CSTD-98-001

FAST ISOSURFACE EXTRACTION USING QUASI-MONTE CARLO METHODS

by

Xu Ji

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

The University of Utah

March 1998

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a thesis submitted by

Xu Ji

This thesis has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

Chair: Krzysztof Sikorski

Frank Stenger

Christopher Johnson

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the thesis of Xu Ji in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date

Krzysztof Sikorski
Chair, Supervisory Committee

Approved for the Major Department

Robert Kessler
Chair/Dean

Approved for the Graduate Council

Ann W. Hart
Dean of The Graduate School

ABSTRACT

A new algorithm for isosurface extraction is proposed and implemented. The algorithm is based on the new mathematical understanding of the theory of the quasi-Monte Carlo methods. Different from the general isosurface extracting methods, which work on the whole data set, this algorithm works on a subset of the original large three-dimensional data set, which is generated by the quasi-Monte Carlo method. The isosurface is generated on this subset data as an approximation to the isosurface generated from the whole data set. Hammersley, Halton and Hyperbolic Cross points are used as the quasi-Monte Carlo points in the implementation.

The results show that the QMC techniques enjoy a linear speedup with the number of QMC points. For large data sets, we usually can reduce the data size remarkbaly and still get a good representation of the original isosurface. The advantage of the techniques becomes more prominent when the data size gets larger. The QMC points generally generate visually better and smoother isosurfaces and these isosurfaces represent the overall shape of the original isosurfaces better than a regular subset of the original data.

The preprocessing of the QMC isosurface extraction might be time consuming. But this is a one-time process. After it is done, the postisosurface extraction is very fast.

To my family

CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Kris Sikorski, for his guidance and encouragement in my thesis research. He is always patient and always gives me valuable suggestions. I would also like to thank my committee member Dr. Chris Johnson, who allowed me to use the NOISE program and a program to generate 3D tedrahetrons from discrete points. I am in debt to my committee member Professor Frank Stenger for his review of my thesis.

Dr. Yarden Livnat helped me with a lot of questions, especially he explained many details of the NOISE algorithm that he designed.

I would like to specially thank Professor Gerard T. Schuster for letting me use his codes to generate the Hammersley and Halton points.

CHAPTER 1

INTRODUCTION

Scientific visualization has played an important role in investigating and understanding large data sets, and isosurface extraction and visualization is one of the most effective and powerful techniques for the investigation of three-dimensional scalar data sets. General isosurface extraction methods use the whole data set, which might be very large. Significant efforts have been undertaken in organizing data structures in order to speed up the process.

Recently, a mathematical breakthrough was achieved in understanding the tractability of multidimensional integrations using nearly-optimal quasi-Monte Carlo methods. These new methods have been successfully applied to exploration geophysics, financial integrals and computer graphics. Inspired by the new mathematical insights and its successful applications, we apply this method for extracting isosurface from large three-dimensional data sets.

In this chapter, we review the general isosurface extracting methods, data reduction methods, the quasi-Monte Carlo theory, and propose our procedure of using the quasi-Monte Carlo (QMC) methods for extracting isosurfaces.

1.1 The Problem of Isosurface Extraction

The problem of isosurface extraction from a volumetric data set can be formally stated as follows (Cignoni et al., 1996):

> A volumetric scalar data set is defined as a pair (V, W), where $V = \{v_i \in R^3, i = 1, ..., m\}$ is a finite set of points in the three-dimensional real space, and $W = \{w_i \in R, i = 1, ..., m\}$ is a corresponding set of scalar

values. Elements of V are sampling points in a domain $\Omega \subset R^3$, for a tri-variate scalar field $W = f(x, y, z)$, i.e., $w_i = f(v_i), i = 1, ..., m$.

A discrete model for volume data sets (V, W) is a pair (Σ, Φ), where Σ is a subdivision of Ω into cells $\sigma_1, ..., \sigma_n$, and Φ is a corresponding family of real tri-variate functions $\phi_j : \sigma_j \to R, j = 1, ..., n$, which interpolate the values of W at all points of V:

$$\forall i = 1...m, \forall j = 1...n, (v_i \in \sigma_j) \Rightarrow (\phi_j(v_i) = w_i).$$

If functions of Φ are coincident at the common boundary of adjacent cells of Σ, then a continuous function ϕ is defined piecewise by Φ on Σ:

$$\phi(p) = \phi_j(p) \quad if \quad p \in \sigma_j, \forall j = 1, ..., n.$$

Function ϕ is in practice an estimate of the measured function f over Ω.

Given $q \in R$, the set $S(q) = \{p \in \Omega \mid \phi(p) = q\}$ is called an isosurface of ϕ at value q. If ϕ is continuous, and q is not an extreme value of ϕ, then $S(q)$ is a 2-manifold embedded in R^3, possibly with a one-dimensional boundary contained in the boundary of Ω. $S(q)$ is defined piecewise on the cells of Σ: each cell $\sigma_j \in \Sigma$ such that $min_{\sigma_j}\phi_j \leq q \leq max_{\sigma_j}\phi_j$, ϕ_j is called active at q, and it contributes to $S(q)$ for a patch corresponding to the locus of points

$$S_j(q) = \{p \in \sigma_j \mid \phi_j(p) = q\}.$$

The isosurface extraction problem consists in approximating all patches $S_j(q)$ that correspond to active cells, given (Σ, Φ) and q.

The data set is often generated from three-dimensional images or simulation techniques, such as from finite difference or finite element methods. When the data set is very large, extracting an isosurface can be a large computational task. Rather than finding a global solution one can seek a local approximation within each cell in Σ. Hence, isosurface extraction becomes a two-stage process: locating the cells that intersect the isosurface, and then locally approximating the isosurface inside each such cell.

Traditional methods for isosurface extraction, such as the Marching Cube, analyze in turn every cell of Σ, and for each active cell compute the corresponding isosurface patch. More recent techniques are aimed at avoiding analyzing non-active cells.

1.2 Isosurface Extracting Methods

Early techniques for generating isosurfaces began as three-dimensional extensions to the image processing problem of finding two-dimensional contours. These approaches required substantial computational logic, and not all of them were foolproof. More recent approaches are more algorithmic, and can generally guarantee a correct solution.

We review different approaches to the general isosurface extraction problem.

1.2.1 Geometric Space Decomposition Methods

Originally, only structured grids were available as an underlying geometry. Structured grids impose order on the given cell set. This fact helps to keep the geometric complexity of the entire cell set in the geometry domain. By utilizing this order, methods based on the geometry of the data set could take advantage of the coherence between adjacent cells. Isosurface can be generated by searching over the geometric space. The isosurface extraction problem can be defined as follows (Livnat et al., 1995):

> **Geometric Search**: Given a point $q \in [min(W), max(W)]$ and a set of cells Σ in Ω, where each cell is associated with a set of values $\{q_j\} \in W$, find the subset of Σ which an isosurface of value q intersects.

1.2.1.1 Marching Cubes

Marching cubes algorithm, which was proposed by Lorensen and Cline (1987), was one of the first constructive algorithms. It processes one cell σ_i at a time, and generates its isosurface geometry immediately before moving to the next cell.

Its basic approach involves taking an l-vertex cube, looking at the scalar value at each vertex, and determining if and how an isosurface would pass through it. This algorithm has a complexity of $O(n)$, where n is the number of cells.

1.2.1.2 Octrees

Wilhems and Van Gelder (1992) proposed the use of a branch-on-need octree, a hierachical spatial decomposition data structure, to purge subvolumes while fitting isosurfaces. The hierarchical nature of the octree enables searching to only the lowest level nodes of the original data set containing at least one active cell. The method adopts a geometric approach. The domain is limited to the case of regular data sets or to curvilinear data sets defined on a warped regular grid.

Wilhelms and Gelder did not analyze the time complexity of the search phase of their algorithm. However, octree decompositions are known to be sensitive to the underlying data. If the underlying data contains some fluctuations or noise, most of the octree will have to be traversed. The octree algorithm has a worst case complexity of $O(k + k\log(n/k))$, where n is the total number of cells in the scalar field, and k is the number of active cells.

1.2.1.3 Extrema Graphs

Another geometric approach, based on the use of *extrema graph*, is proposed by Itoh and Koyamada (1994). The nodes of the graph are the cells of the volume which hold local extrema of the data values (local minima or maxima). Each edge of the graph keeps a list of the cells connecting its two end nodes. Given an isovalue, an active edge is searched for in the extrema graph. The cells connected to this edge are then sequentially scanned until an active one is found and, finally, a propagation method is activated on this cell. Knowing how the isosurface intersects the current cell enables the algorithm to move only to those neighbouring cells that are guaranteed to intersect the isosurface. At the end of the process, boundary cells need to be tested because of possibly isolated local extrema. The propagation algorithm adopted uses a FIFO queue to store the identifiers of the adjacent active cells to be visited, and marks those cells to avoid enqueuing them again. The algorithm can be applied both to regular and irregular data sets. The complexity of the algorithm is at best the size of the boundary list, which Itoh and Koyamada estimate as $O(n^{2/3})$.

Storage requirements for the extrema graph method can be high, since the propagation search requires four links from each cell to its neighbors in addition to the maximum and minimum values of its vertices. In addtion, the algorithm uses a queue during the propagation search, yet the maximum required size of the queue is unknown in advance.

1.2.2 Value Space Decomposition Methods

Another class of algorithms is based on interval search. The search is carried out on the interval set and, for each active interval, on the corresponding active cell. All such techniques apply to both regular and irregular data sets, but waste of memory due to the loss of implicit spatial information makes them more suitable for irregular data sets.

Efficient isosurface extraction for unstructured data set is more difficult, as no explicit order, i.e., position and shape, is imposed on the cells, only an implicit one that is difficult to utilize. Methods designed to work in this domain have to use additional explicit information or revert to search over the value space, W. The advantage of the latter approach is that one needs only to examine the minimum and maximum values of a cell to determine if an isosurface intersects that cell. Hence, the dimensionality of the problem reduces to two for scalar fields. For linear tetrahedron cells we are working with, the minimum and maximum values of a cell are located at the vertices.

These algorithms based on interval search can be stated formally as follows (Livnat et al., 1995):

Interval Search: Given a number $q \in [min(W), max(W)]$ and given a set of cells represented as intervals,

$$I = \{[a_i, b_i]\} \quad \text{such that,} \quad a_i, b_i \in W$$

find $I_q \subset I$ such that,

$$[a_i, b_i] \in I_q \text{ iff } a_i \leq q \leq b_i$$

Posing the search problem over intervals introduces some difficulties. If the intervals are of the same length or are mutually exclusive they can be organized in an efficient way suitable for quick queries. However, it is much less obvious how to organize an arbitrary set of intervals. Indeed, what distinguishes these methods from one another is the way they organize the intervals rather than how they perform searches.

Decomposing the value space, rather than the geometric space, has two advantages. First, the underlying geometric structure is of no importance, so this decomposition works well with unstructured grids. Second, for a scalar field in 3-D, the dimensionality of the search is reduced from three to only two.

1.2.2.1 The Active List

Giles and Haimes (1990) report an approach in which two sorted interval lists are constructed in a preprocessing phase by sorting the cells' minimum and maximum values. The global maximum range of each cell is also computed. Given an isovalue, and *active list* containing all the active intervals is created by referring to the two sorted lists. If the specified isovalue is changed by less than the global maximum range with respect to the previous one, then the active list is augmented with the intervals lying between the two isovalues. Only one of the original lists is used in this process, depending on the sign of the isovalue's change. The active list is then purged of all the cells that do not intersect the new isosurface. Only the cells corresponding to the intervals of the final active list are visited to extract the isosurface. Livnat et al. showed that the complexity of this algorithm is $O(n)$ in time.

1.2.2.2 The Span Filter

A key issue in isosurface extraction is the size of the data set. Gallagher (1991) addressed this issue by scanning the data set and generating a compressed representation suitable for isosurface extraction. The range of data values is divided into subranges, termed buckets. Each cell is then classified based on the bucket its minimum value resides in and on how many buckets the cells range spans i.e. the span of the cell. Cells are then grouped according to their span, and within each such group the cells are further grouped according to their starting bucket. In each such internal group, the representation is compressed according to a unique id assigned to each cell. Rather than requiring a span list for every possible span length, the method uses one span list to catch all the cells that span more than a predefined number of buckets.

The use of this perspective stresses the importance of the first division into buckets. The entire organization of the domain is controlled by only set of parameters, the position of the original buckets. While this may help to ensure even distribution in the first span, it does not provide control over the distribution of the cells in the other spans. Furthermore, this division is not automated and has to be crafted by trial and error for each new data set. Finally, the search algorithm has a complexity of $O(n)$ in time.

1.2.2.3 Sweeping Simplices

Shen and Johnson (1995) tried to overcome the limitations of span filter and active list, by defining the *sweeping simplices* algorithm in which both sorted lists of the active list algorithm, and the spatial decomposition into buckets of the intervals of the span filter algorithm are present.

Sweeping simplices uses two ordered cell lists, a sweep list and a min list. Each element in the sweep list contains a pointer to a cell, the cell's maximum value, and a flag. The sweep list is then sorted according the cell's maximum value. The min list contains the minimum value for each cell as well as a pointer to the corresponding element in the sweep list and is ordered by the minimum values. The initialization step requires a time of $O(n \log(n))$.

Given an isovalue, the sweeping simplices algorithm marks all the cells that have a minimum value less than the given isovalue using the min list by setting the corresponding flag in the sweep list. If an isovalue was previously given, then the min list is traversed between the previous isovalue and the new one. The corresponding flags in the sweep list are then set or reset based on whether the new isovalue is greater or smaller than the previous isovalue.

Once the flags are changed, the sweep list is traversed starting at the first cell with a maximum value greater than the new isovalue. The cells that intersect the isosurface are those cells for which their corresponding flag is set. The complexity of the algorithm is $O(n)$ in both time and space.

The sweeping simplices algorithm uses a hierarchical data decomposition. At the lowest level, the range of data values is subdivided into several subgroups. Other levels are created recursively by grouping consecutive pairs from the previous level. At the top level there exists a single subgroup with the range as the entire data set. The cells are then associated with the smallest subgroup that contains the cell. Each subgroup is then associated with a minimum and sweep list as described before. Isosurface extraction is accomplished by selecting for each level the subgroup that contains the given isovalue and performing the search using its minimum and sweep lists.

1.2.2.4 NOISE

A common obstacle for all the interval methods was that the intervals were ordered according to either their maximum or their minimum value. Both the sweep algorithm and the minimum-maximum attempted to tackle this issue by maintaining two lists of the intervals, ordered by the maximum and minimum values. What was missing, however, was a way to combine these two lists into a single list.

Livnat et al. (1995) proposed a new algorithm, which uses span space as the underlying domain, and employs a kd-tree as a means for simultaneous ordering the cells according to their maximum and minimum values.

This algorithm views the isosurface extraction problem not as a search over intervals in W but rather as a search over points in W^2. Livnat et al. (1995) define the span space as follows:

The Span Space: Let Σ be a given set of cells, define a set of points $P = \{p_i\}$ over W^2 such that,

$$\forall c_i \in \Sigma \quad \text{associate,} \quad p_i = (a_i, b_i),$$

where $a_i = min_j\{q_j\}_i, b_i = max_j\{q_j\}_i$ and $\{q_j\}$ are the values of the scalar field f at vertices of cell c_i.

Using this augmented definition, the isosurface extraction problem can be stated as follows (Livnat et al., 1995):

The Span Search: Given a set of cells Σ, and its associated set of points P, in the span space, and given a value $q \in [min(W), max(W)]$, find the subset $P_q \subseteq P$, such that

$$\forall [x_i, y_i] \in P_q, x_i \leq q \;\; and \;\; y_i \geq q.$$

This algorithm has a worst case complexity of $O(\sqrt{n}+k)$ to locate the cells that are intersected by the isosurface, where n is the total number of cells in the scalar field, and k is the number of isosurface cells. This algorithm is nearly optimal in the sense that for the typical case, $k > \sqrt{n}$, it's asymptotic cost is $O(k)$, and every algorithm has to cost at least $O(k)$. The algorithm performs well for large and small data sets and for any size of isosurface. The number of cells that intersect an isosurface can also be found in $O(\sqrt{n})$ time, which enables fast rough estimates of the surface area and the corresponding volume encompassed by the isosurface.

1.3 Data Reduction Methods

For the problems of isosurface extraction, when the data set is very large, the cost of computation might be huge, and the isosurface extraction becomes intractable. Traditional isosurface extracting methods like Marching cubes search every cell in the scalar field, more recent methods put efforts in reorganizing the data structures, either in the geometric space or in the value space, so only those active cells are verified. But all of these approaches work on the whole data set, the difference is in using different ways for extracting those active cells.

Reducing data set size with preserved information content is an important research problem. By reducing data size, reduction in computation and memory requirement can be realized, which results in better interactive response. It is important to be able to reduce the data size in order to visualize important features of a large data set. Such techniques can be used to reduce image clutter and improve the effectiveness of the visualization.

Various methods have been developed to reduce large data sets. In this section we review some of these methods.

1.3.1 Geometry Extraction

Geometry extraction selects data based on geometric or topological characteristics. A common extraction technique selects a set of points and cells that lie within a specified range. This method has been used frequently in finite element analysis to isolate the visualization to just a few key regions.

Another useful technique called spatial extraction, selects data set structure and associated data attributes lying within a specified region in space. For example, data may be selected within a given sphere.

Subsampling is a method that reduces data size by selecting a subset of the original data. The subset is specified by choosing a parameter k, specifying that every k^{th} data point is to be extracted. Subsampling is not typically performed on unstructured data because of its inherent complexity. A related technique is called data masking, which selects every k^{th} cell instead of every k^{th} point.

1.3.2 Thresholding

Data thresholding selects data based on the values of data set attributes. For example, one can select all points with a velocity magnitude greater than 1.0.

1.3.3 Probing

Probing is a method that obtains data set attributes by sampling one data set (the input) with a set of points (the probe). Probing is also called resampling. The result of the probing is a new data set (the output) with the topological and geometric structure of the probed data set, and point attributes interpolated from the input data set. Once the probing operation is completed, the output data set can be visualized with any of the appropriate techniques.

One important application of probing converts irregular or unstructured data to structured from using a volume of appropriate resolution as a probe to sample the unstructured data. This is useful if we use volume rendering or other volume visualization techniques to view the data.

1.3.4 Decimation

A decimation algorithm can be used after the isosurface has been extracted. The goal of decimation algorithm is to reduce the total number of triangles in a triangular mesh, preserving the original topology and forming a good approximation to the original geometry. It is related to the subsampling techniques for unstructured meshes. The differences are that decimation treats only triangular meshes, and the choice of which point to delete is a function of a decimation criterion, also the triangulation of the hole created by deleting the point is carried out in a way as to preserve edges or other important features.

One of the decimation algorithms is polygon reduction technique. This technique reduces the number of polygons required to model an object.

1.4 Quasi-Monte Carlo Methods

In general terms, the Monte Carlo method may be described as a numerical method based on random sampling. A polish mathematician, Stanislaw Ulam, was the first to develop and apply this method in large scale nuclear simulations (S. Ulam, 1976, 1991). Since the 1940s, the Monte Carlo methods have been widely used for solving multidimensional problems in various branches of science.

It is well known that, for a given error tolerance, the computational work required to numerically evaluate a d-dimensional integral for integrands with bounded derivatives is on the order $(1/\varepsilon)^d$ with using of a regular grid discretization. The integral becomes practically intractable when the dimension d is large. In the classical Monte Carlo methods, the discretization points are randomly selected, and the computational work is of the order $(1/\varepsilon)^2$, which is independent of d. This makes multivariate integration for large d tractable. However The Monte Carlo method has some disadvantages. For numerical integration, first, it yields only a probabilistic bound on the integration error; second, it is not so much the true randomness of the samples that is relevant, but rather that the samples should be spread in a uniform manner over the integration domain. A determinisitic error bound can be established if deterministic nodes are used.

Quasi-Monte Carlo methods can be succinctly described as deterministic versions of Monte Carlo methods. The basic idea of it is to replace random samples in a Monte Carlo method by well-chosen deterministic points. The main aim is to select deterministic points for which the deterministic error bound is smaller than or comparable to the probabilistic Monte Carlo error bound. It is proved that quasi-Monte Carlo method with judiciously chosen deterministic points usually leads to a faster rate of convergence than a corresponding Monte Carlo method.

Recently, a theoretical breakthrough was achieved in the understanding of the computational complexity of several quasi-Monte Carlo methods for multivariate integration. Wozniakowski (1991) proved that some of the intractable problems can be made tractable by a quasi-Monte Carlo approach. One of such problems is integration; another is surface reconstruction, in which discrete information is used to approximate a multivariate function, a technique that is the basis for medical imaging and many other applications.

Wozniakowski (1991) showed that the discretization based on the shifted Hammersley points is nearly-optimal for the integration problem in the *average case* setting, with the computational work $H(\varepsilon) = O(\frac{1}{\varepsilon}[log(\frac{1}{\varepsilon})]^q)$, where $q = (\frac{d-1}{2})$. This means that to guarantee the expected error of the integration to be at most ε (with respect to a large class of probability measures) one must use at least $H(\varepsilon)$ evaluations, no matter what discretization points are chosen. It has also been shown that for the Hammersley points and for the related Halton points the exponent q is respectively $d-1$ and d.

Temlyakov (1987, 1991, 1993) introduced the hyperbolic cross points. He proved that hyperbolic cross points are nearly optimal sampling points for multivariate function approximation in the average case setting.

1.5 Isosurface Extraction Using Quasi-Monte Carlo Methods

G. Schuster and K. Sikorski (1991, 1997) applied the quasi-Monte Carlo method for the three-dimensional migration in their geophysical research, and the results showed great advantage of this approach. The quasi-Monte Carlo methods have also been applied to financial integrals (S. Paskov and J. Traub, 1997), computer graphics (A. Keller, 1997) and other research areas. Niederreiter (1992) discusses the quasi-Monte Carlo methods and their applications. Inspired by the new insight and understanding of the quasi-Monte Carlo method, we apply this method in the three-dimensional isosurface extraction.

As we discussed in previous sections, isosurface extraction becomes difficult when data sizes get very large. Various algorithms have been developed to reduce the data size for fast isosurface extraction. In this thesis, we propose and implement a new isosurface extracting method, which takes a different approach from all previous methods. The idea is similar to some of the data reduction algorithms, instead of working on the whole data set, we choose a subset of data from the original large data set, and then use this subset to construct an approximation of the original isosurface.

The new algorithm uses the quasi-Monte Carlo methods to generate a set of quasi-Monte Carlo points as our resampling points. Then the original large data sets are interpolated or directly subsampled to these quasi-Monte Carlo data sets. This new subset is then used in the isosurface extraction.

The new algorithm takes the following steps:

1. **Preprocessing:**

   (a) Decide on how many quasi random points we want to use, and generate them by a specific quasi-Monte Carlo method.

(b) Interpolate or directly subsample the scalar value into the three-dimensional quasi-Monte Carlo points.

(c) Generate a new geometric representation based on these three-dimensional points.

2. **Isosurface extraction:**

   - Use either of the previous methods to extract the desired isosurface from the new data set.

This algorithm is applicable for both structured and unstructured data, and also for time dependant data. In this thesis, three types of quasi-Monte Carlo points: Hammersley, Halton and Hyperbolic cross points are used to verify the performance of the algorithm.

As only a subset of the original large data set is used, the cost of the isosurface extraction can be reduced remarkably. Reducing the large data sets into smaller ones by using quasi-Monte Carlo methods allows us to interactively render the approximate isosurface. Though the preprocessing might not be trivial, but once it is completed, the isosurface construction becomes much less costly as compared to the original data set.

The results show that the QMC techniques enjoy a linear speedup with the decreasing number of QMC points. For large data sets, we usually can reduce the data size remarkbaly and still get a good representation of the original isosurface. The advantage of the techniques becomes more prominent when the data size gets larger. The QMC points generally generate visually better and smoother isosurfaces and these isosurfaces represent the overall shape of the original isosurfaces better than a regular subset of the original data. The differences among different QMC points are not prominent and depend on the data sets.

This thesis is organized into five chapters. Chapter 2 discusses the quasi-Monte Carlo points generations. Chapter 3 applies the quasi-Monte Carlo isosurface extraction techniques to some structured data sets. Chapter 4 presents the application of the techniques to unstructured data sets. Finally, Chapter 5 contains the conclusions.

CHAPTER 2

QUASI-MONTE CARLO POINTS GENERATION

Three types of quasi-Monte Carlo points are used in this thesis - Hammersley, Halton and Hyperbolic Cross points. This chapter discusses the algorithms for generating these points.

2.1 Hammersley and Halton Points

The Hammersley and Halton points can be generated in a similar way. Let $p_1,\ p_2,\ ...,\ p_d$ be the first d prime numbers, where d is the dimension. Any integer $k > 0$ can uniquely be represented by:

$$k = \sum_{i=1}^{\lceil \log k \rceil} a_i p_j^i$$

with $a_i \in [0, p_j - 1]$.

The radical inverse function Φ_{p_j} is given by

$$\Phi_{p_j}(k) = \sum_{i=1}^{\lceil \log k \rceil} a_i p_j^{-(i+1)}.$$

We then define the sequence of vectors u_k for $k = 1,\ 2,\ ...,\ M - 1$ as

$$u_k^{d-1} = [\Phi_{p_1}(k), \Phi_{p_2}(k), ..., \Phi_{p_{d-1}}(k)]$$

with $M = (p_1 p_2 ... p_{d-1})^{\lceil \log d \rceil}$.

It is proved (Wozniakowski, 1991) that there exists a real number t^* such that the optimal on the average n sampling points in the unit cube $[0,1]^d$ are given by

$$x_k = \bar{1} - z_k$$

where $\bar{1} = (1, ..., 1)^T$ and

$$z_k = ((k + t^*)/n, u_k^{d-1}), k = 1, 2, ..., n.$$

For $t^* = 0$ the z_k's are the classical Hammersley points.

The Halton points z_k's are defined as

$$z_k = (u_k^d), k = 1, 2, ..., n.$$

Figure 2.1 shows an example of the three-dimensional Hammersley points, and Figure 2.2 shows an example of the three-dimensional Halton points.

2.2 Hyperbolic Cross Points

Hyperbolic Cross points were introduced by Temlyakov (1987, 1991) for periodic functions. These points are defined as a subset of grid points whose indices satisfy a *hyperbolic* inequality. Temlyakov proved that the hyperbolic cross points are nearly optimal sampling points for multivariate function approximation in the average case setting. Wozniakowski (1992) extended it to the nonperiodic functions. He proved that optimal or nearly optimal sampling points for the approximation problem can be derived from Hyperbolic Cross points. He also exhibited nearly optimal algorithms for multivariate function approximation.

Let us define a vector

$$\vec{r} = [r_1, ..., r_d]$$

for $j = 1, ..., d$, where d is the number of dimensions and r_j is the regularity of the sampled function in j^{th} dimension.

For given integer m, the *hyperbolic cross* X_n of sample points is given by

$$X_n = \{(\frac{l_1}{2^{s_1+3}}, ..., \frac{l_d}{2^{s_d+3}})\}, \tag{2.1}$$

where $l_j = 1, 2, ..., 2^{s_j+3} - 1,\ \ j = 1, 2, ..., d$, and $\vec{s} \in A$, where A is the set of all nonnegative integers $\vec{s}$ for which the following condition is satisfied:

$$\sum_{j=1}^{d} (r_j + 1)s_j \leq m(r_{min} + 1),$$

where $r_{min} = min(r_j), j = 1, ..., d$.

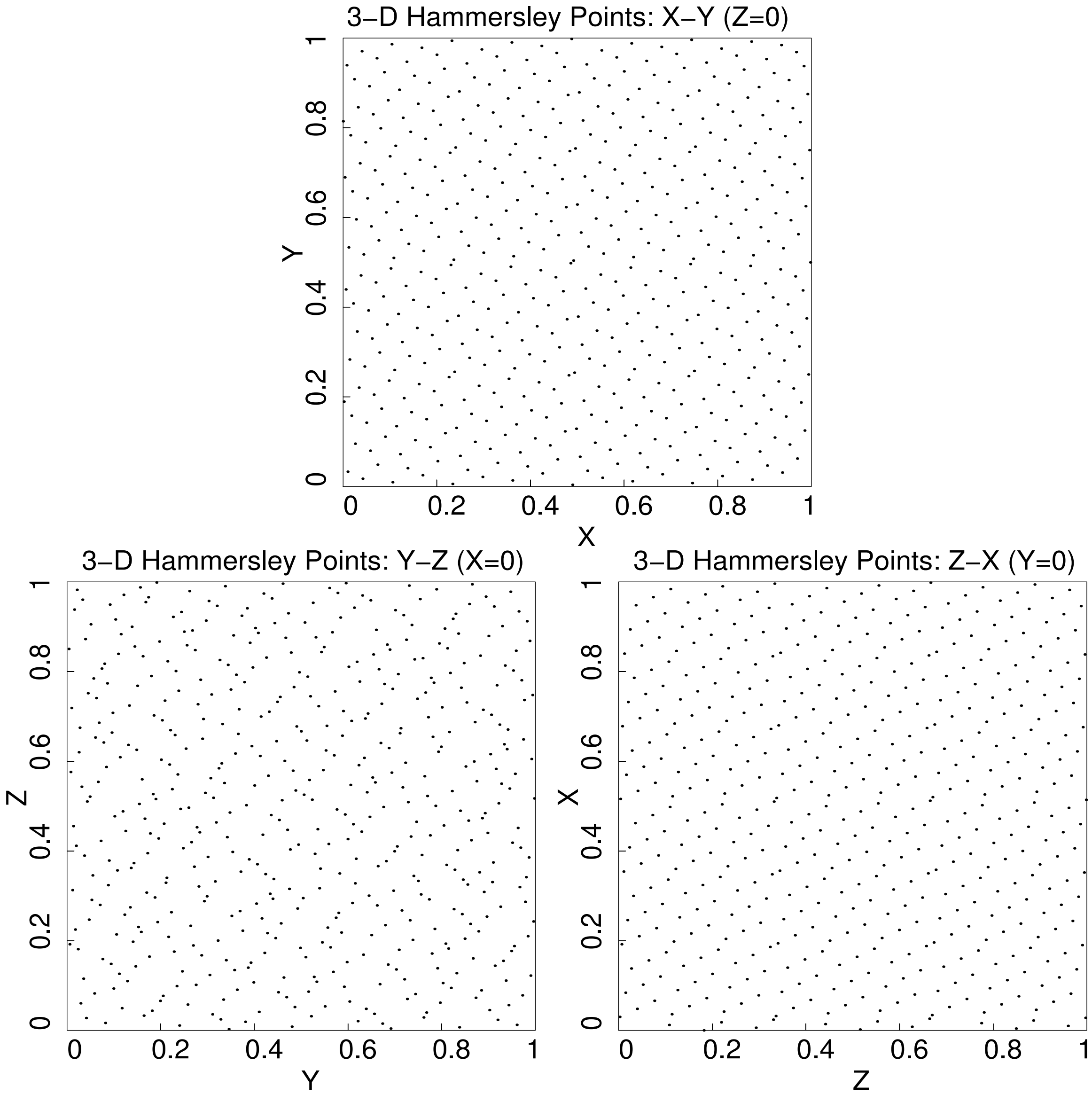

Figure 2.1. Three-dimensional Hammersley points. The total number of points is 500.

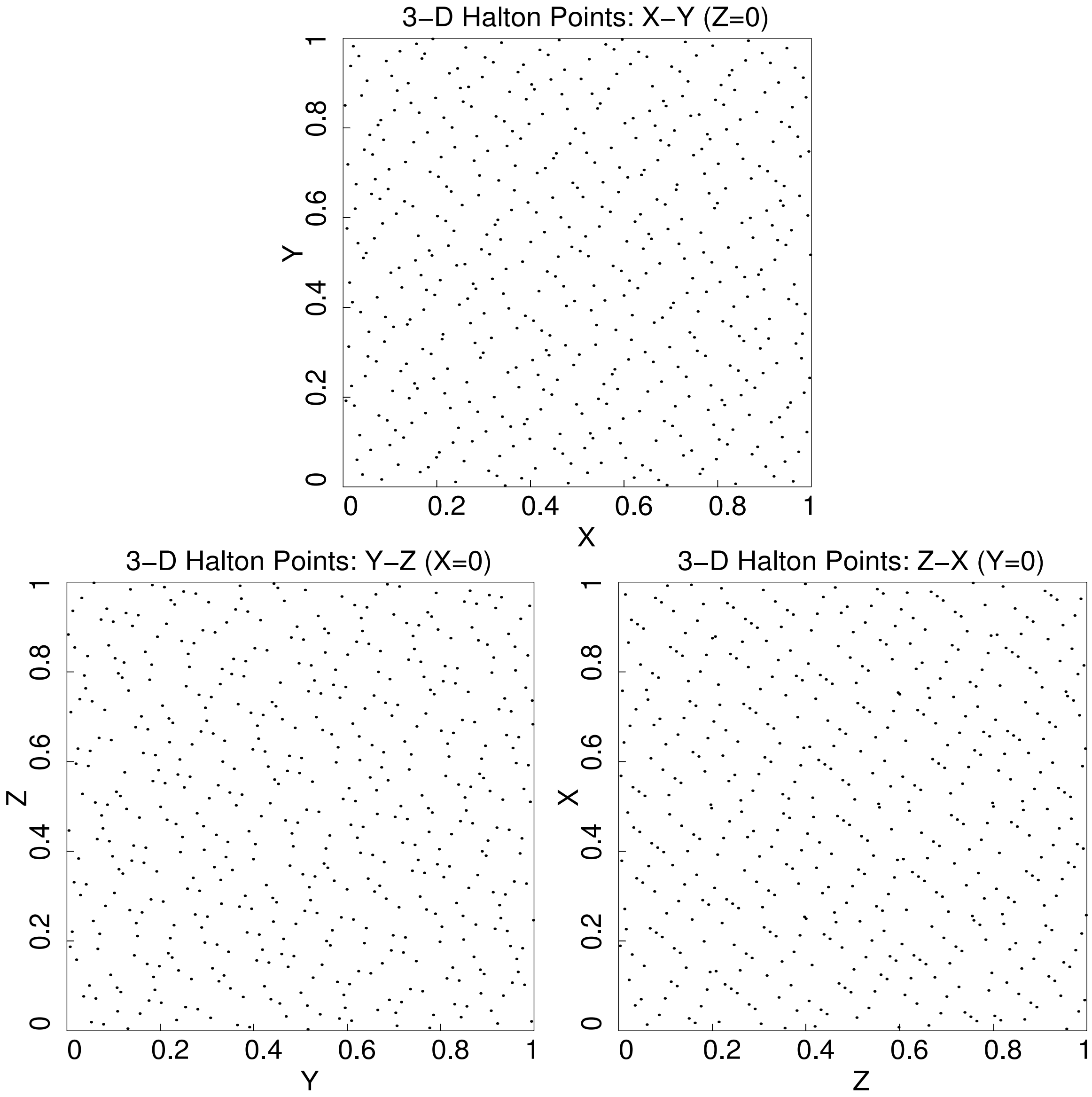

Figure 2.2. Three-dimensional Halton points. The total number of points is 500.

Here $n = \mathrm{card}(X_n)$. For the three-dimensional case,

$$n = \Theta(2^m m^2)$$

Choice for m depends on the size of the original data set. For smaller size (say $\leq 10^5$ cells), a suitable choice for m may be $2 \leq m \leq 5$. Whenever for large sizes 10^9 and more, we could select m in the range $[10, 18]$ or even larger. We stress that the total number of Hyperbolic Cross points can not be arbitrarily chosen. To generate more "flexible" set of points, we slightly modified the generation algorithm. In Equation 2.1, when choosing $l_j = 1, 2, ..., 2^{s_j+3}-1$, the same pattern repeats 8 times in each dimension. Now we change it to $l_j = 1, 2, ..., 2^{s_j+k} - 1$, where $2 \leq k \leq 5$. This change allows the same pattern to repeat 4 to 32 times as needed.

Figure 2.3 shows an example of the 3-D Hyperbolic Cross points for $m = 2$ and $r_j = 1, j = 1, 2, 3$.

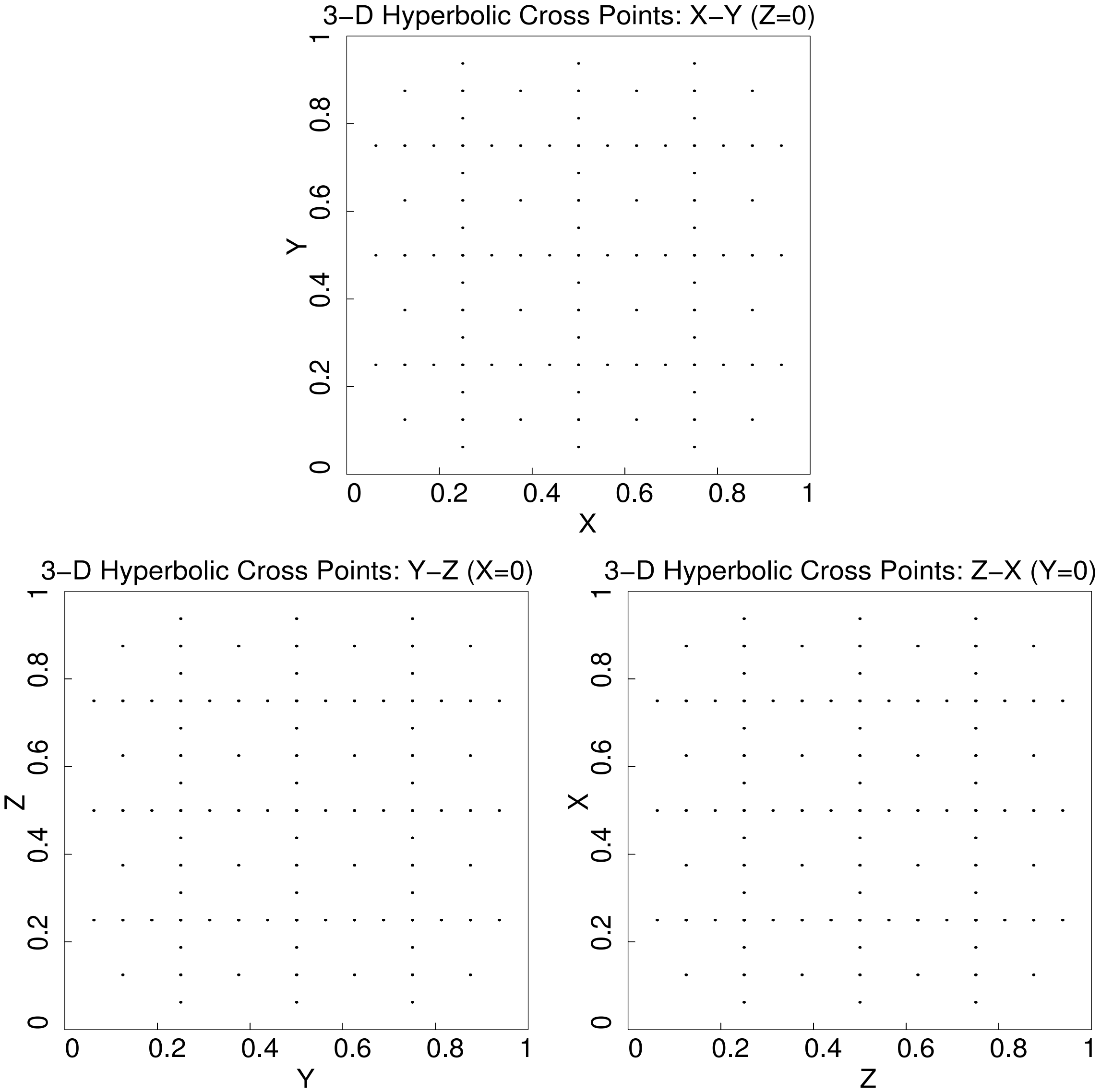

Figure 2.3. Three-dimensional Hyperbolic Cross points for $m = 2$ and $r_j = 1, j = 1, 2, 3$. The total number of points is 495.

CHAPTER 3

QUASI-MONTE CARLO ISOSURFACE EXTRACTION FOR STRUCTURED GRIDS

Structured data is one of the main types in scientific visualization. Such data sets are usually generated by finite-difference or other numerical techniques. In this chapter, we apply the quasi-Monte Carlo (QMC) isosurface extraction technique to structured data sets. We assume the data sets are in the Cartesian coordinates. For structured data sets in other coordinates, a coordinate transformation can be utilized first, and then the same technique can be applied.

The QMC isosurface extraction comprises of the following steps :

- QMC points generation: generate specified number of quasi-Monte Carlo points, which will be used as the subdata set to extract the isosurfaces.
- Mesh generation: generate new geometry representation (tetrahedrons) based on the QMC points.
- Interpolation/subsampling: interpolate or subsample function values from the original data set into the QMC points.
- Isosurface extraction: extract the isosurfaces from the QMC data sets.

After the QMC data reduction, the original structured data becomes unstructured.

We use three types of QMC points in our implementation: Hammersley, Halton and Hyperbolic Cross points. The algorithms generating these points are presented in Chapter 2. The isosurface extraction algorithm used (NOISE algorithm) is presented in Chapter 1. The Delauny algorithm is used in the mesh generation.

To test the new QMC isosurface extraction technique, we apply it to two structured data sets. Specificaly, we investigate the following questions:

- How essential is the interpolation technique?
- How efficient are different QMC point sets?
- How effective is this data reduction technique?

This chapter is intended to answer these questions. Specifically, we apply the QMC technique to various structured data sets by using different interpolation techniques, different QMC points, and different data reduction scales. We then compare the results and draw conclusions.

3.1 Data Sets

Two structured data sets are used for testing. Some information about these data sets is shown in Table 3.1.

The SPHERE is a synthetic data set of size $50 \times 50 \times 50$. The function values are calculated by the following quadratic function:

$$f(x, y, z) = x^2 + y^2 + z^2 - 0.4^2$$

The second data set SEIS is from 3-D finite-difference seismic simulation. The model used is a real model for the Salt Lake Basin. The data recorded are the particle velocity, and the data size is $50 \times 50 \times 50$. Both data sets contain 705894 tetrahedron cells.

The isosurfaces of the two data sets are shown in Figure 3.1.

Table 3.1 Data sets.

Name	Source	Type	SIZE	Cells
SPHERE	SYN	S-grid	$50 \times 50 \times 50$	705894
SEIS	FD	S-grid	$50 \times 50 \times 50$	705894

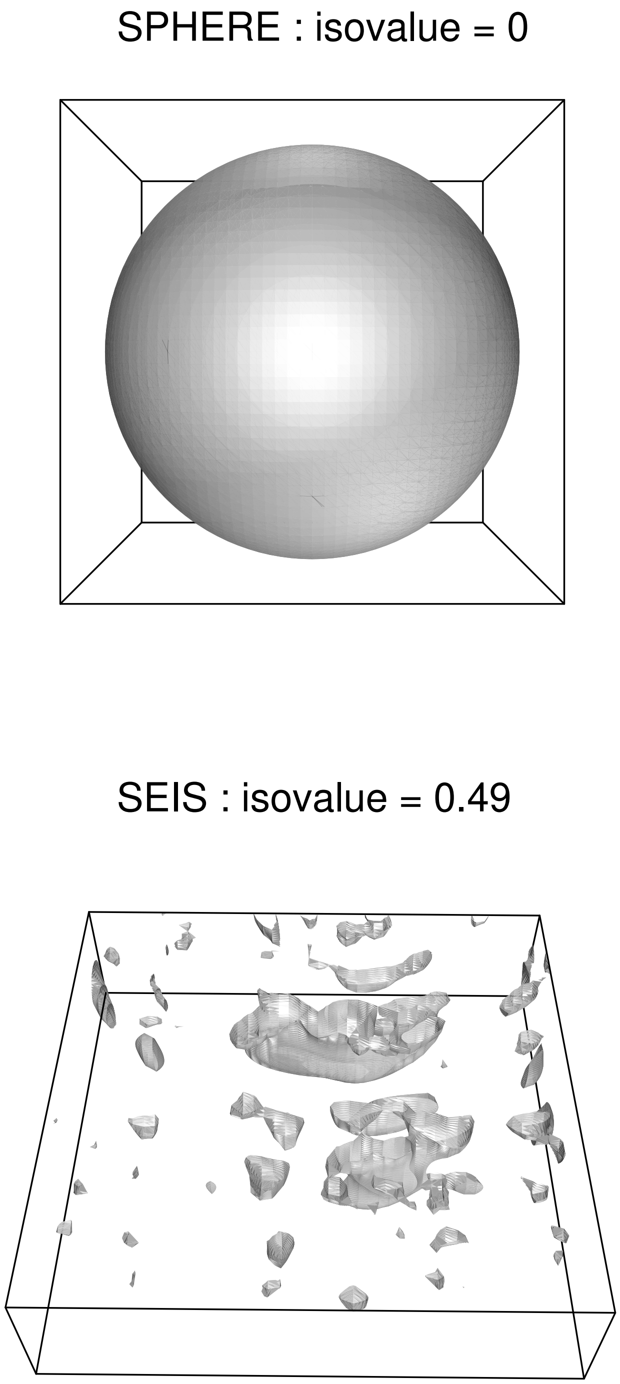

Figure 3.1. Isosurfaces of some structured data sets.

3.2 Interpolation on the Structured Grids

After the QMC points are generated, the function values need to be interpolated into these QMC points before extracting the isosurfaces. The interpolation techniques might be crucial. If a low precision interpolation is used, the errors of the interpolated function value are large, then the extracted isosurfaces are not smooth, and might also lose some information. But usually higher order interpolation is more costly. A lower order interpolation that provides enough precision, and hence saves computation time, would be most useful.

The interpolation for structured grids is relatively simple. It includes the following two steps:

- Search for the cube in which the QMC point is located.
- Interpolate function value to the QMC point inside the cube.

In the second step, two types of interpolation techniques are tested:

- Linear interpolation.
- Cubic polynomial interpolation.

3.2.1 Search in Structured Grids

For structured data, each point can be referenced by the indices (i, j, k). Similiarly, we can reference a cube in the structured grids by the indices of the vertex that has the minimum x, y and z coordinates.

The search for a cube in which the QMC point is located is straightforward for the structured grids. The indices of a cube which contains a QMC point $q = (q_x, q_y, q_z)$ can be calculated as follows:

$$
\begin{aligned}
i &= \lfloor q_x/hx \rfloor, \\
j &= \lfloor q_y/hy \rfloor, \\
k &= \lfloor q_z/hz \rfloor,
\end{aligned}
$$

where (i, j, k) are the indices of the cube, and hx, hy, hz are the grid spacing in each dimension.

3.2.2 Linear Interpolation

The tri-linear interpolation in three dimensions is the simplest interpolation technique. To illustrate, we first give the formula of linear interpolation in one dimension.

The linear interpolation in one dimension (say X) can be calculated as follows:

$$\begin{aligned} \lambda &= \frac{x - x_i}{x_{i+1} - x_i} \\ s_1(x) &= \lambda(x) f_i + (1 - \lambda(x)) f_{i+1}, \end{aligned}$$

where x_i and f_i are the X-coordinate and the function value of the data point, x and $s_1(x)$ are the X-coordinate and the interpolated function value at the QMC point. Apparently we should have $x_i \leq x \leq x_{i+1}$.

The tri-linear interpolation is simply by applying the above formula to each of the X-, Y- and Z-directions.

3.2.3 Cubic Polynomial Interpolation

The cubic polynomial interpolation uses a third order piece-wise polynomial to interpolate the function values. This interpolation technique is widely used in various applications.

Consider a 1-D tabulated function $y_i = y(x_i), i = 1, ..., n$, and concentrate on one particular interval $x_i \leq x \leq x_{i+1}$. The cubic polynomial interpolation is given by

$$s_3(x) = a(x) y_i + b(x) y_{i+1} + c(x) y_i^{''} + d(x) y_{i+1}^{''},$$

where $y_i^{''}$ is the second derivative of the function $y(x)$ at x_i, and can be calculated by

$$y_i^{''} = \frac{y_{i+1}^{'} + y_{i-1}^{'} - 2y_i^{'}}{2h}$$

$$= \frac{y_{i+2} - 4y_{i+1} + 6y_i - 4y_{i-1} + y_{i-2}}{4h^2}.$$

The coefficients $a(x), b(x), c(x)$ and $d(x)$ are defined as

$$\begin{aligned}
a(x) &= \frac{x_{i+1} - x}{x_{i+1} - x_i}, \\
b(x) &= 1 - a = \frac{x - x_i}{x_{i+1} - x_i}, \\
c(x) &= \frac{1}{6}(a^3 - a)h^2, \\
d(x) &= \frac{1}{6}(b^3 - b)h^2,
\end{aligned}$$

where $h = x_{i+1} - x_i$ is the grid spacing.

The cubic polynomial interpolation in three dimensions is simply applying the above formula to each of the X-, Y- and Z-dimensions.

3.2.4 Numerical Results

In this section, we apply the above two interpolation techniques to two data sets, and compare the results. The Hammersley points are used in the computation.

Figures 3.2 and 3.3 show the results by applying linear and cubic polynomial interpolation to two structured data sets. The number of QMC points is 1/4 that of the original data size for both data sets for Figure 3.2, and 1/16 for Figure 3.3. The left columns are for the linear interpolation, and the right columns are for the cubic polynomial interpolation. Table 3.2 shows the performance of the two interpolation techniques on two data sets for using 1/4 of the original data. The numbers shown in the table are the running time in seconds as implemented on IBM RS6000 with 512 MB of virtual memory.

Table 3.2 Interpolation performance

(Running time in seconds).

	SPHERE	SEIS
Linear Interpolation	30.23	30.98
Cubic Spline Interpolation	40.14	41.08
Mesh Generation (DELAUNY)	1833.3	2029.26
Isosurface Extraction (NOISE)	25.67	24.80

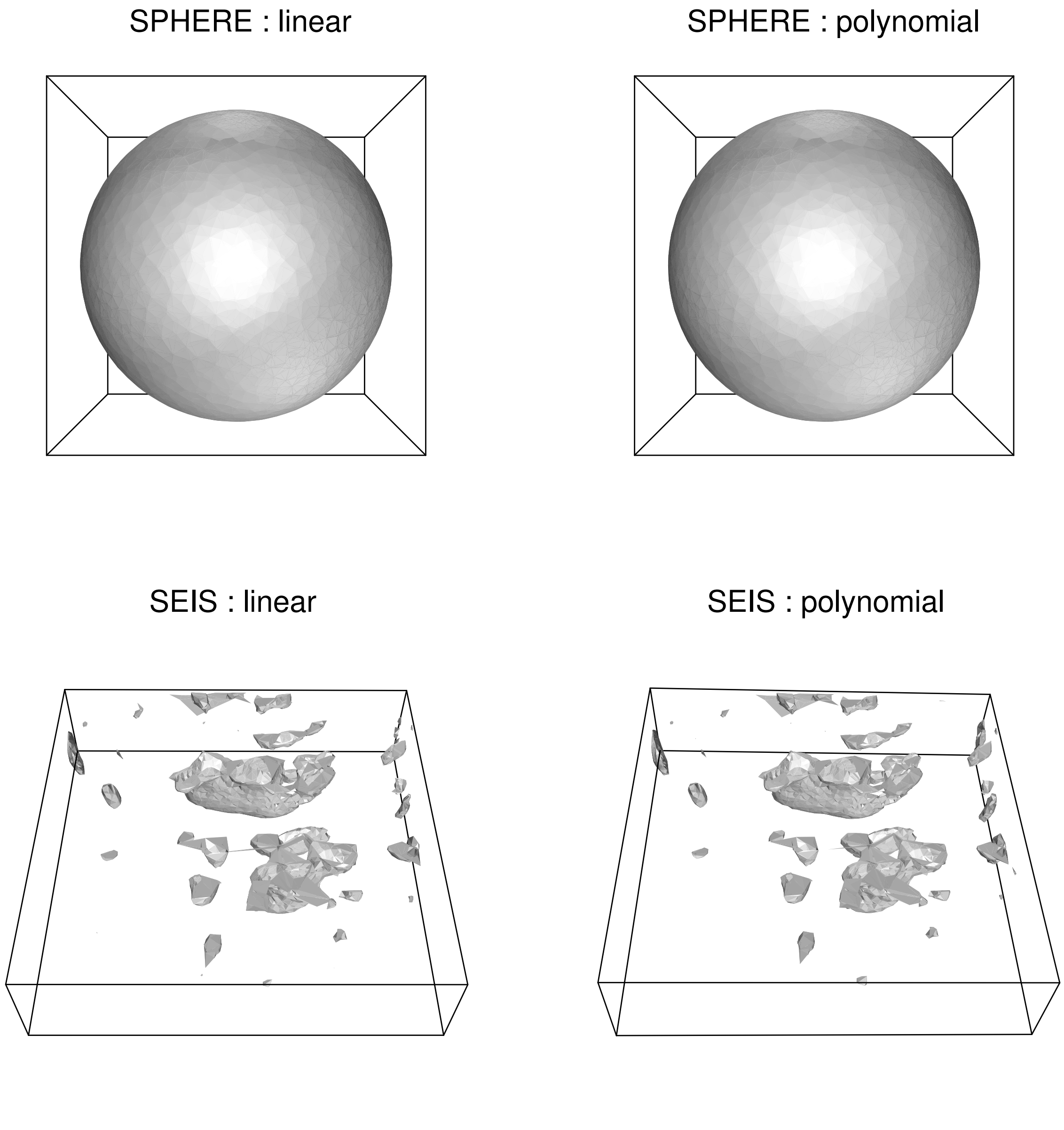

Figure 3.2. Comparisons between linear and cubic polynomial interpolations. One-fourth of the original data size is used for both data sets.

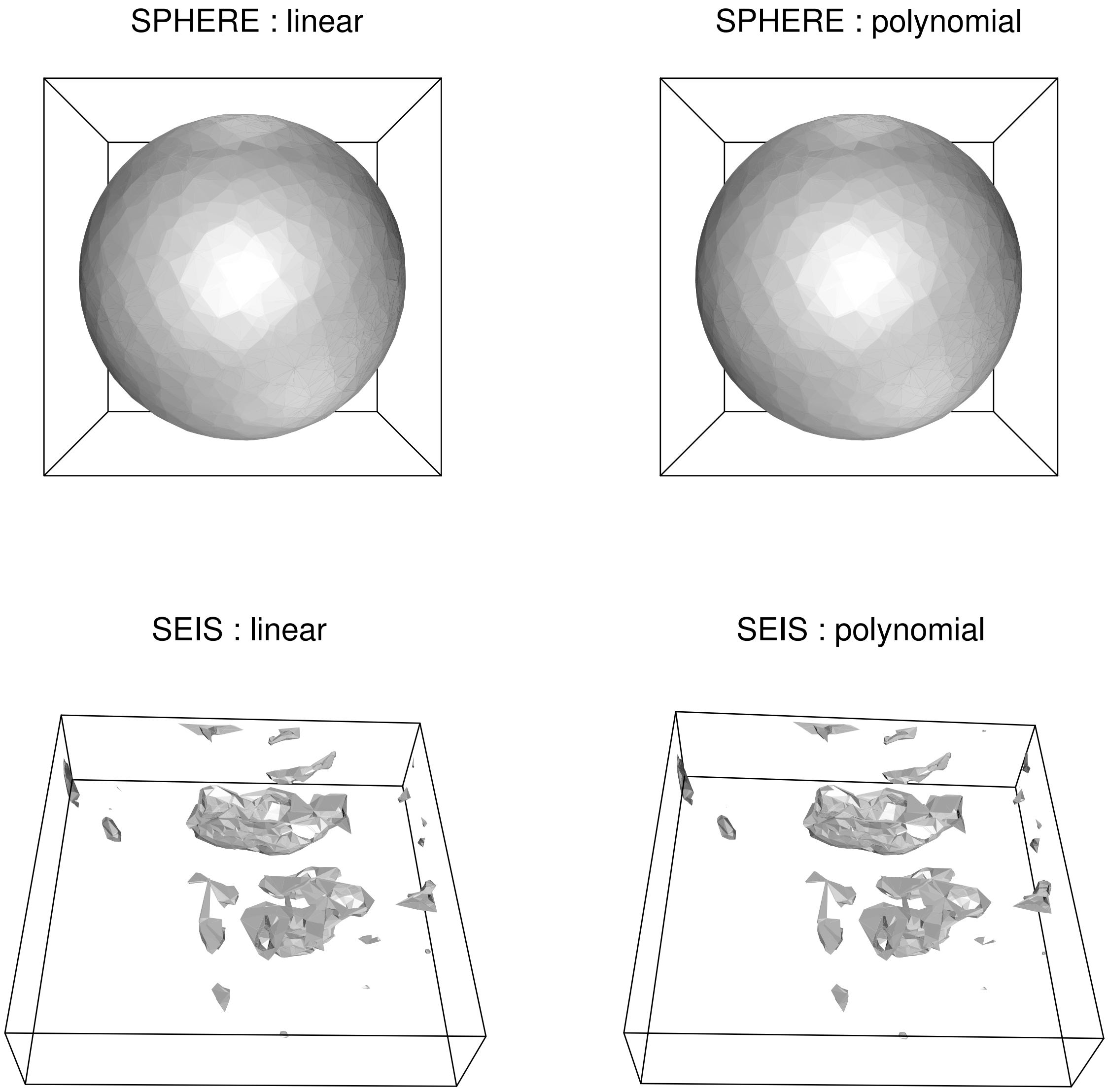

Figure 3.3. Comparisons between linear and cubic polynomial interpolations. One-sixteenth (1/16) of the original data size is used for both data sets.

The cubic polynomial interpolation is more accurate interpolation, which gives more smooth isosurfaces. But the difference between the appearances of the images is very small for both data sets. The explanation for this is that we are working with linear tetrahedral cells. In this case the function values can be approximated by using the linear formula:

$$f(x, y, z) = ax + by + cz + d,$$

where a, b, c, and d are the coefficients of the polynomial, and x, y and z are the global coordinates. The coefficients a, b, c, and d can be calculated by plugging in the function value and coordinates of the four tetrahedron vertices into the above formula and solving the four linear equations. For linear tetrahedral cells, linear interpolation reaches the precision given in the original data.

In Table 3.2, we listed the running time needed for the interpolations. The interpolation time in the table included the cube searching time. As expected, the cubic polynomial interpolation takes a little more time than the linear interpolation. Because the cube searching almost costs no time, the running time for the interpolation increases linearly with the number of QMC points. These results will also be seen in the following sections.

Also shown in Table 3.2, the time spent in interpolation is just a little more than that spent in the isosurface extration, but the mesh generation takes a lot more time.

Comparing the results, we draw the conclusion that linear interpolation is good enough for the case of linear tetrahedral cells, and higher order cubic polynomial interpolation does not yield much improvement (though it might be better for larger compression ratios). Therefore, the rest of the computations in this chapter are employing linear interpolation, unless mentioned otherwise.

3.3 QMC and Regular Points Comparisons

We use three types of QMC points in our computation, Hammersley, Halton and Hyperbolic Cross points. Also as discussed in Chapter 1, some of the data reduction techniques reduce the data by taking a regular subset of the original data. For the structured data sets, this might be done, for example, by taking every other point in each dimension. As discussed in Chapter 2, the Hyperbolic Cross points are a subset of the regular grid points. We can also take advantage of this property to subsample the original data instead of doing interpolations.

This section describes the QMC and regular points comparisons, and also compares the subsamping with interpolation for the Hyperbolic Cross points.

3.3.1 QMC and Regular Points Comparisons

To see the difference among different QMC points and regular grid points in isosurface extraction, we apply the QMC isosurface extration to two data sets using these three different QMC points and regular grid points.

Figures 3.4 and 3.5 show the results from different QMC points and regular grid points for the data sets SPHERE and SEIS respectively. One-eighth of the original data for each data set is used, which is 62500 points. Table 3.3 shows some performance statistics on IBM RS6000.

We see that for data set SPHERE, the Hammersley points generate the smoothest image. The iamge generated by Halton points is similar to that generated by Hammersley points, but probably less smooth. The Hyperbolic cross points generate an isosurface that is similar to the regular grid points. For data set SEIS, the Hammersley, Halton and Hyperbolic Cross points all give smoother images than the regular grid points.

Table 3.3 indicates that generally interpolation just takes slightly more time than the isosurface extraction, but generating the mesh takes a lot more time than

Table 3.3 Performance of QMC points

(Running time in seconds).

	Interpolation		Mesh Generation		Isosurface Extraction	
	SPHERE	SEIS	SPHERE	SEIS	SPHERE	SEIS
Hammersley	19.20	20.14	882.62	910.81	11.97	11.18
Halton	19.49	20.95	458.41	505.45	12.21	11.74
Hyperbolic	21.85	28.91	721.59	730.20	8.0	10.32
Regular			1315.0	676.72	6.55	9.34
Original					74.53	104.27

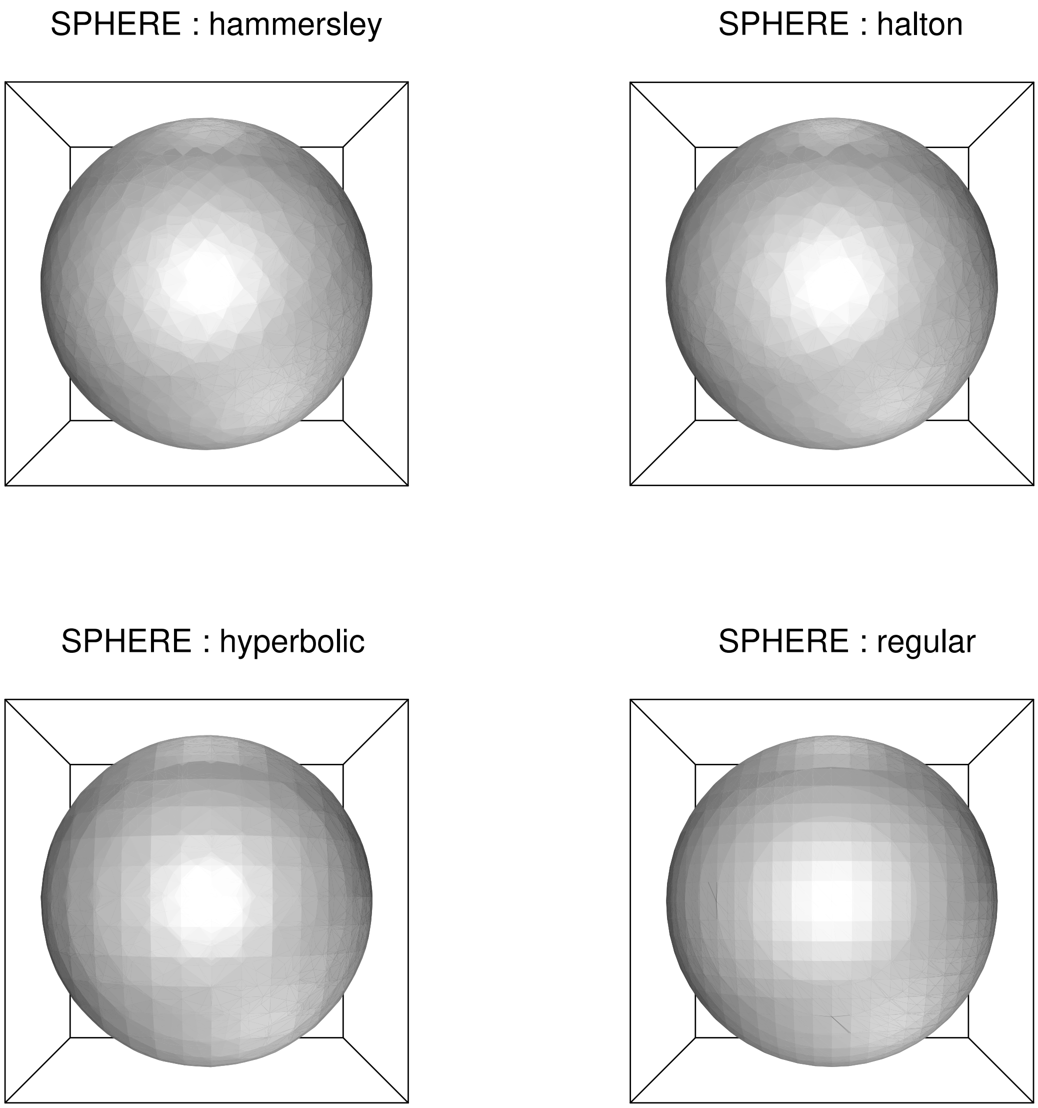

Figure 3.4. Comparison between QMC and regular points for SPHERE. One-eighth of the original data is used, which is 62500 points.

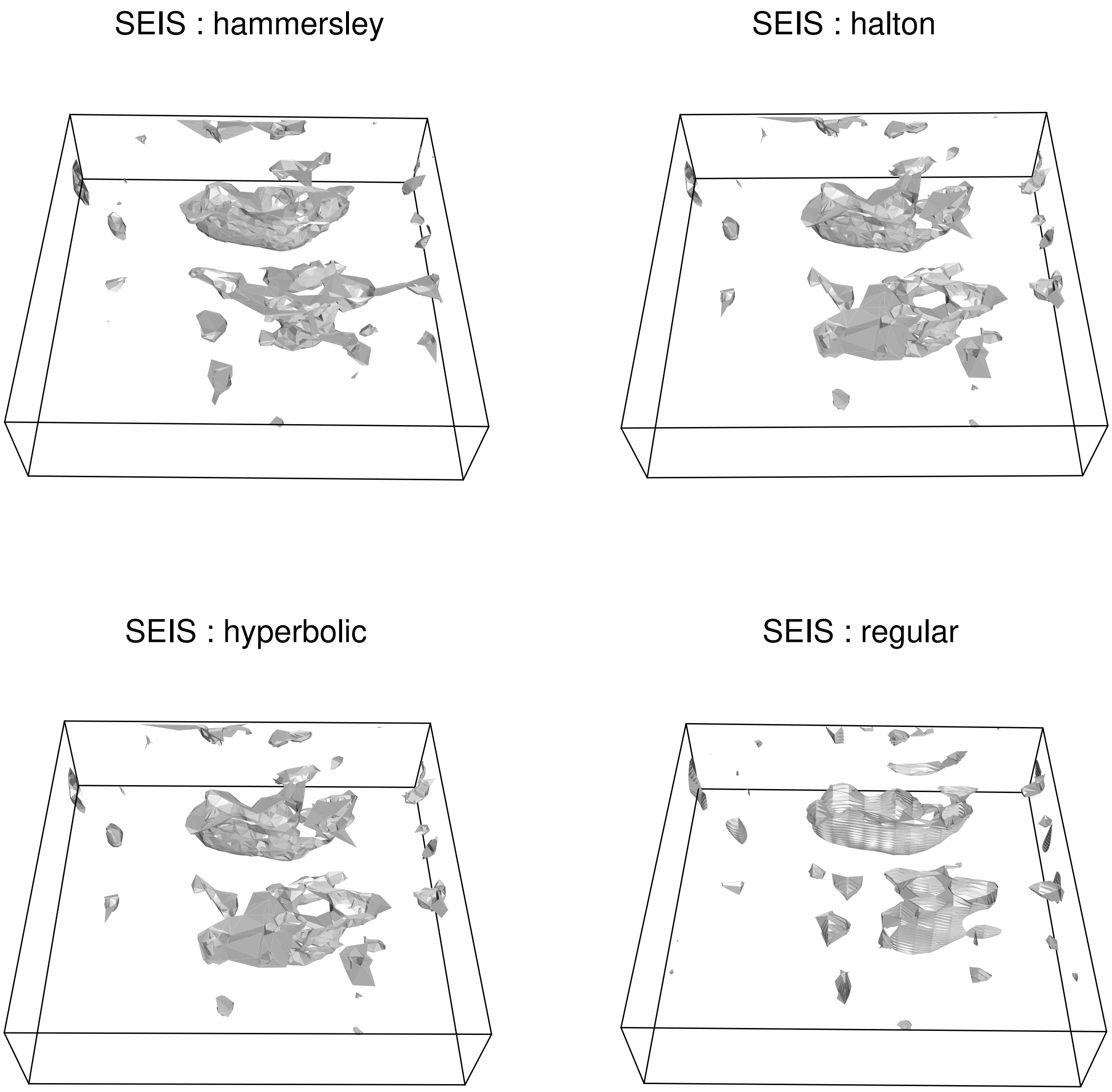

Figure 3.5. Comparison between QMC and regular points for SEIS. One-eighth of the original data is used, which 62500 points.

interpolation and isosurface extraction. So for the QMC isosurface extraction, most of the time in the preprocessing step is spent in the mesh generation. For every single category, the times used for the three sets of QMC points are about the same. Compared to the time spent in isosurface extraction with the full data set, on average only one-eighth of the original time is used when using one-eighth of the original data (1/7 for SPHERE and 1/10 for SEIS).

3.3.2 Subsampling Vs. Interpolation for Hyperbolic Cross Points

As discussed in Chapter 2, the Hyperbolic Cross points are a subset of the regular grid points. By carefully arranging the data, we can generate the Hyperbolic Cross points in such a way that they subsample the original data. By doing this, we take advantage of the property of the Hyperbolic Cross points and avoid interpolations. This section shows some comparisons between subsampling and interpolation.

Figures 3.6 and 3.7 show the results from the subsampling (Hyperbolic Cross points) and interpolation (Hammersley and Halton points) for the data sets SPHERE and SEIS respectively. The number of points used is 10660 for both data sets. Table 3.4 shows some performance statistics on IBM RS6000.

We see from the figures that for SPHERE, the images produced from all the QMC points are in the same quality. For SEIS, the Hyperbolic Cross points give a better image which repproduce the original image better. Seen from Table 3.4, the time spent in isosurface extraction for the Hyperbolic Cross points is a lot less than than those for the Hammersley and Halton points. For Hyperbolic Cross points, because subsampling is used, no time is spent in the interpolation step.

Table 3.4 Performance of subsampling and interpolation (Running time in seconds).

	Interpolation		Mesh Generation		Isosurface Extraction	
	SPHERE	SEIS	SPHERE	SEIS	SPHERE	SEIS
Hammersley	15.63	16.75	564.49	585.1	7.29	6.82
Halton	15.8	16.78	308.9	326	7.45	7.18
Hyperbolic	0	0	385.78	641.18	3.42	4.50

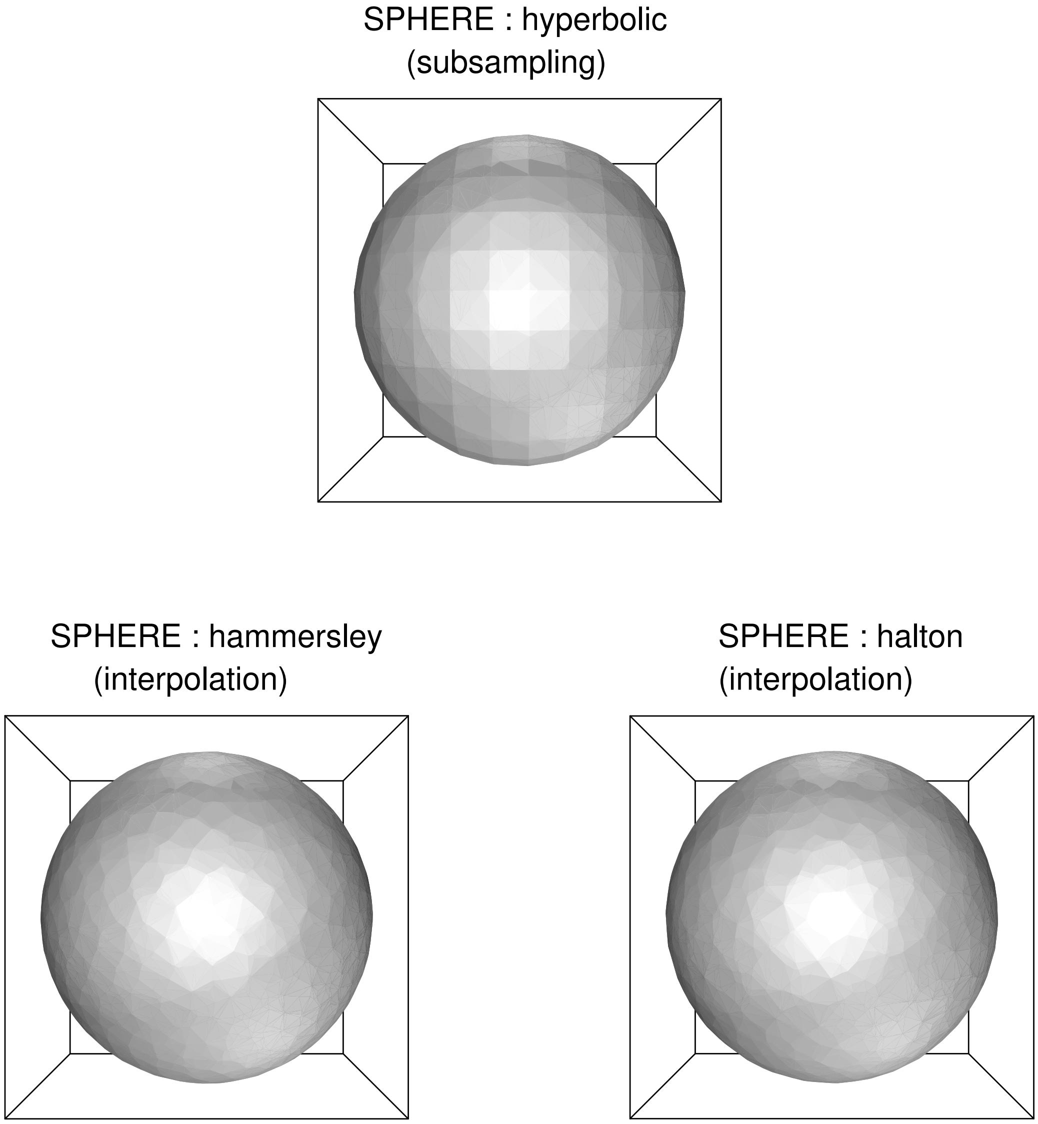

Figure 3.6. Comparison between subsampling (for Hyperbolic Cross points) and interpolation (for Hammersley and Halton points) for SPHERE. The number of points is 10660.

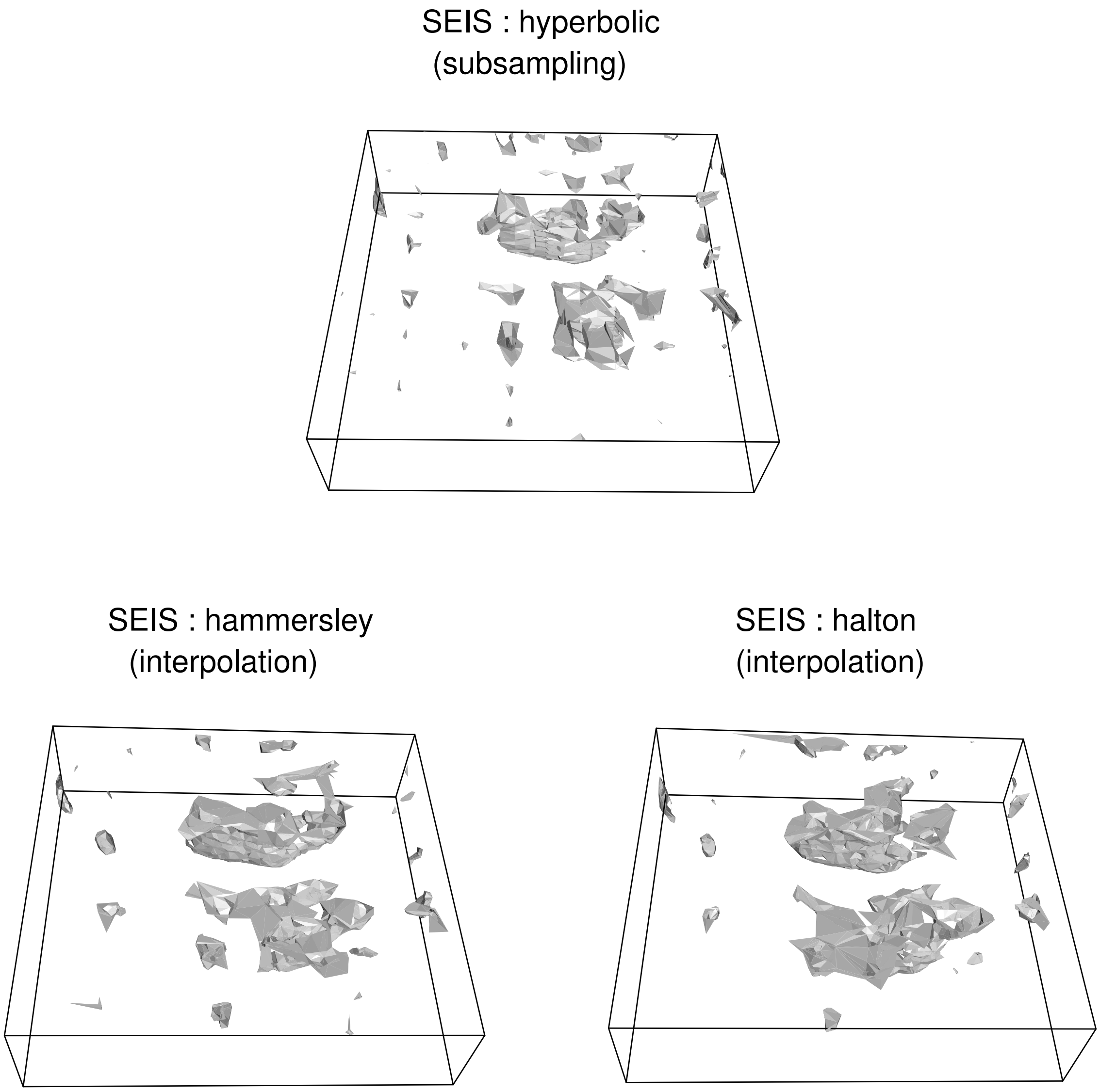

Figure 3.7. Comparison between subsampling (for Hyperbolic Cross points) and interpolation (for Hammersley and Halton points) for SEIS. The number of points is 10660.

The Hyperbolic Cross points have the advantage over other QMC points because they are a subset of the regular grids. As we can directly subsample the original data, no interpolation is needed. This makes the preprocessing step faster for the Hyperbolic Cross points than other QMC points. One disadvantage of the Hyperbolic Cross points is that the number of points cannot be arbitrarily chosen.

3.4 Data Reduction

The key idea of the QMC isosurface extraction is data reduction. Instead of using the whole data set, which is generally very large, we use QMC points to generate a smaller data set to do the isosurface extraction. By wisely choosing the QMC points and data reduction scale, we can extract the isosurfaces fast and accurate enough.

In this section, we aim to test this data reduction technique by reducing the data sets to different scales, and use the reduced data sets to extract the same isosurface. By doing this, we check the effectiveness of this data reduction technique.

We use the two data sets as mentioned before, and reduce them to 1/2, 1/4, 1/8 and 1/16 of the original size. The Hammersley points are used in the computation.

Figures 3.8 and 3.9 show the results for data sets SPHERE and SEIS, respectively.

Figure 3.8 indicates that for the same number of points as in the original data set, the QMC points give a smoother isosurface than the original one. When we use 1/2 and 1/4 of the original data, the extracted isosurfaces are still very smooth. When the data are reduced to 1/8 or 1/16, the extracted isosurfaces still represent the overall shape of the original one, though they are not as smooth as the original one, especially when we use 1/16 of the original data.

In Figure 3.9 for SEIS, when the data are reduced to 1/2 and 1/4, the isosurfaces almost reproduce the original one. When reduced to 1/8 and 1/16, the isosurfaces still represent the overall shape of the original, though there are slight distorsions in some part of the isosurfaces.

Figure 3.10 depicts the running time for the interpolation and the isosurface extraction for different data reduction scales. We conclude that the time spent

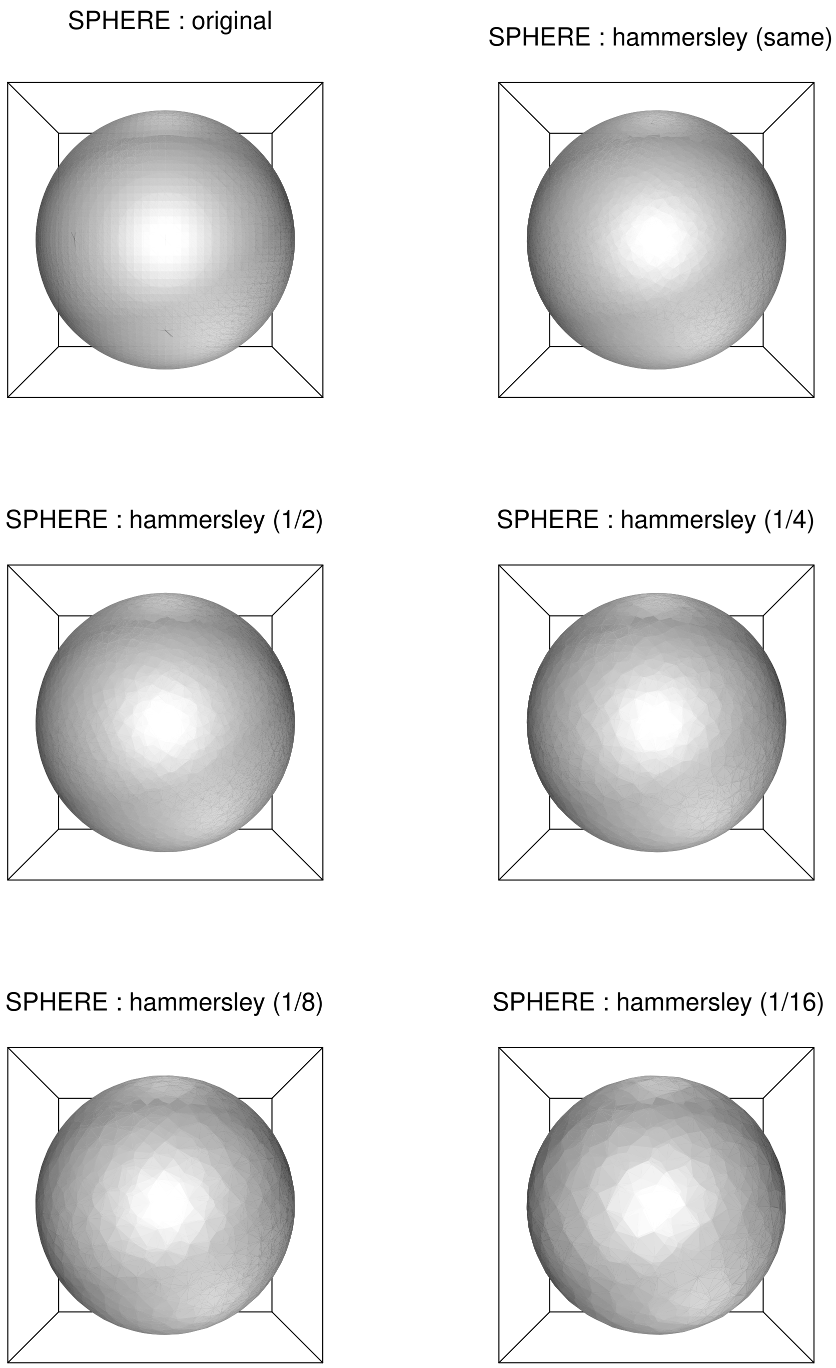

Figure 3.8. Data reduction comparisons for SPHERE.

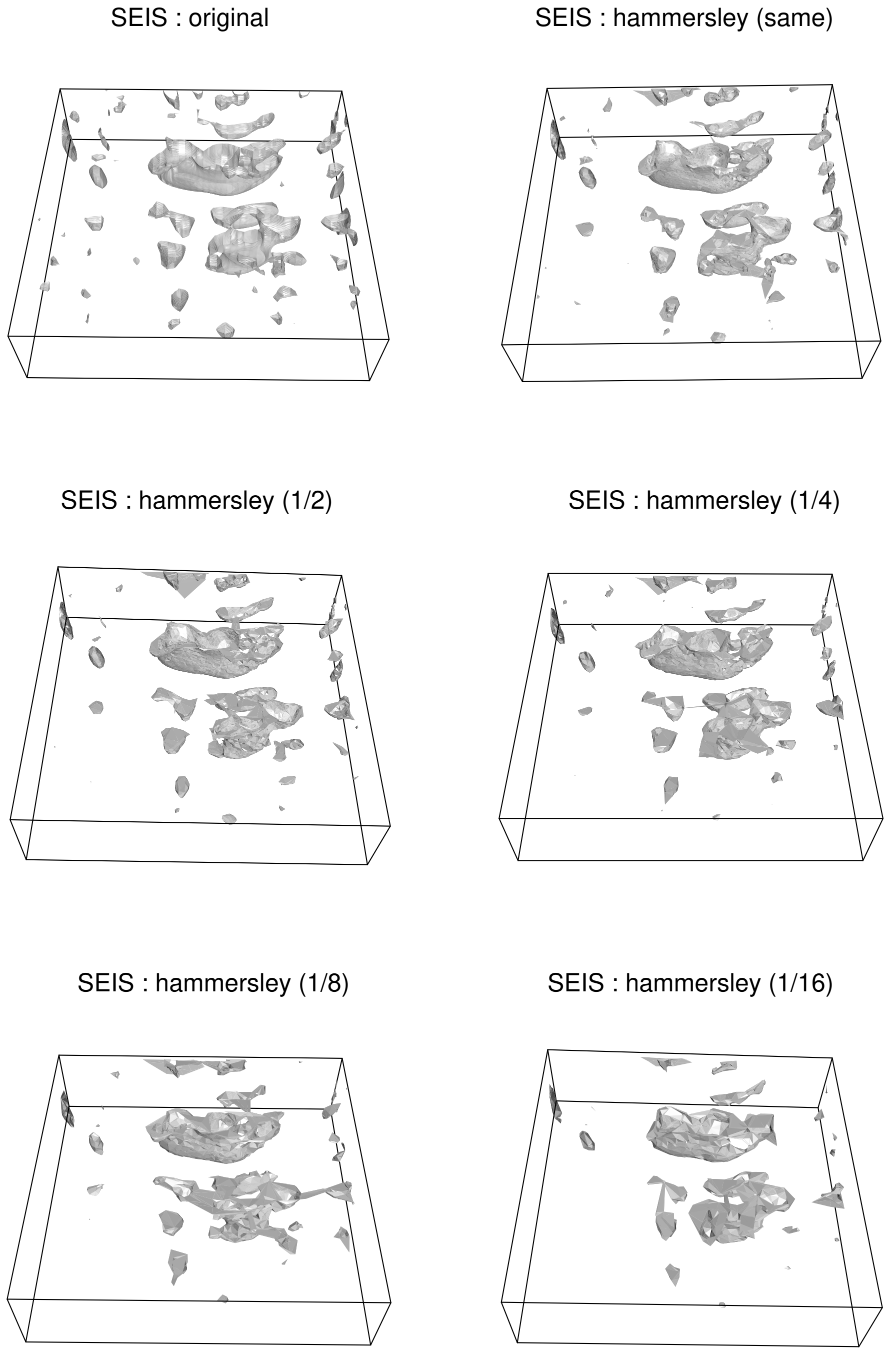

Figure 3.9. Data reduction comparisons for SEIS.

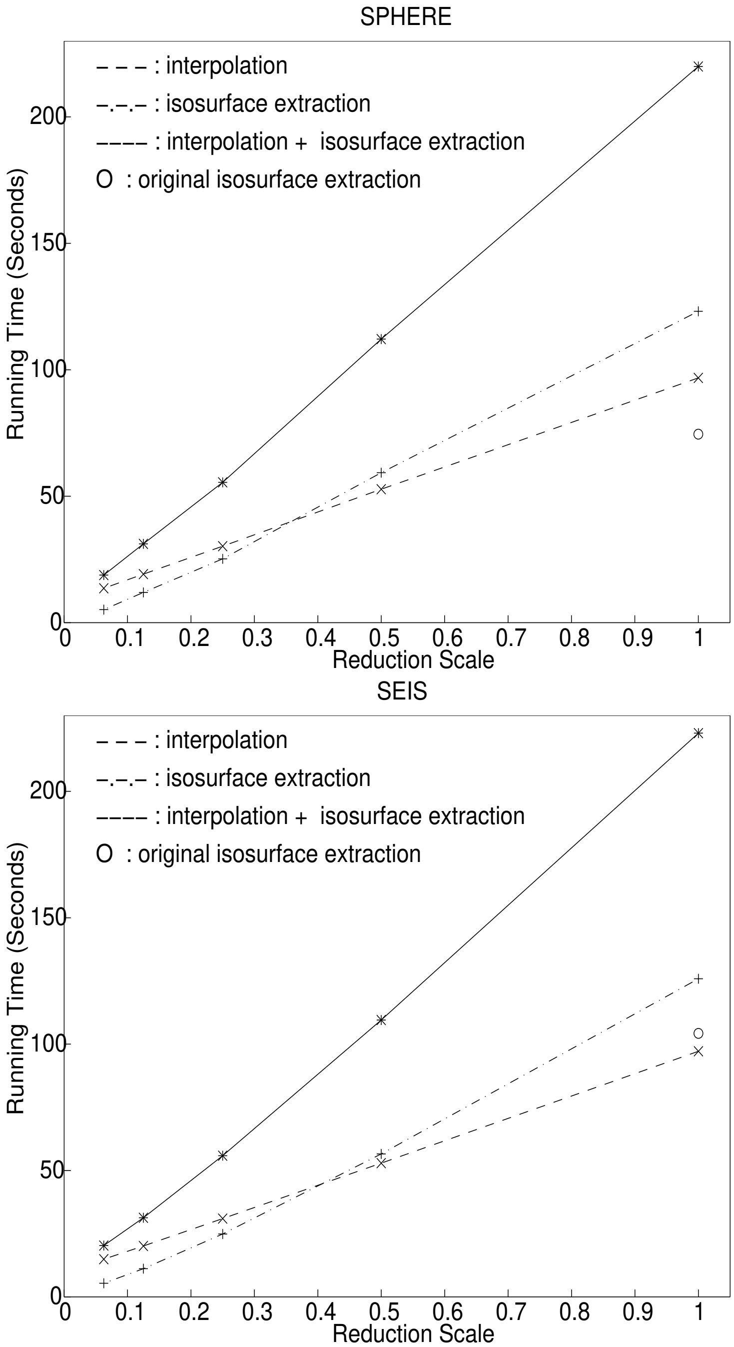

Figure 3.10. Data reduction perfomance on data sets SPHERE and SEIS.

in both interpolation and isosurface extraction are reduced linearly with the data reduction scale. In addition, the time spent in interpolation and isosurface extraction are about at the same level. So the QMC isosurface extraction demonstrates a linear speed up with decreasing the number of points.

3.5 Discussion

We can draw the following conclusions from the numerical tests:

- The linear interpolation provides accurate results for linear tetrahedral cells. Higher order interpolation does not yield much improvement (though it might be better for larger compression ratios).

- The QMC points generally generate a smooth isosurface that represents the global shape of the original surface. The differences among different QMC points are not prominent and depend on the data sets. Generally the QMC points produce better images than sub regular grid points.

- For large data sets, we usually can reduce the data size remarkbly and still get a good representation of the original isosurface. The advantage of the techniques becomes more prominent when the data size gets larger.

- The preprocessing of the QMC isosurface extraction might be time consuming, but after it is done, the post isosurface extraction is very fast. Most of the time spent in the preprocessing is used for mesh generation.

CHAPTER 4

QUASI-MONTE CARLO ISOSURFACE EXTRACTION FOR UNSTRUCTURED GRIDS

In the previous chapter, we applied the QMC isosurface extraction technique to the structured data sets. Another main type of data in scientific visualization is unstructured data. The unstructured data are usually generated by finite element method. Visualization of unstructured data is particularly important because many large applications provide only such data. In this chapter, we apply the QMC technique to some unstructured data sets.

Similar to the visualization of structured data, the QMC isosurface extraction for unstructured data contains the following steps:

- QMC points generation: generate specified number of quasi-Monte Carlo points, which will be used as the subdata set to extract the isosurfaces.

- Mesh generation: generate new geometry representation (tedrahetrons) based on the QMC points.

- Interpolation: interpolate function values from the original data set into the QMC points.

- Isosurface extraction: extract the isosurfaces from the QMC data sets.

Same as in Chapter 3, we use Hammersley, Halton and Hyperbolic Cross points in our computation. The algorithms generating these points are presented in Chapter 2. The isosurface extraction algorithm used is the NOISE algorithm presented in Chapter 1. The Delauny algorithm is used in the mesh generation.

Unlike structured data, the unstructured data are more difficult to deal with, because we cannot directly reference particular points. The whole volume and all the points are related by the mesh provided. This chapter intends to present the algorithm of applying the QMC isosurface extraction technique to unstructured data. Specificly, we investigate the following questions:

- How essential is the interpolation technique?
- How efficient are different QMC point sets?
- How effective is this data reduction technique?

This chapter is intended to answer these questions. We apply the QMC technique to various unstructured data sets by using different interpolation techniques, different QMC points, and different data reduction scales. We then draw conclusions.

4.1 Data Sets

We first present the data sets to be used in this chapter. Basicly, three unstructured data sets will be used in this chapter. These are also shown in the Table 4.1.

All of these sets consist of bio-electric field problems solved using the finite element method on unstructured tetrahedral grids.

An isosurface of each data set is shown in Figure 4.1.

4.2 Local Natural Coordinate

In the searching and interpolation algorithms for unstructured grids, we utilize the local natural coordinate system. In this section, we introduce this system.

Table 4.1 Data sets.

Name	Source	Type	Vertices	Cells
HEART	FEM	U-grid	11504	72641
TORSO	FEM	U-grid	34937	216607
HEAD	FEM	U-grid	137923	884343

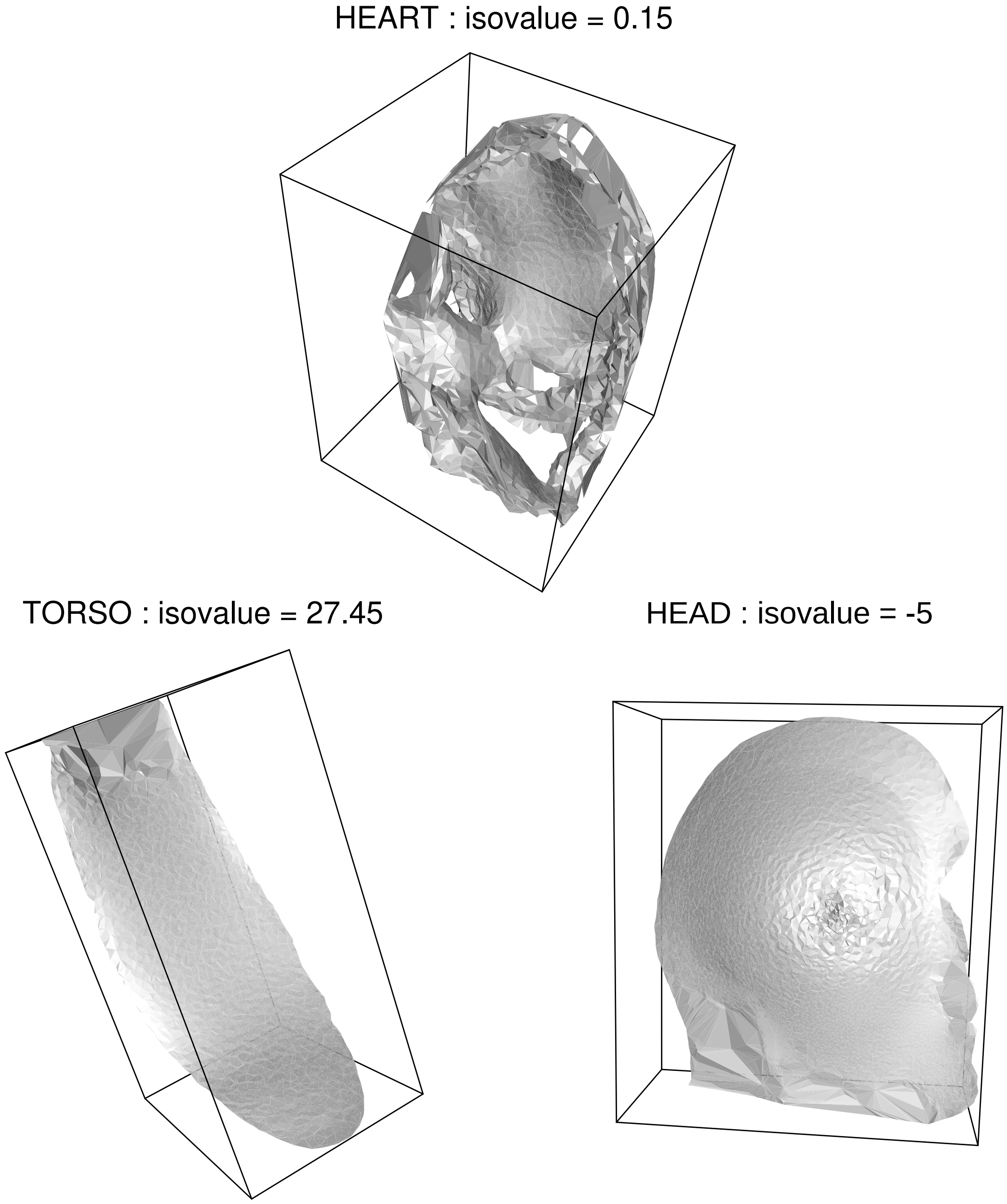

Figure 4.1. Isosurfaces of some unstructured data sets.

Let P_i $(i = 1, 2, 3, 4)$ be the vertices of a tedrahedron. Then we can define the *volume coordidate* for any point P inside the tedrahedron as (Figure 4.2):

$$\lambda_i = \frac{V_i}{V} = \frac{h_i}{H_i}, \; i = 1, 2, 3, 4,$$

where V_1 is the volume of the tedrhedron defined by nodes $PP_2P_3P_4$, V is the volume of the original tedradedron defined by the nodes $P_1P_2P_3P_4$, h_1 and H_1 are the distances from P and P_i to the bottom triangle $(P_2P_3P_4)$. Apparently $\lambda_i = 1$ for P_i, and $\lambda_i = 0$ for other vertices, and

$$\sum_{i=1}^{4} \lambda_i = 1. \tag{4.1}$$

The λ_is $(i = 1, 2, 3, 4)$ are also known as the *local natural coordinates* of the point P. The global Cartesian coordinates and the local natural coordinates are related by:

$$x = \sum_{i=1}^{4} \lambda_i x_i, \tag{4.2}$$

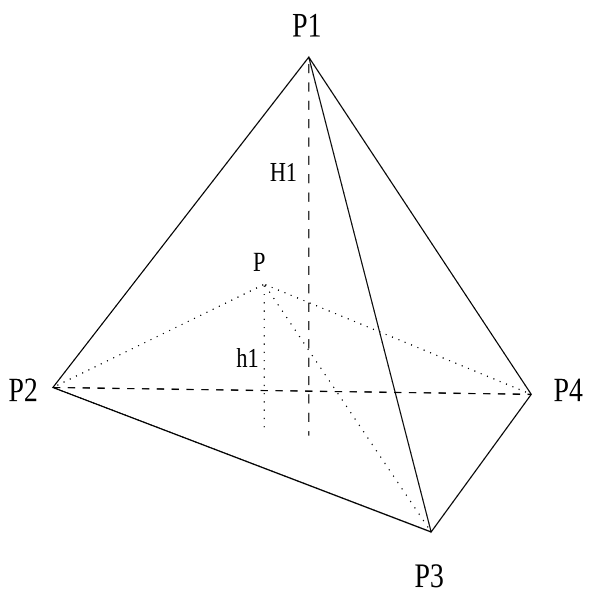

Figure 4.2. Volume coordinate for a point inside a tedrahedron.

$$y = \sum_{i=1}^{4} \lambda_i y_i, \tag{4.3}$$

$$z = \sum_{i=1}^{4} \lambda_i z_i. \tag{4.4}$$

By solving equations 4.1 to 4.4, we get

$$\begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{pmatrix} = \frac{1}{6V} \begin{pmatrix} -V_{234} & X_1 & Y_1 & Z_1 \\ V_{341} & -X_2 & -Y_2 & -Z_2 \\ -V_{412} & X_3 & Y_3 & Z_3 \\ V_{123} & -X_4 & -Y_4 & -Z_4 \end{pmatrix} \begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix},$$

where the constants can be defined as

$$V_{ijk} = \begin{pmatrix} x_i & x_j & x_k \\ y_i & y_j & y_k \\ z_i & z_j & z_k \end{pmatrix},$$

$$X_1 = \begin{pmatrix} 1 & 1 & 1 \\ y_2 & y_3 & y_4 \\ z_2 & z_3 & z_4 \end{pmatrix}, Y_1 = \begin{pmatrix} 1 & 1 & 1 \\ z_2 & z_3 & z_4 \\ x_2 & x_3 & x_4 \end{pmatrix}, Z_1 = \begin{pmatrix} 1 & 1 & 1 \\ x_2 & x_3 & x_4 \\ y_2 & y_3 & y_4 \end{pmatrix}.$$

The other constants can be obtained through a cyclic permutation of the subscripts $1, 2, 3$ and 4.

The volume V of the tedrhedron can be calculated by

$$V = -\frac{1}{6} \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix}.$$

4.3 Search on Unstructured Grids

After the QMC points are generated, the function values need to be interpolated into these QMC points. This process contains two steps:

- Find the cell (tedrahetron) in which the QMC point is located.
- Interpolate the function values.

In this section, we describe the searching algorithm. Next section will explain interpolation algorithms.

As discussed in Chapter 3, the searching process is simple for structured data sets because of the regularity of data. However, for unstructured data, the searching operation is more complex.

To find a cell containing point P in unstructured grids, the simplest way is to traverse all the cells in the data set. This is a naive method, and very inefficient. To speedup the process, people have employed various data structures to help fast searching.

Livnat et al. (1996) employeed a Kd-tree structure in the NOISE isosurface extraction algorithm. The Kd-tree was used to search the active cells that intersect the isosurfaces. We utilize the Kd-tree structure in our searching process as well. The following sections introduce the Kd-tree structure and present the algorithm of using Kd-tree in the searching process.

4.3.1 Kd-Trees

Kd-trees were designed by Bentley (1975) as a data structure for efficient associative searching. In essence, kd-trees are a multidimensional version of binary search trees. Each node in the tree holds one of the data values and has two subtrees as children. The subtrees are constructed so that all the nodes in one subtree, the left one for example, hold values that are less than the parent node's value, while the values in the right subtree are greater than the parent node's value.

Binary trees partition data according to only one dimension. Kd-trees, on the other hand, utilize multidimensional data and partition the data by alternating between each of the dimensions of the data at each level of the tree.

A traditional kd-tree maintains links to its two subtrees. This introduces a high cost in terms of memory requirements. To overcome this, Livnat et al. (1996) represented a pointerless kd-tree as a one-dimensional array of the nodes. The root node is placed at the middle of the array, while the first $n/2$ nodes represent the left subtree and the last $(n-1)/2$ nodes the right subtree.

4.3.2 Searching Using Kd-Trees

Given a data set, a kd-tree that contains pointers to the data cells is constructed. Using this kd-tree as an index to the data set, we can rapidly locate the cell that contains the given point. The searching using kd-trees contains the following steps:

- Kd-tree construction.
- Traversing the Kd-tree.
- Determination if a point is inside a tedrahetron cell.

In the following sections we talk about these three steps.

4.3.2.1 Kd-Tree Construction

As shown by Livnat et al. (1996), the construction of the kd-trees can be done recursively in optimal time $O(nlog(n))$. The approach is to find the median of the data values along one dimension and store it at the root node. The data are then partitioned according to the median and recursively stored in the two subtrees. The partition at each level alternates between the minimum and maximum coordinates.

Our Kd-tree is constructed in a similar way as that of Livnat et al. (1996). However, the Kd-tree that Livnat et al. used is two-dimensional, which means the partition and query at each level alternates between minimum and maximum coordinates in value space. The Kd-tree we used is six-dimensional, the partition and query at each level alternates among x-minimum, x-maximum, y-minimum, y-maximum, z-minimum and z-maximum. This six-dimensional Kd-tree is also constructed recursively.

4.3.2.2 Traversing the Kd-Tree

The traversing of the Kd-tree is to find the tetrahedron cell that contains a certain point $P(x_p, y_p, z_p)$. This task can be done by searching the kd-tree recursively. At each level the coordinates of the point are compared to the value stored at the current node. If the coordinates of the point are within the bounding box of the current cell, then we step further to check if the point is inside the tetrahedron cell (the algorithm for this step will be presented in next section). If the point is not inside the current cell, and if the node is to the left of the current coordinate of the current point, then only the left subtree should be traversed. Otherwise, both subtrees should be traversed recursively.

For efficiency, similar to what Livnat et al. (1996) have done in the NOISE algorithm, we define two search routines, $search-min-max$ and $search-max-min$. The dimension that we are currently checking is named the first, and the dimension we still need to search is named the second.

The pseudo-codes for the $search-min-max$ and $search-max-min$ routines are as following.

```
search-min-max( P-coord, node, index )
{
   if ( node.extrema[2*index] < P-coord[index] ) {
      if ( point P is inside node )
          interpolate the function value

      search-max-min( P-coord, node.right, index );
   }
   search-max-min( P-coord, node.left, index );
}

search-max-min( P-coord, node, index )
{
   if ( node.extrema[2*index+1] < P-coord[index] ) {
      if ( point P is inside node )
          interpolate the function value

      search-max-min( P-coord, node.right, (index+1) % 3 );
   }
   search-max-min( P-coord, node.left, (index+1) % 3 );
```

```
}
```

In the above pseudo-codes, we used a parameter *index* to help searching among x-min, x-max, y-min, y-max, z-min and z-max at each level.

4.3.2.3 Point Inside a Cell

As shown in the pseudo-codes in the previous section, we need to utilize an algorithm to determine if a point $P(x_p, y_p, z_p)$ is inside a tetrahedral cell. Actually this can be done very easily if we work in the local natural coordinates.

Suppose $\lambda_i, i = 1, 2, 3, 4$ are the natural coordinates of the point $P(x_p, y_p, z_p)$ corresponding to a certain tetrahedral cell. We know only three of the four λ_i are independant, because they are constrained by the following equation:

$$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1.$$

The following creteria can be used to determine if the point $P(x_p, y_p, z_p)$ is inside the tetrahedron cell:

$$\lambda_i \in [0, 1], i = 1, 2, 3, 4.$$

The calculation of the natural coordinates λ_i is discussed in Section 4.2.

4.4 Interpolation on the Unstructured Grids

Once we know the tetrahedral cell in which a QMC point $P(x_p, y_p, z_p)$ is located, we can interpolate the function value to this QMC point by utilizing the local natural coordinates. In this section, we describe the interpolation algorithms. Two interpolation algorithms are tested for unstructured grids:

- Linear interpolation.
- Quadratic interpolation.

The following sections explain each of them.

4.4.1 Linear Interpolation

The linear interpolation uses the function values at the four vertices to interpolate the function value of the point P inside a tedrahedron:

$$f_p = \sum_{i=1}^{4} \phi_i f_i,$$

where ϕ_i is the interpolation function that can be defined in the terms of the local natural coordinates. For linear interpolation, we have

$$\phi_i = \lambda_i, i = 1, 2, 3, 4.$$

The $f_i(i = 1, 2, 3, 4)$ are the function values at the four vertices of a tetrahedral cell.

4.4.2 Quadratic Interpolation

The quadratic interpolation uses 10 points to interpolate the function value inside a tedrahedron. The 10 points used in the interpolation include the four vertices and the midpoints of the six edges of the tedrahedron, which are shown in Figure 4.3.

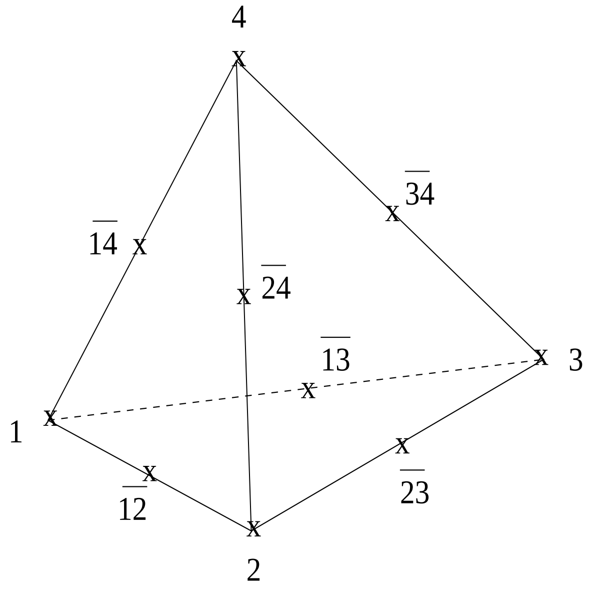

Figure 4.3. Ten points used in quadratic interpolation.

The qrardratic interpolation function (Kenneth, 1982) can be defined in terms of the local natural coordinates as:

$$\begin{aligned}\phi_i &= 2\lambda_i(\lambda_i - \frac{1}{2}) \quad i = 1, 2, 3, 4,\\ \phi_{\bar{i}\bar{j}} &= 4\lambda_i\lambda_j, \quad i \neq j.\end{aligned}$$

The interpolation formula is

$$f_p = \sum_{i=1}^{4} \phi_i f_i + \sum_{i=1, j=1, i\neq j}^{4} \phi_{ij} f_{ij}$$

If the function values at those midpoints are not defined, we can simply use a linear interpolation to get the function values at those points:

$$f_{ij} = \frac{f_i + f_j}{2}, i, j = 1, 2, 3, 4, i \neq j.$$

Then the interpolation formula can be rewritten as:

$$f_p = \sum_{i=1}^{4} \phi_i f_i + \sum_{i=1, j=1, i\neq j}^{4} \phi_{ij} \frac{f_i + f_j}{2}.$$

4.4.3 Numerical Results

In this section, we apply the above two interpolation algorithms to three unstructured data sets and compare the results. Hammersley points are used in the computation.

Figures 4.4 and 4.5 show the results by applying linear and quadratic interpolations to three unstructured data sets. The number of points used is half of the original size of each data set for Figure 4.4 and 1/16 for Figure 4.5. The left columns are for the linear interpolation, and the right columns are for the quadratic interpolation. Table 4.2 shows the performance of the two interpolation techniques on three data sets for using half of the original data. The numbers shown in the table are the running time in seconds as implemented on IBM RS6000 with 512 MB virtual memory.

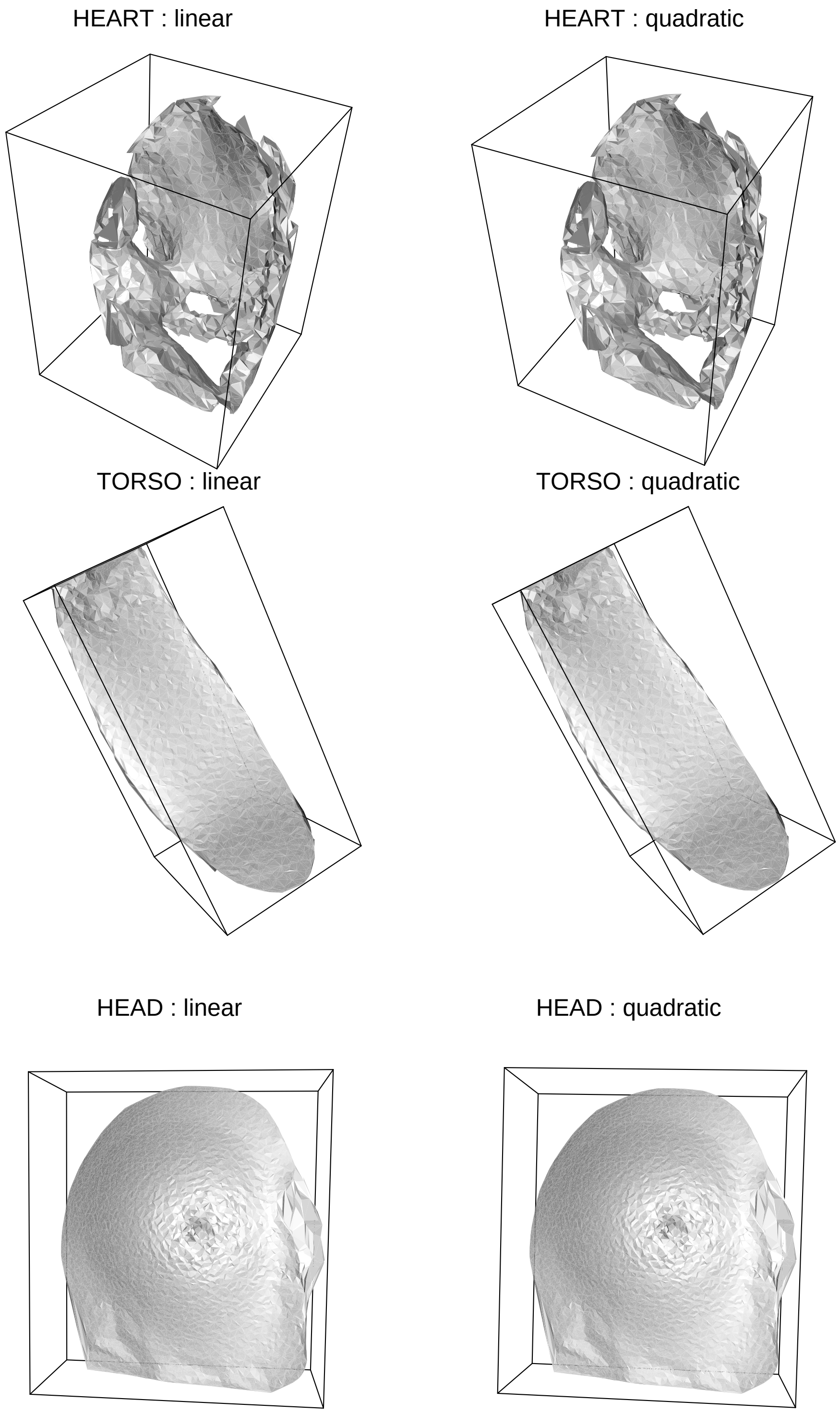

Figure 4.4. Comparisons between linear and quadratic interpolation. Half of the original data size is used for each data set.

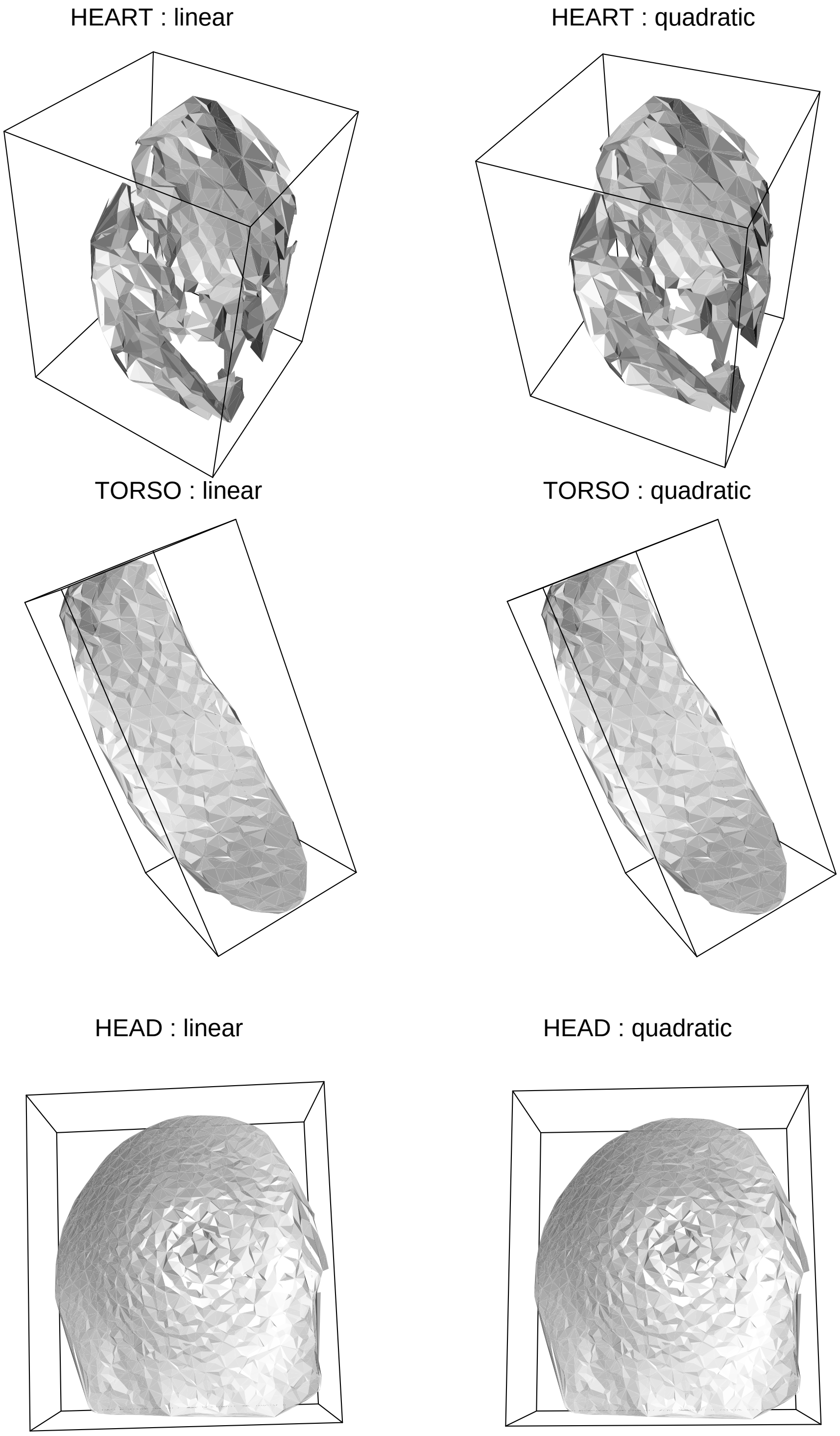

Figure 4.5. Comparisons between linear and quadratic interpolation. One-sixteenth (1/16) of the original data size is used for each data set.

Table 4.2 Interpolation performance

(Running time in seconds).

	HEART	TORSO	HEAD
Linear Interpolation	45.62	195.81	1270.59
Quadratic Interpolation	45.61	196.01	1273.01
Mesh Generation (DELAUNY)	287.87	1056.38	412.98
Isosurface Extraction (NOISE)	3.53	12.14	59.07

The quadratic interpolation is more accurate, which gives a smoother isosurface. But the difference among the appearances of the images is very small. Table 4.2 lists the time spent in the interpolations, which also include the searching time. Unlike in case of the structured grids, the quadratic interpolation takes about the same time as the linear interpolation. This is because most of the time spent in the interpolation is in searching and the local natural coordinates transformation. The quadratic interpolation does not yield much improvements. The reason is as we discussed in Chapter 3 and because we are working on linear tetrahedral cells. In this case linear interpolation is supposed to give enough precision.

Figure 4.6 shows the time spent in linear interpolation and NOISE isosurface extraction. Unlike in the case of the structured grids, interpolation takes more time than NOISE isosurface extraction, especially for the large number of QMC points. This is because seaching the cell and performing the coordinate transformation take more time. Tabe 4.2 also shows that mesh generation takes a lot more time than interpolation and isosurface extraction. So in the preprocessing step, most of the time is spent in generating the tetrahedral cells.

We draw the conclusions that linear interpolation generally povides enough precision for the linear tetrahedral cells, and quadratic interpolation does not yield much improvement. Therefore, the rest of the computations in this chapter are employing linear interpolation, unless otherwise mentioned.

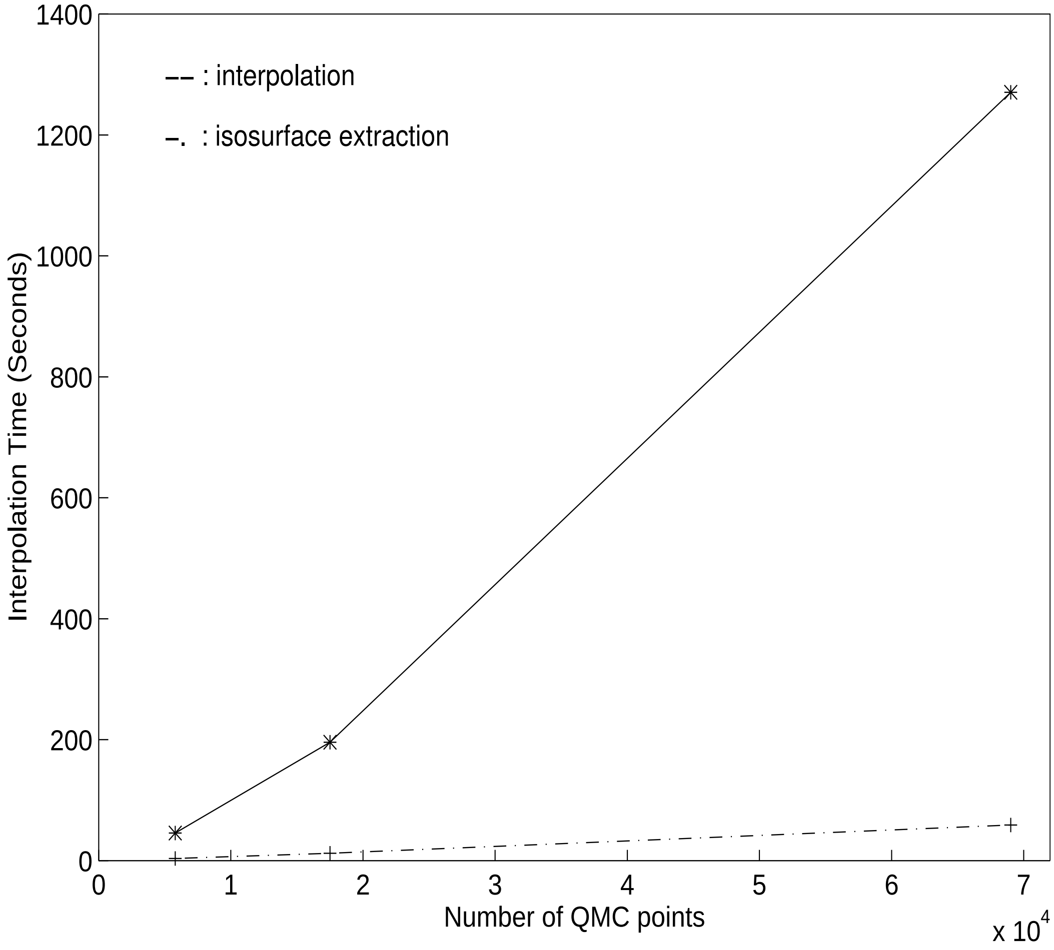

Figure 4.6. Time spent in linear interpolation and NOISE isosurface extraction for different number of QMC points.

4.5 QMC and Regular Points Comparisons

We use three types of QMC points in our computation, Hammersley, Halton and Hyperbolic Cross points. Also as discussed in Chapter 1, some of the data reduction techniques reduces the data by taking a regular subset of the original data. For the unstructured data sets, this might be done, for example, by interpolating the function values from the unstructured grids into the selected structured grids.

To see the difference among different QMC points and regular grid points in isosurface extraction, we apply the QMC isosurface extraction to three data sets using these three different QMC points and regular grid points.

Figures 4.7, 4.8, and 4.9 show the results for data sets HEART, TORSO and HEAD respectively. One-eighth of the original data size is used for each data set (1438 points for HEART, 4367 points for TORSO, and 17240 points for HEAD). Table 4.3 shows some performance statistics on IBM 6000.

For data set HEART, the differences among different grids are not prominent. This is because the HEART data set is small. Using 1/8 of the original data does not give a satisfactory image for all the grids. For data set TORSO, Hammersley and Halton points give better image. This can be seen even better for data set HEAD. When using 1/8 of the original data, the regular grids greatly distorted the isosurface for data set HEAD.

Table 4.3 Performance of QMC points
(Running time in seconds).

		Hammersley	Halton	Hyperbolic	Regular
Interpolation	HEART	30.42	30.52	32.10	30.42
	TORSO	114.47	114.64	119.11	112.92
	HEAD	638.34	639.71	594.40	633.05
Mesh Generation	HEART	64.64	48.36	60.41	47.95
	TORSO	213.65	134.77	203.59	135.19
	HEAD	999.18	530.58	714.19	530.83
Isosurface Extraction	HEART	0.69	0.70	0.51	0.70
	TORSO	2.49	2.55	1.96	2.54
	HEAD	13.35	13.33	9.40	13.31

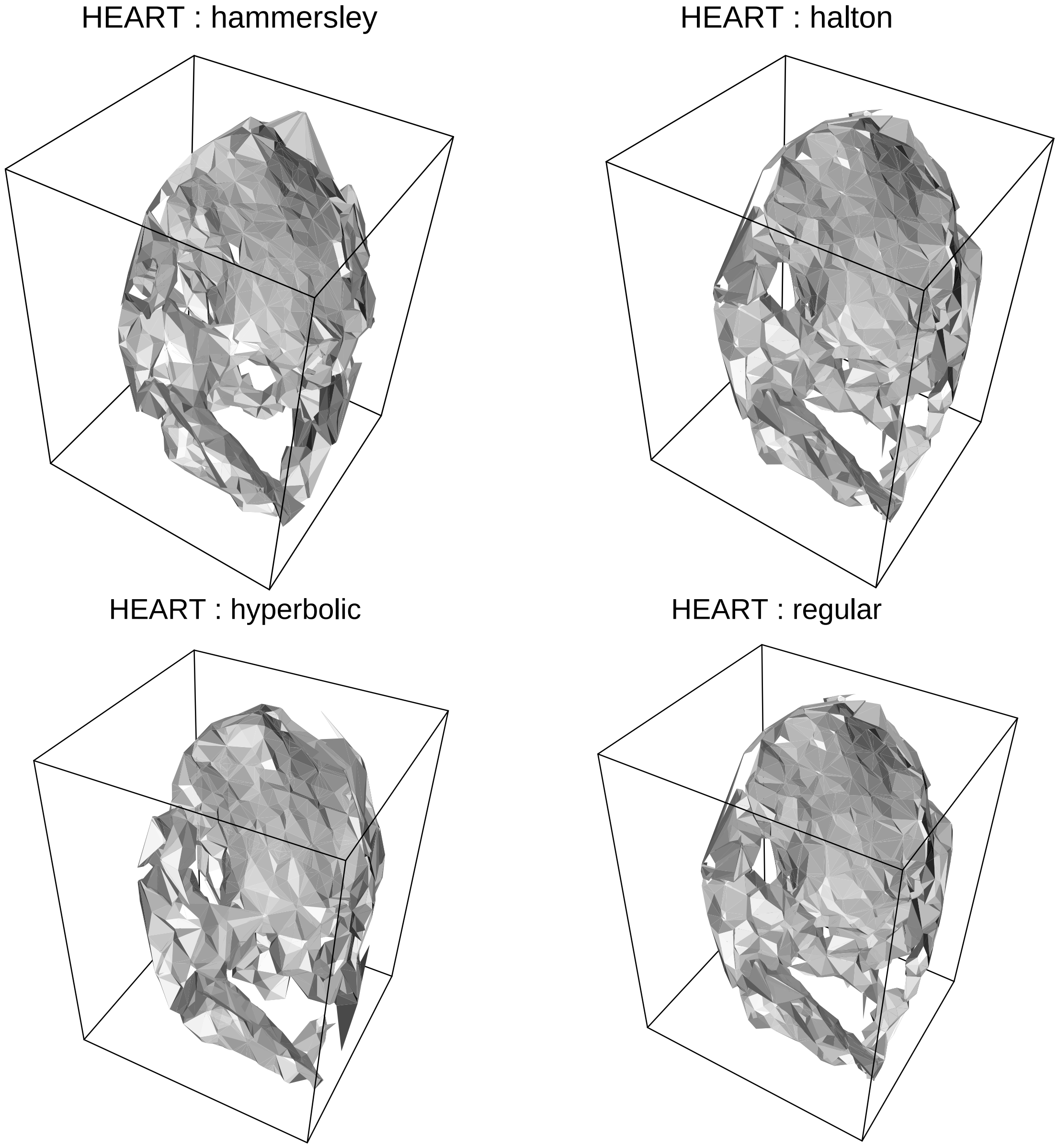

Figure 4.7. Comparison between QMC and regular points for HEART. One-eighth of the original data size is used (1438 points).

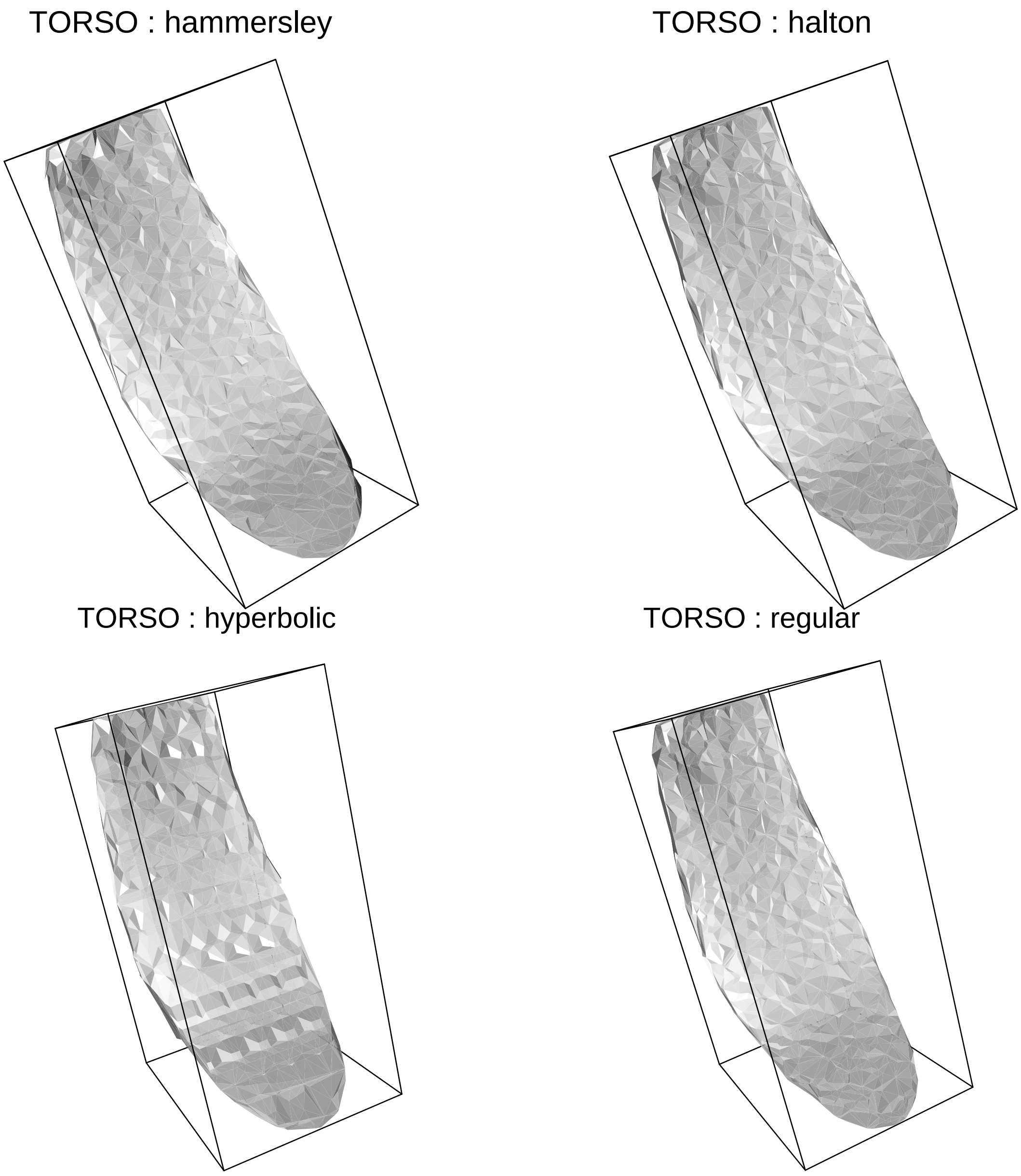

Figure 4.8. Comparison between QMC and regular points for TORSO. One-eighth of the original data size is used (4367 points).

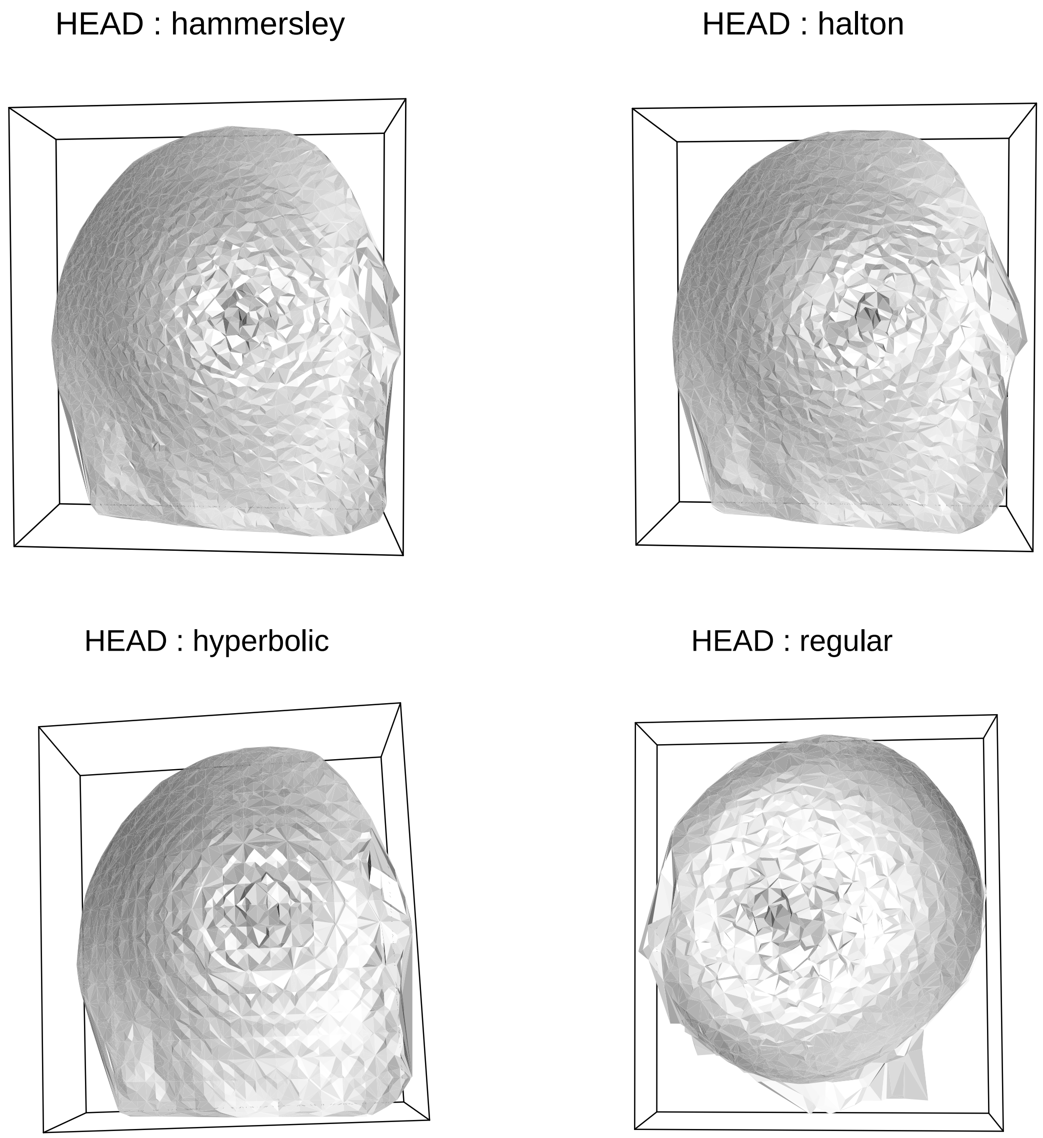

Figure 4.9. Comparison between QMC and regular points for HEAD. One-eighth of the original data size is used (17240 points).

Table 4.3 indicates that generally interpolation takes more time than isosurface extraction. Generating the mesh takes more time than interpolation and isosurface extraction. So for the QMC isosurface extraction, most of the time in the preprocessing step is spent in the mesh generation. For every single category, the times used for the three sets of QMC points and regular grid points are about the same.

4.6 Data Reduction

The key idea of the QMC isosurface extraction is data reduction. Instead of using the whole data set, which is generally very large, we use QMC points to generate a smaller data sets to extract the isosurfaces. By wisely choosing the QMC points and data reduction scale, we can extract the isosurfaces fast and accurate enough.

In this section, we aim to test this data reduction technique by reducing the data sets to different scales, and use the reduced data sets to extract the same isosurface. By doing this, we check the effectiveness of this data reduction technique on unstructured data.

We use the three data sets as mentioned before, and reduce them to 1/2, 1/4, 1/8 and 1/16 of the original size. The Hammersley points are used in the computation.

Figures 4.10, 4.11 4.12 show the results for data sets HEART, TORSO and HEAD, respectively.

From Figures 4.10 and 4.11, we see that for data sets HEART and TORSO, when using the same number of points as the original data set, the QMC points give an isosurface of the same quality as the original one. Sometimes the QMC points might not represent the boundaries very well. When we use 1/2 and 1/4 of the original data, the extracted isosurfaces are still very smooth. When the data is reduced to 1/8 or 1/16, the extracted isosurfaces still represent the overall shape of the original one, though they are not as smooth as the original one, especially when we use 1/16 of the original data.

For data set HEAD, when using the same number of points, the QMC points gives a better image than the original one. The reduction by 1/2 and 1/4 gives a very smooth isosurface. When reduced to 1/8 and 1/16, the isosurface still represents the overall shape of the original one.

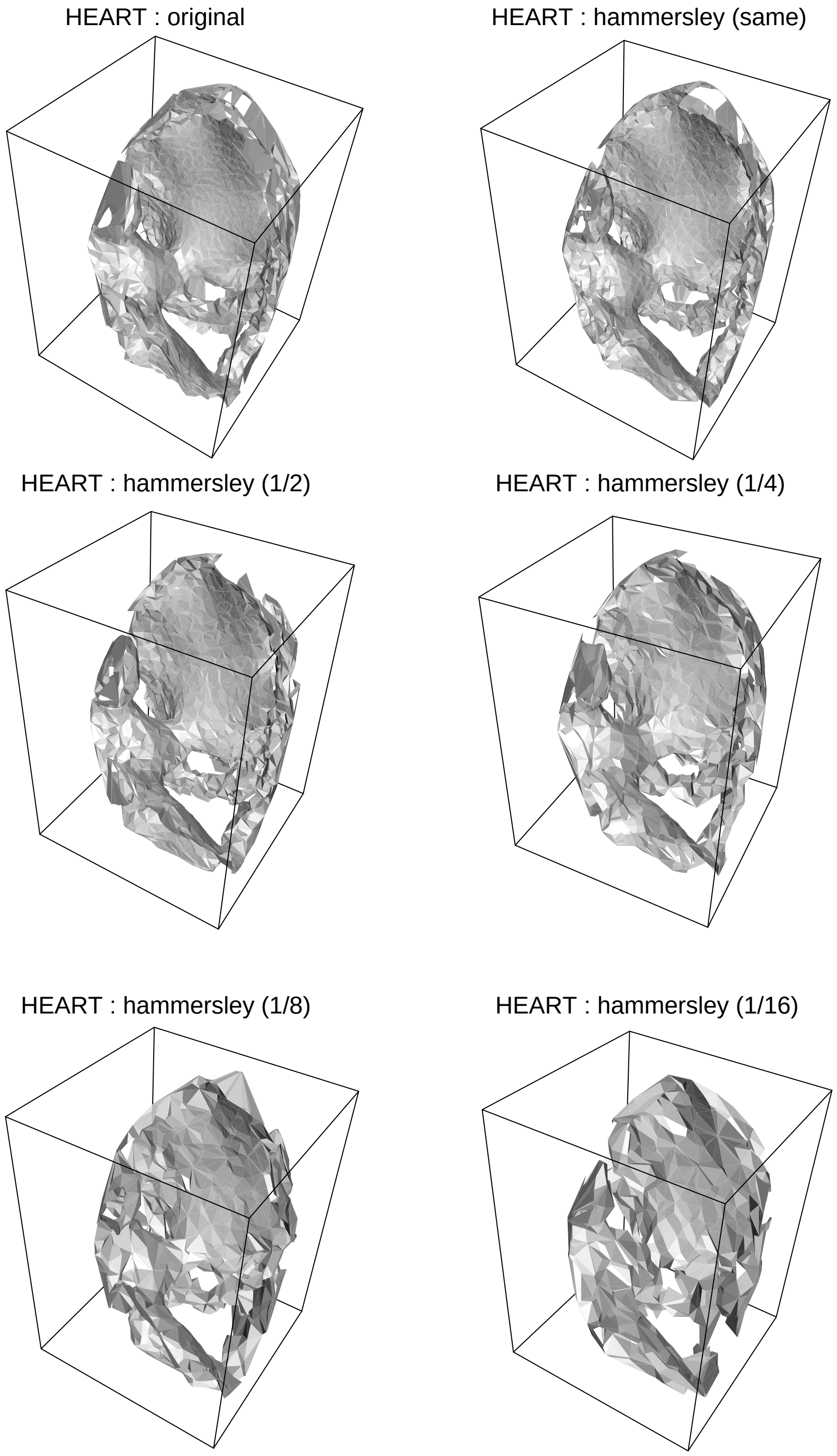

Figure 4.10. Data reduction comparisons for HEART.

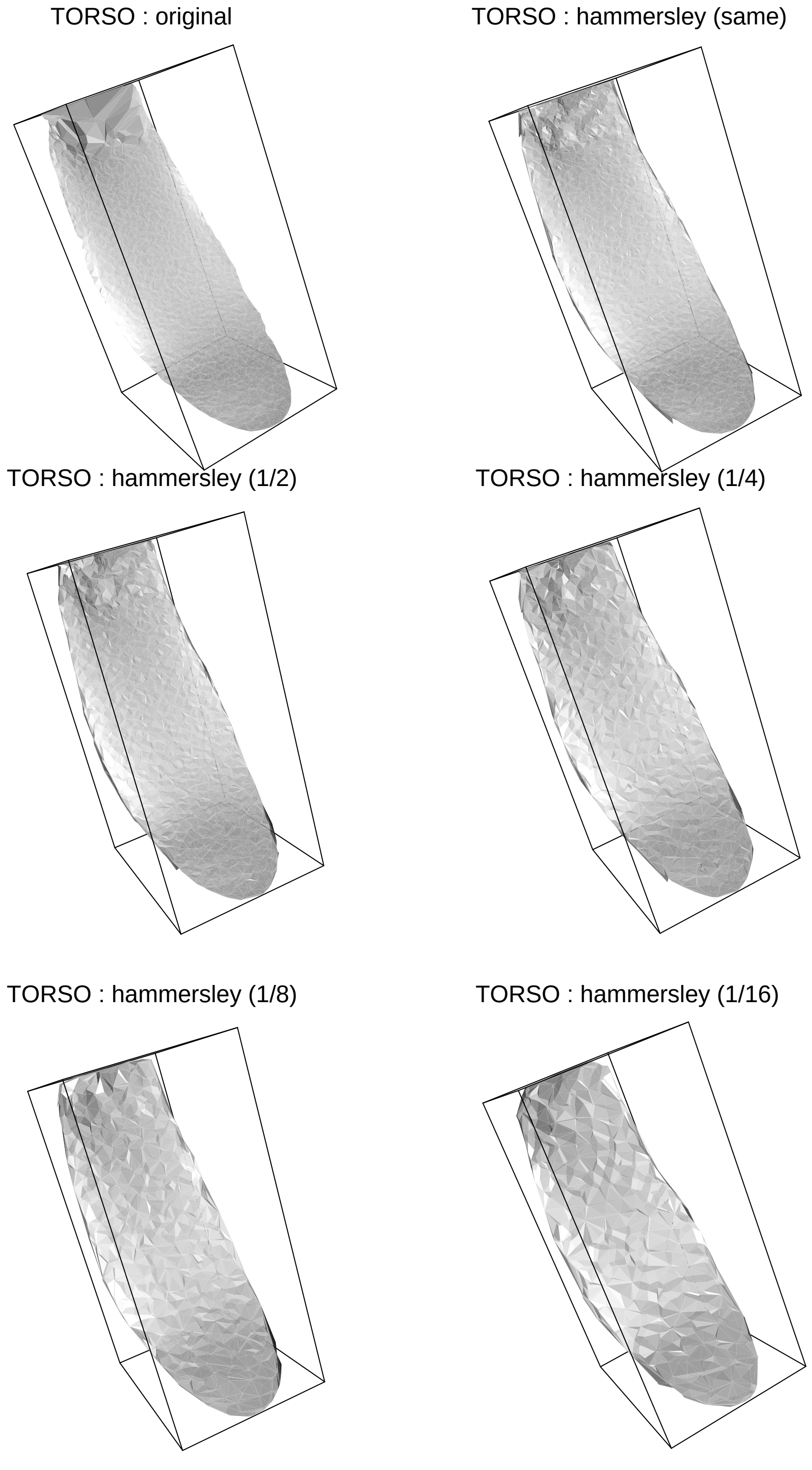

Figure 4.11. Data reduction comparisons for TORSO.

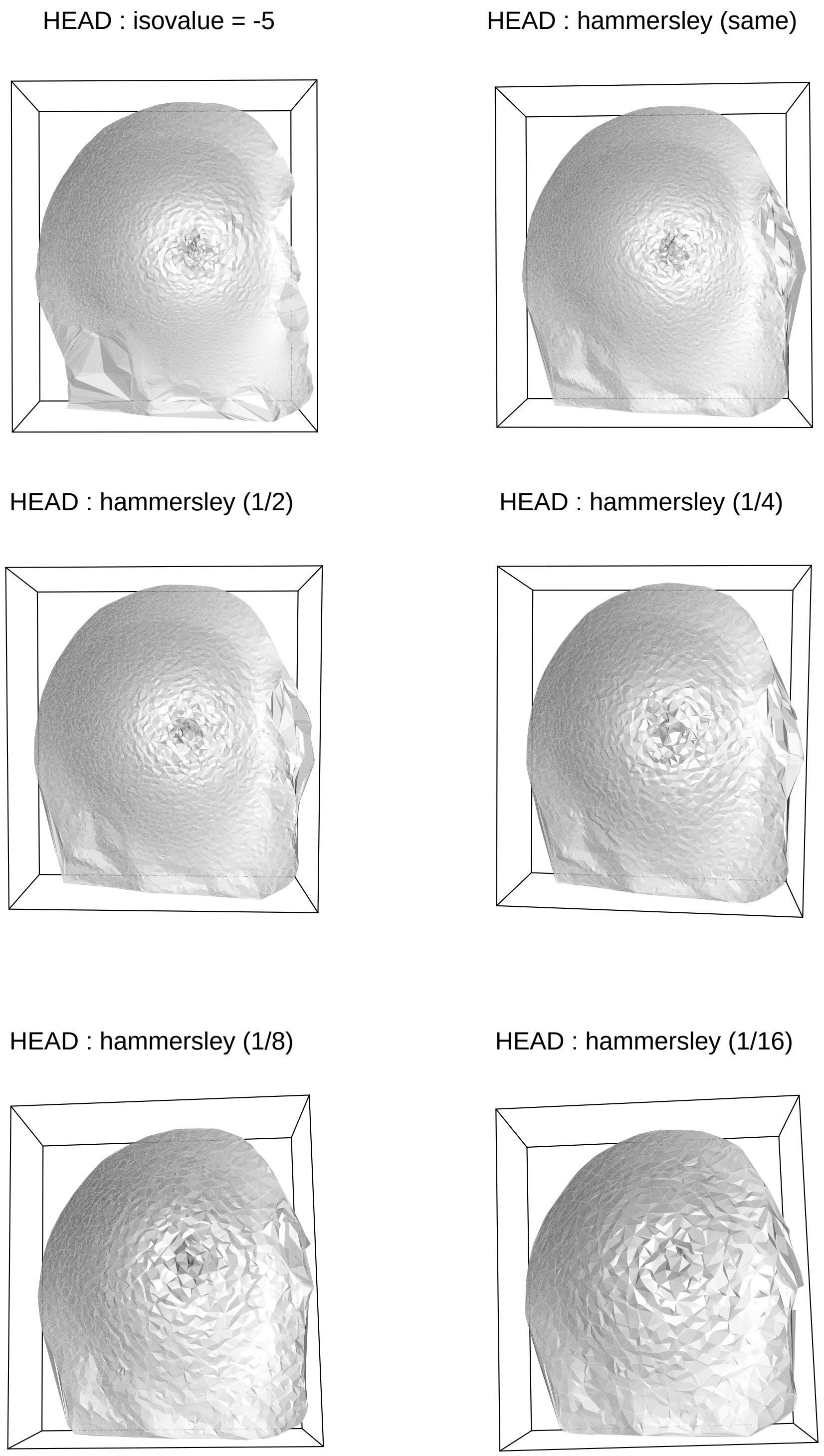

Figure 4.12. Data reduction comparisons for HEAD.

Figure 4.13 shows the running time for the interpolation and the isosurface extraction for different data reduction scales for three data sets. The time spent in both interpolation and isosurface extraction is reduced linearly with the data reduction scale. Interpolation takes more time than isosurface extraction, especially when the number of QMC points is large. So the QMC isosurface extraction has a linear speed up with decreasing the number of points.

When the data size gets larger, the QMC data reduction technique gets more effective. We can reduce the data by a very large scale and still can reproduce a good isosurface.

4.7 Discussion

We can draw the following conclusions from the numerical tests performed in this chapter:

- The linear interpolation provides accurate results for linear tetrahedral cells for unstructured grids. Higher order interpolation does not yield much improvement (though it might be better for large compression ratios).

- The QMC points generally generate a smooth isosurface that represents the global shape of the original surface. The differences among different QMC points are not prominent and depend on the data sets. Generally the QMC points produce better images than subregular grid points.

- For large data sets, we usually can reduce the data size remarkbly and still get a good representation of the original isosurface. The advantage of the techniques can be seen more clearly when the data size gets larger.

- The preprocessing of the QMC isosurface extraction might be time consuming, but after it is done, the postisosurface extraction is very fast. Most of the time spent in the preprocessing is used for mesh generation.

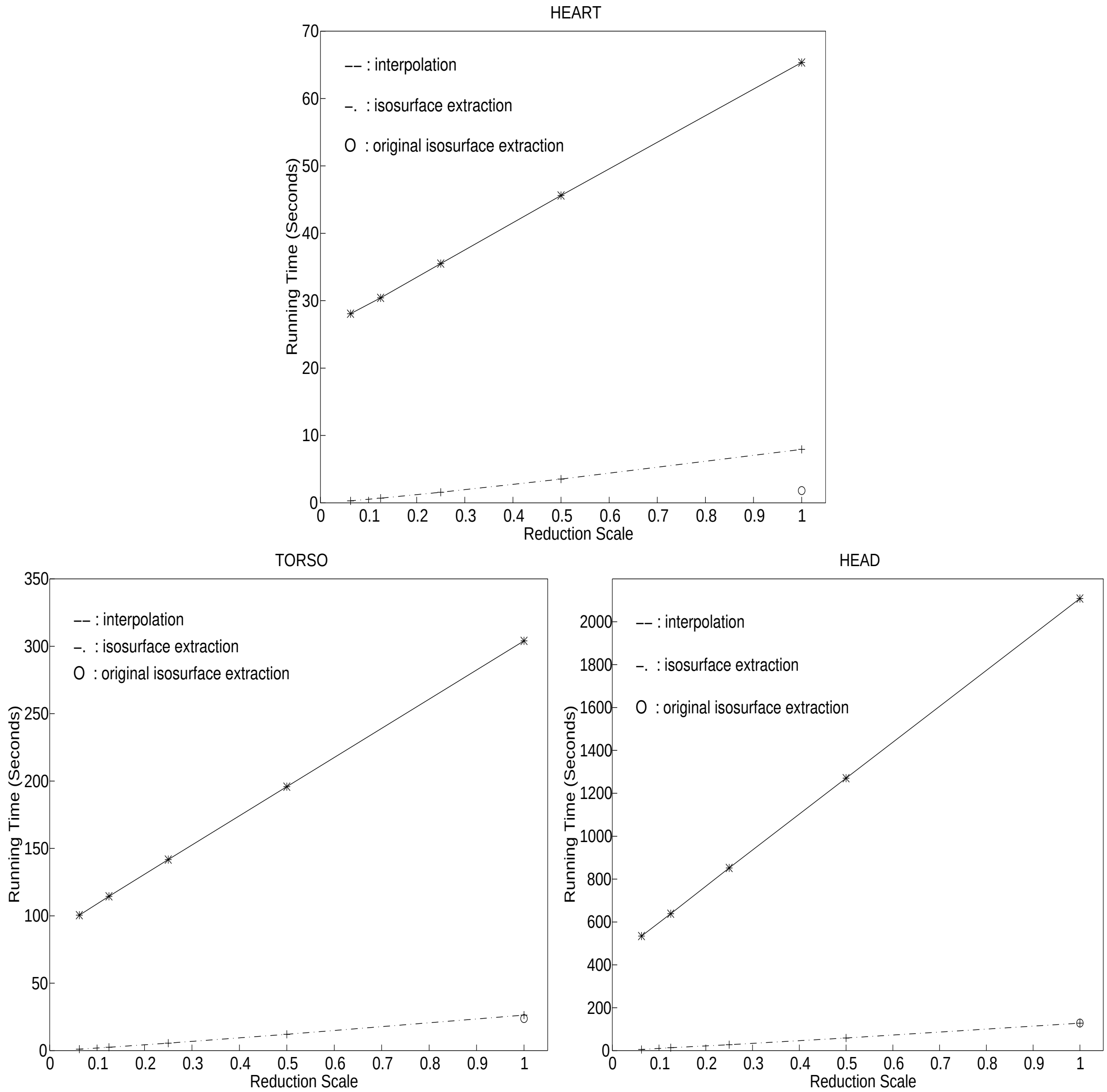

Figure 4.13. Data reduction performance.

CHAPTER 5

DISCUSSION AND CONCLUSIONS

Isosurface extraction and visualization is one of the most effective and powerful techniques for the investigation of three-dimensional scalar fields. However, for large three-dimensional data, this process is computationally slow. In this thesis, a new data reduction algorithm is presented with the use of Quasi-Monte Carlo points. The new algorithm is tested on some structured and unstructured data sets. The new algorithm first generates requested number of QMC points, and then interpolates the function values into these QMC points. Then this subset of the original data is used for the fast isosurface extraction.

The following conclusions can be drawn from the numerical tests performed in this thesis:

- For linear tetrahedral cells, linear interpolation provides accurate results. Higher order interpolations do not yield much improvement (though it might be better for large compression ratios).

- The QMC points generally generate a smooth isosurface that represents the global shape of the original surface. The differences among different QMC points are not prominent and depend on the data sets. Generally the QMC points produce better images than subregular grid points.

- For large data sets, we usually can reduce the data size remarkbly and still get a good representation of the original isosurface. The advantage of the techniques can be seen more clearly when the data size gets larger.

- Most of the time spent in the preprocessing is used for mesh generation.

- The QMC isosurface extraction has a linear speedup with decreasing the number of QMC points used.

The preprocessing of the QMC isosurface extraction might be time consuming. However, this is a one-time process. After it is done, the postisosurface extraction is very fast.

Generally the QMC isosurface extraction technique does not require preknowledge about the data. Sometimes this technique may distort the boundaries of the original isosurface, if there is not enough QMC points generated along the boundaries. Some preknowledge about the data set can improve the precision of the isosurface extracted by QMC method.

Future work should apply this technique to large time-dependant data. Also, more QMC points should be generated in some crucial regions (such as boundaries) to yield more precise representation of such regions.

REFERENCES

[1] J. ED Akin, *Finite Element Analysis for Undergraduates,* Academic Press, 1986.

[2] K. Bathe, *Finite Element Procedures,* Prentice Hall, 1996.

[3] J. L. Bentley, *Multidimentional Binary Search Trees Used for Associative Search*, Communications of the ACM, vol. 18, no. 9 (1975), pp. 509-516.

[4] P. Cignoni, C. Montani, E. Puppo, R. Scopigno, *Optimal Isosurface Extraction from Irregular Volume Data*, Proceedings of Symposium on Volume Visualization '96, ACM SIGGRAPH, 1996, pp. 31-38.

[5] R. S. Gallagher, *Computer Visualization: Graphics Techniques for Scientific and Engineering Analysis*, CRC Press, 1995.

[6] R. S. Gallagher, *Span filter: An Optimization Scheme for Volume Visualization of Large Finite Element Models*, Proceedings of Visualization '91, IEEE Computer Society Press, 1991, pp. 68-75.

[7] M. Giles, and R. Haimes, *Advanced Interactive Visualization for CFD*, Computing Systems in Engeering, vol. 1, no. 1 (1990), pp. 51-62.

[8] Kenneth H Huebner and Earl A. Thornton, *The Finite Element Method for Engineers, Second Edition*, A Wiley-Interscience Publication, 1982.

[9] Alexander Keller, *The Quasi-Random Walk*, Algorithms and Complexity for Continuous Problems, Dagstuhl-Seminar Report, 1996, pp. 8-9.

[10] T. Itoh, and K. Koyamada, *Isosurface Generation by Using Extrema Graphs*, Proceedings of Visualization '94, IEEE Computer Society Press, 1994, pp. 77-83.

[11] T. Itoh, and K. Koyamada, *Volume Thinning for Automatic Isosurface Propagation*, Proceedings of Visualization '96, IEEE Computer Society Press, 1996, pp. 303-310.

[12] D. Kincaid and W. Cheney, *Numerical Analysis: Mathematics of Scientific Computing*, Brooks/Cole Publishing Company, 1991.

[13] Y. Livnat, H.W. Shen and C.R. Johnson, *A Near Optimal Isosurface Extraction Algorithm for Structured and Unstructured Grids*, IEEE Trans. Vis. Comp. Graphics, vol. 2, no.1 (1996), pp. 73-84.

[14] W. E. Lorensen, and H. E. Cline, *Marching Cubes: a High Resolution 3D Surface Construction Algorithm*, Computer Graphics, vol 21, no. 4 (1987), pp. 163-169.

[15] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*, SIAM Press, 1992.

[16] S. H. Paskov and J. F. Traub, *Faster Evaluation of Financial Derivatives*, The Journal of Portfolio Management, vol. 22 (1995), pp. 113-120.

[17] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, *Numerical Recipes*, Cambridge University Press, 1987.

[18] H.W. Shen and C.R. Johnson, *Sweeping Simplices: A Fast Iso-surface Extraction Algorithm for Unstructured Grids*, IEEE Visualization '95, 1995, pp. 143-150.

[19] H.W. Shen, C.D. Hansen, Y. Livnat, and C.R. Johnson, *Isosurfacing in Span Space With Utmost Efficiency (ISSUE)*, IEEE Visualization '96, 1996, pp. 287-294.

[20] K. Sikorski, and G. Schuster, *Application of the Hammersley Quadrature Algorithm to 3-D Migration*, 1991 Annual UTAM Report, University of Utah, 1991, pp. 234-259.

[21] Y. Sun, G. T. Schuster and K. Sikorski, *A Quasi-Monte Carlo Approach to 3-D Migration: Theory*, Geophysics, vol. 62, no. 3 (1997), pp. 918-928.

[22] V. N. Temlyakov, *Approximation Recovery of Periodic Functions of Several Variables*, Math. USSR Sbornik, vol. 56 (1987), pp. 249-261.

[23] V. N. Temlyakov, *Private Communication*, 1991.

[24] V. N. Temlyakov, *On Approximation Recovery of Functions With Bounded Mixed Derivatives*, Journal of Complexity, vol. 9 (1993), pp. 41-59.

[25] V. N. Temlyakov, *Approximation of Periodic Functions*, Nova Sci., Compmack, New York, 1993.

[26] J. E. Traub, and H. Wozniakowski, *Breaking Intractability*, Scientific American, vol. 270 (1994), pp. 102-107.

[27] S. Ueng, K. Sikorski, and K. Ma, *Efficient Streamline, Streamribbon, and Streamtube Constructions on Unstructured Grids*, IEEE Trans. on Vis. and Comp. Graphics, vol. 2, no. 2 (1996), pp. 100-110.

[28] S. Ueng, *Scientific Visualization for Finite Element Analysis Data Sets*, Ph.D Thesis, University of Utah, 1996.

[29] S. Ulam, *Adventures of a Mathematician*, Charles Scribner's Sons, 1974.

[30] S. Ulam, *Adventures of a Mathematician*, University of California Press, 1991.

[31] J. Wilhelms, and A. V. Gelder, *Octrees for Faster Isosurface Generation*, ACM Transactions on Graphics, vol. 11, no. 3 (1992), pp. 201-227.

[32] H. Wozniakowski, *Average Case Complexity of Multivariate Integration*, Bulletin AMS, vol. 24 (1991), pp. 185-194.

[33] H. Wozniakowski, *Average Case Complexity of Linear Multivariate Problems, Part I: Theory; Part II: Application*, Journal of Complexity, vol. 8 (1992) pp. 337-372, 373-392.