Eurographics/ IEEE-VGTC Symposium on Visualization (2006)
Thomas Ertl, Ken Joy, and Beatriz Santos (Editors)

Direct Isosurface Extraction from Scattered Volume Data

Paul Rosenthal Lars Linsen

Department of Mathematics and Computer Science
Ernst-Moritz-Arndt-Universität Greifswald
Greifswald, Germany
{paul.rosenthal,linsen}@uni-greifswald.de

Abstract

Isosurface extraction is a standard visualization method for scalar volume data and has been subject to research for decades. Nevertheless, to our knowledge, no isosurface extraction method exists that directly extracts surfaces from scattered volume data without 3D mesh generation or reconstruction over a structured grid. We propose a method based on spatial domain partitioning using a kd-tree and an indexing scheme for efficient neighbor search. Our approach consists of a geometry extraction and a rendering step. The geometry extraction step computes points on the isosurface by linearly interpolating between neighboring pairs of samples. The neighbor information is retrieved by partitioning the 3D domain into cells using a kd-tree. The cells are merely described by their index and bitwise index operations allow for a fast determination of potential neighbors. We use an angle criterion to select appropriate neighbors from the small set of candidates. The output of the geometry step is a point cloud representation of the isosurface. The final rendering step uses point-based rendering techniques to visualize the point cloud.
Our direct isosurface extraction algorithm for scattered volume data produces results of quality close to the results from standard isosurface extraction algorithms for gridded volume data (like marching cubes). In comparison to 3D mesh generation algorithms (like Delaunay tetrahedrization), our algorithm is about one order of magnitude faster for the examples used in this paper.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-dimensional Graphics and Realism: Isosurface extraction

1. Introduction

Isosurface extraction is the most commonly used visualization technique for scalar volume data beside direct volume rendering. A vast number of isosurface extraction algorithms exists. Many of them address a specific problem of isosurface extraction when applied to certain types of grids. The grid types include all kinds of structured grids, irregular grids, and adaptively refined grids. The algorithms typically operate on hexahedral or tetrahedral cells, and frequently use an approach that marches through all the cells of the grid.

Although all these approaches have been introduced for various data structures, to our knowledge no algorithm exists that directly operates on unstructured or scattered volume data when no grid connectivity is given. To deal with scattered volume data, the data are typically resampled over a regular structured grid using scattered data interpolation techniques or converted into an irregular grid using a polyhedrization technique. While the former approach may produce resampling inaccuracies, the latter is typically computationally expensive and often rather cumbersome to implement.

With the improved technological capabilities of data acquisition systems, the amount and variety of data produced has increased substantially. One example that is gaining increasing attention is that of sensor systems, where a large number of sensors are placed to measure environmental parameters. The wireless sensors have become tiny and cheap and can transmit their measured values any time to a data-collecting server.

Another attractive property of scattered data approaches

is that they naturally include all grid-based configurations. A scattered data visualization approach can always be applied to a gridded data set by neglecting the grid connectivity. This approach does not seem to be worth pursuing at first glance. However, when dealing with adaptively refined grids, most isosurface extraction approaches introduce constraints on the adaptive refinement. Considering the data as being scattered would eliminate such constraints. Moreover, if the data come from simulations executed on various grids of different resolution, which may overlap, exhibit gaps, or suffer from discontinuities, treating the entire data set as scattered data may become an option.

We propose a method for direct isosurface extraction from scattered volume data, i. e. we are not resampling the data over a structured grid nor are we generating global or local polyhedrizations (e. g. Delaunay tetrahedrization). Instead, we compute points on the isosurface by interpolating between neighboring samples of the scattered volume data set. The resulting point cloud describes the isosurface, which is visualized using point-based rendering techniques. In Section 3, we describe in more detail the processing pipeline of our isosurface extraction and rendering method.

Since nearest neighbors computation is not suitable to compute the pairs of neighboring points used to determine the points on the isosurfaces, we use spatial domain decomposition based on a three-dimensional *k*d-tree. To quickly access the cells of the *k*d-tree, we introduce a binary indexing scheme for tree traversal, see Section 4. To address one particular cell, we do not need to traverse the tree but can directly access it. The search for our neighbors in the *k*d-tree is reduced to bitwise operations on the indices representing the cells. In Section 5, we describe our method to obtain a small number of potential candidates for our neighbors. We introduce an angle criterion to select the appropriate neighbors among the candidates, as described in Section 6. The angle criterion can also be used to regulate the number of points we compute.

Once the appropriate neighborhood of a sample has been determined, we examine, which neighbors are separated from the sample by the isosurface. We linearly interpolate between the sample and each neighbor lying beyond the isosurface to compute points on the isosurface, see Section 7. This step leads to a point cloud, which we render in a final step using point-based rendering techniques, see Section 8.

To evaluate our approach, we apply it to synthetic and real data sets. For the synthetic data sets, we can compute how far the computed points are from an analytic representation of the isosurface. We even compare our methods to isosurface extraction methods for structured grids. We observe that the quality of the results we generate with our methods are close to the output of standard gridded isosurface extraction methods, although we directly compute them from the scattered data. In terms of computation time we compare our methods to 3D mesh generation algorithms like Delaunay tetrahedrization. We achieved a speed-up of about one order of magnitude for the data sets we have been using. A detailed analysis of our results is provided in Section 9.

2. Related Work

Isosurface extraction has a long tradition since the pioneer work by Lorensen and Cline [LC87] on marching cubes. Many extensions exist and marching isosurface extraction approaches have also been introduced for other grid structures like tetrahedral grids [GH95] and for adaptively refined grids [VKSM04]. For adaptive grids, a standard marching cubes-like approach may generate surfaces with gaps caused by hanging nodes. This observation led to the development of a family of dual contouring approaches extracting points on the surface in the cell's interior [JLSW02]. Still, the approaches impose constraints on the adaptive grid structure and do not allow for arbitrary configurations.

Recently, approaches have been introduced to deal with arbitrary configurations including scattered data. Co et al. have proposed approaches for isosurface extraction from multigrid data [CPJ04] and scattered data [CJ05]. They generate points on the isosurface, which are rendered using splatting techniques [CHJ03]. Other approaches for rendering isosurfaces in form of point clouds using splatting have been proposed by Livnat and Tricoche [LT04] and Rym-an-Lipinski et al. [vRLHJ*04]. Among these approaches only the method by Co et al. [CJ05] allows scattered volume data as an input. However, instead of generating the points on the isosurface directly from the scattered volume data, they generate local tetrahedral grids and compute the points from the local grid. Our approach, on the other hand, operates directly on scattered volume data and extracts isosurfaces without generating any global or local polyhedral grid. Our approach may also be applied to any gridded volume data or its adaptively refined counterpart without imposing constraints. However, this may only be worthwhile in case of nasty adaptive configurations, as our method would not benefit from any existing structural information.

Standard ways of dealing with scattered volume data would be to resample the data using scattered data interpolation techniques or to compute a polyhedral grid that connects the scattered data points. Scattered data interpolation is a well-studied field. We refer to one of many surveys on this topic for further details [FN91]. Recently, Park et al. [PLK*06, PLK*05] have shown that scattered data reconstruction for large data sets can be achieved at interactive or near-interactive rates when resampling over a regular grid. Unfortunately, such resampling steps always induce inaccuracies.

An alternative is to generate a tetrahedral grid from the scattered data [Nie93]. Delaunay tetrahedrizations are known to produce desirable results. Since the asymptotic complexity for the tetrahedrization is quadratic, the compu-

tation times may be very high for large data sets. We compare our approach to the generation of global and local Delaunay tetrahedrization. Our approach significantly outperforms the Delaunay algorithm.

Point-based surface representations have gained a lot of attention since the first works by Pfister et al. [PZvBG00], Rusinkiewicz and Levoy [RL00], and Linsen [Lin01] had been presented. Current point-based rendering techniques can be grouped into splatting approaches [PZvBG00, RL00] and point set surface approaches [ABCO*01, Lev03]. We make use of splatting approaches to render our isosurfaces.

3. General Approach

Let $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ be a trivariate scalar function, whose values are given for a large, finite set of samples $(\mathbf{x}_i, f(\mathbf{x}_i))$. The sample positions $\mathbf{x}_i \in \mathbb{R}^3$ are scattered, i. e. they are not arranged in a structured way, nor are any connectivity or neighborhood informations known for the sample locations. Our goal is to extract an isosurface $f(\mathbf{x}) = v_{iso}$ with respect to any real isovalue v_{iso} out of the range of function f.

Isosurface extraction over discrete structures is typically performed in two steps. First, a number of points $\mathbf{p}_k \in \mathbb{R}^3$ on the isosurface are computed, i. e. $f(\mathbf{p}_k) = v_{iso}$. For this purpose, an interpolation scheme is applied to the function values $f(\mathbf{x}_i)$ in order to locally reconstruct a continuous scalar field between the given discrete sample positions $\mathbf{x}_i$. We refer to the points $\mathbf{p}_k$ on the isosurface as *isopoints*.

In a second step, some kind of neighborhood information for the isopoints is generated, which is used to render the isosurface. When the samples are arranged in a structured grid, the neighborhood information can be retrieved from the structure of the grid. Typically, polygonal meshes are generated and rendered.

Our idea for isopoint computation from scattered volume data is based on linear interpolation between pairs of samples with close positions $\mathbf{x}_i$ and $\mathbf{x}_j$. The inspiration for this approach is given by isopoint computation using the marching tetrahedra algorithm after partitioning the domain via Delaunay tetrahedrization. The analogy is illustrated in Figure 1 for the 2D case. When considering the scattered sample positions in Figure 1, the Delaunay triangulation connects natural neighbors defined by the Voronoi diagram. For example, sample position $\mathbf{x}_i$ gets connected to its six surrounding neighbors (forming a 1-ring). The marching tetrahedra algorithm computes isopoints on the connecting edges that intersect the isosurface. If in Figure 1 the filled dots represent exactly those sample positions with function values greater than the isovalue, the contour may have the shape indicated by the dashed line. Three of the edges incident to $\mathbf{x}_i$ intersect by the contour. The isopoints on the edges are computed using linear interpolation along the edges.

We want to adopt this isopoint computation method for

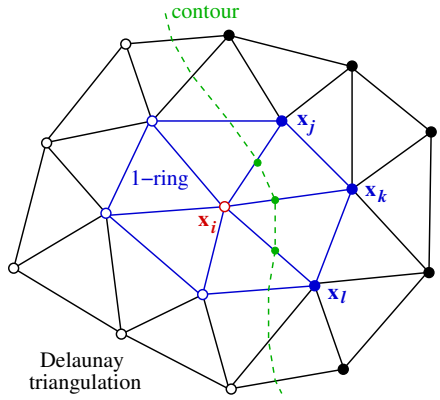

Figure 1: *Isopoint computation for scattered data via Delaunay triangulation and linear interpolation along Delaunay edges. For sample position $\mathbf{x}_i$ the incident edges to sample locations $\mathbf{x}_j$, $\mathbf{x}_k$, and $\mathbf{x}_l$ intersect the contour.*

our purposes while avoiding the computation of the Delaunay tetrahedrization. Thus, we need to estimate the natural neighbors for each sample position $\mathbf{x}_i$. Since the exact natural neighbors can only be determined via the expensive Voronoi diagram computation, we approximate its result using a spatial decomposition. Note that replacing the natural neighbors by the nearest neighbors would fail in case of varying density distributions of the samples. The *k*d-tree data structure [Ben75] is known to be a data structure with well-balanced trade-off between flexibility and efficiency. Operations on a *k*d-tree are considerably fast, yet it is robust against varying density distribution and clustering of sample positions. To perform a fast exploration of the *k*d-tree, we introduce an indexing scheme that, beside saving storage space, allows us to determine neighbors using bitwise operations on the index. We determine a small number of potential candidates for our neighbors and reject some of them using an angle criterion. When considering the 2D case in Figure 1, we ideally would reject all the candidates that do not belong to the 1-ring.

Exploiting this neighborhood information, we select those pairs of neighbors that are separated by the isosurface. We compute the isopoint by linearly interpolating between the two samples.

In a final step, we render the isosurface. The geometry of our isosurface is merely described by the positions of our isopoints. In the field of surface rendering, this surface representation is often referred to as a point cloud. We use point-based rendering techniques to render the isosurface.

4. Scattered Data Storage

We store the n scattered data points in a three-dimensional *k*d-tree. For each sample, we store its position and function

value in an array, such that we only store the respective indices in the *k*d-tree. We build the *k*d-tree recursively. The root represents the bounding box storing all samples. For each node of depth i, we sort the vector v storing the samples of the respective cell in $x^{i \bmod 3}$-direction, where x^0, x^1 and x^2 denote the three dimensions. We split v in two half-sized subvectors v_1 and v_2. If v is odd-sized, a link to the midpoint is stored in the current node of the tree in addition to the pivot value used for the split. We proceed recursively with the children nodes storing v_1 and v_2, respectively. The recursion stops when the subvector has length 1. Thus, each cell of the *k*d-tree contains exactly one sample.

The height of the tree is $\lceil \log_2(n+1) \rceil$. In worst-case $(n = 2^j)$, $j \in \mathbb{N}_+$, the number of nodes in the *k*d-tree is $2n-1$. The whole tree is stored in an array, where the root is at position 1 and the children of the node at position j are stored at positions $2j$ and $2j+1$. This order leads to an indexing scheme that is used to directly access the nodes. The indices can quickly be computed by using their binary representation and bit-wise operators.

In the following, all integers indexed with d, such as a_d or 100_d denote binary numbers. The operator $\oplus$ denotes the bit-wise Boolean exclusive-or operator. Finally, the operators $\ll$ and $\gg$ denote the bit-shift operators, which are recursively defined by

0. $a_\mathrm{d} \ll 0 = a_\mathrm{d}$ and $a_\mathrm{d} \gg 0 = a_\mathrm{d}$.
1. $a_\mathrm{d} \ll j = (a_\mathrm{d} \ll (j-1)) * 2$.
2. $a_\mathrm{d} \gg j = (a_\mathrm{d} \gg (j-1)) \operatorname{div} 2$.

In our *k*d-tree, The father of node with index b_d has index $b_\mathrm{d} \gg 1$ and its children have indices $b_\mathrm{d} \ll 1$ and $(b_\mathrm{d} \ll 1) \oplus 1_\mathrm{d}$, see Figure 2. Thus, we can navigate through the tree using fast binary operations. Moreover, qualitative propositions about the locations of cells can be made. For instance the cells with index 1111_d and 1000_d lie in diagonally distant corners of the *k*d-tree. Thus, many informations are implicitly saved in the indexing scheme. We exploit this property for fast neighbor search.

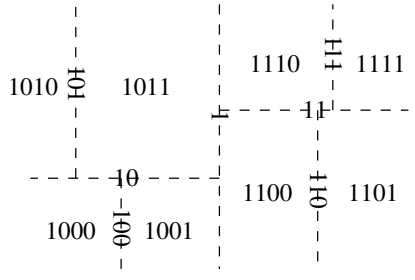

Figure 2: *Indexing scheme for two-dimensional kd-tree.*

5. Neighbors Search

For a point **x** lying in cell c, the neighborhood $N(c)$ contains all other cells or splitting planes that have at least one point in common with c. A 2D example of the neighborhood of a point **x** is shown in Figure 3. All bright areas and solid lines belong to the neighborhood.

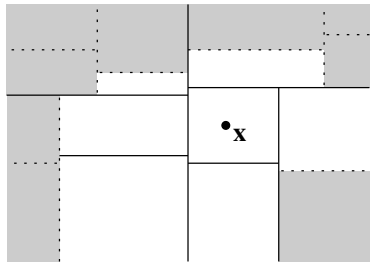

Figure 3: *Neighborhood of* **x**.

The way we store the nodes of the *k*d-tree in the vector constitutes that the position of every node of the tree is uniquely determined by the binary representation of its index. The inner nodes represent splitting planes, the leaf nodes represent cells. We find the neighborhood $N(c)$ of cell c in two steps.

In the first step, we compute the **direct neighbors**, which are the splitting planes and cells that result from the last three splitting steps during tree generation, see Figure 4. Let b_d be the index of cell c. The splitting planes $b_\mathrm{d} \gg 1$, $b_\mathrm{d} \gg 2$, and $b_\mathrm{d} \gg 3$ are direct neighbors, each containing one face of c. The adjacent cells to these faces are also direct neighbors leading to $\{b_\mathrm{d} \oplus 1_\mathrm{d}, b_\mathrm{d} \oplus 10_\mathrm{d}, b_\mathrm{d} \oplus 100_\mathrm{d}\} \subset N(c)$.

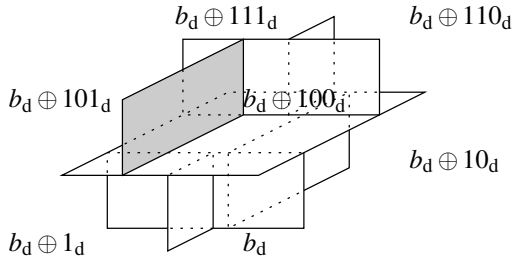

Figure 4: *Direct neighbors search for cell c with index* b_d.

Moreover, we have to check against the splitting planes beyond the already inserted planes whether these planes and the remaining four cells in Figure 4 belong to the neighborhood. For example, the grey plane in Figure 4 separating the cells with indices $b_\mathrm{d} \oplus 101_\mathrm{d}$ and $b_\mathrm{d} \oplus 100_\mathrm{d}$ corresponds to index $(b_\mathrm{d} \gg 1) \oplus 10_\mathrm{d}$. It has no common point with c and does not belong to $N(c)$. Therefore, $b_\mathrm{d} \oplus 101_\mathrm{d}$ does not belong to $N(c)$, either.

We need to check against four such splitting planes and the complete list of direct neighbors using the following criteria:

1. $(b_\mathrm{d} \gg 1) \oplus 1_\mathrm{d} \in N(c) \quad \Rightarrow \quad b_\mathrm{d} \oplus 11_\mathrm{d} \in N(c)$

2. $(b_\mathrm{d} \gg 1) \oplus 10_\mathrm{d} \in N(c) \quad \Rightarrow \quad b_\mathrm{d} \oplus 101_\mathrm{d} \in N(c)$
3. $(b_\mathrm{d} \gg 2) \oplus 1_\mathrm{d} \in N(c) \quad \Rightarrow \quad b_\mathrm{d} \oplus 110_\mathrm{d} \in N(c)$
4. $\{(b_\mathrm{d} \gg 2) \oplus 1_\mathrm{d}, (b_\mathrm{d} \gg 1) \oplus 10_\mathrm{d}\} \subset N(c)$
   $\Rightarrow \quad b_\mathrm{d} \oplus 111_\mathrm{d} \in N(c)$

The obtained neighborhood covers half of the faces of c and includes at least three and at most seven cells plus the respective splitting planes in between.

In the second step, we want to get neighbors for the three other faces of cell c, called **indirect neighbors**. In contrast to the direct neighbors, the maximum number of indirect neighbors is not constant. In the 2D case, the maximum number of indirect neighbors is $O(\sqrt{n})$, see Figure 5. In the 3D case, the maximum number of indirect neighbors is $O(\sqrt[3]{n^2})$. However, this worst case can at most occur for a small number of cells. The average case for a cell in three dimensions has up to nine neighbors for each face, leading to 27 indirect neighbors per cell.

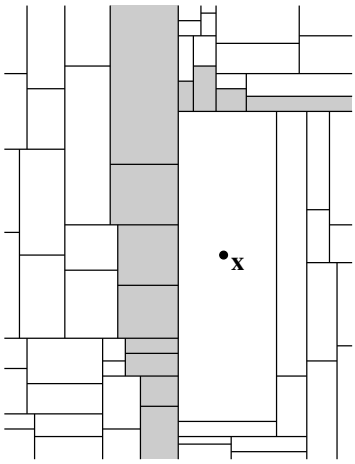

Figure 5: *Indirect neighbors of* **x**.

To compute the indirect neighbors, we first have to find the three splitting planes, that cover the remaining faces of cell c. To determine these cells, we go through the bits of the binary index b_d back to front considering every third bit. Thus, we investigate the location of cell c with respect to the splitting planes in one dimension at a time. We search for the first bit swap. If j is the bit, where the first bit swap occurs, the sought plane is $b_\mathrm{d} \gg j$.

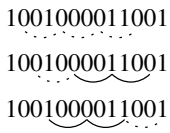

Figure 6: *Bit swap search for dimensions x^0 (top), x^1 (middle), and x^2 (bottom).*

For example, let $b_\mathrm{d} = 1001000011001_\mathrm{d}$, see Figure 6. When investigating the x^0-sequence, no bit swap occurs. Thus, c lies "to the left" of every division plane in this dimension, and c has no bounding plane to the left, i. e. no indirect neighbors in x^0-direction. The search in the x^1-sequence delivers a bit swap from 0 to 1 in the fifth bit from the back. Thus, the plane that bounds c in x^1-direction to the left is $b_\mathrm{d} \gg 5 = 10010000_\mathrm{d}$. Similarly, we get plane 100100_d in x^2-direction.

For each of the at most three found planes p, we have to search for all cells and planes out of $N(c)$ that lie on the opposite side of p with respect to c. The sought cells and planes all have a depth in the kd-tree higher than that of p. Moreover, the sought planes do not split in the same direction as p. Exploiting these properties we recursively search in the subtree starting from p. The algorithm is described by:

```
if (current plane splits in the same dimension as p)
   continue with subtree of current node next to c
else
   if (current plane has a common point with c)
      add current plane to N(c)
      continue with both subtrees of the current node
   else
      continue with subtree of current node next to c
```

In this recursion, planes are iteratively checked for being indirect neighbors of c. The recursion stops in leaf nodes that represent cells belonging and being added to $N(c)$.

In our example of Figure 6, we obtained $p_\mathrm{d} = 10010000_\mathrm{d}$ as a splitting plane for $b_\mathrm{d} = 1001000011001_\mathrm{d}$. The recursion starts with plane 100100000_d. Assuming that it is on the right of c, we proceed with 1001000000_d. If this plane has a common point with c, we have to insert it into $N(c)$ and proceed with 10010000000_d and 10010000001_d. Since we stepped down three times in the tree, these two planes are parallel to p_d. Thus, we proceed with 100100000001_d and 100100000011_d, respectively. The recursion stops in the next step by obtaining all indirect neighbor cells of c.

Only the test on which side of a plane cell c lies uses the coordinates of c when comparing it to the pivot value of the splitting plane. All other operations are performed on the binary representation of c using fast binary operations. Thus, our search for all neighbors has a performance that is similar to nearest-neighbor search in kd-trees.

6. Angle Criterion

Having computed a neighborhood for each sample point, we use the points in the neighborhood for further processing. However, not all points in the neighborhood are appropriate for isopoint computations. An undesired situation is shown in Figure 7. The sample in the lower right corner has two neighbors on the opposite side of the isosurface. Only the closer one should be considered for isopoint computation, as the interpolated isopoint with the farther neighbor would be farther from the surface than the closer neighbor.

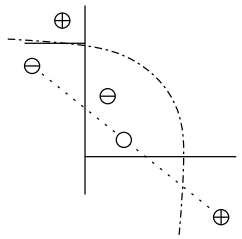

Figure 7: *Undesired neighborhood.*

We establish an angle criterion to avoid such situations. We extend the angle criterion method Linsen and Prautzsch [LP01, LP03] used for point-based surface representations to volume data. All points in the calculated neighborhood $N(c)$ of **x** are sorted according to their distance to **x**. We delete all those neighbors, for which the angle β between the edge from **x** to that neighbor and the edge from **x** to any neighbor closer to **x** is beneath a given threshold α, see Figure 8.

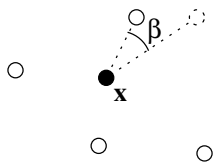

Figure 8: *Dashed Neighbor of* **x** *violates angle criterion* ($\beta < \alpha$).

The choice for angle α is motivated by Figure 9. Assuming gridded data, we not only want to include the six closest neighbors to **x**, but also the diagonal neighbors. Thus, we typically use $\alpha = \cos\left(\frac{1}{\sqrt{3}}\right)$. Such a choice is consistent with the angles obtained when considering the three-dimensional optimal sphere packing problem.

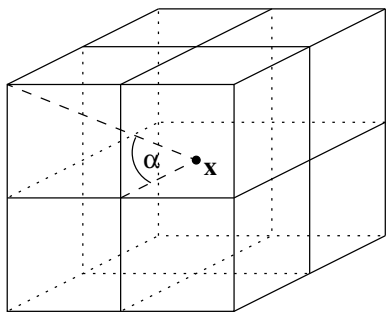

Figure 9: *Choice of minimal angle* α *between edges to the neighbors of* **x**.

7. Isopoint Computation

To compute the isopoints, we have to check whether there are points in the reduced set of $N(c)$ that are separated from **x** by the isosurface. If there are, isopoints are created by linear interpolation between **x** and the respective neighbors. Because of the angle criterion the neighborhood creation is not symmetric. It is possible that **x** is not neighbor of all points in $N(c)$. Thus, we cannot exploit symmetry and may compute some isopoints twice.

8. Point-based Isosurface Rendering

Since we do not generate any connectivity between our isopoints, we apply point-based rendering techniques to visualize our isosurface. We decided to use a standard splatting approach [PZvBG00]. We perform a *k*d-tree based nearest neighbor search and use the least-squares method to derive the splat's surface normals and radii. The orientation of the normal is already determined during isopoint computation.

9. Results and Discussion

We have applied our approach to a sphere data set, which consists of randomly distributed sample points in a $200 \times 200 \times 200$ cube. The sample values describe the distance to the center of a sphere. We extract an isosurface from the distance field using isovalue 70. The generated and rendered sphere can be seen in Figure 10. The computation times are shown in Table 1 and have been taken on a PC equipped with a 3.06GHz XEON processor. When changing the isovalue, neighborhoods do not need to be recomputed, which reduces the isopoint computation time to only five seconds when using 16 million samples.

# sample points	# isopoints	computation time
4M	45K	70 sec
8M	71K	151 sec
12M	100K	243 sec
16M	113K	318 sec

Table 1: *Isopoint computation times for sphere data set.*

In comparison, the tetrahedrization of the data set using four million samples with the algorithms provided by CGAL already lasted 3,266 seconds. Since Delaunay tetrahedrization has a quadratic asymptotic time complexity, the difference would be even more significant for larger data sets. Co et al. [CJ05] state in their paper that their approach using local tetrahedrization on a data set with only 346,000 samples lasted 92 seconds on a cluster with eleven PCs. They did not apply it to data sets of larger size.

The synthetic sphere data set was also chosen to analyze the error behavior of our algorithm. More precisely we were interested how far the generated isopoints lie from the analytic representation of the isosurface. We compared the maximum deviation of the isopoints generated by our algorithm

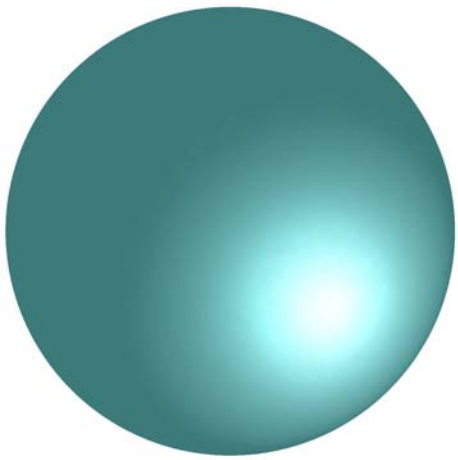

Figure 10: *Isosurface with point-based rendering and Gouraud shading, extracted out of 16M sample points.*

with eight million scattered sample points to the one generated by marching cubes on a regular $200 \times 200 \times 200$ grid. For marching cubes we got a deviation of 0.0018 from the sphere with radius 70. The deviations for our algorithm in dependence of the angle α used for the angle criterion are presented in Table 2.

angle α	# isopoints	deviation
15°	437,473	0.0425
35°	227,588	0.0355
55°	77,512	0.0320
80°	64,165	0.0182

Table 2: *Deviations for sphere data set.*

To show the power of our algorithm in generating smooth and well-shaped surfaces, we include a ray-traced picture of two spheres, each generated out of eight million sample points, in Figure 11. On the right-hand side one can see a yellow reflecting solid sphere, while on the left-hand side there is a reflecting lucent sphere made out of glass.

In Figure 12 and Figure 13, we show results of our approach applied to real data. The first isosurface, shown in Figure 12, is generated from a resampled regular data set of size $256 \times 256 \times 178$. We extracted 243,238 isopoints and rendered them using small splats to show the good extraction of the inner and outer border of the teapot. Also smooth transitions, especially on the spout and the knob, can be observed. In Figure 13, we show the isosurface of an engine. The regular data set of size $256 \times 256 \times 128$ is resampled to 16 million scattered data points. The shown isosurface contains 405,327 isopoints.

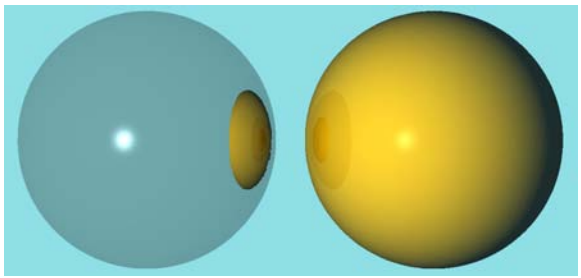

Figure 11: *Point-based ray tracing of two isosurfaces, generated from the sphere data set with 8M sample points. (Image courtesy of Karsten Müller.)*

Figure 12: *Isopoints of Boston Teapot with lobster, extracted out of 16M sample points. (Data set courtesy of Terarecon Inc, MERL and Brigham and Women's Hospital.)*

10. Conclusions and Future Work

We have presented an isosurface visualization algorithm that extracts the surface directly from scattered volume data. No global or local 3D mesh generation or 3D reconstruction over a regular grid is applied. Our approach consists of a geometry extraction and a rendering step. The geometry extraction step computes points on the isosurface by linearly interpolating between neighboring pairs of samples. The neighbor information is retrieved by partitioning the 3D domain into cells using a *k*d-tree. The cells are merely described by their index and bitwise index operations allow for a fast determination of potential neighbors. We use an angle criterion to select appropriate neighbors from the small set of candidates. The output of the geometry step is a point cloud representation of the isosurface. The final rendering step uses point-based rendering techniques to visualize the point cloud.

We evaluated our approach by applying it to synthetic and real data. The synthetic data allowed for an exact computation of the deviation from the analytic surface representation.

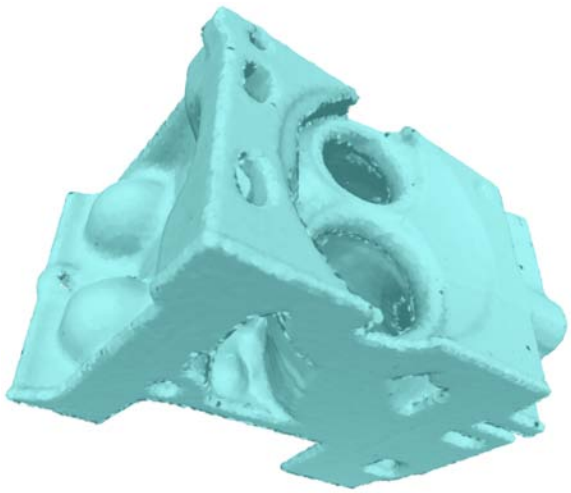

Figure 13: *Engine with point-based rendering out of 16M sample points. (Data set courtesy of General Electric.)*

We observed that our method got very close to what a gridded isosurfacing technique like marching cubes generated. The computation time of our algorithm was compared to 3D mesh generation algorithms like Delaunay tetrahedrization. We observed that we could achieve a speed-up of about one order of magnitude for the data sets we have been using.

In terms of future work, we would like to improve our rendering engine. Currently, we are only using spherical splats instead of elliptical ones and we cannot deal with sharp edges and corners yet.

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *VIS '01: Proceedings of the conference on Visualization '01* (Washington, DC, USA, 2001), IEEE Computer Society, pp. 21–28.

[Ben75] BENTLEY J. L.: Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (1975), 509–517.

[CHJ03] CO C. S., HAMANN B., JOY K. I.: Iso-splatting: A point-based alternative to isosurface visualization. In *Proceedings of the Eleventh Pacific Conference on Computer Graphics and Applications - Pacific Graphics 2003* (2003), Rokne J., Wang W.,, Klein R., (Eds.), pp. 325–334.

[CJ05] CO C. S., JOY K. I.: Isosurface Generation for Large-Scale Scattered Data Visualization. In *Proceedings of Vision, Modeling, and Visualization 2005* (Nov. 16–18 2005), Greiner G., Hornegger J., Niemann H.,, Stamminger M., (Eds.), Akademische Verlagsgesellschaft Aka GmbH, pp. 233–240.

[CPJ04] CO C. S., PORUMBESCU S. D., JOY K. I.: Meshless Isosurface Generation from Multiblock Data. In *Proceedings of VisSym 2004* (May 19–21 2004), Deussen O., Hansen C. D., Keim D. A.,, Saupe D., (Eds.), Eurographics.

[FN91] FRANKE R., NIELSON G. M.: *Geometric Modeling: Methods and Applications*. Springer Verlag, New York, 1991, ch. Scattered Data Interpolation: A Tutorial and Survey, pp. 131–160.

[GH95] GUÉZIEC A., HUMMEL R.: Exploiting triangulated surface extraction using tetrahedral decomposition. *IEEE Transaction on Visualization and Computer Graphics 1*, 4 (1995), 328–342.

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques - SIGGRAPH 2002* (2002), ACM Press, pp. 339–346.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH 1987* (1987), ACM Press, pp. 163–169.

[Lev03] LEVIN D.: Mesh-independent surface interpolation. In *Geometric Modeling for Scientific Visualization*, Brunnett G., Hamann B., Müller H.,, Linsen L., (Eds.). Springer-Verlag, 2003, pp. 37–49.

[Lin01] LINSEN L.: *Point cloud representation*. Tech. rep., Fakultät für Informatik, Universität Karlsruhe, 2001.

[LP01] LINSEN L., PRAUTZSCH H.: Global versus local triangulations. In *Proceedings of Eurographics 2001, Short Presentations* (2001), Roberts J., (Ed.).

[LP03] LINSEN L., PRAUTZSCH H.: Fan clouds - an alternative to meshes. In *Geometry, Morphology, and Computational Imaging, Lecture Notes in Computer Science (2616), Proceedings of 11th International Dagstuhl Workshop on Theoretical Foundations of Computer Vision* (2003), Asano T., Klette R.,, Ronse C., (Eds.), Springer-Verlag.

[LT04] LIVNAT Y., TRICOCHE X.: Interactive point-based isosurface extraction. In *IEEE Visualization* (2004), pp. 457–464.

[Nie93] NIELSON G. M.: Scattered data modeling. *IEEE Computer Graphics and Applications 1* (January 1993), 60–70.

[PLK*05] PARK S., LINSEN L., KREYLOS O., OWENS J. D., HAMANN B.: A framework for real-time volume visualization of streaming scattered data. In *Proceedings of Tenth International Fall Workshop on Vision, Modeling, and Visualization 2005* (2005), Stamminger M., Hornegger J., (Eds.), DFG Collaborative Research Center, pp. 225–232,507.

[PLK*06] PARK S. W., LINSEN L., KREYLOS O., OWENS J. D., HAMANN B.: Discrete Sibson interpolation. *IEEE Transactions on Visualization and Computer Graphics to appear* (2006).

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: surface elements as rendering primitives. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 335–342.

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: A multiresolution point rendering system for large meshes. In *Siggraph 2000, Computer Graphics Proceedings* (2000), Akeley K., (Ed.), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 343–352.

[VKSM04] VARADHAN G., KRISHNAN S., SRIRAM T., MANOCHA D.: Topology preserving surface extraction using adaptive subdivision. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (New York, NY, USA, 2004), ACM Press, pp. 235–244.

[vRLHJ*04] VON RYMON-LIPINSKI B., HANSSEN N., JANSEN T., RITTER L., KEEVE E.: Efficient point-based isosurface exploration using the span-triangle. In *IEEE Visualization* (2004), pp. 441–448.