Overlapped Tiling for Fast Random Oblique Plane Access of 3D Object Datasets

Zihong Fan and Antonio Ortega
Signal and Image Processing Institute
Dept. of Electrical Engineering
University 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., 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 oblique plane from 3D volume datasets. In this paper, 3D experiments are shown but the approach may be extended to higher dimensional datasets. We use multiple redundant tilings of the 3D object, where each tiling has a different orientation. We discuss the 3D rectangular tiling scheme and two main algorithm components of such 3D system, namely, (i) a 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 in average transmission rate can be achieved relative to conventional cubic tiling techniques, e.g., nearly 40% reduction in average transmission rate for less than a factor of twenty overhead in storage before compression. Note that, as shown in our earlier work on the 2D case, the storage overhead will be lower after compression (e.g., in 2D the relative increase in storage in the compressed domain was at least a factor of two lower than in the uncompressed domain.)

I. Introduction

In many fields (e.g., biomedical imaging, earth sciences, computational fluid dynamics, 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 [1]–[6] , 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 [2] (i.e., the dimension of the subset and the dataset from which it is extracted are equal). In this 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[1] 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 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 [2]–[6], including approaches such as JP3D [7]–[9]. 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 3D examples, where the goal is to retrieve arbitrary oblique plane from 3D objects, but our method may be extended to higher dimensions. 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.

Considering the actual storage (in bits) to be proportional to the volume of the encoded 3D tiles, the experimental results demonstrate the average number of bits transmitted when using the proposed tiling scheme can be nearly 15% - 55% lower for different storage overhead, as compared to the traditional cubic tiling scheme. The reduction can be even greater by allowing additional storage overhead using different size of tiles.

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 II). Because tiles overlap, there is no longer a unique way to retrieve data for a given query, and so we propose a 3D search algorithm to identify the most efficient set of tiles to be transmitted in response to a query. 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. Finally, we conclude the paper in Section V.

II. 3D Rectangular Tiling Scheme For Efficient Transmission With Random Access

The intuition behind the proposed redundant tiling is that if each voxel is available from more than one tile, one can achieve lower bandwidth on average by delivering the set of tiles that provide all requested voxels most efficiently (i.e., with lowest number of tiles required)[2]. Since queries of interest are planes at arbitrary angles this suggests that rotated tiles should be used. Thus, we assume that a series of "rotation centers"

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

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

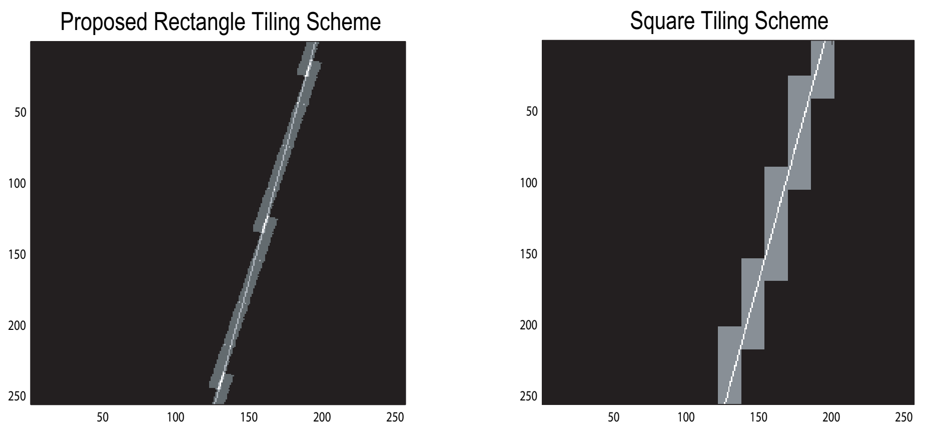
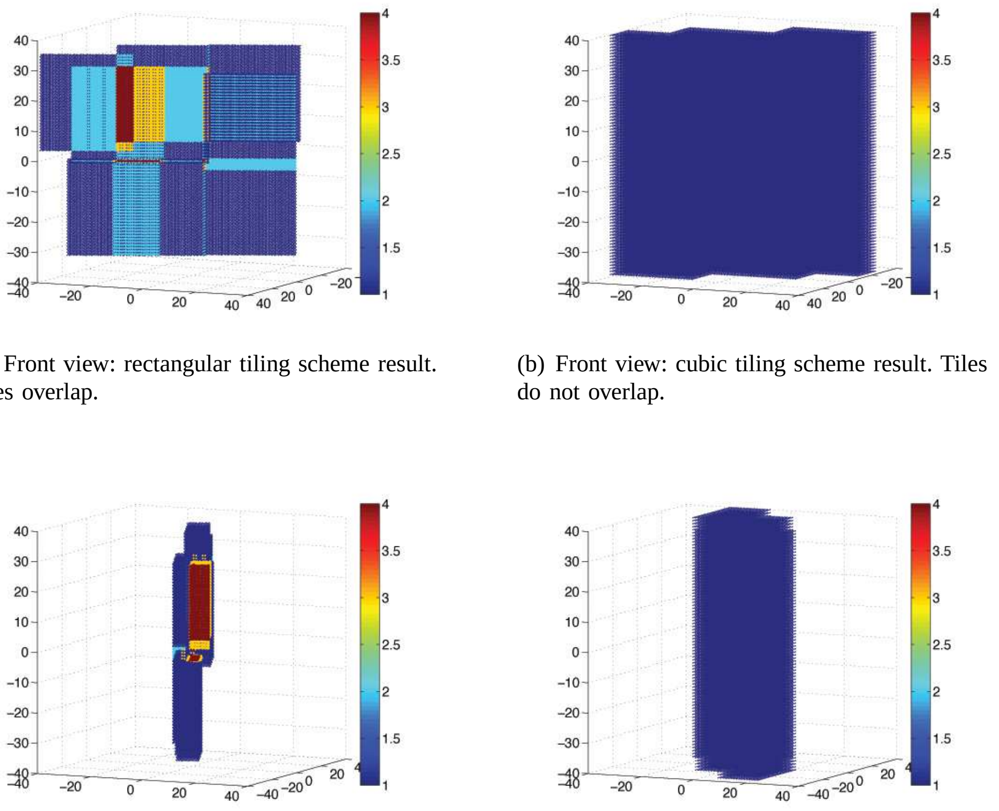


Fig. 1. Comparison of rectangular and square tiling schemes in 2D case(retrieve a random line from a plane). Just 10 rectangular tiles are needed, instead of 21 square tiles (note that tiles have the same size 8×32 and 16×16, respectively).

are chosen, in which several tiles of different orientations are centered. Using rotated rectangular tiles will lead to a lower average transmission rate because each selected tile will lie along the requested plane, which will lead increases in the average number of voxels contained in the intersection between planes being requested and the "best matching tiles". Figure 1 illustrates the potential benefits using a 2D example(with the line being requested for display). 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). Clearly, the number of tiles to be transmitted will depend on the data being requested. Figure 2 illustrates achieving lower transmission rate 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, 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 great amount of transmission bits can be reduced using our proposed method (seen in the side views and the retrieved number of the tiles) , where much more voxels per tile containing the voxels of the requested plane.

Figure 3 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, the distance between rotation centers along the x, y and z directions on the Cartesian grid. Because the selected tiles should lie along the requested plane in order to increase the size of the intersection, we choose $W = L$ and $H < W$, so that the 3D tiles are themselves "thin". The pattern of the rotation centers could be square, hexagonal, or triangular, etc. in the 2D case and the rotation angles can be different for each rotation center. For 3D case, similarly, the pattern of the rotation centers could be cuboctahedral, octahedral, icosahedral or tetrahedral, etc. and the rotation angles can be different for each rotation centers. In this paper 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 [10]. Cases using different patterns for the rotation centers and using



(a) Front view: rectangular tiling scheme result. Tiles overlap.

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

(c) Side view: rectangular tiling scheme result. Tiles lie alone the plane.

(d) Side view: cubic tiling scheme result.

Fig. 2. 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).

different rotation angles will be discussed in future work. Uniformly rotated 3D tiles can be obtained by making their norm vectors (refer to Figure 4) be spread uniformly in a sphere around the rotation center. This means that for tile i, with norm vector $\vec{n_i}$, will be such that this norm vector will be at a constant angle Δ_α, with respect to the norm vectors, $\vec{n_i}$, of all its immediate neighbors. In Figure 4 (b), the norm vectors are 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. The 3D tile angle selection is illustrated by Figure 4 and the number of the tile angles obtained by using the different tessellation methods in this paper are shown in Table I. Different tile angle settings can be used associated to different rotation centers. In this paper, when different tile angle settings are used, the rotation centers with different tile angle settings are interleaved. For the tiling layout in our proposed system, parameter configurations which can cover all the pixels in the 3D

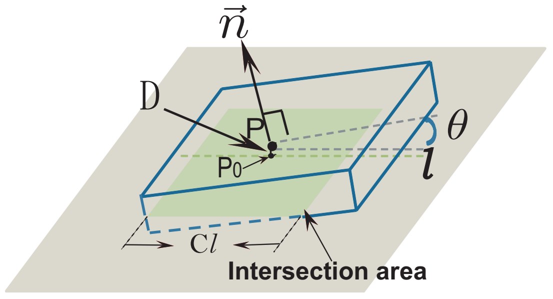
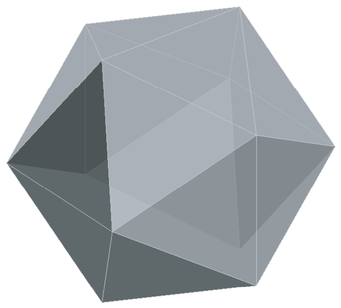
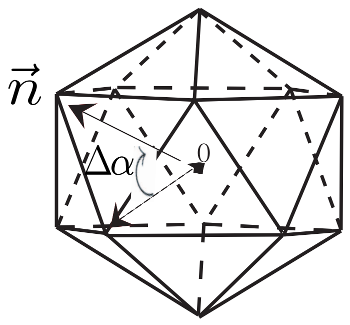


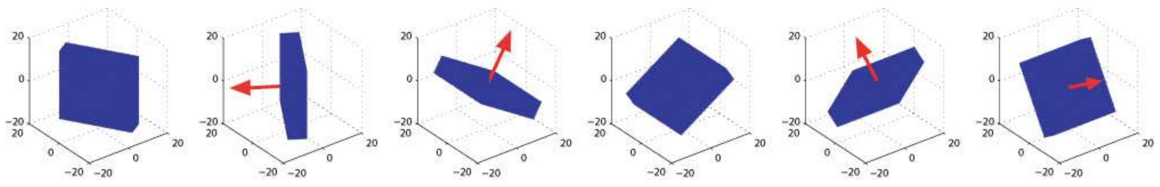
Fig. 3. 3D tile demonstration: D is the distance (between P and P_0) from the rotation center of the tile to the plane. θ is the angle between the tile and the plane. $\vec{n}$ is one norm vector of the rectangle tile. The other norm vector is opposite with 180°.



(a) Icosahedron sphere tessellation: refinement level 0



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



(c) The number of the tiles at each rotation center by using icosahedron sphere tessellation: 6 in total

Fig. 4. 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.

object are chosen. Considering that the actual bits required after compression by each tile are proportional to the volume of the encoded 3D tile, we model the average transmission rate to be proportional to the average number of tiles to be transmitted times the tile volume.

III. 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, Algorithm 1 is provided to determine which set of tiles answers a query with minimum rate. Second, since data is rotated before coding, Algorithm 2 is designed to map 3D original data to the corresponding 3D rotated sampling grids.

	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 I
ICO: ICOSAHEDRON TESSELLATION; OCT: OCTAHEDRON TESSELLATION; TETRA: TETRAHEDRON TESSELLATION. L MEANS REFINING LEVEL.

A. Fast 3D Rectangle Search Algorithm

Since there are many possible rectangle combinations to cover a plane, we consider a solution to be “optimal” when the number of rectangles to cover the plane is minimum. 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 plane, using the same number of rectangles). Our 3D rectangle search algorithm is similar to our previously proposed 2D rectangle search algorithm [11]. Instead of sorting the points associated with a line as in the 2D case in [11], the points on a 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 1, we start by finding the tile that covers the first element of List A (the furtherest 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 on 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 IV.) Additionally, this search algorithm is very fast, since we only need to search for the rectangles located around the rotation centers close to the plane.

Algorithm 1 3D Rectangle Cover Searching Algorithm

```
1: Find the indices for the points in the requested plane. Here the points on Cartesian grid which have distances
   less than 1 are considered. 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 rotation centers which are close to the plane, calling the list $B$.
3: while isempty($A$ == 0) do
4:     Starting with $A(1)$, the beginning point of the list $A$, find the rotation centers 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 rotation centers in $C(i)$, find the one covers $A(1)$ and covers the
           greatest amount of $A$, calling $R(i)$.
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
```

B. Mapping Algorithm for Rotated Tile Encoding

From Figure 5, the points on the rotated rectangular grid (RP) do not coincide with the original Cartesian grid points (CP_O, CP_I). A straightforward approach to represent the data would be to interpolate the values on the rotated rectangular grid before compression, but interpolation results in loss of some high frequency information. Instead we propose to map the original values on the Cartesian grid into the rotated rectangular grid before compression. While many alternative mappings are possible, we are interested in methods that minimize the mapping distance (D in Figure 5), where 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 tile outward. Refer to Algorithm 2 for the implementation details. In Figure 5, 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 show how pixels are remapped.

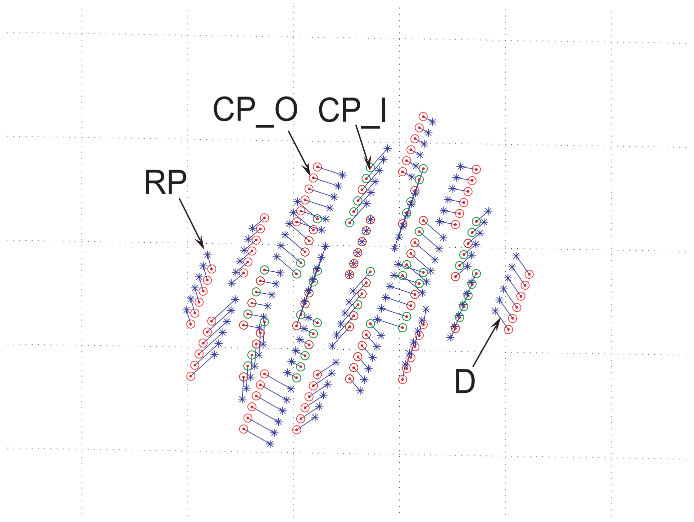


Fig. 5. 3D Mapping: CP_O (cartesian points outside the rectangle), CP_I (cartesian points inside the rectangle), RP (point on rectangle grid), D (mapping points distance).

The main advantage of this approach is that it lowers the encoding/decoding complexity, but at the cost of introducing some distortion (i.e., the pixel values that are aligned for encoding were not aligned in the original representation). In our 3D case study, 3D compression has not been applied yet, however experimental results using compression in the 2D case with the same mapping algorithm [11] demonstrate that the symmetric mapping algorithm leads to better RD performance than a non-symmetric approach and both the mapping algorithms lead better RD performance than the interpolation method. A situation analogous to that observed in compression of Bayer filter images [12] 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. 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.

Algorithm 2 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
```

IV. Experimental Results

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. We report preliminary results for our 3D case study, where 3D compression has not been applied yet to the tiles. Considering that the actual transmission rate or storage in bits is proportional to the volume of encoded 3D tiles, as a preliminary result, we count the number of retrieved voxels. The average numbers of transmitted voxels for both schemes are recorded. The rectangular tile size is fixed to be $W = 32$, $L = 32$ and $H = 8$. The side of a cubic tile is 20. 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 6 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 II. The rotation center

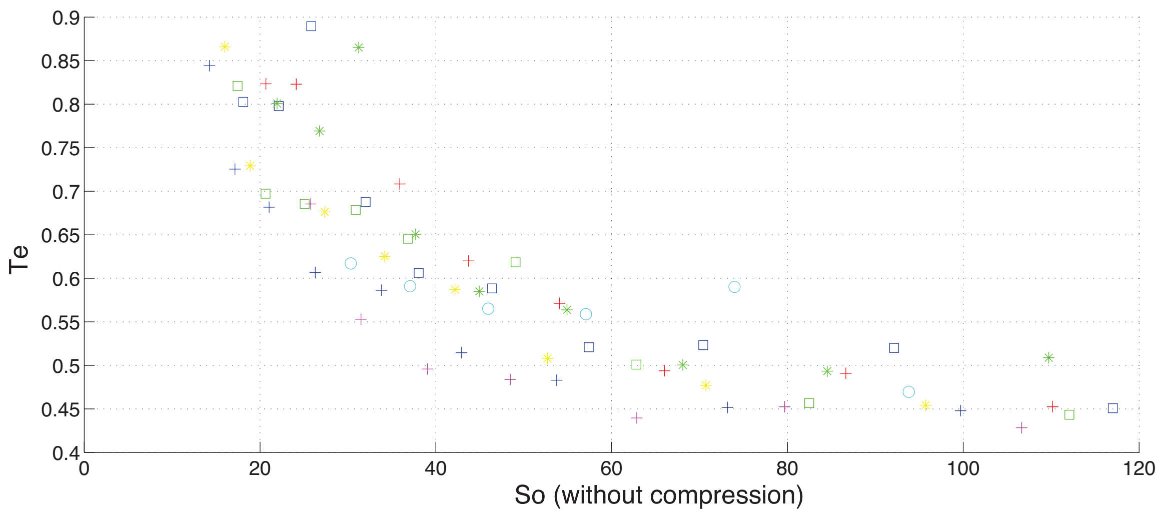


Fig. 6. 3D case result: T_e versus S_o. T_e denotes the ratio of required bandwidths for the rectangular and cubic tiling schemes. S_o denotes the relative storage overhead required by the rectangular tiling scheme. $T_e = T_r/T_s$ $S_o = S_r/S_s$. Note, the storage here is before 3D compression.

positions vary by changing the rotation center distances (D_x, D_y and D_z in Section II). The proposed method 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. Intuitively, the proposed method uses an over-complete set (overlapped rectangular tiles) to represent the 3D data set, while the traditional method uses 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 6 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.

V. Conclusion

In this paper, we proposed a new method for remotely retrieving oblique plane from 3D volume data set with fast random access. This paper has focused on the 3D case, but the methodology may be extended to higher dimensions. Our 3D results have demonstrated that by using overlapped tiles with different orientations and allowing some storage overhead on the server's side, transmission rate can be reduced by close to 55% in 3D case, as compared to the conventional tiling scheme. More reduction is possible by allowing more storage overhead and alternative configurations for the redundant tiles can also be investigated. 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 developed a mapping algorithm, which has low complexity, preserves the Cartesian grid data and allows reconstruction very easily and efficiently.

References

[1] H. Lalgudi, M. Marcellin, A. Bilgin, and M. Nadar, “Lifting-based view compensated compression of volume rendered images for efficient remote visualization,” in *DCC-Proc*, 2008.

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

[3] 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*, pp. 243–250, Jun. 1996.

[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] S. Muraki, “Volume data and wavelet transforms,” in *IEEE Trans. Computer Graphics and Application*, July 1993.

[6] Y. Cho and W. A. Pearlman, “Hierarchical dynamic range coding of wavelet subbands for fast and efficient image compression,” *IEEE Trans. Image Processing*, vol. 16, no. 2005-2015, Aug 2007.

[7] “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.

[8] 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.

[9] 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.*, vol. 6512, 2007.

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

[11] 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.

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