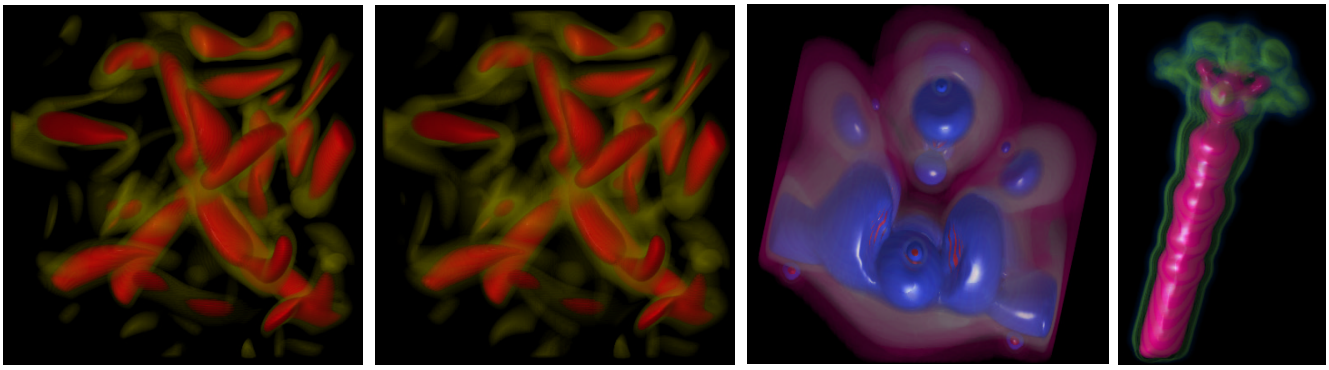
Space-Time Points: 4D Splatting on Efficient Grids

Neophytos Neophytou Klaus Mueller
Center for Visual Computing, Computer Science, Stony Brook University



Abstract

4D datasets, such as time-varying datasets, usually come on 4D Cartesian Cubic (CC) grids. In this paper, we explore the use of 4D Body Centered Cubic (BCC) grids to provide a more efficient sampling lattice. We use this lattice in conjunction with a point-based renderer that further reduces the data into an RLE-encoded list of relevant points. We achieve compression ranging from 50 to 80% in our experiments. Our 4D visualization approach follows the hyperslice paradigm: the user first specifies a 4D slice to extract a 3D volume, which is then viewed using a regular point-based full volume renderer. The slicing of a 4D BCC volume yields a 3D BCC volume, which theoretically has 70% of the datapoints of an equivalent CC volume. We reach compressions close to this in practice. The visual quality of the rendered BCC volume is virtually identical with that obtained from the equivalent CC volume, at 70-80% of the CC grid rendering time. Finally, we also describe a 3.5D visualization approach that uses motion blur to indicate the transition of objects along the dimension orthogonal to the extracted hyperslice in one still image. Our approach uses interleaved rendering of a motion volume and the current iso-surface volume to add the motion blurring effect with proper occlusion and depth relationships.

1 Introduction

In recent years, volume rendering has become quite popular in a variety of domains, such as science, medicine, engineering, business, and entertainment. A growing number of applications exist that involve four dimensions and higher, for example, MRI and 3D Ultrasound motion studies in cardiology, time-varying datasets in computational fluid dynamics, 4D shapes in solid models [31], and n-manifolds in mathematics [10][11][12]. Currently, volume rendering follows along six broad paradigms: raycasting [18][39], splatting [42], shear-warp [17], cell-projection [22][35], texture-mapping hardware-assisted [4][8][30], and custom hardware-accelerated [23][29]. Some of these have been extended into 4D and higher. In order to classify these extended algorithms, one may distinguish them by their intended purpose. While some algorithms were specifically developed for time-varying datasets and typically exploit time-coherency for compression and acceleration [1][9][19][32][33][36][41], other methods have been designed for general n-D viewing [2][3][10][11][12][13][40] and require a more universal data decomposition.

In n-D viewing, the direct projection from n-D to 2D (for n>3) is challenging. One major issue is that there are an infinite number of orderings to determine occlusion (for n=3 there are just two, the view from the front and the view from the back). In order to simplify the user interface and to eliminate the amount of occlusion explorations a user has to do, Bajaj et. al. [2] perform the n-D volume renderings as an X-ray projection, where ordering is irrelevant. The authors demonstrate that, despite the lack of depth cues, much useful topological information of the n-D space can be revealed in this way.

On the other end of the spectrum are algorithms [3] (and the earlier [40]) that first calculate an n-D hyper-surface (a tetrahedral grid in 4D) for a specific iso-value, which can then be interactively sliced along any arbitrary hyperplane to generate an opaque 3D polygonal surface for hardware-accelerated view-dependent display. This approach is quite attractive as long as the iso-value is kept constant. However, if the iso-value is modified, a new iso-tetrahedralization must be generated which can take on the order of tens of minutes [3]. Our method also uses the hyper-slice approach., however, in contrast to [3][40] we skip the intermediate meshing step and instead retain the original volume representation throughout the process. While this currently only allows rendering times in the range of seconds, this (lower) framerate is relatively insensitive to transfer function modifications, and, in addition, also provides full volume renderings with semitransparent object features.

Since 4D datasets can become quite large, a variety of methods to compress 4D volumes have been proposed in recent years. Researchers have used wavelets [9], DCT-encoding [19], RLE-encoding [1], and images [32][33]. All are lossy to a certain degree, depending on a set tolerance. An alternative compression strategy is the use of more efficient sampling grids. A good candidate is the so-called Body-Centered Cartesian (BCC) grid which was recently employed for 3D volume rendering in [38]. The BCC lattice can save almost 30% of the samples, without loss of signal fidelity, under certain conditions. BCC grids are particularly attractive for point-based volume rendering methods, such as splatting, since there the rendering time is directly proportional to the number of points. But, on the other hand, the BCC grid has also been

Email: {nneophyt, mueller}@cs.sunysb.edu

useful in speeding up the shear-warp algorithm [37] and computed tomography [21][27].

The BCC grid is a generalization of the hexagonal grid, and all of these lattices are members of the D_n^* grid family [6], which generalize into very high dimensions and whose duals are the D_n grids. An important corollary from the theory of lattices and sphere packings [6] is that the data reduction rate of the D_n^* grids grows with the number of dimensions n, and that this data reduction is in fact optimal. The projected sample reduction for D_4^* is already 50%, and it is the goal of this paper to explore the prospects and potential of this theoretical finding for 4D volume rendering, in conjunction with a high-quality point-based rendering approach.

Our paper is structured as follows. First, in Section 2, we discuss related work, and in Section 3 we present the theoretical background relevant to our work. In Section 4 we describe the details of our 4D splatting approach, both for cartesian grids and D_4^* grids. In that section, we also describe our 3.5D motion blur extension. Then, in Section 5, we describe our implementation, and Section 6 provides images and run times obtained with our implementation. Section 7 concludes with final remarks and future perspectives.

2 Related Work

Early work on 4D rendering includes a paper by Ke and Panduranga [13] who used the hyperslice approach to provide views onto the on-the-fly computed 4D Mandelbrot set. Another early work is a paper by Rossignac [31], who gave a more theoretical treatment of the options available for the rendering of 4D hypersolids generated, for example, by time-animated or colliding 3D solids. Hanson and co-authors [10][11][12] wrote a series of papers that use 4D lighting in conjunction with a technique that augments 4D objects with renderable primitives to enable direct 4D renderings. The images they provide in [12] are somewhat reminiscent to objects rendered with motion blur. Bajaj et al. [2] use octree-based splatting, accelerated by texture-mapping hardware, to produce 2D X-ray views of n-D objects. They also present a scalable interactive user interface that allows the user to change the viewpoint into n-D space by stretching and rotating a system of n axis vectors. The 4D isosurface algorithms proposed by Weigle and Banks [40] and Bhaniramka, Wenger, and Crawfis [3] both use a Marching Cubes-type approach and generalize it into n-D.

Methods that focus more on the rendering of the time-variant aspects of 3D datasets have stressed the issue of compression and time-coherence to facilitate interactive rendering speeds. Shen and Johnson [33] use difference encoding of time-variant volumes to reduce storage and rendering time. Westermann [41] uses a wavelet decomposition to generate a multi-scale representation of the volume. Shen, Chiang, and Ma [32] propose the Time-Space Partitioning (TSP) tree, which allows the renderer to cache and re-use partial (node) images of volume portions static over a time interval. It also enables the renderer to use data from subvolumes at different spatial and temporal resolutions. Anagnostou [1] extend the RLE data encoding of the shear-warp algorithm [17] into 4D, inserting a new run block into the datastructure whenever a change is detected over time. They then composite the rendered run block with partial rays of temporally-unchanged volume portions. Sutton and Hansen [36] expand the Branch-On-Need Octree (BONO) approach of Wilhelms and Van Gelder [43] to time-variant data to enable fast out-of-core rendering of time-variant isosurfaces. Lum, Ma, and Clyne [19] recently advocated an algorithm that DCT-compresses time-runs of voxels into single scalars that are stored in a texture map. These texture maps, one per volume slice, are loaded into a texture-map accelerated graphics board. Then, during time-animated rendering, the texture maps are indexed by a time-varying color palette that relates the scalars in the texture map to the current color of the voxel they represent. Although the DCT affords only a lossy compression, their rendering results are quite good and can be produced interactively. Another compression-based algorithm was proposed by Guthe and Straßer [9], who use a lossy mpeg-like approach to encode the time-variant data. The data are then decompressed on-the-fly for display with texture mapping hardware.

The research most related to our 4D work is the 3D approach by Theußl, Möller, and Gröller [38], who introduced BCC grids to the visualization community. Rendering was achieved by ways of a pre-classifying splat renderer with per-splat compositing. Pre-shaded splatting requires an estimate of the gradient at each voxel position, and they use central differences for this purpose. This, however, is complicated by the fact that a voxel's nearest BCC grid neighbor along an axis is $\sqrt{2}$ away, which is a greater distance than in the equivalent Cubic Cartesian (CC) grid. To use a closer distance, they compare this gradient estimation method with one that interpolates a point on each cell face and uses interpolated point pairs on opposing cell faces to estimate the gradient. These points are now closer together, i.e., $\sqrt{2}/2$, than on the equivalent CC grid. Nevertheless, there are only slightly noticeable differences in image quality for the three schemes. Our splatting algorithm (described in [28]), in contrast, avoids these issues altogether since it defers classification, gradient estimation, and shading until the volume has been sliced and interpolated into image-aligned sheets. Hence the gradient estimation scheme is identical for both BCC and CC grid rendering. Our image-aligned sheet-buffered approach also provides images with less blur and popping and sparkling artifacts, however, at the expense of higher rendering time [25][26].

3 Theoretical Background

According to the theory on lattices and sphere packings [6], the D_n grids provide the closest lattice packings of hyper-spheres in 2D to 5D (others are known for higher dimensions). This is a fact that can be exploited for packing the frequency spectra of a discretized spherically-bandlimited function in an optimal way in the frequency domain (see Fig. 1 for a 2D example). Fourier analysis tells us that this closest packing in the frequency domain gives rise to the sparsest distribution of sample points in the spatial domain [7], without reducing the frequency content that the signal would have if sampled into the tightest cubic cartesian (CC) grid. Thus the compression is lossless (under the condition of spherical bandlimitness).

The D_n lattice can be constructed by this generator matrix:

$$M = \begin{bmatrix} -1 & -1 & 0 & \dots & 0 & 0 \\ 1 & -1 & 0 & \dots & 0 & 0 \\ 0 & 1 & -1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & -1 \end{bmatrix} \quad (1)$$

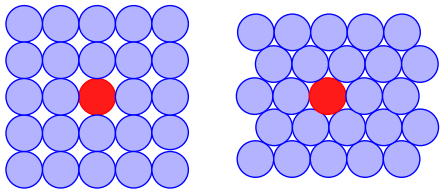


Figure 1: The cartesian grid (left) vs. the hexagonal grid (right) as two possible frequency domain lattices. The latter provides the tightest packing of a discrete 2D signal's circularly-bounded frequency spectrum. (Here, the dark, red circle contains the main spectrum, while the others contain the replicas or aliases.)

For n=3 this gives rise to the so-called face-centered cubic (FCC) lattice, shown in Fig. 2b. If the sampling distance in the cartesian grid is T, then the Nyquist frequency of the cartesian grid is $\omega_{Nyquist}=1/T$. Thus, if we want to pack the frequency spectra conforming to this $\omega_{Nyquist}$ tightly, but without overlap, into the FCC lattice, we must scale the generator matrix by a factor of $1/(T\sqrt{2})$. The generator matrix U for this scaled D_n lattice is then given by:

$$U = \frac{1}{T\sqrt{2}} M \tag{2}$$

To find the sampling matrix V in the spatial domain that gives rise to the lattice generated by U in the frequency domain, one can perform a Fourier transform and finds the following relationship: $U^T V = I$, where I is the identity matrix [7]. V is then given by $V = (U^T)^{-1}$. This results in the dual of the D_n lattice, termed D_n^* lattice [6]. The resulting V is not very intuitive, but we can obtain a better one by a suitable rotation of V. We use the one given by Conway and Sloane [6]:

$$M^* = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \dots & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \tag{3}$$

Including the scaling factor of (2) we get:

$$V = T\sqrt{2}M^* \tag{4}$$

For the 3D case, this yields the sampling pattern for the BCC lattice shown in Fig. 2c. It consists of two interleaved CC lattices, each with spacing $T\sqrt{2}$ and offset by a factor of $T/\sqrt{2}$ in all three axis directions. Thus the BCC lattice has 1/2 of the CC lattice samples (shown in Fig. 2a) within each slice, but $\sqrt{2}$ more slices are required. This amounts to an overall savings of samples of $(1-\sqrt{2}/2)\cdot 100\% = 29.3\%$, which one may translate into an equivalent reduction in storage and rendering time.

In 4D we also get an interleaved embedding of two CC grids, but this time the interleaving is controlled by the fourth dimension. Both offset and grid spacing is again $T/\sqrt{2}$ and $T\sqrt{2}$, respectively. This is illustrated in Fig. 2d, where we have untangled the grid along the fourth dimension and arranged the 3D subgrids side by side, with the fourth dimension axis aligned with the axis of the first dimension. In general, the lattice is always constructed by taking the n-1 base lattice and offsetting and interleaving it for the n-th dimension.

The compression of the D_4^* lattice can be informally calculated as follows. We now have $1/(\sqrt{2})^3$ less samples per CC, but we have a $1/\sqrt{2}$ smaller sampling distance along the fourth dimension, which is equivalent to having $\sqrt{2}$ more samples along the fourth axis. This means that the total amount of savings is $\sqrt{2}/\sqrt{2}^3 = 1/2$. Thus, by sampling a time-varying dataset (or any other 4D dataset) into the D_4 lattice we can get away with 50% of the samples without impairing the frequency content. For 5D, the number of samples reduces even further to 35%.

It should be noted, however, that lossless compression is only warranted if the sampled signal has a (hyper-) spherically bandlimited frequency spectrum. If a signal already sampled into a CC grid is resampled into a D_n grid, then it must be bandlimited into a hyper-spherical spectrum at that time, else aliasing will occur. If the signal has already been bandlimited to such a spectrum before, then the compression is lossless, else it is lossy. For example, a CT reconstruction performed with a Gaussian kernel will be spherically bandlimited, but a reconstruction that employs a trilinear kernel will be not. Although this sounds restrictive in theory, our experiments have indicated that this plays a very small role in practice. Furthermore, if the visualization process uses a Gaussian or any other radially symmetric kernel (as splatting does), then the data would be filtered into a spherically spectrum anyhow before it reaches the screen.

We have used the Marschner-Lobb function (MLF) [20] to verify these theoretical foundations in practice. We first sampled the MLF into a 40^3 CC grid and rendered it (see Fig. 3a). At that sampling rate, 99.8% of the MLF's frequency content falls below the cartesian Nyquist limit. A significant portion of the spectrum is close to that boundary, which makes it a very demanding test function for filters. We found, however, that this frequency boundary is slightly non-spherical which causes significant aliasing when the MLF is rendered from the equivalent BCC grid (see Fig. 3b). In order to constrain the MLF's frequency spectrum into a spherical function, while preserving the sharp fall-off at the (now spherical) Nyquist boundaries, we lowpassed the MLF with a spherical filter of 10 times the MLF's bandwidth. We then sampled this modified MLF both into a 40^3 CC grid and into the equivalent BCC grid, generated by (4). The resulting renderings are shown in Fig. 3c and Fig. 3d, respectively. We make two observations: (i) The CC renderings from the original spectrum and the spherical one are very similar which indicates that the spectrum was only slightly modified (there is no apparent blurring); and (ii) the renderings of the modified MLF from the CC and the BCC grids are very similar which confirms the theory that CC and BCC grid renderings of the same function will lead to very similar results, especially if an ana-

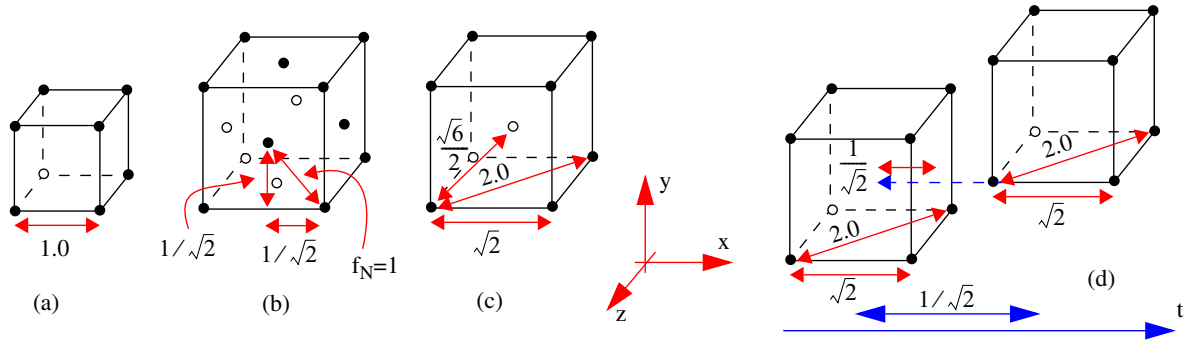
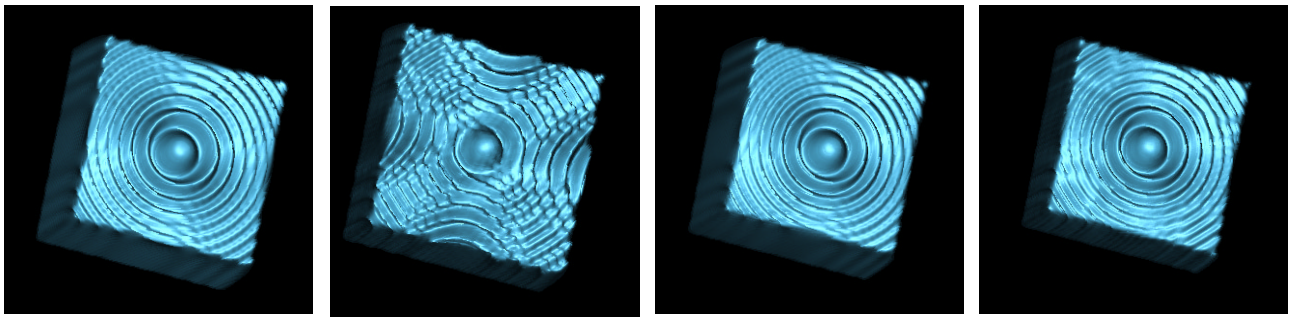


Figure 2: Various grid cells, drawn in relative proportions. We assume that the sampling interval in the CC grid is T=1. (a) Cubic cartesian (CC) for cartesian grids (all other grid cells shown are for grids that can hold the same spherically bandlimited, signal content); (b) Face-centered cubic (FCC) for D_3; (c) Body-centered (BCC) cell for D_3^*; (d) 4D BCC for D_4^*. Here, for illustrative purposes, we have pulled apart the 4D BCC along the t-axis, giving rise to two CC cells, offset by $1/\sqrt{2}$ along all 4 axes. Note that we chose the labeling xyzt for illustrative purposes. The order generalizes to any arbitrary 4D coordinate labeling.



(a) (b) (c) (d)

Figure 3: Rendering of the Marschner-Lobb test function on different grids: (a) 40^3 CC grid, non-spherical frequency spectrum; (b) BCC grid equivalent to the 40^3 CC grid, non-spherical frequency spectrum; (c) 40^3 CC grid, constrained to a spherical frequency spectrum; (d) BCC grid equivalent to the 40^3 CC grid, constrained to a spherical frequency spectrum.

lytic function is available that can directly be sampled into the BCC grid.

4 Splatting in 4D

For the purpose of this discussion, a 4D volumetric dataset is conceived of as a sampled representation of a 4D continuous signal f(x,y,z,t), defined within the hypervolume for which (0≤x<nx, 0≤y<ny, 0≤z<nz). We visualize the dataset in a two-step process. First, the user interpolates an arbitrary hyperplane given by the equation:

$$a \cdot x + b \cdot y + c \cdot z + d \cdot t + e \;=\; 0 \qquad (5)$$

with (a,b,c,d) being the normal vector of the plane and e the distance from the origin. The result is list of 3D voxels, for which shading, visibility-ordering and depth-compositing will have to be performed each time a new 3D viewpoint and transfer function is specified by the user. We use the image-aligned sheet-buffered splatting method described in [28] for the 3D rendering.

The extraction of a hyperslice can be done in two ways. The first method uses the fact that a sliced 4D Gaussian is a 3D Gaussian, weighted by a factor $exp(-a \cdot d^2)$, where d is the distance of the 4D point from the hyperslice and a determines the kernel's spread. All voxels for which $|d| \leq$ kernel halfwidth will be extracted from the 4D volume and included in the list used for 3D rendering.

The other method does not extract voxels from the 4D grid, but interpolates a 3D volume by ways of a 4D filter with bandwidth $(\omega_x,\omega_y,\omega_z,\omega_t)$. This allows the extraction of volumes at any resolution, at any hyperplane orientation, and into any grid, such as a 4D CC into a 3D BCC lattice or a 4D BCC into a 3D BCC lattice. The 4D-to-2D projection filter is then a 4D filter h_{4D} aligned with the axes of the 4D data, that is convolved with the 3D Gaussian h_{3D} used for splatting, aligned with the axes of the extracted volume:

$$h(x, y, z, t) \;=\; h_{4D}(x, y, z, t) \otimes R^{-1} \cdot h_{3D}(x_v, y_v, z_v) \qquad (6)$$

where R is the orientation matrix of the hyperslice and (x_v,y_v,z_v) is the coordinate system of the extracted volume. We can do this via a 3D splatting table if the kernel is rotationally symmetric. In that case, the march through the table is defined by the orientation of the hyperplane. We only splat those points that are within half the kernel width of the hyperslice and we omit points with densities outside the interesting range.

Note that if we interpolate the 3D volume at the same (or lesser) resolution than the 4D volume, which is probably the most reasonable, then the list of voxels of the second method is bound to be much smaller than the list of voxels for the first method. While for the first method, the number of extracted voxels is proportional to the product of kernel width w and the volume resolution, it is only proportional to the volume resolution in the second method. This will increase the 3D rendering time for the first method by a factor of w. On the other hand, the interpolation cost is proportional to w^4, while the extraction cost is only proportional to w.

For the general 4D slicing, we chose the second method for our work since we wanted to minimize the number of 3D voxels for rendering. But there are a few special cases that allow the more efficient extraction method to be applied, without increasing the number of voxels in the list. These special cases arise when the hyperplane is aligned with 3 of the 4 hyper-volume axes, i.e., 3 of the 4 coefficients (a,b,c,d) in (6) are zero. Fortunately, these are likely to be the most popular rendering modes – they slice the volume along x, y, z, or t. A further condition is that the projected 4D grid must coincide with the desired 3D grid. This results in the class of volumes that have the same resolution and the same orientation than the projected 4D volume. However, this is not really a restriction since we can do the resampling and rotations later, by ways of the viewing matrix during the 3D volume rendering.

This restricted rendering mode has certain implications for different grid types, which we shall explain in the following.

4.1 Axis-aligned 4D splatting for CC grids

In this special case we can efficiently collapse pairs of extracted voxels into a single voxel: Two 3D volumes that are immediate neighbors of the hyperslice can be simply combined by linear weighting along the projection direction. We can use a linear filter, which has a box shape in the frequency domain, since the cartesian grid does not impose any radial bandlimitness conditions. The complexity for this projection is N, the number of voxels in the 3D volume.

4.2 Axis-aligned 4D splatting for BCC grids

The situation for the D_4^* grid is more complex. Here we need a radially symmetric filter, such as a Gaussian, for 4D interpolation, since we need to preserve the spherical bandlimitness of the 4D signal. The Gaussian is a non grid-interpolating filter, which means that, even if the projected 4D grid coincides with the desired 3D grid, not only samples along the fourth dimension will affect an interpolated grid point, but also the entire 4D neighborhood that falls into the Gaussian extent of the kernel placed at the grid point. So it seems that we are still stuck with the problems of the general configuration. However, the fact that the grid distances within the nested CC cells are relatively wide helps us here. Fig. 4 serves as an illustration, where we use t as the fourth dimension, without loss in generality. There we see a 4D BCC grid (2 nested CC grids) pulled apart along t. The vertical dashed line indicates the time slice at $t=t_1$ that we would like to interpolate. We realize that the projection of a 4D BCC grid is a 3D BCC grid, and since we

project the 4D grid such that it coincides with the desired 3D grid, we will retain the two nested (x,y,z) spatial CC grids and just interpolate the new grid values. It would save interpolation time if we could just use the original, unfiltered, grid values and only interpolate (weigh) them over time, i.e., just use a 1D Gaussian along the time axis (like we did for the axis-aligned CC case). To see if this is feasible and to estimate how much error we would commit, consider voxel A in Fig. 4. The 4D Gaussian will overlap voxels C and D in the same time instance, and voxel B in the next time instance (as well as others). But realize that both B and C (and D) are a distance of 1.42 away. A relatively narrow Gaussian for interpolation, which still has good frequency behavior, is given by:

$$h(r) = e^{-2 \cdot r^2} \tag{7}$$

For r=1.4 (points B and C) this Gaussian has decreased to less than 2% of the maximum value. If we are willing to commit this error then the interpolation time would be almost twice as fast as for the Cartesian grid, given that we have to interpolate half the number of voxels. If we are not willing to commit this error, then we will need to use 8 B-voxels and 6 C-voxels for interpolation (the weights are constant and don't have to computed) per grid point. However, our experiments have shown that the results obtained with the approximate method are just as good as those with the accurate method, and therefore all results reported here use the approximate method.

The interpolation yields a list of voxels for the resulting BCC grid. While the 4D BCC grid has only 50% of the voxels in the 4D CC, the extracted 3D BCC has 30% of the 3D CC voxels. This is the case because we have 2 lists of $N/\sqrt{2} \cdot 1/\sqrt{2} \cdot 1/\sqrt{2}$ voxels each, which amounts to a saving of 29.3%. This was to be expected for the 3D BCC grid.

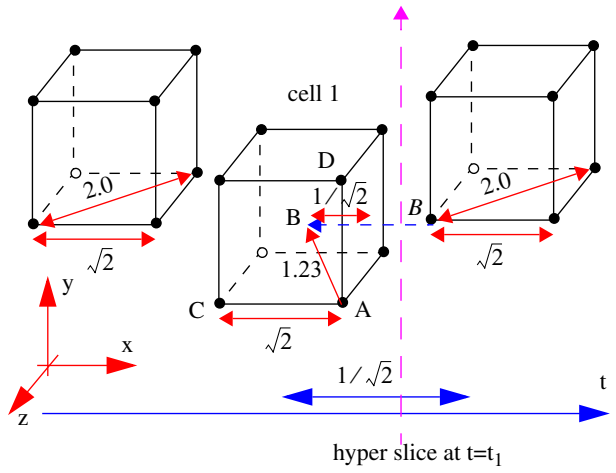


Figure 4: Contributions of neighboring grid points to sample point A.

4.3 Motion blur effects

Although the hyper-slice approach no longer provides full 4D views, we have sought to include at least some visual information about the 4D characteristics of the data within the periphery of the hyperslice. For this purpose, we have added the capability to render these evolving features as semitransparent motion trails, inked in a distinct color. This is alternative to the use of 3D vectors or glyphs, but does not require any motion analysis.

Since we chose a 4D viewing paradigm that only shows the user a shaded 3D slice of the 4D space at a time, and not a 4D X-ray projection of all data onto a 2D plane, we investigated alternative ways to convey a better understanding of the 4D processes in one image. We picked an approach that could be classified as 3.5D and involves adding a motion trail to the features present in a certain hyperslice. For the following discussion suppose that we would like to create a motion blurring along the time axis *t*, without loss in generality. What we would like to achieve is the depiction of a sharp isosurface at t=t_1, merged with the blurry trail that this iso-surface left during the time interval [t_1, t_1-Δt]. Further, we would like to show these two features with different colors to clearly distinguish them. Basically, this is a composite of two renderings: a post-classified rendering of the iso-surface at t_1, and a pre-classified rendering of the 4D volume within [t_1, t_1-Δt]. Our first attempt passed all (interpolated) voxels at t_1 and only the voxels with values > isovalue for the motion-blurred time interval. Then we set the alpha transfer function to opaque for values greater than the isovalue and to semitransparent for lower values. The problem with this approach is that the lower densities and the Gaussian fall-off of the voxels at t_1 are confounded with the lower densities of the motion-blurred features.

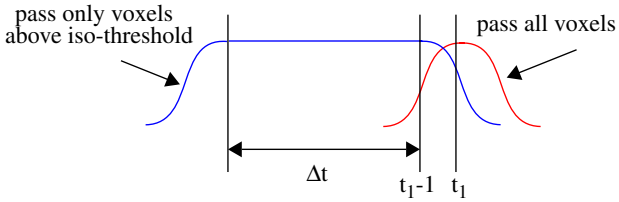


Figure 5: Compositing a motion blurred 4D slice interval of length Δt with a hyperslice at t=t_1.

A more conclusive display can be created by decoupling the transfer functions of the motion-blurred object and the iso-surfaced one at t_1. There are two solutions to achieve this. The first post-blends two separate images, one taken of the hyperslice at t_1 and one taken of the lowpassed hyperslice of width Δt (see Fig. 5). The problem with this post-process approach is that occlusions are not properly handled, and therefore depth relationships involving the motion path are not clearly conveyed. A better solution is to treat the two volume objects, the one at t_1 and the lowpassed one, as two gases that share a common environment, with different color and opacity mappings of their densities. Different strategies for the intermixing of two volumes are also discussed in [5]. We choose the opacity transfer functions like before: an iso-threshold for the hyperslice at t_1 and a flat semitransparent mapping for the motion-blurred hyperslices. In order to preserve occlusions, rendering of the slices must occur simultaneously. For this purpose, we render their densities into two separate sheet-buffers, coloring and classifying them using their individual transfer function. Then we composite the two sheet-buffers into a common compositing buffer using the following equations:

$$\begin{aligned} c_f &= \frac{(c_1\alpha_1 + c_2\alpha_2)}{2}(1-\alpha_f) + c_f \\ \alpha_f &= \frac{(\alpha_1 + \alpha_2)}{2}(1-\alpha_f) + \alpha_f \end{aligned} \tag{8}$$

where c_1, c_2, α_1, α_2 are the colors and opacities of the two objects, and c_f and α_f are the color and opacity of the composited object. We weigh each contribution by a factor of 1/2 to avoid color and opacity values > 1.0.

5 Implementation

In our implementation, the 4D volume is represented by a pointer-indexed RLE list. RLE lists have become popular for volume rendering in the shear-warp algorithm [17] and have also recently been used for splatting [14]. The hyperslicing interpolates a 3D volume in the manner described in Section 4 and encodes the 3D volume into another RLE list that is subsequently used for ren-

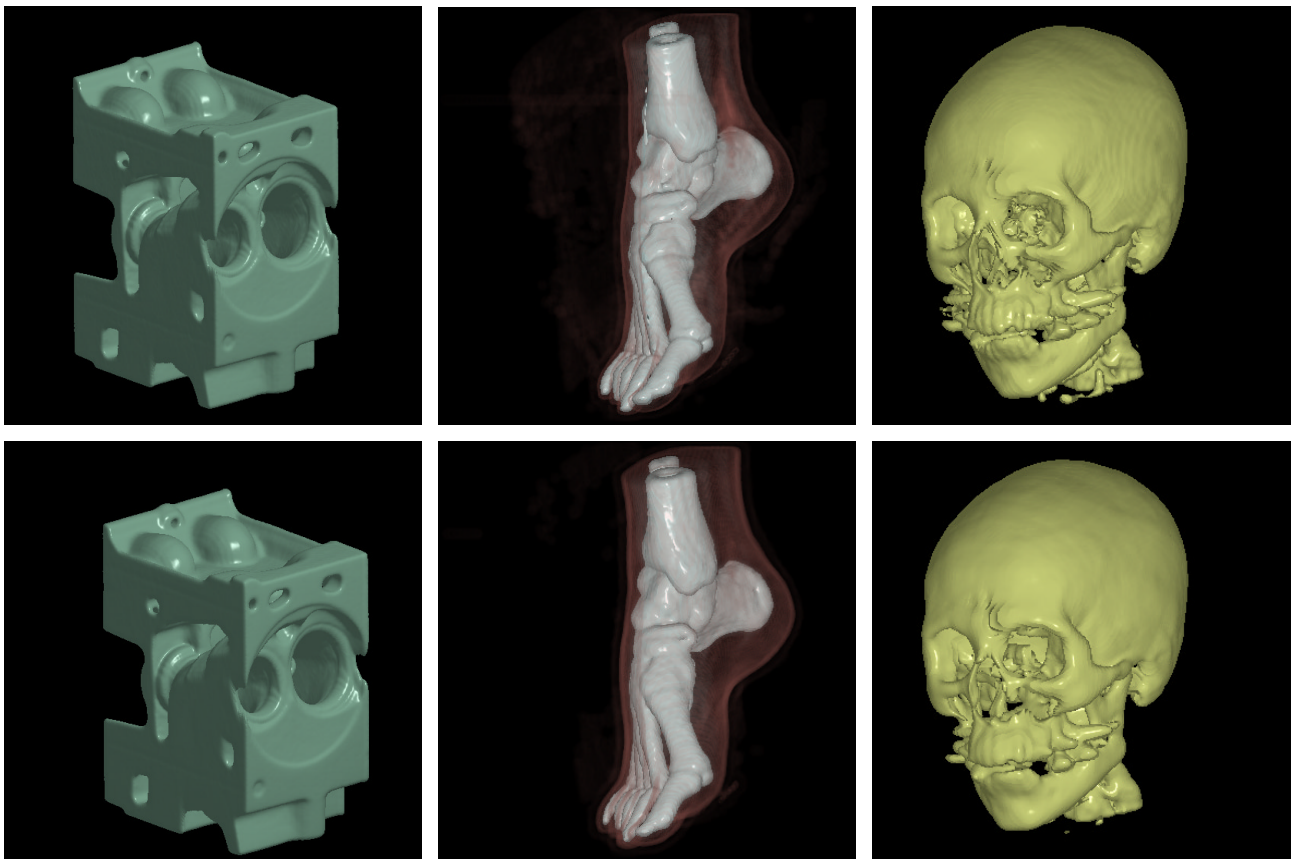


Figure 6: Renderings on the CC (top row) and the BCC grids (bottom row): From left to right: engine dataset, visible human foot, UNC head.

dering. As mentioned before, if an axis-aligned slicing is performed, then the hyperslice interpolation reduces to a simple weighted sum of voxels. The (3D) RLE list is traversed every time the transfer function or 3D viewing parameters are changed, and the voxels are tossed into an array of buckets to provide the depth ordering for rendering. The RLE facilitates differential arithmetic to perform the image-space transformation, requiring less than two vector adds and mults per voxel on average. After the bucket-tossing, we use the image-aligned sheet-buffered splatting algorithm described in [28] to render the 3D volume. Only a modification of the 4D viewing parameters triggers a new hyperslice interpolation. We also have implemented direct slicing where the construction of the (3D) RLE list is bypassed and the extracted 4D voxels are directly tossed into the bucket array.

The modification of the 3D splatter to render BCC grids is straightforward. The only adjustment that is necessary takes place at the onset of the bucket-tossing stage of the algorithm. The transformation matrix that transform the points into image space must be pre-multiplied by the following matrix:

$$\begin{bmatrix} \sqrt{2} & 0 & 0 & a \\ 0 & \sqrt{2} & 0 & a \\ 0 & 0 & \sqrt{2} & a \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{9}$$

where a is set to $1/\sqrt{2}$ when z is odd and to 1 otherwise.

6 Results

We now present experimental results that we have produced with our software. All images were rendered in a pure-software implementation on a Pentium 4, 2Ghz machine with 512 MB of RAM. We first tested the 3D BCC volume rendering with the following datasets: UNC Head, Visible Human Foot, and Engine. Since these datasets are only available on CC grids, we resampled the data with a cubic spline filter into the corresponding BCC grids.

Table 1 compares the speedups that are due to the BCC grid.

Dataset	Size	Eff. points	Time
Head CC	128^3 (2.1 M)	492 K	1.26
Head BCC	91^2 x 181 (1.5 M)	345 K	0.98
Foot CC	128^3 (2.1 M)	208 K	1.26
Foot BCC	91^2 x 181 (1.5 M)	148 K	0.91
Engine CC	256^3 (16.7 M)	1565 K	2.46
Engine BCC	182^2 x 366 (12 M)	1248 K	1.92

Table 1: Comparison of the efficiency of the BCC grid versus the CC grid. The *size* column lists the number of points needed on both grids to store the equivalent dataset. The third column lists the number of points that effectively made it into the rendering pipeline (the relevant voxels). The *Time* column lists the time needed to render one frame at the resolution of the dataset. All timings are listed in seconds.

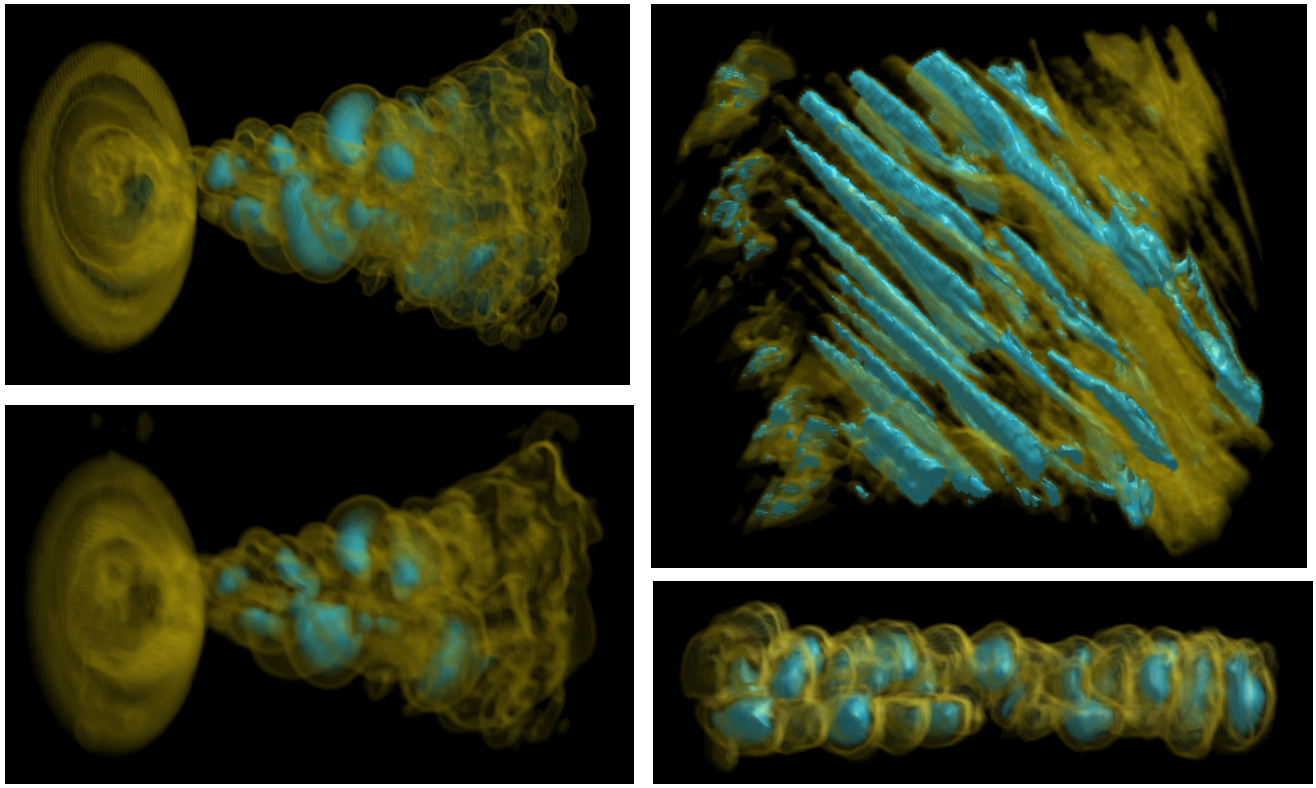


Figure 7: 4D turbulent jet dataset. Top left: 4D CC grid; bottom left: 4D BCC grid; top right: hyperslice at z=62; bottom right: hyperslice at x=35.

Similar to [38], we were also able to achieve speedups (over the CC grid) of around 25% at almost no difference in visual quality. On the front page we show (in the right-most two images) the fuel injection and the neghip datasets rendered on BCC grids, and Fig. 6 shows side-by-side comparisons of the Engine, the Visible Human Foot, and the UNC head. It should be noted that the resampling of the data into the BCC grid adds a certain amount of smoothing, which, for example, can be observed in the teeth section of the skull. The missing detail is about 1-2 voxels wide. The smoothing requires the transfer function to be shifted towards lower densities in order to obtain similar results.

For the 4D measurements we have used the following datasets: Vortex, Jet Shockwave, and Turbulent Jet. Again, the 4D BCC volumes are resampled versions of the corresponding cartesian volumes. On the front page, we show images of the Vortex dataset, rendered from the 4D CC grid (left) and from the 4D BCC grid (right). There is very little difference between the two images. Fig. 7 shows images of the turbulent jet dataset obtained by slicing the data volume at a (non-grid) time step, both for the 4D CC (top) and 4D BCC (bottom) grids. The images on the right in the same figure show two slice-over-time projections along the z and the x-axis, respectively. These views demonstrate an alternative way to visualize change over time for specific parts of the volume, and identify specific motion patterns. There is only little difference in the visual quality for the two different grids, mostly related to the slight smoothing due to the BCC grid resampling. Finally, Fig. 8 shows images of the jet shockwave dataset. On the left we show the 4D CC grid rendering, and on the right we show the 4D BCC grid rendering. Again, there is only little difference between the two.

Table 3 lists the statistics we have obtained. We first observe that the total size of the 4D datasets shrinks by 50% when resampled into the 4D BCC grid. Since point-based rendering does not make use of spatial coherency, we may extract and store only a list of relevant points (those points with values above "air" and noise) from the 4D dataset. Compression ratios ranging between 25% to 95% can be obtained that way for the tested datasets. The point list is RLE encoded to spare us from the need to store the (x,y,z,t) coordinate for each point. The RLE encoding typically requires up to 1.5 bytes per point on average, but we also need a few index tables to navigate the RLE list. We notice that the BCC-related compression of the point list of the Turbulent dataset is much less than 50%. This is due to the smoothing in the resampling process which brought the values of relevant object boundaries close to the values of air. Post-classified rendering requires a thin shell of voxels around iso-boundaries to ensure good iso-surface interpolation. Since now the voxels in this shell have values close to those usually removed as air, they cannot be distinguished and all voxels, both air and boundary, must be included in the list. The other two datasets have more distinct boundaries in terms of value and thus don't suffer from this effect. Hence, their compression remain around 50%. Note that this reduction of compression ratio is due to the required resampling step. Note that if we had obtained the BCC data directly from the domain process, then the smoothing would not have occurred (except the hyper-spherical smoothing).

The point list is then used to interpolate a hyperslice. We show data for the interpolation of an arbitrary time step. The result of interpolating a 4D BCC grid is a 3D BCC grid, which has a theoretical compression to 71%. In practice we get compressions in the range of 71% to 84%. This is again due to the smoothing along the interpolated axis (here time). We also list the size of the RLE list which we construct on-the -fly for the extracted 3D dataset. It is up to 40% larger than complexity of the point list. We construct the RLE only when the user extracts the 4D hyperslice for further exploration in 3D, using the usual 3D rotations and transfer function modifications. If the user is in 4D hyperslicing mode, we toss the points directly into the buckets, without constructing a 3D RLE

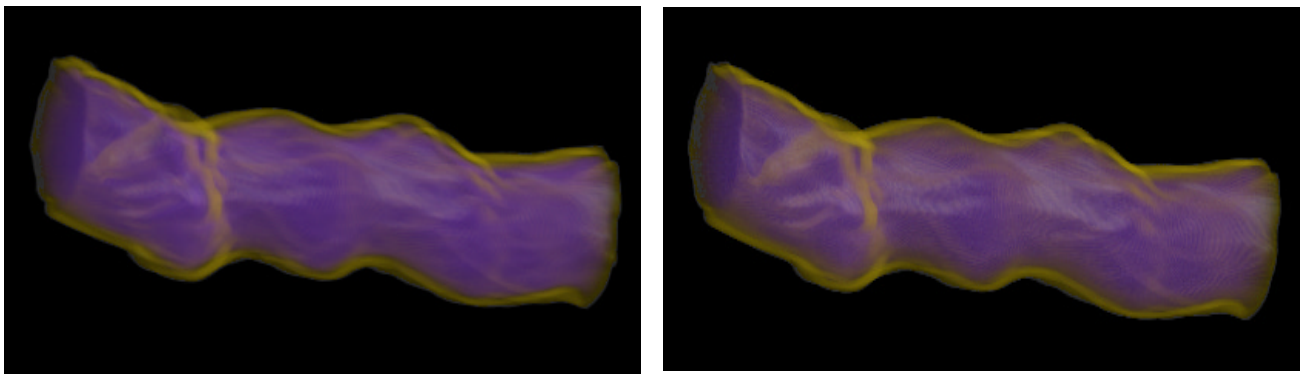


Figure 8: Jet shockwave dataset. Left: 4D CC grid; right: 4D BCC grid.

list. For the axis-aligned slicing, points from different instances but with the same spatial location are merged and added, properly weighted into a single point, before the bucket toss.

The time to render a frame, which includes the 4D extraction, is listed in the last column of Table 3. We observe that the BCC grid gains us about 20-30% speedup for rendering, depending on dataset. The rendering time is very closely related to the number of points, which comes at no surprise since we found that the point rasterization takes up over 70% of the overall rendering effort.

Fig. 9 demonstrates the motion-blur effect for the Vortex dataset at t=60, for an iso-value=150 and a motion trail of Δt=10. The top left image in the one obtained with post-compositing two separate images of iso-surface and motion trail, while the top right image was obtained with slice-wise compositing of these two objects. For both images we have set the transfer function for the motion blurred object to a slight blue and the alpha transfer function to a flat 0.3. The iso-surface for the object at t_1 appears as an opaque purple structure. For comparison, on the bottom of Fig. 9 we also show a few of the actual timesteps (within steps 50-60) that contribute to the blurred features of the final image. We observe that the slice-wise composited image shows occlusions of the two object states correctly, while the post-postprocessed image does not. The time required to render the two images is about the same.

7 Conclusions

In this paper, we have applied well established mathematical theory on hyper-spherical lattices to losslessly compress 4D datasets, under the condition that the frequency spectrum of the data is spherically bandlimited. We found that the raw datasets could be compressed to about 50% of their original size, a ratio that is in par with theory. Since we are using a point-based renderer we can reduce the magnitude of the data even more by only storing an RLE list of relevant data points (those with values greater than "air" and noise). The RLE-encoded BCC point lists compress to a size of 20-50% of the raw CC data for the datasets tested. Since the resampling of the data from the original 4D CC grid into the 4D BCC grid introduced some amount of lowpassing which required the inclusion of some of the air voxels into the point list, we believe that the compression ratios would be even better if the tested datasets were sampled directly into the 4D BCC grid by the data generation process. In these regards, our research (as well as that of [38]) has hopefully provided some pointers to the communities working in these fields of science.

We have used a hyper-slice approach to explore the 4D data, i.e., the user first specifies a 4D slice of the data, which amounts to a 3D volume, and then views this volume using our point-based renderer. Transfer functions and shading are available to bring out certain aspects of extracted volume. Slicing a 4D BCC grid results in a volume on a 3D BCC grid, which theoretically compresses the data to 70% of the CC size. We come close to that, with some reductions due to the lowpassing incurred in the slicing. The quality of the BCC and CC volume is virtually identical, with perhaps a small amount of smoothing visible in the former for some datasets. The reductions in dataset size translate directly into rendering

Dataset	**# time steps**	**Total data size**	**Total # relevant voxels (% total size)**	**# relevant voxels in % BCC / CC**	**# relevant interpolated voxels (% BCC / CC)**	**Size RLE encoding (% of # relevant interpolated voxels)**	**Render time (% BCC / CC)**
Turbulent CC	99	168.3 M	9.2 M (5%)	-	127 k	146 k (114%)	1.23
Turbulent BCC	139	87.0 M	7.4 M (8%)	80%	107 k (84 %)	146 k (136%)	1.01 (82%)
Vortex CC	80	160.0 M	109.7 M (68%)	-	1.3 M	1.6 M (123 %)	5.63
Vortex BCC	112	84.3 M	60.3 M (75%)	54%	986 k (75 %)	1.35 M (137 %)	4.58 (81%)
Jet Shockwave CC	56	89.6 M	38.0 M (42%)	-	727 k	719 k (98%)	5.42
Jet Shockwave BCC	80	48.0 M	20.0 M (41%)	52%	520 k (71 %)	544 k (104 %)	3.90 (71%)

Table 3: Numerical results for the time-varying datasets used in our study. The *relevant voxels* are those voxels that have values above "air" and noise. The *relevant interpolated voxels* are the voxels interpolated for the arbitrarily chosen time step shown in Fig. 7. These voxels are passed into the splat renderer. The RLE is needed for efficient storage and transformation of these spatially non-connected points. The *render time* is the time (in seconds) to render the image.

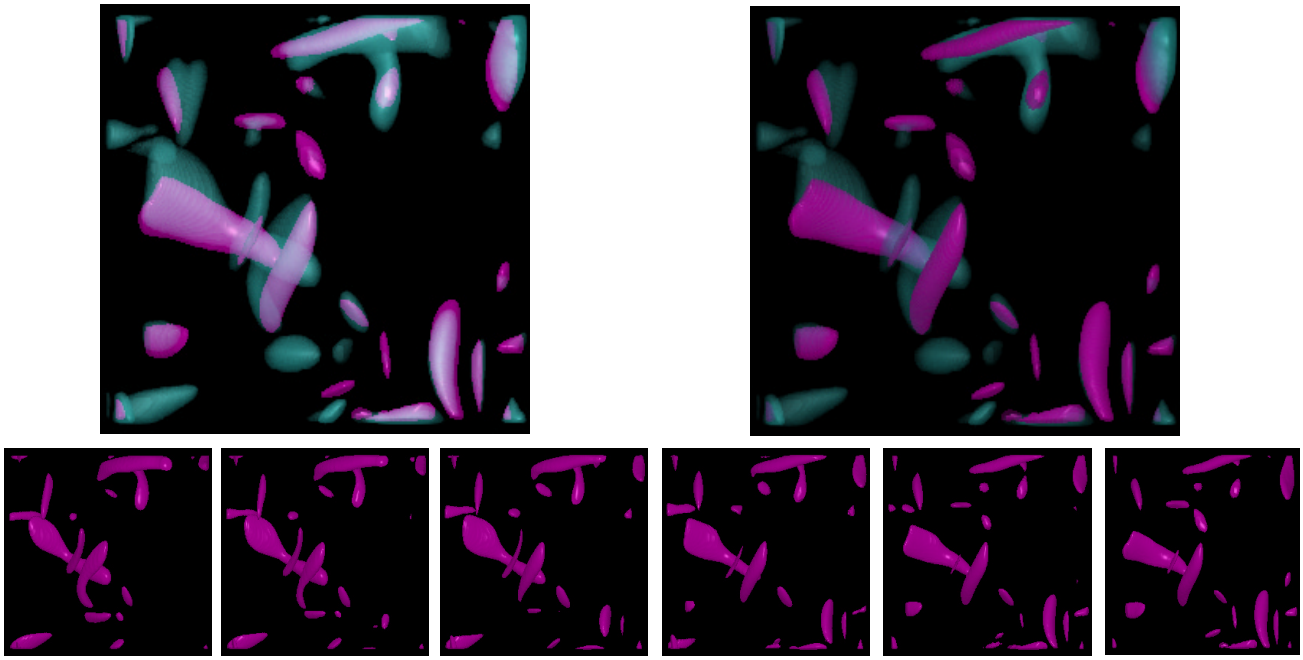


Figure 9: Turbulent vortex dataset with motion trail: Top left: averaged; top right: composited; bottom: time steps 50, 52, 54, 56, 58, 60.

speedups of 20-30%. The speedups are mainly for slices along one of the 4 major axes. For arbitrary axes the 4D interpolation is more expensive.

Rendering is not interactive. This is partially due to the fact that we do not use any type of hardware acceleration, which is in contrast to other methods published [8][30]. We are currently working on an approach that uses texture mapping hardware for splat rasterization which is the main bottleneck in our application. We are also currently exploring more cache-coherent data access. This has been shown to provide major speedups by [14][15][16], although especially the author of the last two references has done a lot more to optimize volume rendering in software.

In the presented work, we have also attempted to integrate the variation of the data in the 4th dimension into one still frame. Instead of glyphs, we use a variation of motion blur to depict past traces of data object into the display. We use a simultaneous interleaved volumetric rendering approach that composites slice-by-slice the current iso-surface along with its motion-blurred trace over a user-selectable interval along the 4th dimension, or perpendicular to the hyperslice. At the current time we can assign different colors to the two data objects. A possible extension would be to subdivide the motion interval and render each with a different transfer function. This multi-channel approach would enable a smooth degradation in color in relation to the time-stamp in the motion-blurred interval.

In future work, we would like to use 4D BCC grids in conjunction with lossy compression techniques, such as wavelets or DCT. An interesting result was reported in [38] for the 3D case. There it was shown that for larger datasets an entropy-coding (gzip) of BCC volumes yielded compression ratios that were 30% higher than those of the equivalent CC volumes. However, one drawback of further compression of the dataset is that extra work is required for decompression before voxel extraction and rendering can take place. Our present implementation can extract voxels right away. We would like to investigate these trade-offs further.

We would also like to explore higher-quality filters [24] for the BCC grid interpolation in order to eliminate, or at least lower, the smoothing artifacts apparent in the images we rendered from the BCC grids. However, the best strategy would be to acquire the volumes directly on BCC grids, which would eliminate the need for the intermediate interpolation step. For this purpose, we plan to evaluate BCC renderings of CT volumes that will be reconstructed both on BCC grids as well as on CC grids [27].

Acknowledgements

This research was supported by NSF CAREER grant ACI-0093157 and DOE grant MO-068. The turbulent vortex data set (described in [34]) is courtesy of Deborah Silver at CAIP of Rutgers University. The turbulent jet data is courtesy of Robert Wilson at the IIHR of the University of Iowa. The Jet Shockwave data is part of the Advanced Visualization Technology Center's data repository, courtesy of the University of Chicago. The other datasets were obtained from www.volvis.org, maintained by Michael Meißner. We would like to thank Kwan-Liu Ma for his help in obtaining the 4D datasets, and Tom Theußl and Torsten Möller for helpful discussions on BCC grids. Finally, we would like to thank the anonymous reviewers as well as Steve Kilthau for their valuable comments.

References

[1] K. Anagnostou, T. Atherton and A. Waterfall, "4D volume rendering with the Shear Warp factorisation," *Symp. Volume Visualization and Graphics'00*, pp. 129-137, October, 2000.

[2] C. Bajaj, C. Pascucci, G. Rabbiolo, and D. Schikore, "Hypervolume visualization: A challenge in simplicity," *Proc. 1998 Symposium on Volume Visualization*, pp. 95-102, 1998.

[3] P. Bhaniramka, R. Wenger, and R. Crawfis, "Isosurfacing in higher dimensions," *Proc. Visualization'00*, pp. 267-273, 2000.

[4] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware", *Symp. on Volume Visualization'94*, pp. 91-98, 1994.

[5] W. Cai and G. Sakas, "Data intermixing and multi-volume rendering.," *Computer Graphics Forum*, vol. 18, no. 3, (Eurographics'99), September 1999.

[6] J.H. Conway and N.J.A. Sloane, *Sphere Packings, Lattices and Groups*, 2nd edition, Springer Verlag, 1993.

[7] D. Dudgeon and R. Mersereau, *Multi-dimensional Digital Signal Pro-*

cessing, Prentice-Hall:Englewood Cliffs, 1984.

[8] K. Engel, M. Kraus, and T. Ertl, "High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading," *Proc. SIGGRAPH Graphics Hardware Workshop'01*, pp. 9-16, 2001.

[9] S. Guthe and W. Straßer, "Real-time decompression and visualization of animated volume data," *Proc. Visualization'01*, pp. 349-356, 2001.

[10] A. Hanson and P. Heng, "Four-dimensional views of 3D scalar fields," *Proc. Visualization'92*, pp. 84-91, 1992.

[11] A. Hanson and R. Cross, "Interactive visualization methods for four dimensions," *Proc. Visualization'93*, pp. 196-203.

[12] A. Hanson and P. Heng, "Illuminating the fourth dimension," *IEEE Computer Graphics & Applications*, vol. 12, no. 4, pp. 54-62, 1992.

[13] Y. Ke and E. Panduranga, "A journey into the fourth dimension," *Proc. Visualization'89*, pp. 219-229, 1989.

[14] S. Kilthau and T. Möller, "Splatting Optimizations", Technical Report, School of Computing Science, Simon Fraser University, (SFU-CMPT-04/01-TR2001-02), April 2001.

[15] G. Knittel, "The ULTRAVIS system," *Proc. Volume Visualization and Graphics Symposium 2000,* pp. 71-80, 2000.

[16] K. Knittel, "High-speed software raycasting on a dual-CPU PC using cache-optimizations, MMX, Streaming SIMD extensions, multitherading and DirectX," *Visual Data Exploration and Analysis VII, Proc. SPIE Vol. 3960*, pp. 164-174, 2000.

[17] P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," *Proc. SIGGRAPH '94*, pp. 451- 458, 1994.

[18] M. Levoy, "Display of surfaces from volume data", *IEEE Computer Graphics. & Applications*, vol. 8, no. 5, pp. 29-37, 1988.

[19] E. Lum, K.-L. Ma, and J. Clyne, "Texture hardware assisted rendering of time-varying volume data," *Proc. Visualization'01*, pp. 262-270, 2001.

[20] S. Marschner and R. Lobb, "An evaluation of reconstruction filters for volume rendering," *Proc. Visualization'94,* pp. 100-107, 1994.

[21] S. Matej and R.M. Lewitt, "Efficient 3D grids for image reconstruction using spherically-symmetric volume elements," *IEEE Transactions on Nuclear Science*, vol. 42, no 4, pp 1361-1370, 1995.

[22] N. Max, P. Hanrahan, and R. Crawfis, "Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions," *Computer Graphics*, vol. 24, no. 5, pp. 27-33, 1990.

[23] M. Meißner, U. Kanus, and W. Straßer, "VIZARD II: A PCI-card for Real-Time Volume Rendering", *Proc. Siggraph/Eurographics Workshop on Graphics Hardware'98,* pp. 61--67, 1998.

[24] T. Möller, R. Machiraju, K. Mueller, and R. Yagel, "Evaluation and Design of Filters Using a Taylor Series Expansion", *IEEE Transactions on Visualization and Computer Graphics,* vol. 3, no. 2, pp. 184-199, 1997.

[25] K. Mueller and R. Crawfis, "Eliminating Popping Artifacts in Sheet Buffer-Based Splatting," *Proc. Visualization'98,* pp. 239-245, 1998.

[26] K. Mueller, T. Möller, and R. Crawfis, "Splatting without the blur," *Proc. Visualization'99*, pp. 363-371, 1999.

[27] K. Mueller and R. Yagel, "The use of dodecahedral grids to improve the efficiency of the Algebraic Reconstruction Technique (ART)," *Annals of Biomedical Engineering,* p. S-66, 1996.

[28] K. Mueller, N. Shareef, J. Huang, and R. Crawfis, "High-quality splatting on rectilinear grids with efficient culling of occluded voxels," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 2, pp. 116-134, 1999.

[29] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler, "The VolumePro real-time raycasting system," *Proc. SIGGRAPH 99,* p. 251-260, Los Angeles, CA, August 1999.

[30] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl, "Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage-rasterization" *Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware'00*, pp. 109-118, 2000.

[31] J. Rossignac, "Considerations on the interactive rendering of four-dimensional volumes," *Chapel Hill Workshop on Volume Visualization*, pp. 67-76. 1989.

[32] H.-W. Shen, L. Chiang, and K.-L. Ma, "A fast volume rendering algorithm for time-varying fields using a time-space partitioning tree," *Proc. Visualization'99*, pp. 371-377, 1999.

[33] H.-W. Shen and C. Johnson, "Differential volume rendering: A fast volume visualization technique for flow animation," *Proc. Visualization'94*, pp. 180-187, 1994.

[34] D. Silver and X. Wang, "Tracking and Visualizing Turbulent 3D features," *IEEE Transactions on Visualization and Computer Graphics,* vol. 3, no. 2, 1997.

[35] P. Shirley and A. Tuchman. "A polygonal approximation to direct scalar volume rendering," *Computer Graphics*, vol. 24, no. 5, pp. 63-70, (San Diego Workshop on Volume Rendering), 1990.

[36] P. Sutton and C. Hansen, "Isosurface extraction in time-varying fields using a temporal branch-on-need tree (T-BON)," *Proc. Visualization'99*, pp. 147-153, 1999.

[37] J. Sweeney and K. Mueller, "Shear-Warp Deluxe: The Shear-Warp algorithm revisited," *Joint Eurographics - IEEE TCVG Symposium on Visualization 2002*, pp. 95-104, Barcelona, Spain, May 2002.

[38] T. Theussl, T. Möller, and E. Gröller, "Optimal regular volume sampling," *Proc. Visualization'01*, pp. 91-98, 2001.

[39] H. Tuy and L. Tuy, "Direct 2D display of 3D objects", *IEEE Computer Graphics & Applications*, vol. 4 no. 10, pp. 29-33, 1984.

[40] C. Weigle and D. Banks, "Extracting iso-valued features in 4-dimensional datasets," *Proc. Visualization'98*, pp. 103-110, 1998.

[41] R. Westermann, "Compression domain rendering of time-resolved volume data," *Proc. Visualization'95*, pp. 168-174, 1995.

[42] L. Westover, "Footprint evaluation for volume rendering", *Proc. SIGGRAPH'90*, pp. 367-376, 1990.

[43] J. Wilhelms and A. Van Gelder, "Octrees for faster isosurface generation," *ACM Trans. on Graphics*, vol. 11, no. 3, pp. 201-227, 1992.