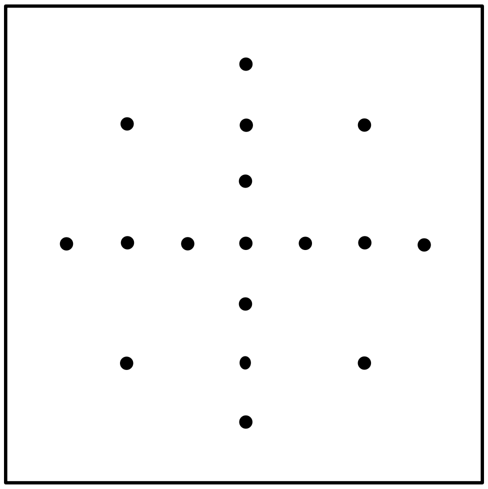

Figure 4: Two-dimensional sparse grid of level 3.

Therefore, a tremendous amount of memory is saved if sparse grids are used instead of full grids.

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 (12) show that a very small loss of accuracy is rewarded with a huge amount of saved storage.

Finally, 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 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*.

COMBINATION TECHNIQUE

Since the described sparse grid interpolation of function values is quite complicated and rather time consuming, we have implemented the so called combination technique. This method was introduced by Griebel, Schneider, and Zenger in 1992 [6]. Actually, the combination method has been used in numerical simulations in order to combine partial solutions computed on smaller, suitable full grids to the wanted sparse grid solution. However, we start with a data set given on a sparse grid and decompose the grid so that the data set is represented on certain uniform full grids of low resolution. Now the quick and easy tri-linear interpolation can be performed on each of these full grids. The resulting value is computed by linear combination of the tri-linear interpolated full grid results.

Going into details, it can be proofed that the two-dimensional interpolated function $\tilde{f}_n \in \tilde{L}_n$ is given by

$$\tilde{f}_n = \sum_{i_1+i_2=n+1} f^c_{i_1,i_2} - \sum_{i_1+i_2=n} f^c_{i_1,i_2} \tag{13}$$

where $f^c_{i_1,i_2}$ denotes the bi-linear interpolation of function values on the respective full grid. Figure 5 reveals the two-dimensional situation. Then, it is easily shown that the three-dimensional interpolated function $\tilde{f}_n \in \tilde{L}_n$ can be computed in a similar way:

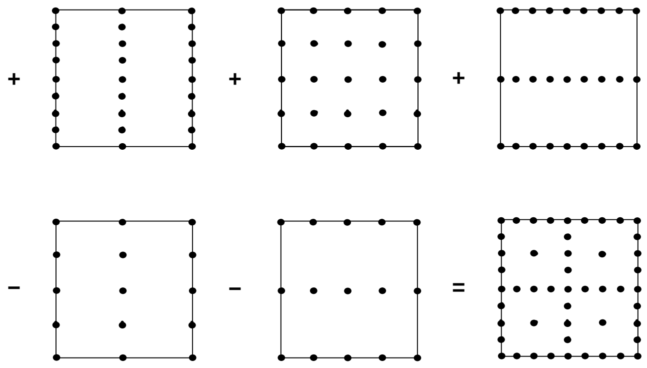

Figure 5: A two-dimensional sparse grid of level 3 can be reconstructed by linear combination of five full grids of low resolution.

$$\begin{aligned}\tilde{f}_n \quad = \quad & \sum_{i_1+i_2+i_3=n+2} f^c_{i_1,i_2,i_3} \\ & - \quad 2 \quad \cdot \sum_{i_1+i_2+i_3=n+1} f^c_{i_1,i_2,i_3} \\ & + \sum_{i_1+i_2+i_3=n} f^c_{i_1,i_2,i_3}\end{aligned} \tag{14}$$

where $f^c_{i_1,i_2,i_3}$ denotes the tri-linear interpolation of function values on the respective full grid. Notice that the used full grids consist of the same nodes as the corresponding sparse grid.

Now let us turn to the benefit of the combination technique. The total number of summands of the standard sparse grid interpolation on a three-dimensional sparse grid of level n is given by

$$\sum_{i=1}^{n} \frac{i(i+1)}{2} = \frac{1}{6} n(n+1)(n+2) \tag{15}$$

(compare equation (5)), whereas the total number of tri-linear interpolations of the combination method adds up to

$$\sum_{i=n-2}^{n} \frac{i(i+1)}{2} = \frac{3}{2} n(n-1)+1 \tag{16}$$

in the three-dimensional case (see equation (14)).

However, the main advantage of the combination technique is the fact that uniform full grids are used. Thus, it has been possible to exploit the texture hardware of Silicon Graphics workstations for the interpolation of function values. Hereby, we are able to use volume ray casting on sparse grids interactively.

IMPLEMENTATION

Volume rendering continues to be an important utility of displaying volumetric three-dimensional scalar data sets resulting from measurement or simulation. Additionally, the importance of sparse grids in numerical simulations is continuously growing. However, so far volume visualization algorithms could only handle data sets given on full grids. Here we present a volume ray caster that can cope with sparse grids, i.e. our tool directly works on compressed data sets.

In this section we are going to describe the implementation of our volume ray caster. The first subsection presents the different ways of implementing the needed interpolation routine. In the second subsection the used lighting models and transfer functions are described.

The graphical user interface of our program was developed using RapidApp, an interactive tool for creating applications [13]. RapidApp's predefined interface components facilitate the use of OpenGL and Open Inventor.

Our new interpolation routines on sparse grids and the adapted volume visualization methods have been implemented in an object-oriented manner as C++ classes.

Interpolation on Sparse Grids

All volume ray casting methods have in common that they must evaluate the function f at certain positions, which are in general not at grid points. Therefore, the value of f at such a position has to be interpolated. As mentioned above, this interpolation on sparse grids is different from that one on full grids.

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 complicated sparse grid interpolation is one of the most time consuming operations during the volume rendering process on sparse grids, it is important to execute the interpolation as fast as possible.

Using Improved Sparse Grid Interpolation

Normally, the contribution coefficients of the sparse grid are stored in a binary tree [1, 2, 8]. 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 [8], the computation of the values remains rather time consuming.

In order to avoid the tree traversal and to accelerate the access to the contribution coefficients, we have developed a new, very efficient data structure based upon arrays. 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.

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 with disjunct supports. 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 gradient, which is needed for the illumination of iso-surfaces, in this loop by looking up the correct 'height' of the derivative of the hat-function, a simple box-function.

Using the Combination Technique

Since the low resolution full grids needed by the combination method are uniform grids, the function values can be stored in three-dimensional arrays. Thus, the `calcValue(...)` method of the according derived class `hbCombinationSparseGrid` can address the necessary function values in a tight loop. This fact and the smaller number of required basic arithmetic operations makes the combination technique an order of magnitude faster than the previously described sparse grid interpolation even for low levels.

Employment of Texture Hardware

OpenGL is a graphics standard introduced by Silicon Graphics and now maintained by the OpenGL Architecture Review Board (ARB). OpenGL contains routines for texture mapping, using special graphics hardware whenever possible. Several vendors added some extensions for three-dimensional so called volume textures. By using these extensions our class `hbOglSparseGrid` is able to exploit the hardware in SGI's graphics systems *Reality Engine* and *Maximum Impact*, which are able to perform the necessary tri-linear interpolation for volume texturing in hardware. For more information about the extensions and Silicon Graphics hardware see [12].

Because single point interpolations are rather inefficient — several functions have to be called, the hardware has to be initialized, results have to be fetched back — the ray caster was rewritten so that it renders all rays simultaneously. Now the texture hardware can be utilized to draw a complete plane textured with the volume data. According to formula (14), several values have to be added and subtracted for each sparse grid interpolation. For each of these interpolation functions, a plane is drawn with a certain volume texture representing the function. Then the occurring planes are composed using the `GL_FUNC_ADD_EXT` and `GL_FUNC_REVERSE_SUBTRACT_EXT` blending extensions of OpenGL.

In order to switch quickly between the different volume textures, they all have to fit into texture memory at the same time. By using the texture-name extension of OpenGL, pre-loaded textures can be selected for rendering almost instantly. Recall that the combination technique uses full grids of size $(2^i+1)\times(2^j+1)\times(2^k+1)$. On the other hand, volume textures have to be of size $2^p\times 2^q\times 2^r$. Hence, quite a lot of texture memory is wasted, which is a scarce resource.

In principle, OpenGL supports volume textures featuring a so called border of size 2 in every direction so that the mentioned full grids would fit almost seamlessly. Unfortunately, these texture borders are not implemented in todays hardware.

When using hardware for mathematical computations, accuracy can be quite a problem. On Silicon Graphics machines the blending operation can be performed with 12 bits at most. Since pixel values are automatically clamped to values in the interval [0, 1], all texture elements have to be scaled down by the number of functions contributing positive values. For a level 10 sparse grid, 91 of 136 functions contribute positive data, which means a loss of almost 7 bits, resulting in only 5 bits of accurate information. Since we have at best 12 bits of accurate information in the color-buffer, it is sufficient to use only 2 bytes of texture memory per voxel. Although visible artifacts are remarkably small, some can be seen in the video and color plates (compare Figures (8), (9), and (10)).

Ray Casting on Sparse Grids

In this section the used volume visualization algorithms are described briefly. For a detailed presentation of volume rendering techniques we refer to [3, 4, 7, 9, 10, 11, 16].

The idea behind volume ray casting is that rays are traced back from each pixel of the wanted image into the volume. While trac-

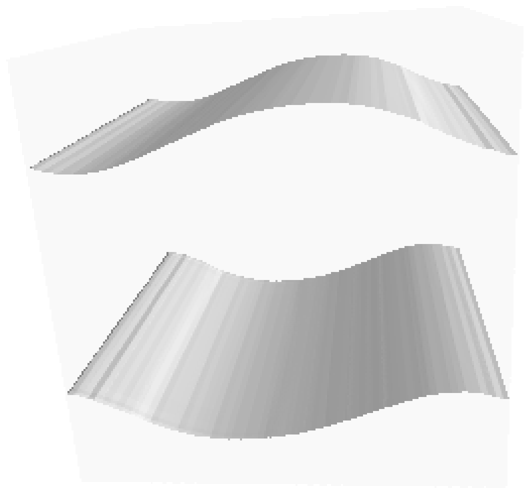

Figure 6: Iso-surface of pressure in a cavity flow computed by the combination technique.

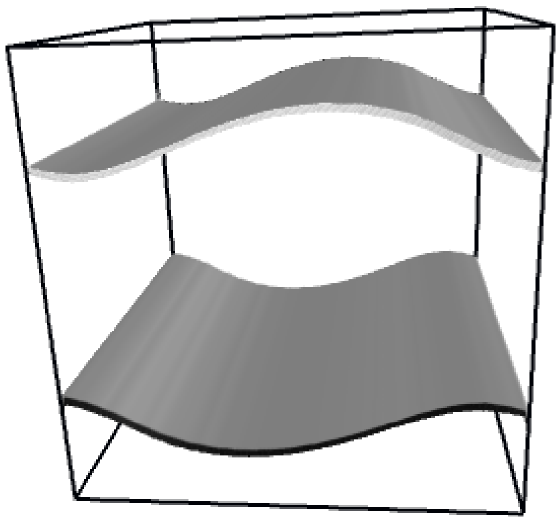

Figure 7: Iso-surface of pressure in a cavity flow displayed by Tivor.

ing back a ray from a pixel, contribution values of hit voxels are integrated or summed up in order to compute the intensity of this pixel.

We have implemented three different visualization techniques into our program. In the X-ray method, which provides X-ray like images, the intensity of a pixel is calculated by evaluating the line integral of function values along the ray.

Another algorithm we have implemented is the so called maximum intensity projection method. Here, the intensity of a pixel is the maximum function value occurring on the corresponding ray.

Extracting iso-surfaces is a third technique we have implemented in our tool so far. To find an iso-surface to a given iso-value, each ray is traced and a patch is displayed if the difference of current function value and iso-value changes the sign. In order to use Lambertian or diffuse reflection for illumination of the resulting surface, the gradient of the function is needed. Recall that the gradient is normal to all iso-surfaces.

RESULTS

In order to appraise the presented approaches, we compare the results of our visualization tool with results obtained by the program Tivor, a very efficient hardware based volume renderer working on uniform full grids [14]. The tests were performed on several data sets. Two of them are cavity flow data sets, given on a full grid of level 6, i.e. 64^3 nodes. These data sets are the result of a numerical flow simulation and contain pressure and temperature distributions of the flow. Another data set, given on a full grid of level 8 (256^3 nodes), contains a spherical harmonic (Legendre's function), which displays a solution of the Schrödinger equation of a hydrogen atom.

Figures (8), (9), and (10) show X-ray images of the pressure values in the mentioned cavity flow. These images have been rendered in order to reveal differences between the three implemented interpolation algorithms. In the OpenGL picture, small flaws can be detected, whereas the combination technique and the sparse grid method render nearly identical images. The artifacts in the OpenGL picture occur because of the already mentioned loss of accuracy using the hardware for mathematical operations.

In the other figures, sparse and full grid results are compared using different lighting models and data sets. The full grid results have been obtained by using Tivor.

Figures (6) and (7) display iso-surfaces of the pressure in the cavity flow data set. In Figures (11) and (14), the same data set is visualized by means of the maximum intensity projection lighting model. In further maximum intensity projection images (Figures (12) and (15)), the data set of a spherical harmonic function is visualized. Finally, the temperature distribution in the cavity flow is depicted in Figures (13) and (16) by using an X-ray and an emission-absorption lighting model.

Now let us consider the performance of our volume visualization program. Since the time consumption of the X-ray and maximum intensity projection algorithms is data independent and exactly the same, we used the X-ray method on the first data set to determine the computing times. The iso-surface extraction takes about two times longer than the other methods. Furthermore, the total time of the iso-surface computation depends on the size of the extracted surface.

In order to estimate the times listed below, images of resolution 200×200 pixels were computed. The pictures in the color plate are rendered in the same resolution as well. In our visualization tool, it is possible to scale such images at nearly no cost and display for instance an image of size 500×500 on screen.

The time measurements of the sparse grid and software combination method were performed on a Silicon Graphics O2 with a 195 MHz R10000 processor. The performance of the hardware based combination technique (OpenGL) was measured on an SGI Onyx with RealityEngineII graphics pipe and 16 MB texture memory. Additionally, we tested this method on an SGI Indigo^2 Maximum Impact with 4 MB texture memory. This machine is slightly slower than the Onyx. Moreover, due to the smaller amount of texture memory, the Indigo^2 can just handle grids up to level 8 in hardware, whereas the Onyx works even with grids of level 11.

The times of the software based algorithms are given in CPU-seconds. However, the times of the hardware based method are measured in real-time seconds because the CPU was not working to its capacity but was waiting until the graphics hardware had finished its computations.

level	5	6	7
sparse grid	257 s	792 s	2125 s
combination	19 s	61 s	167 s
OpenGL	1.2 s	2.8 s	6.3 s
level	8	9	10
sparse grid	5750 s	15800 s	42125 s
combination	570 s	1725 s	5080 s
OpenGL	15.4 s	36.6 s	88.5 s

The table shows that rendering times of both software implementations approximately increase by a factor of three every time the used level rises by one, whereas the measured times of the hardware based implementation increase by a factor of around two if the level rises by one.

Also the table reveals the speed up factors of the different algorithms. The combination method is between eight and fifteen times faster than the sparse grid algorithm. Furthermore, the OpenGL hardware method is between 15 and 60 times faster than the software combination technique. This results in a speed up factor between 225 and 500 from the sparse grid to the hardware based method.

Finally, we have to correlate these results to the rendering times of Tivor. Both cavity data sets, given on a full grid of size 64^3, can be handled and displayed by Tivor on both an SGI Onyx RE2 and Indigo^2 Maximum Impact interactively. However, the spherical harmonic data set of size 256^3 cannot be processed on an Indigo^2. Rendering this data set on the Onyx takes about 20 seconds. However, to achieve the result, Tivor needs around 500 MB main memory. Such a huge amount of memory is used because, for fast rendering, Tivor requires additional information like normals besides the actual data set. Of course, a data set of level 9 (512^3) could not be rendered using traditional full grids.

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 simulation. Assume that a floating point value is given at each grid node. Thus, we obtain the following results:

level	5	6	7
points of full grid	33^3	65^3	129^3
full grid	128 kB	1 MB	8 MB
sparse grid	6 kB	15 kB	35 kB
combination	22 kB	59 kB	152 kB
OpenGL	43 kB	124 kB	338 kB
num. low res. full grids	31	46	64
level	8	9	10
points of full grid	257^3	513^3	1025^3
full grid	64 MB	512 MB	4 GB
sparse grid	83 kB	200 kB	450 kB
combination	377 kB	914 kB	2.1 MB
OpenGL	884 kB	2.2 MB	5.4 MB
num. low res. full grids	85	109	136

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

The last row of the table contains the number of full grids that are combined to a specific sparse grid. In case of the OpenGL implementation, this number is equivalent to the number of used textures.

The combination technique requires more storage than the actual sparse grid algorithm since some of the needed nodes are stored several times. The OpenGL version of the combination methods consumes about twice as much memory as the software version because each of the used textures has to have dimensions that can be written as two to the i-th power. Nevertheless, compared with the original full grid data set, both implementations of the combination technique require a negligible amount of memory.

CONCLUSION

We have introduced volume ray casting on sparse grids. This allows to carry out volume visualization directly on sparse grids without transforming the results of numerical simulations on sparse grids to the associated full grids. Note that in real applications it is often impossible to load such full grids of about 1025^3 nodes into the main memory of a workstation.

Secondly, the sparse grid approach can be used as a compression method in order to realize volume rendering in huge data sets on workstations with a small amount of main memory.

By dint of using texture hardware, we have been able to overcome the tardiness of sparse grid interpolation. Therefore, it is possible to use volume visualization on sparse grids interactively, in contrast to other compression approaches.

There are several directions of future work. The first aim is to implement further transfer functions and lighting models into our visualization program, for instance an emission-absorption model. As a second goal, we intend to use OpenGL in order to accelerate the surface and volume illumination as well.

ACKNOWLEDGMENTS

We are grateful to S. H. Enger from the Lehrstuhl für Strömungsmechanik of the University of Erlangen for making the cavity data set available to us. Additionally, we would like to thank our colleague R. Westermann for his valuable remarks about OpenGL programming.

This work was partially supported by FORTWIHR, the Bavarian Consortium for High Performance Scientific Computing.

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] J. Danskin and P. Hanrahan. Fast Algorithms for Volume Ray Tracing. *Transactions Workshop on Volume Visualization*, pages 91–98, 1992.

[4] B. Drebin, L. Carpenter, and P. Hanrahan. Volume Rendering. *Computer Graphics*, 22(4):65–74, August 1988.

[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] H.-C. Hege, T. Höllerer, and Stalling D. Volume Rendering. Technical Report 93-7, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1993.

[8] 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.

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

[10] M. Levoy. Display of Surfaces from Volume Data. *Computer Graphics & Applications*, 8(3):29–37, May 1988.

[11] M. Levoy. Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[12] Silicon Graphics Inc., Mountain View, California. *OpenGL on Silicon Graphics Systems*, 1996.

[13] Silicon Graphics Inc., Mountain View, California. *RapidApp User's Guide*, 1996.

[14] O. Sommer, A. Dietz, R. Westermann, and T. Ertl. An Interactive Visualization and Navigation Tool for Medical Volume Data. In N. M. Thalmann and V. Skala, editors, *WSCG '98, The Sixth International Conference in Central Europe on Computer Graphics and Visualization '98*, volume II, pages 362–371, Plzen, Czech Republic, February 1998. University of West Bohemia Press.

[15] C. Teitzel, R. Grosso, and T. Ertl. Particle Tracing on Sparse Grids. In R. Rau, U. Lang, and D. Bartz, editors, *Ninth Eurographics Workshop on Visualization in Scientific Computing*, Blaubeuren, Germany, April 1998. EG, The EuroGraphics Association.

[16] J. Wilhelms and A. van Geldern. A Coherent Projection Approach for Direct Volume Rendering. *Computer Graphics*, 25(4):275–284, July 1991.

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

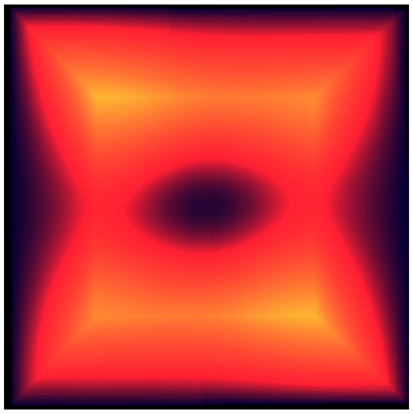
Figure 8: Sparse grid algorithm

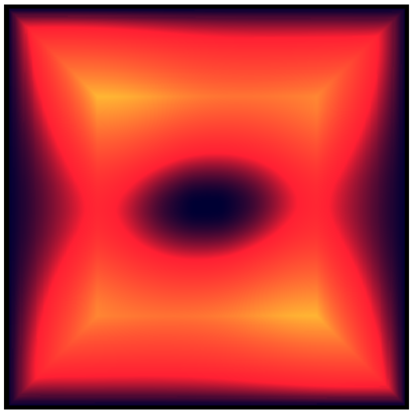
Figure 9: Combination technique

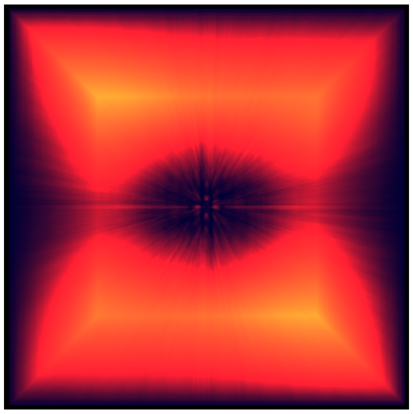
Figure 10: OpenGL method

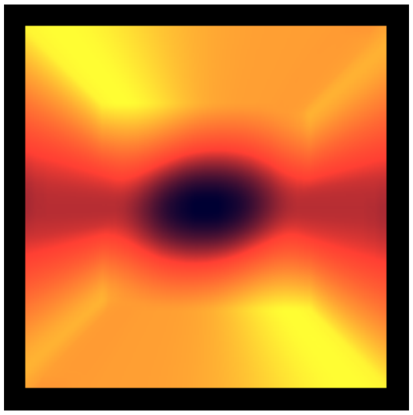
Figure 11: MIP image of a cavity flow on a sparse grid.

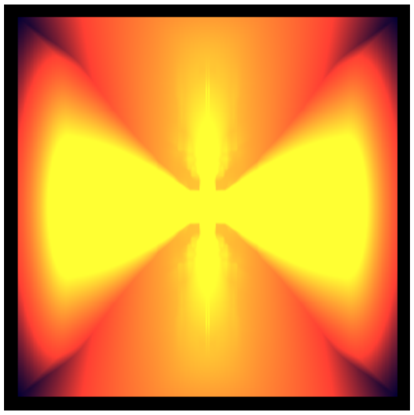
Figure 12: MIP image of a hydrogen atom on a sparse grid.

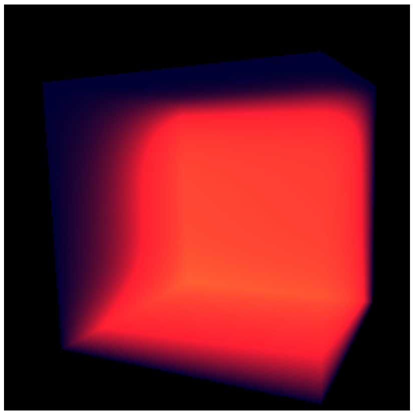
Figure 13: X-ray image on a sparse grid data set.

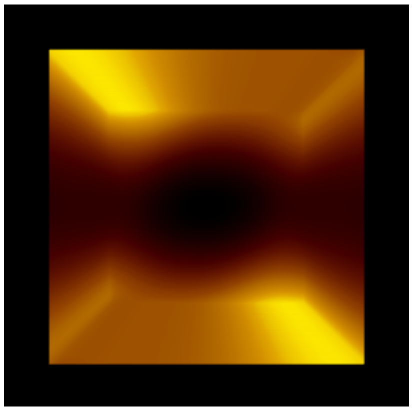
Figure 14: For comparison: MIP image of a cavity flow on a full grid.

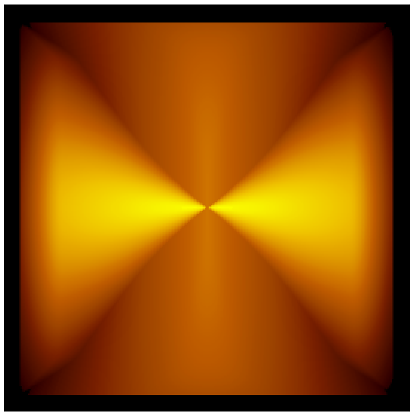
Figure 15: For comparison: MIP image of a hydrogen atom on a full grid.

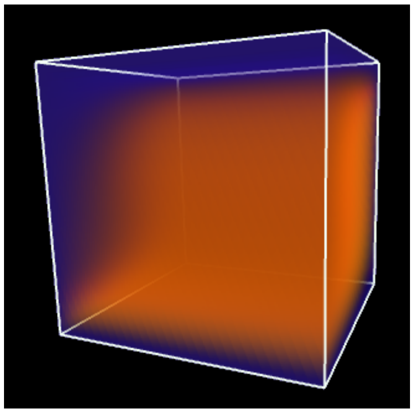
Figure 16: For comparison: Emission-absorption image on a full grid data set.