Overlapped Tiling for Fast Random Access of Low Dimensional Data from High Dimensional Datasets

Zihong Fan and Antonio Ortega

Signal and Image Processing Institute
Dept. of Electrical Engineering
Univ. of Southern California, Los Angeles, CA, USA

ABSTRACT

Volume visualization with random data access poses significant challenges. While tiling techniques lead to simple implementations, they are not well suited for cases where the goal is to access arbitrarily located subdimensional datasets (e.g., displaying a "band" of several paralllel lines of arbitrary orientation from a 2D image; being able to display an arbitrary 2D planar "cut" from a 3D volume). Significant effort has been devoted to volumetric data compression, with most techniques proposing to tile volumes into cuboid subvolumes to enable random access. In this paper we show that, in cases where subdimensional datasets are accessed, this leads to significant transmission inefficiency. As an alternative, we propose novel server-client based data representation and retrieval methods which can be used for fast random access of low dimensional data from high dimensional datasets. In this paper, 2D experiments are shown but the approach can be extended to higher dimensional datasets. We use multiple redundant tilings of the image, where each tiling has a different orientation. We discuss the 2D rectangular tiling scheme and two main algorithm components of such 2D system, namely, (i) a fast optimal search algorithm to determine which tiles should be retrieved for a given query and (ii) a mapping algorithm to enable efficient encoding without interpolation of rotated tiles. In exchange for increased server storage, we demonstrate that significant reductions, a factor of 2 in bandwidth reduction, can be achieved relative to conventional square tiling techniques. The transmission rate can be reduced even more by allowing more storage overhead. This method speeds up the random access procedure and saves memory on the user's side. Here we use the 2D example to retrieve random lines (or sets of lines) from a 2D image. While our experiments are based on extracting 1D data from 2D datasets, the proposed method can be extended to 3D or higher dimensions. The associated basic concepts and analysis (namely the extraction of 2D slices from 3D datasets) and a more detailed discussion focusing on the 3D and higher dimensional case will be presented in another paper. In this paper, we design a tiling method that locates the rotation centers at points on a square Cartesian grid pattern and has the tile rotation angles uniformly spread out around each rotation center. The angles of the tiles associated to each rotation center are the same. Other various ways of tiling method design are also possible. The performance by using other tiling methods will be addressed in future work.

Keywords: fast random access, retrieval, tiling

1. INTRODUCTION

In many fields (e.g., biomedical imaging, earth sciences, computational fluid dynamics, satellite image processing, etc.) researchers need 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 are efficient when most of the information included in the tiles is used for display. This is the case, for example, when a 2D region of a larger 2D dataset (e.g., a map) is to be displayed or when small sub-volumes of a large 3D volume are needed[1] (i.e., the dimension of the subset and the dataset from which it is extracted are equal). In this

Further author information: (Send correspondence to Zihong Fan)
Zihong Fan: E-mail: zihongfa@usc.edu, Telephone: 1 213 740 4679
Antonio Ortega: E-mail: antonio.ortega@sipi.usc.edu, Telephone: 1 213 740 2320

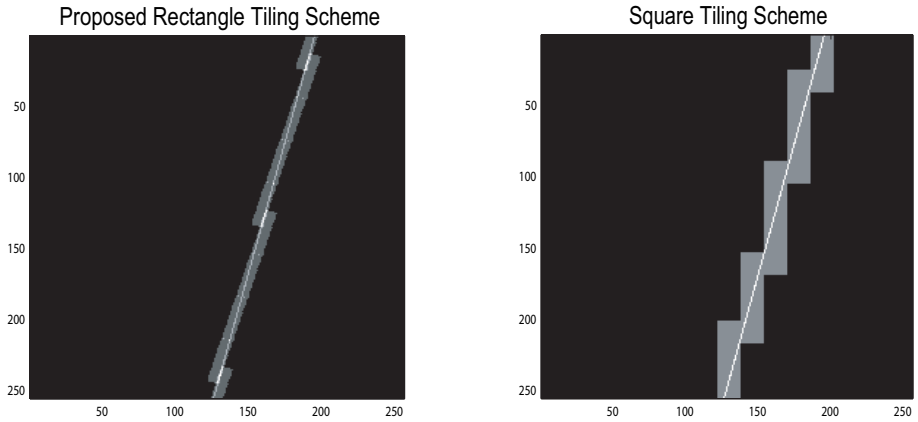


Figure 1. Comparison of rectangular and square tiling schemes in 2D case (where we retrieve a random line from a plane). Only 10 rectangular tiles are needed, instead of 21 square tiles (note that tiles have the same size 8×32 and 16×16, respectively).

paper we tackle a more challenging problem for which conventional tiling techniques are inefficient. Specifically, we focus on situations where lower dimensional portions of a dataset need to be accessed. For example, in the volumetric image example, arbitrary oblique planes of the volume may need to be extracted and rendered, as is required in some medical imaging applications. Standard tiling can be inefficient in this scenario, because for each retrieved cubic tile the only voxels* that are "useful" are those near the intersection between the cube and the desired 2D plane. Because tiles are the basic unit for compression, 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 another example, this time 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). As we shall see, it is more efficient to use overlapping rotated tiles to represent the data-set; this leads to an increase in the average number of useful voxels per tile so that the total number of tiles to be retrieved is smaller (and hence a lower transmission bitrate is achieved). This comes at the cost of an increase in storage at the server. Thus we trade-off increased storage at the server's side for lower bandwidth during the interactive access to the data-set.

Many techniques have been proposed for volumetric image coding,[1–5] including approaches such as JP3D.[6–8] In all cases, some form of random access is provided via tiling, with non-overlapping, independently encoded cuboid subvolumes being used. Clearly, the improved random access achievable by tiling comes at the expense of some reduction in coding efficiency (i.e., there would be less overhead if bigger tiles were used for encoding). The main novelty of our approach comes from further exploring this trade-off by allowing tiles that i) overlap and ii) are rotated at various angles. In this paper, we illustrate our proposed tiling scheme with 2D examples, where the goal is to retrieve arbitrary oblique lines from 2D images, but our method can be extended to higher dimensions with some modifications. Extension to 3D with further details will be presented in another paper. We demonstrate that by using overlapped tiles with different orientations, we can achieve lower transmission overhead for arbitrary access to lower dimensional datasets, at the cost of requiring additional storage at the server.

In our preliminary results in the 2D case, the average number of bits transmitted when using the proposed tiling scheme can be reduced by a factor of 2 relative to the square tiling scheme. The reduction can be even greater by allowing additional storage overhead using different tile sizes. We start by considering tiling methods, parameterized by the location and orientation of the tiles, that can guarantee that all data can be retrieved (Section 2). Because tiles overlap, there is no longer a unique way to retrieve data for a given query, and so we propose a 2D search algorithm to identify the most efficient set of tiles to be transmitted in response to a query (Section 3.1). We also propose a simple mapping technique to compress the data points on the rotated tiles that do not coincide with the Cartesian grid points. This mapping technique is shown in 2D case to outperform tile representation methods based on interpolation (Section 3.2). Experimental results are provided in Section 4. In Section 5, a metric will be proposed for the parameters selection and we conclude the paper in Section 6.

*a volume element, representing a value on a regular grid in three dimensional space.

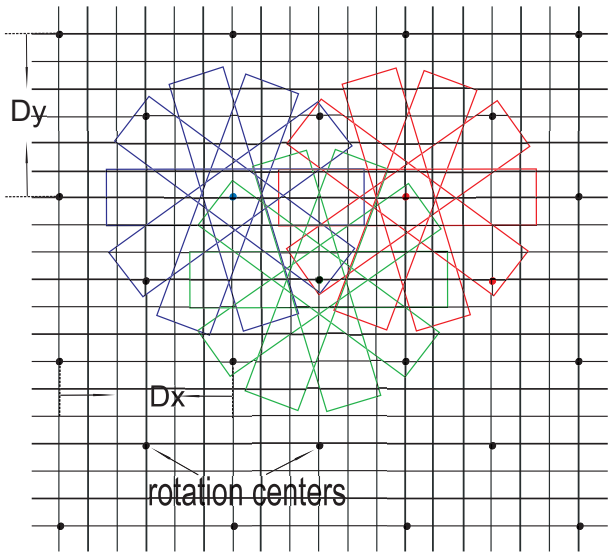


Figure 2. Example of proposed overlapped rectangular tiling.

2. RECTANGULAR TILING SCHEME FOR EFFICIENT TRANSMISSION WITH RANDOM ACCESS

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)[†]. Since queries of interest are lines (or sets of lines) at arbitrary angles in a 2D image, random planes from a 3D data set or random hyperplane from higher dimensional data set, this suggests that rotated tiles should be used. The potential benefits of the proposed method are illustrated in Figure 1 in 2D case. It can be seen that fewer tiles have to fetched when rectangular rotated tiles are used (in this example rectangular and square tiles contain the number of pixels closely). 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". 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 5. For a given tiling strategy we compute the average number of tiles per pixel (i.e., the average redundancy for a pixel), which is a proxy 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 this paper, we present a design that assumes that a series of "rotation centers" are chosen, in which several tiles of different orientations are centered. The pattern of the rotation centers can be square, hexagonal, or triangular, etc. and the rotation angles can be different for each rotation center. In this paper, 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. A 2D example where random lines requested from an image are used to illustrate our proposed method. Our analysis in Section 5 will show that the choice of parameters for this tiling structure can be based on the variance/average in the corresponding number of tiles per pixel.

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

The tiling layout in our 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, the horizontal and vertical distance between rotation centers on the same Cartesian horizontal or vertical grid, as illustrated in Figure 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 is covered by at least one tile. Each rectangular tile is compressed for storage. In this paper, we use rectangular sizes of 8×32 pixels and square sizes of 16×16. In the current version we use JPEG compression for each 8×8 block, using the mapping algorithm for rotated tiles to be discussed in Sections 3.2 and 4. The parameters can be chosen to achieve different levels of trade-off between transmission efficiency and storage overhead.

3. ALGORITHMS FOR THE RECTANGULAR TILING SCHEME

After selecting a tiling strategy, two algorithms have to be designed for our proposed approach. First, since multiple sets of tiles can be used to answer any given query, we need to design an efficient algorithm to determine which set of tiles answers a query with minimum rate. Algorithm 1 is provided to achieve this goal. Second, since data is rotated before coding we need to define algorithms to map original data to the corresponding rotated sampling grids. Refer to Algorithm 2. This mapping algorithm will be presented here in 2D case and can be shown to achieve better coding performance (with lower complexity) as compared to techniques that would interpolate pixel values in the rotated tile prior to compression.

3.1 Optimal Fast Rectangle Search Algorithm

As discussed in Section 2, overlap between rectangles means that there are multiple ways (different rectangle sets) to provide all the information corresponding to a line in 2D. This leads to a set coverage problem with the objective of finding which collection of rectangles representing a line is best for transmission. Normally, set coverage problems are NP problems, potentially resulting in considerable complexity. Our particular problem, however, 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. 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 will be roughly proportional to the number of rectangles that have to be retrieved. Note that 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[‡], 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 at the rotation centers, which are close to the line and the starting point S_i, need to be searched.

Algorithm 1 Optimal Rectangle Cover Searching Algorithm

1: Starting with the beginning point of the line (S_1 in Figure 3), find the rectangle which covers the starting point (S_1) and covers the greatest amount of the line. We only need to look at the rectangles at the rotation centers which are close to the line.
2: Set the point after the end point of the line segment covered by the rectangle as the new starting point. S_i is the new starting point for the i^{th} iteration (e.g. S_2 is the new starting point for the second iteration in Figure 3).
3: Remove the line segment covered by the rectangle. Record the rectangle.
4: Until the line is completely covered, repeat steps 1 to 3 with the new starting point S_i.

[‡]In Figure 3, $\{S_1, S_2, S_3, \ldots\}$ 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.

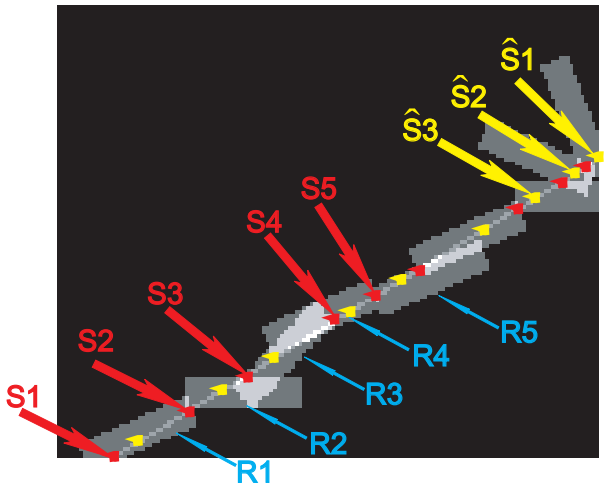


Figure 3. 2D demonstration of the optimal rectangle cover searching algorithm.

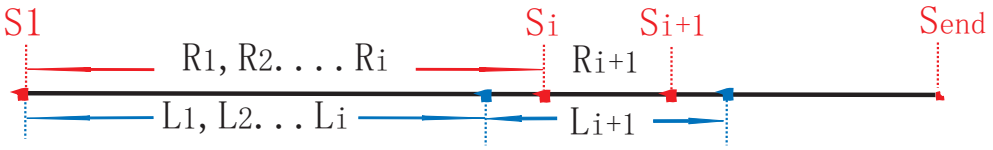


Figure 4. Demonstration of the proof for Part 1.

Proof. [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 of the other techniques. To prove that our algorithm is optimal, we need to show that $m \geq n$. Assume that for each index $i < n$

$$\mathcal{C}\{L_1, L_2, ...L_i\} \subseteq \mathcal{C}\{R_1, R_2, ...R_i\}, \tag{1}$$

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

$$\mathcal{C}\{L_1, ..., L_i, L_{i+1}\} \subseteq \mathcal{C}\{R_1, ..., R_i, R_{i+1}\}. \tag{2}$$

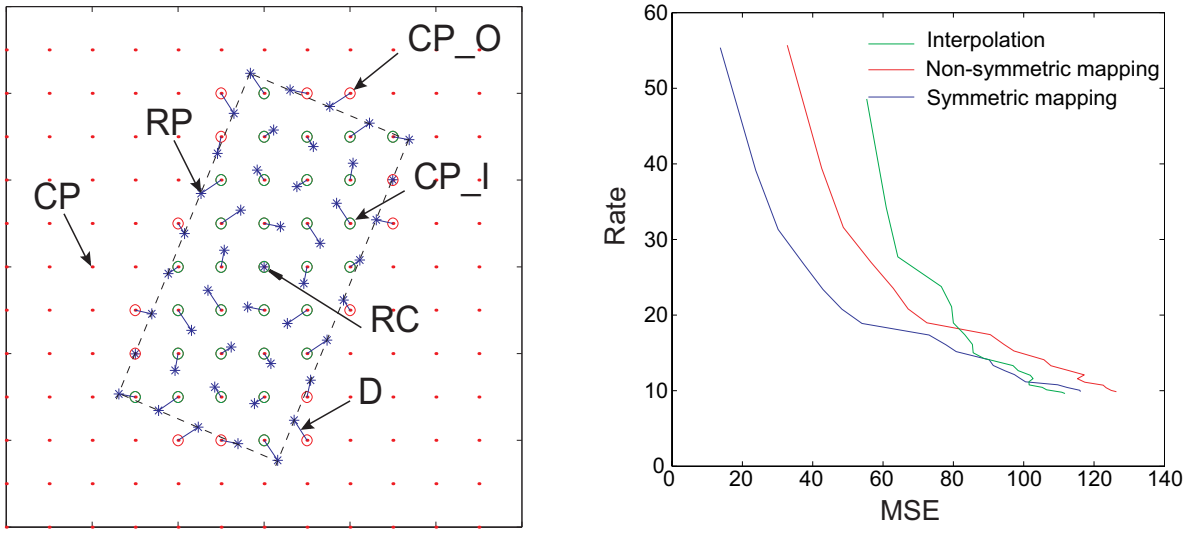
Equation (2) follows by considering Figure 4, where we see that if there existed a rectangle L_{i+1} which would satisfy the relationship

$$\mathcal{C}\{L_1, ..., L_i, L_{i+1}\} \supsetneq \mathcal{C}\{R_1, ..., R_i, R_{i+1}\}, \tag{3}$$

then the endpoint of L_{i+1} must 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) is thus proved by contradiction. Based on the induction of (1) and (2) , we have that

$$\mathcal{C}\{L_1, ..., L_i, ..., L_n\} \subseteq \mathcal{C}\{R_1, ...R_i, ..., R_n\}. \tag{4}$$

[§]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.



(a) (b)

Figure 5. (a) 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). (b) Rate distortion for two mapping algorithms and the interpolation method in 2D case. Using "Lena" as the test image. $N = 10$, $D_x = D_y = 20$.

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

$$\mathcal{C}\{L_1\} \subseteq \mathcal{C}\{R_1\}. \tag{5}$$

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 (1), (2), and (5), the rectangle searching algorithm we propose is optimal according to our definition. Note that as mentioned earlier, it may be shown that this greedy solution is actually globally optimal as well. □

Additionally, regardless if one starts from either the beginning or end of the line, an optimal solution (achieving the minimum number of tiles to cover the line, even if those tiles themselves are different) will be found. This may be easily shown as follows. In Figure 3, $\{\hat{S}_1, \hat{S}_2, \hat{S}_3, \ldots\}$ are the starting points resulting from applying the same cover searching algorithm when started from the "end" of the line. These correspond to the points $\{S_1, S_2, S_3, \ldots\}$ 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. For example, if $\hat{S}_k$ and $\hat{S}_{k+1}$ were between S_i and S_{i+1}, then the selected tile which covers $\hat{S}_k$ would not be the tile that covers the greatest amount of the line, since the tile covering the region S_i to S_{i+1} covers the most, including the point $\hat{S}_k$. Thus the number of points S_i is equal to the number of points $\hat{S}_k$. From Algorithm 1, we know the number of the starting points is equal to the number of selected tiles. 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.

3.2 Mapping Algorithm for Rotated Tile Encoding

Consider a rotated rectangle of length L and width W. From Figure 5(a), we see that the points on the rotated rectangle grid do 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.

This idea is general and the algorithm could be extended to higher dimensions. Our proposed algorithm is illustrated by using 2D case in Figure 5(a). 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. This "mapping" process is shown by the blue lines in Figure 5(a). Note that 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 (D in Figure 5(a)), i.e., the 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. Our proposed mapping algorithm seeks to: i) map all Cartesian points inside the rectangular area, ii) minimize the average and maximum of the pointwise mapping distances D, and iii) be symmetric about the rotation center. Starting from the rotation center, we map the Cartesian and rotated grid points to each other in a 1-to-1 mapping moving from the center of the rectangle outward. Refer to Algorithm 2 (experimental results are provided in Section 4). 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.

Algorithm 2 Mapping Algorithm

Find the Cartesian points inside the rotated rectangle area (CP_I in Figure 5(a)). Calculate the distances from the Cartesian points (from the step above) to the rotation center. Sort the distances, calling the sorted list A.
Calculate the distances from the rotated rectangle grid points (RP in Figure 5(a)) to the rotation center. Sort the distances, calling the sorted list B.
while $i \leq$ length(A) and $k \leq$ length(B) **do**
  while $A(i)$ has been mapped **do**
    $i = i + 1$
  end while
  while $B(k)$ has been mapped **do**
    $k = k + 1$
  end while
  if $A(i) \leq B(k)$ **then**
    Find the rotated grid point closest to $A(i)$ from the unmapped points in B.
    Record the mapping points and label the points as "mapped" in A and B.
    $i = i + 1$
  else
    Find the unmapped Cartesian point closest to $B(k)$.
    Record the mapping points and label $B(k)$ as "mapped".
    Also label the Cartesian point as "mapped".
    The Cartesian point may or may not be in A.
    $k = k + 1$
  end if
end while

4. EXPERIMENTAL RESULTS

In this section we evaluate our proposed methods. We tile a 2D image (both Lena and MRI images were used) and use our search algorithm (Section 3.1) to identify the best tiles to display a random oblique line within the image. 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. Hence, if the size of the image is $M \times M$ and the length of the line segment is M, then M uniformly distributed samples on the line segment will be chosen. 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. The pixels' values within a line are calculated by using a simple "nearest neighbor" interpolation at the uniformly sampled positions on the line segment. 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).

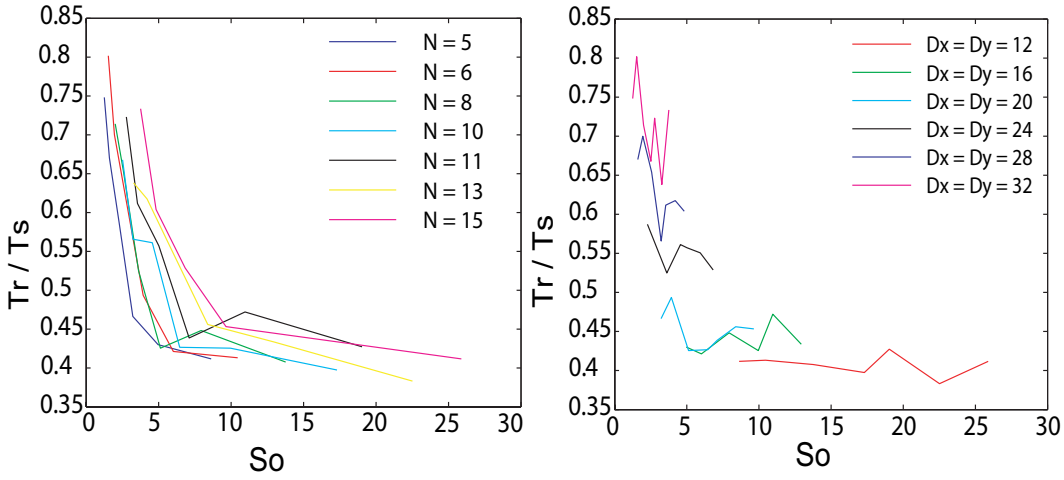


Figure 6. T_e versus S_o: left side, for a fixed value of N, D_x and D_y are varied; right side, for a fixed value of D_x, D_y, N is varied. Using JPEG for compression.

4.1 Mapping algorithm

We start by comparing the various mapping algorithms. From Figure 5(a), 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 of mapping techniques is shown in Figure 5(b) 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. A situation analogous to that observed in compression of Bayer filter images[9] arises: interpolation 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.

4.2 Result for Proposed method

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 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 study, the rotation center parameters D_x and D_y and the number of rotation angles N vary, while we fix $W = 8$ and $L = 32$.

Figure 6 (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. 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 6 (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_o). 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).

5. METRIC FOR PARAMETERS SELECTION

Thus far, we have demonstrated that by using our proposed method the transmission rate can be reduced by 20% - 60% depending on the desired trade-off with storage overhead, compared to the conventional tiling scheme. This is shown in Figure 6. As we mentioned earlier, for a fixed storage overhead there may be possibly several different parameters

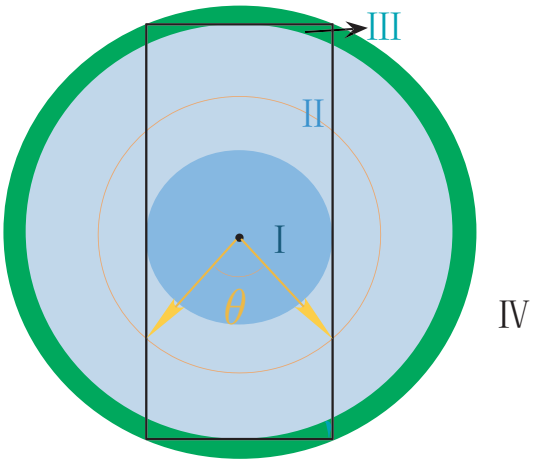


Figure 7. Illustration of the metric for calculating the average number of tiles of an arbitrary pixel around one rotation center.

settings (D_x, D_y, N) which achieve different level transmission efficiencies. How to choose the setting to achieve the best transmission efficiency given a fixed storage overhead? In this section, a metric will be proposed for the parameter selection that provides insight as to why different parameter settings produce different transmission efficiencies with a fixed storage overhead.

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 rectangle tile and N be the number of the tiles associated with one rotation center. Assume the tiles are distributed uniformly around the rotation center without fixing the rotation angles. Then the rotation angle of each tile can be written as $\varphi_k = \frac{\pi}{N} \cdot (k-1) + \varphi_1$, where tile index $k = 1, 2, \cdots, N$, and the angle of first tile φ_1 is a random variable with uniform distribution $\mathbf{U}(0, \frac{\pi}{N})$. Figure 7 illustrates the possible cases involved in computing the number of tiles covering one pixel. 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 7, notice that the coverage probability depends on the distance (d) from the pixel (x, y) to the rotation center (x_c, y_c), where $d = \sqrt{(x-x_c)^2 + (y-y_c)^2}$. Hence the coverage probability for any pixel can be calculated as below. There are four cases categorized by the value of d:

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

$$p(x, y) = 1$$

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

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

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

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

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

$$p(x, y) = 0$$

Then, associated with one rotation center, the average number of tiles covering the pixel can be calculated by $\llcorner p(x, y) \times N \lrcorner$. Similarly, considering all the neighborhood rotation centers of the pixel[¶], the average number of tiles covering the pixel j is $T_j = \sum_i \llcorner p_i(x, y) \times N \lrcorner$, $\forall$ rotation center i $\in$ neighborhood(pixel j),where p_i is the probability, that pixel j is covered

[¶]We consider the neighborhood rotation centers of the pixel are the ones with the distance to the pixel less or equal to $\sqrt{(L/2)^2 + (W/2)^2}$.

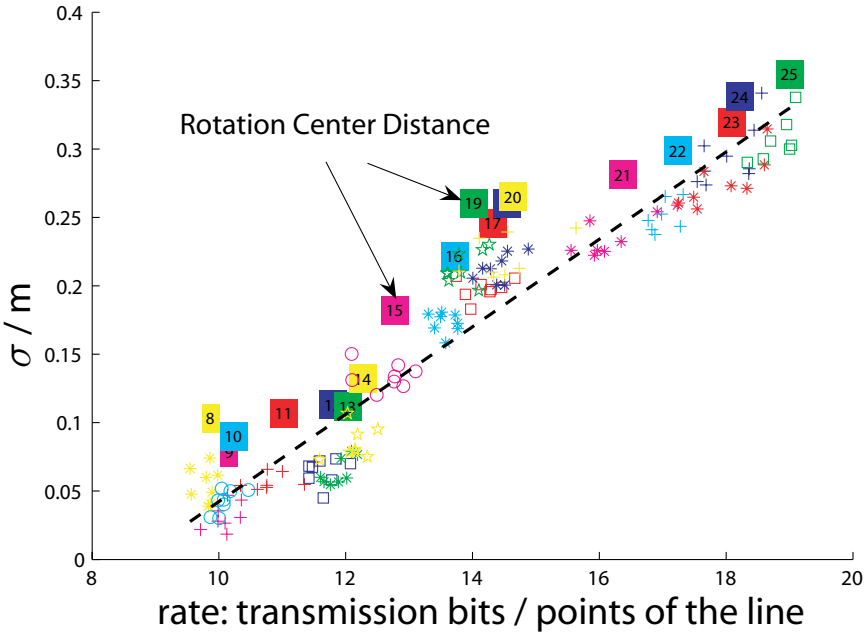


Figure 8. Linear regression: σ/m vs. rate

by one tile associated with the neighborhood rotation center i, . Hence the average (m) and the standard deviation (σ) of the tile coverage can be calculated. This only needs to be done for one Voronoi region spanned by the rotation centers, since rotation centers are located uniformly. Voronoi region associated with the rotation center C_i is defined by

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

where $d(p, C_i)$ is the distance between pixel p to the rotation center C_i. Denote $M = |V(C_i)|$, the number of pixels in $V(C_i)$. m and σ can be calculated by using one Voronoi region.

$$m = \frac{1}{M}\sum_{j=1}^{M} T_j \qquad \sigma = \sqrt{\frac{1}{M}\sum_{j=1}^{M}(T_j - m)^2}$$

Using linear regression, Figure 8 shows that there is a highly linear relationship between the ratio σ/m characterizing the tile coverage and the required rate.[‖] Figure 9 represents the σ/m as a function of average storage (m) for different parameter settings, where focusing on the range of m $0 \sim 40$. Note that the storage overhead showing in Figure 8,9 are without using compression technique. From Figure 8, we can see for a fixed rectangular tile size, in order to achieve lower transmission rate, a lower σ/m is preferred. Hence, for the same storage overhead, the parameter setting, which produces the smallest σ/m should be chosen. Notice that in our proposed 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 lead to smaller σ/m. In order to find the optimal parameter settings considering both m and σ/m to be small from Figure 9, a polynomial may be fit to the lower portion of the convex hull defined by the points in the figure, with which the parameters are associated would be preferred for selection.

6. CONCLUSIONS AND FUTURE WORK

In this paper we proposed a new method for retrieving lower dimension data from higher dimension data with fast random access. This paper has focused on the 2D case, but the methodology may be extended to the 3D case (and higher dimensions). The detailed discussion of the 3D case is left for another paper. In our preliminary results we have demonstrated

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

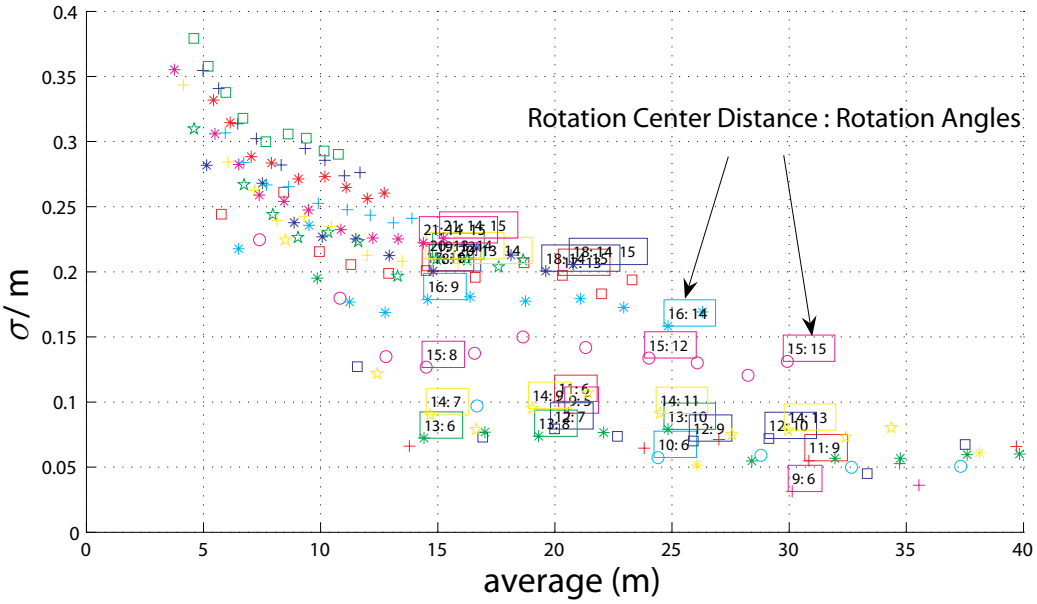


Figure 9. Linear regression: σ/m vs. average storage(m), $m < 40$ (without using compression technique.)

that by using overlapped tiles with different orientations and allowing some storage overhead on the server's side, transmission rate can be reduced by 20% - 60% depending on the desired trade-off with storage overhead, as compared to the conventional tiling scheme. This method has the potential to considerably speed up the random access procedure, requiring less data storage at the client compared to conventional tiling. We demonstrated a fast rectangle searching algorithm. We also have developed a mapping algorithm, which has low complexity, preserves the Cartesian grid data and allows reconstruction very easily and efficiently. Better performance at high rate has been shown for the mapping algorithm when compared to the interpolation method. In our current tiling scheme, the angle setting for the tiles associate with all rotation centers are the same. The potential gain by using different angle settings for different rotation centers will be presented in another paper.

REFERENCES

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

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

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

[4] Muraki, S., "Volume data and wavelet transforms," in [*IEEE Trans. Computer Graphics and Application*], (July 1993).

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

[6] "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).

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

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

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