Discrete Sibson Interpolation

Sung W. Park, Lars Linsen, Oliver Kreylos, John D. Owens, and Bernd Hamann

***Abstract*— Natural-neighbor interpolation methods, such as Sibson's method, are well-known schemes for multivariate data fitting and reconstruction. Despite its many desirable properties, Sibson's method is computationally expensive and difficult to implement, especially when applied to higher-dimensional data. The main reason for both problems is the method's implementation based on a Voronoi diagram of all data points. We describe a discrete approach to evaluating Sibson's interpolant on a regular grid, based solely on finding nearest neighbors and rendering and blending d-dimensional spheres. Our approach does not require us to construct an explicit Voronoi diagram, is easily implemented using commodity three-dimensional graphics hardware, leads to a significant speed increase compared to traditional approaches, and generalizes easily to higher dimensions. For large scattered data sets, we achieve two-dimensional (2D) interpolation at interactive rates and 3D interpolation (3D)with computation times of a few seconds.**

***Index Terms*— Scattered Data Interpolation, Natural-Neighbor Interpolation, Graphics Hardware.**

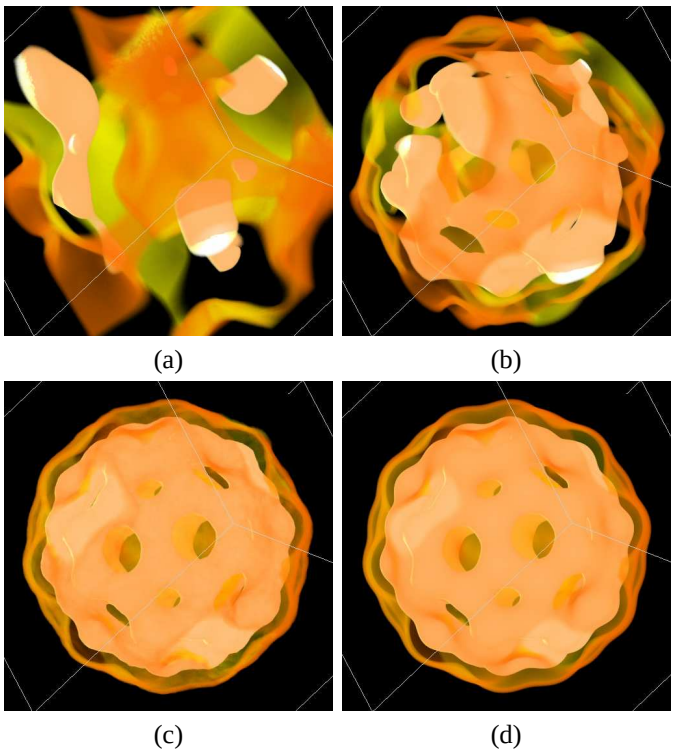


Fig. 1. Discrete Sibson interpolation and visualization of Bucky Ball data set (data size: 128^3) using (a) 100, (b) 1000, and (c) 10,000 scattered data points compared to visualization using (d) 128^3 grid points.

S. W. Park, L. Linsen, O. Kreylos, J. D. Owens, and B. Hamann are with the Institute for Data Analysis and Visualization (IDAV), Department of Computer Science, University of California, Davis, CA 95616, U.S.A.

L. Linsen is also with the Department of Mathematics and Computer Science, Ernst-Moritz-Arndt-Universität Greifswald, D-17487 Greifswald, Germany

{sunpark|okreylos|jowens|bhamann}@ucdavis.edu
linsen@uni-greifswald.de

I. Introduction

Scattered data visualization has been an area of interest for several decades. Recently, it has been experiencing a revival due to advances in distributed sensor networks, where small, independent sensors are used to collect data at random (scattered) locations in space and time. In order to apply standard visualization techniques such as contouring, slicing, and volume rendering to scattered data, it is necessary to define a reconstruction function that can be evaluated at arbitrary locations. A new challenge arises with the decreasing manufacturing costs of the sensors, which allows for many sensors to be deployed simultaneously. Highly efficient reconstruction methods are needed to process large streams of time-varying data originating from large sensor networks in real time.

A measured quantity, e. g., temperature or humidity, varying over time, can be described as a mathematical function $f(x,y,z,t)$. Each sensor i reports a stream of samples $f_i(t)$ of that function, taken at the sensor's position (x_i,y_i,z_i)[1]. In order to analyze or visualize a measured data set, one has to reconstruct the function f from the samples reported by the sensor network. It is desirable to have visualization tools with the capability to reconstruct and visualize two- and three-dimensional scattered data sets with many data points in real time.

Most scientific visualization techniques require data to include connectivity information. Scattered data in "raw format" does not provide such information. A common approach to deal with unconnected data is the use of field reconstruction methods producing an analytical definition that is later resampled to a grid format supported by standard visualization methods such as contouring or volume rendering. Existing scattered data techniques such as radial basis function methods, Shepard's method and its variants, Hardy's method, and triangulation-based methods are based on using all samples or selected local subsets of samples. Many of these techniques are computationally expensive and do not scale well with the number of data points; they are not geared toward real-time visualization.

Sibson's natural-neighbor interpolation method is a scattered data interpolation scheme based on a Voronoi (Dirichlet, Thiessen) diagram of a data set's sample locations. We present the mathematical background of, and definitions regarding, Voronoi diagrams and Sibson's method in Section III. Sibson's method is known to produce good interpolation results and has desirable properties such as linear precision, locality, and C^1-continuity [1]. Similar to many other schemes, however, Sibson's method is computationally expensive and difficult to

[1]In the case of moving sensors, the values x_i, y_i, and z_i are themselves functions of time.

compute especially when it is extended to higher dimensions. The high computational cost is due to the fact that each evaluation of the interpolant requires the insertion of a site into the sample location's Voronoi diagram to calculate the proper interpolation weights of neighboring samples [2]. The implementation difficulty is due to the difficulty of explicitly representing higher-dimensional Voronoi diagrams, especially when facing limited numerical precision and samples in degenerate positions.

We present a discretized approximation to Sibson's interpolation scheme. Instead of computing the interpolation function using its traditional geometric definition, we use a more efficient scheme operating on a discrete domain. One straightforward way of discretizing Sibson's method is to use a discrete Voronoi diagram, which can be computed efficiently using three-dimensional graphics hardware [3]. However, the algorithm described by Hoff et al. [3] does not scale well to large data sets. We decided to use an alternative approach exploiting geometrical properties of Sibson's method, which reduce the interpolation algorithm to rendering and blending d-dimensional spheres whose radii are determined by the distance between an evaluation location and its nearest neighbor in the data set. This approach no longer requires us to use an explicit Voronoi diagram; we use a kd-tree to efficiently find nearest neighbors instead. Our approach, described in detail in Section IV, generalizes to higher dimensions.

Moreover, we show that our algorithm maps well to modern programmable graphics hardware. In the past few years, there has been an explosion in both the performance and programmability of desktop graphics hardware. Today's graphics processors offer the capability of running user-specified programs on each vertex and fragment. These capabilities are not only used in producing more complex and detailed shaders for graphics applications [4], but also for tasks diverging from traditional polygon-based graphics, such as visual simulation [5], numerical computation [6], or more general-purpose computing [7]. The rapid performance growth of these graphics processors coupled with their ubiquity in modern computers make them particularly interesting targets for visualization tasks such as the one described in this paper.

With full optimization, we show that in the two-dimensional case, function/image reconstruction can be done at interactive frame rates, e. g., we can reconstruct an image at an output resolution of 512×512 pixels from $50,000$ scattered data points in 0.125 seconds. In the three-dimensional case, the speed-up is also quite significant, e. g., we can reconstruct a volume field at an output resolution of $128 \times 128 \times 128$ voxels from $100,000$ scattered data points in a few seconds. Implementation details and results are in Section V. Overall, our algorithm leads to a significant increase in speed for large data sets, reduced storage space, and simplicity in implementation compared to previous methods.

II. Related Work

Scattered data interpolation has been a research topic for several decades and a large number of scattered data interpolation techniques are well-understood today. Typically, scattered data methods are generalized into methods based on inverse distances, radial basis functions, and natural-neighbors. Many good surveys discuss and compare extensively on different scattered data methods [1], [2], [8]–[13].

Inverse distance weighted methods are based on the assumption that the interpolated value should be influenced more by nearby points and less by more distant points. The original work by Shepard [14] used a global scheme, which tends to flatten the recontruction and whose computational costs make it non-practical for applications to large data sets. Many local and modified schemes have been introduced to address these issues. Using inverse ditance weighted methods, each point and has a radially symmetric influence. This isotropism can lead to undesired field reconstruction artifacts when sampling rates differ significantly in different directions.

Radial basis function methods is regarded as being among the most elegant schemes from the mathematically point of view [8]. Functions like Hardy's multiquadrics, inverse multiquadrics, and thin-plate spline (TPS) interpolants have been applied to a large variety of scattered data problems. They allow for anisotropic support, but require solving a system of N linear equations with N unknowns. Thus, many methods cannot effectively handle large data sets due to instability and computational costs associated with the computation of radial interpolation that grows with increasing distance from scattered data sites. Local versions of these methods have been proposed to deal with larger data sets.

Natural-neighbor interpolation, introduced by Sibson [15], is a popular method used in many fields such as environmental and geotechnical engineering and solid mechanics [16], [17]. Natural-neighbor interpolation is constructed on the basis of an underlying Voronoi diagram of a data set's sample locations. For natural-neighbor interpolation, a sample's weight is not dictated by the same length measure in all dimensions, but by the appropriate Lebesgue measure of the space directions [17]. This allows for anisotropic support. Among other properties, natural-neighbor interpolation methods are local, require local neighbors only, has linear precision, and is C^1 continuous everywhere except at the data sites. Natural-neighbor interpolation is superior to distance-based weights due to its density variation in the area-based weights. Unlike the radial basis function methods, it doesn't require solving a linear system of equations. Its ability to handle highly irregular distribution of nodes. However, natural-neighbor interpolation in its traditional implementation is computationally more intense than other approaches (especially in higher dimensions) due the Voronoi diagram computation and Voronoi insertion per interpolant.

Implementation of scattered data algorithms has been traditionally done in the CPU. The Advances of GPUs in recent years, however, have made it an attractive target for

computationally complex and demanding scientific and visualization applications. Hoff et al. [3] have computed a discrete Voronoi diagram taking advantage of the graphics hardware. Since then, GPUs have been used in other computational geometry problems, for example, for accelerating distance field computations [18]. GPUs have been applied to scattered data interpolation methods as well. Jang et al. [19] describe a method of accelerating radial basis functions on GPUs. Applying-GPU assisted natural-neighbor interpolation has also been described by Fan et al for 2D data set. [20]. Their approach utilizes GPU to compute areas of contribution and require a Delaunay triangulation of a given data set.

III. Definitions and Background

A. Voronoi Diagram

A standard Voronoi diagram is a partitioning of a given domain into *cells* (*tiles*) based on a finite set of given scattered data points, called *sites*. Each cell contains of the points closest to a particular site. More specifically, consider a d-dimensional, convex domain $\Omega \subset \mathbf{R}^d$ and a set $N = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m\}$ of m scattered data points. The Voronoi diagram $\mathscr{V}(N)$ of the set N over domain Ω is a domain partitioning into regions $V(\mathbf{p}_i) \subset \Omega$, such that any point in $V(\mathbf{p}_i)$ is closer to site $\mathbf{p}_i$ than to any other site $\mathbf{p}_j \in N$ $(j \neq i)$. The region $V(\mathbf{p}_i)$ associated with site $\mathbf{p}_i$ is called a *Voronoi cell* and is defined as

$$V(\mathbf{p}_i) = \{\mathbf{p} \in \Omega : d(\mathbf{p}, \mathbf{p}_i) < d(\mathbf{p}, \mathbf{p}_j) \ \forall j \neq i\},$$

where d is a distance metric, usually the Euclidean one. The Voronoi cells are convex polygons/polyhedra in $\mathbf{R}^d$. Figure 2(a) shows the Voronoi diagram for an example with four sites $\mathbf{p}_1, \ldots, \mathbf{p}_4$, where the colors of the four Voronoi cells represent the function values at the sites.

A number of algorithms exist today for computing Voronoi diagrams of points in two-dimensions, three-dimensions, and higher-dimensional spaces. Efficient adaptive precision and exact arithmetics are active areas of research. There are also many approximate versions of Voronoi diagrams, and computations based on space subdivision.

B. Sibson's Interpolant

Sibson [15] introduced a natural-neighbor interpolation scheme that uses a weighted average of surrounding or neighboring data sites to compute the interpolated function. The fundamental difference to Shepard's method is the assignment of the weights using natural-neighbors.

Figure 2 illustrates how Sibson's interpolant is computed using an example with four sites. Given a set of sites, the Voronoi diagram of the set of sites is computed first, as shown in Figure 2(a). To interpolate the function value at a point $\mathbf{p}$, $\mathbf{p}$ is inserted into the Voronoi diagram, as shown in Figure 2(b). Its Voronoi cell $V(\mathbf{p})$ has k neighboring cells $V(\mathbf{p}_1), \ldots, V(\mathbf{p}_k)$. The k sites $\mathbf{p}_1, \ldots, \mathbf{p}_k$ are called the *natural-neighbors* of $\mathbf{p}$. The area/volume of $V(\mathbf{p})$ is the union of areas/volumes u_i that belonged to the Voronoi cells $V(\mathbf{p}_i)$ of the neighbors in the original Voronoi diagram, as illustrated in Figure 2(c). Sibson's interpolant of a function f evaluated at $\mathbf{p}$ is defined as

$$f(p) = \frac{\sum_{i=1}^{k} u_i f(p_i)}{\sum_{i=1}^{k} u_i} = \sum_{i=1}^{k} u_i' f(p_i) \ ,$$

where

$$u_i' = \frac{u_i}{\sum_{i=1}^{k} u_i} \ .$$

The reconstructed function f for the given example is shown in Figure 2(d). Sibson's interpolation is a local scheme, as only the values at the natural-neighbors of $\mathbf{p}$ influence the interpolated value $f(\mathbf{p})$.

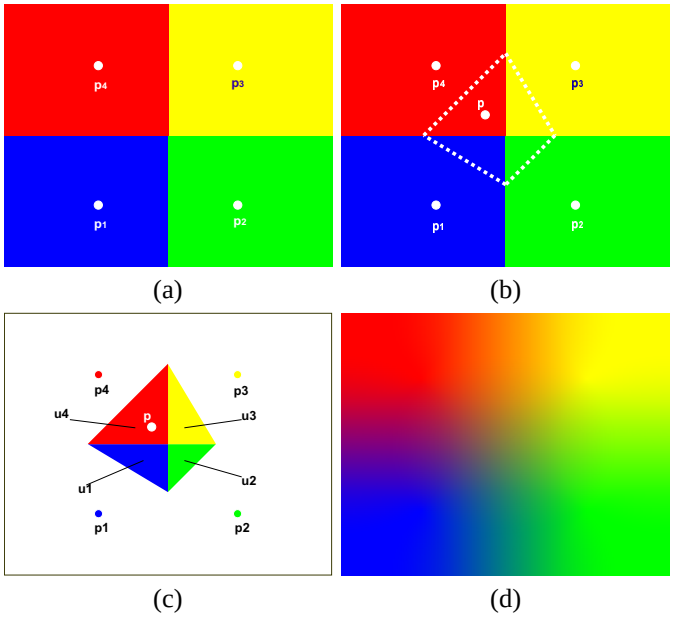


Fig. 2. Sibson's interpolation starts out with the Voronoi diagram (a) and tentatively inserts the new site $\mathbf{p}$ (b) to compute areas u_i (c), which are used as weights for the interpolation (d).

Typically, Sibson's interpolation is implemented geometrically, i. e., one computes the weights for the interpolant by determining the areas/volumes u_i associated with $\mathbf{p}_i$ upon insertion of $\mathbf{p}$. Another method, proposed by Yee [21], first computes the Voronoi region $V(\mathbf{p})$ and then subdivides the region into the natural-neighbors' sub-regions. The dual structure of the Voronoi diagram, known as Delaunay triangulation, also has been used to calculate Sibson's interpolant. The Delaunay triangulation can be used directly to calculate the weights of the neighboring regions for interpolation [22]. Previous approaches to Sibson's interpolation are computationally expensive and difficult to implement. They do not fit the needs of real-time visualization.

IV. Discrete Sibson Interpolation

We describe how Sibson's interpolation can be performed in a discrete fashion. We start with the description of a discrete Voronoi diagram computation, which leads to a straightforward approach to discrete Sibson interpolation. We improve

this approach by exploiting the geometrical properties of Sibson's interpolant, which makes the computation much more efficient, especially when dealing with many sites, and makes unnecessary the requirement of computing an explicit Voronoi diagram.

A. A Discrete Voronoi Diagram

Hoff et al. [3] presented a graph-based approach for the computation of generalized Voronoi diagrams both in two-dimensional and three-dimensional space. Their approach is based on the observations that geometric shapes representing distance fields can be rendered to generate discrete Voronoi diagrams. For the two-dimensional case, cones are rendered as distance functions for each site. By drawing the sites' distance fields and utilizing graphics hardware's depth buffer function to only render parts the geometry being closest to each site, Hoff et al. showed how Voronoi diagrams can be constructed. Their approach even generalizes to scenes that contain data primitives more general than points as sites. However, the algorithm does not scale in the number of sites and dimensions. The bottleneck is a result of each site having to render a distance field that covers the entire domain, i. e., a distance cone that covers the entire Voronoi image in two-dimensional space and a distance hyperboloid sheet that covers the entire Voronoi volume in three-dimensional space. For a Voronoi diagram with a large number of sites, this approach is inefficient.

Although Hoff's algorithm works well for constructing generalized discrete Voronoi diagrams, for our purposes we only need a discrete Voronoi diagram of point sites that scales well to many sites and to higher dimensions. Instead of rendering distance fields, we present a more efficient approach for generating Voronoi diagrams that scale well in both number of sites and in dimensions by using a spatial structure, i.e., a *k*d-tree. In a discrete Voronoi diagram, at every position $\mathbf{p}$, we store information about the nearest site $\mathbf{p}_i$ and the distance $d(\mathbf{p},\mathbf{p}_i)$ to that nearest site. Depending on the desired precision/resolution of the discrete Voronoi diagram, a grid size is determined. We can use a *k*d-tree to query the desired information at each grid location. Initially, the *k*d-tree is used to store all sites. The *k*d-tree is queried at each grid location $\mathbf{p}$ to determine the site that is closest to the grid location. This query returns the associated Voronoi region. Then, the nearest site's value and the distance to this site can be stored in a grid, which represents the discrete Voronoi diagram, after the *k*d-tree has been queried for all grid locations.

This approach is efficient for constructing Voronoi diagrams with large number of sites, since a *k*d-tree can be created in $O(n\log n)$ time. Performing the nearest site search is done in $O(\log n)$ time. In contrast to the discrete Voronoi approach in [3], our method scales well in the number of sites: Constructing the discrete Voronoi diagram solely depends on the computational costs of inserting and querying the *k*d-tree, as opposed to having to render geometric shapes that cover the entire grid. Due to the nature of a *k*d-tree, our approach also scales well in dimension.

B. Discretizing Sibson's Interpolation

In the discretized domain, Sibson's interpolation can be computed in a straightforward manner. Since calculating Sibson's interpolant at point $\mathbf{p}$ is equivalent to taking the weighted average of values at the neighboring sites (see Figure 2), this value can be computed by averaging the discrete elements. Accumulating all data values from the original Voronoi diagram within the Voronoi region $V(\mathbf{p})$, and dividing the accumulated value by the number of elements accumulated, defines the average of the region. This value is the discretized equivalent to Sibson's interpolation value. Since discrete elements are accumulated at one location, we call this interpolation scheme the "gather approach."

Given a discrete d-dimensional Voronoi diagram for a set of m sites $\mathbf{p}_1,\ldots,\mathbf{p}_m \in \Omega \subset \mathbf{R}^d$ over a rasterized domain Ω, one can access the following information for every raster position $\mathbf{i} \in \Omega$: the distance $d_v = V_d(\mathbf{i})$ to the site $\mathbf{p}_j$ closest to raster position $\mathbf{i}$, and the data value $c_v = V_c(\mathbf{i})$ at the site $\mathbf{p}_j$. The d-dimensional fields V_d and V_c representing the Voronoi diagram are precomputed as described in Section IV-A. To reconstruct the function f over the entire domain Ω using a discrete Sibson interpolant, we scan the rasterized domain and compute an interpolated value $f(\mathbf{p})$ for each raster position $\mathbf{p}$ of the desired output region. To compute $f(\mathbf{p})$, we accumulate all the values within the region $V(\mathbf{p})$ that we would obtain by inserting $\mathbf{p}$ as a new site into the Voronoi diagram. For the accumulation process, we initialize $c(\mathbf{p})$ to zero and scan all raster positions $\mathbf{i}$ of the discrete Voronoi diagram. For each raster position $\mathbf{i}$, we determine whether it lies in $V(\mathbf{p})$ by comparing the distance $V_d(\mathbf{i})$ of raster position $\mathbf{i}$ to its closest site with the distance from raster position $\mathbf{i}$ to raster position $\mathbf{p}$. If $\mathbf{i} \in V(\mathbf{p})$, we add $V_c(\mathbf{i})$ to $c(\mathbf{p})$. We then count the number $n(\mathbf{p})$ of raster positions $\mathbf{i} \in V(\mathbf{p})$ and assign $f(\mathbf{p}) = c(\mathbf{p})/n(\mathbf{p})$.

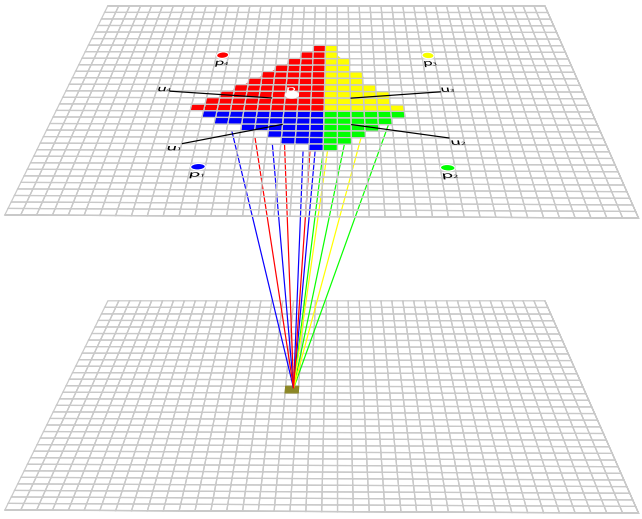


Fig. 3. The gather approach: Values in Voronoi cell $V(\mathbf{p})$ are gathered for Sibson's interpolant $f(\mathbf{p})$.

This approach can be implemented efficiently, without explicitly inserting new sites into the existing Voronoi diagram. Assuming that a discrete Voronoi diagram has been computed, we perform a two-pass computation. In the first pass, we accumulate all values contributing to each $c(\mathbf{p})$. In the second pass, the accumulated values are divided by the number of accumulations $n(\mathbf{p})$ to obtain the interpolated value $f(\mathbf{p})$. The gather approach can be summarized in the following way:

- Compute the discrete Voronoi diagram for m sites.
- For every output raster position $\mathbf{p}$ do
  - Initialize $c(\mathbf{p}) = 0$ and $n(\mathbf{p}) = 0$.
  - For every Voronoi raster position $\mathbf{i}$ do
    If $\mathbf{i} \in V(\mathbf{p})$, add $V_c(\mathbf{i})$ to $c(\mathbf{p})$ and increment $n(\mathbf{p})$ by 1.
  - Set $f(\mathbf{p}) = c(\mathbf{p})/n(\mathbf{p})$.

C. Efficient Discrete Sibson Interpolation

The key idea to making Sibson's interpolant more efficient is to consider the problem in the "opposite" direction, where values are "scattered" rather than gathered. Instead of iterating over all positions $\mathbf{p}$ and gathering the values at all raster positions $\mathbf{i} \in V(\mathbf{p})$ into $c(\mathbf{p})$, one can iterate over the raster positions $\mathbf{i}$ and determine which $f(\mathbf{p})$ are influenced by the value at $\mathbf{i}$. This approach has two major advantages: It allows us to utilize the efficient rasterization units in modern graphics hardware, and it limits computations to only those values $\mathbf{i}$ that affect the final result.

In more detail, the scatter approach works as follows: Consider one raster position $\mathbf{i}$ in the Voronoi diagram. We want to determine to which positions $\mathbf{p}$ the data value $V_c(\mathbf{i})$ contributes. This is easily achieved by rendering a d-dimensional block over all positions $\mathbf{p}$, and determining, for each $\mathbf{p}$, whether the distance between $\mathbf{i}$ and $\mathbf{p}$ is less than the distance $V_d(\mathbf{i})$ between $\mathbf{i}$ and its closest site. If it is less, then $\mathbf{i} \in V(\mathbf{p})$, and we accumulate the value $V_c(\mathbf{i})$ into $f(\mathbf{p})$.

This approach can be summarized in the following way:

- Compute the discrete Voronoi diagram for m sites.
- For every output raster position $\mathbf{p}$ do
  - Initialize $c(\mathbf{p}) = 0$ and $n(\mathbf{p}) = 0$.
- For every Voronoi raster position $\mathbf{i}$ do
  - For every output raster position $\mathbf{p}$ do
    If $\mathbf{i} \in V(\mathbf{p})$, add $V_c(\mathbf{i})$ to $c(\mathbf{p})$ and increment $n(\mathbf{p})$ by 1.
- For every output raster position $\mathbf{p}$ do
  - Set $f(\mathbf{p}) = c(\mathbf{p})/n(\mathbf{p})$.

Moreover, we observe that in both the scatter and the gather approach, particularly with dense samples, most $V_c(\mathbf{i})$ do not contribute to any given $f(\mathbf{p})$. For the gather approach, it would require us to determine a bounding box for $V(\mathbf{p})$, for each $\mathbf{p}$, to optimize the procedure, but the computation of the bounding boxes would be expensive. For the scatter approach, we can take advantage of this observation. More specifically, we can determine exactly the area to which a value at raster position $\mathbf{i}$ contributes:

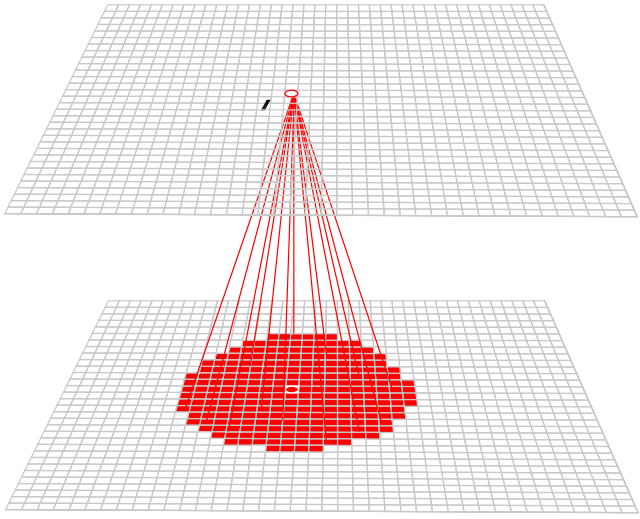


Fig. 4. The scatter approach: Value at a location $\mathbf{i}$ contributes to Sibson's interpolant within certain region and thus is scattered to all positions in that region. The region is defined by all positions $\mathbf{p}$ with $\mathbf{i} \in V(\mathbf{p})$.

Theorem: *In a discrete version of Sibson's interpolant, the value $V_c(\mathbf{i})$ at a raster position $\mathbf{i}$ influences exactly all those output locations $\mathbf{p}$ that are within a d-dimensional sphere around $\mathbf{i}$, whose radius is the distance from $\mathbf{i}$ to the closest site.*

Proof: Let $\mathbf{p}_1, \ldots, \mathbf{p}_m$ be the sites to be interpolated, and $\mathbf{p}$ an output location. In the discrete Sibson approach, we accumulate and blend all the values that lie within the Voronoi cell $V(\mathbf{p})$ to compute the interpolation result $f(\mathbf{p})$ at output location $\mathbf{p}$ (see Figures 2 and 3). We have to show that, for all raster positions $\mathbf{i} \in V(\mathbf{p})$, a d-dimensional sphere around $\mathbf{i}$ with radius $r = \min_{i=1}^{m} d(\mathbf{i}, \mathbf{p}_i)$ includes output location $\mathbf{p}$, and that such a sphere does not include $\mathbf{p}$, for all $\mathbf{i} \notin V(\mathbf{p})$. Let $\mathbf{i} \in V(\mathbf{p})$. Then, by $V(\mathbf{p})$, $d(\mathbf{i}, \mathbf{p}) < \min_{i=1}^{m} d(\mathbf{i}, \mathbf{p}_i) = r$, a d-dimensional sphere around $\mathbf{i}$ with radius r does include $\mathbf{p}$ (see Figure 5(a)). Now, let $\mathbf{i} \notin V(\mathbf{p})$. Then, $d(\mathbf{i}, \mathbf{p}) \geq \min_{i=1}^{m} d(\mathbf{i}, \mathbf{p}_i) = r$, and a d-dimensional sphere around $\mathbf{i}$ with radius r does not include $\mathbf{p}$ (see Figure 5(b)). □

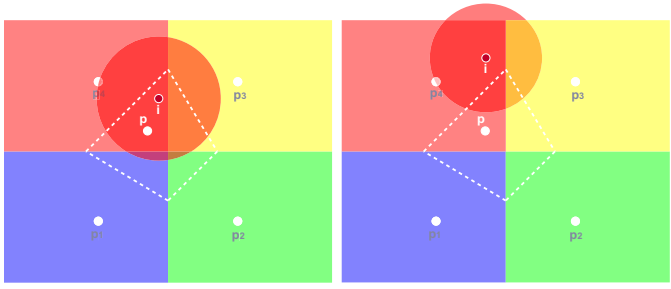


Fig. 5. The value at Voronoi raster position $\mathbf{i}$ influences exactly all those output locations $\mathbf{p}$ that are within a circle around $\mathbf{i}$, whose radius is the distance from $\mathbf{i}$ to the closest site $\mathbf{p}_4$.

The above theorem implies that in order to scatter the value

$V_c(\mathbf{i})$ at point $\mathbf{i}$ to all output raster positions $\mathbf{p}$ that are influenced by it, we merely have to draw a d-dimensional sphere of value $V_c(\mathbf{i})$ around $\mathbf{i}$, where the radius of the sphere is the distance from $\mathbf{i}$ to its closest site. Blending these spheres results in a discrete equivalent of Sibson's interpolant, as shown in Figure 6. This observation also allows us to eliminate discrete Voronoi diagrams altogether. The only use for a Voronoi diagram in the described algorithm is to find the closest site of a point $\mathbf{i}$, and this can easily and efficiently be achieved by querying a kd-tree containing all sites directly.

The scatter approach thus simplifies to the following algorithm:

- Construct a kd-tree for m sites.
- For every output raster position $\mathbf{p}$ do
  - Initialize $c(\mathbf{p}) = 0$ and $n(\mathbf{p}) = 0$.
- For every raster position $\mathbf{i}$ do
  - Find the closest site $\mathbf{p}_n$ in the kd-tree.
  - Calculate $r = d(\mathbf{i}, \mathbf{p}_n)$.
  - For each raster position $\mathbf{p}$ inside a d-dimensional sphere of radius r around $\mathbf{i}$ do

    Add $V_c(\mathbf{i})$ to $c(\mathbf{p})$ and increment $n(\mathbf{p})$ by one.
- For every output raster position $\mathbf{p}$ do
  - Set $f(\mathbf{p}) = c(\mathbf{p})/n(\mathbf{p})$.

By rendering just these small regions, the implementation becomes considerably more efficient, especially when dealing with very large number of samples." Figure **??** shows how the scatter approach is implemented in a fragment program.

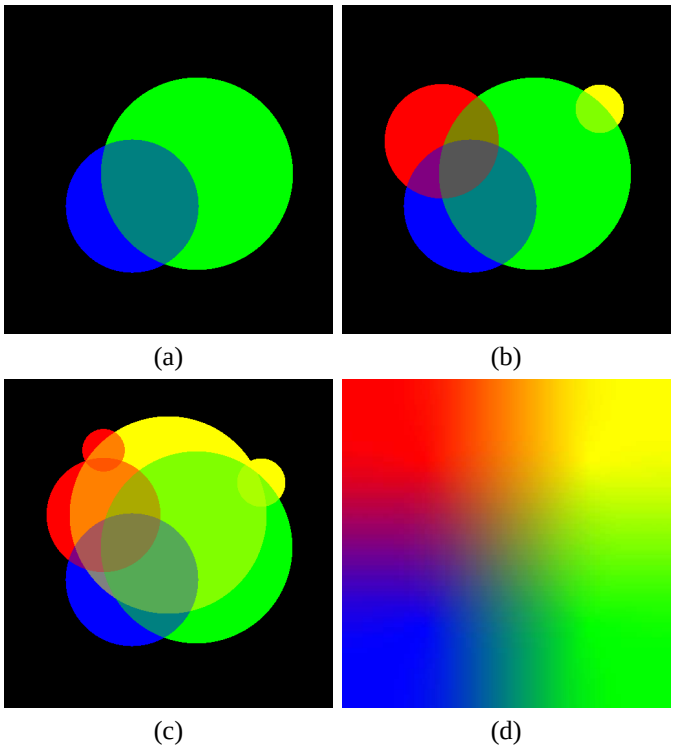


Fig. 6. The gather approach iteratively draws disks of certain radii for every raster position and blends the values of the disks. The approach is illustrated for two, four, six, and 512^2 disks.

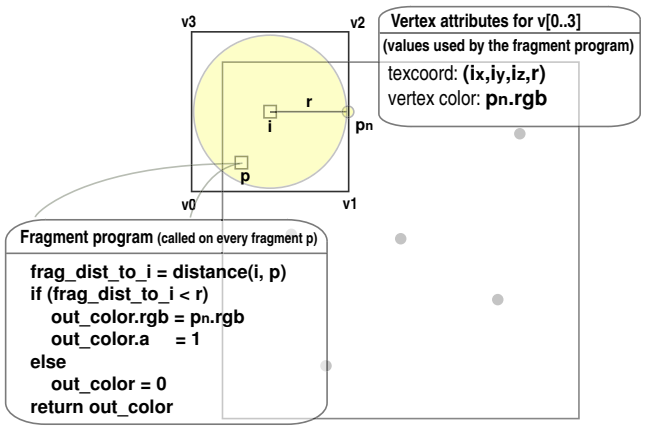


Fig. 7. Scatter approach implementation: Value at each Voronoi raster position $\mathbf{i}$ is scattered by rendering a quad of twice the length of the distance r to the closest site $\mathbf{p}_n$. Illustrated are the fragment program scattering values at one raster position. For efficient processing, raster location $\mathbf{i}$ and value at site $\mathbf{p}_n$ are stored using texture coordinates and color attributes of each vertex.

D. Implementation

To take full advantage of the algorithm described, we utilize the fast raster processing and programmability capabilities of modern graphics hardware. For implementation purposes, the following processing capabilities are desirable: floating-point precision framebuffer and textures and floating-point blending. Floating-point precision is desired for accuracy in data accumulation because using only eight-bit precision, i. e., 256 values, does not suffice to accurately accumulate data values during interpolation.

In our implementation, we use OpenGL libraries and NVIDIA's Cg language for vertex and fragment programming with an NVIDIA GeForce 6800GT card. For storing the discrete Voronoi diagram and Sibson's interpolation function, we use a four channel 32-bit floating-point textures. The RGB channels are used for storing the nearest site's value, and the A channel is used for storing the distance to the nearest site. Voronoi diagram generation using a kd-tree is implemented on the CPU by querying the kd-tree at each discrete location for the Voronoi region it is in. For computing Sibson's interpolant, the RGB channels are used to accumulate data values, and the A channel is used for counting the number of values accumulated in each pixel.

An efficient implementation would use hardware's floating-point blending to accumulate intermediate values. Floating-point blending is available in the current generation of cards, however, the precision it supports is only 16 bits which for small data sets, especially for 3D applications is not enough for proper interpolation. Using 16-bit blending can produce incorrect visual results due to data overflow. To approximate the performance on graphics hardware we measure our time using 16-bit floating point blending. Although an interpolation using the 16-bit implementation requires the same amount of computations, it requires less memory transfer. For our time measurements, we assumed that using the 16-bit implementa-

tion is a reasonably good approximation as we expect the next generation of cards to only get faster.

The gather approach accumulates data values at each raster position. This can be implemented in a fragment program, but certain inefficiencies must be considered. First, in order to accumulate values at a single raster position, points must be rendered instead of triangles or quadrilaterals: Every fragment that is rendered must output to a single location, and this is only possible by rendering points. Also, since the exact region that contributes to the particular raster position is not known, regions that do not contribute still must be examined. Although a bounding region could be calculated by computing the convex hull, it would require additional computations. In either case, the additional computations make the gather approach less efficient.

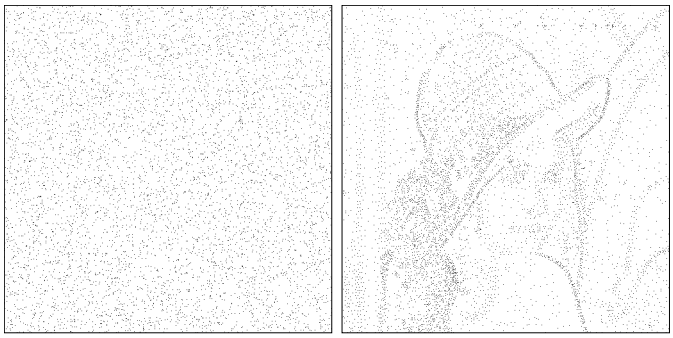
Implementing the fully optimized scatter approach is simple and very efficient. In two dimensions, we first construct a *k*d-tree of all sites. Sibson's interpolant is then computed by drawing circles at each given raster position, while querying the *k*d-tree for data values and radii. The data value is associated with function values of the circle and the radius information is used to render a square, centered at the raster position, with side length being twice the radius. The circles within each square are drawn by comparing distances in the fragment program. Once all values are accumulated, we use a fragment program to divide the total accumulated value by the number of accumulated values to generate the final image.

For Sibson interpolation in three dimensions, spheres are deposited at each raster position of the volume. This is achieved by computing one slice at a time. For each slice, cross section of the intersecting spheres are accumulated and later normalized. Like in the two dimensional case, a *k*d-tree is queried for data values and radii.

V. Results

Although our optimal Sibson's interpolation algorithm no longer requires knowledge of an explicit Voronoi diagram, we still have compared our new approach for generating discrete Voronoi diagrams with Hoff et al.'s method of generating discrete Voronoi diagrams. We have compared the computation time and the interpolation results for two-dimensional Sibson's interpolant using the traditional geometric approach and our optimized hardware-accelerated version. For comparison, we also have tested our hardware-accelerated version with a software implementation of our discrete algorithm. We provide computation times of our implementation for the three-dimensional case. Our data were generated on a 3.2 GHz Pentium 4, equipped with 2 GB of RAM and an NVIDIA GeForce 6800GT graphics card running Windows XP.

For performance evaluation of the two discrete Voronoi approaches, it is reasonable to compare computation times of both algorithms while varying the number of input values. The two-dimensional test case is based on a Voronoi diagram on a 512×512 pixel grid, and the three-dimensional test case is based on a $128 \times 128 \times 128$ voxel grid using m randomly generated sites. In the two-dimensional case, the results obtained with the geometric computation of the Voronoi diagram are compared with those obtained with the discrete approaches using distance field rendering and *k*d-trees. The three-dimensional case comparison is done only between the two discrete approaches. Figure 9 shows a graph of computation times for different Voronoi approaches. For a relatively smaller number of samples, the *k*d-tree approach takes more time but as the number of samples increase, the *k*d-tree approach exhibits only a slow growth, while both the distance field rendering approach and the geometrical approach exhibit much faster growth in computation time.



(a) (b)

Fig. 8. (a) 6400 uniform random sampling of sites vs. (b) non-uniform sampling of sites of Lena image.

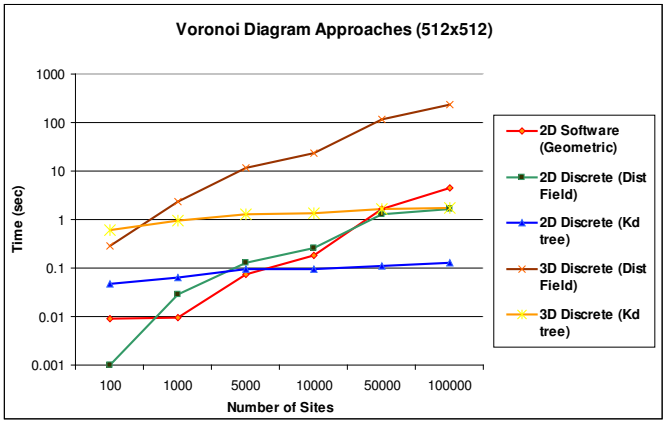


Fig. 9. Comparison of performance of Voronoi diagram computation using a geometric approach vs. discrete approaches using distance fields and *k*d-trees.

To analyze the performance of the optimized version of Sibson's two-dimensional interpolation scheme versus an optimized software implementation of the geometric approach, we have choosen n randomly generated sites on a 512×512 pixel grid. Instead of just using uniformly distributed random sites, we also have tested our algorithm against adaptively sampled sites to more accurately simulate "real" scattered data. We have used the adaptive sampling as described by Kreylos et al. [23]. Figure 8 shows an example of 6400 sites sampled with uniform distribution and adaptively sampled sites. Figure 10 shows the computational time required to perform interpolation. For small number of sites, the geometric approach of Sibson's

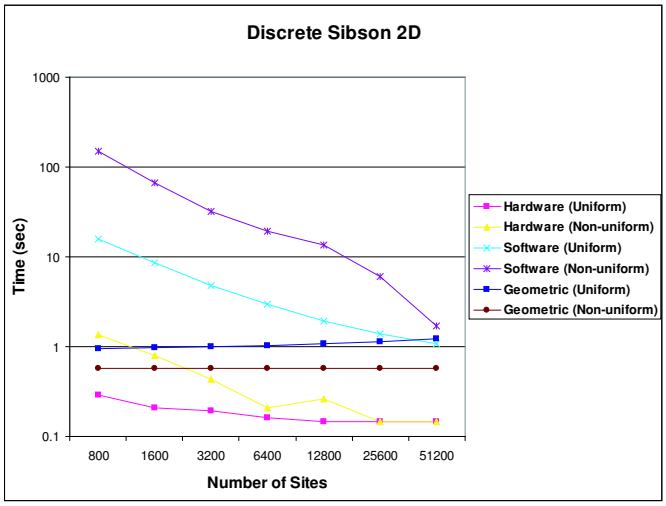


Fig. 10. Comparison of performance of two-dimensional Sibson interpolation using geometric vs. discrete approach.

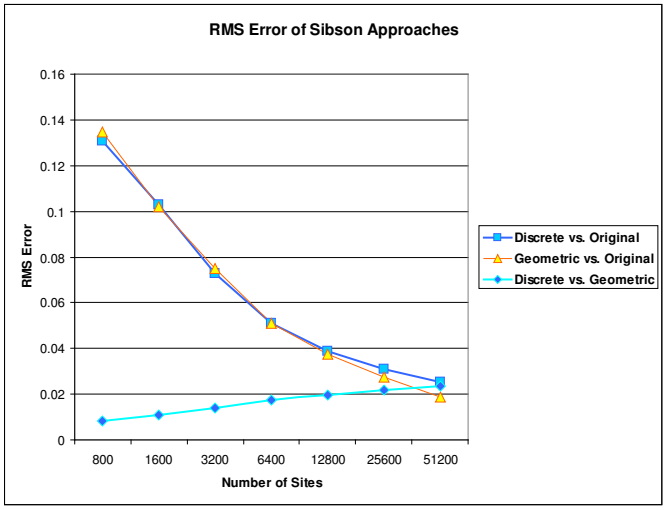


Fig. 11. Root-mean-square (RMS) of two-dimensional Sibson interpolation resulting from geometric, discrete, and original approaches.

interpolation exhibits a superior performance when compared with the discrete algorithms. As the number of data sites increases, the discrete version exhibits a significantly better performance. To test the correctness of the discrete approach, we have computed differences between the two images. As shown in Figure 12, slight differences can be seen around sharp edges.

Table I shows the performance of the discrete Sibson algorithm in the three-dimensions based on a $128 \times 128 \times 128$ voxel grid using m randomly generated sites from a CT head data set. Only the optimized case has been tested. For few sites, the discrete approach is slow. This is due to the relatively large radii of spheres that have to be rendered for each voxel. With more sites, however, the interpolation is significantly faster. We have also reconstructed data from adaptively sampled version of the CT head data set as described by Co et al. in [24]. Due to adaptive sampling, there are less points outside the head. As a result, the overall speed is significantly slower than the

TABLE I

PERFORMANCE DATA FOR 3D DISCRETE SIBSON INTERPOLATION OF UNIFORM-RANDOMLY SAMPLED CT HEAD DATA SET (DATA SIZE: 128^3) .

# sites	Voronoi (sec)	Discrete 3D (sec)	Total Time (sec)
100	1.39	92.05	93.44
1000	2.16	13.05	15.21
10000	3.08	6.45	9.53
100000	3.83	3.98	7.81
200000	4.17	3.55	7.72

randomly sampled points. The performance trend as in the other case increases with increasing number of sample points. Figure 13 shows a visualization of the reconstruction.

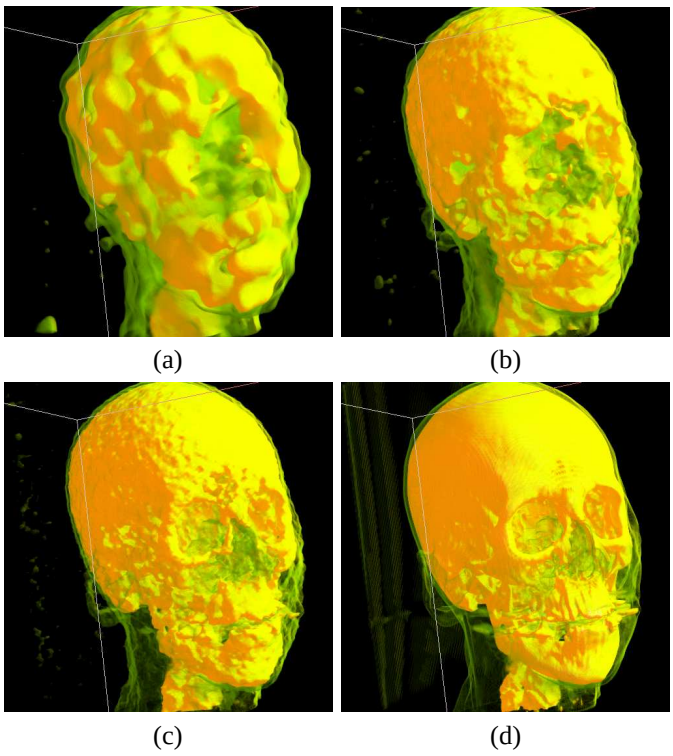


Fig. 13. Discrete Sibson interpolation and visualization of CT Head data set (data size: 128^3) using (a) 10,000, (b) 100,000, and (c) 500,000 scattered data points, compared to visualization using (d) 128^3 gridded points.

VI. DISCUSSION

Considering the two discrete Voronoi approaches presented, we observe that both methods have advantages and disadvantages. Using the distance field rendering approach yields very good performance, up to a certain number of sites. Computational time increases drastically with increasing number of additional sites. In the two-dimensional case, as shown in Section IV-A, the computation time for up to 1,000 sites is less than the time required by the *k*d-tree method. For 1,000 points on a grid of 512×512, the rendering time increases linearly with additional sites: The algorithm is a brute-force approach, where each additional site rendered compares its distance to every discrete point in the grid. The time complexity of this approach is $O(mP)$, where m is the number of sites and P is

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

Fig. 12. Comparison of computational geometry-based and discrete Sibson interpolation using 6,400 data points. (a) Original image (512×512 pixels); (b) result using geometric Sibson interpolation; (c) result using discrete Sibson interpolation; (d) difference image between (b) and (c).

the number of pixels in the grid. The *k*d-tree approach exhibits slightly different behavior. Computation time when dealing with a small number of sites is slow when compared to the distance field approach. However, computational time for a large number of sites is faster since the computational time increases slower compared to the distance approach with the increasing number of sites. This behavior is expected since the Voronoi diagram is rendered by querying the *k*d-tree at each grid location for its closest neighbor, and a search in *k*d-trees requires $O(\log m)$ leading to a total time complexity of $O((m+P)\log m)$. In the three-dimensional case, the results show analogous behavior for the two approaches.

When comparing the discrete approaches to the "exact" geometrical approach, the *k*d-tree approach is significantly faster when the number of sites increases. Compared to the distance field approach, the geometrical approach exhibits better performance until the number of sites reaches 5,000. Although the *k*d-tree approach shows better performance for higher numbers of sites, the discrete approach suffers from a few drawbacks. For Voronoi sites that lie within a discrete element, it would require us to use a higher-resolution grid to produce accurate results, in which case it could take much longer than the geometric approach. Also, the discrete approach would be less efficient in the case of a small number of sites and a large grid, since the geometric approach's performance is less dependent on grid size.

Discrete Sibson interpolation is, considering a small number of sites, computationally slow – significantly slower than the geometrical approach. However, as the number of sites increases, computation time decreases, and our approach shows a significant speedup over the software implementation. In the two-dimensional case, for around 1,000 sites the computational time required for constructing Sibson's interpolant is less than a second. Computing the Voronoi diagram and Sibson's interpolant can be done in under a second. We have not yet compared our discrete three-dimensional algorithm to a software implementation, since we do not have access to a fast and robust implementation. We expect our three-dimensional method to perform well, since it uses the same simple algorithm underlying the two-dimensional method. A three-dimensional geometrically exact method would be much more complicated to implement, and much more difficult to optimize. Furthermore, computation times for rendering a discrete three-dimensional Sibson's interpolant are very low, and computation time, as in the two-dimensional case, decreases with an increasing number of sites.

To investigate the approximation quality of our algorithm, we compare images generated by discrete Sibson interpolation with images generated using a computational geometry-based algorithm. One test case is shown in Figure 12: The difference image (Figure 12(d)) shows that the two images are almost identical. However, differences can be noticed around the edges of the image. (The resulting image was generated from adaptively sampled data as shown in Figure 8.) The reason for such differences is due to aliasing errors: There are more samples around the silhouette of the face, which implies drawing of many small circles in that area. Discrete Sibson interpolation is prone to aliasing, especially as the number of data points increases and the radius of rendered *d*-dimensional spheres decreases. One could compute more accurate Sibson values by computing the coverage area of the rendered spheres on partially covered pixels. Such values for pixels could be used as weights when accumulating data values to lead to accurate results. Calculating the exact coverage area would be compute-intensive. A good estimate, however, can be obtained by simply increasing grid sizes or rendering anti-aliased primitives.

To quantify the errors of our discrete approach, we have computed root-mean-square (RMS) error values. Figure 11 shows the results obtained with the exact geometric and discrete approach are very similar, for small number of sites. As the number of samples increases, however, the geometric approach produces better approximations. Also, when the discrete approach is compared against the geometric approach, the RMS error increases in linear fashion as the number of samples increases. The errors, in both cases, are due to aliasing resulting from drawing increasingly large numbers of increasingly smaller circles.

VII. Conclusions and Future Work

We have presented a method for rapid computation of Sibson's interpolant for two-dimensional and three-dimensional scattered data using contemporary graphics hardware. Our technique is based on discretization and takes advantage of geometrical properties of Sibson's interpolation scheme, which reduces the d-dimensional interpolation problem to rendering and blending d-dimensional spheres of easily computable radii. The interpolation algorithm is simple and can be accelerated by using modern programmable graphics hardware. Our approach is designed for interpolation of scattered data sets with many sites, for which it achieves a major improvement in performance compared to classical approaches. We have analyzed and compared the two-dimensional discrete approach with a software implementation of the traditional geometric approach, and have tested it for the three-dimensional case. Our discrete Sibson interpolation approach is fast, easy to implement, and generalizes to higher dimension.

We plan to adapt the discrete approach to other distance-based scattered data interpolation methods. Moreover, we plan to investigate acceleration techniques for the three-dimensional approach in support of interactive visualization of volumetric scattered data applications.

Acknowledgments

This work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, through the National Partnership for Advanced Computational Infrastructure (NPACI) and a large Information Technology Research (ITR) grant; and the National Institutes of Health under contract P20 MH60975-06A2, funded by the National Institute of Mental Health and the National Science Foundation. We thank the members of the Visualization and Computer Graphics Research Group at the Institute for Data Analysis and Visualization (IDAV) at the University of California, Davis.

References

[1] P. Alfeld, *Mathematical Methods in Computer Aided Geometric Design*, ch. Scattered data interpolation in three or more variables, pp. 1–33. Academic Press, 1989.

[2] G. Farin, "Surfaces over dirichlet tessellations," *Computer Aided Geometric Design*, vol. 7, no. 1-4, pp. 281–292, 1990.

[3] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver, "Fast computation of generalized Voronoi diagrams using graphics hardware," in *Proceedings of SIGGRAPH 99* (A. Rockwood, ed.), pp. 277–286, ACM, ACM Press, August 1999.

[4] W. Heidrich and H.-P. Seidel, "Realistic, hardware-accelerated shading and lighting," in *Proceedings of SIGGRAPH 99* (A. Rockwood, ed.), pp. 171–178, ACM, ACM Press, August 1999.

[5] M. J. Harris, G. Coombe, T. Scheuermann, and A. Lastra, "Physically-based visual simulation on graphics hardware," in *Graphics Hardware 2002*, pp. 109–118, Sept. 2002.

[6] J. Krüger and R. Westermann, "Linear algebra operators for GPU implementation of numerical algorithms," *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, vol. 22, pp. 908–916, July 2003.

[7] C. J. Thompson, S. Hahn, and M. Oskin, "Using modern graphics architectures for general-purpose computing: A framework and analysis," in *35th Annual International Symposium on Microarchitecture*, pp. 306–317, Nov. 2002.

[8] I. Amidror, "Scattered data interpolation methods for electronic imaging systems: A survey," *Journal of Electronic Imaging*, vol. 2, no. 11, pp. 157–176, 2002.

[9] T. A. Foley, D. A. Lane, G. M. Nielson, R. Franke, and H. Hagen, "Interpolation of scattered data on closed surfaces," *Comput. Aided Geom. Des.*, vol. 7, no. 1-4, pp. 303–312, 1990.

[10] R. Franke and G. Nielson, *Geometric Modeling: Methods and Applications*, ch. Scattered Data Interpolation: A Tutorial and Survery, pp. 131–160. SpringerVerlag, New York, 1991.

[11] S. K. Lodha and R. Franke, "Scattered data techniques for surfaces," in *Proceedings of Dagstuhl Conference on Scientific Visualization*, pp. 182–222, IEEE Computer Society Pres, 1999.

[12] G. Nielson, "Scattered data modeling," *IEEE Computer Graphics and Applications*, pp. 60–70, January 1993.

[13] G. M. Nielson, T. A. Foley, B. Hamann, and D. Lane, "Visualizing and modeling scattered multivariate data," *IEEE Comput. Graph. Appl.*, vol. 11, no. 3, pp. 47–55, 1991.

[14] D. Shepard, "A two-dimensional interpolation function for irregularly spaced data," in *Proceedings of 23rd National Conference*, pp. 517–524, ACM, August 1968.

[15] R. Sibson, "A vector identity for the dirichlet tessellation," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 87, no. 1, pp. 151–155, 1980.

[16] S. J. Owen, "An Implementation of Natural Neighbor Interpolation in Three Dimensions," Master's thesis, Brigham Young University, 1992.

[17] N. Sukumar, B. Morann, and T. Belyschko, "The natural element method in solid mechanics," *International Journal for Numerical Methods in Engineering*, vol. 43, pp. 839–887, Nov. 1998.

[18] C. Sigg, R. Peikert, and M. Gross, "Signed distance transform using graphics hardware," in *Proceedings of IEEE Visualization '03*, 2003.

[19] Y. Jang, M. Weiler, M. Hopf, J. Huang, D. S. Ebert, K. P. Gaither, and T. Ertl, "Interactively Visualizing Procedurally Encoded Scalar Fields," in *Proceedings of EG/IEEE TCVG Symposium on Visualization VisSym '04 (to appear)* (O. Deussen, C. Hansen, D. Keim, and D. Saupe, eds.), 2004.

[20] Q. Fan, A. Efrat, V. Koltun, S. Krishnan, and S. Venkatasubramanian, "Hardware assisted natural neighbour interpolation," in *Proc. 7th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2005.

[21] Y. L. H. Yee, "An implementation of natural neighbor interpolation in three dimensions," Master's thesis, Brigham Young University, 1992.

[22] J.-D. Boissonnat and F. Cazals, "Smooth surface reconstruction via natural neighbour interpolation of distance functions," in *Proceedings of the Sixteenth Annual Symposium on Computational Geometry*, pp. 223–232, ACM Press, 2000.

[23] O. Kreylos and B. Hamann, "On simulated annealing and the construction of linear spline approximations for scattered data," in *Proceedings of the Joint EUROGRAPHICS and IEEE TVCG Symposium on Visualization (VisSym '99)* (E. Groeller, H. Loeffelmann, and W. Ribarsky, eds.), pp. 189–198, Springer-Verlag, Vienna, Austria, 1999.

[24] C. S. Co, B. Heckel, H. Hagen, B. Hamann, and K. I. Joy, "Hierarchical Clustering for Unstructured Volumetric Scalar Fields," in *Proceedings of IEEE Visualization 2003* (G. Turk, J. J. van Wijk, and R. Moorhead, eds.), pp. 325–332, IEEE, Oct. 19–24 2003.