Christopher S. Co
June 2006
Computer Science

Meshless Methods for Volume Visualization

Abstract

A revolution in sensor technology is introducing new problems that must be addressed by visualization research. These technologies will enable earthquake engineers to monitor tectonic plate activity, park rangers to respond more effectively to wildfires, and marine biologists to uncover mysteries from the ocean depths using a network of small, lightweight sensors collecting data in real-time. Visualizing data from these *sensor networks* is crucial to understanding the information the data represents. However, data collected from such networks are scattered through space and time without explicit connectivity information, posing difficult challenges to researchers and engineers. While past research on such *meshless* data has focused primarily on interpolation schemes, we propose the development of visualization algorithms and processing frameworks in which existing interpolation schemes can be applied. This work not only plays a significant role in supporting the continued development of sensor network technology; it fills a major gap in visualization research, since effective *direct* methods for meshless data visualization did not previously exist. Meshless methods can be applied to existing problems encountered in visualizing data with a mesh, giving a set of common techniques that can be applied to a broad class of data sets.

Professor Kenneth I. Joy
Dissertation Committee Chair

Meshless Methods for Volume Visualization

By

CHRISTOPHER S. CO

B.S. (University of California, Davis) 2001
M.S. (University of California, Davis) 2002

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

_____

_____

_____

Committee in charge

2006

Meshless Methods for Volume Visualization

Christopher S. Co

For my family, especially my beloved Yunn.

Acknowledgments

This document represents the cumulative effort of several people, all of whom had a profound impact on the work presented here and on me. In fact, there have been so many people who have contributed to this work that I hesitate to list them, for fear that my poor memory will overlook some people. Nevertheless, those I do happen to remember deserve mentioning.

First, I wish to thank my advisor, Ken Joy, who took a chance on me on as an undergraduate researcher so long ago and has served as my primary source of guidance and inspiration. I have learned so much from Ken academically, professionally, and personally, and for that I am deeply indebted to him. He inspired me to be the best student I could be. But even during the hard times, Ken always showed patience, support, and uplifting enthusiasm. He has been the best boss I have ever had, providing me excellent resources and a large amount of freedom (perhaps more than I needed or deserved). A large part of this accomplishment belongs to Ken, a great mentor and an even greater friend.

Second, I wish to thank all those who have served as mentors and colleagues through the years. Everyone in the Visualization and Graphics Research Group at the Institute for Data Analysis and Visualization (IDAV) have helped me grow intellectually. In particular, I would like to thank my other committee members, Bernd Hamann and Nina Amenta. Hans Hagen helped jump start my career in research, leading me to my first research publication. Wes Bethel, John Shalf, and Terry Ligocki of Lawrence Berkeley Lab and Nelson Max offered me my first opportunities to solve application-driven research in a large-scale research environment. N. Sukumar, Oliver Staadt, and John Owens often provided sound advice and voices of encouragement. Aaron, Adam, Alex Pomeranz, Ben, Bjoern, Brett, Brian, Christian, Christof, Conrad, David, Deb, Gunther, Haeyoung, Ikuko, Janine, Jaya, Joerg, Karim, Ken Waters, Lars, Lok, Louis, Nameeta, Oliver Kreylos, Ralph, Serban, Shubho, Soon Tee, Timo, Yong—they and countless others made my time in the lab not only intellectually stimulating but also a heck of a lot of fun. Pat, Jamie, Maria, Grace, Lu—although they work tirelessly and thanklessly behind the scenes, I know that this work could not have been done without them.

Third, I would like to thank the generous funding agencies that supported my research. The Na-

tional Science Foundation, the National Partnership for Advanced Computational Infrastructure, the National Institutes of Health, Lawrence Livermore National Laboratory, and Lawrence Berkeley National Laboratory—without their financial backing, this work would not have been possible.

Most importantly, I am forever grateful to my family and friends for sharing my successes (and failures) with me. Mom, Dad, Patrick, Mark, Michele, Dominic, Grace, Grandma Hsu, Grandpa Hsu, Mama Hsu, Papa Hsu, Emily, Stanley, Lee, Eddie, Adriane, Fabien, Sophie, Alden—they have given me strength by showing me that my success in life is also their own. My wife Winnie shares the largest part of this accomplishment, as our life together makes all of this worthwhile.

Contents

Abstract

A revolution in sensor technology is introducing new problems that must be addressed by visualization research. These technologies will enable earthquake engineers to monitor tectonic plate activity, park rangers to respond more effectively to wildfires, and marine biologists to uncover mysteries from the ocean depths using a network of small, lightweight sensors collecting data in real-time. Visualizing data from these *sensor networks* is crucial to understanding the information the data represents. However, data collected from such networks are scattered through space and time without explicit connectivity information, posing difficult challenges to researchers and engineers. While past research on such *meshless* data has focused primarily on interpolation schemes, we propose the development of visualization algorithms and processing frameworks in which existing interpolation schemes can be applied. This work not only plays a significant role in supporting the continued development of sensor network technology; it fills a major gap in visualization research, since effective *direct* methods for meshless data visualization did not previously exist. Meshless methods can be applied to existing problems encountered in visualizing data with a mesh, giving a set of common techniques that can be applied to a broad class of data sets.

Professor Kenneth I. Joy
Dissertation Committee Chair

Chapter 1

Introduction

Today we are capable of collecting massive amounts of highly sophisticated data. To use this data effectively, our ability to derive meaningful information from this data must also evolve in sophistication. Visual analysis methods will play a key role in helping us accomplish this.

Visualization is about visual communication and visual data analysis. To date, visualization techniques have predominantly focused on structured data, where samples are aligned in a controllable and predictable fashion. We are on the verge of an unstructured data revolution, where samples are scattered in time and space without an underlying mesh. Imagine every cell phone or digital camera equipped with GPS technology and sensors. Imagine wireless sensor networks [14, 39] actively monitoring the structural soundness of a building during an earthquake. Imagine micro-machines navigating patients' bodies collecting real-time medical information for surgical planning. The impact of these technologies relies on accurate visual communication of the data, which implies visualization and analysis methods for this meshless unstructured data must be developed. The research presented here focuses on the development of fundamental algorithms for meshless data visualization.

Until recently, state-of-the-art algorithms in this field globally resampled meshless unstructured data to a structured format and used conventional structured-data display techniques. The main drawback of this approach is that the resampling step introduces several approximation (or interpolation)

steps resulting in accumulated error. Obscured by error, meaningful information becomes lost. Resampling to a high-resolution mesh can reduce the error, but increases storage cost. Little theory has been provided to determine sampling parameters necessary to provide accurate results. In most scenarios, domain-specific knowledge is applied to estimate these parameters, however, such domain-specific knowledge does not always exist. The goal of this research is to remove guesswork from meshless data visualization by focusing on methods that work directly with the original data.

The work presented takes important first steps toward building a viable solution to meshless data exploration and lays some theoretical foundation for continued research in this area. It is likely that meshless representations will become more attractive with the availability of algorithms to produce visual representations of the data. Subsequently, these methods will foster the continued development of meshless data collection technologies, such as sensor networks. The meshless methods described in some cases solve existing problems in the visualization of non-meshless data and thus are broadly applicable to a wide variety of data sets.

Our development and study of meshless methods for scientific visualization primarily focuses on isosurface generation. After providing the necessary background in Chapter 2, we discuss a point-based isosurface generation algorithm designed for data sets containing a mesh called *iso-splatting* in Chapter 3. In Chapter 4, we describe how this point-based technique is adapted to compute continuous isosurfaces from multiblock data sets. The flexibility of these two mesh-based methods enable their application to computing isosurfaces from arbitrary meshless data sets, which we describe in Chapter 5. Aside from isosurface visualization, meshless hierarchical representations for volumetric data have been developed and are described in Chapter 6. A description for how future work can build upon the concepts outlined in this exposition is provided in Chapter 7. Though not a major focus of this project, scattered data approximation remains a central component of visualization, so the basic approximation method used in this work is presented in Section 2.1.1.

Chapter 2

Related Work

Many existing meshless methods were developed in the context of scattered data analysis. The study of scattered data has primarily focused on the development of robust interpolation and approximation methods for reconstructing a continuous function over the data points. A related topic is point-based computer graphics, which has matured into an active area of research concerned with the use of point primitives containing little or no connectivity information to synthesize computer-generated images. Much of the work we have developed uses isosurface contouring methods and practices employed by the point-based computer graphics community to produce visualizations of volumetric data.

2.1 Scattered Data Interpolation

Numerous publications have been devoted to the study of scattered data interpolation and fitting. Traditional methods for scattered data approximation include Shepard's method, radial basis functions (RBFs), tessellation-based methods, tensor-product methods, and blended local methods. A complete review of all of these techniques is beyond the scope of this dissertation. Instead, we refer the reader to past and recent survey papers [4, 22, 23, 26] for more information. We note, however, that RBFs [7, 24, 31] and local methods have proven effective, particularly in reverse engi-

neering and surface reconstruction. There has been a renewed interest in the application of moving least-squares [1, 3, 42, 44] as a local method for fitting smooth approximations to point cloud data. Partition of unity blending functions [25, 53, 64, 76] have promoted the use of local approximations used to construct a globally continuous approximation.

2.1.1 Multiquadrics and Partition of Unity

Although the major focus of the work presented in this dissertation is on visualization algorithms and processing frameworks, it is important to describe the interpolation and data approximation techniques that are employed by these methods. We use a multiquadric radial basis function (RBF) originally proposed by Hardy [33]. In some cases, local RBF constructions are generated and blended together using partition-of-unity blending functions. In the work presented, many other choices for interpolation and approximation can generally be used without modification to the original algorithms.

Hardy's multiquadric method [33] offers high interpolation quality for a relatively low computational cost. Given a set of sample points S defined by a set of points $\{\mathbf{p}_i\},\ i = 1, 2, \ldots, |S|$, let f_i be the scalar value associated with $\mathbf{p}_i$. Let $\mathbf{x}$ be an arbitrary location at which we wish to evaluate the interpolant. We construct Hardy's interpolant

$$H(\mathbf{x}) = \sum_{i}^{|S|} \lambda_i \phi_i(\mathbf{x}),\ \text{where}\ \phi_i(\mathbf{x}) = \sqrt{|\mathbf{x} - \mathbf{p}_i|^2 + c^2}$$

by solving the system of equations

$$H(\mathbf{p}_i) = f_i,\ i = 1, 2, \ldots, |S|$$

for the coefficients λ_i, often called the *blending coefficients*. Here, c denotes Hardy's constant, which is set to 0.025 in our implementation, a value we have observed to be good in previous experimentation for data sets we have used [17]. Traditionally, S consists of all data points in a given scattered data set, forming a global interpolation scheme. When S is allowed to be a subset of data points in a given scattered data set, we obtain a local interpolant that must be blended with other neighboring local interpolants to form a global approximation.

The partition-of-unity method is frequently used to compose local approximations into a single continuous global approximation. Let $\{Q_i(\mathbf{x})\}$ be a set of local approximations each with a spherical domain of influence of radius R_i. Let $\{w_i(\mathbf{x})\}$ be a set of nonnegative functions defined over a bounded domain Ω. A set of partition of unity functions $\{\varphi_j(\mathbf{x})\}$ can be defined as

$$\varphi_j(\mathbf{x}) = \frac{w_j(\mathbf{x})}{\sum_k^n w_k(\mathbf{x})},$$

where n is the number of local approximations to blend whose spherical domains contain the point $\mathbf{x}$. The union of these n domains form Ω. Thus, the global approximation $F(\mathbf{x})$ takes the form

$$F(\mathbf{x}) = \sum_j^n \varphi_j(\mathbf{x}) Q_j(\mathbf{x}).$$

For partition of unity blending functions, the inverse distance singular weights of Franke and Nielson [25]

$$w_j(\mathbf{x}) = \left[\frac{(R_j - |\mathbf{x}-\mathbf{c}_j|)_+}{R_j|\mathbf{x}-\mathbf{c}_j|}\right]^2, \text{ where } (a)_+ = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases},$$

are used where the radius R_j and the center point $\mathbf{c}_j$ define the sphere of influence of the j-th local approximation that contributes to the interpolated value at $\mathbf{x}$.

2.2 Point-Based Computer Graphics

Researchers have investigated the use of point primitives as a rendering and representation alternative, and this topic has received much attention in recent years. The use of points as a rendering primitive can be traced back to 1985, when Levoy and Whitted [45] described its advantages and limitations. Grossman and Dally [29] revisited the notion of using points more than ten years later. The use of points to represent a surface was also promoted by Rusinkiewicz and Levoy [66], who proposed a hierarchical surface splatting technique for rendering surfaces of large complexity. Pfister et al. [59] proposed the use of *surfels* for the representation and rendering of surfaces. The surface splatting algorithm was later formalized and improved using *elliptical weighted averaging* (EWA) to support texturing, hidden surface removal, edge anti-aliasing, and transparency [81]. Ren et al. [63] developed a hardware-accelerated approach based on an object space formulation of the EWA surface splatting algorithm. (We refer the reader to [62] for an overview of splatting theory

and implementation issues.) Besides surface splatting, many other techniques were developed using points to render surfaces. Kalaiah and Varshney [40] developed an approach to rendering a surface through the use of *differential points*. Alexa et al. [1] used a point set and *moving least-squares* techniques to define a surface that can be down-sampled and up-sampled as needed to meet rendering and modeling requirements. Fleishman et al. [21] extended this point-set-surface approach using multiresolution methods. Botsch et al. [5] use octrees and a point-based hierarchical encoding scheme to compress large models consisting of point-sampled geometry. Pauly et al. [57] use point primitives and moving least-squares methods to interactively create and edit point-sampled geometry.

2.3 Isosurface Visualization

Isosurface visualization is an extremely valuable method for the exploration of volumetric scalar fields. An isosurface is an implicit surface defined as the intersection between a trivariate scalar function and a threshold value, often called the *isovalue*. Isosurfaces can be directly rendered but are frequently approximated, stored and rendered as polygonal meshes, usually triangle meshes. A large number of publications propose various methods for optimizing the extraction and rendering phases of isosurface visualization. Isosurfacing algorithms can be classified as either output-sensitive or input-sensitive.

Output-sensitive approaches focus on the resulting image and therefore attempt to perform computation mainly in regions that contribute substantially to the final image. At approximately the same time when Lorensen and Cline published their well-known Marching Cubes (MC) paper [50], they developed a lesser known method called Dividing Cubes (DC) [15]. Based on viewing parameters, the DC algorithm renders grid cells as points after iteratively or recursively subdividing cells to a pixel or sub-pixel level in screen space. Later, ray tracing was employed to render isosurfaces of large data sets interactively [54]. This method is attractive since it is relatively insensitive to input data size and thus scales well. Output-sensitive approaches are attractive in general as no intermediate form of the isosurface needs to be stored explicitly, which greatly decreases storage require-

ments. One drawback of output-sensitive approaches is that each time a new view is specified the isosurface extraction process must be repeated. For interactive applications, where viewing parameters are being changed frequently, such methods perform a relatively large number of computations. Output-sensitive approaches often offer excellent image quality, but frequently no geometric representation of the isosurface is generated, making them undesirable for use in geometric modeling applications, for example.

Input-sensitive approaches in general generate geometry approximating the isosurface. Most methods are based on the MC method and use triangles to approximate an isosurface [8, 9, 11, 32, 38, 41, 49]. The use of *interval trees* [11] and the *span space* [49] domain decomposition can greatly decrease the amount of work necessary to identify cells intersected by an isosurface (also called *active cells*), a major bottleneck in the extraction process. One advantage of generating geometry is that extraction need not be performed for each view. However, storing geometry becomes a burden as data resolution increases. *Dual contouring* methods were introduced to preserve sharp features and to alleviate storage requirements by reducing triangle count [38, 41]. Such methods produce high-quality polygonal models, but are not ideal for interactive visualization of large data due to the added computation and intermediate data storage requirements.

Chapter 3

Point-Based Isosurface Extraction

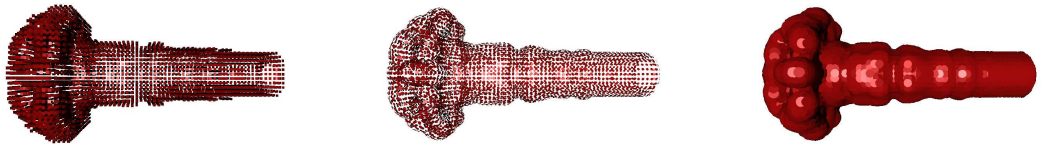


Figure 3.1: Iso-splatting applied to a fuel injection simulation data set. *Left*: point samples generated near the isosurface; *middle*: points projected onto the isosurface; *right*: resulting point set rendered using surface splatting.

3.1 Introduction

A large fraction of isosurface visualization research has followed one methodology: a user specifies an isovalue of interest, portions of the domain intersected by the isosurface are determined, geometric representations are computed, and geometric primitives are rendered. The process is performed for each new isovalue specified by the user.

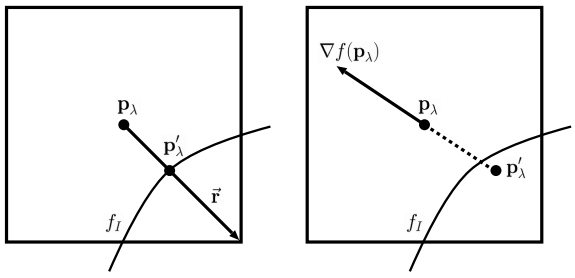
This basic methodology is quite straightforward, but it is also wasteful. A relatively large amount of computation must be performed to produce geometry specific to each isovalue. One set of geometric primitives created by this method can only be used to represent one specific isosurface. The situation is made worse when considering the cost involved in storing polygonal meshes.

Iso-splatting provides an alternative to this paradigm by splitting the work of extraction into two independent steps. In the first step, discussed in Section 3.4, point samples are generated. In the second step, these point samples are projected onto the isosurface of interest, which we discuss in Section 3.3. The resulting point set is rendered using a surface splatting technique. Figure 3.1 illustrates the iso-splatting technique applied to a fuel injection data set. The crucial difference between this approach and "standard" polygon-based methods is that the point geometry can be used to represent a range of isosurfaces. Geometry must be slightly adjusted only to a relatively small degree to represent a slightly different isosurface. By virtue of this fact, we greatly decrease computation time. The use of points, in many cases, can also decrease storage requirements. Furthermore, this new isosurface visualization paradigm can be tailored to meet application-specific needs, which is discussed in Section 3.5. The discussion centers around the use of regular grid data and is later generalized for arbitrary meshes.

3.2 Previous Work

Many point-based techniques [1, 5, 40, 59, 63, 66, 81] deal primarily with static surface models. Isosurfaces can exhibit vast geometric and topological changes when changing the isovalue of interest. Due to this "dynamic" nature, the idea of using points to represent isosurfaces seems daunting. However, points provide a large degree of flexibility and therefore are often more suitable than triangles when dealing with such changing surfaces.

The method we present focuses on improving interactivity through storage and computational efficiency via the use of point samples. Iso-splatting is an output-sensitive approach, as it generates the isosurface geometry once per isovalue and does not require repeated extraction when viewing parameters change. We show how large storage and computational savings can be achieved through the use of points as a basis for surface representation and rendering. We employ surface splatting techniques, although other point-based rendering techniques can be used without altering core aspects of our algorithm.



(a) (b)

Figure 3.2: Projection of sample point $\mathbf{p}_\lambda$ onto isosurface f_I, producing point $\mathbf{p}'_\lambda$ using (a) exact projection and (b) approximate projection.

3.3 Projection

Given the trilinear function

$$f(x,y,z) = \sum_{i,j,k=0}^{1} \rho_{ijk} x^i y^j z^k ,$$

where ρ_{ijk} denote the eight polynomial coefficients associated with one trilinearly interpolated cell, we are interested in the isosurface defined by the isovalue

$$f_I = \sum_{i,j,k=0}^{1} \rho_{ijk} x^i y^j z^k. \tag{3.1}$$

We project a sample point $\mathbf{p}_\lambda$, which is inside an active cell, to obtain $\mathbf{p}'_\lambda$, a point on or near the isosurface.

3.3.1 Exact Projection

Given a ray in parametric line equation form

$$\vec{\mathbf{r}}(t) = \mathbf{p} + t\vec{\mathbf{d}},$$

where $\mathbf{p}$ is a point on the ray and $\vec{\mathbf{d}}$ is an associated direction vector, the exact intersection between $\vec{\mathbf{r}}$ and the isosurface defined implicitly by Equation (3.1) can be determined by solving

$$\begin{aligned} f_I &= f(\mathbf{p} + t\vec{\mathbf{d}}) \\ &= \sum_{i,j,k=0}^{1} \rho_{ijk} (\mathbf{p}_x + t\mathbf{d}_x)^i (\mathbf{p}_y + t\mathbf{d}_y)^j (\mathbf{p}_z + t\mathbf{d}_z)^k \end{aligned} \tag{3.2}$$

for the parameter t. Once t is known, the sample on the exact isosurface is $\mathbf{p}'_\lambda = \mathbf{p} + t\vec{\mathbf{d}}$. For an arbitrary ray direction, Equation (3.2) leads to a cubic equation in t of the general form

$$At^3 + Bt^2 + Ct + D \quad = \quad 0\,. \tag{3.3}$$

Cardan's solution [51] can be used to determine the roots of Equation (3.3). (For details on implementing ray-isosurface intersections, we refer the reader to [54].)

The key to computing this intersection is defining the ray $\vec{\mathbf{r}}$. From the sample point $\mathbf{p}_\lambda$, rays can be shot to a subset of the eight corners of the cell, using the MC case table to determine which corners should be considered. One can also use the gradient of the scalar function computed at $\mathbf{p}_\lambda$ as a direction vector for $\vec{\mathbf{r}}$. However, a ray defined in such a way may not intersect the isosurface inside the cell. Figure 3.2 (a) illustrates exact projection.

This approach produces points that lie on the isosurface but at a high computational cost. Roots of a cubic polynomial must be determined in order to obtain ray intersections, thereby slowing down the projection phase of this method. An approximation can be performed to alleviate this computational burden.

3.3.2 Approximate Projection

We use one iteration of the Newton-Raphson [61] root-finding method to compute an approximation of $\mathbf{p}'_\lambda$. First, we describe how this formula is used to find an approximate isopoint of a univariate scalar function and then describe an extension to find a point approximately near the isosurface of a trivariate scalar function.

The Newton-Raphson method essentially uses the first two terms of the Taylor series expansion of a function $h(x)$ near a root. Given a univariate function $f(x)$ and an isovalue f_I, we seek the roots of

$$h(x) = f(x) - f_I\,.$$

The Taylor series of $h(x)$ at a point $x_0 + \varepsilon$ is defined as

$$h(x_0 + \varepsilon) = h(x_0) + h'(x_0)\varepsilon + \frac{h''(x_0)}{2}\varepsilon^2 + \ldots\,.$$

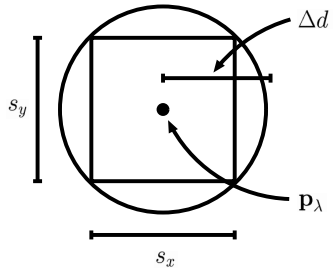


Figure 3.3: Displacement criterion. The square symbolizes the cell, the black dot the sample location $\mathbf{p}_\lambda$. The sample point $\mathbf{p}_\lambda$ may be projected onto an arbitrary isosurface that intersects the cell as long as the projection does not exceed the displacement boundary, defined by the sphere of radius Δd, which is determined by cell dimensions s_x and s_y.

The first iteration of the Newton-Raphson method considers only the first-order terms and solves

$$h(x_0+\varepsilon) = h(x_0) + h'(x_0)\varepsilon = 0\ . \tag{3.4}$$

If we consider $x_0+\varepsilon$ to be a line parameterized by t, where $\varepsilon = h'(x_0)t$, we obtain

$$h\big(x_0+h'(x_0)t\big) = h(x_0) + h'(x_0)h'(x_0)t = 0\ ,$$

which, when we solve for t, results in

$$t = \frac{-h(x_0)}{h'(x_0)h'(x_0)}\ .$$

Since $h(x_0) = f(x_0) - f_I$ and $h'(x) = f'(x)$ we obtain

$$t = \frac{f_I - f(x_0)}{f'(x_0)f'(x_0)}\ .$$

We now consider Equation (3.4) for a trivariate function $h(\mathbf{x})$ at a point $\mathbf{x}_0+\vec{\varepsilon}$, i.e.,

$$h(\mathbf{x}_0+\vec{\varepsilon}) = h(\mathbf{x}_0) + \nabla h(\mathbf{x}_0)\cdot\vec{\varepsilon} = 0\ .$$

If we consider $\mathbf{x}_0+\vec{\varepsilon}$ to be a line parameterized by t, where $\vec{\varepsilon} = \nabla h(\mathbf{x}_0)t$, we obtain

$$h\big(\mathbf{x}_0+\nabla h(\mathbf{x}_0)t\big) = h(\mathbf{x}_0) + \nabla h(\mathbf{x}_0)\cdot\nabla h(\mathbf{x}_0)t = 0\ ,$$

which leads to

$$t = \frac{-h(\mathbf{x}_0)}{\nabla h(\mathbf{x}_0)\cdot\nabla h(\mathbf{x}_0)}\ .$$

By substituting $h(\mathbf{x}_0) = f(\mathbf{x}_0) - f_I$ and $\nabla h(\mathbf{x}_0) = \nabla f(\mathbf{x}_0)$, one obtains

$$t = \frac{f_I - f(\mathbf{x}_0)}{\nabla f(\mathbf{x}_0) \cdot \nabla f(\mathbf{x}_0)} . \tag{3.5}$$

By setting $\mathbf{x}_0 = \mathbf{p}_\lambda$, we can evaluate $f(\mathbf{p}_\lambda)$ and $\nabla f(\mathbf{p}_\lambda)$ efficiently at the same time using trilinear interpolation. By computing t according to Equation (3.5), the approximation of $\mathbf{p}'_\lambda$ is given as

$$\mathbf{p}'_\lambda = \mathbf{p}_\lambda + \nabla f(\mathbf{p}_\lambda) t .$$

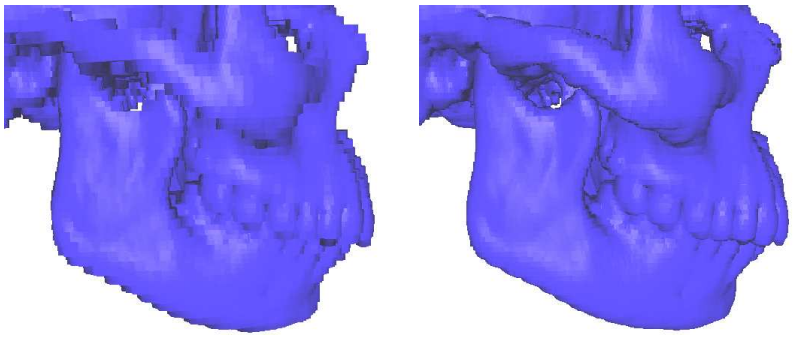
Figure 3.2 (b) illustrates this approximate projection.

The developers of *Kizamu* [58] used a similar approach to project arbitrary point samples onto the zero-set of an adaptive distance field. Their technique was meant as a method to preview a surface defined by a volumetric distance field, while iso-splatting is a method designed to visualize isosurfaces of arbitrary scalar field data.

The Newton-Raphson method converges quadratically, provided that the initial guess $\mathbf{x}_0$ is sufficiently close to a root. This method may result in roots far away from the "desired" root when $\mathbf{x}_0$ is near a local maximum or minimum. For this reason, we discard "far-away" solutions when encountered using a displacement criterion. After a single iteration of the Newton-Raphson procedure, if the point is "too far" from the initial guess, we do not consider the point. A displacement threshold Δd is computed based on the size of the cell. Given cell edge lengths s_x, s_y, and s_z, we define the maximum displacement threshold as $\Delta d = \frac{1}{2}\sqrt{{s_x}^2 + {s_y}^2 + {s_z}^2}$. Geometrically speaking, when the sample point $\mathbf{p}_\lambda$ is the center of the cell, this displacement criterion is the same as restricting the location of $\mathbf{p}'_\lambda$ to reside inside the sphere of radius Δd with center $\mathbf{p}_\lambda$, see Figure 3.3. Figure 3.4 illustrates the difference between unprojected and projected point samples.

3.4 Sampling

Often, the contribution to the isosurface of a given cell can be approximated cheaply without sacrificing overall rendering quality. In a standard triangle-based isosurfacing method, active cells are determined. For these cells, edge intersection points are computed, triangulated and added to a



(a) Unprojected (b) Projected

Figure 3.4: Impact of point sample projection. Shown is an isosurface of a skull rendered with quadrilateral surface splats. (a) Samples are grid-aligned when unprojected. (b) Same iso-splatted surface using projection.

polygonal model. In iso-splatting, instead of generating a set of triangles inside a cell, we generate a point sample. Although a given cell may contain several disjoint components of the same isosurface, we have found that many high-resolution data sets only rarely exhibit this phenomenon. In fact, restricting the amount of geometry to one point per cell is a common technique used in large model simplification [46, 65].

A surface rendering algorithm requires certain information necessary to synthesize an image. For surface splatting, a fairly regular sampling of the surface is required in order to produce a proper rendering. Each sample should be characterized by location, normal, and a local variance matrix, which indicates roughly the average distance to neighboring samples. Uniform rectilinear volume data implicitly provides locations of volume samples. For this type of data, gradients are often used for computing local illumination properties. The variance matrix can be derived by knowing the grid spacing. We can therefore derive all the elements required to define a surface splatting primitive.

Point samples may not lie on the desired isosurface and must be projected onto that surface. If an exact projection is used, the cell corner values are necessary to compute the projection of this sample point onto the isosurface. If our approximation is used, a function value and gradient must be computed for this sample point. The gradient can be reused for shading purposes in the rendering phase.

In general, iso-splatting requires that each point sample consist of sample location, function value,

gradient, and a local variance matrix. For gridded data sets, several methods exist to obtain location, function value, and gradient. The local variance matrix can be derived by analyzing the spatial distribution of the volume samples or from the size and shape of grid cells. Thus, iso-splatting is applicable, in principle, to any type of grid.

3.5 Implementation Issues

The use of the point as a representation and rendering primitive for isosurfaces results in many desirable features. One primary advantage is its flexibility at the application level. Computations necessary for iso-splatting can be performed in different stages of the visualization pipeline, providing more flexibility than many triangle-based approaches.

There are three stages of any geometry-based isosurfacing pipeline:

1. **Preprocessing**

   Data reorganization or data structure initialization is performed to improve the extraction and/or rendering stages.

2. **Extraction**

   Geometry approximating an isosurface is computed.

3. **Rendering**

   The isosurface geometry is rendered.

Two phases characterize the iso-splatting algorithm:

1. **Point generation**

   Sample points are selected, and key information for the projection and rendering phases are computed.

2. **Point projection**

   Sample points are projected onto the isosurface.

Stage	(A)	(B)	(C)	(D)
Preprocessing	G	G		
Extraction	P		G,P	G
Rendering		P		P

G = point generation P = point projection

Table 3.1: Iso-splatting paradigms.

In iso-splatting, point sample generation and projection can be performed in different stages of the isosurface visualization pipeline to satisfy application-specific needs. We recognize four basic application paradigms, shown in the columns of Table 3.1.

The paradigms described in Table 3.1 illustrate the quintessential trade-off between computational efficiency and memory consumption. Paradigms (A) and (B) are useful when a high level of interactivity is desired but can be storage-intensive, since point samples are pre-generated and must be stored for later use. Using the approximate projection scheme, location, function value, gradient, and the value interval spanned by the cell must be stored for each point sample. Extraction in this case consists primarily of collecting point samples in active cells. Several isosurface extraction techniques exist to perform active cell lookups efficiently out-of-core [8, 9, 32]. Performing point sample generation in the preprocessing stage can be thought of as a "partial evaluation" of the isosurface. Paradigms (C) and (D) require more computation but provide better storage efficiency than paradigms (A) and (B). These two paradigms require only the original data set, from which we can compute the information needed for projection and rendering.

Programmable hardware on modern graphics cards makes the efficient computation of point projections feasible in the rendering phase, as in paradigms (B) and (D). The approximate projection procedure consists of a single floating-point division and a vector dot product, both operations supported by existing graphics hardware. Point location and gradient will already be transferred to the hardware for rendering the surface splat. The only additional piece of information that must be sent is the function value approximated for the point and the isovalue, which can be sent once per frame. The displacement criterion can be implemented with the use of texture lookups. However, the next generation of graphics processing units (GPUs) will also support branches (i.e., conditional

statements), thereby allowing a more straightforward implementation of the displacement criterion. Discarded points can be rendered outside of the viewing frustum.

We believe that iso-splatting leads to computational efficiency, while using a geometric primitive that is highly storage-efficient. In many cases, iso-splatting offers improved extraction efficiency over standard triangle-based schemes. A standard MC implementation performs several edge intersections and perhaps computes a gradient for each point for smooth shading. In the method of Ju et al. [38], a quadratic error function must be minimized in addition to computing these edge intersections in order to place a single point inside the cell. In iso-splatting, a single function value and gradient computation, which can be computed simultaneously, is followed by one iteration of the Newton-Raphson procedure. In terms of storing geometric primitives to represent the isosurface, consider n cells that contribute to an isosurface. Let us assume that each vertex has an associated normal vector used for shading purposes. Suppose a triangle-based isosurfacing algorithm generates one triangle per cell, and, moreover, outputs the entire mesh as one triangle strip, offering the most compact storage representation. Such a surface requires $n+2$ vertices of storage. Iso-splatting generates one point per cell, producing at most n vertices. While the benefit is small, one must consider that many current triangle-based isosurfacing methods generate "triangle-soup" representations efficiently, which requires $3n$ vertices assuming one triangle per cell. (Standard MC implementations generate on average more than one triangle per cell.) Producing more memory-efficient triangle representations—such as a triangle strip in the ideal case—would most likely require more computation and thus slow down extraction time.

3.6 Results

We have performed tests on a Pentium4 PC with 2 GB of main memory and an nVidia GeForce4 Ti video card with 128 MB of memory on a motherboard supporting a $4\times$ accelerated graphics port (AGP). We implemented paradigms (A) and (C), see Table 3.1. QSplat-style surface splatting [66, 67] was chosen due to its rendering efficiency. However, EWA surface splatting [81, 63] could easily be incorporated into the system without affecting our algorithm.

We have used quadrilateral and elliptical surface splatting kernels, which offer different degrees of quality and efficiency. Bandwidth to the graphics card is currently a bottleneck in geometry-intensive applications. Quadrilateral splats can be more efficient, since they can often be implemented by passing a single vertex to the graphics hardware, which usually rasterizes this vertex as a quadrilateral. Hardware extensions exist to automatically scale the size of this rasterized quadrilateral based on viewing parameters. Elliptical splats offer better image quality, but, since they are often rendered using alpha-textured polygons, more geometry must be sent to the graphics hardware, thereby reducing rendering efficiency. When better support for surface splatting in graphics hardware exists, this need to balance the trade-off between quality and efficiency may disappear.

To quantitatively compare a triangle-based approach with iso-splatting, we extracted 16 isosurfaces 10 times for four data sets, measuring average extraction time and memory usage for a standard MC implementation and a paradigm-(C) iso-splatting implementation. In addition, we measured average framerate considering 36 frames for each isosurface, each time rotating the model by 10 degrees about the y-axis. For both implementations, interval trees were used to accelerate active cell lookup. We chose a standard MC implementation over a dual-contouring implementation, as it offers the most competitive extraction times. Figure 3.5 summarizes extraction time, memory usage, and framerate information. For our framerate calculations, we used quadrilateral kernels due to their rendering efficiency. In the graphs, two curves of the same color represent measurements collected for the same data set. Of these two curves, the dotted curve corresponds to measurements made for the MC implementation, and the solid curve corresponds to measurements made for the iso-splatting implementation. The left-most graph in Figure 3.5 indicates that iso-splatting generally performs extraction in approximately half the amount of time used by a standard MC implementation, even when both point generation and projection are performed in the extraction stage. The fact that fewer primitives are generated and that these primitives are more storage-efficient is shown in the middle graph. This storage efficiency accounts for framerates that can exceed those achieved by rendering an isosurface represented by triangles, as seen in the right-most graph of Figure 3.5. This graph indicates that iso-splatting generally provides quadruple the framerates using quadrilateral splats when compared with a standard MC implementation.

Figure 3.6 provides side-by-side comparisons of an implementation of MC and iso-splatting. Some

Data set	Dimensions
Fuel injection	$64 \times 64 \times 64$
Skull	$68 \times 256 \times 256$
Bucky ball	$128 \times 128 \times 128$
Aneurysm	$256 \times 256 \times 256$
Primate lung	$266 \times 512 \times 512$
Stanford bunny CT	$361 \times 512 \times 512$
Lobster	$80 \times 324 \times 301$
Engine	$128 \times 256 \times 256$
Boston teapot	$178 \times 256 \times 256$
Argon bubble	$640 \times 256 \times 256$

Table 3.2: Data set sizes.

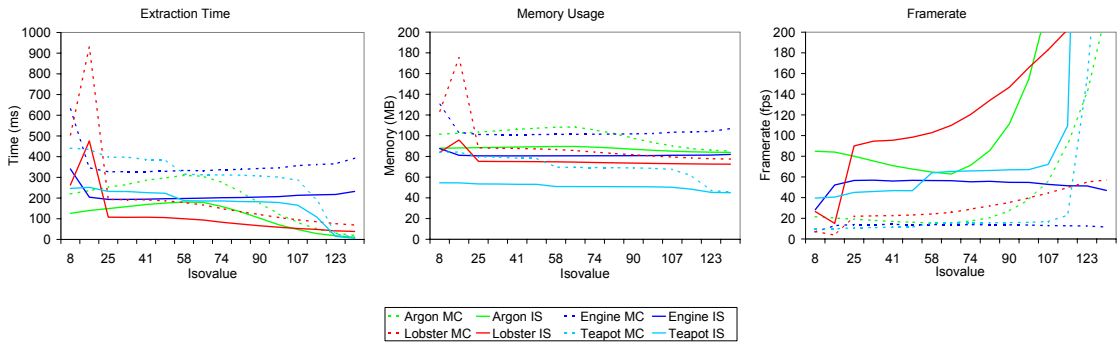
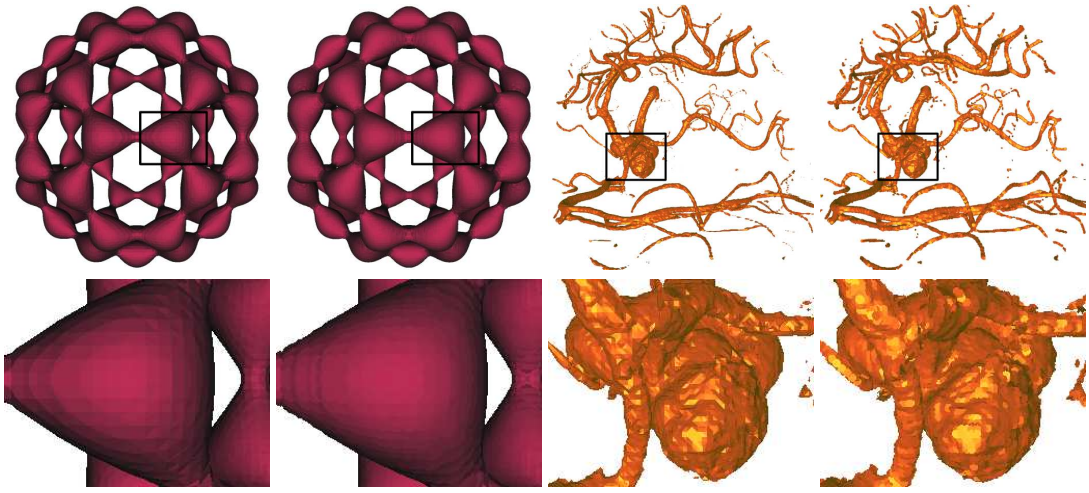


Figure 3.5: Extraction time (left), memory usage (middle), and framerate (right) comparisons. A standard Marching Cubes (MC) implementation and a paradigm-(C) iso-splatting (IS) implementation are compared. (See also Section 3.5 and Table 3.1.)

detail in the aneurysm data set is lost due to the fine structure of the arteries, but the overall essence of the isosurface is not lost. Details could be recovered by using an adaptive sampling scheme. Figure 3.7 shows images from an out-of-core implementation of iso-splatting using paradigm (A). We used a Morton-order indexing scheme to perform lookups efficiently on disk. The dimensions of all data sets used in our experiments are provided in Table 3.2. All data sets consist of values in the range $[0,255]$.

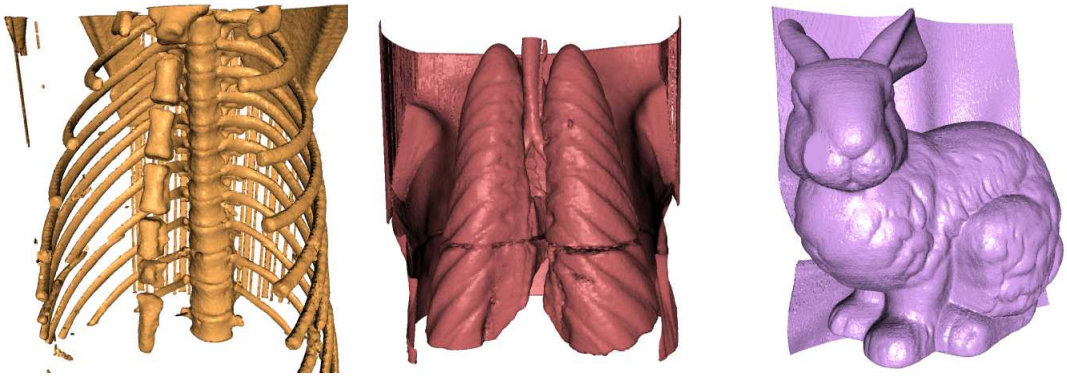
3.7 Summary

We have presented a new algorithm called iso-splatting for isosurface visualization. Improved extraction, storage, and rendering efficiency can be obtained through the use of points. The point



Marching Cubes Iso-splatting Marching Cubes Iso-splatting

Figure 3.6: Comparisons of Marching Cubes and iso-splatting rendering results of a Buckminsterfullerene ("bucky ball") data set and an aneurysm data set.



0.8M splats extracted in 0.38 s

2.2M splats extracted in 0.52 s

2.1M splats extracted in 0.52 s

Figure 3.7: Images generated with an out-of-core implementation of iso-splatting applied to a computed tomography (CT) scan of a primate lung (left and middle images) and a CT scan of the Stanford bunny (right image).

samples are, in general, more cheaply stored than triangles, and they require little connectivity information, opening up possibilities for parallelism. Our results indicate that iso-splatting performs well in out-of-core settings. Since computation can be divided in various ways, implementations can be tailored relatively easily to meet resource limitations or to take advantage of emerging GPU technology. These characteristics make iso-splatting suitable for large, high-resolution data sets, when a geometry-based isosurface visualization algorithm is desirable. We point out that the point set generated by our method is useful not only for rendering purposes. Many techniques, such as the point-set-surface method [1, 21], have been developed to support geometric modeling using point sets. Iso-splatting extends well to data represented by any type of mesh. An additional strength of iso-splatting is its simplicity and ease of implementation.

Chapter 4

Isosurfaces from Multiblock Data

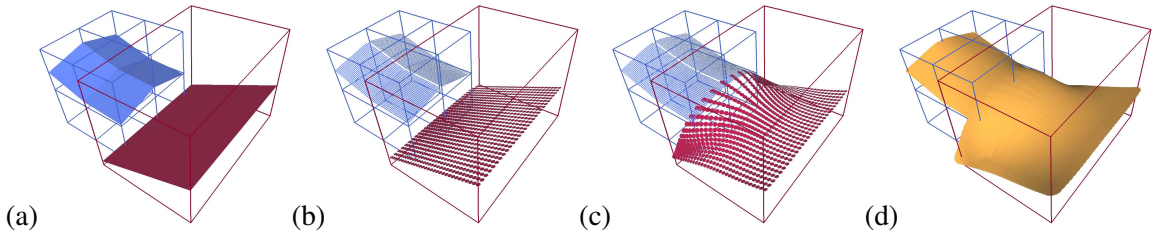


Figure 4.1: Illustration of our isosurface generation method. (a) Marching Cubes triangles are generated inside each cell. (b) The point samples are generated in each triangle and then (c) projected onto the isosurface. (d) The point set is rendered using a surface splatting algorithm.

4.1 Introduction

Much of the research in isosurface generation has focused on regular grid data sets, but relatively little research has addressed isosurfacing data represented by multiple grids of arbitrary layout and orientation. We refer to such data as *multiblock* data sets. Multiblock data representations arise in many important applications. In particular, flow simulations used in computational fluid dynamics (CFD) very frequently use multiple grids (see Figure 4.2). Moreover, these grids often overlap one another, a situation not addressed by standard isosurfacing techniques. Multiblock data sets appear in the form of hierarchical data representations as well, such as octrees [80] or adaptive mesh refinement (AMR) [79] grids. Volumetric data represented by such adaptive grid structures

can be thought of as a specific class of multiblock data sets.

A major issue encountered when contouring multiblock data is the appearance of undesirable cracks or self-intersections, which result from a naive application of methods designed for single grid data sets. There are several reasons why it is important to resolve these discontinuities in the surface. Generally, the original object being sampled is continuous, and the cracks appear only as an artifact of a poor reconstruction. Moreover, many applications require watertight surface models. This issue has been treated to some extent in the study of adaptive grids for representing volume data, but such techniques frequently do not generalize to arbitrary multiblock data sets. The fundamental problem is two-fold. First, interpolation across the boundaries of these grids is often not well-defined or not easily dealt with. For instance, it is not well-defined how a trilinear interpolant defined in one grid should incorporate data samples from another grid overlapping it (see Figure 4.3(d)). We refer to this as the *interpolation problem*. Second, it is not clear how to extract continuous isosurface geometry across grid boundaries. In particular, a direct application of the Marching Cubes algorithm [50] to each separate grid will often produce a triangle-based surface containing triangles that do not meet up (forming cracks) and triangles that intersect one another. We refer to this as the *isosurface generation problem*.

Meshless techniques for scientific computation have been introduced to alleviate mesh generation requirements by supplying robust interpolation methods that do not rely heavily on mesh connectivity. We use principles of meshless methods to address the interpolation problem by defining a continuous interpolation across grid boundaries. This continuous interpolation is used in conjunction with a flexible point-based surface sampling approach to solve the isosurface generation problem.

Our contribution is a general approach for the construction of isosurfaces from arbitrary multiblock data through the use of meshless techniques. We define a continuous interpolation scheme based on a set of radial basis functions, discussed in Section 4.3. The isosurface generation problem is addressed by first computing the triangles of a Marching Cubes surface in each grid. Inside the generated triangles, points are sampled and then projected onto the isosurface of interest using our continuous interpolant and principles of the iso-splatting method [16]. Isosurface generation

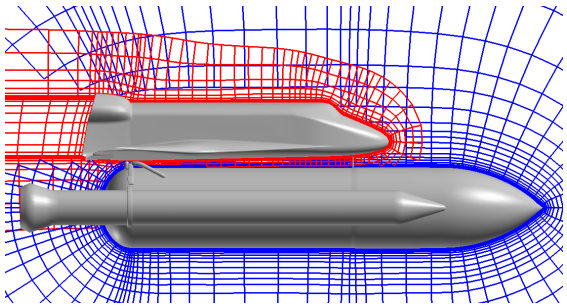


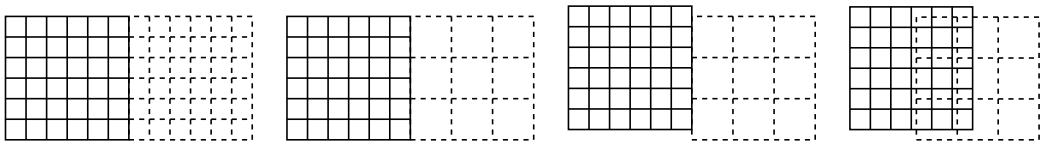
Figure 4.2: An example of a multiblock data representation used in the simulation of flow around a space shuttle launch vehicle. Notice that the red and blue grids overlap one another. (The data set is available online at http://www.nas.nasa.gov/Research/. Visualization courtesy of Edward A. Mayda of the Department of Mechanical and Aeronautical Engineering at UC Davis.)

is discussed in Section 4.4. What results is a point set representation suitable for rendering and geometry processing.

4.2 Previous Work

The Marching Cubes technique of Lorensen and Cline [50] generates triangles to approximate an isosurface of interest for regular grid data sets using a case-driven approach. The elegance and relative simplicity of the method make it a desirable technique to apply to data sets composed of hexahedral cells, such as multiblock data sets. However, few have successfully managed to extract continuous isocontours from arbitrary multiblock data. Although few have addressed this issue directly, several have treated a similar problem in attempting to reduce the amount of geometry generated by the standard Marching Cubes approach. Namely, the use of adaptive resolution volumetric representations [43, 70, 73, 80] often introduces discontinuities in the interpolation that ultimately result in surface cracks that must be patched. This area of research essentially addresses a specific instance of a broader class of multiblock isosurfacing problems. Solutions to the "crack problem" resolve discontinuities in situations where two grids *semi-conform* at the interface where the grids meet (see Figure 4.3(b)).

Several strategies exist to address the crack problem. Polygons residing in the plane of the crack can be computed to cover the hole [73]. Cracks can be patched by aligning vertices of triangles



(a) Conforming (b) Semi-conforming (c) Non-conforming (d) Overlapping

Figure 4.3: Grid layouts in multiblock data sets. Many techniques exist to generate isocontours from (a) conforming grids and (b) semi-conforming grids, where some sample points are shared among grids at the interface. Relatively few techniques exist to generate isocontours from (c) non-conforming grids or from (d) overlapping grids. In these environments, common methods for interpolation are not well defined, and it is unclear how to extract a continuous isosurface representation.

obtained from the high-resolution grid to the contour of the lower-resolution grid [70]. Coarse triangles can be replaced with an alternative triangulation that stitches the triangles from the two resolutions together [43, 80]. Another option is to avoid the generation of a volumetric mesh that would introduce a discontinuity in the volume [28, 79]. In this way, the interpolation and isosurface generation remain continuous.

These methods work well for conforming and semi-conforming grids, where all or some of the sample points along the interface between the grids are shared. These methods do not address problems with non-conforming or overlapping grids, sometimes called *overset grids* in computational mechanics. The fundamental issue here is that grid-based interpolants are not well defined in these domains, or, in the case of overlapping grids, it is not clear how to extract continuous geometry from this irregularly sampled subdomain. Our method specifically addresses non-conforming and overlapping grids and generalizes to conforming and semi-conforming grids without modification. Figure 4.3 shows various possible grid layouts for multiblock data.

We overcome the interpolation problem by disregarding the grid connectivity and defining a continuous RBF approximation over the data points. In order to generate geometry to represent the isosurface, we extend principles from the iso-splatting method [16] to obtain a set of oriented points. The resulting point set is rendered using a surface splatting algorithm [5, 57, 66, 81] and is suitable for further geometry processing. The simplicity and generality of our method make it an attractive solution for obtaining high-quality isosurfaces from arbitrary grid data.

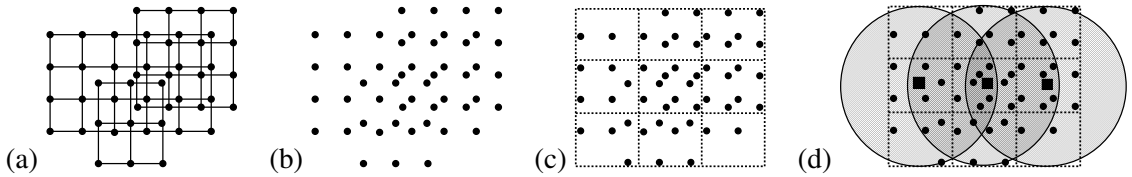


Figure 4.4: Local RBF Construction. (a) The input is multiblock data. (b) The grid connectivity is disregarded. (c) Data points (shown as black dots) are binned into RBF grid cells. (d) Local RBFs are computed with centers placed at the center of the cells. An RBF center is shown here as a square point, and its radius of influence is shown as a shaded circle. To avoid visual clutter, only the three middle RBFs are shown.

4.3 Approximation

Let $H_k(\mathbf{x})$ be a local multiquadric interpolant, as defined in Section 2.1.1. To define a local interpolation $H_k(\mathbf{x})$, we first disregard the connectivity of the grid and consider only the data points. These data points are then binned into a global regular grid, which we call the *RBF grid*, defined by the bounding box around the data points. We associate a local interpolant $H_k(\mathbf{x})$ with each cell center point $\mathbf{c}_k$ of this grid. We refer to this point as an *RBF center*. We define a radius of influence R_k around each RBF center to be

$$R_k = 2\sqrt{{s_x}^2 + {s_y}^2 + {s_z}^2},$$

where s_x, s_y, and s_z define the edge lengths of a single cell in the RBF grid. The coefficients of the local RBF associated with $\mathbf{c}_k$ are computed by considering all points within the sphere defined by the RBF center $\mathbf{c}_k$ and the radius of influence R_k. The points necessary to compute these parameters are obtained by checking each point in the neighboring 26 cells—the eight vertex neighbors, twelve edge neighbors, and six face neighbors—to see if the point is inside the sphere. If this region happens to be empty, we do not construct an RBF center. The RBF construction process is illustrated in Figure 4.4.

In creating the RBF grid, we attempt to make the cells as close to cubes in shape as possible. This

is achieved by computing the RBF grid's dimensions $n_x \times n_y \times n_z$ using

$$\begin{aligned} s &= \left(m\frac{w_x w_y w_z}{n}\right)^{1/3}, \\ n_x &= round\left(\frac{w_x}{s}\right), \\ n_y &= round\left(\frac{w_y}{s}\right), \\ n_z &= round\left(\frac{w_z}{s}\right), \end{aligned}$$

where n is the number of data points in the entire data set, (w_x, w_y, w_z) is the size of the bounding box enclosing all the data grids, and m is a user-defined parameter roughly specifying the number of data points to place in each cell. We refer to m as the *binning constant*. These equations are commonly used to create a grid for the accelerated ray tracing of complex scenes [72].

When evaluating the function at an arbitrary point $\mathbf{x}$, we determine in which RBF grid cell the point $\mathbf{x}$ resides to obtain the local RBF interpolant $H_k(\mathbf{x})$. We collect the 26 neighboring local RBF interpolants adjacent to this cell in the RBF grid. At the point $\mathbf{x}$, these 27 local interpolants are evaluated and blended together using the partition of unity functions described in Section 2.1.1 to obtain a final function value and gradient. If a cell does not have an associated local interpolant, we do not consider it for our calculation.

4.4 Isosurface Generation

To generate geometry to represent the isosurface, we first compute triangles inside each cell of the original multiblock grids according to the Marching Cubes algorithm. Inside these triangles, we compute a set of points at which we sample our volume for function value and gradient using our approximation scheme. We generate a uniform lattice of samples inside the triangle using barycentric coordinates and determine the number of samples to generate by specifying the number of samples to compute in the u- and v-directions. We do not sample along the edge of a given triangle to avoid redundant computation, since the triangle's edge neighbor will duplicate the same exact points. Each sample point, its function value, and associated gradient are used as input to a Newton-Raphson root-finding procedure to project the point onto the isosurface of interest. Newton-Raphson iteration finds approximate roots of a function by using function value and first derivative

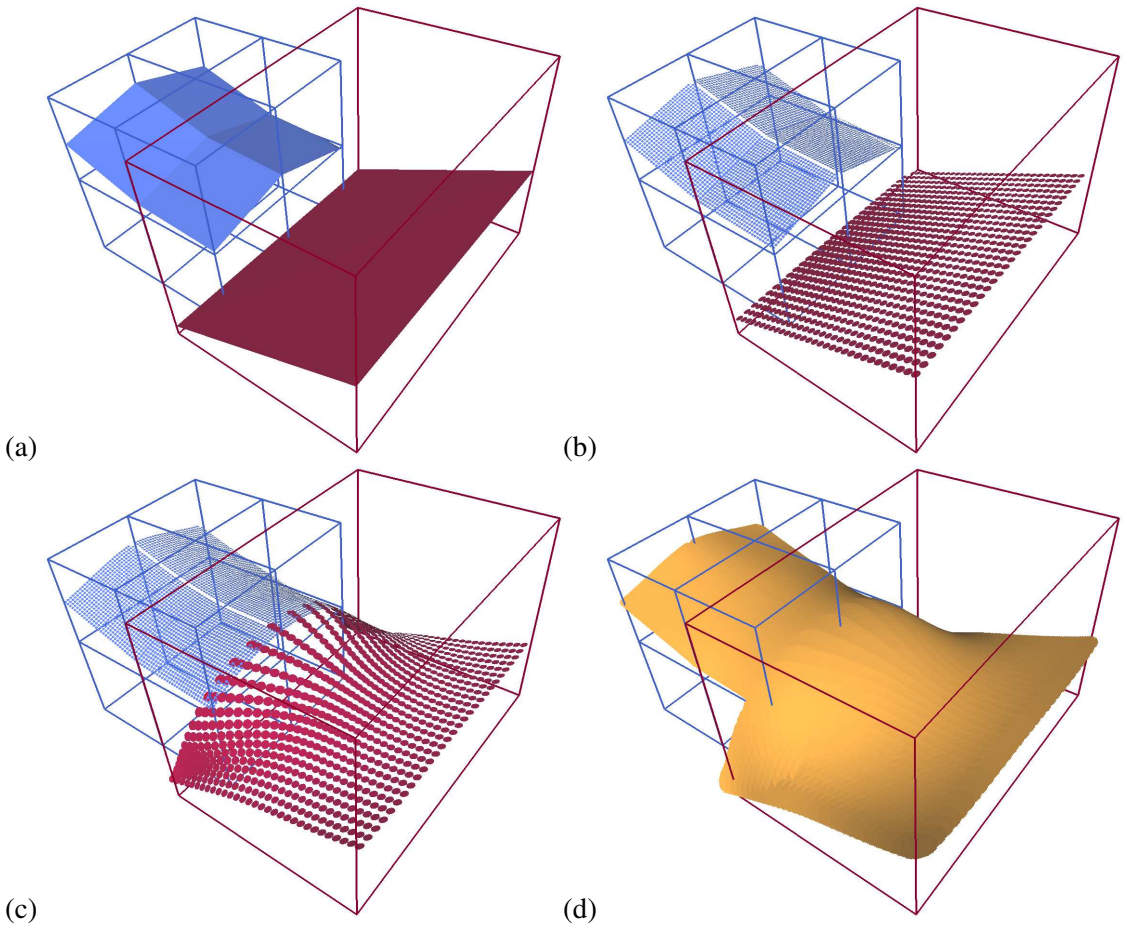


Figure 4.5: Illustration of our isosurface generation method. (a) Marching Cubes triangles are generated inside each cell. (b) The point samples are generated in each triangle and then (c) projected onto the isosurface. (d) The point set is rendered using a surface splatting algorithm.

information to move an initial guess closer to the actual solution. The Newton-Raphson procedure converges to the isosurface quadratically when given a reasonably good initial guess, which the triangles inside cells intersected by the isosurface provide. The result is a set of points and normals which collectively define the isosurface. Figure 4.5 illustrates our isosurface generation scheme.

For rendering and geometry processing, we define the radius of each point sample to be proportional to the area of the triangle in which it was sampled and the distance it traveled from the original point sample taken. Let A denote the area of the triangle, n be the number of samples taken uniformly in the u- and v-directions inside the triangle, l be the length of the diagonal of the cell in which the triangle exists, and d be the Euclidean distance between the original sample point and its final computed location. The number of points sampled inside a triangle is $\frac{n(n+1)}{2}$. The radius r of a point

sample is computed to be

$$r = 2\left(1+\frac{d}{l}\right)\sqrt{\frac{2A}{\pi n(n+1)}} \quad .$$

Thus we compute the size of each point to be large enough such that if no displacement of the points occur, the original triangle is rendered without holes. However, when the point is displaced from the original sample location, we scale the size of the point linearly with respect to the distance it travels from that point. The premise is that as a point moves further from its original sample location, the surface sampling becomes less dense, and thus the point must be made larger to compensate for gaps evolving between it and the neighboring points.

4.5 Results

To test the effectiveness of this approach, we used a set of synthetic data sets created by sampling available scalar data sets on multiple grids. The "bucky ball" data set was sampled using semi-conforming grids. The "fuel injection" and "neghip" data sets were sampled using multiple overlapping grids. Our test machine was a 2.8 GHz Pentium4 PC with 2 GB of memory. We fixed the multiquadric parameter to 0.025. The binning constant was set to five. The RBF construction computation is dominated by the linear system solver for computing the blending coefficients. The time complexity of this computation is $O(n^3)$, where n is the number of data points used to compute the local interpolant. Each local RBF used approximately 150-300 data points. Information about the data sets as well as the RBF construction performance are provided in Table 4.1.

Figure 4.6 demonstrates how our method can be used on semi-conforming multiblock data sets, such as those which result from octree subdivisions. Again, the cracks exhibited by a naive application of Marching Cubes (seen in the left and middle images) are not realized in the isosurface extracted using our approach (right-most image).

Figure 4.7 shows the neghip data set sampled by two overlapping grids. The rendering of the isosurfaces obtained from a naive application of Marching Cubes are color-coded to match the color of the grids. Cracks and self-intersections in the isosurface are observed, since the blue surface

Data Set	# of Grids	# of Data Points	# of RBF Centers	RBF Construction Time
bucky ball	2	21,114	4,096	1 min 42 s
fuel injection	5	2,688	496	5 s
neghip	2	16,000	3,060	50 s

Table 4.1: RBF construction statistics performed on a 2.8 GHz Pentium4 with 2 GB of memory.

intersects the pink one, and vice versa. This shows that the trilinear interpolant of one grid does not agree with the other grid as to the location of the isosurface. The bottom images of Figure 4.7 provide a side-by-side comparison of Marching Cubes versus our method.

Figure 4.8 illustrates our approach applied to the fuel injection data set consisting of several overlapping grids. The isosurfaces are color-coded to match the grid in which the geometry was sampled. Note that the Marching Cubes result (middle image) also contains several self-intersections and cracks. Again, this is an indication of a discontinuous interpolation scheme combined with an isosurface generation method that makes it difficult to extract a continuous surface. The same isosurface using our approach (right-most image) offers improved representation quality by using a continuous approximation scheme along with a flexible geometry generation method.

4.6 Discussion

One major advantage to sampling the isosurface using points is that the point samples are not adversely affected by the overlap of nearby samples. For instance, triangles generated in one grid may be extremely close to triangles generated in an overlapping grid. When we sample these triangles for points and project them onto the isosurface, it is likely that many points will end up overlapping one another significantly. While this usually causes problems when using other geometric primitives, such as triangles, point geometry is less sensitive to this redundancy. Since all of these points snap to the same surface, redundancy is not a problem. As long as there is consistency among the point samples as to where the isosurface exists, no cracks or self-intersections will appear. This consistency is provided by our continuous interpolation scheme. The right-most image of Figure 4.8 shows an example where significant point sample overlap occurs due to overlapping grids.

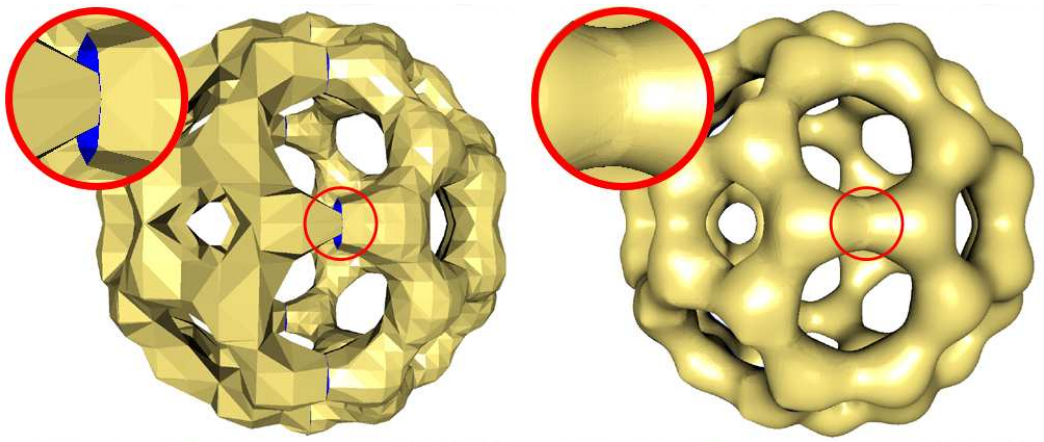


Figure 4.6: Isosurface visualization of a semi-conforming multiblock representation of the bucky ball data set. *Left*: Marching Cubes results in cracks at the coarse-fine boundary. *Right*: Our method applied to the same data set produces a continuous closed isosurface representation.

Another major advantage to sampling the isosurface using points is that we can adjust our sampling in a flexible manner to meet application-specific requirements. The surface onto which we wish to map points is implicitly known and can be sampled less densely, for instance, in areas of low curvature and more densely in areas of high curvature. Similarly, it is possible to steer the generation of point samples based on viewing parameters, as in the point-set-surface method by Alexa et al. [1], where a moving least-squares surface is dynamically sampled to meet screen space resolution. While this feature is not a major focus of our work, we feel that this added flexibility makes our approach valuable for a variety of application domains.

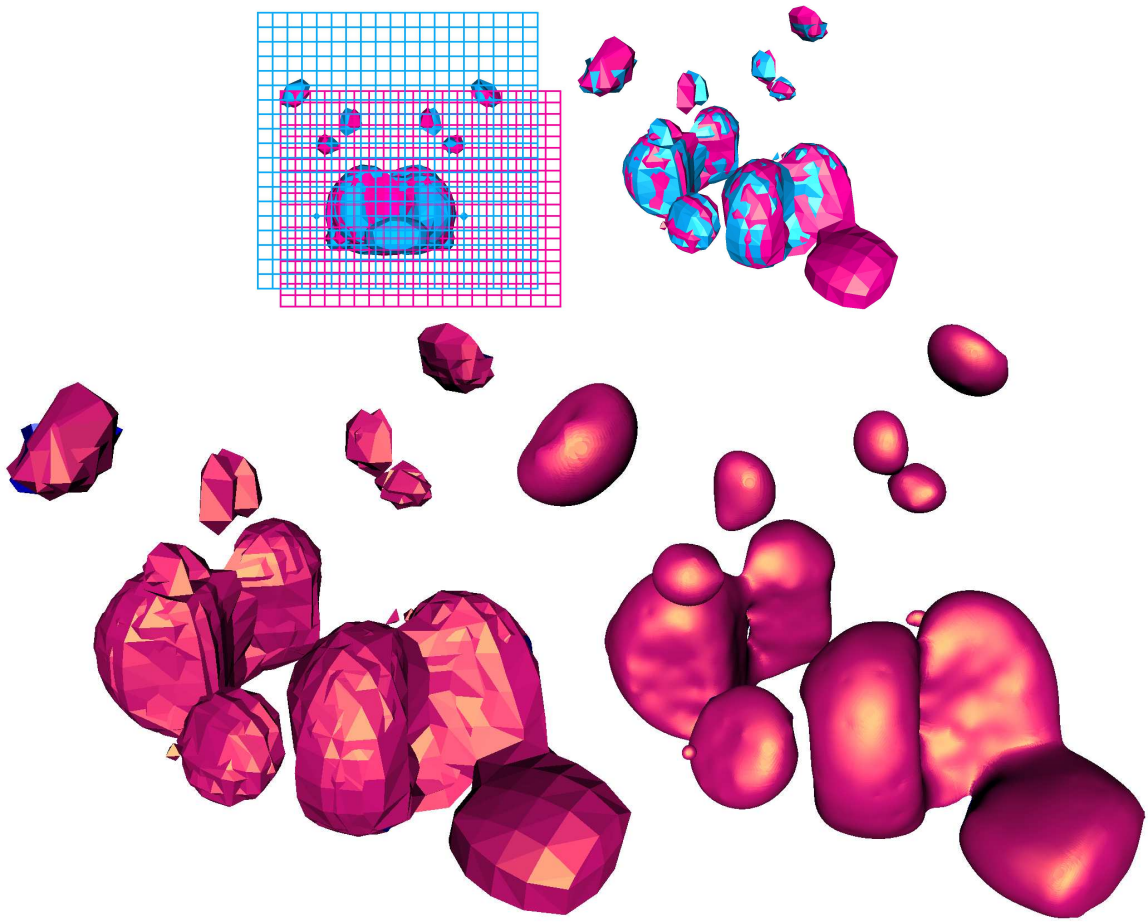


Figure 4.7: Isosurface visualization of a multiblock representation of the neghip data set. *Top-left*: Axis-aligned view showing two overlapping grids representing the data. *Top-right*: Marching Cubes triangles color-coded by grid. Notice the self-intersections and cracks in the isosurface. *Bottom*: Side-by-side comparison of Marching Cubes versus our approach.

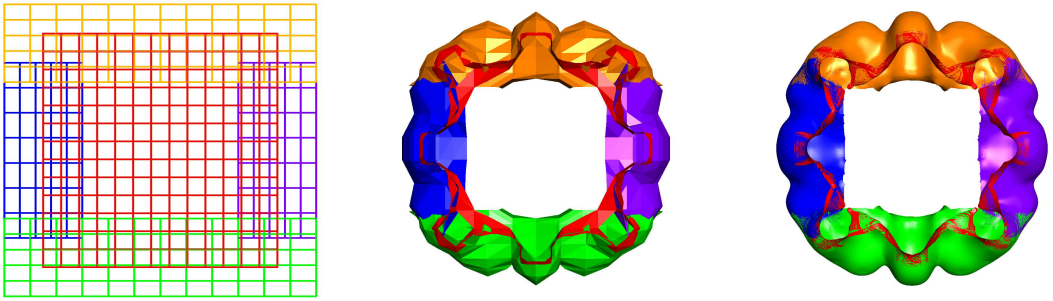


Figure 4.8: A fuel injection data set represented by five overlapping grids. *Left*: A visualization of the five grids. *Middle*: A naive application of Marching Cubes produces several cracks and self-intersections in regions of overlap. *Right*: Our method generates a single continuous surface. Overlapping points are not a problem, as they project to a globally defined isosurface.

Chapter 5

Isosurfaces from Scattered Data

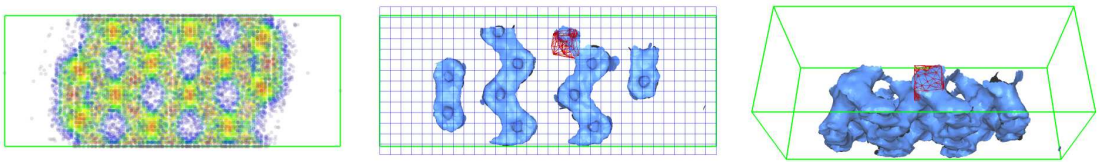


Figure 5.1: Demonstration of our volumetric scattered data sampling framework applied to generating an isosurface from a silicium data set. *Left*: Visualization of the point distribution. The axis-aligned bounding box of the data is shown in green. *Middle*: Axis-aligned view of an early stage of isosurface generation using a "marching stencil," shown in red. The binning grid is shown in blue. *Right*: Later stage of contouring.

5.1 Introduction

We build upon the devised in Sections 3 and 4 to contour scattered data sets. In the method described in Section 4, point geometry representing the isosurface is derived from an initial triangle representation extracted from the original cells of the data set, even in the presence of cell overlap. This process can be similarly performed for data defined by multiple tetrahedral meshes with the same insensitivity to cell overlap. Given a field described by a set of sample points scattered arbitrarily in 3D space, the same process can be used provided that a set of conforming or overlapping local triangulations can be defined over the points. Special care must be taken to ensure that the union of these triangulations cover the domain of the field. In this section, we describe how these

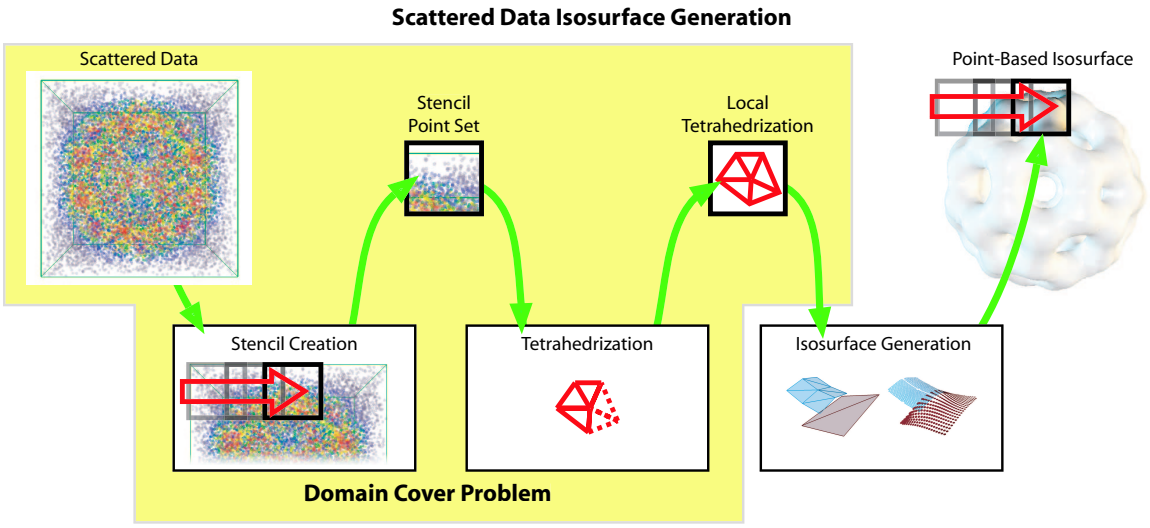


Figure 5.2: Overview of our algorithm. Given a scattered data set, our method constructs stencil point sets. A tetrahedrization is computed over each stencil point set. The local tetrahedrizations provide a domain cover. The portion highlighted in yellow (left) represents the portion that solves the domain cover problem. Isosurface generation is accomplished using point-based techniques.

local triangulations are constructed.

The major contribution of this chapter is a new localized approach to visualize scattered data that uses direct sampling. Specifically, we extract isosurface geometry by directly point-sampling the scalar field. Our algorithm has several desirable features. First, the locality of our method reduces the amount of memory necessary during processing and makes a parallel implementation possible. Second, since no resampling step is employed in the extraction of isosurface geometry, rendering artifacts are avoided. Finally, the sampling and interpolation procedures are decoupled, allowing application-specific interpolations to be incorporated without modifying the sampling procedure. This chapter introduces a novel way of approaching the visualization of scattered data and represents a significant step to building effective visualization tools for scattered data sets. Figure 5.2 illustrates our method for isosurface extraction from scattered data, highlighting our solution to the domain cover problem.

5.2 Previous Work

Meshing algorithms for point clouds play an important role in computational science, parameterization, and surface reconstruction. In particular, the Delaunay tetrahedrization [12,20], and its dual, the Voronoi tessellation, are standard tools often used in natural neighbor interpolation [74,76] and 3D surface reconstruction [2]. However, global tessellations often become undesirable in practice as data sets grow in size and complexity.

Our work is a visualization system in which various interpolation methods can be incorporated. Rather than construct a global tessellation of the data points, we define a set of local tetrahedrizations that are guaranteed to cover the domain. Inside each local tetrahedrization, point-based contouring is applied to extract isosurface geometry. This work differs from previous work in that we make very few assumptions about the density of the point distribution, and we construct direct visualizations of the data without global tessellation or resampling. The contribution of this direct approach is a relatively general method that produces faithful scientific visualizations.

5.3 Regularization

To facilitate the construction of local tetrahedral meshes, we first regularize the point distribution. We add data points to the data set such that the resulting point distribution is quasi-uniform. We create a regular grid around the data points and distribute the points among the cells of this grid. Empty cells are filled with a single data point at the center of the cell whose function value is determined by interpolating. (Interpolation is discussed in Section 5.5.) These additional points are used for creating local tetrahedrizations and are not considered when evaluating the approximation. We refer to the resulting grid as the *binning grid*.

The properties of a desirable binning grid balance two competing desires. First, the grid should be coarse enough to minimize the number of evaluations of the scattered data interpolant. Since the interpolant is evaluated at points added to empty cells of the binning grid, the number of interpolant evaluations is equal to the number of empty cells in the binning grid. Second, the grid should be

fine enough to minimize computation time of the local tetrahedrizations.

The properties of the binning grid are determined by using an octree to balance these competing constraints. The data points are rescaled to fit inside an axis-aligned unit cube, the root node of the octree. This cube is refined adaptively until each leaf cell contains at most one data point of the data set. The binning grid is a set of cells resulting from one full level of octree subdivision. We refer to this level as the *binning grid level* l_b, which corresponds to a $2^{l_b} \times 2^{l_b} \times 2^{l_b}$ grid. The binning grid is computed in two steps. In the first step, l_b is determined by traversing the octree in a depth-first fashion. In the second step, the binning grid is obtained by trimming or augmenting the grid such that a single layer of cells of the binning grid surrounds the axis-aligned bounding box of the data points.

To compute l_b, the octree is traversed in a depth-first fashion until a node is reached that contains at least one empty child node. The level of this node is appended to a list. If a leaf cell is encountered, its level is also added to the list. (A leaf node is a node containing *only* empty children.) The median level of the resulting list of node levels is selected as l_b. It is possible to choose the average level of this list, however we have found that the resulting binning grid is often *too fine*.

5.4 Local Tetrahedral Mesh Generation

Points inside a given $3 \times 3 \times 3$ stencil of binning grid cells are used to construct a local tetrahedrization. The stencils are centered in alternate cells to avoid excessive tetrahedrization overlap, see Figure 5.3. We use Delaunay methods to tetrahedrize the stencil point set. We note that the tetrahedrization provides starting point information for sampling the isosurface and not a globally consistent interpolating function.

Constructing local tetrahedrizations in this manner solves the domain cover problem. Consider that points are added to the outer layer of binning grid cells. The convex hull of these outer layer points form an axis-aligned box that contains the domain of the data points. After regularization, each cell of the binning grid contains at least one point. The union of the stencil tetrahedrizations cover the

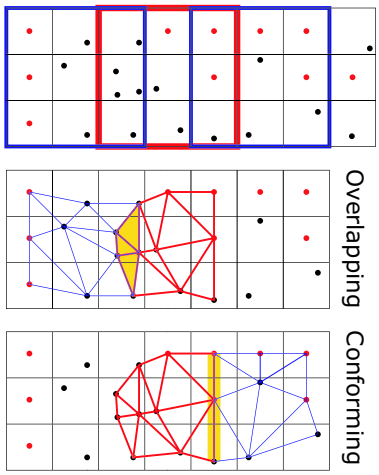


Figure 5.3: Illustration of boundary conditions in stencil triangulations. *Top*: Local triangulations are constructed from points inside 3×3 stencil of cells, shown in blue and red. *Middle*: An example of two neighboring stencil triangulations that overlap. Three triangles from the blue triangulation overlap three triangles of the red triangulation; the overlap is shaded in light orange. *Bottom*: An example of two neighboring stencil triangulations conforming at their boundaries, shaded in light orange. The fact that local triangulations either overlap or conform guarantee domain cover.

box formed by the outer layer points, since these tetrahedrizations span the space covered by the grid cells. This remains true for non-stencil-center cells since the tetrahedrizations either conform or overlap at their boundaries. Thus, since the union of the stencil triangulations cover the outer layer grid, and the outer layer grid contains the domain of the scattered data, the domain of the scattered data is appropriately covered.

Figure 5.3 shows 2D examples of conforming and overlapping local triangulations. Our algorithm naturally produces meshes that conform. However, it is not necessary for the local meshes to conform to achieve domain cover. The local tetrahedrizations do not need to be computed *a priori*, but rather are generated on-the-fly and later discarded when the stencil is no longer needed. This greatly reduces the memory requirement for storing tetrahedrizations and enables a distributed implementation.

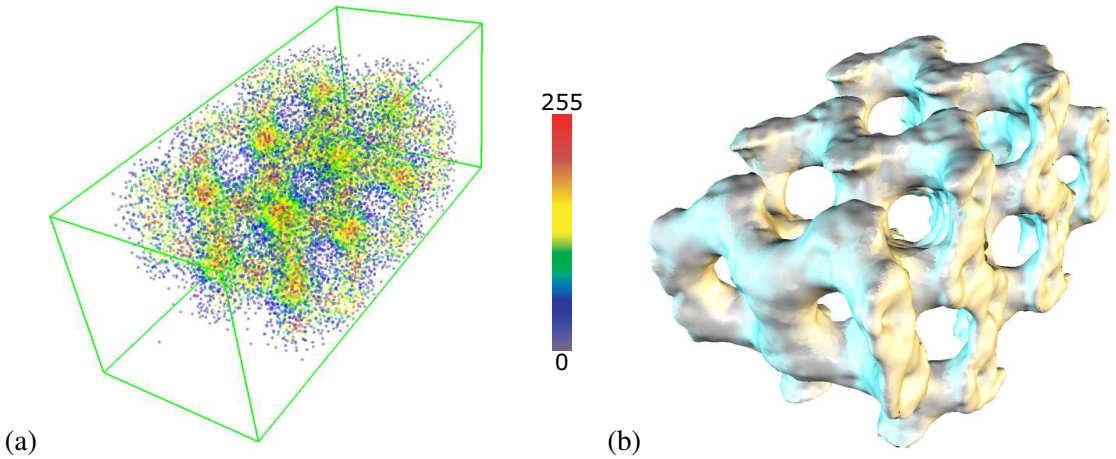


Figure 5.4: (a) A visualization of the irregular data point distribution for the silicium data set. (b) Point-based isosurface extracted from the scattered data.

5.5 Interpolation

For interpolation, we reuse the octree data structure computed for regularization. Our interpolation method uses the algorithmic structure of the multi-level partition-of-unity (MPU) implicits method of Ohtake et al. [53]. This method constructs a global approximation by blending local quadric approximations placed with the aid of an octree. Instead of using piecewise quadric approximations at octree nodes, we use local radial basis function (RBF) interpolants. During octree construction, we compute at each node information about the number of points contained in each cell as well as the radius of a sphere of influence centered at the centroid of the cell. The radius is computed to be αd, where d denotes the length of the octree node's diagonal. (We use $\alpha = 0.75$, which was shown to be an effective choice of α in the MPU implicits method.)

We use partition-of-unity weight functions to construct a global approximation of the scalar field by blending together local RBF interpolants. We use the multiquadric RBF discussed in Section 2.1.1. The local RBF interpolants are placed adaptively using our octree by associating them with the centroids of selected octree nodes, called *interpolation nodes*. We partition the data points into clusters manageable by the matrix solver [19]. At a given node, we collect the points inside the node's sphere of influence for the local interpolant. If the number of collected points is above a user-defined threshold N, we traverse the octree deeper until the matrix problem size is less than or

equal to N. It is possible for the sphere of a leaf node in the octree to contain "too many" points, in which case, we augment the tree with further subdivisions until the threshold is satisfied. (In our implementation, we chose N to be 100 because the linear system can be solved relatively quickly while still produce high-quality results.)

Evaluation of the approximation is performed using a recursive routine in $O(log\ m)$ time, where m is the number of interpolation nodes in the octree [53]. The contribution of each local interpolation is calculated, the sum of the weights is accumulated, and the final value is normalized by dividing the sum of the contributions by the sum of the weights. It is important to note that only the original sample points are used for interpolation.

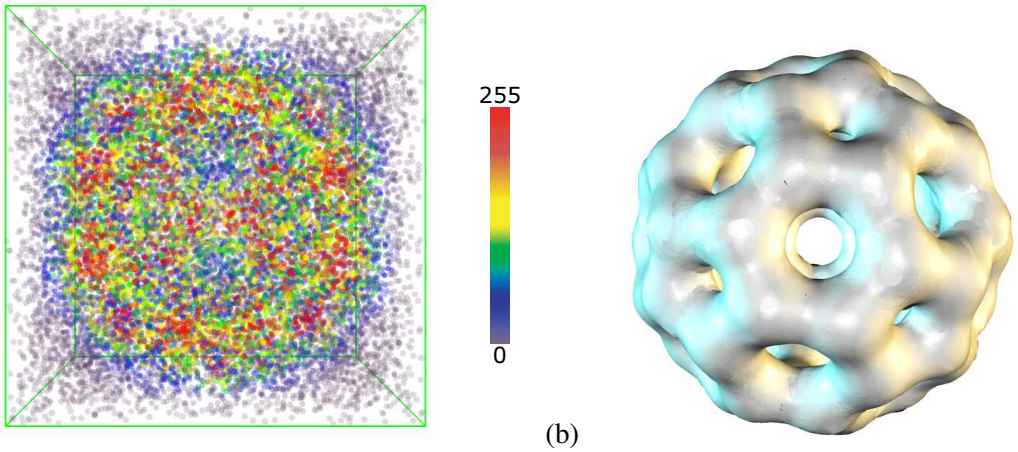
5.6 Isosurface Generation

To construct isosurfaces from scattered data sets, we adapt the point-based, meshless isosurface generation method [19] outlined in Section 4. Our adaptation uses *marching tetrahedra* to obtain an initial triangle approximation for the isosurface of interest. These triangles are decomposed into points and projected to the actual isosurface as defined by the interpolation scheme. Since the triangles are derived from data points using the original data set, we believe the triangles serve as a reasonable initial approximation to the final surface.

The resulting surfaces are continuous point-based representations. The initial set of triangles resulting from the local tetrahedrization may be discontinuous. Once these triangles are decomposed into points and projected to the implicit surface defined by the interpolating function and the isovalue, the discontinuities disappear, and the surface defined by the interpolating function can be seen.

Isosurfaces are accurately approximated by computing points via Newton-Raphson iteration. Convergence of the iteration scheme is achieved when the displacement of the point between iterations is less than a user-defined threshold. (In our results, this threshold was set to 0.01.) In this way, isosurfaces accurate to within a user-defined threshold are produced.

The isosurface generation process is performed in a streaming fashion, such that only the current



(a) (b)

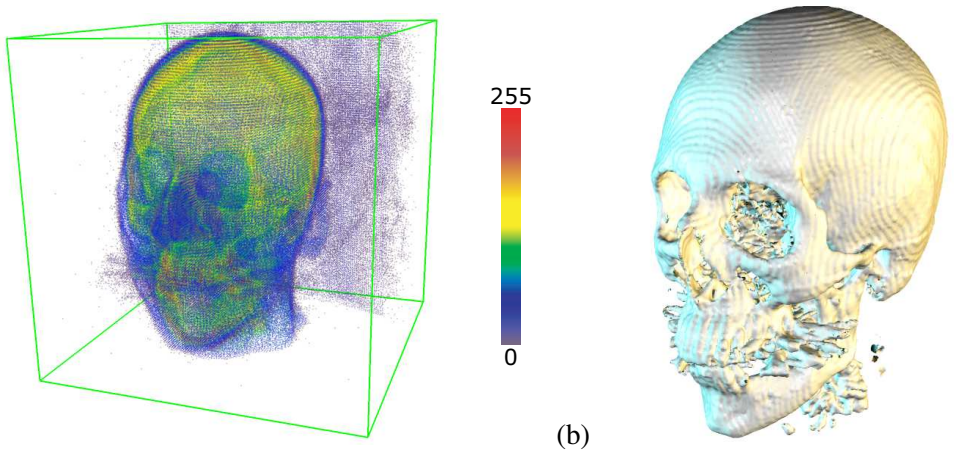
Figure 5.5: (a) A visualization of the irregular data point distribution for the buckyball data set. (b) Point-based isosurface extracted from the scattered data.

stencil tetrahedrization needs to be stored in main memory at any given time. A *marching stencil* generates the tetrahedrization, the triangles approximating the isosurface, and the resulting point-based isosurface on-the-fly.

5.7 Results

We implemented a parallel version of the isosurface contouring method, where the machines concurrently generate isosurface fragments while marching through separate stencils. We implemented a simple master-slave architecture, where a single master monitors the progress of slaves generating the isosurface geometry. We used a cluster of eleven desktop PCs communicating through MPI over a 100 Mb/s line. We employed a hybrid computing network consisting of computers with processors ranging from 2.80 GHz to 3.20 GHz and memory ranging from 1 GB to 4 GB. Given this configuration, we performed isosurface generation for a variety of data sets, measuring preprocessing time (octree construction, local RBF construction, and regularization) to assess the effectiveness of the approach. Timing results for the preprocessing are given in Table 5.1.

To test the method, we utilized four data sets from a variety of sources. The buckyball data set was obtained by sampling a 128^3 buckyball data set for 20,000 points using a uniform random distribution. For many of the data sets, we used levels of detail obtained from a meshless multiresolution



(a) (b)

Figure 5.6: (a) A visualization of the irregular data point distribution for the HighResHead data set. (b) Point-based isosurface extracted from the scattered data. We note that the bumpiness in the isosurface is part of the data and not an artifact of our reconstruction.

hierarchy [17]. Data points in the hierarchy are determined using an iterative refinement process based on principal component analysis and binary space partitioning. While the original data sets are regular grid structures, the point distributions resulting from the meshless hierarchy are irregularly spaced. We used the meshless multiresolution hierarchy to generate the silicium data set, the "LowResHead" data set, and the "HighResHead" data set. Table 5.1 provides the sizes of all data sets. Visualizations of the data distributions as well as of the point-based isosurfaces extracted in the timing experiments are shown in Figures 5.4, 5.5, and 5.6. We note that the "bumpiness" in the isosurfaces of Figure 5.6 is part of the data and not an artifact of our surface reconstruction.

In the case of HighResHead, regularization increases the number of data points by over a factor of five. This observation is indicative of the high density regions of data points focused around fine features in the data (see Figure 5.6), which influence the binning grid level in favor of a higher resolution binning grid. One remedy to avoid storing a large number of additional samples is the generation of samples in the empty cells on-the-fly. Nevertheless, regularization allows our system to easily generate local tetrahedrizations that are guaranteed to cover the domain. This is an important advantage of our algorithm, since it opens up the possibility to exploit the computational power of several machines by processing local tetrahedrizations in parallel.

Data set	Number of data points	Number of local RBFs	Number of empty cells
silicium	17,532	4,180	2,519
buckyball	20,000	4,096	31
LowResHead	203,359	66,961	199,206
HighResHead	345,452	92,022	1,758,408

Data set	Octree construction	Local RBF construction	Regularization	Preprocess Total	Isosurface generation
silicium	0.1 s	3.2 s	0.1 s	3.4 s	176.3 s
buckyball	0.2 s	3.1 s	0.1 s	3.4 s	69.2 s
LowResHead	1.4 s	26.9 s	4.9 s	33.2 s	364.2 s
HighResHead	1.9 s	48.8 s	45.5 s	96.2 s	375.1 s

Table 5.1: Summary of data sets and results of the timing experiments.

5.8 Discussion

Octree construction time is $O(n \log n)$, where n is the number of points in the data set. The time complexity of the local RBF construction depends heavily on the distribution of the data points. Regularization is proportional to the number of cells in the binning grid that are empty. The stencil containing the largest number of data points determines an upper bound for the construction of the stencil tetrahedrizations. While pathological point distributions can be artificially generated causing the tetrahedrization of a possibly large number of points, our method works well for data sets commonly encountered in practice.

Our system's performance is sensitive to the computational requirements of the interpolation method, which in the case of scattered data is slow. However, the quality of the images is better. While the speed advantage of resampling techniques and trilinear interpolation may justify the potential lack of accuracy in the visualized result in certain applications, we attempt to retain the information and the related context of the original data to the greatest degree possible and not introduce any biases into the visualization. Our technique scales to large parallel computational systems and can be easily modified to utilize new interpolation methods as they become available.

Chapter 6

Meshless Volumetric Hierarchies

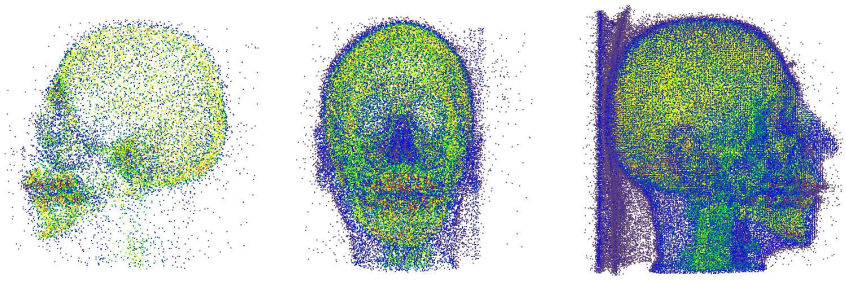


Figure 6.1: Scatter plots of cluster center points visualizing three levels of detail of a head data set.

6.1 Introduction

Multiresolution techniques are commonly used in computer graphics to manipulate objects at different levels of detail. It is a topic also intimately related to object simplification and compression. While most multiresolution techniques developed thus far for surfaces and volumes require the use of connectivity information, relatively few methods exist that can be directly applied to meshless data sets.

We propose a new approach for the construction of a hierarchical representation of any volumetric scalar data set. In a preprocessing step, we iteratively refine an initially coarse representation using

clustering techniques to generate a hierarchy. At runtime, we extract levels of detail from this hierarchy to support interactive exploration.

We start with a coarse data representation, consisting of a single cluster containing all the sample points of the data set. This cluster is partitioned into two sub-clusters, which are inserted into a priority queue sorted by error. This procedure is applied iteratively; in each step, the cluster with the highest error is partitioned, and its sub-clusters are placed into the queue. Refinement terminates when a maximum number of iterations have been completed or when the maximum error in the priority queue is below some user-defined threshold. A hierarchy of clusters is built using the natural parent-child relationship created by this splitting procedure. We refer to the resulting hierarchy as a cluster binary tree (CBT). Cluster partitioning is discussed in Section 6.3.

The level-of-detail extraction phase consists of a depth-first traversal over the CBT. We discuss two traversal methods: level-based and error-based. The level-based approach collects data in the hierarchy in a depth-first fashion, traversing the tree down to a user-defined maximum depth. The error-based approach gathers data in the cluster hierarchy based on an error threshold. The set of nodes collected by CBT traversal constitutes a level-of-detail representation of the original data. As a result, multiple resolutions are represented by one compact binary tree. Level-of-detail extraction is discussed in Section 6.4.

Our goal is to build a multiresolution hierarchy without specific knowledge of the source of the volumetric scalar data. Although this information can often be employed to design more effective algorithms, it reduces the applicability of the method. The strength of our approach is that it can be applied to any volumetric scalar data set and avoids the costly generation of a grid.

6.2 Previous Work

Although many simplification and multiresolution efforts have focused primarily on surface meshes, many techniques have been developed for volumetric data. Typically, multiresolution methods organize volumetric data based on regular or irregular grid structures. Regular grid structures in-

clude octrees, which have been used to provide adaptive levels of detail, see [27, 60, 70]. Linsen et al. [47,48] used wavelets and subdivision connectivity to represent and visualize regular grid data in a hierarchical fashion. Adaptive mesh refinement (AMR) techniques use a set of nested regular grids of varying resolution to represent volumes [52, 79]. With respect to irregular grid methods, tetrahedral meshes have played an important role in constructing multiresolution hierarchies. Cignoni et al. [10, 13] described a system based on tetrahedral meshes to represent and visualize volumetric scalar data. Trotts et al. [77] and Staadt and Gross [75] used edge-collapse techniques to extend Hoppe's work [36] for building progressive tetrahedrizations. Grosso and Greiner [30] built hierarchical adaptive meshes using tetrahedra and octahedra.

Unfortunately, most of these methods cannot be applied directly to scattered data, i.e., data with no connectivity information, without first meshing the scattered data points or resampling the data to a regular grid. Triangulations can be expensive to compute and store, especially considering the increasing size of modern scientific data. Weber et al. [78] proposed creating local triangulations at runtime in a given region of interest. While this approach provides a good solution for runtime visualization of scattered data, it does not lend itself to the construction of a multiresolution representation of volume data. Further, triangulation algorithms can be difficult to implement in practice [71], as they require complex mesh data structures to be maintained [56].

Resampling to a regular grid can produce many unnecessary redundancies, although adaptive sampling via an octree or an AMR representation can help. Regular grids typically sample in an axis-aligned fashion, which, though simple to implement and store, may not always produce a desirable partitioning of the volume. Grid data and multiresolution methods for grid data offer several advantages when exploring large data spaces, but these methods are not always easily generalized for all types of volume data. For example, methods for tetrahedral meshes can be applied to hexahedral meshes by decomposing hexahedra into tetrahedra, but the reverse is not true. The most general type of volumetric data is scattered data. Thus, any multiresolution method for scattered data could be applied to any gridded data by simply ignoring the mesh.

To create a multiresolution hierarchy for scattered scalar data, it is desirable to use methods that use "simple" connectivity, or no connectivity at all. Similar methods have been developed in surface

simplification, surface reconstruction, and vector field hierarchy creation. These methods partition data points into similar sets, or clusters. Inspired by vector quantization methods, Brodsky and Watson [6] used principal component analysis (PCA) to simplify models by refining an initially coarse representation. Their work prompted Shaffer and Garland [69] to apply PCA-based vertex clustering and the dual quadric error metric to simplify models adaptively in an out-of-core fashion. Pauly et al. [56] developed several extensions of multiresolution methods for point-sampled surfaces. Heckel et al. [34] used PCA to determine near-planar polygonal tiles for the reconstruction of surfaces from point cloud data. Vector field hierarchies were constructed using PCA in a similar way by clustering vectors that are locally similar [35].

In our approach, we refine clusters of scattered data points using PCA to define partitioning planes intelligently. While the use of PCA is not new in surface simplification [6, 56, 69], we extend PCA for use in volumetric scalar field simplification. We maintain a point hierarchy, similar in spirit to many multiresolution representations of surfaces [56,66] with the exception that our points represent samples of a scalar field and not samples on a surface. We use RBFs defined using levels of detail from this point hierarchy for field reconstruction and value approximation.

6.3 Clusters

We define a cluster C to be a subset of points in the data set. The center of this cluster, $\mathbf{p}_c$, is

$$\mathbf{p}_c = \frac{1}{|C|} \sum_{\mathbf{p}_j \in C} \mathbf{p}_j.$$

We associate with each $\mathbf{p}_c$ of each cluster a function value f_c approximating the entire cluster. This function value is computed by evaluating a locally defined multiquadric $H(\mathbf{x})$ as defined in Section 2.1.1 at $\mathbf{p}_c$ using only the N nearest neighbors to $\mathbf{p}_c$ to construct the coefficients of the approximation. We define $N = min(\,|C|, k\,)$, where k is some threshold.

Given a cluster C, we define its error σ_c as

$$\sigma_c = \max_{\mathbf{p}_j \in C} |\, f_j - f_c \,|$$

where f_j is the scalar value associated with point $\mathbf{p}_j$, and f_c is the value approximated at cluster center $\mathbf{p}_c$. In other words, the error σ_c is the maximum deviation in scalar value between the approximated value and the values of all points in C. This error measure is simple to compute and suffices to identify clusters to be refined.

In each refinement step, the cluster with the highest error is obtained from the queue. We partition it into two smaller clusters by defining a splitting plane that divides the cluster into two distinct subsets. Center points, value approximations at center points, and errors are computed for the two resulting sub-clusters. The sub-clusters are then inserted into the queue. A binary tree is maintained during refinement by setting the sub-clusters to be children of the cluster just split. In this hierarchy, each cluster is interpreted as a data point of a given resolution in the hierarchy whose location is the cluster center $\mathbf{p}_c$.

One possible refinement strategy uses an axis-aligned scheme. In this method, the cluster center and one coordinate axis determine the splitting plane. This kd-tree style splitting scheme [68] is efficient and allows us to determine the splitting plane simply, but it may not produce a good decomposition of the data set. Figure 6.2 demonstrates this effect. Furthermore, we wish to define a splitting plane that reduces the error in the sub-clusters the most. We use the cluster center and a normal vector obtained from principal component analysis (PCA) on all four components of the scalar field (x_i, y_i, z_i, f_i) to define a more adaptive splitting plane. For a detailed explanation of PCA, we refer the reader to [6, 34, 37, 69].

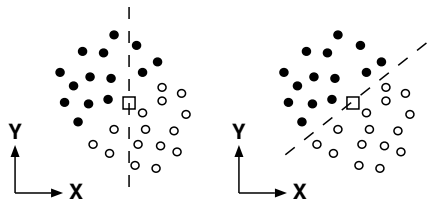
We first discuss how 3D PCA can be used for bivariate scalar fields. For bivariate scalar field data (x_i, y_i, f_i), we can perform PCA in 3D to obtain an orienting normal for a splitting line. PCA performs an eigen-decomposition of the covariance matrix of a set of samples producing, in the 3D case, eigenvalues

$$\lambda_1 \geq \lambda_2 \geq \lambda_3$$

and corresponding eigenvectors

$$\vec{e}_1, \vec{e}_2, \text{ and } \vec{e}_3,$$

which define a local orthogonal coordinate system related to an ellipsoid induced by the data points. We use the vector corresponding to the dominant axis of this ellipsoid, i.e., $\vec{e}_1$, as the splitting



(a) (b)

Figure 6.2: Comparison of splitting schemes. Black dots represent points with scalar value one, and white dots represent points with scalar value zero. (a) Non-optimal splitting using an axis-aligned scheme, (b) near-optimal splitting.

plane's normal. However, to partition the 2D points in the domain, we require a 2D normal vector. We project the 3D eigenvector to xy-space, see Figure 6.3. In most cases, we obtain a suitable normal by simply dropping the last component of the eigenvector.

We must consider the case when $\vec{e}_1$ is a multiple of the vector $\langle 0,0,1 \rangle$. Such a vector projected to xy-space produces the null vector. In this case, we choose the second dominant eigenvector $\vec{e}_2$ which is guaranteed to be non-null when projected to xy-space, since it is orthogonal to $\vec{e}_1$. In practice, this occurs infrequently. (This never occurred in the results presented in Section 6.6.) Figure 6.4 illustrates the progression of the cluster splitting procedure.

This technique generalizes to trivariate scalar field data (x_i, y_i, z_i, f_i). PCA returns eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_4$ with corresponding eigenvectors $\vec{e}_1, \vec{e}_2, \vec{e}_3$, and $\vec{e}_4$. Again, we use the projection of $\vec{e}_1$ to xyz-space to define a splitting plane normal. When projection of $\vec{e}_1$ maps to the null vector in xyz-space, we choose $\vec{e}_2$ as our orienting normal.

Defining the splitting plane in this way divides a cluster across the axis of its greatest variation, thereby decreasing the cluster's error. Geometrically speaking, this partitioning splits one ellipsoid into two "rounder" child ellipsoids with reduced eccentricity. This shape can have an important positive side-effect on subsequent partitioning. When clusters become very thin in the split direction, it is possible that splitting a cluster produces a sub-cluster with no data points due to numerical error. Splitting across the dominant axis as defined by PCA avoids the creation of thin clusters by producing sub-clusters that are as "round" as possible. This numerical issue cannot be avoided completely; when it occurs, the "problem cluster" is not re-inserted into the error queue.

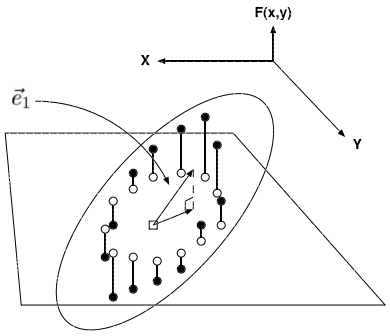


Figure 6.3: Example of 3D PCA normal projected to xy-space. The white dots are data points in the xy-plane. The height of each black dot indicates the scalar value at the data point. The ellipsoid represents the local coordinate system computed by 3D PCA. The dominant eigenvector $\vec{e}_1$ is shown with its projection onto the xy-plane.

Cluster size can also have an effect on splitting. Eventually, clusters contain only a few data points. A minimum cluster size can be defined to set a "pseudo-compression ratio" for the finest resolution in the data hierarchy. Defining a minimum cluster size can also reduce the occurrences of the zero-size cluster problem.

6.4 Level-of-Detail Extraction

Extraction of levels of detail from the CBT is performed by traversing the tree in a depth-first fashion, using either a *level-based* traversal that obtains the clusters in the hierarchy at a given level of the tree, or an *error-based* traversal that returns clusters in the hierarchy that have an error below a threshold. If a leaf node is encountered in the CBT during extraction, we use the cluster at the leaf and continue traversal. When traversing the CBT in a level-based manner, levels of detail are specified by the maximum depth to traverse the binary tree. When traversing the CBT in an error-based manner, levels of detail are specified by a maximum error that the level-of-detail should exhibit. (The hierarchy can only guarantee clusters with an error less than or equal to the maximum error at the termination of the preprocessing.) With the error metric we have defined, a low error threshold returns a high fidelity representation. Conversely, a higher error threshold returns a less faithful but memory efficient representation.

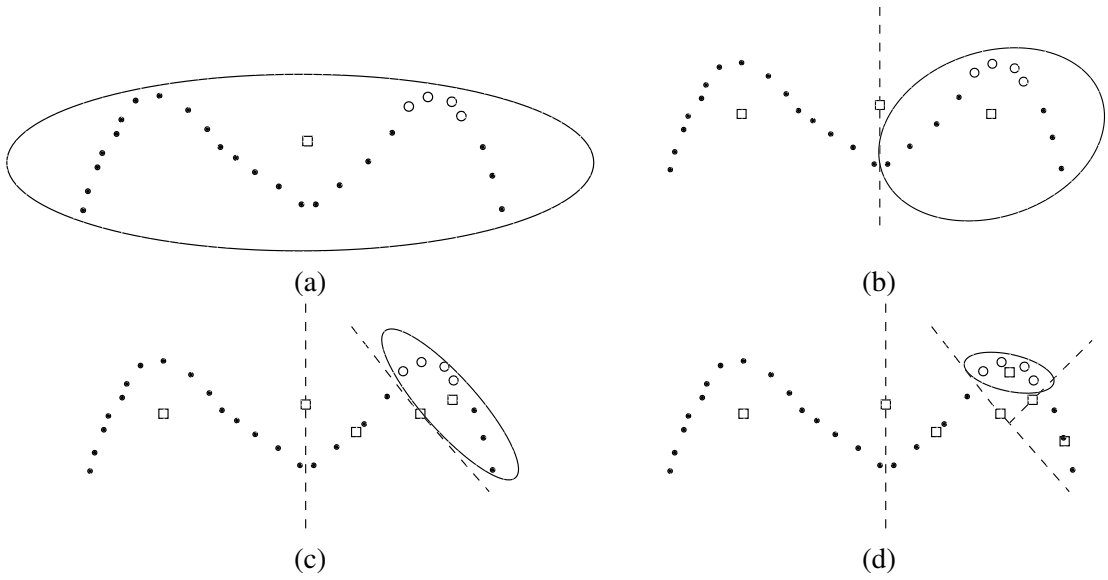


Figure 6.4: Example of clustering of 2D scattered data. Black dots indicate points with scalar value one, and white dots indicate points with scalar value zero. Squares represent cluster centers, which become new data points in the generated hierarchy. (a) CBT generation begins with one initial cluster; (b) cluster is split into two sub-clusters; only the right cluster is chosen for splitting; (c) right cluster is split into two sub-clusters; (d) final split; all clusters have zero error.

Error-based traversal offers a more compact representation, whereas the level-based approach provides better spatial distribution by covering the space spanned by the hierarchy with more data points. This phenomenon can be seen in the examples shown in Figure 6.5 and is further discussed in Section 6.6. In Figure 6.5, two levels of detail are shown for each traversal method. Between two resolutions of the hierarchy traversed in a level-based way, additional points are added in a spatially uniform manner. Between two resolutions of an error-based hierarchy, the distribution of the additional points depends more strongly on the nature of the field. Figure 6.5 demonstrates this effect for a silicium data set. In the error-based traversal examples (Figure 6.5, right column), only a few clusters are extracted to represent the volume outside the silicium structure, whereas several more clusters are used in that same region using a level-based approach (Figure 6.5, left column).

6.5 Multiresolution Representation

The clusters extracted from the CBT define a level-of-detail representation of the volume by considering the $\mathbf{p}_c$ and f_c for each cluster as locations and values of a scattered scalar field. This set

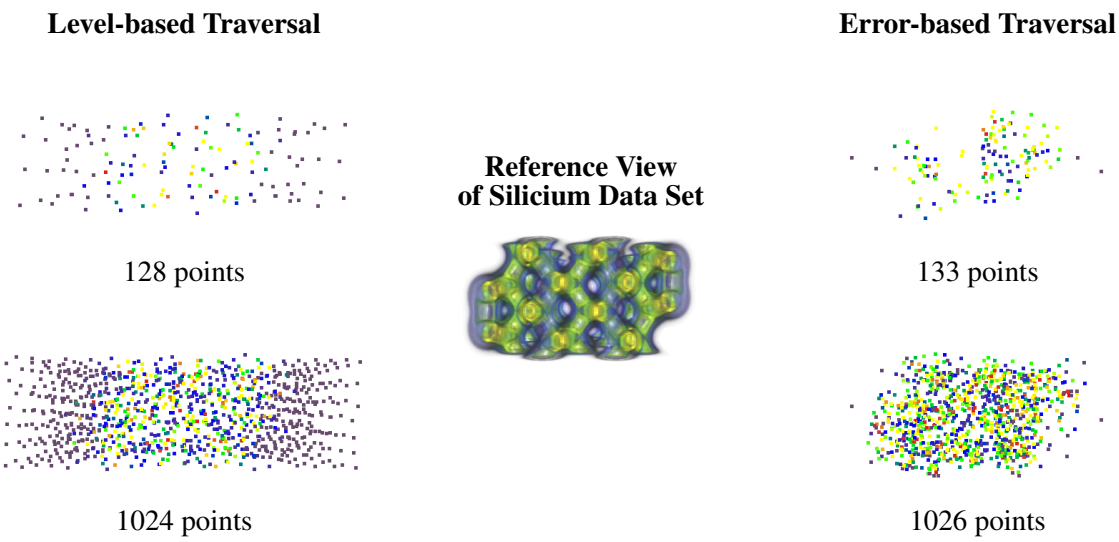


Figure 6.5: Demonstration of the difference between level- and error-based CBT traversal. Shown are 3D cluster center point scatter plots visualizing levels of detail extracted from a CBT. The volume rendering (center) is provided for reference. Level-based CBT traversals (left column) provide better spatial data point distribution while error-based CBT traversals (right column) provide more detail using roughly the same number of data points.

of data points, in conjunction with the field reconstruction basis functions allows us to evaluate the function at any arbitrary location, and thus construct visualizations.

6.6 Results

We generated CBT hierarchies for three data sets. We extracted low- and high-resolution levels of detail from each CBT and volume rendered the fields to inspect the quality of the representation.

Table 6.1 summarizes the preprocessing results to generate the CBTs. For value approximation at the cluster centers and for sampling the data, we used 25 nearest neighbors and fixed the multiquadric parameter to 0.025. The minimum splittable cluster size was set to two. All function values are between zero and 255.

Figure 6.6 shows volume rendered visualizations of different levels of detail represented by the CBTs. The first column shows a volume rendering of the original data. Columns two and three provide volume visualizations of high and low resolutions, respectively. Information concerning

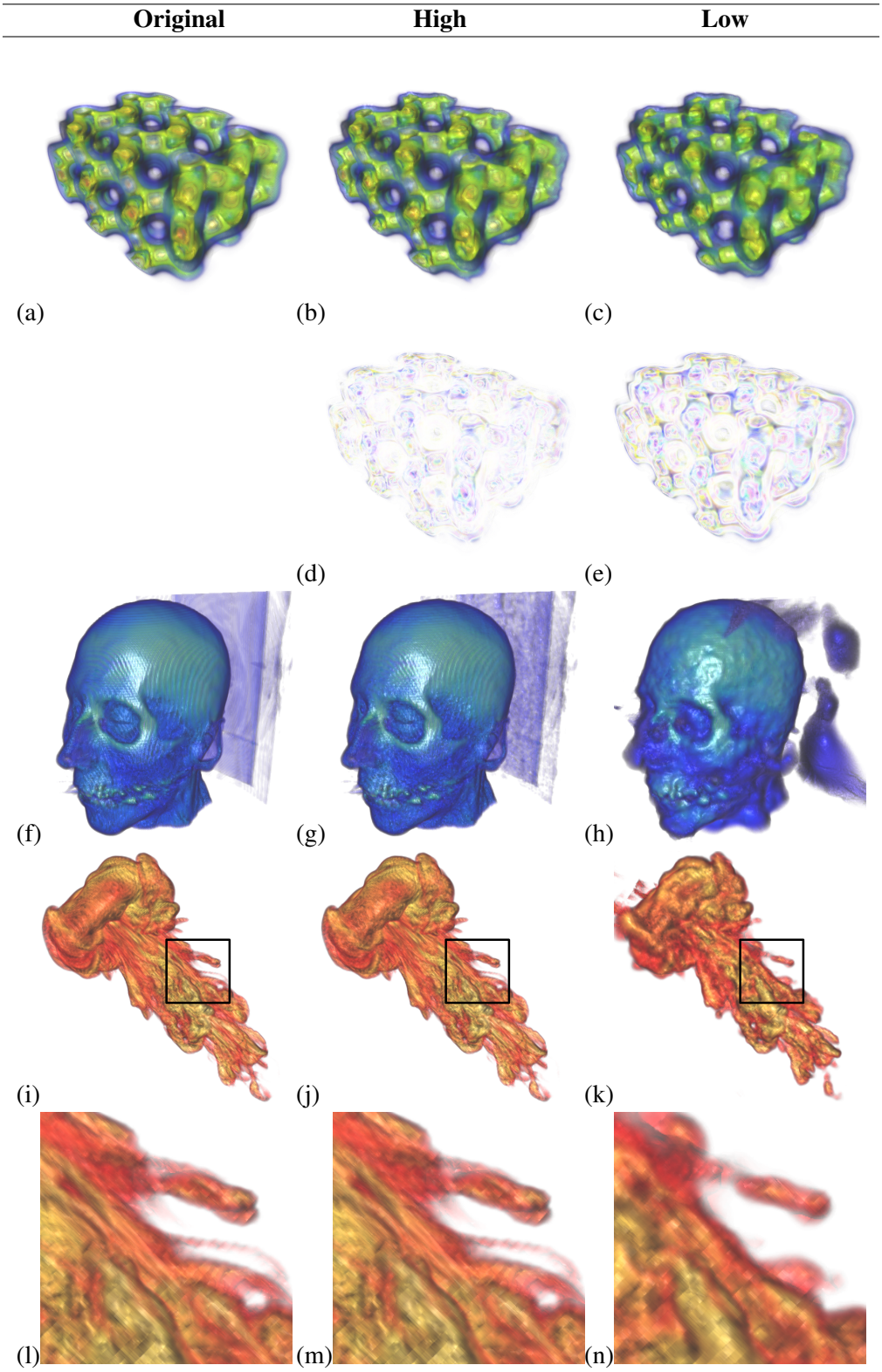


Figure 6.6: Quality comparisons for various resolutions of three data sets.

the parameters used to extract these levels of detail and the number of data points used for rendering is provided in Table 6.2. To illustrate the quality of the various resolutions, "zooms" are given for the argon bubble data set in Figures 6.6 (l), (m), and (n), and difference images are provided for the silicium data set in Figures 6.6 (d) and (e). The difference images were obtained by differencing images in Figure 6.6 (b) and (c) against image (a) of the original data set.

This method lends itself to data and computation parallelism. Once a cluster is split, its sub-clusters can be further split on separate machines. Since the hierarchy is based on a binary tree, merging the final results is low in cost. For each data set, preprocessing was performed in parallel on a hybrid PC cluster consisting of three machines with Pentium4 2.8 GHz processors with 2 GB of main memory, and one machine with a Pentium4 2.2 GHz processor with 1 GB of main memory.

As mentioned in Section 6.3, eventually clusters either converge to the granularity of the data set or suitable partitioning is not possible due to lack of precision. "Skipped splits" can result in the refinement, and their numbers are listed in Table 6.1.

Naturally, lower resolutions are less faithful to the original data set. This fact is seen in the results for the head data set, shown in Figures 6.6 (f), (g), and (h). A low-resolution representation produces a low-quality volume rendered result as seen in Figure 6.6 (h), where the entire back plate from the scan is not represented in detail. Although less than one percent of the number of data points (17,528 data points) was used to construct the image, salient features of the head, such as the eye sockets and spikes around the teeth, are still discernible. Deficiencies of low resolutions are eliminated in higher resolutions, as seen in Figure 6.6 (g).

The images shown in Figure 6.6 of the silicium data set demonstrate some of the differences between error-based and level-based hierarchies. The argon bubble and head results were generated using an error-based approach, whereas the silicium data set results were generated using level-based extraction. As can be seen in the difference images, Figures 6.6 (d) and (e), successively higher resolutions of the data set improve the overall quality of the field representation but not any particular region. By contrast, the images of the head demonstrate that successively higher resolutions of the error-based hierarchy adaptively improve specific regions. The back plate and nose appear "blobby" in Figure 6.6 (h) and not in (g), while the regions around the teeth and eye socket

Data set	Number of splits	Skipped splits	CBT depth	Processing time
Silicium	64,084	2	19	50 s 735 us
Head	300,663	19	23	44 min 32 s
Argon bubble	224,009	6	25	8 min 11 s

Table 6.1: CBT hierarchy generation statistics.

Data set	Figure	Extraction	Threshold	Size (points)	% Size
Silicium	(b)	level	15	20,982	18.52 %
	(c)	level	16	39,324	34.71 %
(original)	(a)	n/a	n/a	113,288	100 %
Head	(h)	error	60.00	17,528	0.84 %
	(g)	error	4.50	300,648	14.34 %
(original)	(f)	n/a	n/a	2,097,162	100 %
Argon bubble	(k), (n)	error	50.00	11,397	0.22 %
	(j), (m)	error	1.00	224,003	4.27 %
(original)	(i), (l)	n/a	n/a	5,242,880	100 %

Table 6.2: Statistics for visualizations shown in Figure 6.6. Column two lists the location(s) of images in Figure 6.6 to which a row's information applies.

remain fairly consistent across resolutions.

Levels of detail of the argon bubble data set exhibit high-quality using few data points. In the low and high resolutions of the data set, shown in Figures 6.6 (i), (j), and (k), less than five percent of the number of the original data points was used to reconstruct the field. Zooms shown in Figures 6.6 (l), (m), and (n) illustrate the quality of the representation.

Chapter 7

Future Work

While much has been accomplished through the techniques presented in Chapters 3–6, many open problems exist, and there are several areas that should be explored.

- First, point-based isosurface sampling methods need to be extended to improve convergence to the isosurface.
- Next, refinements to the sampling framework are necessary to cope with the possibility of pathological point distributions.
- Appropriate tools to analyze the "field complexity" of a given meshless data set are necessary in order to ensure that a faithful visual representation of the data is constructed.
- Finally, volume visualization algorithms that build upon the existing framework are necessary to realize a complete suite of meshless data exploration tools.

We address each of these issues briefly in the following sections.

7.1 Improvements to Isosurface Sampling

The iso-splatting method makes use of an approximate projection scheme that relies on a "good" initial guess for isosurface geometry. This sensitivity to the initial guess causes the single Newton-Raphson step to produce an approximate projection to the isosurface that can possibly be far from the isosurface. The situation is aggravated by the existence of *vanishing gradients*, gradients that degenerate to the null vector in certain areas of the field. In the application of iso-splatting to multi-block data sets and scattered data sets, additional iterations were employed to address this issue. We strongly believe that these problems can be ameliorated by examining the gradient behavior and using spatial subdivision strategies to determine the maximum displacement realizable by a sample point. Such a sampling strategy will improve the convergence of this method and possibly other point-projection methods [1, 3]. We feel that improved sampling strategies are necessary and important, since the extraction of isosurfaces continues to be an important area of research.

7.2 Arbitrary Point Distributions

The scattered data isosurfacing technique [18] presented in Chapter 5 consists of methods to define effective local tetrahedrizations. It works well in many cases, however, pathological point distributions can occur causing the system to perform a large amount of work in the regularization process. Since the target application is sensor network data, where point distributions will not always be controllable, it is important to develop strategies to cope with even pathological cases. Ideally, each local tetrahedrization produced by the system would consist of tetrahedra forming a subset of a global Delaunay triangulation over the original data points. Producing such local tetrahedrizations such that the entire domain of the field is covered is a largely unsolved problem. Alternatively, other fast local meshing strategies that can guarantee domain cover while simultaneously guaranteeing a specific sampling accuracy are desired. In general, methods for processing volumes defined by arbitrary point distributions is an immensely challenging problem that will occupy the research community for years to come.

7.3 Field Complexity Analysis Tools

Resampling meshless data sets to a regular grid and subsequently performing visualization on the resampled grid is an unacceptable solution. It is possible to overlook important features that cannot be reconstructed accurately if the grid is not defined properly. Furthermore, error can be introduced by attempting to visualize a double resampling of the data. However, if a resampling grid could be defined such that features could be preserved faithfully and error tolerably controlled, this approach would become extremely attractive. Visualization algorithms designed for grid data are highly optimized by taking advantage of the implicit connectivity of the grid, and several options for interactive visualization of this type of data exist. Ideally, the "complexity" of the field can be characterized by analyzing the field in frequency domain as Pauly and Gross [55] did in their spectral analysis of surfaces defined as a point cloud.

7.4 Additional Volume Visualization Tools

While most of the work presented concerns isosurface generation, many other techniques for volume visualization are possible. Such visualization methods could leverage the tetrahedrization component of the scattered data isosurfacing algorithm. For instance, volume slicing could be accomplished by collecting stencils intersected by a slice of interest and using the local tetrahedra associated with each stencil to adaptively sample in the plane of the slice. Volume rendering could be achieved by computing several such slices, where each slice is view-aligned, and blending them together. Another possible approach to volume rendering is the generation of volume splats inside the cells of each local triangulation, although special care must be taken in dealing with regions of overlap. We firmly believe that the sampling framework will support the development of these and other visualization algorithms for volumetric scattered data exploration.

Chapter 8

Conclusions

In summary, the main contributions of this dissertation are (1) a flexible point-based isosurface generation algorithm; (2) a general solution to multiblock isosurface generation; (3) an accurate isosurface generation algorithm for meshless data; and (4) a meshless multiresolution method.

Exploring meshless data sets is a problem of growing importance that has its roots in scattered data visualization. The rapid development of sophisticated sensor networks and other meshless data collection technologies is increasing the need for visualization algorithms and processing paradigms that operate directly on the data without relying heavily on an underlying mesh representation. The work presented thus far takes important first steps toward accomplishing this goal. There are several contributions of the work already accomplished, namely the isosurface and multiresolution techniques developed in this dissertation specifically for meshless data. Because many non-meshless data representations can be considered meshless data sets augmented with connectivity information, the methods described are broadly applicable to a wide variety of data. In some cases, meshless methods solve existing problems in processing non-meshless data, as was seen in the application of meshless isosurface generation to multiblock data. Most importantly, through the use of powerful visualization tools, the algorithms described will foster the continued development of meshless data collection and simulation technology, both in sensor networks and in other areas of science and engineering.

Bibliography

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point Set Surfaces. In *IEEE Visualization '01 (VIS '01)*, pages 21–28, Washington - Brussels - Tokyo, October 2001. IEEE.

[2] N. Amenta, M. Bern, and M. Kamvysselis. A New Voronoi-Based Surface Reconstruction Algorithm. In M. Cohen, editor, *Proceedings of SIGGRAPH 98*, Annual Conference Series, Addison Wesley, pages 415–422. Addison Wesley, 1998.

[3] N. Amenta and Y.J. Kil. Defining point set surfaces. In *ACM*, volume 23 of *ACM Transactions on Graphics (TOG)*, pages 264–270, 1515 Broadway, New York, NY 10036, 8 2004. ACM, ACM Press.

[4] I. Amidror. Scattered Data Interpolation Method for Electronic Imaging Systems: a Survey. *Journal of Electronic Imaging*, 11(2):157–176, 4 2002.

[5] M. Botsch, A. Wiratanaya, and L. Kobbelt. Efficient high quality rendering of point sampled geometry. In S. Gibson and P. Debevec, editors, *Proceedings of the 13th Eurographics Workshop on Rendering (RENDERING TECHNIQUES-02)*, pages 53–64, Aire-la-Ville, Switzerland, June 26–28 2002. Eurographics Association.

[6] D. Brodsky and B. Watson. Model simplification through refinement. In *Proceedings of the Graphics Interface 2000*, pages 221–228, Toronto, Ontario, May 15–17 2000. Canadian Information Processing Society.

[7] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and Representation of 3D Objects with Radial Basis Functions. In *Proceedings ACM SIGGRAPH 2001*, pages 67–76, Los Angeles, CA, August 2001. ACM SIGGRAPH.

[8] Y. J. Chiang and C. T. Silva. I/O optimal isosurface extraction. In R. Yagel and H. Hagen, editors, *IEEE Visualization '97*, pages 293–300. IEEE, 1997.

[9] Y. J. Chiang, C. T. Silva, and W. J. Schroeder. Interactive out-of-core isosurface extraction. In D. Ebert, H. Hagen, and H. Rushmeier, editors, *IEEE Visualization '98*, pages 167–174. IEEE, 1998.

[10] P. Cignoni, L. De Floriani, C. Montoni, E. Puppo, and R. Scopigno. Multiresolution modeling and visualization of volume data based on simplicial complexes. In A. Kaufman and

W. Krueger, editors, *1994 Symposium on Volume Visualization*, pages 19–26. ACM SIGGRAPH, October 1994.

[11] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, April 1997.

[12] P. Cignoni, C. Montani, and R. Scopigno. DeWall: A Fast Divide and Conquer Delaunay Triangulation Algorithm in E^d. *Computer-Aided Design*, 30(5):333–341, 1998.

[13] P. Cignoni, E. Puppo, and R. Scopigno. Multiresolution representation and visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):352–369, October 1997.

[14] Center for Information Technology Research in the Interest of Society (CITRIS). URL: http://www.citris.berkeley.edu/.

[15] H. E. Cline, W. E. Lorenson, S. Ludke, C. R. Crawford, and B. C. Teeter. Two algorithms for the three-dimensional reconstruction of tomograms. In *Medical Physics*, volume 15, pages 320–327, June 1988.

[16] C. S. Co, B. Hamann, and K. I. Joy. Iso-splatting: A Point-based Alternative to Isosurface Visualization. In J. Rokne, W. Wang, and R. Klein, editors, *Proceedings of the Eleventh Pacific Conference on Computer Graphics and Applications - Pacific Graphics 2003*, pages 325–334, October 8–10 2003.

[17] C. S. Co, B. Heckel, H. Hagen, B. Hamann, and K. I. Joy. Hierarchical Clustering for Unstructured Volumetric Scalar Fields. In G. Turk, J. J. van Wijk, and R. Moorhead, editors, *Proceedings of IEEE Visualization 2003*, pages 325–332. IEEE, October 19–24 2003.

[18] C. S. Co and K. I. Joy. Isosurface Generation for Large-Scale Scattered Data Visualization. In G. Greiner, J. Hornegger, H. Niemann, and M. Stamminger, editors, *Proceedings of Vision, Modeling, and Visualization 2005*, pages 233–240. Akademische Verlagsgesellschaft Aka GmbH, November 16–18 2005.

[19] C. S. Co, S. D. Porumbescu, and K. I. Joy. Meshless Isosurface Generation from Multiblock Data. In O. Deussen, C. D. Hansen, D. A. Keim, and D. Saupe, editors, *Proceedings of VisSym 2004*. Eurographics, May 19–21 2004.

[20] B. Delaunay. Sur la sphère vide. *Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.

[21] S. Fleishman, M. Alexa, D. Cohen-Or, and C. T. Silva. Progressive point set surfaces. *ACM Transactions on Computer Graphics*, 2003. In press.

[22] T. Foley and H. Hagen. Advances in Scattered Data Interpolation. *Surveys on Mathematics for Industry*, 4:71–84, 1994.

[23] R. Franke. Scattered Data Interpolation: Tests of Some Methods. *Mathematics of Computation*, 38(157):181–200, January 1982.

[24] R. Franke and H. Hagen. Least Squares Surface Approximation Using Multiquadrics and Parametric Domain Distortion. *Computer Aided Geometric Design*, 16:177–196, 1999.

[25] R. Franke and G. Nielson. Smooth Interpolation of Large Sets of Scattered Data. *International Journal for Numerical Methods in Engineering*, 15(11):1691–1704, 1980.

[26] R. Franke and G. M. Nielson. Scattered Data Interpolation and Applications: A Tutorial and Survey. In H. Hagen and D. Roller, editors, *Geometric Modelling, Methods and Applications*, pages 131–160. Springer-Verlag, 1991.

[27] L. A. Freitag and R. M. Loy. Adaptive, multiresolution visualization of large data sets using a distributed memory octree. In *Proceedings of SC99: High Performance Networking and Computing*, Portland, OR, November 1999. ACM Press and IEEE Computer Society Press.

[28] B. F. Gregorski, M. A. Duchaineau, P. Lindstrom, V. Pascucci, and K. I. Joy. Interactive view-dependent rendering of large isosurfaces. In *Proceedings of the IEEE Visualization 2002*. IEEE, IEEE, 10 2002.

[29] J. P. Grossman and W. J. Dally. Point sample rendering. In G. Drettakis and N. Max, editors, *Rendering Techniques '98*, Eurographics, pages 181–192. Springer-Verlag Wien New York, 1998.

[30] R. Grosso and G. Greiner. Hierarchical meshes for volume data. In F. E. Wolter and N. M. Patrikalakis, editors, *Proceedings of the Conference on Computer Graphics International 1998 (CGI-98)*, pages 761–771, Los Alamitos, CA, June 22–26 1998. IEEE Computer Society.

[31] H. Hagen, R. Franke, and G. Nielson. Repeated Knots in Least Squares Multiquadric Functions. *Computing Suppl. 10*, pages 177–187, 1995.

[32] C. D. Hansen and P. Hinker. Massively parallel isosurface extraction. In *Proceedings Visualization '92*, pages 77–83. IEEE, October 1992. LANL.

[33] R. L. Hardy. Multiquadric Equations of Topography and Other Irregular Surfaces. *Journal of Geophysical Research*, 76:1906–1915, 1971.

[34] B. Heckel, A. E. Uva, B. Hamann, and K. I. Joy. Surface reconstruction using adaptive clustering methods. In G. Brunnett, H. Bieri, and G. Farin, editors, *Geometric Modelling: Dagstuhl 1999 ,Computing Suppl*, volume 14, pages 199–218. Springer-Verlag, 1999.

[35] B. Heckel, G. H. Weber, B. Hamann, and K. I. Joy. Construction of vector field hierarchies. In D. S. Ebert, M. Gross, and B. Hamann, editors, *Proceedings IEEE Visualization '99*, pages 19–26, San Francisco, CA, October 1999. IEEE, IEEE.

[36] H. Hoppe. Progressive meshes. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996.

[37] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, NY, 1986.

[38] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. In S. Spencer,

editor, *Proceedings of the 29th Conference on Computer Graphics and Interactive Techniques (SIGGRAPH-02)*, volume 21, 3 of *ACM Transactions on Graphics*, pages 339–346, New York, July 21–25 2002. ACM Press.

[39] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Mobile Networking for Smart Dust. In *ACM/IEEE Intl. Conf. on Mobile Computing and Networking (MobiCom 99)*, Seattle, WA, August 1999.

[40] A. Kalaiah and A. Varshney. Differential point rendering. In S. J. Gortler and K. Myszkowski, editors, *Rendering Techniques '01*, pages 139–150. Springer-Verlag, August 2001.

[41] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H. P. Seidel. Feature-Sensitive surface extraction from volume data. In S. Spencer, editor, *Proceedings of the Annual Computer Graphics Conference (SIGGRAPH-01)*, pages 57–66, New York, August 12–17 2001. ACM Press.

[42] P. Lancaster and K. Salkauskas. Surfaces Generated by Moving Least Squares Methods. *Mathematics of Computation*, 37(155):141–158, July 1981.

[43] R. S. Laramee and R. D. Bergeron. An Isosurface Continuity Algorithm for Super Adaptive Resolution Data. In J. Vince and R. Earnshaw, editors, *Advances in Modelling, Animation, and Rendering: Computer Graphics International (CGI 2002)*, pages 215–237, Bradford, UK, July 1-5 2002. Computer Graphics Society, Springer.

[44] D. Levin. The Approximation Power of Moving Least-Squares. *Mathematics of Computation*, 67(224):1517–1531, October 1998.

[45] M. Levoy and T. Whitted. The use of points as a display primitive. Technical Report 85-022, University of North Carolina at Chapel Hill, 1985.

[46] P. Lindstrom. Out-of-Core simplification of large polygonal models. In S. Hoffmeyer, editor, *Proceedings of the Computer Graphics Conference 2000 (SIGGRAPH-00)*, pages 259–262, New York, July 23–28 2000. ACMPress.

[47] L. Linsen, J. T. Gray, V. Pascucci, M. A. Duchaineau, B. Hamann, and K. I. Joy. Hierarchical large-scale volume representation with $\sqrt[3]{2}$ subdivision and trivariate B-spline wavelets. In G. Brunnett, B. Hamann, H. Müller, and L. Linsen, editors, *Geometric Modeling for Scientific Visualization*. Springer-Verlag, Heidelberg, Germany, to appear, 2003.

[48] L. Linsen, V. Pascucci, M. A. Duchaineau, B. Hamann, and K. I. Joy. Hierarchical representation of time-varying volume data with 4th-root-of-2 subdivision and quadrilinear B-spline wavelets. In S. Coquillart, H.-Y. Shum, and S.-M. Hu, editors, *Proceedings of Tenth Pacific Conference on Computer Graphics and Applications - Pacific Graphics 2002*. IEEE Computer Society Press, 2002.

[49] Y. Livnat, H. Shen, and C. R. Johnson. A Near Optimal IsoSurface Extraction Algorithm Using the Span Space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[50] W. E. Lorensen and H. E. Cline. Marching Cubes: A high resolution 3D surface reconstruction algorithm. In M. C. Stone, editor, *Siggraph 1987, Computer Graphics Proceedings*,

volume 21, pages 163–169. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, July 1987.

[51] R. W. D. Nickalls. A new approach to solving the cubic: Cardan's solution revealed. In *Mathematical Gazette*, volume 77, pages 354–359. 1993.

[52] M. Ohlberger and M. Rumpf. Hierarchical and adaptive visualization on nested grids. *Computing*, 59(4):365–385, 1997.

[53] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level Partition of Unity Implicits. In J. Hart, editor, *Siggraph 2003, Computer Graphics Proceedings*, volume 22, pages 463–470, July 2003.

[54] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. In *IEEE Visualization '98 (VIS '98)*, pages 233–238, Washington - Brussels - Tokyo, October 1998. IEEE.

[55] M. Pauly and M. Gross. Spectral processing of Point-Sampled geometry. In S. Spencer, editor, *Proceedings of the Annual Computer Graphics Conference (SIGGRAPH-01)*, pages 379–386, New York, August 12–17 2001. ACM Press.

[56] M. Pauly, M. Gross, and L. P. Kobbelt. Efficient simplification of point-sampled surfaces. In R. Moorhead, M. Gross, and K. I. Joy, editors, *Proceedings of the 13th IEEE Visualization 2002 Conference (VIS-02)*, pages 163–170, Piscataway, NJ, October 27– November 1 2002. IEEE Computer Society.

[57] M. Pauly, R. Keiser, L. P. Kobbelt, and M. Gross. Shape Modeling with Point-Sampled Geometry. In J. Hart, editor, *Siggraph 2003, Computer Graphics Proceedings*, volume 22, pages 641–650, July 2003.

[58] R. N. Perry and S. F. Frisken. Kizamu: A system for sculpting digital characters. In *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 47–56, Los Angeles, CA, August 12–17 2001. ACM Press / ACM SIGGRAPH.

[59] H. Pfister, J. van Baar, M. Zwicker, and M. Gross. Surfels: Surface elements as rendering primitives. In S. Hoffmeyer, editor, *Proceedings of the Computer Graphics Conference 2000 (SIGGRAPH-00)*, pages 335–342, New York, July 23–28 2000. ACMPress.

[60] D. V. Pinskiy, E. S. Brugger, H. R. Childs, and B. Hamann. An octree-based multiresolution approach supporting interactive rendering of very large volume data sets. In H. R. Arabnia, R. F. Erbacher, X. He, C. Knight, B. Kovalerchuk, M. M. O. Lee, Y. Mun, M. Sarfraz, J. Schwing, and M. H. N. Tabrizi, editors, *Proceedings of The 2001 International Conference on Imaging Science, Systems, and Technology*, volume 1, pages 16–22, Athens, Georgia, 2001.

[61] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, England, second edition, 1992.

[62] J. Räsänen. Surface splatting: Theory, extensions and implementation. Master's thesis, Helsinki University of Technology, May 2002. URL:http://www.hybrid.fi/research/splat/thesis01.pdf.

[63] L. Ren, H. Pfister, and M. Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Eurographics 2002*, 2002.

[64] R. J. Renka. Multivariate interpolation of large sets of scattered data. *ACM Transactions on Mathematical Software*, 14(2):139–148, June 1988.

[65] J. Rossignac and P. Borrel. Multi-resolution 3D approximation for rendering complex scenes. In *Second Conference on Geometric Modelling in Computer Graphics*, pages 453–465, June 1993. Genova, Italy.

[66] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 343–352. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[67] S. Rusinkiewicz and M. Levoy. Streaming QSplat: a viewer for networked visualization of large, dense models. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pages 63–68, New York, NY, 2001. ACM Press.

[68] H. Samet. *The Design and Analysis of Spatial Data Structures*. Series in Computer Science. Addison-Wesley, Reading, MA, reprinted with corrections edition, April 1990.

[69] E. Shaffer and M. Garland. Efficient adaptive simplification of massive meshes. In T. Ertl, K. Joy, and A. Varshney, editors, *Proceedings of the Conference on Visualization 2001 (VIS-01)*, pages 127–134, Piscataway, NJ, October 21–26 2001. IEEE Computer Society.

[70] R. Shekhar, E. Fayyad, R. Yagel, and J. F. Cornhill. Octree-based decimation of marching cubes surfaces. In R. Yagel and G. M. Nielson, editors, *Proceedings of the Conference on Visualization*, pages 335–344, Los Alamitos, October 27–November 1 1996. IEEE.

[71] J. R. Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, May 1997. Available as Technical Report CMU-CS-97-137.

[72] P. Shirley. *Realistic Ray Tracing*. AK Peters Limited, first edition, 2000.

[73] R. Shu, C. Zhou, and M. S. Kankanhalli. Adaptive marching cubes. *The Visual Computer*, 11(4):202–217, 1995. ISSN 0178-2789.

[74] R. Sibson. A Brief Description of the Natural Neighbour Interpolant. Technical report, Math Department, U. of Bath, 1980.

[75] O. G. Staadt and M. H. Gross. Progressive tetrahedralizations. In D. Ebert, H. Hagen, and H. Rushmeier, editors, *Proceedings of IEEE Visualization '98*, pages 397–402, Research Triangle Park, NC, October 1998.

[76] N. Sukumar. Meshless Methods and Partition of Unity Finite Elements. In V. Brucato, editor, *Proceedings of the Sixth International ESAFORM Conference on Material Forming*, pages 603–606, Salerno, Italy, April 2003.

[77] I. J. Trotts, B. Hamann, and K. I. Joy. Simplification of tetrahedral meshes with error bounds.

IEEE Transactions on Visualization and Computer Graphics, 3(5):224–237, July/September 1999.

[78] G. H. Weber, B. Heckel, B. Hamann, and K. I. Joy. Procedural generation of triangulation-based visualizations. In A. Varshney, C. M. Wittenbrink, and H. Hagen, editors, *Proceedings of IEEE Visualization '99 (Late Breaking Hot Topics)*. IEEE, 10 1999.

[79] G. H. Weber, O. Kreylos, T. J. Ligocki, J. M. Shalf, H. Hagen, B. Hamann, and K. I. Joy. Extraction of crack-free isosurfaces from adaptive mesh refinement data. In D. S. Ebert, J. M. Favre, and R. Peikert, editors, *Data Visualization 2001 (Proceedings of "VisSym '01")*, pages 25–34, Vienna, Austria, 2001. Springer-Verlag.

[80] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of iso-surfaces. *The Visual Computer*, 15(2):100–111, 1999.

[81] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In E. Fiume, editor, *Siggraph 2001, Computer Graphics Proceedings*, pages 371–378. ACM Press / ACM SIGGRAPH, 2001.