INTERACTIVE FAST RANDOM ACCESS, RETRIEVAL, AND NAVIGATION OF LARGE DATASETS

by

Zihong Fan

A Dissertation Presented to the
FACULTY OF THE VITERBI SCHOOL of ENGINEERING
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
Ph. D.
(ELECTRICAL ENGINEERING)

June 2011

Dedication

To my mom.

Acknowledgements

My graduate study at both USC and Stanford has been one of the most memorable periods of my life. I feel very fortunate to have learned from and worked with so many incredible people. Firstly, I would like to thank my advisor Professor Antonio Ortega for his excellent guidance, mentorship, and support for my research. It is thanks to him that I have had the opportunity to work on a new research area and enjoy it as much as I have. He has always been enthusiastic to discuss my research and has continually provided superb suggestions and criticisms which have greatly improved the quality of my work. I would also like to thank Professors Richard Leahy and Shanghua Teng for being on my dissertation committee, as well as Professors Giuseppe Caire, C.C. Jay Kuo, and Shrikanth Narayanan for being on my qualifying exam committee. It is indeed a great privilege and honor to have all their advice on my work.

I am also grateful to Professor John Pauly at Stanford, who encouraged me to pursue Ph.D. studies and has inspired me with his innovative thinking and creative attitude towards research. Professors Uel Jackson McMahan and David Ress, at Stanford and Brown Universities respectively, also provided me with my first real research experience and I am very grateful to them both.

In addition, I am deeply indebted to many current and past members of the Compression Research Group at USC, especially Ngai-Man Cheung, Roger Pique, and Sunil Narang. Very importantly, I would like to thank all of my dear friends for making the years enjoyable and for the treasured support they have given me during hard times: Shuguang Cui, Da Ha, Yunsong Huang, Quanzheng Li, Huazhou Liu, Yousong Mei, Michael Padilla, Shuxue Quan, Ran Ren, Yun Wang, Bo Wu, Junfeng Xu, and many other wonderful friends too numerous to mention here.

Finally, I would like to express my deepest gratitude to my family. In particular, I devote all my work to my mom who devoted her life to me and passed away four years ago without seeing my graduation.

Table of Contents

List Of Tables

List Of Figures

Abstract

This research is motivated by two important trends. First, more than ever before large amounts of data and information are being accessed through mobile devices such as smart phones, tablet computers, book readers, etc. Second, significant amounts of complex and high-volume information (e.g. maps, medical images, scientific datasets, virtual museums, etc.) are now available over networks. In addition to entertainment applications (e.g. video sharing), a significant driver of traffic over networks is likely to come from professional applications. Such applications, like those that might provide doctors with pervasive access to medical information, will demand high quality performance according to a variety of different metrics, such as latency, resolution, interactivity, and perceptual quality.

We have proposed a novel system for interactive, fast, and random access and navigation of large datasets. Of particular interest in our system is the random retrieval of lower dimensional data from high dimensional datasets. The system makes it possible to allow limited-memory mobile devices to quickly access complex large datasets over low-bandwidth connections. This approach can reduce the transmission rate dramatically (by factor of 2) and also improve the level of interactive navigation. Both 2D and 3D systems, referring to the dimensionality of the datasets that may be accessed, for fast and random access have been developed and each system is composed of a tiling scheme, linear searching algorithm, compression methodology, mapping algorithm, and reconstruction scheme. The mapping algorithms can also be applied to various other applications and areas. We also have proposed models for parameter selection and system optimization, tools to both analyze and quantify the benefits of our proposed re-mapping algorithm as well as wavelet based approach 3D system. This work offers both theoretical and practical contributions and can be used in a wide-array of exciting future applications.

Chapter 1

Introduction

Motivation: Our research is motivated by two important trends. First, more than ever before large amounts of data and information are being accessed through mobile devices such as smart phones, tablet computers, book readers, etc. Second, significant amounts of complex and high-volume information (e.g. maps, medical images, scientific datasets, virtual museums, etc.) are now available over networks. In addition to entertainment applications (e.g. video sharing), a significant driver of traffic over networks is likely to come from professional applications. Such applications, like ones that might provide doctors with pervasive access to medical information, would demand high quality performance according to a variety of different metrics, such as latency, resolution, interactivity, and perceptual quality. Interactive navigation of large high-dimensional media datasets aims at allowing viewers to freely navigate content, selecting a subset of the high-dimensional data of interest for display. With current techniques, the visualization of volumetric data with random access poses significant technical challenges even in 3 dimensional case.

We propose a novel system for interactive, fast, and random access navigation of large datasets. Of particular interest in our system is the more challenging problem, random retrieval of lower dimensional data from high dimensional datasets. One potential 2D application can be with map data, in which the requested portion covering a route may be retrieved first and/or with higher resolution, followed by other parts of the map. In a volumetric image example, arbitrary oblique planes from the volume may need to be extracted and rendered, as is required in some medical imaging applications. For example, doctors may wish to navigate and visualize any slide from a large 3D volume data as shown in Figure 1.1. Our

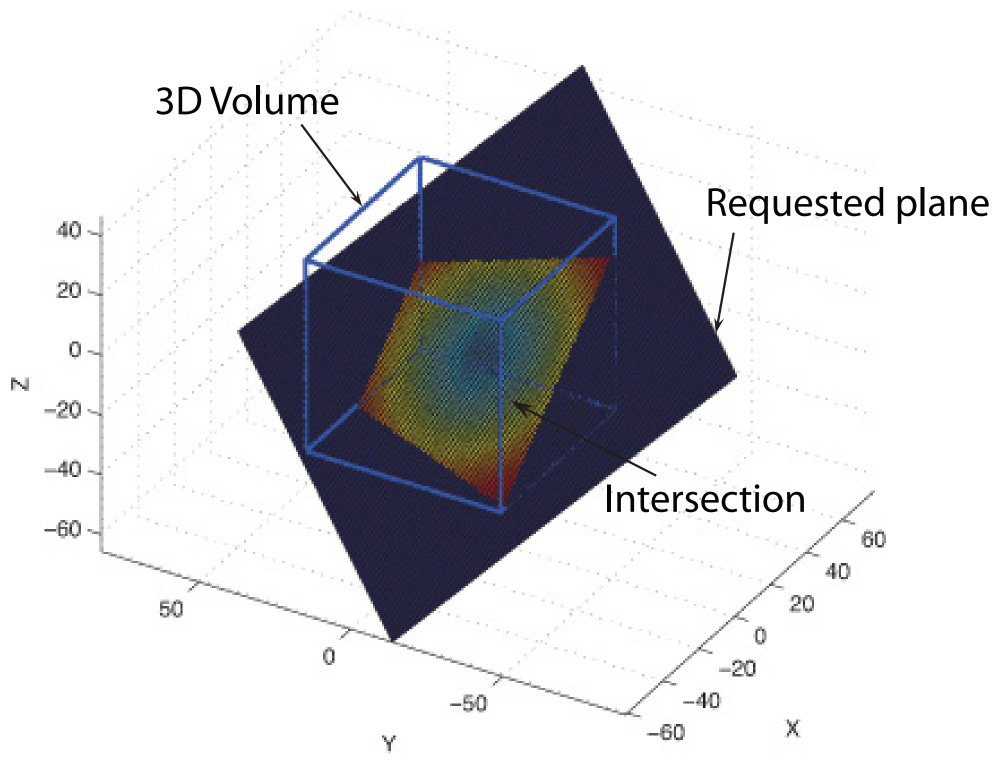


Figure 1.1: Random oblique plane acquisition illustration.

proposed methodology makes it possible to allow limited-memory mobile devices to quickly access complex large datasets over low-bandwidth connections. This approach can reduce the transmission rate dramatically (by factor of 2) and also improve the level of interactive navigation. We will illustrate our proposed method for both 2D and 3D cases, but our method can be extended to higher dimensions with some modifications.

Background and Problems: In many fields, there is a high demand to manipulate and visualize very large datasets. Often it is not practical for a personal computer to be equipped with sufficient memory so as to enable manipulation, visualization and rendering of the complete dataset. In these kinds of applications, a client-server approach can be more effective, with the server providing only the data needed for the specific visualization task at the client. Such client-server approaches are widely used, e.g., for interactive viewing of maps at various resolutions, and are often supported by tiling techniques, so that the server provides only those tiles corresponding to data requested by the client. Tiling-based techniques efficient when most of the information included in the tiles is used for display. For the challenging problem we tackle, the conventional tiling techniques can be very inefficient. This is because the conventional tiling techniques tile the object into non-overlapped regular small tiles. For example, the techniques proposed for volumetric image coding [3, 4, 12, 14, 16], including approaches such as JP3D [1, 2, 15], where a non-overlapped, independently encoded, cuboid tiling is used in order to assist random access. This scenario is shown in Figure 1.2, where the 3D volume dataset has been tiled into small non-overlapped cubes. The inefficiency is due to the fact that for each retrieved cubic tile the only voxels[1] that are “useful” are those near the intersection between the cube and the desired 2D plane. Because tiles are the basic unit, complete tiles have to be retrieved, even in cases when just a small number of voxels in each tile will be used for display. As we shall see from the examples in Figure 1.2 and Figure 1.3, it is more efficient to use overlapping tiles to represent the data-set (our approach here uses rectangular tiles with different orientations).

[1]a volume element, representing a value on a regular grid in three dimensional space.

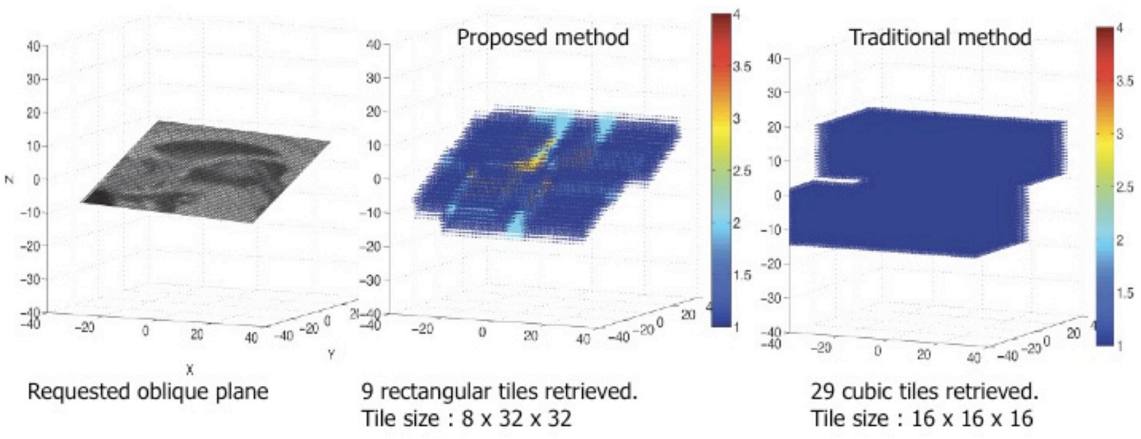


Figure 1.2: 3D Example: retrieve a random plane from a 3D data set. The colors represent the number of tiles covering the pixels. Tiles used in the rectangular tiling scheme are overlapped, and the tiles used in the cubic tiling scheme are non-overlapped. Much less voxels from the proposed method (rectangular tiles) are needed comparing to the traditional method (cubic tiles).

As another example in 2D, it may be necessary to extract narrow "bands", sets of parallel lines of arbitrary orientation, from complete 2D images in various cartographic or medical imaging applications. Similarly, lower transmission bitrates may be achieved by using rectangular tiles which lie along the area of interest (Figure 1.3). As we shall see, by using overlapped rotated tiles to represent the dataset we are able to increase the average number of useful voxels per tiles so that the total number of tiles to be retrieved is smaller, leading to a lower transmission bitrate. Thus we trade-off increased storage at the server's side for lower bandwidth during the interactive access to the data-set and lower memory requirements at the client's side.

Our Novel Approach and Challenges: The two examples in Figure 1.2 and Figure 1.3 have illustrated the potential benefits of using overlapped tiling to represent a dataset. Our approach trades-off storage at the server's side for significant reductions in average transmission rate relative to the conventional cubic tiling techniques. The run-time memory space requirement can also be reduced accordingly at the client's side.

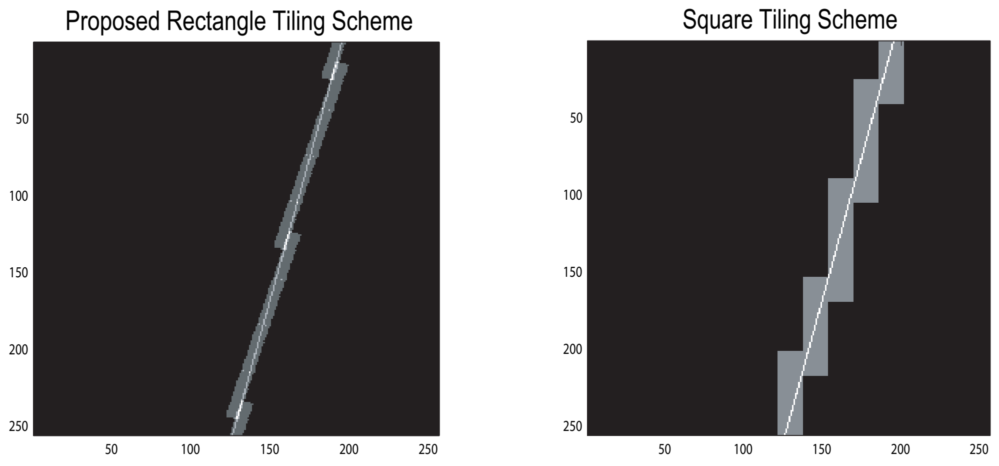


Figure 1.3: Comparison of rectangular and square tiling schemes in 2D case (where we retrieve a random line from a plane). Only 10 rectangular tiles are needed, instead of 21 square tiles (note that tiles have the same size 8×32 and 16×16, respectively).

There are many challenging problems that arise for this approach, particularly that of how to design the overlapped tiling scheme? If we choose to use rectangles (as we do here), the dominant question becomes where to locate the tiles and how to rotate the rectangular tiles to provide different orientations? What is the insight for the relationship between transmission rate and different tiling schemes? Can we find the best way to tile the object to achieve the best transmission gain? Given that the tiles are overlapped and hence there are a multiplicity of tile sets that can represent one query (the cover is not unique), how can we find the best tile set in linear time? In addition, transforms operate on pixels organized in a regular grid. In a rotated tile, the points on the rotated rectangular grid do not coincide with the original Cartesian grid points. How can we solve this problem without introducing any distortion and also without introducing additional computation time? The tiles may overlapped, how to display the queue, for example a random oblique plane in the 3D case?

In this work, we have developed both 2D and 3D systems for fast, random access, which address all the issues raised above. Each system is composed of a tiling scheme, linear searching algorithm, compression methodology, mapping algorithm, and reconstruction scheme. The tiling scheme provides multiple redundant tilings of the 2D image/3D volume dataset, where each tile has a different orientation. The searching algorithm determines which tiles should be retrieved for a given query in linear time and with minimum number of tiles necessary. The mapping algorithm enables efficient coding without interpolation of rotated tiles. Each tile is compressed and encoded independently using the compression techniques. The reconstruction and display component allows the acquisition and display of the requested queue from the selected overlapped rectangular tiles. The mapping algorithms can also be applied to various other applications and areas. In addition, we have proposed models for parameter selection and system optimization, a Haar transform based approach, which can reduce the transmission rate even more, as well as tools to both analyze and quantify the benefits of our proposed re-mapping algorithm. This work offers both theoretical and practical contributions and can be used in a wide-array of future applications.

This approach achieves a substantial reduction in the average transmission rate as compared to traditional square/cubic tiling. In the 2D case, the average number of bits transmitted when using the 2D tiling scheme can be reduced by a factor of 2 relative to the square tiling scheme and 15 - 40% additional transmission rate reduction can be achieved using an optimized 2D tiling scheme. In the 3D case, for different storage overhead as compared to the traditional cubic tiling scheme, the average number of bits transmitted can be nearly 15 - 55% lower using the 3D tiling scheme and an additional 10 - 30% reduction can be achieved in the average transmission rate using an optimized 3D tiling scheme. The reduction can be even greater by allowing additional storage overhead using different tile sizes.

Additionally, this approach leads to improved random access, with less storage and run-time memory required at the client.

The five components of the system are very different for the 2D and 3D cases. For each chapter, we first describe the 2D case to assist with the understanding of the methodology and we then emphasize on the more difficult 3D case. We start by introducing our overlapped tiling random retrieval system and describing the system components in Chapter 2. In Chapter 3, optimization of the tiling scheme will be presented. A prediction model will be introduced, which enables parameters selection without the need to implement the system with different parameter settings and also provide insights for designing new tiling schemes. With these insights, a new tiling scheme is proposed to achieve better performance. Chapter 4 will describe and evaluate in detail the mapping algorithm for rotated tile encoding, which will show that remapping without interpolation leads to overall much better RD performance and that the more symmetric the mapping is, the better RD performance can be achieved. Using a spectral graph theory approach, a tool will be proposed that can analyze and quantify the performance and benefits of our proposed re-mapping algorithm. In Chapter 5, a wavelet approach will be described, which can potentially achieve a 25% reduction in the average transmission rate at the cost of a factor of five on the server side. Finally, we present our conclusions as well as discuss future work in Chapter 6. Work described in this proposal has been published in [6–11].

Chapter 2

Overlapped Tiling Random Retrieval System and its Components

In this chapter, we start by introducing our overlapped tiling random retrieval system and then describing the details for each component. We will start with 2D case to assist with the understanding of the methodology and then present the 3D case. We consider tiling methods, parameterized by the location and orientation of the tiles, that can guarantee that all data can be retrieved. Because tiles overlap, there is no longer a unique way to retrieve data for a given query, and so we propose search algorithms in both 2D and 3D case to identify the most efficient set of tiles to be transmitted in response to a query (Sections 2.3.1 and 2.3.2). We also propose mapping techniques to compress the data points on the rotated tiles that do not coincide with the Cartesian grid points (Section 2.4).

2.1 System

In this section, we provide overviews of the systems, where for the 2D case, the goal is to retrieve arbitrary oblique lines from 2D images (A practical application can be retrieve a path from a map. We simplified the case as retrieving lines for

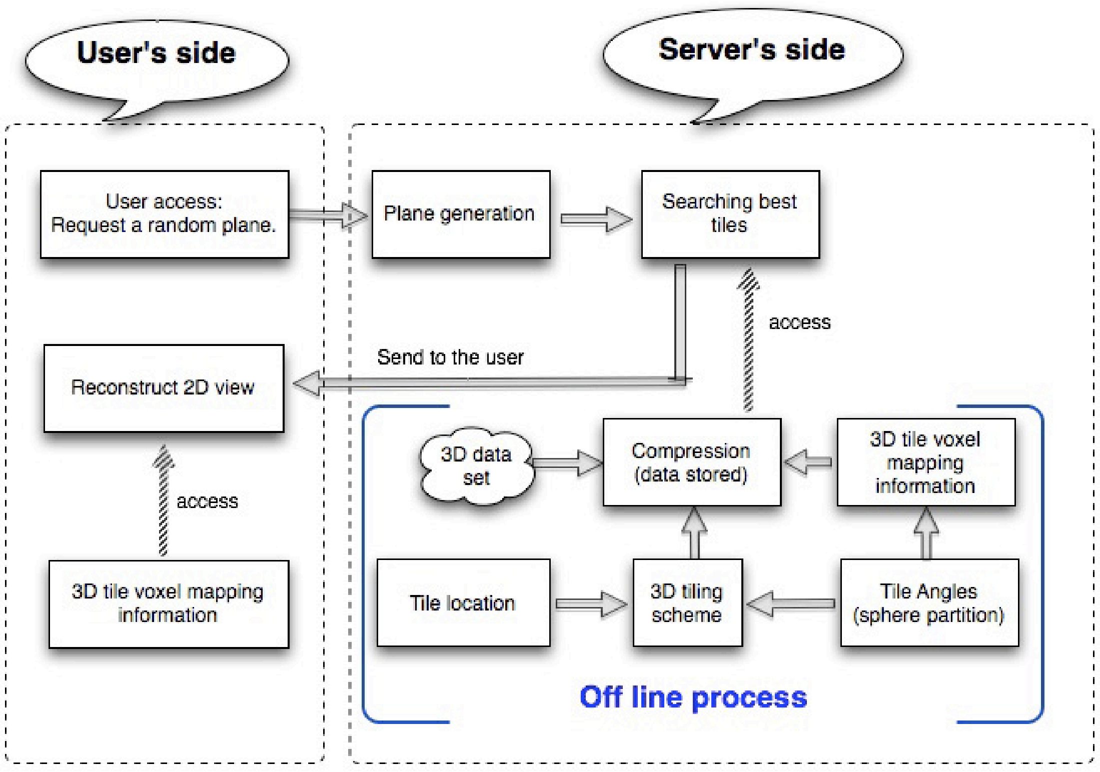


Figure 2.1: 3D Retrieval system

experiments.) and for the 3D case, the goal is to retrieve an arbitrary oblique planes from 3D volume datasets.

The structure of the whole 3D system is displayed in Figure 2.1. For the 3D case, on the server's side, the 3D dataset is tiled by overlapped 3D rectangular tiles with different orientations. The tiles are compressed and saved at the server, which is done offline. When a user requests a oblique random plane, the plane parameters are sent to the server, and the voxels that contribute to the plane are then recorded in order for server to find the associated tiles and transmit the compressed tiles back. After user receives the data, the 2D oblique plane can be reconstructed. Similarly, for the 2D case, on the server's side, the 2D image is tiled by overlapped 2D rectangular tiles with different orientations. The tiles are compressed and saved at the server, which is done offline. When a user requests a oblique random line or a band, the parameters are sent to the server, and the pixels contributing to the line/band are then recorded in order for server to find the associated tiles and transmit the compressed tiles back. After user receives the data, the line/band can be reconstructed. This system consists of 5 parts: 1) tiling scheme, 2) mapping algorithm, 3) tile searching algorithm, 4) compression, and 5) arbitrary oblique plane/line (band) reconstruction and display.

Our tiling scheme provides multiple redundant tilings of the 2D images/3D volume datasets, where each tiling has a different orientation. Since the tiles are overlapped, and there are many possible tile combinations to cover a query, we consider a solution to be "optimal" when the number of rectangles to cover the query is minimum. Note that several solutions may be optimal (different tile combinations that cover the query, using the same number of tiles). We propose searching algorithms (Section 2.3) to identify the most efficient set of tiles to be transmitted in response to a query. Mapping algorithms enables efficient coding without interpolation of rotated tiles (Section 2.4). Each tile is compressed and encoded independently. JPEG is used in 2D compression. 3D DCT, a 3D zig-zag scanning order and VLC are used in 3D compression experiments.

2.2 Rotated Center Based Tiling

The intuition behind the proposed redundant tiling is that if each pixel is available from more than one tile, one can achieve lower bandwidth on average by delivering the set of tiles that provide all requested pixels most efficiently (i.e., with lowest number of tiles required)[1]. Since queries of interest are lines (or sets of lines) at

[1]Note that in a non-redundant case the set of tiles required to answer a query is unique.

arbitrary angles in a 2D image, random planes from a 3D data set, or random hyperplanes from higher dimensional data set, this suggests that overlapped tiles with different locations should be used. While square tiles with well designed overlap can be used, in this work, we use rectangular tiles with different rotation angles. The potential benefits of the proposed method using rectangular tiles are illustrated in Figure 1.3 for the 2D case. It can be seen that fewer tiles have to be fetched when rectangular rotated tiles are used (in this example rectangular and square tiles contain the same number of pixels). Clearly, the number of tiles to be transmitted will depend on the data being requested.

Using rotated rectangular tiles will lead to a lower average transmission rate because each selected tile will lie along the requested line/plane/hyperplane, which will lead to increases in the average number of pixels contained in the intersection between line/plane/hyperplane being requested and the "best matching tiles".

For the 3D case, in our work, 3D rotated rectangular tiles (flat tiles, two sides are big and one side is small) are used. Using rotated rectangular tiles will lead to a lower average transmission rate because each selected tile is more likely to lie along the requested plane, which will lead to increases in the average number of voxels contained in the intersection between planes being requested and the "best matching tiles". The tiling scheme can be extended to higher dimensions too. For example, if there is need for extracting a line (or a pipe) from 3D volume data, the tile can be designed as 3D thin rectangular (tall tiles, one side is big and two sides are small) or with cylinder shape. There are many different ways to design the tile shape, which depends on the application and the dimension of the datasets. In this section, we will present one way of the tiling schemes for both 2D and 3D cases. In Chapter 3, two prediction models will be presented and we will describe approaches to optimize tiling.

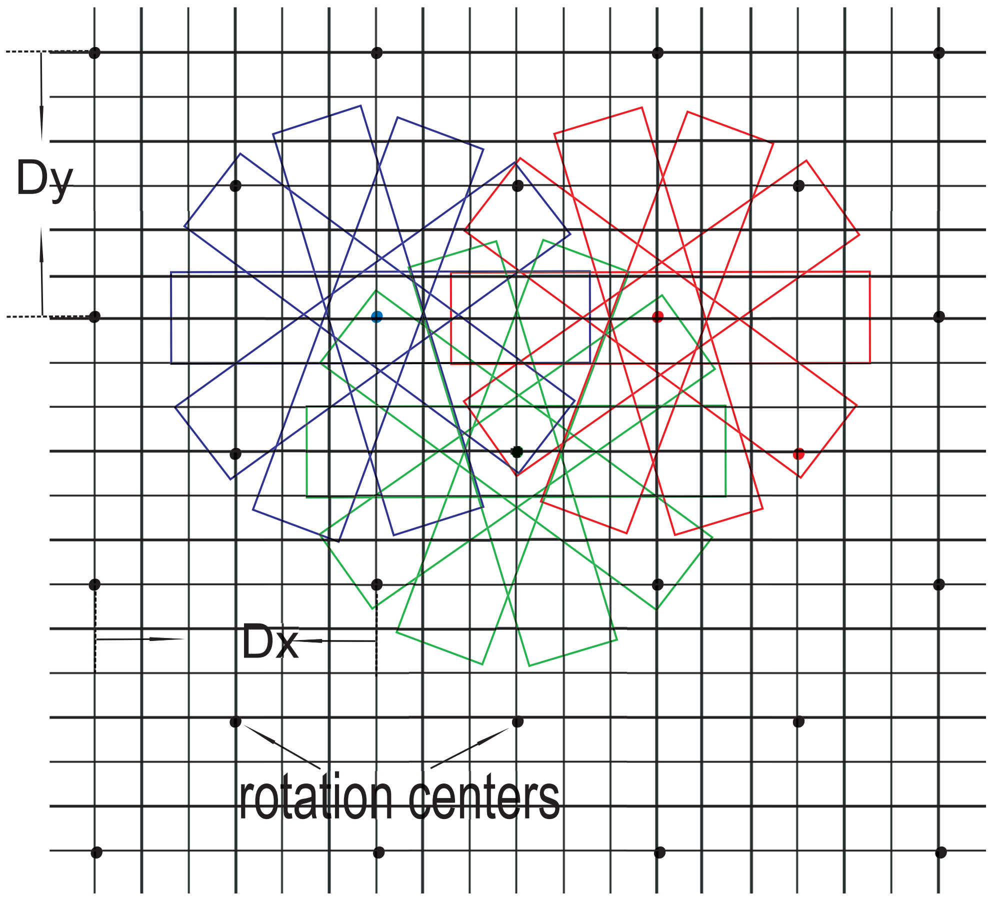


Figure 2.2: Example of proposed overlapped rectangular tiling.

2.2.1 2D Case

The tiling layout we first considered in the proposed system can be described by the width W and length L of each rectangle, the number N of rotated rectangles associated with each rotation center, and D_x and D_y respectively, and the horizontal and vertical distance between rotation centers on the same Cartesian horizontal or vertical grid, as illustrated in Figure 2.2. The even/odd row and column rotation centers are interleaved with shifts of $D_x/2$ horizontally and $D_y/2$ vertically. At each rotation center, the N rotated rectangles are uniformly spread out. We test different configurations for these parameters, always chosen so that each pixel in the image

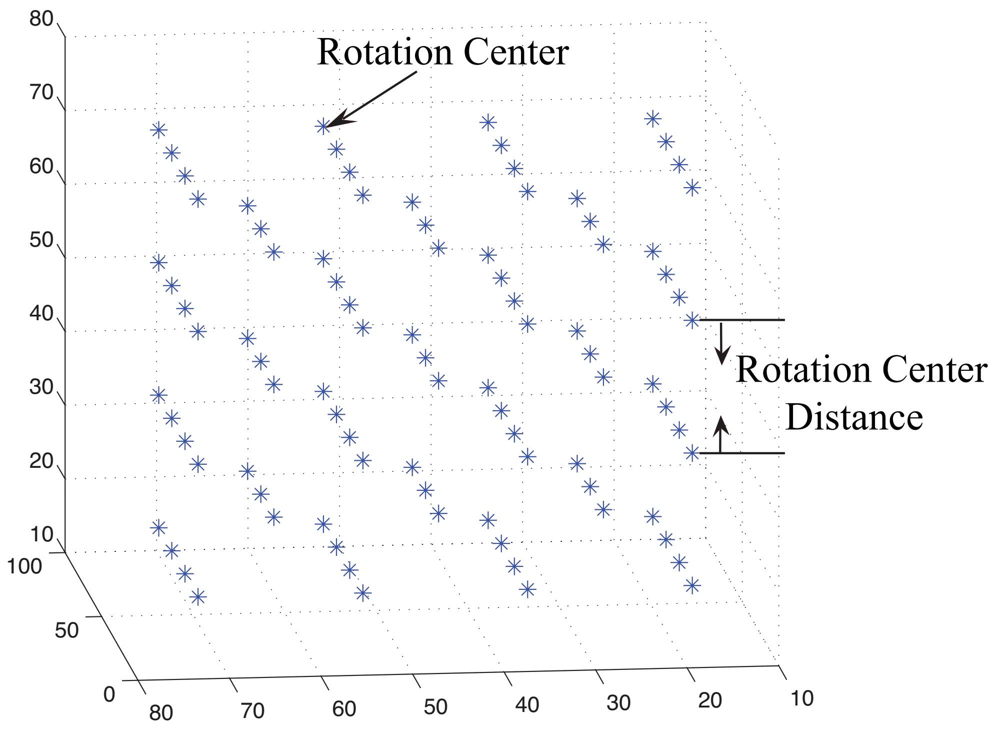


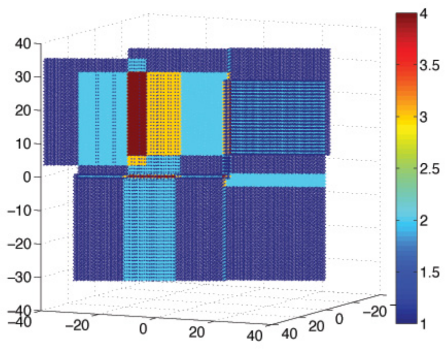
Figure 2.3: Example of proposed overlapped rectangular tiling.

is covered by at least one tile. Each rectangular tile is compressed independently for storage. The parameters can be chosen to achieve different trade-offs between transmission efficiency and storage overhead.

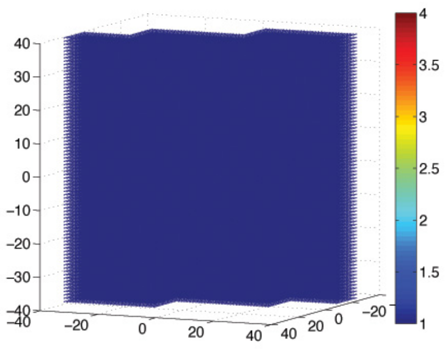
2.2.2 3D Case

Similar to the 2D case, for the 3D case, at each rotation center in Figure 2.3, N rotated 3D rectangles are uniformly spread out. We test different configurations for these parameters, always chosen so that each voxel in the 3D volume is covered by at least one tile. Each 3D rectangular tile is compressed independently for storage.

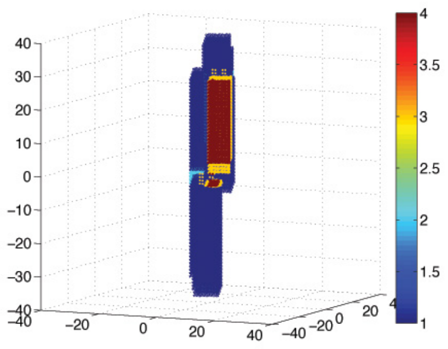
Figure 1.2 illustrates the potential benefits using a 3D example (retrieve a random plane from a 3D data set). Just 9 rectangular tiles are needed, instead of 24



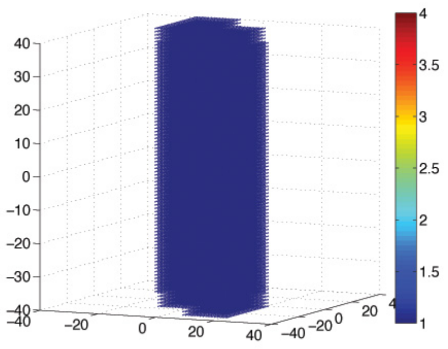
(a) Front view: rectangular tiling scheme result. The flat rectangular tiles are overlapped.



(b) Front view: cubic tiling scheme result. The cubic tiles do not overlap.



(c) Side view: rectangular tiling scheme result. The flat tiles lie alone the plane, so we can see the tiles are very thin from this side view of the requested plane.



(d) Side view: cubic tiling scheme result. The tiles are cubic shape, so the tiles are thick from this side view of the requested plane.

Figure 2.4: Comparison of rectangular and cubic tiling schemes in 3D case(retrieve a random plane from a 3D data set). Just 9 rectangular tiles are needed, instead of 24 cubic tiles (note that tile sizes are close, $32 \times 32 \times 8$ and $20 \times 20 \times 20$, respectively). The colors represent the number of tiles covering the pixels. Tiles used in the rectangular tiling scheme are overlapped, and the tiles used in the cubic tiling scheme are non-overlapped.

cubic tiles. The rectangular tiles have size $32 \times 32 \times 8$ and the tile size for the cubic tile is $20 \times 20 \times 20$. The colors represent the number of tiles covering the pixels. Tiles used in the rectangular tiling scheme are overlapped, and the tiles used in the cubic tiling scheme are non-overlapped. From Figure 2.4(a), we can see that the flat rectangular tiles are overlaped and from Figure 2.4(b), the cubic tiles do not overlap. Because by using the rectangular tiling scheme, the flat tiles lie alone the plane, we can see the tiles look very thin from Figure 2.4(c), side view of the requested plane. For the cubic tiling scheme, tiles are cubic shape, so the tiles are thick from Figure 2.4(d), side view of the requested plane. It can be seen that fewer voxels have to be fetched when rectangular rotated tiles are used. Clearly, the number of tiles to be transmitted will depend on the data being requested. Figure 2.4 also illustrates the lower transmission rate that can be achieved in 3D case, where a random plane is retrieved from a 3D data set. By showing the front view and the side view of the selected tiles in Figure 2.4, we can see that even though overlapping exists in the rectangle tiling scheme, as compared with the traditional cubic tiling scheme (seen in the front views), a significant amount of transmission bits can be reduced using our proposed method (seen in the side views and the retrieved number of the tiles).

Figure 2.5 illustrates the 3D tiling scheme. The tiling layout in our proposed system can be described by the width W, length L and height H of each 3D tile, the number N of rotated rectangles associated with each rotation center, and D_x, D_y and D_z respectively, the distance between rotation centers along the x, y and z directions on the Cartesian grid. Here we choose $W = L$ and $H < W$. Uniformly rotated 3D tiles can be obtained by making their norm vectors (refer to Figure 2.6) be spread uniformly in a sphere around the rotation center. This means that the minimum angle between any two tiles is Δ_α. In Figure 2.6 (b), the norm vectors are

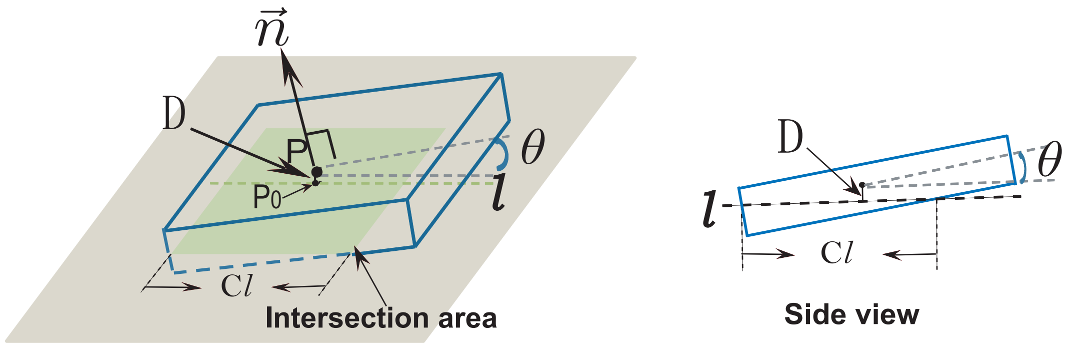
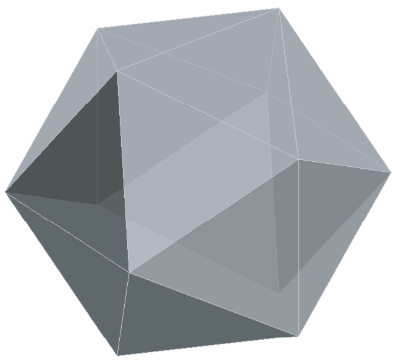


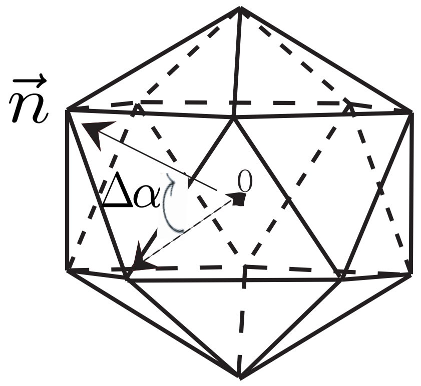
Figure 2.5: 3D tile demonstration: D is the distance (between P and P_0) from the rotation center of the tile to the plane. θ is the angle between the tile and the plane. $\vec{n}$ is one norm vector of the rectangle tile. The other norm vector is opposite with $180°$.

represented with a common origin and with their end points uniformly spread out on a 3D sphere, so that sphere tessellation methods can be used for designing the tile angles. Therefore, icosahedron, octahedron and tetrahedron sphere tessellations with different refining levels can be used for choosing the directions of the uniformly rotated rectangular tiles. When two vectors with exactly opposite directions are part of the tesselation, only one of them is used as the norm vector of a tile. For 3D case, in order to try to let the rotation centers located in a uniform way, the pattern of the rotation centers could be the regular patten like, cuboctahedral, octahedral or tetrahedral, etc. and the rotation angles can be different for each rotation centers. In this Chapter, we locate the rotation centers at points on a regular octahedron grid patten (the rotation centers are located at the regular octahedron vertices) and at each rotation center, the N rotated rectangles are uniformly spread out by using equal sphere tessellation method [5]. Cases using different patterns for the rotation centers and using different rotation angles will be discussed in future work.

The 3D tile angle selection is illustrated by Figures 2.6 and 2.7. The number of tile angles obtained by using the different tessellation methods are shown in Table 2.1. Different tile angle settings can be used associated to different rotation



(a) Icosahedron sphere tessellation: refinement level 0



(b) Tile direction vectors $\vec{n}$

Figure 2.6: Angles selection for 3D tiles: Note (b) shows points from both possible $\vec{n}$ vectors associated with a given tile, contrary to the definition, for illustrative purposes of the geometry.

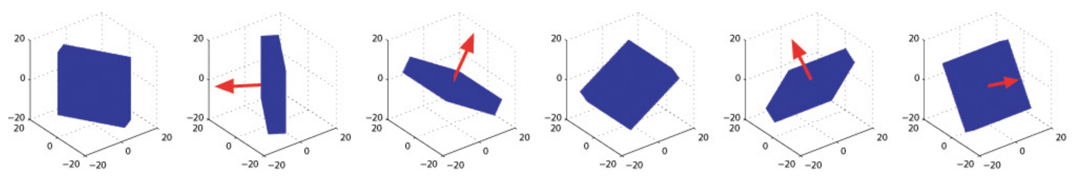


Figure 2.7: The number of the tiles at each rotation center by using icosahedron sphere tessellation: 6 in total

	Ico: L0	Ico: L1	Oct: L0	Oct: L1	Oct: L2	Tetra: L1	Tetra: L2
No. Vertices	12	42	6	18	66	10	34
No. Tile angles	6	21	3	9	33	7	24

Table 2.1: Ico: icosahedron tessellation; Oct: octahedron tessellation; Tetra: tetrahedron tessellation. L means refining level.

centers. In this work, when different tile angle settings are used, the rotation centers with different tile angle settings are interleaved. For example, the close neighbor rotation centers have different number of tiles rotated around of them. For the tiling layout in this proposed system (a better layout in Chapter 3), parameter configurations which can cover all the voxels in the 3D object are chosen. For example, if for some values of D_x, D_y, D_z and N, not all the voxels of the 3D object are covered, these parameter settings will not be used.

2.3 Search Algorithm

As discussed in Section 2.2.1, the tiles are overlapped. Overlap between rectangles means that there are multiple ways (different rectangle sets) to provide all the information corresponding to a line in 2D image, or a plane in 3D volume. This leads to a set covering problem with the objective of finding which representation, ie., which collection of rectangles, is best for transmission. Normally, set covering problems are NP problems, potentially resulting in considerable complexity. Our particular problem, however, has certain characteristics that allow for a feasible algorithm. We propose a fast search algorithm to solve the problem that is *optimal* in the sense that it *minimizes the number of rectangles to cover the line.* This is a reasonable criterion, because the total transmission rate needed will be roughly proportional to the number of rectangles that have to be retrieved. Note that several solutions may be optimal (different rectangle combinations that cover the query, using the same number of rectangles).

2.3.1 Optimal Fast 2D Rectangle Search Algorithm

Our particular problem in the 2D case has certain characteristics that allow for a feasible algorithm. Specifically, each rectangle that crosses over the line contains a contiguous line segment of the straight line we are trying to cover. For example in Figure 2.8, tile R_1 contains the line segment between points S_1 and S_2. It is possible to have several optimal solutions (different rectangle combinations that cover the line, using the same number of rectangles). For example, regardless of whether one starts searching at the beginning or end of the line[2], an optimal solution (achieving the minimum number of the tiles to cover the line, even if those tiles themselves are different) will be found. While greedy, Algorithm 1 can be shown to provide a globally optimal solution (where optimality is as defined above). At each step only the rectangles that are close to the line and the starting point S_i, need to be searched.

Proof of optimality for Algorithm 1. Let $\{R_1, R_2, R_3, \ldots, R_n\}$ be the n rectangles obtained by our algorithm to cover the line. Let $\{L_1, L_2, L_3, \ldots, L_m\}$ be the m rectangles selected by any other technique. To prove that our algorithm is optimal, we need to show that $m \geq n$. Assume that for each index $i < n$

$$\mathcal{C}\{L_1, L_2, ...L_i\} \subseteq \mathcal{C}\{R_1, R_2, ...R_i\}, \tag{2.1}$$

[2]In Figure 2.8, $\{S_1, S_2, S_3, \ldots\}$ are the starting points resulting from applying the cover searching algorithm starting from the "beginning" of the line. These correspond to the points $\{\hat{S_1}, \hat{S_2}, \hat{S_3}, \ldots\}$ when the algorithm is applied starting from the "end" of the line.

Algorithm 1 Optimal Rectangle Cover Searching Algorithm

```
1: Starting with the beginning point of the line ($S_1$ in Figure 2.8), find the rectangle that covers the starting point ($S_1$) and maximizes the number of line points covered.
2: Set the point in the line that follows the last point covered by the last rectangle chosen as the new starting point $S_i$ for the $i^{th}$ iteration (e.g. $S_2$ is the new starting point for the second iteration in Figure 2.8).
3: Record the rectangle and mark the points in the line that covered by the rectangle as "covered".
4: Repeat steps 1 to 3 with the new starting point $S_i$ until all the points in the line are marked as "covered".
```

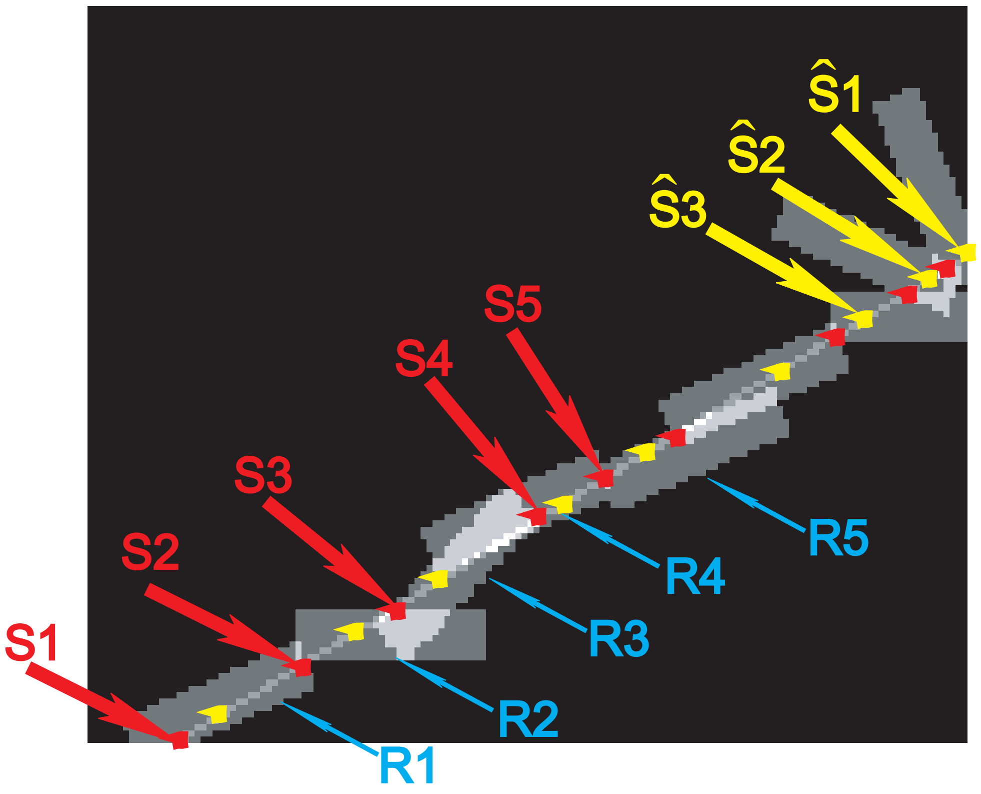


Figure 2.8: 2D Demonstration of the optimal rectangle cover searching algorithm.

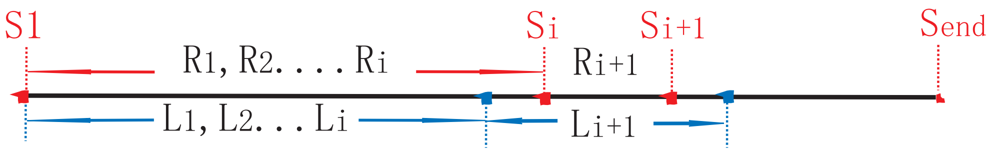


Figure 2.9: Demonstration of the proof for Part 1.

where $\mathcal{C}$ denotes the (contiguous) portion of the line covered by the corresponding rectangles[3]. With this assumption, it follows that we also have

$$\mathcal{C}\{L_1, ..., L_i, L_{i+1}\} \subseteq \mathcal{C}\{R_1, ..., R_i, R_{i+1}\}. \tag{2.2}$$

This is because (refer to Figure 2.9) if there existed a rectangle L_{i+1} satisfying the relationship

$$\mathcal{C}\{L_1, ..., L_i, L_{i+1}\} \supsetneq \mathcal{C}\{R_1, ..., R_i, R_{i+1}\}, \tag{2.3}$$

then the endpoint of L_{i+1} would extend beyond the endpoint of R_{i+1} and would also overlap with R_i. However, if such an L_{i+1} were to exist, then it would have been chosen by R_{i+1} given the steps of Algorithm 1. Statement (2.2) is thus proved by contradiction. Based on the induction of (2.1) and (2.2), we have that

$$\mathcal{C}\{L_1, ..., L_i, ..., L_n\} \subseteq \mathcal{C}\{R_1, ...R_i, ..., R_n\}. \tag{2.4}$$

It thus follows that $m \geq n$ and we see that the proposed algorithm is optimal under the assumption above. From our algorithm, we know that

$$\mathcal{C}\{L_1\} \subseteq \mathcal{C}\{R_1\}. \tag{2.5}$$

[3]Notational note: $A \subseteq B$ denotes that A is a subset of B and could be equal to B. $A \subsetneq B$ denotes that A is a proper (or strict) subset of B, and hence is a subset but is not equal.

Since R_1 covers S_1 and covers the line furthermost to the right, the assumption is true at the first step. Hence by induction based on (2.1), (2.2), and (2.5), the rectangle searching algorithm we propose is optimal according to our definition. $\square$

Additionally, an optimal solution will be found, regardless of whether one starts from either the beginning or end of the line. This may be easily shown as follows. In Figure 2.8, $\{\hat{S}_1, \hat{S}_2, \hat{S}_3, \ldots\}$ are the starting points resulting from applying the same cover searching algorithm when starting from the "end" of the line, while the points $\{S_1, S_2, S_3, \ldots\}$ correspond to the case when the algorithm is applied starting from the "beginning" of the line. By the proposed searching algorithm, for each iteration, the selected tile covers the starting point of that iteration and covers the greatest length of the line. Hence, the points $\{S_1, S_2, S_3, \ldots\}$ and $\{\hat{S}_1, \hat{S}_2, \hat{S}_3, \ldots\}$ are interleaved with one another. In other words, between S_i and S_{i+1} there must lie exactly one point $\hat{S}_k$. Similarly, between $\hat{S}_k$ and $\hat{S}_{k+1}$ there must lie exactly one point S_i. If this were not so, then there would be contradiction. The proof is shown in the following paragraph. Therefore the optimal solution is obtained regardless of the starting point being at the "beginning" or "end" of the line. Similarly, we can show that if we start the searching algorithm from any point of the line between the end points that the number of selected tiles will be either the minimum or the minimum + 1.

Proof of optimality for Algorithm 1, regardless starting point of the line. Refer to Figure 2.9, let $\{R_1, R_2, R_3, \ldots\}$ be the rectangles obtained by our algorithm starting from the "beginning" of the line and the points $\{S_1, S_2, S_3, \ldots\}$ are the associated staring points. Let $\{L_1, L_2, L_3, \ldots\}$ be the rectangles obtained by our algorithm starting from the "end" of the line and let the corresponding starting points be $\{\hat{S}_1, \hat{S}_2, \hat{S}_3, \ldots\}$ (not shown in Figure 2.9). To prove that our algorithm is optimal regardless of whether the starting points are chosen at the beginning or end of the line, we need to show that the starting points $\{S_1, S_2, S_3, \ldots\}$ and $\{\hat{S}_1, \hat{S}_2, \hat{S}_3, \ldots\}$ are interleaved with one another.

Assume that both S_i and S_{i+1} were between $\hat{S}_k$ and $\hat{S}_{k+1}$ (referring to Figure 2.9). This would mean that

$$S_i, S_{i+1} \subseteq \mathcal{C}\{L_k\}, \tag{2.6}$$

and

$$\mathcal{C}\{R_i\} \subseteq \mathcal{C}\{L_k\} \tag{2.7}$$

This tells us that the selected tile L_k (not R_i) would be the tile that covers S_i and covers the greatest amount of the line. This is contradict with our algorithm. Thus the number of points S_i is equal to the number of points $\hat{S}_k$ and they are interleaved with one another. □

2.3.2 Fast 3D Rectangle Search Algorithm

Our 3D rectangle search algorithm is similar to Algorithm 1 defined for the 2D case. Instead of sorting the points associated with a line as in the 2D case in Algorithm 1, the points associated with the plane are sorted in order of decreasing distance to the center point of the plane (this sorted list of points is called List A.) In Algorithm 2, we start by finding the tile that covers the first element of List A (the furthest point from the center of the plane) *and* covers the largest number of points in the list. Then the selected tile will be recorded and all the points covered by the selected tile will be removed from List A. The procedure is then repeated on the updated List A, until all the points associated with the plane are covered (i.e., List A is empty). While it cannot be guaranteed that this algorithm is optimal in the 3D case, the resulting transmission rate can be significantly reduced by using this search algorithm (as will be seen from our experiments in Section 2.6.2.) Additionally, this search algorithm is very fast, since we only need to search for the rectangles located close to the plane.

Algorithm 2 3D Rectangle Cover Searching Algorithm

```
1: Find the indices for the points associated with the requested plane. Sort the
   indices with decreasing order by calculating the distance from the points in a
   plane to the plane origin, calling the sorted list $A$.
2: Find tile centers which are close to the plane, calling the list $B$.
3: while $A$ is not empty do
4:    Starting with $A(1)$, the beginning point of the list $A$, find the tiles in $B$,
      which are close to $A(1)$, calling the list $C$.
5:    for i=1:length(C) do
6:       For the rectangle tiles associate with the tile position in $C(i)$, find the
         one that covers $A(1)$ and maximizes the number of the plane points in $A$
         covered , call this $R(i)$.
7:    end for
8:    MaxR = max(R);
9:    In $A$, remove the points covered by MaxR, the rectangle tile selected from
      the previous step. Update list $A$ and record the rectangle.
10: end while
```

2.4 Mapping Algorithm for Rotated Tile Encoding

We use the rotated rectangular tiles to represent the dataset. After we rotate a rectangular tile, the points on the rotated rectangle grid will not coincide with the original Cartesian grid points. A straightforward approach to represent the data would be to interpolate the values on the rotated rectangle grid before compression. Instead we propose to map the original values on the Cartesian grid into the rotated rectangular grid before compression.

For each point on the rectangle grid, there is a Cartesian point to map to. The distance between the mapping pair is called as “mapping distance”. Certain points, such as the rotation center, are in the same positions in both grids, while in other cases, the pixel in a given location in the rotated grid has been copied from a neighboring location in the Cartesian grid. The pixels are copied unchanged, i.e., no interpolation is performed. While many alternative mappings are possible, we are interested in methods that minimize the "mapping distance", by which pixels are displaced from their original positions for encoding. Clearly this is desirable as these displacements "distort" the frequency contents of the blocks prior to encoding. The details of the mapping algorithm in both 2D and 3D cases will be described in Chapter 4, as well as details of the mapping evaluation and analysis tools. Note that rectangles having the same rotation angles around their (different) rotation centers have rectangular grid points that are in the same locations relative to the Cartesian grid points in their neighborhood. Thus, we only need to store a small number of tables (one per angle) to specify the mapping.

2.5 Reconstruction, Display and Compression

Our redundant representation is based on representing 2D image/3D volumes with several sets of tiles. While the original tiling scheme is based on non-overlapping squares/cubes that are aligned with the axes of the Carditional basis in 2D/3D space, the tiles in our approach can have any position and orientation. As described in Section 2.4, pixels (in 2D)/voxels (in 3D) on the original Cartesian grid are directly mapped (without interpolation) to the nearest point in the rotated grid. Each tile (rotated or not) is encoded independently and stored on the server. The details of compression, reconstruction and display for both 2D and 3D cases will be described in the following sections.

2.5.1 2D Compression and Reconstruction

The number of pixels within a line are determined by the length of the line segment across the image, which is proportional to the size of the image. In general, if the length of the line segment is L (L is not an integer), $\llcorner L \lrcorner$ uniformly distributed samples on the line segment will be chosen. While other approaches could be used, in our implementation, the pixels' values within a line are calculated by using a simple "nearest neighbor" interpolation at the uniformly sampled positions on the line segment. We tile a 2D image (both Lena and MRI images were used) and use our search algorithm (Section 2.3.1) to identify the best tiles to display a random oblique line within the image. Distortion is measured by mean squared error (MSE) between the pixels in the line generated from the original image data and the line obtained by decoding and then interpolating compressed tile data. Rate is measured in terms of bits per pixel in a line (i.e., we compute the total rate required to transmit all the needed tiles and divide this by the total number of pixels in the decoded line). Both rectangular (8×32) and square (16×16) tiles are coded using JPEG tools (i.e. four 8×8 blocks are encoded using JPEG).

2.5.2 3D Compression

Each tile is divided into small blocks of size $8 \times 8 \times 8$ on which a 3D DCT is performed (similar to approaches in [17] and [18]). The coefficients are rounded to integers and a simple uniform quantizer is applied. The transformed coefficients are then scanned in a predetermined order within each block using the 3D zig-zag table shown in Figure 2.10. The coding method for DC/AC coefficients is similar to that used by JPEG. When encoding transform coefficients in three dimensions, different coefficient scanning orders will lead to different performances depending on the texture correlation along each direction. For example, when 3D compression is applied on video frames (where time is the other dimension), in still scenes, correlations along the time direction are very high. In this case, a two-dimensional zig-zag scan which is in the spatial image plane and repeats in the temporal direction will lead to lower compression rates compared to other possible scanning orders. In our case, we consider a 3D object dataset (e.g., MRI volume data) where the correlations along all three directions are statistically similar. Hence, a simple three-dimensional zig-zag scanning order is used in our application. As in [18], the DCT coefficient scanning order for a given three-dimensional block is in order of increasing distance from the DC coefficient for that block. The left bottom corner point represents the DC coefficient location (1,1,1)[4] The order after sorting will form a 3D zig-zag table. Only the first 9 points are shown in the figure for easier visualization. Entropy coding method for DC/AC coefficients is essential the same table used by JPEG, added one more symbol code to accommodate a larger dynamic range in the quantized coefficients.

2.5.3 Oblique Plane Acquisition and Display

When a user requests a random oblique plane, the plane parameters are sent to the server. The plane parameters are the norm vector of the plane $\vec{n}$ and any points (x, y, z) in the plane. Since the points close to the requested oblique plane are needed to interpolate the values on the plane, the server will find the Cartesian points within a small distance to the plane in order to search for the best tile combinations to cover the plane. Because it is in 3D the largest distance between neighboring pixels is $\sqrt{3}$. In our experiments, $\frac{\sqrt{3}}{2}$ is used as the distance range to

[4]The index starts from 1, instead of 0.

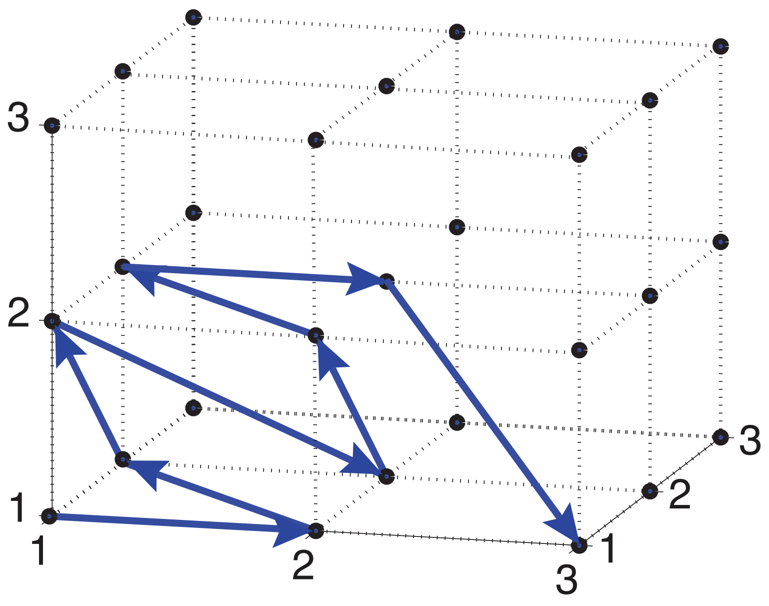


Figure 2.10: Three-dimensional scanning order illustration. Only the first 9 points are showed in order to be clearly visualized.

the plane. The compressed data[5] for the selected tiles is then transmitted to the user.

After the user receives the data, tiles are decoded. Since the tiles are partially overlapped, the voxel values which are covered by multiple tiles take values that are formed from averaging the values provided by each of the overlapping tiles. Note that the voxels here are on a Cartesian grid. In order to display the image with the same resolution[6] no matter what the plane orientation is, the requested oblique plane coordinate matrices need to be generated and then the values on the new grid can be interpolated by the received voxels on Cartesian grid (Note that the values on the rotated rectangular tile are mapped from the values on the Cartesian grid using our mapping algorithm) .

[5]These are saved on the server during offline process and are independent for each tile.
[6]Scaling will be considered separately.

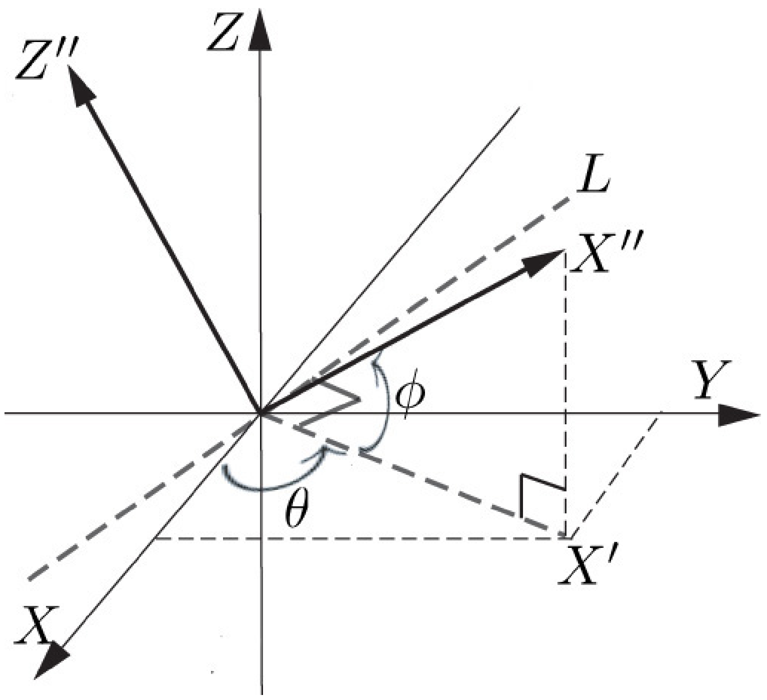


Figure 2.11: Three-dimensional coordinate rotation

An oblique plane can also be represented by $P(x_0, y_0, z_0)$ and θ, ϕ. $P(x_0, y_0, z_0)$ is the point on the plane closest to the origin. θ is the rotation angle in the X, Y plane and ϕ is the rotation angle from the X, Y plane to the Z plane. $P(x_0, y_0, z_0)$, θ and ϕ can be generated from the norm vector of the plane $\vec{n}$ and any point (x, y, z) in the plane. From Figure 2.11, the requested plane coordinate can be generated from the following steps. Let X, Y, Z represent the plane coordinate for each axis, where $Z = 0$. First, the coordinate are rotated in the X, Y plane by θ degrees as follows

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.8}$$

They are then rotated around line L by ϕ degrees denoted as R_ϕ as follows:

$$\begin{bmatrix} X'' \\ Y'' \\ Z'' \end{bmatrix} = R_\phi \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} \tag{2.9}$$

Line L passes through the origin and the point (u, v, w). (u, v, w) can be generated by (2.10):

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} & 0 \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{2.10}$$

with R_ϕ obtained as:

$$R_\phi = \begin{bmatrix} u^2 + (v^2 + w^2)\cos\phi & uv(1-\cos\phi) - w\sin\phi & uw(1-\cos\phi) + v\sin\phi \\ uv(1-\cos\phi) + w\sin\phi & v^2 + (u^2 + w^2)\cos\phi & vw(1-\cos\phi) + u\sin\phi \\ uw(1-\cos\phi) - v\sin\phi & vw(1-\cos\phi) + u\sin\phi & w^2 + (u^2 + v^2)\cos\phi \end{bmatrix} \tag{2.11}$$

Finally, the requested plane coordinate matrices X''', Y''', Z''' can be obtained by shifting X'', Y'', Z'' by $P(x_0, y_0, z_0)$, as follows:

$$\begin{bmatrix} X''' \\ Y''' \\ Z''' \end{bmatrix} = \begin{bmatrix} X'' \\ Y'' \\ Z'' \end{bmatrix} + P(x_0, y_0, z_0) \tag{2.12}$$

While there are many interpolation methods can be used, we use bi-linear interpolation which is specified by the received Cartesian grid voxels.

Since the "cut" of the 3D data set is random, the intersection of the requested plane and the 3D volume can be polygon with different shape (refer to Figure 1.1).

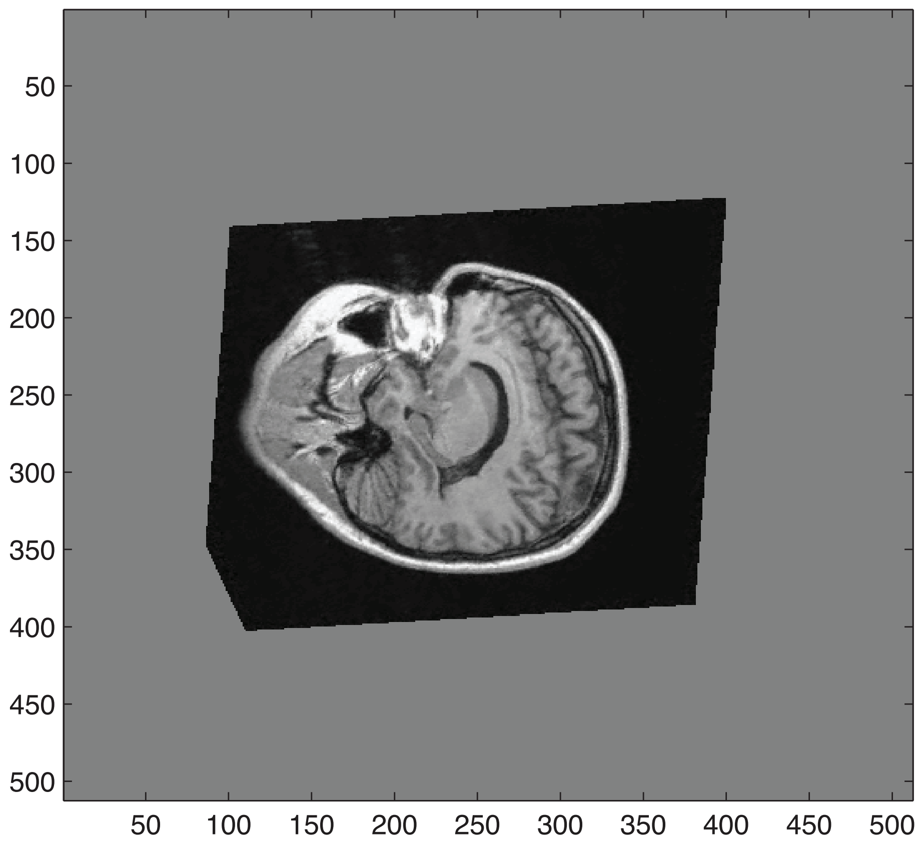


Figure 2.12: Display of the reconstructed requested plane using proposed system. 3D volume size is 256x256x256. Transmission rate is 1.2 bits per pixel, PSNR is 35.28 with quantization scaling value 30.

One experimental result can be seen from Figure 2.12. A 3D compression result is shown in Figure 2.13 and compared to 2D compression for an MRI volume data of a patient's head[7] of size $256 \times 256 \times 256$. Uniform quantization scale value Q ranges from 10 to 25 in increments of 5. The same quantization scale values and JPEG encoding are used for both 3D compression and 2D compression. For 3D compression, the 3D data are divided into $8\times8\times8$ blocks. For 2D compression, each slice (256×256) will be coded independently with block size 8×8 and procedure is repeated for all the slices. Figure 2.13 shows that the 3D compression has a much higher PSNR compared to the 2D compression for the same compression rate. Similar comparison results can also be seen [18] and [17].

2.6 Experiments

2.6.1 2D Case

We now compare our proposed tiling (with rectangular, overlapping tiles) to a standard tiling strategy (with square, non-overlapping tiles). Let T_r and T_s be the average total number of bits to transmit in the rectangular and square tiling modes, respectively, and let $T_e = T_r/T_s$ denote the ratio of required bandwidths for the rectangular and square tiling schemes. Similarly, let S_r and S_s be the total storage for the image using the rectangular and square tiling schemes, respectively, and let $S_o = S_r/S_s$ be the relative storage overhead required by the rectangular tiling scheme. In this Chapter, the rotation center parameters D_x and D_y and the number of rotation angles N vary, while we fix $W = 8$ and $L = 32$.

[7]This volume data was downloaded from http://www9.informatik.uni-erlangen.de/External/vollib/.

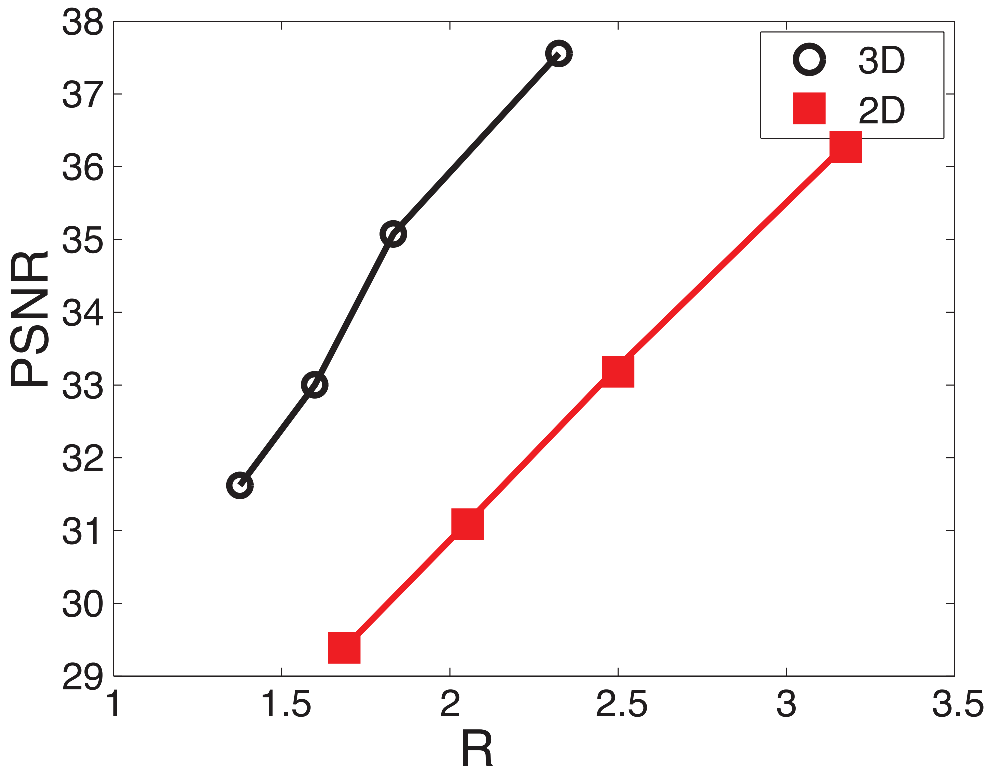


Figure 2.13: Compression Result: PSNR versus R (Rate). Q is **10** $\sim$ **25** with step 5. The circle points are using 3D compression. The square points are using 2D compression.

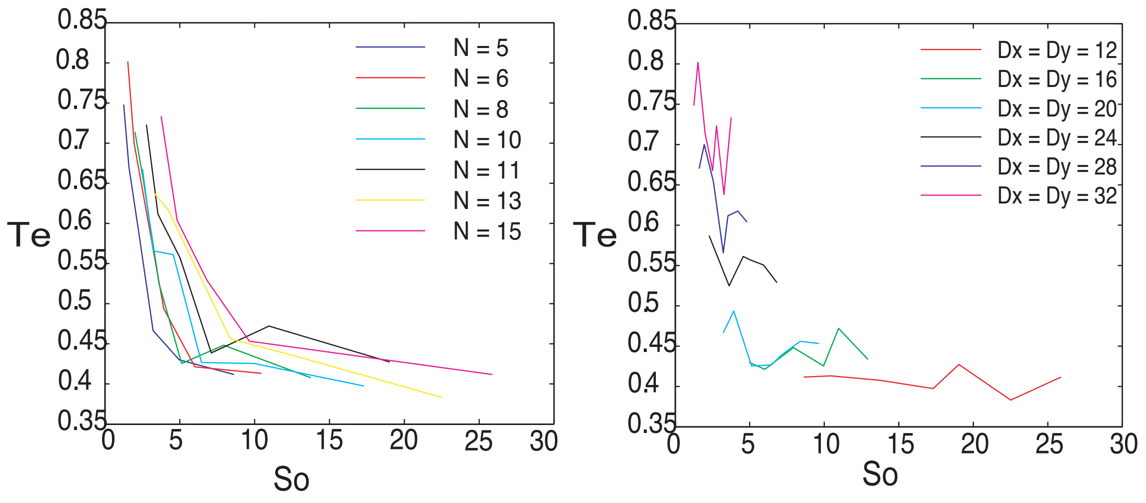


Figure 2.14: T_e versus S_o: left side, for a fixed value of N, D_x and D_y are varied; right side, for a fixed value of D_x, D_y, N is varied. Using JPEG for compression.

Figure 2.14 (left) represents the tradeoff between T_e and S_o as we vary the number of angles N while keeping the center positions of the rectangles fixed (an optimized tiling layout in Chapter 3.2 will not keep the center positions of the rectangles fixed, which will improve the performance dramatically.). The curves are not smooth because N is discrete and because each point is obtained by averaging the results from only 20 random retrieval experiments (in this case we use an MRI image). Figure 2.14 (right) shows the trade-off between T_e and storage S_o while varying the center positions of the rectangles and fixing the number of angles N. These two figures demonstrate that as compared to conventional tiling, the proposed method leads to increases in random access efficiency of 20% - 60% as compared to square tiling, depending on the chosen storage overhead (S_0). Hence, we can choose the parameters to achieve different levels of higher transmission efficiency according to the different storage requirements. For example, choosing $N = 6$, $D_x = D_y = 20$, leads to a 50% reduction in bandwidth at the expense of a 4-fold increase in storage at the server ($T_e = 0.49$ and $S_o = 3.93$).

2.6.2 3D Case

We now compare our proposed 3D tiling (with rectangular, overlapping tiles) to a standard tiling strategy (with cubic, non-overlapping tiles). 80 experiments of random oblique plane retrieval were done. For each request of a random oblique plane, both rectangular and cubic tiling schemes are applied. Figure 2.15 reports the results for our 3D case study, where 3D compression has not been applied yet to the tiles. The results after using 3D compression are provided in Chapter 3. Considering that the actual transmission rate or storage in bits is proportional to the volume of encoded 3D tiles, in this experiment, we compare the number of retrieved voxels for the various methods. The rectangular tile size is fixed to be $W = 32$, $L = 32$ and $H = 8$. The side of a cubic tile is 20. Similarly to the 2D case, T_r and T_s indicate the total numbers of bits transmitted in the rectangular and cubic tiling modes, respectively, and $T_e = T_r/T_s$ denotes the ratio of required bandwidths for the rectangular and cubic tiling schemes. S_r and S_s are the total storage sizes when using the rectangular and cubic tiling, respectively, and $S_o = S_r/S_s$ is the relative storage overhead required by the rectangular tiling scheme. Figure 2.15 shows the trade-off between the ratio of required bandwidth T_e and the storage overhead S_o, while we fix one and vary the other between the center position and the number of angles. The number of rotation angles varies by using the tile angle selection method in Section 2.2.2. The tile center positions vary by changing the rotation center distances (D_x, D_y and D_z in Section 2.2.2). The proposed method presented in this Chapter leads to an increase in random access transmission efficiency of 15% - 55% as compared to cubic tiling, depending on the chosen storage overhead (S_0). This transmission rate can be further reduced by allowing more storage overhead. An optimized tiling method will be proposed in Chapter 3, which will improve the

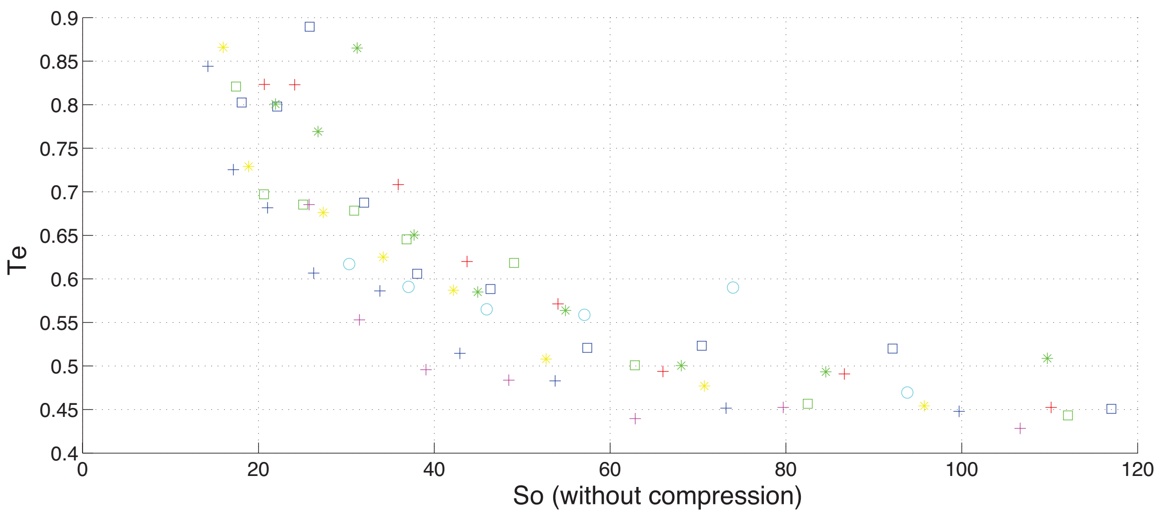


Figure 2.15: 3D case result: T_e versus S_o. T_e denotes the ratio of required bandwidths for the rectangular and cubic tiling schemes. S_o denotes the relative storage overhead required by the rectangular tiling scheme. $T_e = T_r/T_s$ $S_o = S_r/S_s$. Note, the storage here is before 3D compression.

performance dramatically. Intuitively, the proposed method uses an over-complete set (overlapped rectangular tiles) to represent the 3D data set, while the traditional method uses an orthogonal set (non-overlapped cubic tiles) to represent the 3D data set. Therefore, more over-completeness (more storage overhead) tends to lead more transmission efficiency. In addition, Figure 2.15 shows that for the same storage overhead (S_o), different transmission efficiencies (T_e) are achieved by using different parameter settings (D_x, D_y, D_z and the number of the tile rotation angles). Hence, we can choose the parameters to achieve the best transmission efficiencies according to the different storage requirements. Prediction models for both 2D and 3D cases will be introduced in Chapter 3 for the parameter selection. The models will allow users to select best tiling parameters without running any simulation.

2.7 Conclusions

We proposed a new approach for fast random access retrieval of large datasets. For the 2D system, random paths are retrieved from 2D images. For the 3D system, random oblique planes are retrieved from large 3D volume. In our experiments we have demonstrated that by using overlapped tiles with different orientations and allowing some storage overhead on the server's side, for the 2D case, transmission rate can be reduced by 20% - 60% depending on the desired trade-off with storage overhead, as compared to the conventional square tiling scheme. For the 3D case, transmission rate can be reduced by 15% - 55% depending on the desired trade-off with storage overhead, as compared to the conventional cubic tiling scheme. Additional reductions in transmission rate or storage overhead can be achieved by using the optimized tiling scheme in Chapter 3. This system has the potential to considerably speed up the random access procedure, requiring less data storage at the client compared to conventional tiling. We demonstrated two fast rectangle searching algorithm for both 2D and 3D cases. For the 2D searching algorithm, we have proved its optimality. The 2D system has the potential to be used in map applications on small devices, eg. GPS and mobile phone and the 3D system has the potential to be used on different applications in low memory devices.

Chapter 3

Tiling Optimization

We have presented a rotation center based tiling in Chapter 2. We consider a "redundant" tiling system, i.e., where each pixel is available from multiple different tiles. However, in general, a given tiling strategy may lead to different redundancy for different pixels. We propose a simple tool to evaluate different tiling systems, which will be described in detail in Section 3.1.1. For a given tiling strategy we compute the average number of tiles per pixel (i.e., the average redundancy for a pixel), which provides us an estimate for the total storage required. In addition, we also consider the variance in the number of tiles per pixel. Since arbitrary lines can be retrieved from an image, it is reasonable to assume that every pixel is equally likely to be requested. Therefore, it is in general undesirable for this "tiling variance" to be large, since this would mean that some pixels are represented in a significantly more redundant manner than others. In short, good tiling schemes will tend to be such that i) tiles of evenly distributed orientations are available and ii) the variance in the number of tiles per pixel is low.

There are various ways of designing the tiling, e.g., different tile sizes, different orientations, tile centers, etc. In Chapter 2, we have presented a rotation center based design that assumes that a series of "rotation centers" are chosen, in which several tiles of different orientations are centered. The rotation centers are placed at points on a square Cartesian grid patten and have the tile rotation angles uniformly distributed around each rotation center. These rotation angles are the same for each rotation center.

In this Chapter, prediction models for both 2D and 3D cases will be introduced in Section 3.1 for the parameter selection that provides insight as to why different parameter settings produce different transmission efficiencies for a given storage overhead. The models will allow users to select best tiling parameters without running any simulation. With the intuition provided by the prediction models, optimized tiling schemes for both 2D and 3D cases will be presented in Section 3.2.

3.1 Prediction Model

As was shown in Section 2.6.1, by using the rotation center based tiling method transmission rate can be reduced dramatically as compared to the conventional tiling scheme, depending on the desired trade-off with the storage overhead. But the question is whether better results are possible. By using the rotation center based tiling scheme, for a fixed storage overhead, different values of distance between rotation centers and the number of rotation tiles around each rotation center achieve different levels of transmission efficiency. Thus, in designing such redundant storage systems it is important to determine how to select parameter settings in order to achieve the best transmission efficiency for a given storage overhead.

We start by noting that with a redundant tiling strategy each pixel/voxel is available from more than one tile. Intuitively, since each pixel/voxel is equally likely to be selected, it would make sense for all pixels/voxels to be available from a similar number of tiles. Thus our model provides a way to estimate relative transmission efficiency by computing the mean and standard deviation of the number of tiles per pixel/voxel. While we use the rotation center based tiling schemes to illustrate the models, they can be readily applied to other tiling approaches (e.g., to the optimized tiling schemes of Section 3.2). In order to compute the mean (μ) and standard deviation (σ) of the number of tiles per voxel, assume that a tile has a random orientation around a rotation center. The mean and standard deviation of the tile coverage for a whole volume can be computed based on one Voronoi region spanned by the rotation centers, given that regular tilings will be used (i.e., the same tiling structure will be repeated throughout the volume). The Voronoi region associated with rotation center C_i is defined by

$$V(C_i) = \{k : d(k, C_i) \leq d(k, C_j),\ j \neq i, \quad i, j \in I_n\} \tag{3.1}$$

where k here denotes a pixel/voxel at position in 2D/3D space and $d(k, C_i)$ is the distance between pixel/voxel k and rotation center C_i. Denote $M = |V(C_i)|$, the number of pixels/voxels in $V(C_i)$. Then μ and σ can be calculated by using one Voronoi region,

$$\mu = \frac{1}{M}\sum_{k=1}^{M} T_k \qquad\qquad \sigma = \sqrt{\frac{1}{M}\sum_{k=1}^{M}(T_k - \mu)^2}. \tag{3.2}$$

where T_k represents the number of tiles covering pixel/voxel k. Let N be the number of rotation angles we use in 2D case or 3D case, so that the N norm vectors of the tiles are $\tilde{\mathbf{n}} = \{\vec{n}_1, \vec{n}_2, \vec{n}_3, \cdots, \vec{n}_N\}$. Tiles are thus rotated by selecting one of the N angles. In order to compute μ and σ, T_k for every pixel/voxel k in M needs to be calculated. We assume that a tile has a random orientation around a rotation center. The analysis is based on randomly rotated tiles, then T_k can be measured as the average number of tiles covering pixel/voxel k. For both 2D and 3D cases, each parameter will be calculated differently. We will show that T_k only depends on the distance d between pixel/voxel k and all the rotation centers close to k. The details of the 2D case will be described first in Section 3.1.1, followed by the 3D case in Section 3.1.2.

3.1.1 2D Case

The metric will provide a way to relate transmission efficiency to statistics (average and standard deviation) describing how well pixels in the image are covered by the tiles. Let W x L be the size of the rectangular tile and N be the number of the tiles associated with one rotation center. N norm vectors of the tiles are $\tilde{\mathbf{n}} = \{\vec{n}_1, \vec{n}_2, \vec{n}_3, \cdots, \vec{n}_N\}$. Tiles are thus rotated by selecting one of the N angles. We select a tile with norm vector $\tilde{\mathbf{n}}' = \mathbf{R}_\phi \tilde{\mathbf{n}}$, where ϕ is a random variable with uniform distribution $\mathbf{U}(0, \pi)$. $\mathbf{R}_\phi$ is the corresponding rotation matrices. The analysis is based on randomly rotated tiles. Define "coverage probability" as the probability of one tile covering a specific pixel. Figure 3.1 illustrates the possible cases involved in computing the "coverage probability". Intuitively, the coverage probabilities are higher for pixels closer to the rotation center, lower for pixels farther from the rotation center. For example, considering only one rotation center, the pixels in the range I can always be covered and the pixels in the range IV can never be covered regardless of the rotation angle of the tile. From Figure 3.1, notice that the coverage probability of an arbitrary pixel depends only on d: $P(d)$ can be obtained as the ratio between i) the lengh of the intersection of a circle of radius d and a tile with arbitrary orientation and ii) the circumference of the circle.

range I $[d \leq \frac{W}{2}]$

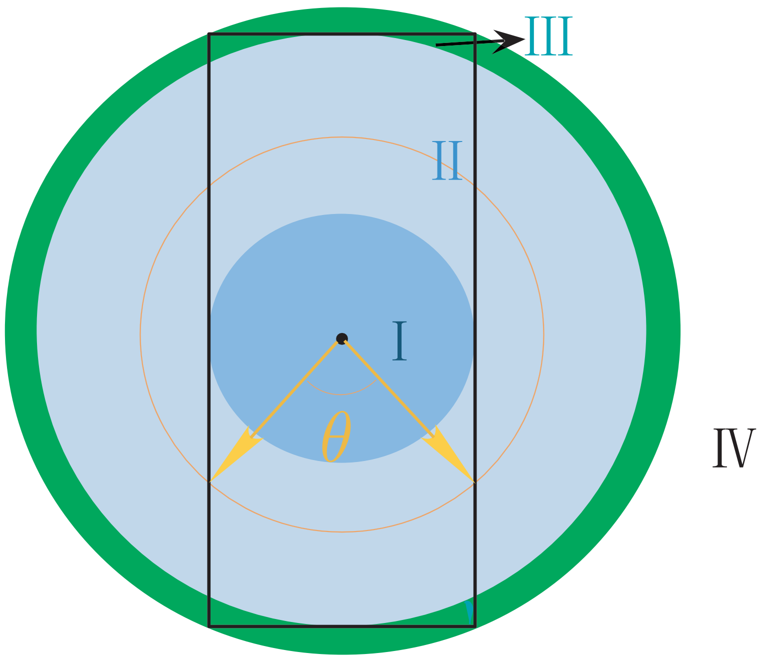


Figure 3.1: Illustration of the metric for calculating the average number of tiles of an arbitrary pixel around one rotation center.

$$p(d) = 1$$

range II $[\ \frac{W}{2} < d \leq \frac{L}{2}\ ]$

$$p(d) = 2 \times \arcsin \frac{W}{2d} \times \frac{1}{\pi}$$

range III $[\frac{L}{2} < d \leq \frac{1}{2} \times \sqrt{L^2 + W^2}]$

$p(d) = 2 \times (\theta_1 - \theta_2 + \theta_3) \times \frac{1}{\pi}$ where, $\theta_1 = \arctan \frac{W}{L}$, $\theta_2 = \arccos \frac{L}{2d}$, $\theta_3 = \theta_1 - \arcsin \frac{L}{2d}$

range IV $[d > \frac{1}{2} \times \sqrt{L^2 + W^2}]$

$$p(d) = 0$$

The four cases illustrated in Figure 3.1, for different ranges of d, lead to different expressions for $P(d)$.

For the rotated tiling scheme, assume we have N tiles around a rotation center, with orientations chosen independently with an identical uniform angle distribution, as discussed. Then, the average number of tiles covering a pixel is $P(d) \times N$. Alternatively, assume there are multiple rotation centers in the neighborhood $\mathcal{N}\{k\}$ of pixel k. Here we consider neighboring rotation centers to be those that are within a distance $\sqrt{(L/2)^2 + (W/2)^2}$ of a given pixel. Then the average number of tiles covering pixel k is $T_k = \sum_i P(d_{ki}) \times N$, where $P(d_{ki})$ denotes the coverage probability for one of the rotation centers i in $\mathcal{N}\{k\}$ and pixel k can be described with 2D coordinate (x, y). d_{ki} is the distance between k and the rotation center i. This coverage probability will depend on the distance between the pixel and each of the neighboring rotation centers. Hence the average (μ) and the standard deviation (σ) of the tile coverage can be calculated (3.2). This only needs to be done for one Voronoi region spanned by the rotation centers, since rotation centers are located uniformly. Figure 3.2 shows the Voronoi regions. Voronoi region associated with the rotation center C_i is defined by Equation 3.1, where $d(k, C_i)$ is the distance between pixel k to the rotation center C_i. M is the number of pixels in $V(C_i)$. Algorithm 3 summarizes how μ and σ are computed.

Algorithm 3 Estimation algorithm of μ and σ for different tiling layout

```
1: Find one Voronoi region $V$ and calculate $M$, the number of pixels in $V$.
2: for Each point $k \in M$ do
3:     Find the rotation centers in the neighborhood $\mathcal{N}\{k\}$ of pixel $k$ (within a
       distance $\sqrt{(L/2)^2 + (W/2)^2}$).
4:     Calculate $T_k = \sum_i P(d_k i) \times N$, where $P(d_k i)$ is the coverage probability for
       one of the rotation centers $i$ in $\mathcal{N}\{k\}$ and pixel $k$.
5: end for
6: Calculate $\mu$ and $\sigma$ according to (3.2).
```

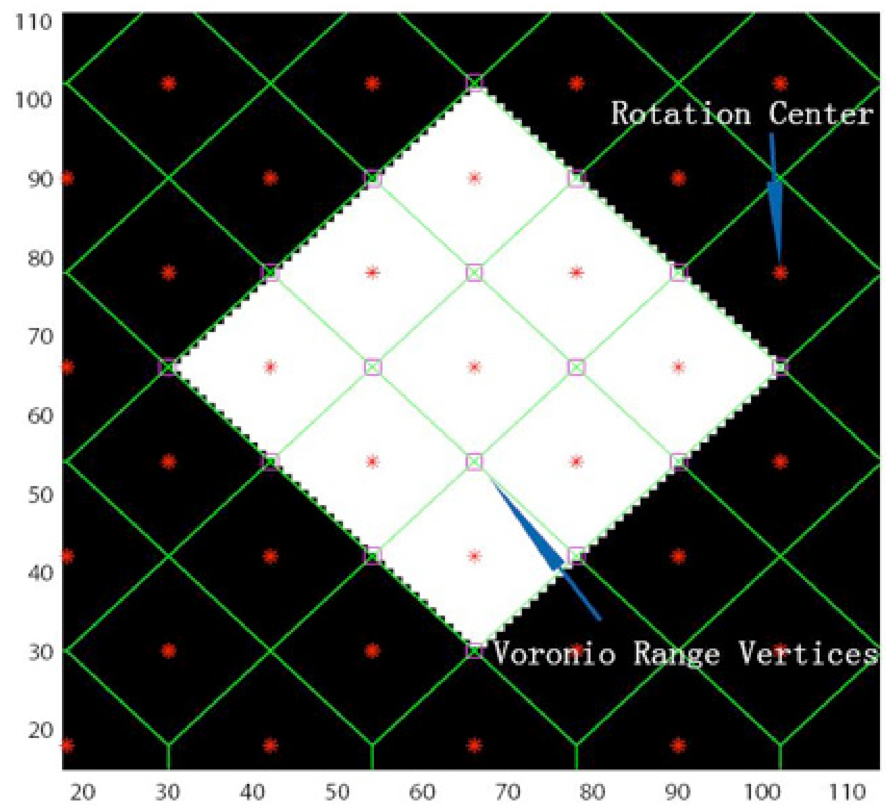


Figure 3.2: 2D Voronoi regions. 9 Voronoi regions are shown in the center part.

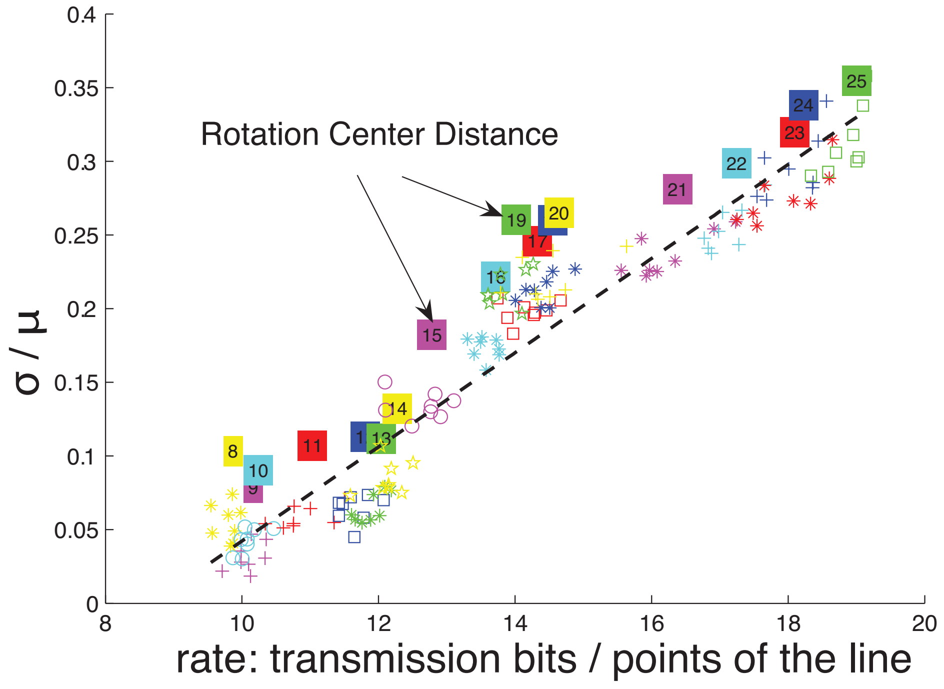


Figure 3.3: Linear regression: σ/μ vs. rate

Using linear regression, Figure 3.3 shows that there is a highly linear relationship between the ratio σ/μ characterizing the tile coverage and the required rate[1]. Figure 3.4 represents the σ/μ as a function of average storage (μ) for different parameter settings, where focusing on the range of m $0-40$. Note that the storage overhead plotted in Figures 3.3 and 3.4 does not take into account compression. From Figure 3.3, we can see that for a fixed rectangular tile size, in order to achieve lower transmission rate, a lower σ/μ is preferred. Hence, for the same storage overhead, the parameter setting that produces the smallest σ/μ should be chosen. Notice from Figure 2.14 that in the rotated center based tiling scheme, for the same average value (storage overhead), decreasing the rotation center distance is better than increasing the rotation angles because decreasing the rotation center distance will

[1]The calculation of the rate is the same as in the previous sections, which use 30 random lines.

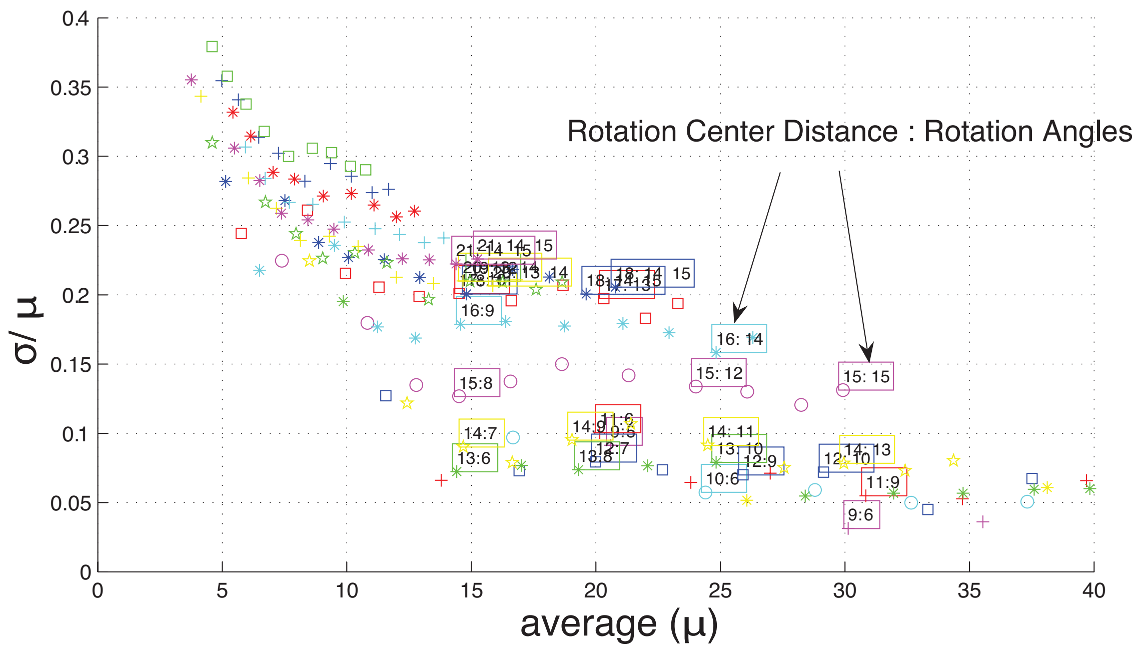


Figure 3.4: Linear regression: σ/μ vs. average storage(m), $\mu < 40$ (without using compression technique.)

lead to smaller σ/μ. In order to find the optimal parameter settings that lead to a small σ/μ for a given target μ from Figure 3.4, a polynomial may be fit to the lower convex hull defined by the points in the figure. Parameters associated with the lower convex hull would be preferred for selection. Using this prediction model with the intuition, one can build other tiling layouts to achieve better performance.

3.1.2 3D Prediction Model

Let $L \times W \times H$ be the size of each tile (with $L = W$ and $L, W > H$) and let N be the number of rotation angles we use, chosen using sphere tesselation (refer to Section 2.2.2), so that the N norm vectors of the tiles are $\tilde{\mathbf{n}} = \{\vec{n}_1, \vec{n}_2, \vec{n}_3, \cdots, \vec{n}_N\}$. Tiles are thus rotated by selecting one of the N angles. In the first tiling scheme

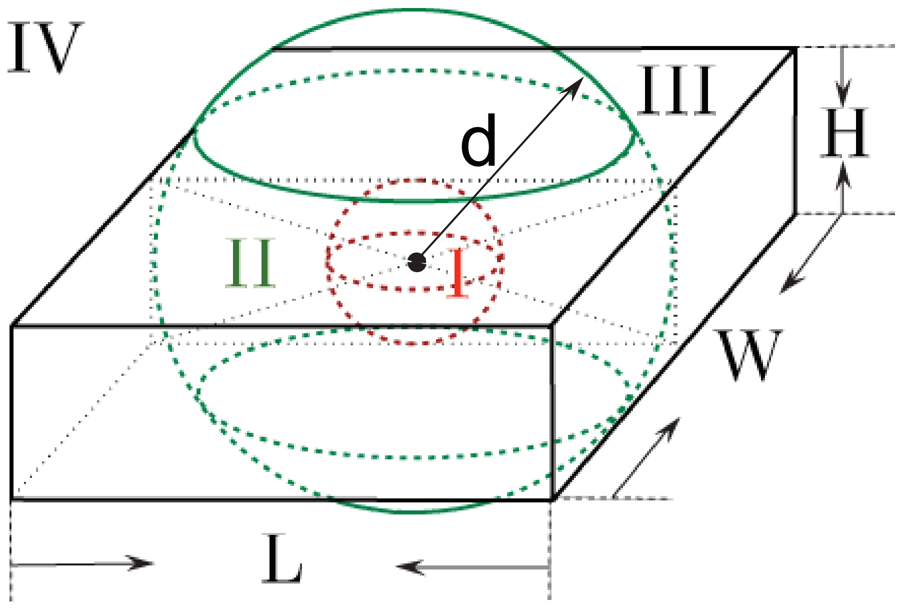


Figure 3.5: Model for calculating the average number of tiles covering an arbitrary voxel around a rotation center.

multiple tiles with different orientations shared the same rotation center, while in our new approach in Section 3.2.2, the tile centers can be different.

In order to compute the mean and standard deviation of the number of tiles per voxel, assume that a tile has a random orientation around a rotation center. More precisely, we select a tile with norm vector $\tilde{\mathbf{n}}' = \mathbf{R}_\phi \mathbf{R}_\varphi \tilde{\mathbf{n}}$, where φ and ϕ are independent random variables with uniform distribution $\mathbf{U}(0, 2\pi)$. $\mathbf{R}_\phi$ and $\mathbf{R}_\varphi$ are the corresponding rotation matrices. Since φ and ϕ are independent random variables with uniform distribution $\mathbf{U}(0, 2\pi)$, the possible norm vectors of the tiles form a unit sphere.

Based on this we derive expressions for the coverage probability, $P(d)$ similar to the 2D case, i.e., the probability that a voxel at location (x, y, z) will be covered by a randomly rotated tile. Let $d = \sqrt{(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2}$ be the distance between this voxel and the tile rotation center at (x_c, y_c, z_c). Figure 3.5 illustrates the possible cases to consider. Intuitively, the coverage probabilities are higher for

Range	Distance(d)	Coverage Probability
I	$d \le \frac{H}{2}$	$P(d) = 1$
II	$\frac{H}{2} < d \le \frac{L}{2}$	$P(d) = 1 - \frac{1}{2}(\cos\theta - \cos(\pi - \theta))$, where $\theta = \arcsin(\frac{H}{2d})$.
III	$\frac{L}{2} < d \le \frac{1}{2} \times \sqrt{L^2 + W^2 + H^2}$	$P(d) \approx 0$
IV	$d > \frac{1}{2} \times \sqrt{L^2 + W^2}$	$P(x, y, z) = 0$

Table 3.1: Coverage probability for any voxel by one tile with random norm vector.

voxels closer to the rotation center. For example, as shown in Figure 3.5, voxels in the range I (inside the ball with radius H) can always be covered and voxels in the range IV can never be covered, regardless of the rotation angle of the tile. From Figure 3.5, notice that the coverage probability of an arbitrary voxel depends only on d: $P(d)$ can be obtained as the ratio between i) the area of the intersection of a sphere of radius d and a tile with arbitrary orientation and ii) the area of the sphere:

$$A = \int_0^{2\pi} \int_\theta^{\pi-\theta} \sin\alpha d\alpha d\varphi, \qquad \text{where } \theta = \arcsin(\frac{H}{2d}) \tag{3.3}$$

$$P(d) = \frac{A}{4\pi}. \tag{3.4}$$

The four cases illustrated in Figure 3.5, for different ranges of d, lead to different expressions for $P(d)$ which are summarized in Table 3.1.

For the rotation center based tiling scheme, assume we have N tiles around a rotation center, with orientations chosen independently with an identical uniform angle distribution, as discussed before. Then, the average number of tiles covering a voxel is $P(d) \times N$. Alternatively, assume there are multiple rotation centers in the neighborhood $\mathcal{N}\{k\}$ of voxel k. Here we consider neighboring rotation centers to be those that are within a distance $\sqrt{(L/2)^2 + (W/2)^2 + (H/2)^2}$ of a given voxel.

Then the average number of tiles covering voxel k is $T_k = \sum_i P(d_{ki}) \times N$, where $P(d_{ki})$ denotes the coverage probability for one of the rotation centers i in $\mathcal{N}\{k\}$ and voxel k can be described with 3D coordinates (x, y, z). d_{ki} is the distance between k and the rotation center i. This coverage probability will depend on the distance between the voxel and each of the neighboring rotation centers.

The mean (μ) and standard deviation (σ) of the tile coverage for a whole volume can be computed based on one Voronoi region spanned by the rotation centers, given that regular tilings will be used (i.e., the same tiling structure will be repeated throughout the volume). The Voronoi region associated with rotation center C_i is defined by (3.1). where k here denotes a voxel at position (x, y, z) in 3D space and $d(k, C_i)$ is the distance between voxel k and rotation center C_i. M is the number of voxels in $V(C_i)$. Then μ and σ can be calculated by using (3.2) using one Voronoi region. Figure 3.6 shows the 3D Voronoi regions. Each numbered shaded area represents one Voronoi region in 3D. Since the rotation centers are located at points on a regular octahedron grid patten (the rotation centers are located at the regular octahedron vertices), the Voronoi regions are all the same polyhedrons. Algorithm 3 summarizes the steps of how μ and σ are computed, which can also be used in the 3D case.

Using linear regression, Figure 3.7 shows that there is a highly linear relationship between the ratio $\sigma^{\frac{2}{3}}/\mu$, which characterizes the tile coverage, and the transmission rate[2]. The reason for using the third root of the variance $\sigma^{\frac{2}{3}}$ is the dimensionality of 3D volume data. As we see here and also from the 2D case, the ratio of variance per dimension and μ has highly linear relationship with the transmission rate. In the future work, it will be interesting to evaluate the relationship in higher dimension. Figure 3.8 represents $\sigma^{\frac{2}{3}}/\mu$ as a function of the average storage (μ) for different

[2]Rate = transmission bits / points in the plane. The rate is averaged over 80 random planes.

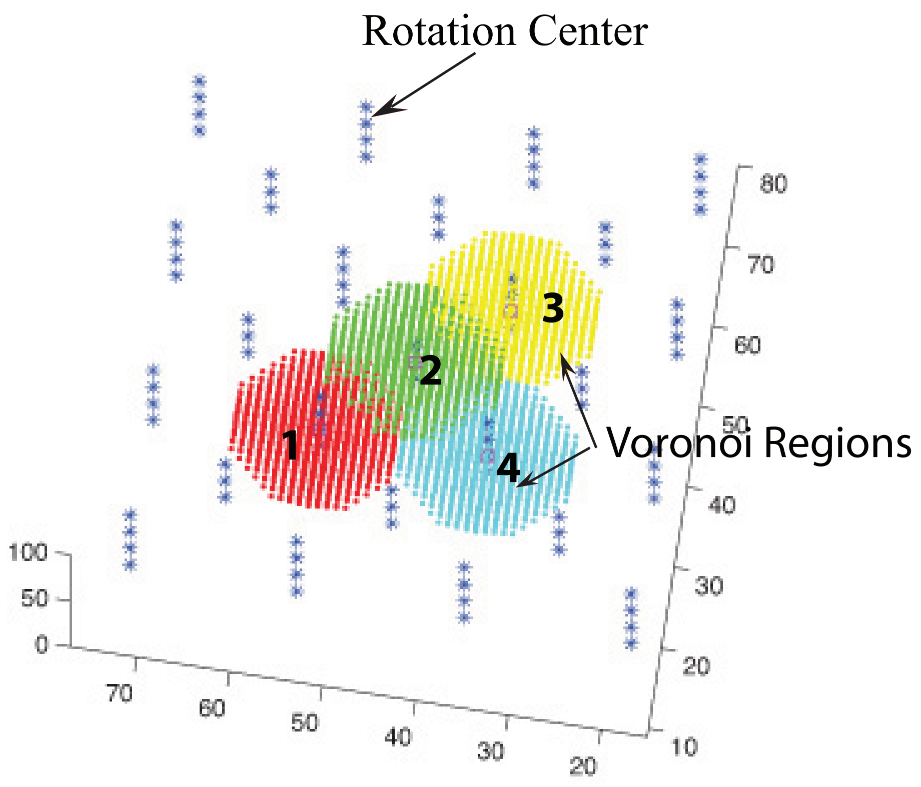


Figure 3.6: 3D Voronoi region: The blue star points are the locations of the rotation centers. Each numbered shaded area represents one Voronoi region. Since the rotation centers are located at points on a regular octahedron grid patten (the rotation centers are located at the regular octahedron vertices), the Voronoi regions are all the same polyhedrons.

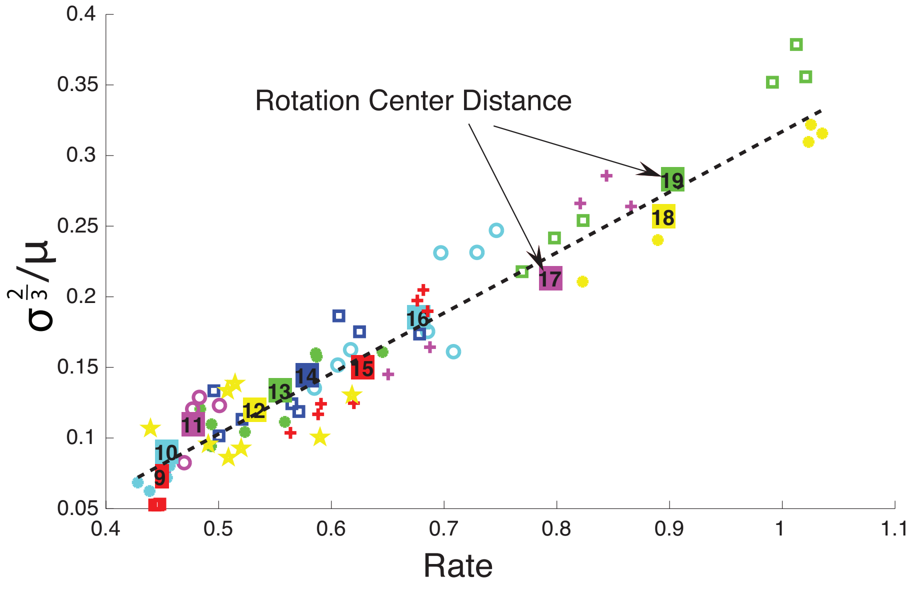


Figure 3.7: Linear regression: $\sigma^{\frac{2}{3}}/\mu$ vs. Rate.

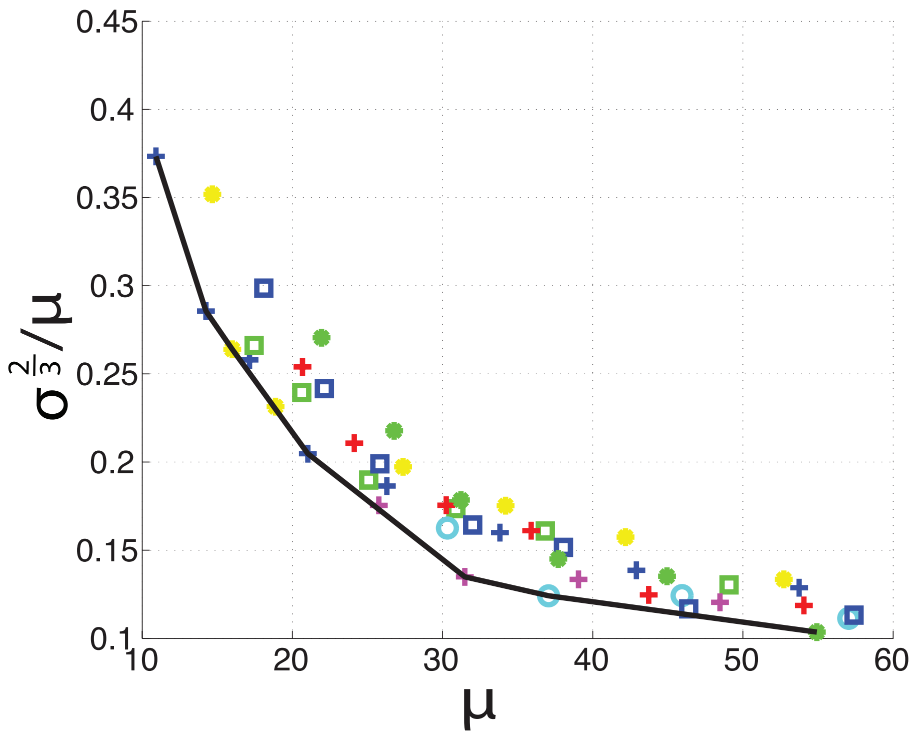


Figure 3.8: $\sigma^{\frac{2}{3}}/\mu$ vs. μ, ($\mu < 60$)

parameter settings, focusing on the range of $0 \leq \mu \leq 60$. Note that the values of μ shown in Figures 3.7 and 3.8 are calculated from (3.1) and (3.2). From Figure 3.7, we observe that in order to achieve a lower transmission rate, a lower $\sigma^{\frac{2}{3}}/\mu$ is preferred. Hence, for the same value μ, the parameter setting which produces the smallest $\sigma^{\frac{2}{3}}/\mu$ should be chosen. Notice that in our previously proposed tiling scheme, for the same value μ, decreasing the rotation center distance is better than increasing the rotation angles (equivalent to increasing the number of tiles per rotation center) because decreasing the rotation center distance will lead to smaller values of $\sigma^{\frac{2}{3}}/\mu$. This can be seen in Figure 3.9. In order to find the optimal parameter settings considering both μ and $\sigma^{\frac{2}{3}}/\mu$ to be small, from Figure 3.8, a polynomial may be fit to the lower portion of the convex hull defined by the points in the figure, the associated parameters of which would be preferred for selection.

3.2 Optimized Rectangular Tiling Scheme

We now use the results of the previous section to motivate our proposed tiling schemes. As shown by in Figures 3.7 and 3.8 for the 3D case, a lower variation in the number of tiles per voxel is desirable. The rotation center based tiling scheme centered multiple tiles with different orientations around each rotation center, leading to high variability in coverage (all tiles covered voxels near the rotation center, while only a few tiles covered those voxels further away). As an alternative, we no longer place multiple tiles around each rotation center. More importantly, we propose a technique to search for tile positions and angles *in order to minimize overlap with previously chosen tiles.* This ensures that tiles are more evenly distributed and the overall variance of tile coverage is reduced.

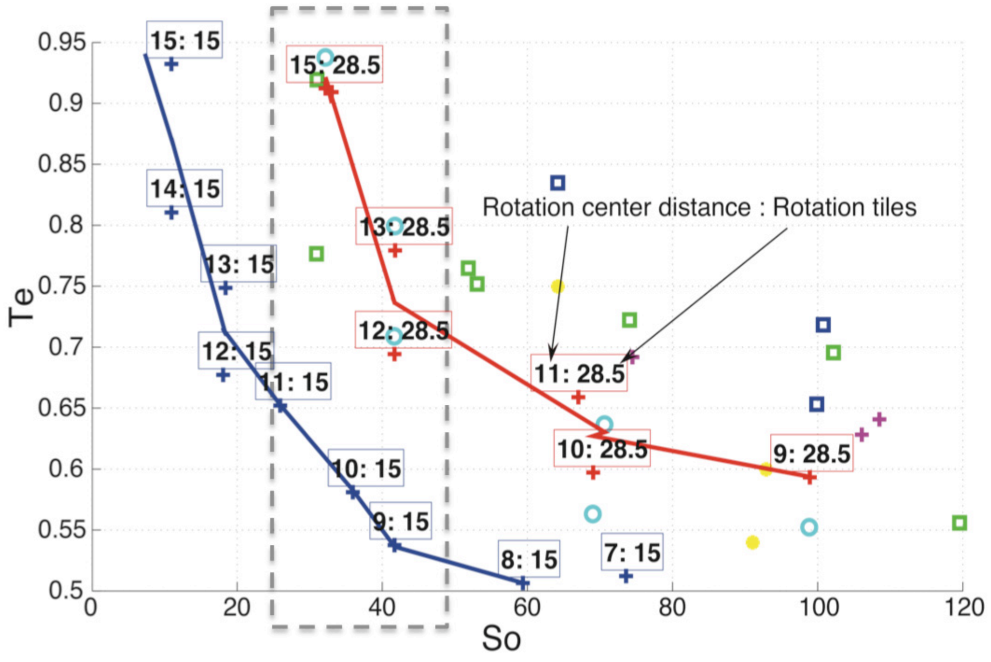


Figure 3.9: 3D case result using first tiling scheme: T_e versus S_o. T_e denotes the ratio of required bandwidths for the rectangular (T_r) and cubic (T_s) tiling schemes. S_o denotes the relative storage overhead required by the rectangular tiling scheme. $T_e = T_r/T_s$ $S_o = S_r/S_s$. Rotation center distance: the distance between the rotation centers. Rotation tiles: the average number of the rotated tiles around each rotation center. Note that the average number of rotation tiles on the red curve is not integer because different numbers of rotated tiles are used at different rotation centers and this is their average.

In our new tiling scheme we define multiple “layers” of tiles, where the tiles for each layer have the same angle and are located in a regular lattice throughout the image in 2D case and the volume in 3D case, as shown in Figure 3.11. Each tile is represented by the tile center location p and the norm vector $\vec{n}_i$ representing the tile angle. For each layer, the tiles could be arranged according to a regular pattern. There are N layers in total, each with different tile angles. Each layer does not have to cover all the pixels/voxels, but each pixel/voxel has to be covered by at least one tile. The new tiling scheme aims to decrease the standard deviation in the number of tiles that cover a typical voxel. In this scheme, starting with first layer tiles with arbitrary locations on a regular grid pattern and an arbitrary angle, the new layer tiles are determined by minimizing the maximum overlap value between the new tiles and the existing tiles. We then record the new layer of tiles in the existing tile list, and proceed to find the next layer of tiles until all the layers (with different angles) are determined. Because we request sets of pixels/voxels, low maximum overlap between tiles means that when a query gets a bad fit from one tile, it is more likely that the query can get a good fit from another tile. Intuitively, the 2D/3D rectangular tiling method uses an over-complete set (overlapped rectangular tiles) to represent the 2D/3D dataset, while the traditional method uses an orthogonal set (non-overlapped square/cubic tiles) to represent the 2D/3D dataset. Minimizing the value of the maximum overlap between the new tile and the existing tiles leads to less correlation within the over-complete set, which then leads to greater transmission efficiency.

We now describe the algorithm we propose to select tile locations and angles for each of the layers. The genearl algorithm will work for both 2D case and the 3D case. Let $B \equiv \{1, 2, \ldots, N\}$ be the set of angles indexes. We let (P_i, I_i) denote the selected tiles for layer i, where $P_k = \{p_{k1}, p_{k2}, \cdots\}$ are the k^{th} layer tiles locations and $I_k \in B$ is the k^{th} layer angle index. Let B^k be the set of angles used up to the k^{th} layer so that $B - B^k$ is the set of angles unused up to k^{th} layer. We let A^k denote the set of tiles that have been assigned to the first k layers. Initially $A^0 = \emptyset$ and $B^0 = \emptyset$, but these sets grow as the algorithm assigns tiles to the layers. Note that each angle index will be assigned to one of the layers, such that at the end of the entire algorithm we will have $B - B^N = \emptyset$. For each layer Algorithm 4 searches for the tile positions/angles that minimize the maximum overlap with tiles in all the previously chosen layers. Algorithm 5 is called in order to break ties, whenever multiple angles and positions lead to the same maximum overlap (detail is in the next paragraph).

Since the tile centers of all layers are on regular grid, it follows that once we know the position of one tile location in a given layer, we know the locations of all the tile centers for that same layer. Because the Voronoi regions are the same within each layer, we only need to determine the location and the angle index of one tile in the center Voronoi region, denoted by V, in order to generate all the tiles for that layer. We let C_k denote the tiles in A_k that are inside or close to the Voronoi region. Let $O(\{p_1, I_1\}, \{p_2, I_2\})$ denote the overlap between two tiles $\{p_1, I_1\}$ and $\{p_2, I_2\}$ and let

$$O_{max}(\{p_1, I_1\}, C^k) = \max_{\{p_m, I_m\} \in C^k} \{O(\{p_1, I_1\}, \{p_m, I_m\})\}$$

be the maximum overlap between tile $\{p_1, I_1\}$ and all the tiles in C^k. We then search for solutions $\{p_k^*,\ I_k^*\}$[3] that minimize the maximum overlap, i.e.,

$$\{p_k^*, I_k^*\} = \arg\min\{O_{max}(\{p_k, I_k\}, C^k)\},$$

[3] * / ** denotes the result before / after the adjustment Algorithm 5, respectively.

where $p_k \in V$, and $I_k \in B - B^{k-1}$. The new tile $\{p^*,\ I^*\}$ minimizes the maximum overlap with previously chosen tiles. For a given tile location p, it is possible that a continuous range of tile angles may be equally optimal due to the fact that tile overlap is measured by the number of discrete grid points that are contained in the overlapping regions. In this case, Algorithm 5 chooses the median angle index. For example, if tiles $\{p,\ 3\}, \{p,\ 4\}, \{p,\ 5\}$, $\{p,\ 6\}, \{p,\ 7\}$ are equivalent in terms of overlap, then tile $\{p,\ 5\}$ will be selected. Similarly, if the tile at location p^* minimizes overlap, then it may be possible for some points $(x, y, z) \in \mathcal{N}\{p^*\}$ to achieve the same maximum overlap. Let $\mathbf{S}$ denote the set containing all the points $(x, y, z) \in \mathcal{N}\{p^*\}$ (including p^*) that achieve the minimum max-overlap value. Algorithm 5 chooses the central point of $\mathbf{S}$ as the selected tile location p^{**} with its associated angle index I^{**} to be the new tile in the Voronoi region for the next tile layer.

Algorithm 4 Optimized Rectangle Tiling Algorithm

```
1: Starting with first layer tiles $\{P_i, I_i\}$ with arbitrary locations on a regular grid
   pattern and with tile angle index $I_i = 1$. $A^0 = \emptyset$, $B^0 = \emptyset$.
2: for k = 1 to N do
3:    Sort $A_k$ according to the distance from the tile center location to the center
      of the data (2D image/3D volume).
4:    Find $C_k$.
5:    for $\forall\ p_k \in |V|$ and $\forall\ I_k \in B - B^{k-1}$ do
6:       $\{p_k^*,\ I_k^*\} = \arg\min O_{max}(\{p_k, I_k\}, C_k)$
7:    end for
8:    The parameter pairs achieving the minimum value of the maximum overlap
      may not be unique. Algorithm 5 is used to select the proper one $\{p_k^{**},\ I_k^{**}\}$.
9:    Using the new tile $\{p_k^{**},\ I_k^{**}\}$, generate all tiles $\{P_k, I_k\}$ for layer $k$.
10:   $A^k = A^{k-1} \cup \{P_k, I_k\}$. $B^k = B^{k-1} \cup I_k$.
11: end for
```

Algorithm 5 Adjustment Algorithm

```
   for $\forall\ p_k \in |\mathcal{N}\{p_k^*\}|$ and $\forall\ I_k \in B - B^{k-1}$ do
       $\mathbf{S} = \{\{p^*, I^*\}\ |\{p^*, I^*\} = \arg\min O_{max}(\{p_k, I_k\}, C_k)\}$
3: end for
   Find the central point $p^{**}$ of $\mathbf{S}$.
   if Continuous angles lead the same minimum max-overlap value then
6:     The median angle index will be chosen as $I^{**}$.
   end if
   Return $\{p^{**},\ I^{**}\}$.
```

3.2.1 2D Case

The new tiling scheme aims to decrease the standard deviation in the number of tiles that cover a typical pixel. In this scheme, starting with first layer tiles with arbitrary locations with a square grid patten and an arbitrary angle, the new layer tiles are determined by minimizing the maximum overlap value between the new tiles and the existing tiles. The tiles on different layers are located with the same square grid patten but with different shift. We then record the new layer of tiles in the existing tile list, and proceed to find the next layer of tiles until all the layers (with different angles) are determined. Figure 3.10 shows that with the same storage overhead, the optimized tiling scheme leads much better transmission efficiency comparing to the rotation center based tiling scheme. Figure 3.10 is calculated in terms of the average number of tiles. Since the tile centers for each layer are uniformly and regularly spread out, all the Voronoi region are the same as shown in Figure 3.6. So we only search for one tile in the one Voronoi region to generate all the tiles for the new layer. More detailed analysis and results can be seen next in the 3D case.

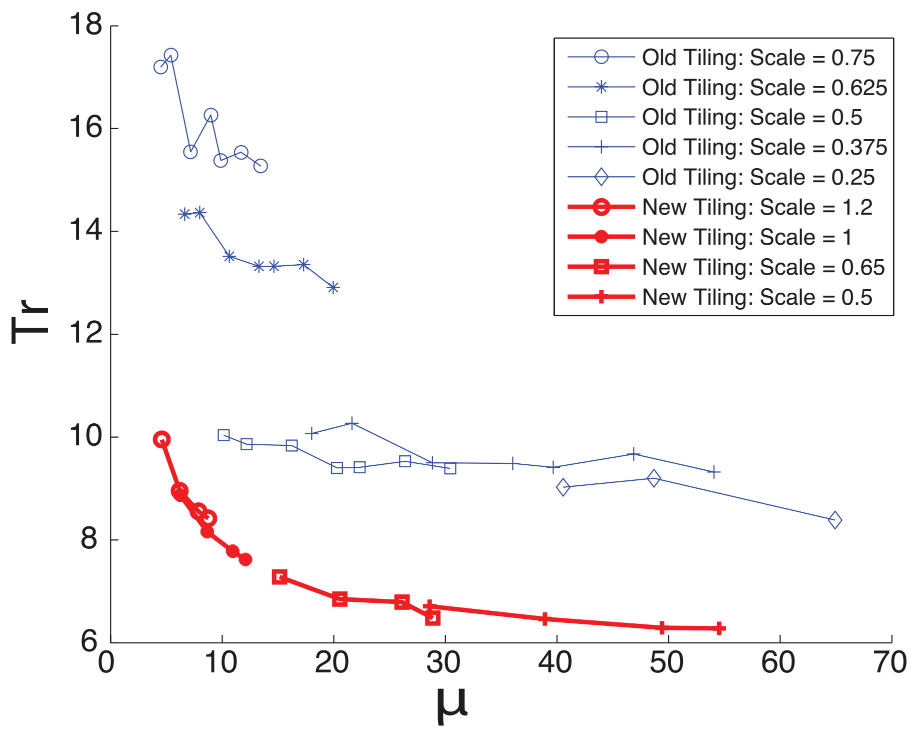


Figure 3.10: Here, tile center distance = 32 × Scale. Fix the tile center distance, the number of rotation angles vary. T_r is the average number of tiles needed to be transmitted. μ is the average number of tiles coverage per pixel.

3.2.2 3D Case

In the 3D case optimized tiling scheme we define multiple "layers" of tiles, where the tiles for each layer have the same angle and are located uniformly throughout the volume, as shown in Figure 3.11. Each tile is represented by the tile center location p and the norm vector $\vec{n}_i$ representing the tile angle. For each layer, the tiles could be arranged according to a cuboctahedral, octahedral, icosahedral or tetrahedral pattern; here we locate the tiles (corresponding to a set of tile centers $P = \{p_1, p_2, \cdots\}$) at points on a regular octahedron grid patten (the tile centers are located at the vertices of an octahedron). There are N layers in total, each with different tile angles. The N norm vectors (one for each layer) are obtained by using the equal sphere tessellation method [5] and are represented with a common origin

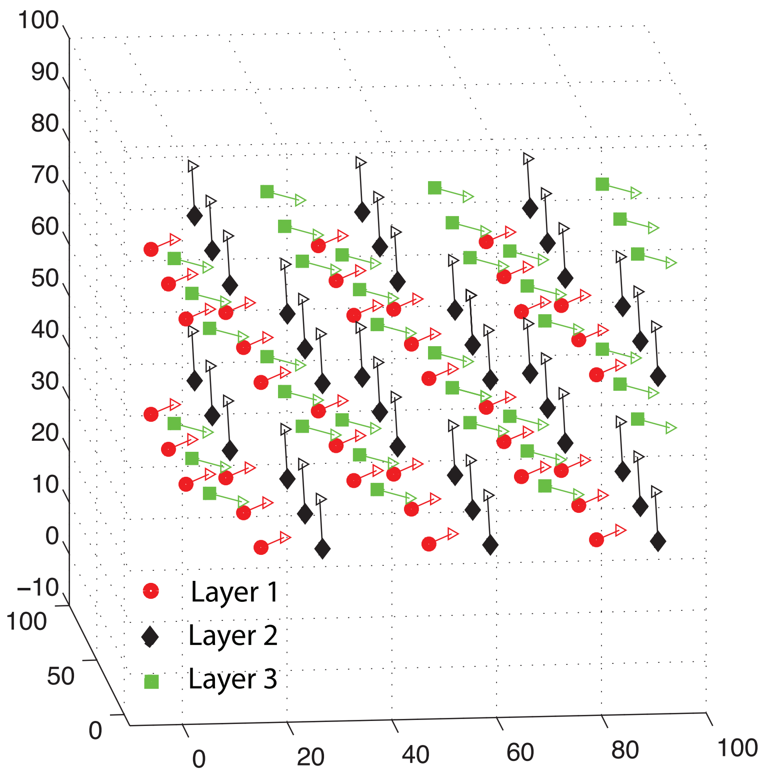


Figure 3.11: New 3D rectangular tiling scheme. The tile centers for each layer are shown. 3 layers of tiles are in this figure. The arrows show the norm vector $\vec{n}_i$ representing the tile angle.

and with their end points uniformly spread out on a 3D sphere. When two vectors with exactly opposite directions are part of the tessellation, only one of them is used as the norm vector in Section 2.2.2.

Figure 3.6 shows the 3D Voronoi regions, with each color representing a separate region. Since the Voronoi regions are the same within each layer, we only need to determine the location and the angle index of one tile in the center Voronoi region In order to speed up the search, we partition the Voronoi region into several 3D subregions shown in Figure 3.12. Each color represents one subregion. At the center point of each subregion, the maximum overlap value is calculated. Then, we pick the subregion with the minimum value to be used for the next level partition. The partition procedure stops once the selected subregion is very small (e.g., less than

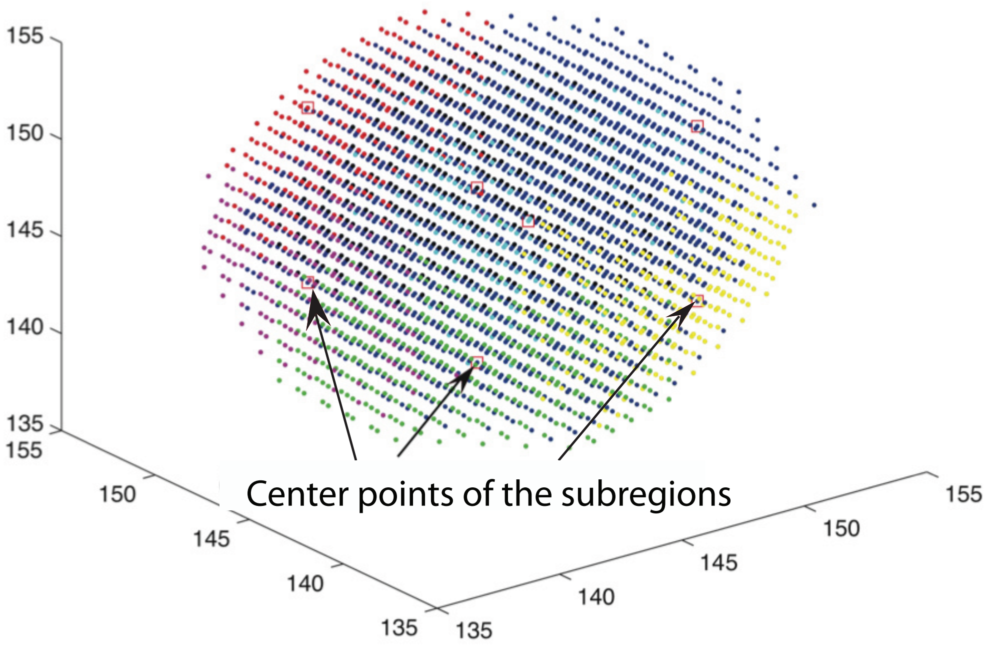


Figure 3.12: 3D Partition for speeding up the searching in Algorithm 4. Each color represents each subregion.

70 voxels within the final selected subregion in our experiments). Algorithm 4 lines 5-7 will then be employed for the final selected subregion, rather than using V. In our experiments, this procedure speeds up the search time dramatically. Since the search is operated on the Voronoi region, the complexity only depends on M, the size of the Voronoi region and it does not increase when the data size increases.

3.2.3 3D Experimental Results

The performances of the optimized 3D rectangular tiling scheme and the rotation center based tiling scheme of Section 2.2.2 are compared in Figure 3.13. The two tiling schemes are also compared in Figure 3.14 by using the model proposed in Section 3.1.2, which shows that the model is also useful for designing new tiling schemes. For all the experimental results in this section, MRI volume data for

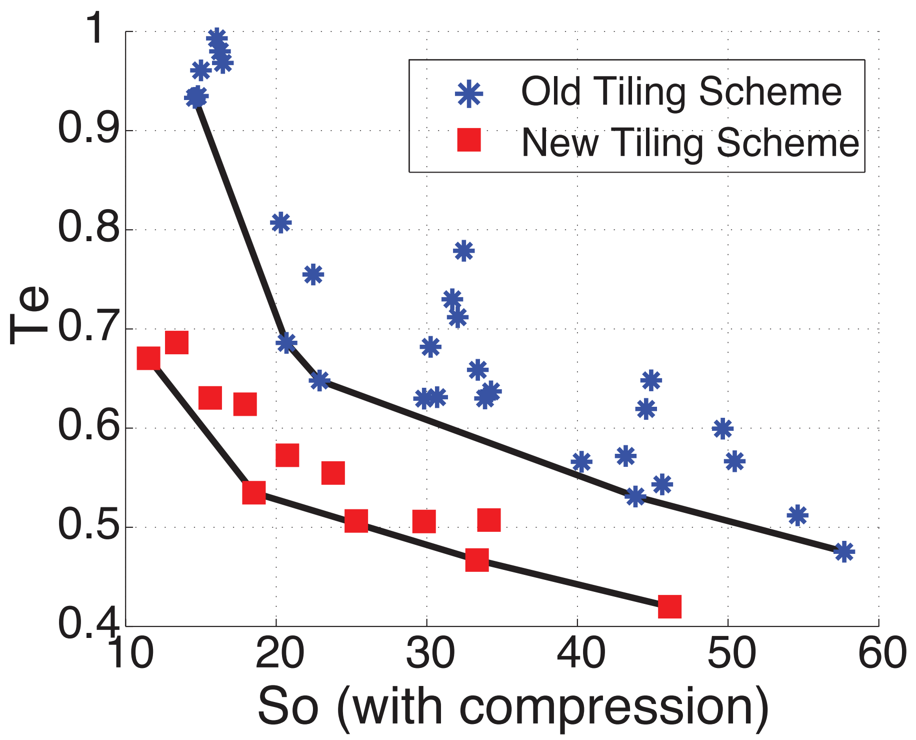


Figure 3.13: 3D case result: T_e versus S_o. T_e denotes the ratio of required bandwidths for the rectangular and cubic tiling schemes. S_o denotes the relative storage overhead required by the rectangular tiling scheme. $T_e = T_r/T_s$ $S_o = S_r/S_s$.

a patient's head[4] of size $256 \times 256 \times 256$ is used. Because we are measuring the transmission ratio between using our proposed schemes and using the traditional square/cubic cases, similar results are expected by using different datasets. After mapping the Cartesian grid points to neighboring points in the rotated rectangular tiles, 3D compression is applied independently to each tile.

Performance is shown compared to a standard tiling strategy (with cubic, non-overlapping tiles). The rectangular tile size was fixed at $W = 32$, $L = 32$ and $H = 8$. The side of a cubic tile was 16. Figure 3.13 shows the trade-off between the ratio of required bandwidth T_e and the storage overhead S_o while we fix one

[4]This volume data was downloaded from http://www9.informatik.uni-erlangen.de/External/vollib/.

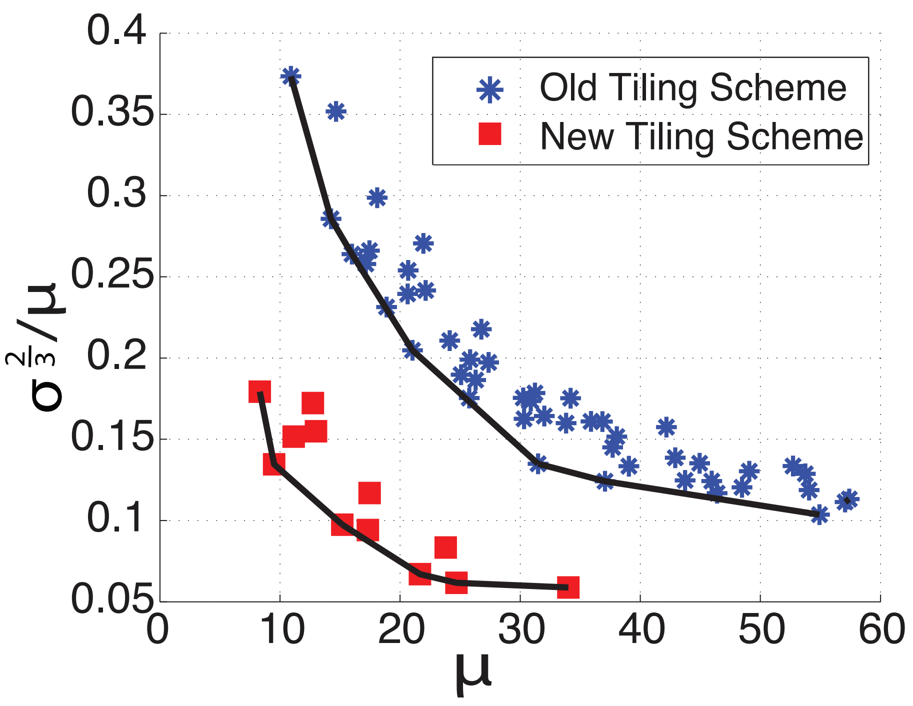


Figure 3.14: $\sigma^{\frac{2}{3}}/\mu$ vs. μ. μ and σ are the average and the standard deviation of the tile coverage.

and vary the other by changing the distance between tile centers and the number of different tile angles N. N varies by using the tile angle selection method in Section 2.2.2. As shown in Figure 3.13, the proposed new tiling scheme leads to an additional increase in random access transmission efficiency of 10% - 30% when compared to the first proposed tiling scheme in Section 2.2.2, depending on the chosen storage overhead (S_0). In addition, Figure 3.13 shows that for the same storage overhead (S_o), different transmission efficiencies (T_e) are achieved by using different parameter settings (the distance between the tile centers and the number of tile rotation angles). By using the model in Section 3.1.2, the parameter setting to achieve the best transmission efficiency (given a fixed storage overhead) can be chosen without experimental tests, shown in Figure 3.14. Smaller variance will lead to greater transmission efficiency. Intuitively, this is because by decreasing the coverage variance across all voxels it follows that all voxels will tend to be covered by a similar number of tiles.

3.3 Conclusions

In this Chapter two prediction models were presented for both 2D and 3D cases. The prediction model provide a way to relate transmission efficiency to pixel/voxel coverage statistics (mean value and standard deviation) for the fast random retrieval system we have proposed in Chapter 2. These models enable designers to select parameter settings that achieve the best transmission efficiency without performing experimental tests and also provide us with insight on designing better tiling schemes. We have thus proposed two optimized tiling schemes for both 2D and 3D cases which achieve an additional 10% - 30% reduction in the average transmission rate compared to the rotation center based rectangular tiling scheme in Chapter 2, We have demonstrated that in exchange for increased server-side storage, significant reductions in the average transmission rate can be achieved. Furthermore, compared to the conventional tiling scheme the client side benefits from random data access and relaxed storage requirements. The run-time memory space can also be reduced accordingly.

Chapter 4

Mapping for Rotated Tile Encoding and Analysis

In this chapter we present a non-interpolated symmetric mapping algorithm, which maps each pixel/voxel in the original image/volume to a rotated Cartesian grid point. We will show that this approach outperforms tile representation methods based on interpolation and non-symmetric mapping. We first evaluate in detail the mapping algorithm for rotated tile encoding in Section 4.1. Furthermore, a metric in Section 4.2 is proposed for automatically checking the mapping symmetry. In Section 4.3, a tool will be presented to analyze and quantify the performance and demonstrate the benefits of our proposed re-mapping algorithm. It will show that in general the more symmetric the mapping is, the better RD performance can be achieved. Our analysis, based on spectral graph theory, could be used for measuring the performance of different mapping algorithms on a grid of any dimension. In Section 4.4, we will experiment in the 2D and the 3D cases and will show that the non-interpolated symmetric mapping approach, which maps each pixel/voxel in the original image to a rotated Cartesian grid point, outperforms tile representation methods based on interpolation and non-symmetric mapping. The experimental results are consistent with the analysis results using the tools presented in Section 4.3. In particular, the lack of interpolation means that complexity is significantly lower and the running time is 20 to 30 times lower. The mapping algorithm, the metric and the analysis tool can be applied for any tile size and with any dimension. Finally Section 4.5 concludes this chapter.

4.1 Mapping Algorithm for Rotated Tile Encoding

Consider a rotated 2D rectangle of length L and width W. Refering to Figure 4.1, CP represents the Cartesian points (CP_I and CP_O represent the Cartesian points inside and outside of the rectangle respectively.), RP represents the points on the rectanglular grid and RC is the center point of the rectangle, around which the rectangle is rotated.

Figure 4.1 shows a example of a rotated 2D tile such as those used in our system. All tiles are encoded using standard transform coding techniques (e.g., the discrete cosine transform, DCT). Transforms operate on pixels organized in a regular grid. But, as seen in Figure 4.1, the points on the rotated rectangular grid (RP) do not coincide with the original Cartesian grid points (CP_O, CP_I). Thus, it is necessary to decide how to assign the intensity values on Cartesian grid in this new rotated grid (RP), this is what we call "mapping algorithm". A straightforward approach to represent the data would be to interpolate the values on the rotated rectangular grid before compression (e.g., using all pixels surrounding a given rotated grid position), but any interpolation will result on smooth, and thus loss of some high frequency information. Additionally, interpolation is computationally expensive.

Thus, instead we propose to simply re-map the original values on the Cartesian grid into the rotated rectangular grid before compression. That is, we select one of the neighboring pixel values and assign it to the rotated grid points. For example in Figure 4.1, for each point on the rotated rectangular grid (RP), there is a Cartesian

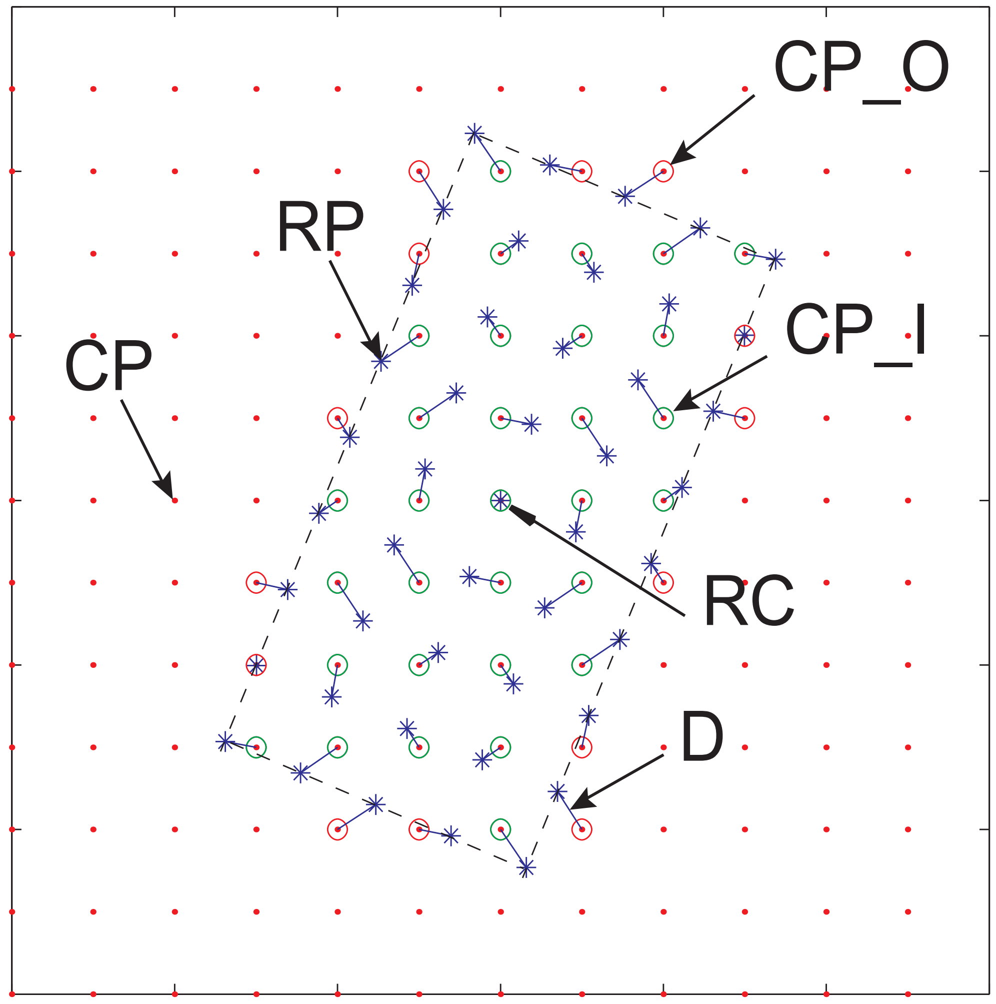


Figure 4.1: 2D Mapping: RC (rotation center), CP (cartesian points), CP_O (cartesian points outside the rectangle), CP_I (cartesian points inside the rectangle), RP (point on rectangle grid), D (mapping points distance).

point to map to and the two mapping points are connected in Figure 4.1 in order to show the mapping relationship. This "mapping" relationship is shown by the blue lines in the Figure. The distance between the mapping pair is denoted as D. Certain points, such as the rotation center (RC), are in the same positions in both grids, while in other cases, the pixel in a given location in the rotated grid has been copied from a neighboring location in the Cartesian grid. Note that the pixels are copied unchanged, i.e., no interpolation is performed.

Our proposed mapping can be used for high dimensional datasets. The algorithm is presented, then 2D and 3D simulations will be provided to evaluate performance. While many alternative mappings are possible, we are interested in methods that minimize the mapping distance (D in Figure 4.1), so that pixels are displaced by a minimal amount from their original positions for encoding. Clearly this is desirable as these displacements "distort" the frequency content of the blocks prior to encoding. Our proposed symmetric mapping algorithm seeks to i) map all Cartesian points inside the rectangular area, ii) minimize the maximum pointwise mapping distances, D, and iii) be symmetric about the rotation center. Starting from the rotation center, we first map all the Cartesian points inside the rectangular area (CP_I) to the rotated grid points (RP) in a 1-to-1 mapping moving from the center of the tile outward. This ensures that all the Cartesian points inside the rectangle will be mapped. We then map the non-mapped rotated grid points to the Cartesian points outside of the rectangular area (CP_O) in a 1-to-1 mapping moving outwards relative to the center. Refer to Algorithm 6 for the details.

In Figure 4.1, certain points, such as the rotation center, are in the same positions in both grids, while in other cases the pixel in a given location in the rotated grid has been copied from a neighboring location in the Cartesian grid. Note that the pixels are copied unchanged, i.e., no interpolation is performed. The blue lines

Algorithm 6 Mapping Algorithm

```
1: Find the Cartesian points inside the rotated rectangle volume. Calculate the
   distances from the Cartesian points (from the step above) to the rotation center.
   Sort the distances with increasing order, calling the sorted list $A$.
2: Calculate the distances from the rotated rectangle grid points to the rotation
   center. Sort the distances with increasing order, calling the sorted list $B$.
3: while $i \leq$ length($A$) and $k \leq$ length($B$) do
4:   while $A(i)$ has been mapped do
5:     $i = i + 1$
6:   end while
7:   while $B(k)$ has been mapped do
8:     $k = k + 1$
9:   end while
10:  if $A(i) \leq B(k)$ then
11:    Find the rotated grid point closest to $A(i)$ from the unmapped points in
       $B$.
12:    Record the mapping points and label the points as “mapped” in $A$ and $B$.
13:    $i = i + 1$
14:  else
15:    Find the unmapped Cartesian point closest to $B(k)$.
16:    Record the mapping points and label $B(k)$ as “mapped”.
17:    Also label the Cartesian point as “mapped”.
18:    The Cartesian point may or may not be in $A$.
19:    $k = k + 1$
20:  end if
21: end while
```

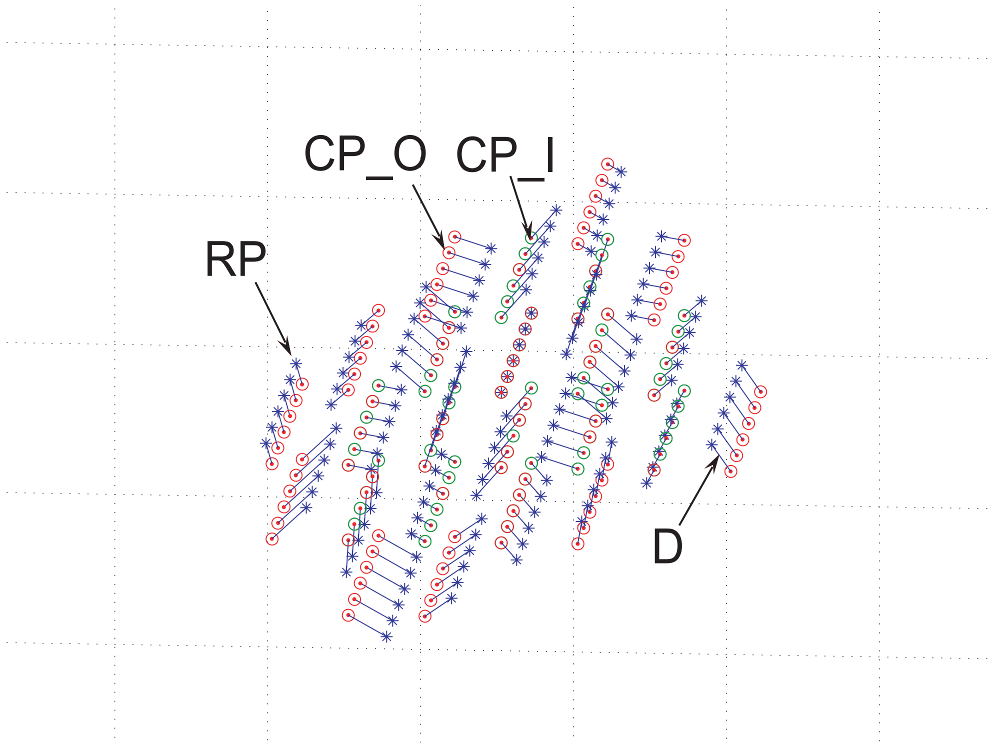


Figure 4.2: 3D Mapping: CP_O (cartesian points outside the rectangle), CP_I (cartesian points inside the rectangle), RP (point on rectangle grid), D (mapping points distance).

show how pixels are remapped. There are many ways to achieving a non-symmetric mapping. The non-symmetric mappings we use in our work are "minimum error mapping", which seek to reduce the mapping distance D and happen to be non-symmetric about the rotation center. In the next sections, we will show that our proposed symmetric mapping has better performance even if its average mapping distance D maybe higher. Algorithm 6 ensures the mapping is symmetric. Figure 4.2 shows a mapping example in 3D case. The blue lines show how voxels are remapped.

Figures 4.3 and 4.4 show a 3D rotated tile and its mappings using symmetric and non-symmetric mapping, respectively. There are many ways to achieving a non-symmetric mapping. Here we use "minimum error mapping" to generate the non-symmetric mappings. Figures 4.3(a) and 4.4(a) show a example of the 3D rotated rectangular tile and the mapping relationships. Figures 4.3(b) and 4.4(b) display the mapping vectors, which are defined as the vectors $\vec{V}$ from the Cartesian points to the mapped rotated grid points. The length of each vector $\vec{V}$ is the mapping distance D analogous to that shown in Figure 4.1. In this context, by "symmetry" we mean that if there is a mapping vector $\vec{V}$ then there must be a mapping vector $-\vec{V}$ in the mapping vector plot[1]. We can check that the right-hand plot in Figure 4.3 has a symmetric mapping while that of the right-hand plot of Figure 4.4 has a non-symmetric mapping. The tile size in the figures is $3 \times 3 \times 3$. When the tile size increases or we move into higher dimensions, it will be impossible to tell the mapping symmetry from the mapping vector plot. Therefore, in Section 4.2, a metric for checking the mapping symmetry will be described.

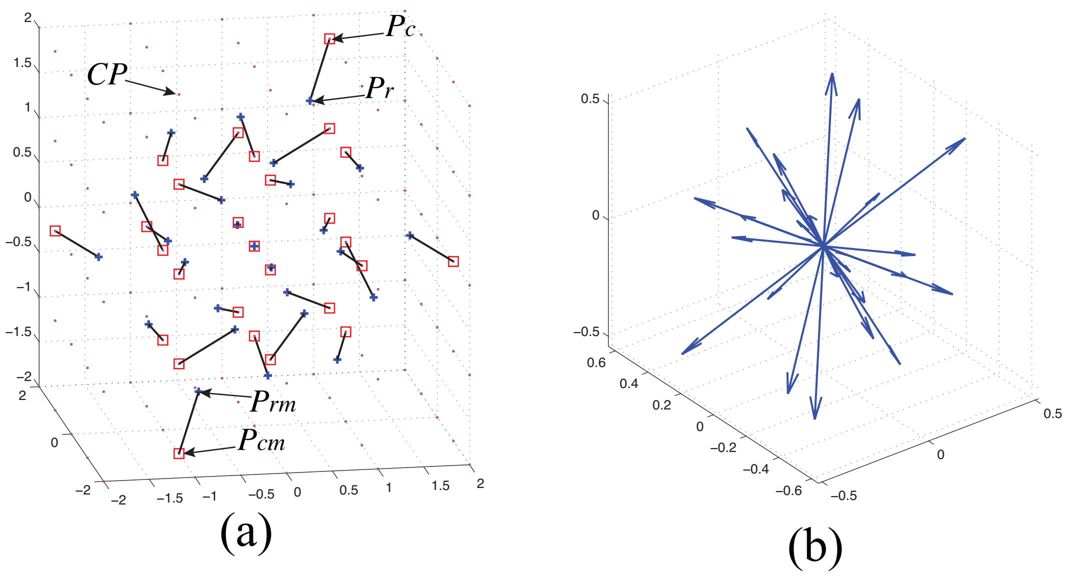
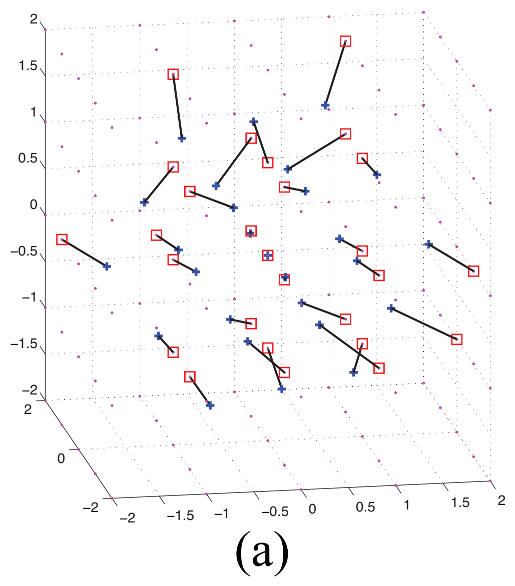


Figure 4.3: (a): 3D rotated tile symmetric mapping illustration. Size $3 \times 3 \times 3$. CP (cartesian points). Other labels are denoted in Section 4.2. (b): mapping vectors (symmetric).

The main advantage of this approach is that it lowers the encoding and decoding complexity, but at the cost of introducing some distortion in the compression

[1]We only consider sides of the rectangle that have odd length. For sides with even length, we do not take into account the last column points which are far from the middle column.



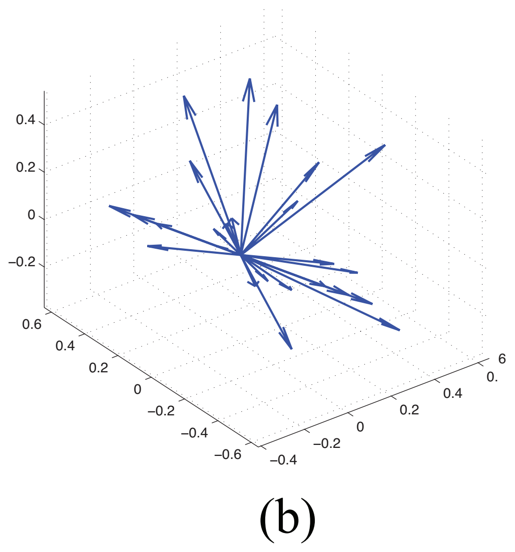


Figure 4.4: (a): 3D rotated tile non-symmetric mapping illustration. Size $3 \times 3 \times 3$. (b): mapping vectors (non-symmetric)

process (i.e., the pixel values that are aligned for encoding are not aligned in the original representation). In our 2D and 3D case study, compression techniques in Sections 2.5.1 and 2.5.2 are applied and the experimental results in Section 4.4 demonstrate greatly improved RD performance. Note that tiles having the same rotation angles around their (different) rotation centers have the same mapping table relative to the Cartesian grid points with just a shift between the rotation centers. Thus, we only need to store a small number of tables (one per angle) to specify the mapping, which is convenient during tiling, searching, and reconstructing tiles onto the Cartesian grid at the client side.

4.2 Metric to Check the Symmetry of Mapping Algorithms

We now introduce a metric to quantify the degree of symmetry. This is introduced in the 3D case but can easily be used in 2D and any dimension cases. Assume the coordinate of the rotation center is $(0, 0, 0)$. Let $P_r(x, y, z)$ denote a point in the rotated rectangular grid and let $P_c(x, y, z)$ denote the associated point in the Cartesian grid that it is mapped to. Then the mapping vector is defined as:

$$\vec{V}(x, y, z) = P_r(x, y, z) - P_c(x, y, z).$$

Let $P_{rm}(x, y, z) = P_r(-x, -y, -z)$ denote the mirror image point of $P_r(x, y, z)$, where P_{rm} and P_r are 180 degrees from each other and let $P_{cm}(x, y, z)$ denote the Cartesian grid point corresponding to $P_{rm}(x, y, z)$. Then

$$\vec{V}_m(x, y, z) = P_{rm}(x, y, z) - P_{cm}(x, y, z).$$

Hence, if the mapping is symmetric,

$$P_{rm}(x, y, z) = P_r(-x, -y, -z)$$

$$\Longrightarrow P_{cm}(x, y, z) = P_c(-x, -y, -z)$$

is ture. This is equivalent to saying that if the mapping is symmetric,

$$\vec{V}_m(x, y, z) = \vec{V}(-x, -y, -z)$$

The total number of vectors that have this property can be counted, and the larger the number of vectors having this property, the more symmetric the mapping is. Algorithm 7 shows the details of the metric computation. Using this metric, the symmetry of the mapping results can be checked automatically regardless of the tile size or the tile dimension. Additionally, this metric can also be used for detecting how many voxels are mapped symmetrically and how many are not, i.e., what percentage of the voxels are not symmetrically mapped.

Algorithm 7 Metric for checking mapping symmetry

```
1: for Each point $P_{ri} \in \{P_{r1}, P_{r2}, \cdots\}$ do
2:     Find their corresponding mirror point $P_{mri}$. [1]
3: end for
4: for each pair $(P_{ri}, P_{mri})$ do
5:     Check and see if their mapping Cartesian points are 180 degree to each other.
6: end for
```

4.3 A Spectral Graph Theory Approach for Data Re-mapping

In this section, we will present tools to both analyze and quantify the performance and benefits of our proposed re-mapping algorithm. We will start in Section 4.3.1, which defines the graphs for the regular grid and the re-mapping grid. In Section 4.3.2, an analysis using spectral transforms on graphs will be shown. The analysis results will be verified by the experiments in Section 4.4.

4.3.1 A Graph Representation of Remapping Problem

Transforms (e.g., DCT) represent data on a regular grid. When we consider a rotated tile we select a subset of pixels from regular grid. Then the question is how

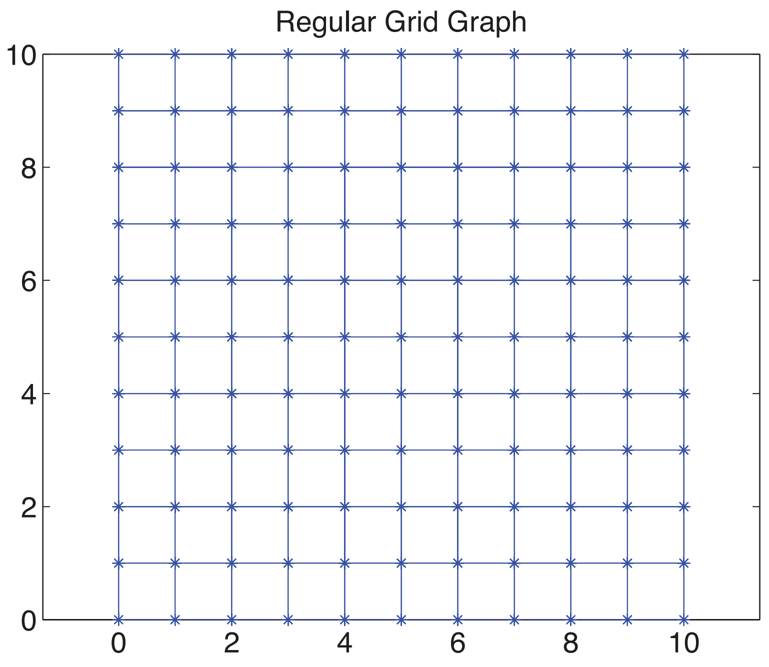


Figure 4.5: Regular grid graph. Size: 11×11

to apply a regular grid transform to this rotated set of pixels. Our solution is to "remap" the pixels in the rotated tile to a regular grid. This is illustrated in Figure 4.6. The key problem is that when placing these pixels in this new regular grid a geometric distortion is incurred. We start by providing a more formal definition of the problem. Then, in Section 4.3.2, a tool will be introduced to evaluate different mapping algorithms quantify the geometrical distortion.

Let $G = (V, E)$ be a graph, which represents a grid. Vertex set V contains N nodes indexed by $n \in \{1, 2, \ldots, N\}$. The set of edges E represents the connectivity of grid pixels. In Figure 4.5 for example, except for the pixels on the boundaries, each pixel is connected with its four neighboring pixels (up, down, left and right). Figures 4.6 and 4.7 display rotated regular grids (blue stars) superimposed on top of their corresponding original Cartesian grids (red dots) as well as the mapping relationship (matching pairs) between pixels in the two (blue vectors). When the

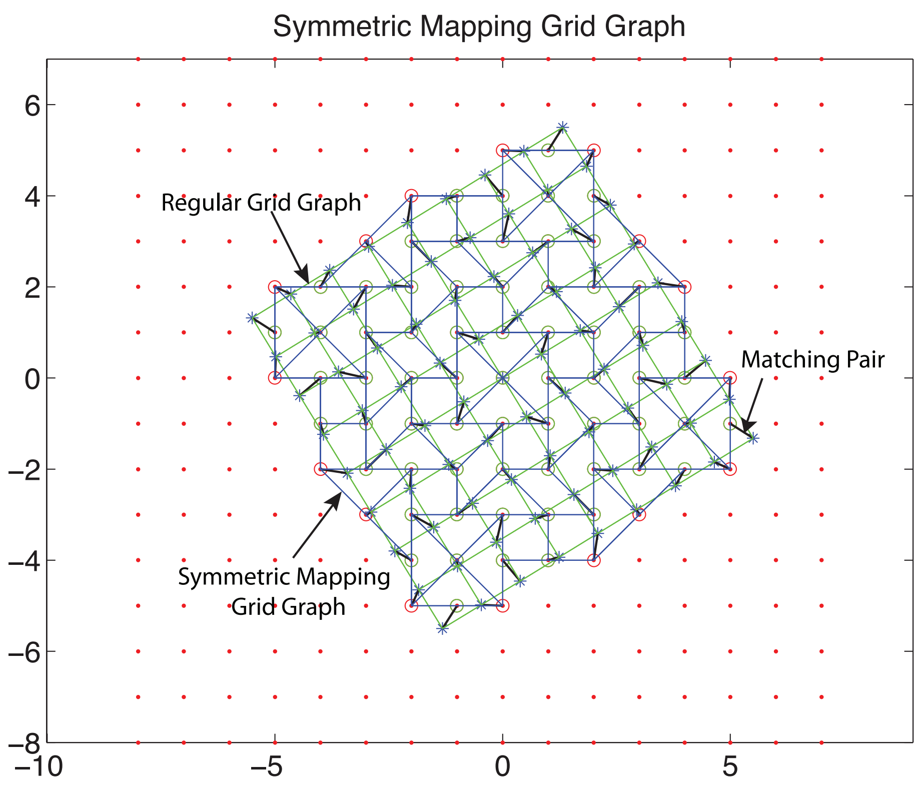


Figure 4.6: Mapping of the Cartesian grid pixels onto the rotated grid using the symmetric mapping algorithm. Size: 9×9

symmetric mapping algorithm is used, connectivity between pixels in the original Cartesian grid is preserved between the pixels they are matched to, as seen in Figure 4.6. This leads to a regular pattern to the blue vectors showing the mapping relationships. In contrast to this is the irregular pattern seen in Figure 4.7, where we see the results of the non-symmetric mapping algorithm.

Let A_G denote the adjacency matrix of the grid graph, with entries $a_{i,j}$ given by

$$a_{i,j} = \begin{cases} A_G(i,j) = \frac{1}{w(i,j)} & \text{if } (i,j) \in E \\ 0 & \text{otherwise,} \end{cases} \tag{4.1}$$

where $w(i,j)$ is the Euclidean distance between nodes i and j. $a_{i,j}$ can be seen as the relationship between the two nodes, which is smaller when the distance between

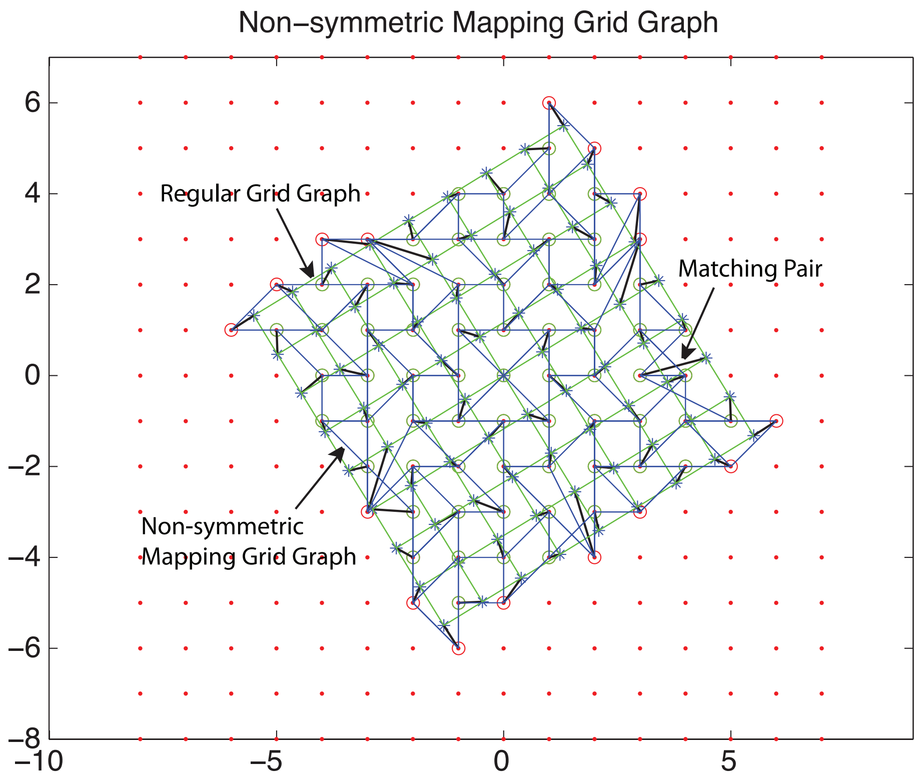


Figure 4.7: Mapping the Cartesian grid pixels onto the rotated grid using non-symmetric mapping algorithm. Size: 9×9

nodes i and j is larger and vice versa. The degree matrix of a weighted graph G will be denoted D_G with diagonal elements

$$D_G(i,i) = \sum_j A_G(i,j). \tag{4.2}$$

The Laplacian matrix of a weighted graph G will be denoted by L_G and is defined as

$$L_G = D_G - A_G. \tag{4.3}$$

Since L_G for undirected graphs is symmetric and positive semidefinite, the eigenvectors of L_G form an orthonormal basis in $\mathbb{R}^N$. Let $\{\lambda_i, \nu_i\}_{i=1}^N$ be the eigenvalues and eigenvectors of L_G arranged in non-decreasing order with respect to the eigenvalues. Similarly to L_G, we let L_{RG}, L_{SG}, L_{NSG} and $\{\lambda_{ri}, \nu_{ri}\}_{i=1}^{n}$, $\{\lambda_{si}, \nu_{si}\}_{i=1}^{N}$, $\{\lambda_{nsi}, \nu_{nsi}\}_{i=1}^{N}$ denote the Laplacian matrices and the eigenvalues and vectors for the regular grid graph, symmetric mapping graph, and the non-symmetric mapping graph, respectively. We also denote $< v_1, v_2 >$ as inner-product between vectors v_1 and v_2.

4.3.2 Analysis and Results

As mentioned earlier, the transform operates on a regular grid, producing transform coefficients that are encoded. Compression is achieved when the data is smooth (or sparse) in the bases of the transform. Note, however, that the geometrical distortion introduced by remapping may mean that even though the original data was smooth (in a traditional 2D or 3D separable transform), the information in the block obtained after remapping may not be as smooth and, thus, overall coding efficiency may suffer. We next propose a way that will allow us to compare mapping algorithms and quantify the difference, e.g., Figures 4.6 and 4.7, and predict which approach will achieve better performance.

L_{RG} is symmetric and positive semidefinite, the eigenvectors of L_{RG} form an orthonormal basis in $\mathbb{R}^N$ and the eigenvectors can be used to characterize various properties of the graph. The transform is based on the regular graph L_{RG}, but the mapping graph grids are not regular any more, which can be seen as geometric distortion with respect to the regular graph. In order to measure the geometric distortion, here we measure the difference of the eigenvectors between L_{RG} and L_{SG}, L_{NSG}.

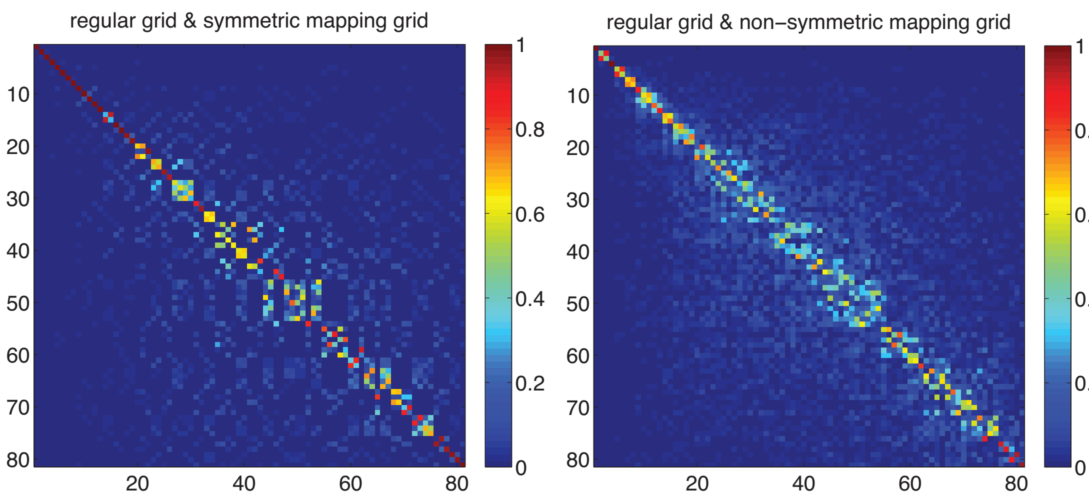
Given that both the symmetric grid graph and the non-symmetric grid graph share the same topology, but different weights, we calculate the inner product between all respective eigenvectors (i.e., regular grid graph vs. symmetric grid graph and regular grid graph vs. non-symmetric grid graph). We let R_S denote the correlation matrix between the eigenvectors of L_{RG} and L_{SG} and let R_{NS} denote the correlation matrix between the eigenvectors of L_{RG} and L_{NSG}. The entries of the correlation matrix R_S are given by

$$R_S\{i,j\} =< \nu_{ri}, \nu_{sj} > \qquad i,j \in \{1,2,\ldots,N\}, \tag{4.4}$$

and for the correlation matrix R_{NS} they are given by

$$R_{NS}\{i,j\} =< \nu_{ri}, \nu_{nsj} > \qquad i,j \in \{1,2,\ldots,N\}. \tag{4.5}$$

Figure 4.8 shows the absolute values of the correlation matrices R_S and R_{NS}. Figure 4.8 (a) indicates that the energy of the correlation matrix R_S is more concentrated along the diagonal terms, while (b) indicates that the energy of the correlation matrix R_{NS} is more spread out along the diagonal terms. Assume that a smooth signal is such that it can be described using a small number of eigenvectors of the original graph Laplacian. Then, the intuition is that the effect of remapping is to "disperse" the energy, since eigenvectors in the new graph have non-zero correlation with all eigenvectors corresponding to the original graph. Thus, it would be desirable for this dispersion to be minimal, and hence for the off-diagonal energy to be as small as possible. From this observation, we can conclude that the spectrum of the weighted grid graph does not change as much when using the symmetric mapping algorithm, as compared to the non-symmetric mapping case.



(a) R_S: Absolute value of the Correlation matrices for the symmetric grid graph.

(b) R_{NS}: Absolute value of the Correlation matrices for the non-symmetric grid graph.

Figure 4.8: Absolute values of the correlation matrices R_S and R_{NS}.

We notice from Figure 4.8 that the largest correlation terms are not always exactly on the diagonal of the matrices. For each row i of the correlation matrix R (R can be R_S or R_{NS}), the maximum correlation term is found and denoted $R_{max}(i)$, i.e.

$$R_{max}(i) = \arg \max_{j \in \{1,2,\ldots,N\}} \{R(i,j)\}. \tag{4.6}$$

The modified diagonal terms of the correlation matrix are thus the set $\{R_{max}(i)\}_{i=1}^{N}$, the energy of which can be calculated as

$$E_{diag} = \sum_{i=1}^{N} R_{max}(i)^2. \tag{4.7}$$

Denoting the energy of the modified off-diagonal terms as $E_{\text{off-diag}}$, we have

$$E_{\text{off-diag}} = N - E_{diag}, \tag{4.8}$$

where the total energy of the correlation matrix is N, since the energy of each row is 1. Hence the percentage of the total energy attributed to the modified off-diagonal terms of the correlation matrix is $\frac{E_{\text{off-diag}}}{N}$. The off-diagonal energy resulting from using either symmetric or non-symmetric mapping is shown in Figure 4.9. In this figure, different sizes of the grid graphs are used, which are 9×9, 11×11, 13×13, 15×15, 17×17, and 19×19. For all grid graph sizes considered, there is consistently more off-diagonal energy using the non-symmetric mapping algorithm than using the symmetric mapping algorithm.

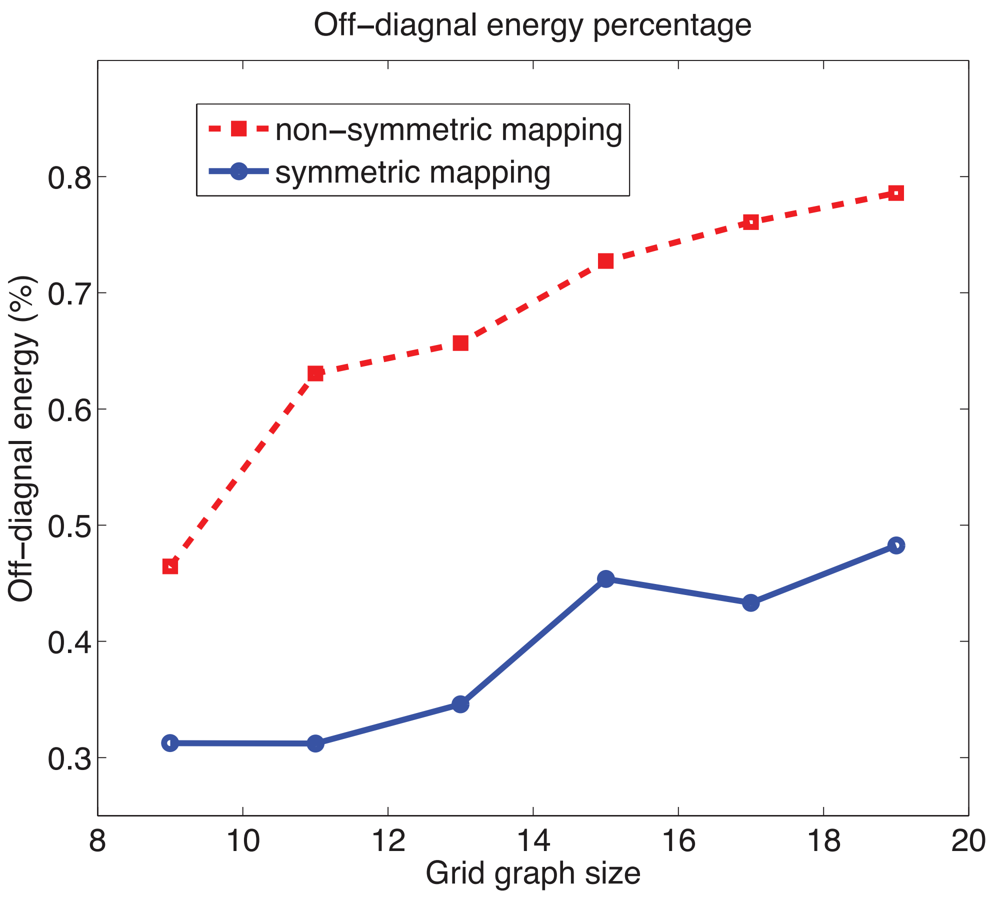


Figure 4.9: Percentage of the off-diagonal energy

4.4 Experimental Results

In this section we evaluate our proposed symmetric mapping algorithm by comparing the performance against the usage of non-symmetric mapping and interpolation methods. With different mapping order (not moving outwards relative to the center), two different non-symmetric mappings are generated (refer to Section 4.1).

4.4.1 2D Case

From Figure 4.1, we can see that our mapping is symmetric. In general, the distance between paired points in the mapping is less than $\sqrt{2}$. The worst case for the mapping is for rotation angles of $\pi/4$ and $3\pi/4$. A rate-distortion (RD) comparison

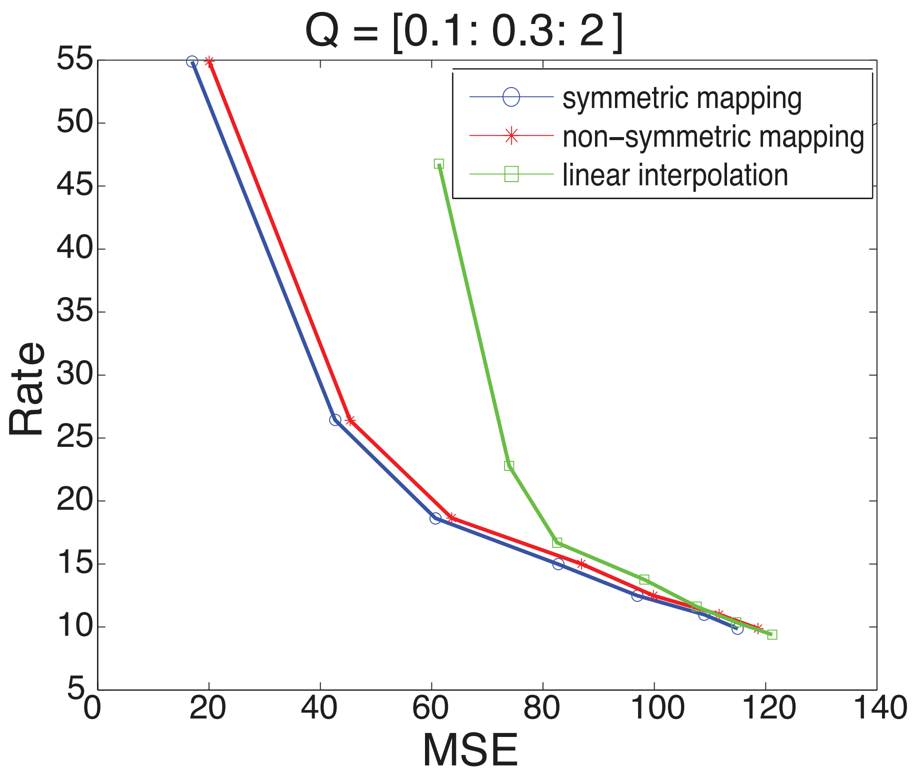


Figure 4.10: Rate distortion for two mapping algorithms and the interpolation method. Using "Lena" as the test image. $N = 8$, $D_x = D_y = 20$

of mapping techniques is shown in Figure 4.10 using the Lena image. We compare our proposed symmetric mapping algorithm and a non-symmetric approach. While their average mapping distances are comparable, the symmetric approach leads to better RD performance. For the interpolation method, we use cubic interpolation to calculate the values of the tiles on the rectangle grid. 30 random lines are used to generate the curves.

4.4.2 3D Case

In this study, 3D MRI volume data has been tiled using the tiling method in 3.2.2, the tiles's voxels then being been mapped via either the symmetric or non-symmetric mapping algorithms onto the Cartesian grid, or interpolated using the linear interpolation method. The 3D interpolation method we used in the experiments is tesselation-based linear interpolation.

The rate distortion curves in Figure 4.11 and Figure 4.12 are generated using the quantization step values from 10 to 80, with increment 10. The mean squared error is calculated to measure the distortion and the rate is measured in terms of the average bits per voxel of the transmitted tiles for all random oblique planes. In Figure 4.11, we interpolate (linearly) the values on the 2D oblique plane grid using the neighborhood Cartesian voxel values. We measure the distortion between the 2D oblique planes, with and without passing through our system. From Figure 4.11, we can see non-interpolated mapping algorithms lead much to better performance than interpolation method. This is because in the non-interpolated case the only loss is due to compression, while in the interpolated case smoothing is performed, leading to loss of high frequency information. A situation analogous to that observed in compression of Bayer filter images [13] arises: interpolation

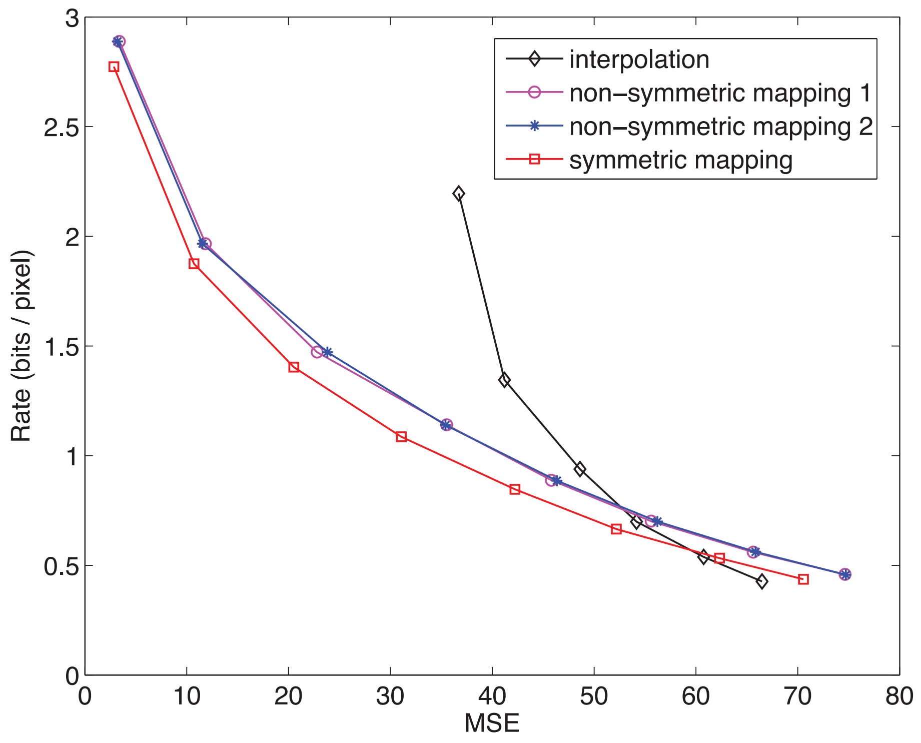


Figure 4.11: RD curves by using symmetric mapping/non-symmetric algorithms and using interpolation method

leads to smoother images, but also removes information from the original data so that at high rates re-mapping leads to better performance than interpolation. The experimental results in Figure 4.12 shows that different non-symmetric mapping algorithms lead to different RD performance in terms of coding, and the symmetric mapping algorithm has better performance than the non-symmetric mapping algorithms. The distortion is measured by using the Cartesian voxel values, which are used for reconstructing the 2D oblique plane from a 3D volume dataset and the distortion only comes from the compression process. For the non-symmetric mapping here, we do select remapping to the nearest neighbor at each point, so that the overall remapping distortion incurred is not very high and the main difference is the lack of symmetry. The experiment uses the same 30 random oblique planes for all different cases.

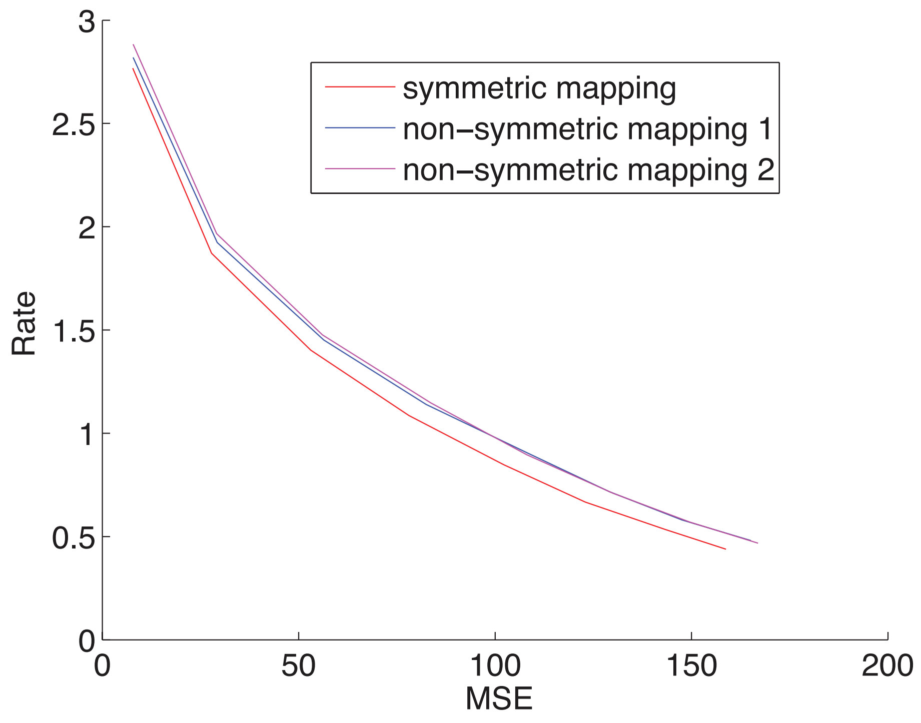


Figure 4.12: Symmetric and non-symmetric mapping algorithms' RD curves.

4.5 Conclusion

In this chapter we have presented a non-interpolated symmetric mapping algorithm which maps each original pixel/voxel in the original image/volume data to a rotated Cartesian grid point. We have shown that this approach outperforms tile representation methods based on interpolation and non-symmetric mapping in both 2D and 3D case, a result that generalizes to higher dimensions. In particular, the lack of interpolation also means that the run-time complexity is significantly lower and is approximately 30 times less for the whole system in our experiments. Moreover, remapping without interpolation has been shown to lead to overall better RD performance and the more symmetric the mapping is, the better the RD performance that can be achieved. Additionally, we have proposed a metric for automatically checking mapping symmetry and measuring the percentage of non-symmetric mapped points produced by the non-symmetric mapping algorithms. Furthermore, we have presented a tool to both analyze and quantify the performance and benefits of our proposed re-mapping algorithm. Across a range of grid graph sizes, there is always more off-diagonal energy using non-symmetric mapping algorithm, as compared to the symmetric mapping algorithm. Intuitively, when using the symmetric mapping algorithm the weighted grid graph's spectrum does not change as much as when the non-symmetric mapping algorithm is used. Moreover, this result explains the experiments where it was found that in general the more symmetric the mapping is, the better the RD performance that can be achieved. Our analysis based on spectral graph theory can be used for measuring the performance of different mapping algorithms on a grid of any dimension.

Chapter 5

A Wavelet Based Approach for Overlapped Tiling

We have shown that our proposed 3D random image retrieval system using a 3D rectangular tiling scheme leads to greater transmission efficiency compared to the traditional cubic tiling scheme. The compression result with the new tiling scheme shows a nearly 30%/45% reduction in the average transmission rate. However, this reduction comes at the cost of a significant storage overhead (e.g., a factor of ten) as compared to the traditional cubic tiling scheme. In this chapter we propose a new technique to reduce the storage overhead while preserving improvements in transmission efficiency.

5.1 Wavelet-Domain Redundant Tiling

We start by applying the Haar wavelet to the 3D volumetric data X in order to generate different frequency 3D subbands, $\{W_{LLL}, W_{LLH}, W_{LHL}, \cdots, W_{HHH}\}$ as shown in Figure 5.1. For each band, different rectangular tiling settings can be used. While the same 3D compression method is used for all tiles (as described in Section 2.5.2), we allow each band to use different tile sizes, shapes and, most importantly, different levels of tile redundancy. When a random plane is requested,

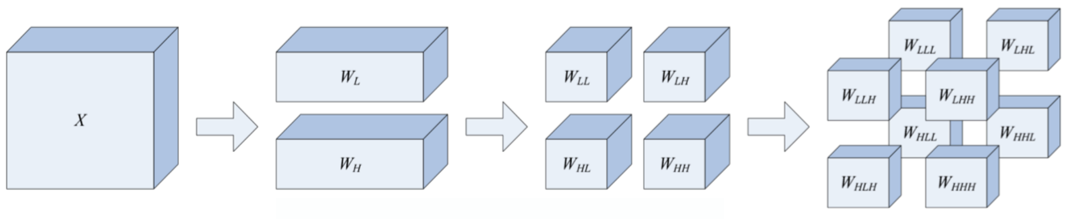


Figure 5.1: Applying Haar wavelet to the 3D volumetric data. This only shows the one level decomposition. Different levels decomposition are used in our experiments.

in all bands, the tiles associated with the requested plane are fetched from storage and transmitted. For instance, if a 3D volume size is $N \times N \times N$, pixel (x, y, z) on the requested plane will be associated with the coefficients $(\lceil x/N_i \rceil, \lceil y/N_i \rceil, \lceil z/N_i \rceil)$ in each band with decomposition level i, where $N_i = N/2^i$. The minimum number of tiles that cover the associated coefficients in all the bands will be selected and transmitted.

Table 5.1 shows the normalized transmission efficiency gains associated with different storage overheads for the optimized tiling scheme in Section 3.2.2.

So	1	11.5	18.3	25.1	33.25	46.6
Te	1	0.68	0.52	0.505	0.47	0.42

Table 5.1: Storage Overhead (S_o). Transmission efficiency (T_e). $T_e = T_r/T_c$, where T_r represents the average transmission bits using rectangular tiling scheme, and T_c represents the average transmission bits using cubical tiling scheme.

The goal here is to improve the storage overhead versus overall transmission efficiency by selecting the best tiling scheme for different frequency bands. Denote S_{ci} the storage (in bits) for band i, which is compressed using a cubic tiling scheme and our compression method. Then, once we decide a rectangular tiling scheme (with S_{oi} storage overhead and T_{ei} transmission efficiency) for band i, the total storage for band i using the rectangular tiling scheme (S_{ri}) can be approximated as $S_{ri} = S_{oi} \times S_{ci}$. T_{ri} and T_{ci} represent the average transmission bits for frequency band i using the overlapped tiling scheme and the non-overlapped cubical tiling scheme, respectively, and $T_{ri} = T_{ei} \times T_{ci}$. Therefore, the approximate overall storage and transmission efficiency can be computed as

$$S = \sum_i S_{oi} \times S_{ci}, \tag{5.1}$$

and

$$T_e = \frac{T_r}{T_c} = \frac{\sum_i T_{ri}}{\sum_i T_{ci}}. \tag{5.2}$$

The optimal settings for the bands can be obtained by using the Lagrange multipliers method. We define a new function Z

$$Z = S + \lambda \times T_e \tag{5.3}$$

where $\lambda \geq 0$ is the Lagrange multiplier. In this scenario, retrieving random planes from 3D data, the number of retrieved bits in frequency band i using the cubic tiling scheme, T_{ci}, is proportional to ${B_i}^2$, where $B_i \times B_i \times B_i$ is the size for the 3D frequency band i. We know S_{ci} is proportional to ${B_i}^3$. Therefore, we have that $T_{ci} = K \times \frac{S_{ci}}{B_i}$, where K is a constant. Equation (5.3) then can be written as

$$\begin{aligned} Z &= \sum_i S_{oi} \times S_{ci} + \lambda \frac{\sum_i T_{ei} \frac{S_{ci}}{B_i}}{\sum_i \frac{S_{ci}}{B_i}} \\ &= \sum_i S_{oi} \times S_{ci} + \lambda \frac{T_{ei} \frac{S_{ci}}{B_i}}{\sum_i \frac{S_{ci}}{B_i}} \\ &= \sum_i Z_i, \end{aligned}$$

where

$$Z_i = S_{oi} \times S_{ci} + \lambda \frac{T_{ei} \frac{S_{ci}}{B_i}}{\sum_i \frac{S_{ci}}{B_i}}.$$

Hence, $\min Z = \sum_i \min Z_i$. The simulation results for optimizing the tiling scheme for each frequency band are shown in Figure 5.2. Table 5.2 shows the total storage

J	0	2	3	4
S_c	2698345	2730124	2744989	2745917

Table 5.2: J: Decomposition level, J = 0 means without using Haar transform on the object; S: Total storage (in bits)

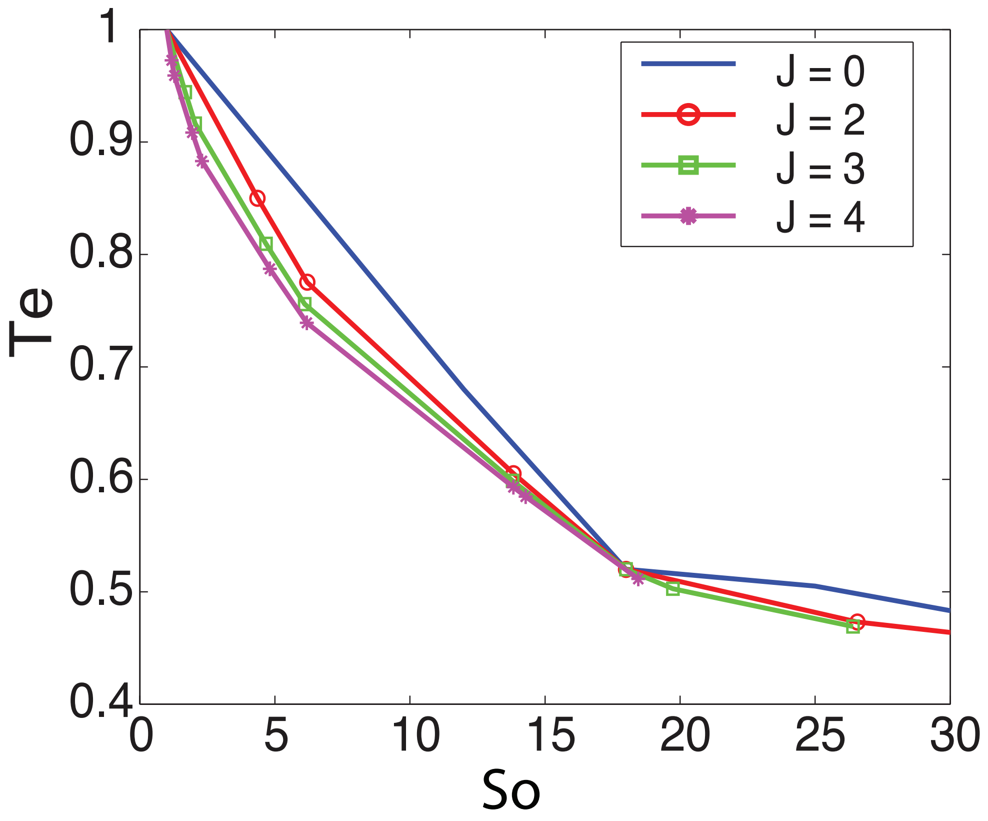


Figure 5.2: Approximation Result: Transmission efficiency (T_e) versus Storage overhead (S_o).

using the cubic tiling scheme for different Haar transform decomposition levels. The cubic tile size used in this experiment is $8 \times 8 \times 8$. Just as in Section 3.2.3, where the transmission rate vs. storage overhead trade-off of our previously published rectangle tiling scheme (without the Haar wavelet) was shown relative to the cubic tiling scheme (without the Haar wavelet), in Figure 5.2 we again show the relative trade off of this rectangular tiling scheme (without the Haar wavelet, labeled as "Rect $J = 0$") and that of the currently proposed scheme using the Haar wavelet, again with respect to the cubic tiling scheme (without the Haar wavelet). The result shows that using the wavelet based approach achieve 25% reduction in the average transmission rate with a storage overhead of just a factor of five. In this experiment, same block size of $8 \times 8 \times 8$ is used for compressing all the tiles in the different frequency bands. In future work, different block sizes can be used for compressing the tiles in different frequency bands (e.g., smaller tiles can be used in high frequency bands, leading to a lower compression rate). In addition, 3D DCT compression may not be an optimal compression tool for compressing the frequency bands after the Haar transform. Different compression methods will be investigated in future work.

5.2 Conclusions

We have proposed a new wavelet based approach, which could achieve 25% reduction in the average transmission rate at the cost of a factor of five in storage overhead (half of what was required for our previously proposed method). In this approach, a wavelet transform is first computed and then each of the 3D subbands can be tiled with different levels of redundancy in each subband. This allows us to select a more redundant representation for those bands requiring a higher bit-rate (e.g., lower frequency bands) while reducing the redundancy for bands whose bit-rate is already low (e.g., higher frequency bands). We demonstrate that similar reductions in bandwidth to those in Section 3.2.3 can be achieved but with significantly lower storage overhead. In future work, different block sizes can be used for compressing the tiles in different frequency bands (e.g., smaller tiles can be used in high frequency bands, leading to a lower compression rate). In addition, 3D DCT compression may not be an optimal compression tool for compressing the frequency bands after the Haar transform. Different compression methods will be investigated in future work.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We have proposed a server-client based data overlapped representation and retrieval system which can be used for fast random access of low dimensional data from high dimensional datasets. 2D and 3D systems have been proposed and evaluated, however, our approach can be extended to higher dimensional datasets. We used multiple redundant tilings of the image/3D data set, where each tiling has a different orientation. This approach achieved a substantial reduction (a factor of 2 in bandwidth reduction) in average transmission rate as compared to traditional square/cubic tiling. Using the first tiling strategy, transmission rate can be reduced by 20% - 60% in 2D case and 15% - 55% in 3D case. Additionally, this approach leads to improved random access, with less storage and run-time memory required at the client.

Our proposed system consists of 5 components: 1) the rectangular tiling scheme, 2) the mapping algorithm, 3) the tile searching algorithm, 4) compression, and 5) random line/oblique plane reconstruction and display. The five components of the system are very different for the 2D and 3D cases. We have designed all the components for both 2D and 3D cases. The rectangular tiling scheme provides multiple redundant rectangular tilings of the object and the tiles have different orientations. The searching algorithm determines which tiles should be retrieved for a given query while the mapping algorithm enables efficient coding without interpolation of rotated tiles.

Two models (2D and 3D cases) for our system were also proposed that allow us to optimize system parameter selection without the need for simulations of system performance. The proposed models can be used to predict transmission rate and storage, given how the cells are partitioned. Then, based on the insights obtained from the proposed models, two new rectangular tiling schemes (2D and 3D cases) have been proposed, which reduce the transmission rate an additional 10% - 30% as compared with the first two tiling schemes (15% - 50% for 2D and 3D cases comparing with the traditional non-overlapped tiling schemes).

We also evaluated in detail the mapping algorithm for rotated tile encoding, which showed that the lack of interpolation means that complexity is significantly lower and the running time is 20 to 30 times less. Moreover, remapping without interpolation leads to overall much better RD performance and the more symmetric the mapping is, the better RD performance that will be achieved. Using spectral graph theory approach, we proposed a tool that can analyze and quantify the performance and benefits of our proposed re-mapping algorithm.

Furthermore, we have proposed a new wavelet based approach in the 3D case, which can potentially achieve 25% reduction in the average transmission rate at the cost of a factor of five on server's side. The 2D system has the potential to be used in map applications on small devices, eg. GPS and mobile phone. The 3D system has the potential to be used in mobile devices.

6.2 Summary of Future Work

We summarize the future work as follows.

(i) For the more difficult 3D case, currently, we use a simple 3D DCT based compression scheme with uniform quantization. In the future work, different compression schemes can be used.

(ii) For the wavelet based approach, different block sizes can be used for compressing the tiles in different frequency bands (e.g., smaller tiles can be used in high frequency bands, leading to a lower compression rate). In addition, 3D DCT compression may not be an optimal compression tool for compressing the frequency bands after the Haar transform. Different compression methods will be investigated in future work.

(iii) In the future, users' statistics can be considered. Different part of the image/3D data can be tiled with different size and with different redundancy.

(iv) Caching can also be considered to optimize the tiling scheme to achieve better performance.

(v) While the applications potentially can be used on different devices, the resolution and bandwidth requirements may vary. Scalable coding can be used to compress tiles.

(vi) Data streaming can also be used in some future 3D random access applications.

(vii) Different tile sizes and angles as well as other tiling methods can be explored for potential gain.

Reference List

[1] Information technology - JPEG 2000 image coding system: Part 10 - extensions for three-dimensional data (jp3d) - fcd v1.0. *ISO/IEC JTC1/SC29/WG1 N4101*, 2006.

[2] T. Bruylants, A. Munteanu, A. Alecu, R. Deklerck, and P. Schelkens. Volumetric image compression with JPEG2000. In *SPIE The International Society for Optical Engineering*, 2007.

[3] Y. Cho and W. A. Pearlman. Hierarchical dynamic range coding of wavelet subbands for fast and efficient image compression. *IEEE Trans. Image Processing*, 16(2005-2015), Aug 2007.

[4] Y. Cho, A. Said, and W. A. Pearlman. Low complexity resolution progressive image coding algorithm: PROGRES(PROGressive RESolution decompression). In *IEEE ICIP*, 2005.

[5] Coxeter and H. S. M. *Regular Polytopes*. ISBN 0-486-61480-8. Dover Publications., New York, 3rd edition edition, 1973.

[6] Z. Fan and A. Ortega. Overlapped tiling for fast random access of 3-d datasets. In *Proc of Data Compression Conference (DCC)*, 2009.

[7] Z. Fan and A. Ortega. Overlapping tiling for fast random access of low-dimensional data from high-dimensional datasets. In *SPIE Multimedia Content Access: Algorithms and Systems III*, 2009.

[8] Z. Fan and A. Ortega. Optimization of overlapped tiling for efficient 3d image retrieval. In *Proc of Data Compression Conference (DCC)*, 2010.

[9] Z. Fan and A. Ortega. Wavelet-based redundant representation for efficient random access of volumetric images. In *IEEE ICIP*, 2010.

[10] Zihong Fan and A. Ortega. Mapping data on a rotated grid in high-dimensions for lossless compression. In *IEEE ICIP*, 2011.

[11] Zihong Fan and A. Ortega. A spectral graph theory approach for data remapping. 2011.

[12] I. Ihm and S. Park. Wavelet-based 3D compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum*, March 1999.

[13] S. Lee and A. Ortega. A novel approach for image compression in digital cameras with bayer color filter array. In *IEEE ICIP*, 2001.

[14] S. Muraki. Volume data and wavelet transforms. In *IEEE Trans. Computer Graphics and Application*, July 1993.

[15] J. P. W. Pluim and J. M. Reinhardt. Compression of medical volumetric datasets: physical and psychovisual performance comparison of the emerging JP3D standard and JPEG2000. In *SPIE, Medical Imaging 2007: Image Processing.*, volume 6512, 2007.

[16] A. Said and W. A. Pearlman. A new, fast, and efficient image codec using set partitioning in hierarchical trees. *IEEE Trans. Circuits and Systems for Video Technology*, pages 243–250, June 1996.

[17] Yung-Gi Wu Shen-Chuan Tai and Chang-Wei Lin. An adaptive 3-d discrete cosine transform coder for medical image compression. In *IEEE Transactions on Information Technology in Biomedicine*, 2000.

[18] Akihiko Sugiura and Minoru Inatsu. A study of dct image coding using adaptive three- dimensional scanning. In *Electronics and Communications in Japan, Part 3*, volume 79, pages 103–112, 1996.