Discretized Sibson's Interpolation

Sung W. Park* Lars Jörn Linsen† Oliver Kreylos‡ Bernd Hamann§ John D. Owens¶

Center for Image Processing and Integrated Computing (CIPIC)
University of California, Davis

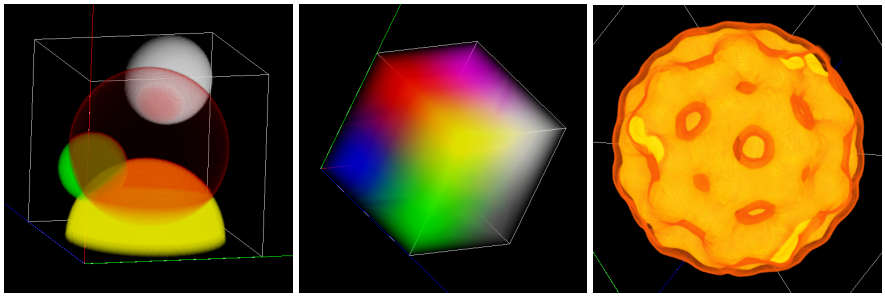

Figure 1: Sibson's 3D

Abstract

Natural neighbor interpolation such as Sibson's are well-known schemes for multivariate data fitting and reconstruction. Despite its nice properties, typical implementations of Sibson's interpolation are compute intensive and difficult to implement especially when applying it to higher-dimensional data. The main costs orignate from computing Voronoi tessellations for every position to be interpolated. Instead of computing Sibson's interpolant geometrically, we operate on a discretized domain. Discretizing Sibson's algorithm can be easily implemented by using a discrete Voronoi tessellation. We present a more efficient approach by exploiting geometrical properties of Sibson's interpolant. We show that this approach does not require an explicit Voronoi tessellation and that it can be easily mapped to commodity graphics hardware to exploit its parallelism and speed. Our approach gives a significant increase in speed compared to traditional approaches, is easy to implement, and generalizes to higher dimensions.

CR Categories: K.6.1 [Management of Computing and Information Systems]: Project and People Management—Life Cycle; K.7.m [The Computing Profession]: Miscellaneous—Ethics

Keywords: Scattered Data Interpolation, Natural Neighbor Interpolation, Graphics Hardware

1 Introduction

"Point-cloud" or "scattered data" visualization is becoming increasingly important in new emerging applications, especially in sensor network data analysis. With the advances in wireless sensor networks, where small, independent sensors are deployed to collect data at random points in space and time, interpolating between such points is necessary to make meaningful visualization. The measured quantity, e.g., temperature or humidity, can be described as a time-varying 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). In order to analyze or visualize the measured function f, one has to reconstruct it from the samples reported by the sensor network. While traditional applications of scattered data approaches are limited to handle a few thousand sites, sensor netowrks are steadly increasing, leading to millions of sites and generating vast amounts of data.

Common scientific visualization techniques require data to include connectivity information. Scattered data does not provide such information. Techniques used to deal with such unconnected data include the use of field reconstruction methods producing an analytical definition that is then resampled to a standard grid format supported by standard visualization methods, such as volume rendering and isosurface extraction. Existing scattered data techniques including radial basis function methods, Shepard's methods and its variants, and triangulation-based methods imply an interpolation scheme based on using all samples or selected subset of samples. Many of these techniques are computationally expensive and do not scale well with the number of data points, thus are not geared toward real-time visualization.

*e-mail:sunpark@ucdavis.edu
†e-mail:llinsen@ucdavis.edu
‡e-mail:kreylos@cs.ucdavis.edu
§e-mail:hamann@cs.ucdavis.edu
¶e-mail:jowens@ucdavis.edu

Natural neighbor interpolation, such as Sibson's, is a scattered data interpolation scheme based on Voronoi (or Dirichlet or Thiessen) diagram of the dataset.[1] Sibson's interpolant is known to produce good interpolation results and has remarkable properties such as linear precision, requirement of only local neighbors, and C^1 continuity[Alfred]. As is with many other schemes however, Sibson's interpolation is computationally expensive and difficult to compute especially when it's extended to higher dimensions. The high computational costs are due to the fact that for each interpolant, an insertion of a site into the Voronoi diagram is required for the calculation of proper weight contribution from neighboring sites.

With recent advancement and breakthrough in graphics hardware, modern hardware not only provides much improvement in performance, it also provides programmability in vertex and fragment engines. The new features allow for some of the general computing to be done in graphics hardware to exploit its speed and parallelism. [references...]

In this paper, we present a discrete Sibson's interpolation scheme. Instead of computing the interpolation geometrically, we present an extremely fast and efficient Sibson's interpolation scheme operated on a discretized domain. Discretizing Sibson's algorithm can be easily implemented by using a discrete Voronoi diagram. Hoff III et al. [Hoff III et al. 1999] have shown that a generalized discrete Voronoi diagram can be efficiently computed using graphics hadware. Their approach works well for generating Voronoi diagram of small datasets but doesn't scale very well for large datasets. For handling large datasets, we introduce a simple and efficient method for consructing discretized Voronoi diagrams using Kd-trees. Using a discretized Voronoi diagram, Sibson's interpolation can be easily discretized. A simple discretization however, is inefficient. By exploiting geometrical properties of Sibson's interpolant, we show that Sibson's interpolation can be done very efficiently. This approach doesn't require explicit Voronoi diagram, and it can be easily mapped to commodity graphics hardware to exploit its parallelism and speed. This approach also generalizes very well to higher dimensions. In Section 3, we give details of our discretized Sibson's interpolation algorithmWith full optimization, we show that in the 2D case, image reconsruction can be done at an interactive frame rate. In the 3D case, the speed up is also quite significant. Our implmentation and results are discussed in Section 4. Our algorithm gives significant increase in speed, reduced storage space and simplicity in implementation. A detailed discussion is provided by Section 5.

2 Related Work

2.1 Scattered Data Interpolation

Scattered data interpolation has been a research focus for a long time. Thus, a large number of scattered data interpolation techniques are available today. There are many good surveys that discuss different scattered data approachs. [cite Neilson, Hagen, Lodha, Alfred...] Lohda and Franke [Lodha and Franke 1999] have broadly classified them in the following way: polynomial or piecewise continuous polynomial parametric solutions, algebraic solutions, radial basis function methods, Shepard's methods and its variants, and subdivision solution. Shepard's [Shepard 1968] inverse distance weighting functions, Hardy's radial basis multi-quadrics [Hardy 1971], and Sibson's nearest neighbor interpolation [Sibson 1981] are few examples.

Shepard's method or its variant defines the scattered data interpolant f as a weighted means of the values f_i by choosing some blending functions or weight functions. More precisely, $f = \sum_{i=1}^{N} w_i f_i$, where the weight functions is usually isotropic (circular in 2D and spherical in 3D with fixed radius), non-negative, and monotonically decreasing with distance from the point $\mathbf{x}$ (*NOTE. Define x). The isotropism can lead to undesired field reconstruction artifacts when sampling rates differ significantly in different directions.

Radial basis function methods such as Hardy's multiquadrics, inverse multiquadrics, and thin-plate spline interpolants have been applied to a large variety of scattered data interpolation problems. They allow for anisotropic support however, many methods cannot effectively handle large datasets due to instability and computational cost associated with computation of radial interpolation that grow with distance away from centers. but cannot effectively handle typical data sets consisting of more than 500 data points. Typically, data sets of 500 or more points will yield condition numbers that will swamp single-precision accuracy.[Nielson 1993]

For natural neighbor interpolation, the weight at point $\mathbf{x}$ is not dictated by the same length measure in all dimension, but by the appropriate Lebesgue measure of the space-directions. This allows for anisotropic supports, were the support size is not given by an L_2-metric but is ascerntained based on a geometric construct that defines the region-of-interaction between the nodes [sukumar].

2.2 Voronoi Diagram

A standard Voronoi diagram is a partitioning of a given region into cells based on a finite set of points, the Voronoi sites. Each cell consists of the points closest to a particular input point. More specifically, consider a d-dimensional, usually convex domain $\Omega \subset \Re^d$ and a set $\mathbf{N} = \{p_1, p_2, \ldots, p_m\}$ of m scattered points. The Voronoi diagram $\mathscr{V}(\mathbf{N})$ of the set $\mathbf{N}$ over domain Ω is a domain partitioning into regions $V(\mathbf{p}_i)$, such that any point in $V(\mathbf{p}_i)$ is closer to node $\mathbf{p}_i$ than to any other node $\mathbf{p}_j \in \mathbf{N}$ $(j \neq i)$. The region $V(\mathbf{p}_i)$, a Voronoi cell, for a node $\mathbf{p}_i$ within the convex hull is a convex polygon (polyhedron) in $\Re^d$:

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

where $d(x_i,x_j)$ is a metric, usually the Euclidean distance between x_i and x_j.

A number of algorithms exist today for computing Voronoi diagrams of points in 2D, 3D, and higher dimension. [shamo75, fortu86,inaga92,sugih94, guibas, boissonat...] Efficient adaptive precision and exact arithmatic is an active area of research. There are also many approximate Voronoi diagram computation based on space subdivision [Etzion99,Vleugels95,Vleug96].

2.3 Sibson's Interpolation

A natural neighbor interpolation known as Sibson's interpolation, was first introduced by Sibson [Sibson 1981]. This scheme uses a weighted average of surrounding or neighboring data points to compute the interpolant. Although it is similar to Shepard's method [Franke,80] the fundamental difference is how the weights are assigned to the neighboring data points. To define the weight of a neighboring data point, Sibson introduces the idea of natural neighbors. Natural neighbors define the weight or the amount of influence any data point will have on the computed function value at an interpolation point. The weight is entirely dependent on the area or the volume of the neighboring data points. The area or volume of

influence is defined by Voronoi tessellation of the dataset. For interpolating the data value at a point $\mathbf{p}$, insert $\mathbf{p}$ into the Voronoi diagram. When point $\mathbf{p}$ is inserted into the diagram as shown in Figure 2b, its Voronoi cell $V(\mathbf{p})$ has m neighboring cells $V(\mathbf{p}_1),\ldots,V(\mathbf{p}_m)$. $\mathbf{p}_1,\ldots,\mathbf{p}_m$ are called the natural neighbors of $\mathbf{p}$. it will have m neighbors $\mathbf{p}_1,\ldots,\mathbf{p}_m$. Each neighbor would contribute a portion of its own tile area to create p's tile. Each of these areas would then be a fraction u_i of p's total tile area, where

$$\sum_{i=1}^{m} u_i = 1$$

Sibson's interpolant of p is then,

$$p = \sum_{i=1}^{m} u_i p_i$$

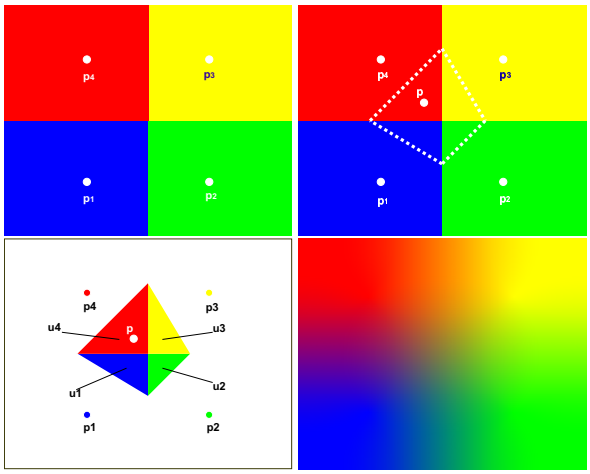

Figure 2: Sibson's Interpolation

An example can be seen in figure x.

Typically, Sibson's interpolation is implemented geometrically. One computes the weight of the interpolant by determining the volume of u_i of the natural region NR_{x,p_i}, associated with $\mathbf{p_i}$ upon insertion of $mathbf p$. Since this region is delimited by Voronoi centers in convex position, the convex hull and the volume can be computed from the collected natural region. Another method proposed by Owens [Owens], first computes the Voronoi region $V((p))$ and then subdivides the region into the natural neighbors sub-regions. The dual structure of Voronoi diagram, known as Delaunay triangulation also has been used to calculate Sibson's. Delaunay triangulation is used directly to calcuate the weight of the neighboring regions for interpolation [Boissonnat]. Most Sibson's algorithms are computationally expensive and very difficult to implement.

3 Discretized Sibson's Interpolation

In this section, we describe how Sibon's interpolation can be performed in a discrete domain. Before getting into the details of interpolation, we first describe how discrete Voronoi diagram is constructed for the interpolation purpose. We then present a simple, straighforward approach to discrete Sibson's interpolation and discuss its shortcomings. We exploit the geometrical properties of Sibson's interpolant to develop an improved, efficient approach of discretized Sibson's interpolation.

3.1 Discretized Voronoi Diagram

Hoff et al. [Hoff III et al. 1999] have presented a graphical way of computing generalized Voronoi diagram both in 2D and 3D. Their approach was based on the observation suggested in [Haebe90] and in the OpenGL 1.1 Programming Guide [Woo97], which gives the idea of rendering 2D Voronoi diagram using cones as distance functions for each site. By drawing the site's distance fields and utililizing graphics hardware's depth buffer function to render parts of geometry that are closest to each site, Hoff *et al.* show how Voronoi diagrams can be constructed. Their approach even generalizes to scenes that allow more than just points as sites. However, the algorithm doesn't scale well both in the number of sites and in dimensions. The bottleneck comes from each site having to render a distance field that covers the entire domain area, ie. a distance cone that covers the entire Voronoi image in 2D and the entire Voronoi volume in 3D. For a Voronoi diagram with a large number of sites, this approach is inefficient.

Hoff's algorithm works well for generating generalized discrete Voronoi diagram. For our purpose, we only need a discrete Voronoi diagram of point sites that can scale well to many sites and to higher dimensions. Instead of rendering distance fields, we present a more efficient approach for generating Voronoi diagram of point sites that scales well in both number of sites and in dimensions by using a spatial structure such as Kd-tree. A discrete Voronoi diagram holds information about the Voronoi site it belongs to and the distance to that site at each discrete location. We can use Kd-tree to query the desired information at each location. The Kd tree is used to store all the Voronoi sites. Then, depending on the precision of the discrete Voronoi diagram desired, a grid size is determined. At each grid location, the Kd-tree is queried to determine the site that is closest to the grid location. This query returns the associated Voronoi region. When the entire region is queried, Using the information, the nearest site value and the distance can be stored in the grid. When the entire region is queried, its result is a discretized Voronoi diagram. This approach is very efficient for building Voronoi digrams with large number of sites because Kd tree can be very efficiently built in $O(n\log n)$. Also, doing the nearest point search is done in $O(\log n)$. Compared to the discrete Voronoi approach in [Hoff] this method scales well because constructing a Voronoi diagram solely depends on the computational cost of inserting and querying the Kd tree as opposed to having to render a geometry that covers the entire grid region. This approach also scales very well in dimension. Unlike the other approach, it doesn't require each site to render geometry at each slice to cover the entire volume.

3.2 Discretizing Sibson's Interpolation

In the discretized domain, Sibson's interpolation can be easily computed. Since calculating Sibson's interpolant at point $\mathbf{p}$ is equivalent to taking the weighted average of values at the neighboring sites (see figure x), this value can be computed by averaging the discrete elements. Accumulating all the 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 gives the average of the region. This value is a good approximation to Sibson's interpolation value. Since discrete elements are accumulated at one location, we call this interpolation scheme "gather approach".

Given a discrete d-dimensional Voronoi diagram of m samples $\mathbf{p}_1,\ldots,\mathbf{p}_m \in N \subset \Re^d$, we can look up the following informaions for every raster position $\mathbf{p} \in \Re^d$ the distance $d_v = V_d(\mathbf{p})$ and the data value $c_v = V_c(\mathbf{p})$ at the site $\mathbf{p_j}$ closest to raster position $\mathbf{p}$. The d-

dimensional fields V_d and V_c representing the Voronoi diagram are precomputed as described in Section 3.1. To reconstruct the entire data field using a disretized Sibson's interpolant, we scan the rasterized domain and compute for each raster position $\mathbf{p}$ an interpolated value c. For deterimining the value c at each raster position $\mathbf{p}$, we accumulate all the values within the region $V(\mathbf{p})$ that we would get by inserting $\mathbf{p}$ as a new site into the Voronoi diagram. For the accumulation process, we initialize the value $c = 0$ and scan all the raster position $\mathbf{i}$. For all raster positions i, we check whether it lies in $V(\mathbf{p})$ by comparing the distance $V_d(\mathbf{i})$ of raster position $\mathbf{i}$ to the closest site with the distance from raster position $\mathbf{i}$ to raster poisition $\mathbf{p}$. If $\mathbf{i} \in V(\mathbf{p})$, we add $V_c(\mathbf{i})$ to value c. We count the number n of raster positions

$\mathbf{i} \in V(p)$ and divide the final value of c by this number n.

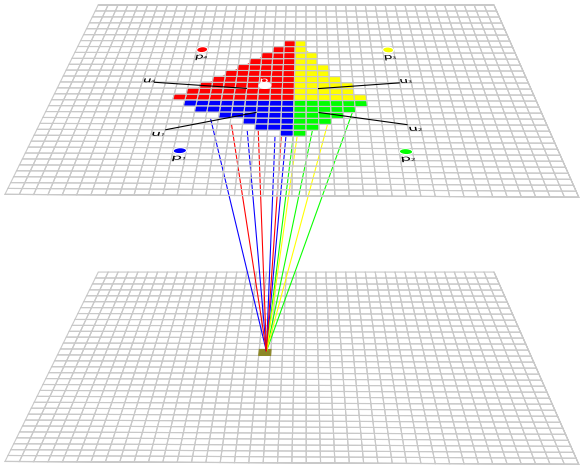

Figure 3: Gather approach - shows how values are gathered for one sibson's interpolant

(*Create a figure for this. 1)Voronoi Diagram 2)Inserted Region 3) Accumulate 4) Divide

This approach can be mapped to modern programmable graphics hardware easily at least in lower dimensions without explicitly inserting a new site to the existing Voronoi diagram in two rendering passes assuming that a discrete Voronoi diagram has been computed as described in the previous section. First pass involves accumulating all the values contributing to each values c at point $\mathbf{p}$.(***) In the second pass, the accumulated values are divided by the number of elements accumulated to get a Sibon's interpolated image.

The two steps can be easily implemented in a fragment program, but there are few inefficiencies. First is that, in order to go through every $mathbfi$ and accumulate the values at $\mathbf{p}$, a d-dimensional block quad that covers each possible $mathbfi$ has to be rendered. Unfortunately, since $\mathbf{p}$ is constant, this block will map to the single pixel $\mathbf{p}$ on the screen and hence only produce one fragment instead of P fragments. To produce P fragments, P points must be rendered, one at each possible $\mathbf{i}$. The fragment program is run on each of these P fragments, and the contribution can be summed into the framebuffer at $\mathbf{p}$. This process can be repeated for every $\mathbf{p}$, which requires rendering of P arrays of P points. Secondly, since the exact region of $\mathbf{i}$ that contribute to the value c at point $\mathbf{p}$ is not precisely known, every $\mathbf{i}$ has to be examine, even though most elements don't get used. A better approximation can be applied by figuring out the convex hull of all the $\mathbf{p}$'s, which would require added computation. In either case, the added computation makes this process less efficient.

This approach can be summarized in the following way:

- Precalculate the Voronoi diagram for n sites.
- Initialize value $c = 0$ and pixel count $n = 0$ for each $\mathbf{p}$.
- For every output location $\mathbf{p}$
  - for raster position $\mathbf{i}$
    - if $\mathbf{p} \in V(\mathbf{i})$, add $V_c(\mathbf{i})$ to value c at position $\mathbf{p}$ and increment n by 1.
- for every output location $\mathbf{p}$
  - Divide the accumulated value c by n.

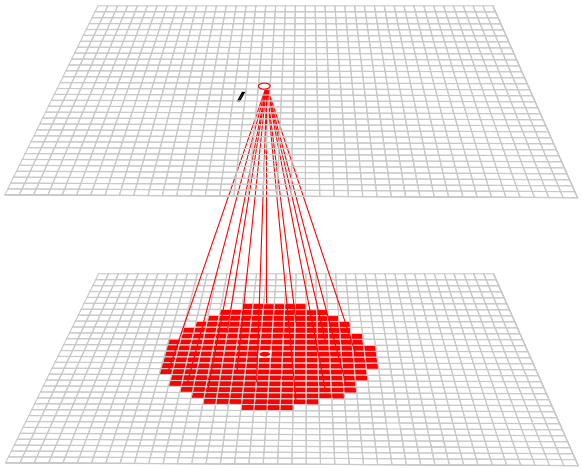

Figure 4: Scatter approach - show how one location contributes to the rest of the Sibson's image

3.3 Efficient Discretized Sibson's Interpolation

The key insight to making Sibson's interpolation more efficient is to consider the problem in the opposite direction where values are "scattered" rather than "gathered". Instead of looping over all the output locations $\mathbf{p}$ and gathering all $\mathbf{i}$ values into the value c at point $\mathbf{p}$, we loop over input locations $\mathbf{i}$ and consider at which positions $\mathbf{p}$ the value c is affected. This approach allows us to utilize the efficient rasterization units in the graphics hardware and to limit our computation to only those $\mathbf{i}$ values that affect the final result.

The scatter approach, works in the following way: Consider one single input point in the Voronoi diagram $\mathbf{i}$. We want to determine data value $V_c(\mathbf{i})$ contributes to. This is easily determined by rendering a d-dimensional block over all positions $\mathbf{p}$, and for each $\mathbf{p}$, determine if the distance between $\mathbf{i}$ and $\mathbf{p}$ is less than the distance $V_d(\mathbf{i})$ between i and its closest site. If it is, then $\mathbf{i} \in V(\mathbf{p})$, and we accumulate the value $V_c(\mathbf{i})$ to the value c at position $\mathbf{p}$

This approach can be summarized in the following way:

- Precalculate the Voronoi diagram for n sites.
- Initialize value $c = 0$ and pixel count $n = 0$ for each $\mathbf{p}$.
- For every pixel raster position $\mathbf{i}$:
  - for every output location $\mathbf{p}$
    - if $\mathbf{i} \in V(\mathbf{p})$, add $V_c(\mathbf{i})$ to value c at position $\mathbf{p}$ and increment n by 1.
- for every output location $\mathbf{p}$

– Divide the accumulated value c by n.

The scatter approach has advantage that we no longer need to render P separate points for a proper result; because we are scattering the result from $\mathbf{i}$ across all $\mathbf{p}$, we can render a single block that covers all $\mathbf{p}$ in the output image. Thus, P blocks can be rendered for the interpolation rather than P point arrays of P points. Moreover, we observed that in both scatter and gather approaches, particularly with dense samples, most $V_c(\mathbf{i})$ do not contribute to the values at position $\mathbf{p}$. In the gather approach, we noted that it would require finding a bounding box for $V(\mathbf{p})$ for each $\mathbf{p}$ to minimize this problem which is computationally expensive. In the scatter approach however, we can take advantage of this observation. We noted that any point i only contributes to the value at position $\mathbf{p}$ if the distance between $\mathbf{i}$ and $\mathbf{p}$ is less than its stored distance $V_d(\mathbf{i})$, the distance to the closest ssite. Consequently, we know that the contribution of $\mathbf{i}$ will never ranger farther than $V_d(\mathbf{i})$, and thus instead of rendering a block at $\mathbf{i}$ that covers the entire output, we only need to render a d-dimensional sphere with radius $V_d(\mathbf{i})$ at each site $\mathbf{i}$. This observation eliminates the need for distance comparasion at each $\mathbf{i}$ and minizes the Sibson's problem down to rendering d-dimensional spheres at each $\mathbf{p}$ by looking up the Voronoi diagram for its properties. Also, just rendering this smaller region, particularly with a large number of samples, makes the implmentaion considerably more efficient.

3.4 Implementation

To fully take advantage of the algorithm described in the last subsection using graphics hardware, the following capabilities are desired: floating point precision framebuffer and textures, floating point blending, and programmable vertex and fragment engines. Floating point precision is desired. When storing distance information in the Voronoi diagram, using only 8-bit precision, i.e. 256 values, information gets lost for distances between close points. Moreover, 8 bits are not enough to accumulate data values during the interpolation process. There are potentially thousands of data values that are accumulated. For 2D applications, programmable capability is not required, since Voronoi diagrams can be rendered using cones in 2D, and Sibson's interpolant can be calculated by simply rendering circles in 2D. In 3D however, programmability allows for easier and more precise implementation of the algorithm. For rendering Voronoi diagrams, instead of rendering distance fields, exact distance can be computed directly in the fragment program. Also, circles and spheres can be easily rendered using a fragment program and the extension from 2D to 3D is straightforward.

In our implmentation, we used OpenGL libraries and OpenGL's ARB fragment program in an ATI Radeon 9800 Pro card. For storing our Voronoi diagram and Sibson's interpolated image, we used a 4 channel 32-bit floating-point framebuffer. The RBG channels are used for storing the nearest site's data value and the A channel for storing the distance to the nearest site. Voronoi diagram generation using Kd-trees was implemented by querying the Kd-tree at each discrete location for the Voronoi region it was in. For computing Sibson's, the RGB channels were used to accumulate data values and the A channel was used for counting number of pixels accumulated. One shortcoming of the current generation graphics card that we had to get around is the lack of floating point blending. Fortunately, the next generation cards that are just about to come out will support the floating point blending. In our implmentation, we simulate floating point blending in the fragment program by storing output values and reading them back again. This feature is undocumented feature that we exploited on ATI card.

Implementing the fully optimized approach in 2D is very simple. First, we render a Voronoi diagram using graphics hardware by rendering a single quad that covers the entire screen for each site. Instead of rendering cones for each site, we computed distance from pixel raster position to the specific site in the fragment program. The data value is ouptted as color value and distance is outputted as both alpha value and depth value. Sibson's is rendered by drawing circles at each given raster position while looking up the Voronoi diagram for data value and radius. The data value is associated with color values of the circle and the radius information was used to render a square, centered at the raster position with side length twice the radius. Once all the values were accumulated, we used a fragment program to divide the accumulated values with the number of values accumulated to generate the final image.

In the 3D case, we took advantage of the graphics card by computing a slice of the volume at a time. We implemented the Voronoi diagram using both distance fields and Kd-trees. For Sibson's, spheres were rendered at each raster position of the volume. Again, the volume was reconstructed by looking at 2D slices at a time.

4 Results

For performance evaluation of our algorithm, we first compare the computing time of the two approaches we use for generating discretized Voronoi diagrams. Then, we compare both computation time and the difference image of 2D Sibson's interpolation using the traditional geometric approach and our optimized version of the hardware-accelerated for speed and approximation quality. Finally, we list computation times of our 3D implementation. Our programs were run on a 3.2 GHz Pentium 4 equipped with 2GB of RAM and an ATI Radeon 9800 Pro graphics card.

4.1 Discretized Voronoi Diagram

For perfomance evaluation of the two discretized Voronoi approach we used, we compared the computing times of both algorithms while varying the number of input values in 2D and 3D. The 2D test case shows the creation of the Voronoi diagram on a 512x512 grid and 3D test case shows 128x128x128-sized grid using n randomly generated number of sites. Table x. shows the computation times. Kd-tree approach shows a very slow growth whereas the distance field rendering approach shows much faster growth in computational time.

num sites	Dist 2D	Kd 2D	Dist 3D	Kd 3D
100	0.001	0.047		0.609
1000	0.028	0.065		0.975
5000	0.13	0.094		1.285
10000	0.26	0.094		1.369
50000	1.29	0.11		1.65
100000	1.64	0.13		1.716

To analyze the performance of the optimized version of Sibson's 2D interpolation versus a highly optimized software implementation of the geometric approach, we again choose n randomly generated numbers of sites on a 512x512 grid size. Table x shows the computational time to perform the interpolation from given Voronoi diagram.

num sites	Software 2D	Discretized 2D	Speedup
100		2.89	
1000		0.3	
5000		0.075	
10000		0.038	
50000		0.038	
100000		0.038	

Table x demonstrates the peformance of the discretized Sibson's algorithm in 3D. Only the optimized case is tested.

num sites	Voronoi	Discretized 3D	Total Time
100 0.619	170.33		
1000 0.975	20.96		
10000 1.359	7.425		
100000 1.69	6.309		
200000 1.913	6.188		

Finally, we test by changing the grid size for...

grid size	num sites	Voronoi	Discrete Sibson
64x64x64	10000	0.178	0.638
	100000	0.234	0.581
128x128x128	10000	1.40	7.406
	100000	1.79	6.403
256x256x256	10000	10.284	
	100000	12.91	

5 Discussion

From the two discretized Voronoi approaches examined in this paper, we observe that both methods have its advantages and disadvantages. Using the distance field rendering approach yields in very good performance up to a certain number of sites. Once it goes over a certain threhold, the computational time increases drasticaly with the number of additional sites added. In the 2D case as shown in Table x, the computation time up to 1,000 sites was faster than the Kd-tree mehtod. After 1,000 points on a grid of 512x512, the rendering time increased linearly with additional sites. This was to be expected because the algorithm by nature is a brute-force approach where each additional site that gets rendered compares its distance to every discrete point in the grid. The time complexity of this approach is $O(nP)$ where n is the number of sites and P is the number of pixels n the grid. The Kd-tree shows slightly different behavior. The computation time of computing small number of sites is very fast but slower compared to the distance field approach. The computation time however stays more or less constant as more points are addeded. The time increase for additional number of sites was very small. This behavior is expected because the Voronoi diagram is rendered by querying the Kd-tree at each grid location for its closest neighbor. The search in Kd-trees take $O(\log n)$ and, thus, it takes a total of $O(n \log n + P)$. In the 3D case, the results show analogous behavior for the two approaches. We conclude that only if we're dealing with sites less than some threhold number, typically small, it's is more efficient to use the distance field rendeing approach. Otherwise, Kd-tree approach works well and scales much better.

For the correctness of our algorithm, we tested our image generated from discretized Sibson's interpolation with an image generated from software implmentation. Figure x shows two images generated. As it can be seen on the difference image, the two images are almost identical, however differences are noticed around edges. As the number of sites increase, discretized Sibson's interpolation method is prone to resolution error, a result of discrete sampling. If the sampling is too coarse, we may miss some region or add too much of one region. For a more precise interpolation, an adaptive resolution method can be employed. A region of interest can be zoomed in for reducing potential resolution error.

Discretized Sibson's interpolation shows that for a few number of sites, the approach is computationally slow. It is signficantly slower than the geometrical approach. However, as the number of sites increase, the computation time goes down and our approach shows a signficant speedup over the software implmentation. In the 2D case, around 1000 sites the computational time of rendering Sibson's is much lower than a second. Computing both Voroni diagram and Sibson's can be done under a second. We did not get to test the 3D version with a software implmentation but the computation time for rendering discrete Sibson's interpolation was still very fast. The computation time, as in 2D case, decreased with increasing number of sites.

6 Conclusion and Future Work

We have presented a method for rapid computation of Sibson's interpolation in both 2D and 3D using graphics hardware. It is based on the idea of discretization. We have presented a technique that takes advantage of nice geometrical properties of Sibson's interpolation, which reduces the d-dimensional problem to rendering d-dimensional spheres of known radii and blending them. The interpolation is simple and can be speeded up using graphics hardware. This approach does not perform as well for small datasets but with more input data, the approach performs very well. We have analyzed and compared the 2D discrete approach with a software implementation of the traditional geometric approach and implemented the generalization of our approach to 3D. To conclude, we have shown that our discretized Sibson's interpolation approach is fast, easy to implement, and generalizes well to higher dimension.

For future work, we would like to transfer discrete approach to other distance-based scattered data interpolation methods. Moreover, we plan on looking into acceleration techniques for the 3D approach for interactive visualization of the scattered data interpolation.

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; the National Institutes of Health under contract P20 MH60975-06A2, funded by the National Institute of Mental Health and the National Science Foundation; and the U.S. Bureau of Reclamation. We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

References

EXLUNA, INC. 2002. *Entropy 3.1 Technical Reference*, January.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual simulation of smoke. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 15–22.

HARDY, R. 1971. Multiquadratic equations of topography and other irregular surfaces. *Journal of Geophysical Research 76*, 1905–1915.

HOFF III, K. E., KEYSER, J., LIN, M., MANOCHA, D., AND CULVER, T. 1999. Fast computation of generalized Voronoi diagrams using graphics hardware. *Computer Graphics 33*, Annual Conference Series, 277–286.

JOBSON, D. J., RAHMAN, Z., AND WOODELL, G. A. 1995. Retinex image processing: Improved fidelity to direct visual observation. In *Proceedings of the IS&T Fourth Color Imaging Conference: Color Science, Systems, and Applications*, vol. 4, 124–125.

KARTCH, D. 2000. *Efficient Rendering and Compression for Full-Parallax Computer-Generated Holographic Stereograms*. PhD thesis, Cornell University.

LANDIS, H., 2002. Global illumination in production. ACM SIGGRAPH 2002 Course #16 Notes, July.

LEVOY, M., PULLI, K., CURLESS, B., RUSINKIEWICZ, S., KOLLER, D., PEREIRA, L., GINZTON, M., ANDERSON, S., DAVIS, J., GINSBERG, J., SHADE, J., AND FULK, D. 2000. The digital michelangelo project. In *Proceedings of SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH, New York, K. Akeley, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 131–144.

LODHA, S. K., AND FRANKE, R. 1999. Scattered data techniques for surfaces. In *Proceedings of Dagstuhl Conference on Scientific Visualization*, IEEE Computer Society Pres, 182–222.

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

PARKE, F. I., AND WATERS, K. 1996. *Computer Facial Animation*. A. K. Peters.

SAKO, Y., AND FUJIMURA, K. 2000. Shape similarity by homotropic deformation. *The Visual Computer 16*, 1, 47–61.

SHEPARD, D. 1968. A two-dimensional interpolation function for irregularly spaced data. In *Proceedings of 23rd National Conference*, ACM, 517–524.

SIBSON, R. 1981. A brief description of natural neighbor interpolation. 517–524.

YEE, Y. L. H. 2000. *Spatiotemporal sensistivity and visual attention for efficient rendering of dynamic environments*. Master's thesis, Cornell University.