Hierarchical Splatting of Scattered 4D Data

Matthias Hopf* Michael Luttenberger* Thomas Ertl*

November 18, 2003

Abstract

Visualizing large uncorrelated point data sets with regular volume rendering produces unsatisfying results due to low spatial resolution. We present a system that can deal with time-varying uncorrelated data at high resolution and interactive rates.

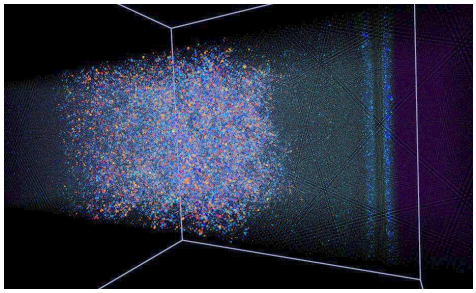


Figure 1: Molecular dynamics simulation with one million particles per time step.

Quite a number of physical simulations create large point-based data sets, for example molecular dynamics (Fig. 1), n-body simulations in astrophysics (Fig. 2), and smoothed particle hydrodynamics (SPH) (Fig. 17). Other sources of scattered point data are astronomical observations where new techniques for measuring three dimensional positions of stars as in the GAIA project will create huge real-world data sets in the near future as well. These data sets contain up to hundreds of millions of points, each with information about position $\mathbf{x}_i$, diameter s_i, and intensity c_i at various wavelengths. We cannot visualize these data sets interactively by simply rendering millions of colored points for each frame.
Therefore, in many visualization applications a scalar density corresponding to the point distribution is resampled on a regular grid for direct volume rendering. However, this technique imposes a low-pass filter on the data set, and many fine details are

*{hopf,luttenml,ertl}@vis.uni-stuttgart.de,
Visualization and Interactive Systems Group,
University of Stuttgart, Germany

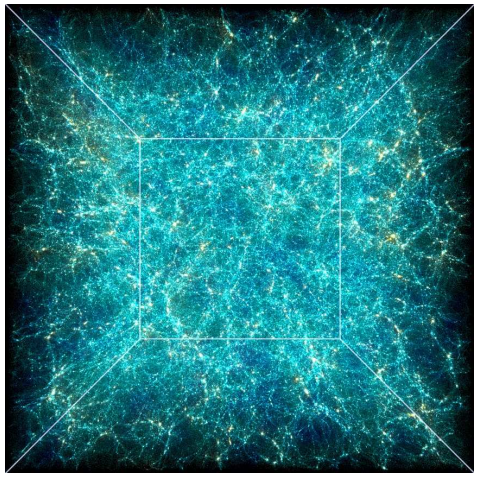


Figure 2: n-body simulation with 64 million particles.

usually lost for voxel resolutions which still allow interactive visualization on standard workstations. For time-dependent data sets things get even worse, as complete volume data sets have to be downloaded to the graphics hardware for each time step. Since no surface geometry is associated with our data sets, the recently introduced point-based rendering algorithms cannot be applied as well.
In order to allow scientists to view these data sets at high resolution interactively on desktop workstations or PCs, we want to visualize the scattered data directly without resampling them to a volume density. We propose to accelerate the visualization of scattered point data by a hierarchical data structure based on a principal component analysis (PCA) clustering procedure. By traversing this structure for each frame we can trade-off rendering speed vs. image quality, and lower hierarchy levels can be used during interaction. Our scheme also allows interpolating the given point positions using cubic splines, and it reduces memory consumption by using quantized relative coordinates and Lempel-Ziv compression of delta encoded control points. Additionally, it supports fast approximative sorting of semi-transparent clusters. We demonstrate that

Related Work

There has been quite a lot of work in the area of using footprints as rendering primitives for sampled data — for a more thorough overview please read our previous paper [2].
Researchers like Mueller et al. [6], Swan et al. [10], and Zwicker et al. [11] focus mainly on the improvement of the visual quality of texture splatting; however, the techniques described in these papers only apply to the reconstruction of continuous functions, e.g. for volume rendering of regular grid data, and they do not address adaptive rendering or data size reduction.
Using points as rendering primitives is a topic of ongoing research. However, almost all publications in this area deal with the rendering of geometric surfaces, as can be seen in the other articles of this CG&A issue. As the intrinsic model of points describing a surface is fundamentally different to the model used for scattered data, their clustering techniques cannot be applied in our case.
Pauly et al. [7] used PCA for clustering, but with a different hierarchy concept compared to our approach. Some systems like QSplat [8] use quantized relative coordinates for storing the points in a hierarchical data structure, but these approaches were not optimized for fast GPU access, because the data structures had to be interpreted by the CPU. Additionally, the presented rendering techniques have been designed to create smooth surfaces without holes and they allow no or only few layers of transparency. Again, this does not meet our requirements.
Meredith and Ma [5] introduced multiresolution splatting for rendering irregular volume data. Most of the techniques they developed deal with the handling of the unstructured data, but the adaptive rendering based on octrees could be the basis for an extended algorithm working with grid-less data as well. Jang et al. [3] introduced a multiresolution splatting approach for non-uniform data. However, in their solution the higher hierarchy levels are always stored in uniform grids, and they cannot render more than approximately 135,000 splats per second. This technique seems to be more appropriate for almost flat and regular data.
As PCA is a standard technique, we only present a short summary of the PCA-split algorithm, more details can be found e.g. in [4]. More information about interpolation using cubic splines can be found for example in [1]. Our system uses Lempel-Ziv for compression, a standard technique that is covered e.g. by [9].

References

[1] G. Farin, J. Hoschek, and M. S. Kim. *Handbook of Computer Aided Geometric Design*. Academic Press, 2002.

[2] M. Hopf and T. Ertl. Hierarchical Splatting of Scattered Data. In *Proc. Visualization '03*, pages 433–440. IEEE, 2003.

[3] J. Jang, W. Ribarsky, C. D. Shaw, and N. Faust. View-Dependent Multiresolution Splatting of Non-Uniform Data. In *Proc. Eurographics VisSym '02*, pages 125–132. IEEE, 2002.

[4] I. T. Jolliffe. *Principal Component Analysis*. Springer, New York, 1986.

[5] J. Meredith and K.-L. Ma. Multiresolution View-Dependend Splat Based Volume Rendering of Large Irregular Data. In *Symposium on Parallel and Large-Data Visualization and Graphics '01*, pages 92–99. IEEE, 2001.

[6] K. Mueller, T. Möller, and R. Crawfi s. Splatting without the Blur. In *Proc. Visualization '99*, pages 363–370. IEEE, 1999.

[7] M. Pauly, M. Gross, and L. Kobbelt. Effi cient Simplifi cation of Point-Sampled Surfaces. In *Proc. Visualization '02*, pages 163–170. IEEE, 2002.

[8] S. Rusinkiewicz and M. Levoy. QSplat: A Multiresolution Point Rendering System for Large Meshes. In *Proc. SIGGRAPH '00*, pages 343–352. ACM, 2000.

[9] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 2000.

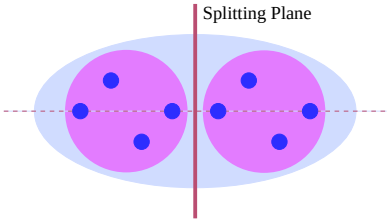
[10] J. E. Swan, K. Mueller, T. Möller, N. Shareef, R. Crawfi s, and R. Yagel. An Anti-Aliasing Technique for Splatting. In *Proc. Visualization '97*, pages 197–204. IEEE, 1997.

[11] M. Zwicker, H. Pfi ster, J. van Baar, and M. Gross. EWA Volume Splatting. In *Proc. Visualization '01*, pages 29–36. IEEE, 2001.

The PCA Split Algorithm

Principal component analysis can be used to find a splitting plane for a set of points, that divides the set into two clusters, so that the distortion of the individual sets gets minimal. Basically, this plane is perpendicular to the Eigenvector corresponding to the smallest Eigenvalue of the inertial tensor. After some simplifications, you finally get the PCA split algorithm:

- Select cluster j (point indices I_j) with largest distortion Δ_j
- Calculate auto-covariance matrix from centroid X_j: $A = \sum_{i \in I_j} (\mathbf{x}_i - \mathbf{X}_j)(\mathbf{x}_i - \mathbf{X}_j)^T$
- Find Eigenvector $e_{\max}$ of A corresponding to the largest Eigenvalue $\lambda_{\max}$
- Split cluster j into two new clusters: $I_{n1} = \{i \in I_j : \langle \mathbf{x}_i - \mathbf{X}_j, e_{\max} \rangle \geq 0\}$ $I_{n2} = \{i \in I_j : \langle \mathbf{x}_i - \mathbf{X}_j, e_{\max} \rangle < 0\}$
- Calculate centroids and distortions for the new clusters



For additional information about PCA please have a look at the related work section.

we can now visualize time-dependent data sets with millions of points interactively with sub-pixel screen space error on current PC graphics hardware employing advanced vertex shader functionality.

1 Creating the Hierarchy

In order to create one level of the hierarchy the input data points have to be sorted into bins. For each bin a point on the next higher hierarchy level is created, representing all points that fell into that bin. The properties of the newly created point are chosen so that its visual representation matches that of the substituted points best. To obtain the set of bins several clustering schemes can be used. The most common solution is to subdivide the data set into an octree.

Another approach that has much better spatial adaptation properties is to perform a series of PCA splits, each dividing the cluster with the currently highest distortion as defined below into two halves.

In the following I_j denotes the set of indices of the points of cluster j. That is, cluster j consists of all points $\mathbf{x}_i$, $i \in I_j$ with diameters s_i, and it has the weighted centroid $\mathbf{X}_j$ and distortion Δ_j with

$$\mathbf{X}_j = \frac{\sum_{i \in I_j} s_i \cdot \mathbf{x}_i}{\sum_{i \in I_j} s_i} \quad , \qquad \Delta_j^2 = \sum_{i \in I_j} \left\| \mathbf{x}_i - \mathbf{X}_j \right\|_2^2 \quad .$$

These equations only depict the steady state — for time-varying data sets with T time steps, $\mathbf{x}_i \in \mathbf{R}^{3T}$ represents the position of a single point in all time steps simultaneously. Of course, the PCA split has to be implemented for higher dimensional matrices in this case.

This splitting process is continued until the maximum distortion or the average cluster size falls below pre-defined minima. After the visual properties of the new points have been obtained, these points undergo another series of PCA-splits in order to create the next hierarchy level.

For creating a visually approximative representation of the cluster j compared to its children the most important aspect is that the radiant flux Φ has to be the same. For the irradiance c_j of the new centroid point representing the cluster with area A_j this means for each of the representative wave lengths

$$\Phi_j = A_j \cdot c_j = \frac{\pi}{4} s_j^2 \, c_j = \frac{\pi}{4} \sum_{i \in I_j} s_i^2 \, c_i \quad . \qquad (1)$$

The cluster representative should be larger than the largest of its children in order to keep some visual continuity. Additionally, small cluster points would have very high local intensities, which could finally saturate the covered pixels in the blending step during rendering. Distributed clusters — that is clusters with a large average distance of their children to the centroid compared to the children's point sizes — should have larger representatives than locally agglomerated ones. On the other hand, they must not be too large, as the human eye is very sensitive to edges, and enlarging a point implies reducing its intensity, diminishing the visibility of the edge.

After comparing several different functions, we found a trade-off that creates acceptable results for almost all point distributions. It tries to combine point sizes and their distances to the centroid, and ensures, that the final size does not fall below the size of the largest point of the cluster:

$$\begin{aligned} m_j &= \operatorname*{argmax}_{i \in I_j} s_i \quad , \\ s_j &= \frac{0.5}{|I_j| - 1} \sqrt{\sum_{i \in I_j \setminus \{m_j\}} s_i \left\| \mathbf{x}_i - \mathbf{X}_j \right\|_2} \\ &\quad + s_{m_j} \quad . \end{aligned} \qquad (2)$$

The scaling factor $\frac{1}{2}$ in Eq. 2 of the weighted average point size of all points except the largest one

before adding to the largest point size s_{m_j} has been determined empirically.
This calculated point size is subject to further restrictions, if intensities are stored in main memory as unsigned bytes in order to save memory. The system has to assure that the calculated point size does not overflow the intensity domain, and it has to increase the point size in case of saturation.
Eq. 1 and Eq. 2 are highly dependent on the blending function, and our definition only holds for cumulative blending ($C = c_1 + c_2$). For other blending functions, like the over operator ($C = \alpha_1 c_1 + (1-\alpha_1)c_2$), c_j may be view dependent, as the total flux of overlapping points is no longer necessarily the sum of the individual fluxes of the points. With the current approach view dependent intensities cannot be modeled. However, with adaptive rendering we will use coarser hierarchy levels only for clusters that are projected to areas on the screen that are small or outside some region of interest, and it is very unlikely that view dependencies will be noticed in such small regions.

2 Interpolating Coordinates

In order to have a smooth visualization of time-varying data, the positions of the points have to be interpolated between the given time steps. We are using cubic splines for the evaluation of particle positions, which will help us with compressing the data as well.
First, the equations for evaluating the spline coefficients are set up and solved for each data point. For n points this process creates $3n-2$ different coefficients that need to be stored in order to evaluate the splines at arbitrary positions. Luckily, the coefficients are not as independent from each other as it seems, and we can convert the Bezier splines to B-Splines and store the control points $\mathbf{D}_j = \mathbf{b}_{j,2} + 2(\mathbf{b}_{j,1} - \mathbf{b}_{j,2})$ instead. The spline coefficients can be easily regenerated during rendering:

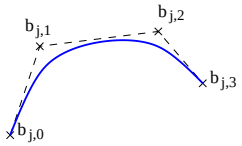
$$\mathbf{b}_{j,0} = \mathbf{b}_{j-1,3} = \frac{\mathbf{D}_{j-1} + 2\mathbf{D}_j + \mathbf{D}_{j+1}}{6} \quad ,$$

$$\mathbf{b}_{j,1} = \frac{2\mathbf{D}_j + \mathbf{D}_{j+1}}{3} \quad , \quad \mathbf{b}_{j,2} = \frac{\mathbf{D}_j + 2\mathbf{D}_{j+1}}{3} \quad .$$

Thus, only $n+2$ control points have to be stored.
During rendering the splines are evaluated in order to reveal the positions of the particles at the rendering time step t. For this process, the 4 nearest control points are needed. When t crosses spline segment boundaries, only one control point has to be loaded as the other three control points can be reused from the last segment by means of a ring buffer.

Cubic Bezier Splines

Cubic splines are defined by piecewise cubic Bezier segments with some additional constraints. Bezier curves can be created using the so-called Bernstein polynomials B_i^n by



$$\mathbf{x}_j(t) = \sum_{i=0}^{3} \mathbf{b}_{j,i}\, B_i^3(t) \quad , \quad B_i^n(t) = \binom{n}{i} (1-t)^{n-i} t^i \quad .$$

The segments are stitched together so that the given points $\mathbf{x}_j$ are interpolated and the transition between the segments is smooth (C^2 continuous):

$$\mathbf{x}_j(0) = \mathbf{b}_{j,0} = \mathbf{x}_j \quad , \quad \mathbf{x}_j(1) = \mathbf{b}_{j,3} = \mathbf{x}_{j+1} \quad ,$$

$$\frac{d\mathbf{x}_j(1)}{dt} = \frac{d\mathbf{x}_{j+1}(0)}{dt} \quad , \quad \frac{d^2\mathbf{x}_j(1)}{dt^2} = \frac{d^2\mathbf{x}_{j+1}(0)}{dt^2} \quad .$$

By inserting the derivatives into these formulas we get the additional conditions

$$\mathbf{x}_j = \frac{\mathbf{b}_{j-1,2} + \mathbf{b}_{j,1}}{2} \quad ,$$

$$\mathbf{b}_{j-1,1} + 2(\mathbf{b}_{j-1,2} - \mathbf{b}_{j-1,1}) = \mathbf{b}_{j,2} + 2(\mathbf{b}_{j,1} - \mathbf{b}_{j,2}) \quad .$$

Together with one additional constraint at each boundary this leads to a set of equations, which have to be solved for each point of the hierarchy.

3 Data Storage and Compression

Point data sets tend to get really large, and they need high positional resolution. Memory requirements can be reduced to one half or even one quarter by storing coordinates relatively to the centroids of the inspected clusters as depicted in Fig. 3. As the necessary positional resolution is much lower for encoding relative coordinates, they can be quantized using bytes or shorts instead of floats.
However, for time-varying data we do not store coordinates, but the control points of the splines. The splines do no longer encode particle positions directly, but the difference vectors of the points to the parent centroids. As these vectors tend to get smaller with each level of the hierarchy, we can encode the spline coefficients with smaller quantization factors for higher levels, and still get a high

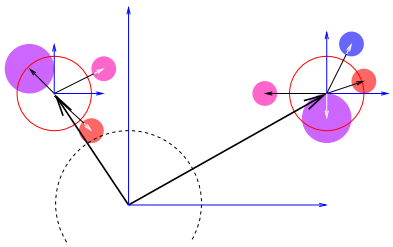


Figure 3: Point coordinates can be scaled and quantized relatively to the position of the cluster centroid for storage.

positional resolution. Figure 4 shows this relation. Note that during clustering different levels may have different time resolutions for the spline coefficients due to memory constraints. The control points in between can then be calculated using subdivision.

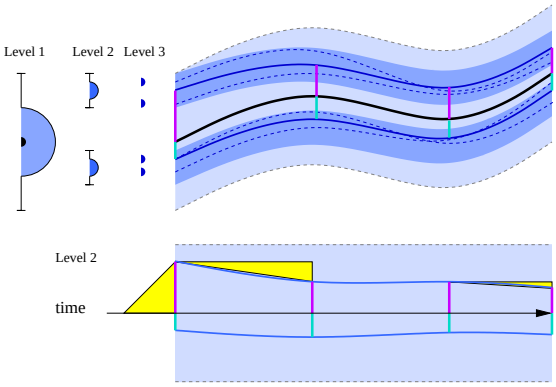


Figure 4: For time-varying data spline coefficients are encoded instead of particle positions. Encoding points in the hierarchy relative to their parent centroids creates smaller extension tubes for higher levels. Thus we can quantize the spline coefficients with a few bits only, and still get a high positional resolution. Additionally, all control points are delta encoded in time.

Particles in time-dependent data sets usually show high positional coherence with respect to their neighbors. In order to exploit this regularity, we are not encoding the relative vectors themselves by splines, but their differences to the previous time step. Ideally, for smooth particle flows this delta coding would create a sequence of zeros on the higher levels, with only few exceptions. This sequence can be further compressed using Lempel-Ziv with Huffman coding. This combination of techniques has been chosen because of its high decompression speed.
Using hierarchical structures imposes higher memory requirements than storing the same data in flat arrays. Memory bandwidth is limited, and traversing the hierarchy for rendering adds overhead for recursive function calls and pointer dereferencing. Additionally, with current graphics APIs there is no means to hand this process over to the GPU. Therefore, we decoupled the hierarchy structures (clusters) from data structures (points). Each cluster contains a pointer to the next hierarchy level, a pointer to offspring point data, and the number of children. The point data itself only contains the spline control points for coordinates and size, and color values, as depicted in Fig. 5. In principle one would like to store raw data values and use runtime classification for point size and color selection, but the cluster hierarchy itself and especially the pre-processed cluster representatives highly depend on point sizes and colors.

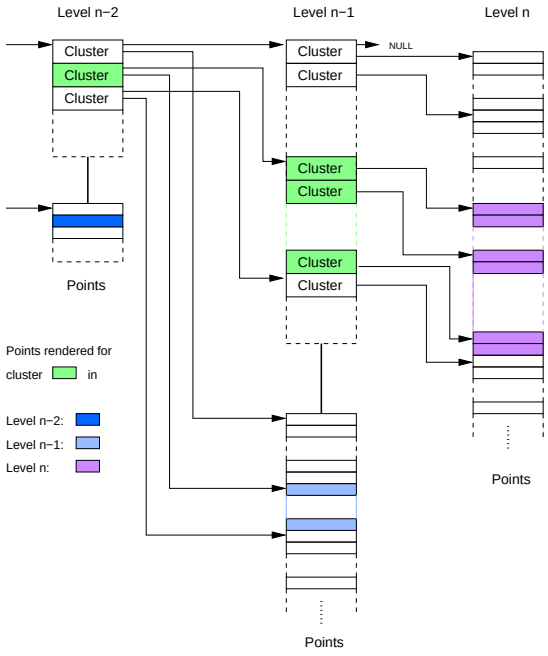


Figure 5: The last three levels of a typical data hierarchy. The point data correlated to the cluster centroids is not embedded in the clusters but stored in point structures parallel to the clusters. The green clusters show that all data structures that are related to their offsprings are packed tightly, which improves cache coherence and enables us to render all points of a single cluster in a tight loop.

Finally, the encoded spline control points are stored into a file together with hierarchy information and index pointers for fast out-of-core access during the rendering process.

4 Hierarchy Traversal

During rendering the hierarchy is traversed recursively. For each cluster the system may decide to descend further down into the hierarchy, render the centroid at the current level, or skip the cluster altogether when it is not visible. The decision can be based upon some maximum screen error metrics, the distance to the viewer, or some given region of interest. These rules should be computationally cheap. As a rule of thumb, evaluating the rule should be cheaper than interpolating, transforming, and rendering one point of the cluster.

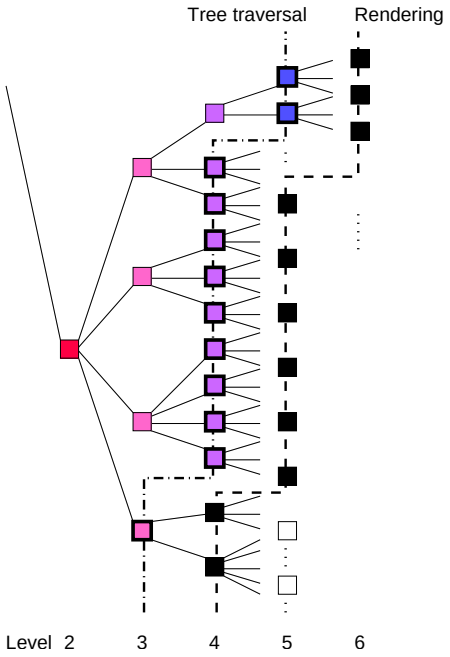


Figure 6: During traversal, the final rendering level should be selected at some higher level of the hierarchy for speedup reasons ($\delta_n = 1$ in this case).

For more complex rules and for accelerating the traversal process, the system may already decide on a higher level n, that it will render all offsprings of level $n + \delta_n$ (see Fig. 6). Then the children do not have to be traversed. Even for simple adaptivity rules this has a strong effect on rendering performance. However, $\delta_n > 1$ complicates the interpolation process, as several levels of spline coefficients have to be evaluated. For time-dependent data sets the complex interpolation easily annihilates any advantages in these cases.
As described in Sect. 3, the point data of all children is stored sequentially in the file, thus they can be read, decompressed, interpolated, and rendered in tight loops. Interpolation and rendering could even be done by a single OpenGL array rendering call using advanced vertex shader functionality. See Fig. 7 for a pseudo code fragment.

```
void render_cluster (cluster_t *c,
                     time_t t, int level) {
  if (cluster_visible (c)) {              // trivial reject
     if (spline_segment_changed (level, t)) {
       // shift ring buffer of spline control points
       shift       (c, c→len)
       // load out-of-core data
       read        (c, c→len, level, t)
       // lz decompression + delta coding
       decompress  (c, c→len)
     }
     interpolate (c, c→len, t)        // spline evaluation
     if (descend_cluster (c)) {
       cluster_t *cn = c→children
       for i = 0...c→len                    // recursion
         render_cluster (&cn[i], t, level+1)
     } else {
       render_points (c→pts, c→len)     // render δn = 1
} } }
```

Figure 7: Pseudo code for hierarchical rendering. Cluster and point data are combined here into a single structure for clarity.

5 Sorting

For many of the investigated data types cumulative blending is an effective way of visualizing both global and local structures in the data sets. However, with other data sets, for instance reversible Apollonian packings (Fig. 19), the over operator is necessary to visualize the visual depth. Non-commutative blending operators require the data points to be sorted according to view distance. The implemented hierarchy can be used to efficiently sort the cluster centroids using quicksort or bucketsort. The cluster points themselves have to be sorted before rendering as well. Bucketsort only creates an approximative sorting order, but has the advantage of lower computational complexity ($O(n)$ vs. $O(n \log n)$) and its implementation is much simpler and thus faster. The rendered images are almost indistinguishable when using a relatively large number of buckets.
Sorting the cluster centroids is equivalent to the typical BSP-tree sorting, as long as the distances of any two neighboring centroids to their common splitting plane are equal (Fig. 8). The octree clustering approach has this property, however, its spatial adaption to the local point density is much worse than the proposed PCA-split approach. Still, we have no choice but to use octrees if we really care about the correct sorting order.
With time-dependent data sets, things become even worse. Clusters do no longer subdivide the space

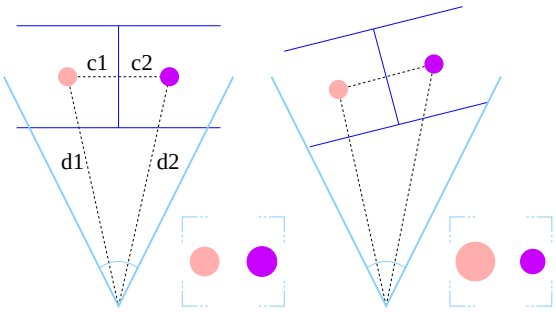


Figure 8: Distance sorting according to $d1$, $d2$ is equivalent to BSP sorting for $c1 = c2$. Note that the sorting order of $d1$, $d2$ changes exactly at the same time the visibility order of the two cells changes.

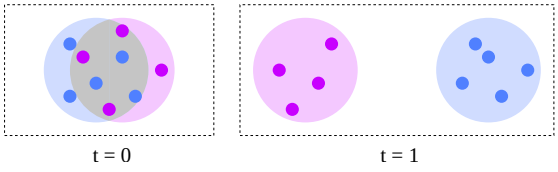


Figure 9: Points that are nearby in one time step may be split apart in the future, and may thus belong to different clusters. These clusters overlap in the former time step and cannot be sorted correctly.

into cells in 3D, as they may even overlap, if their children are grouped tightly together in one time step, and split apart in another. Figure 9 depicts an example. Even if we have a decently behaving data set that does not show this property, we can have particles crossing cluster boundaries due to spline interpolation, as shown in Fig. 10. So far, these issues are unresolved.

Even with correctly sorted clusters, there is a chance that overlapping points are rendered in the wrong order, as it can be seen in Fig. 11. For many data sets the points can be thought to be infinitely small, in that case the points are rendered correctly. Other data sets are more sensitive to their sorting order, and require larger average cluster sizes by combining several octree levels to a single level or using large PCA cluster sizes. This helps reducing the chance of sorting errors, as the points of a single cluster are always rendered in the correct order, except for overlapping clusters. For the static case in order to sort the cells produced by the PCA-splits, additional connectivity and splitting plane information is needed for MPVO or equivalent algorithms. This implies a huge additional memory overhead, and it will not help with overlapping clusters and points that drop out of their cluster boundaries.

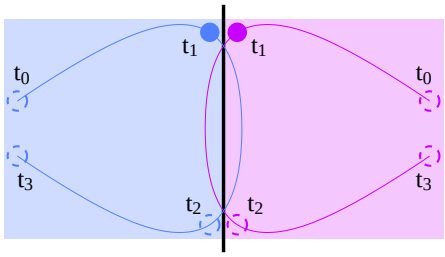


Figure 10: Even points of non-overlapping clusters can cross cluster boundaries between time steps due to interpolation.

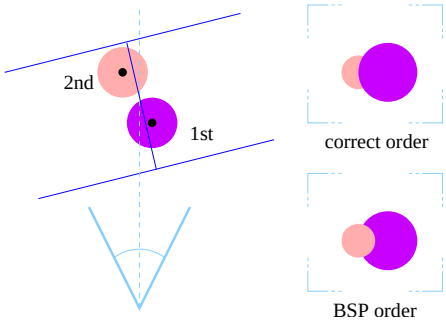


Figure 11: Back-to-front distance sorting according to BSP fails for non-split overlapping points. In this example the right cluster is rendered before the left one due to BSP order.

6 Rendering

Since using only one vertex per primitive can accelerate the rendering process significantly, we will usually approximate the splats using OpenGL antialiased points. Other footprints can be used by rendering point sprites without additional cost (see Fig. 19), but they are only available on NVidia hardware right now. On ATI's Radeon series, fragment programs can be used in order to emulate sprites, though. For rendering large quantities of points the generally fastest approach is to use vertex coordinates and attributes that are given by vertex arrays or display lists. As we create the vertex positions on the fly from out-of-core data, we cannot store them in GPU memory. Still, it is much faster to fill render buffer caches with the calculated vertex positions and render these buffers in one go, instead of sending each and every vertex with an individual operation to the graphics hardware.

When a point projects to an area with diameter $\tilde{s}$ smaller than a single pixel on the screen, its brightness has to be attenuated. The new alpha value is

$$\tilde{\alpha} = \alpha \cdot \tilde{s}^2 \quad , \tag{3}$$

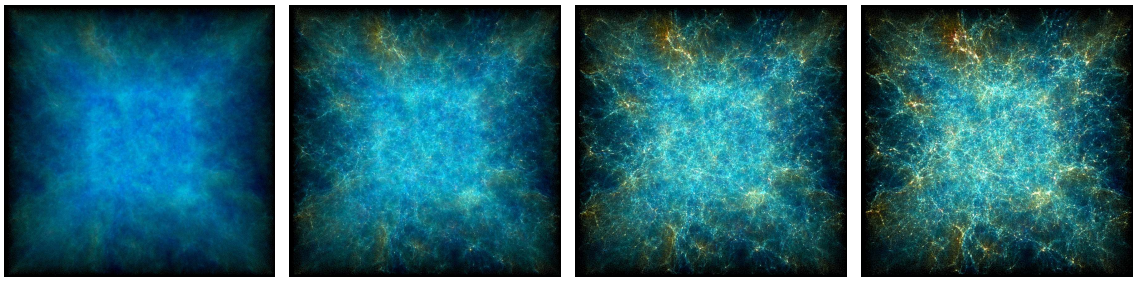


Figure 12: Some frames of an animated view of one of the VIRGO n-body simulations consisting of 10 time steps with 16 million points each.

assuming that point color is multiplied with alpha during blending. Note that attenuation increases quantization artifacts due to the limited frame buffer depth. Therefore, adaptive rendering can even improve the image quality by choosing higher levels for parts of the cluster that tend to project to very small screen areas.

For drawing points with varying sizes we can use vertex programs on programmable graphics hardware. Besides changing the point size on a per vertex level, the vertex shader is responsible for correct alpha attenuation as indicated in Eq. 3. Now we can use vertex arrays for accumulating the evaluated point positions into a render buffer cache to send a large part of the hierarchy to the graphics hardware. These calls are highly optimized, and the CPU can already continue to select the next cluster to be rendered in parallel to the rasterization process.

Currently, the rendering process is dominated by the evaluation of the cubic splines for finding the current point positions. As long as the points do not need to be sorted, this could be handed over to the vertex program as well by sending the control points to the GPU.

For comparison, several other techniques have been developed and integrated into the rendering framework. The different rendering backends can be selected during runtime at almost no cost. We want to be able to use the system in virtual reality environments as well, which are still often based on Silicon Graphics systems, in contrast to regular PC workstations used by typical end users. As the InfiniteReality hardware does not have a programmable graphics pipe, we additionally implemented a regular billboard renderer. Using billboards is less efficient compared to OpenGL anti-aliased points or point sprites, as four vertices have to be calculated and sent down the pipeline for a single data point.

7 Results and Discussion

The images in Figs. 2 and 12 to 15 show visualizations of n-body simulations carried out by the Virgo Supercomputing Consortium. All images show the τCDM model. The velocities of the galaxies relative to the simulated base cube have been color coded.

In Figs. 13 and 14 you can see different levels of the data set. Note that level 3 would usually not be used for rendering, but it is a potential level for deciding on the rendering depth, as shown in Fig. 6. Figures 17 to 19 show other data sets and rendering modes. Please note that the clearly visible aliasing in Fig. 17 is inherent to the according data set and not an artefact of the presented rendering technique. In most areas the data set contains an almost regular grid and the splats are used for visualizing the grid structure and not for approximating any underlying function.

Compared to our previous approach, which is optimized for static data (see related work section for reference), the rendering of time-depended data sets is much more involved and thus slower. We still get up to 38% of the speed of the static algorithm for $\delta_n = 1$, or approximately 4.6 million points per second, which is quite remarkable for the large overhead we get by the hierarchical spline interpolation. However, we can accelerate the visualization of static data by deciding the rendering depth on a higher level ($\delta_n > 1$), which is not manageable with time-dependent data. Rendering sorted points using bucket sort reduces the performance to approximately 2.9 million points per second. Loading the spline parameters for a single time step from the local hard disc takes less than one third of a second. The system used for the evaluation was an Pentium4 2800 MHz with an Intel 7225 chipset with 4 GB dual channel DDR 333 memory and a GeForce FX 5800 Ultra graphics pipe on a Windows XP system. The Linux drivers showed similar

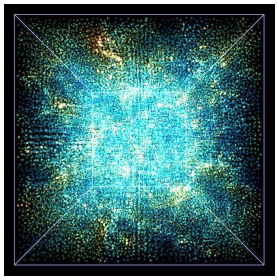


Figure 13: VIRGO at level 3. (123K clusters = 0.74%)

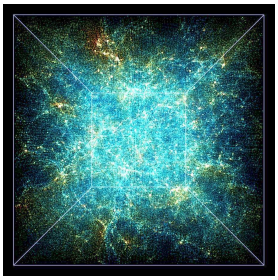


Figure 14: Level 4. (671K clusters = 4%)

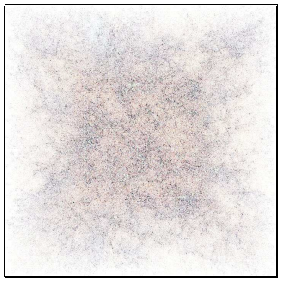


Figure 15: Differences of adaptive vs. full data rendering (contrast enhanced by 400%).

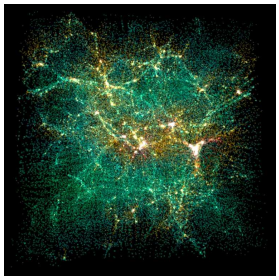


Figure 16: A dark matter SPH simulation from the Texas Advanced Computing Center (262K pts).

performance figures for the GeForce, while the performance of ATI's Radeon is still somewhat slower. Please note that time-varying data sets need at least four times the main memory for storing the spline control points. With these constraints, the interactive visualization of extremely large data sets like the one depicted in Fig. 2 is hardly possible on 32 bit systems for the unsteady case.

The adaptive algorithm we are using for selecting the best hierarchy level during rendering uses a simple adaptive scheme, selecting the clusters that should be traversed on the CPU by the maximum screen projection size of the cluster children and the given maximum traversal depth. If the projected size exceeds 2 pixels, the cluster is traversed further, otherwise its children are rendered to screen. Using these settings, there is almost no visual difference between the data set rendered in full resolution compared to the adaptive rendering. The difference image in Fig. 15 shows quite some changes in the visualization, however, they have the same quality as additional noise and do not disturb the visual appearance. Most of the screen space difference comes from points that happen to be rendered one pixel off to their original positions. While the human visual system is not able to notice these differences, they have a rather large impact on difference images. The contrast of the difference image has been enhanced by 400 percent in order to show its properties more clearly.

The cluster selection scheme has about the same performance impact on the rendering system as the flexible rendering backend (less than 2 percent each), which allows us to switch the rendering technique on the fly. Additionally, we get early view frustum culling at almost no cost for the adaptive rendering algorithm, as this can be incorporated in the point projection size calculation process. Please note that

Figure 20: A closeup of the Texas SPH data set, rendered at the coarse level 5 (12.8K pts = 11.8%, left), adaptively with approximately the same number of points and high potential projection error (12.3K pts, middle), and with all visible points (108K pts, right), respectively.

for large viewports like 1000^2 the effect of adaptive hierarchy traversal is not noticeable for low maximum traversal depths, as all clusters are traversed due to their large projected size.

Figures 16 and 20 show a dark matter n-body and smoothed particle hydrodynamics simulation carried out by the Texas Advanced Computing Center. The middle image in Fig. 20 shows a close up region of this data set, rendered intentionally with a very high projection size error of 36 in order to reveal the differences. The other two images show the same region rendered without adaption with approximately the same number of points, and with all points, respectively. Note that the projected screen size is only an approximation for the maximum screen space error, as the centroid size that used for evaluating the screen space error is not directly coupled with the maximum distribution width of the children, which influences the error as well.

For more images and some realtime animations please take a look at our web site:
`www.vis.uni-stuttgart.de/pointclouds/`

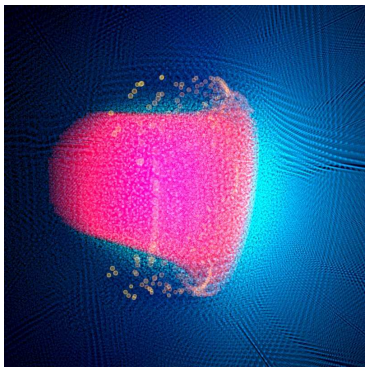


Figure 17: Visualization of a diesel injection shock front, simulated with SPH (2.5 million points per time step).

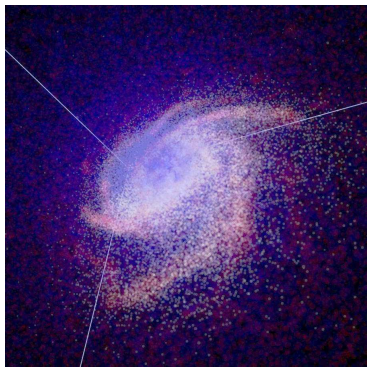


Figure 18: SPH and dark matter galaxy formation simulation rendered with sorted anti-aliased points (540,000 points).

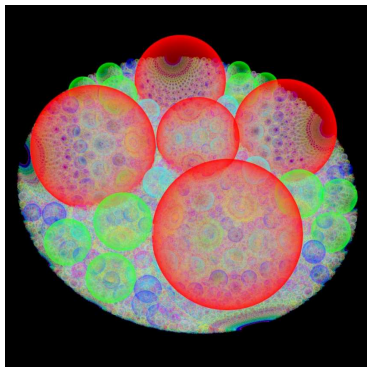


Figure 19: Reversible Apollonian packing of 6 million particles, rendered with sorted point sprites.

8 Conclusion and Future Work

In this article we presented a technique for accelerating the visualization of time-varying scattered point data. We employed principal component analysis for creating a hierarchy of point clusters, encoding the time varying point positions with cubic splines, which are stored out-of-core with quantized relative coordinates using delta encoding in a data structure that separates cluster from point data. With this data representation visualization quality can be traded for speed with an adaptive rendering algorithm. The rendering process itself was accelerated using vertex programs on modern PC graphics hardware. Finally, we are now able to visualize data sets with millions of moving points interactively on standard workstations.

With the current implementation, a lot of work is still done by the CPU that could be off-loaded to the graphics card. For instance, the evaluation of the particle positions using the cubic Bezier splines could be performed by the GPU, as long as commutative blending modes are used. Sorting the points for non-commutative blending modes has still several unresolved issues, though it works astonishingly well with most data sets.

One of the most promising — but also most challenging — extensions to the algorithm is the handling of time-varying clustering. This process will have to handle cluster transitions of single particles in a smooth way, such that popping artifacts will not occur.

Additionally, there are some issues with the overestimation of the radiant flux during rendering with cumulative blending in areas of high saturation. The system should detect these areas and reduce the brightness of the generated clusters accordingly. Alternatively, high dynamic range rendering to floating point frame buffers could be used.

An open question is how to handle multivariate data and how to change the visualized data during runtime. Storing several color values per point structure is one possibility, but this increases memory usage again. Color quantization and table lookup in the rendering step could help with regard to this aspect.

Acknowledgments

This work has been financed by the project SFB 382 of the German Research Foundation (DFG).

Some of the data sets shown in this article are based on simulations carried out by the Virgo Supercomputing Consortium using computers based at the Computing Center of the Max-Planck Society in Garching and at the Edinburgh Parallel Computing Center. The data sets are publicly available at `www.mpa-garching.mpg.de/NumCos/`.

Another data set shown in this article shows a cosmological simulation performed at the Texas Advanced Computing Center by Hugo Martel, Paul R. Shapiro, and Marcelo Alvarez, of the Galaxy Formation and Intergalactic Medium Research Group, Department of Astronomy, UT-Austin. The data set was downloaded from the Center for Computational Visualization, Institute of Computational Engineering and Sciences, UT-Austin. Please see `www.ticam.utexas.edu/CCV/` for more details.

We would like to thank Sven Ganzenmüller (TI) from the University of Tübingen, Volker Springel and Martin Jubelgas from the Max-Planck Society in Garching, Bong-Soo Sohn (ICES) from the University of Texas, as well as Reza Mahmoodi Baram (ICA) and Johannes Roth (ITAP) from the University of Stuttgart for their support with data sets and insight into their visualization needs. Additionally, we would like to thank our colleagues Marcelo Magallón, Guido Reina, and Daniel Weiskopf for helpful discussion.