Fast Cross-sectional Display of Large Data Sets

V.J. Dercksen
Zuse Institute Berlin
dercksen@zib.de

S. Prohaska
Zuse Institute Berlin
prohaska@zib.de

H.-C. Hege
Zuse Institute Berlin
hege@zib.de

Abstract

One of the steps to reconstruct the 3-D geometry of biological objects from a stack of 2-D microscopic images, is to align the individual slices with respect to each other. Due to complex internal structures and imaging artifacts, automatic methods do not always lead to reliable results. Visual feedback consisting of sagittal and coronal cross-sections is very useful to validate the alignment result. For data sets which are too large to fit into main memory, this is an expensive operation as each slice in the volume needs to be referenced, resulting in a lot of disk I/O. Array data (including images) is conventionally stored in row- or column-major order. This is however not always optimal. In our application, we chose to store the data in Z-order (Morton order), resulting in a significantly improved performance.

1 Introduction

A topic that currently receives much interest in biology and medicine is the creation of standard atlases [2, 8]. The purpose of such atlases is to serve as a geometrical reference model into which different kinds of experimental data can be integrated. The results of our work emerged in a biologically oriented research project, which aims to create a 4-D standard atlas of plant seeds. As a first step, a 3-D surface representation of the grain is to be created from microscopic images taken from physical sections by performing a number of image processing steps.

An essential step is the alignment of the section images. In our alignment application this problem is treated as a series of pair-wise registrations (see Fig. 1). Starting at the bottom of the stack, two neighboring slices are aligned at a time, by running an automatic registration method or interactively. The user works towards the end of the stack, always aligning the upper slice with the respect to the lower one. Currently only rigid transformations are allowed. Due to complex physiological structures and image imperfections, automatic alignment methods do not always lead to satisfying results. Therefore, it is imperative that the user is offered the opportunity to visually inspect the result of such automatic methods and that she can control the alignment process interactively. The application provides two windows for viewing the current alignment status in planes parallel to the *xz*- and the *yz*-planes. Experts find these orthogonal views a very important feature for validating the global alignment status and results. Alignment errors easily propagate and deteriorate through the volume. Therefore, much attention should be paid to the accuracy. The user should be able to zoom in and look at the image data at its full resolution.

Common data sets in the project consist of thousands of high-resolution color images, leading to a total data set size of roughly 10-30 GB. Such data sets do not fit into the main memory of current desktop computers. In order to be able to perform the alignment, the application only loads the two slices that are currently being aligned into memory. For slices which have been aligned, only the transformation parameters remain resident; the image data is removed from memory. This way, the whole volume can be processed without the need to have the complete data set in main memory.

However, in order to display the orthogonal views, we need to extract a line from each slice in the volume. This line has an arbitrary position and orientation, due to the alignment transformation. This makes updating these windows an expensive operation as lots of disk I/O may be involved. The storage order for the data is an important parameter for the performance. This paper shows that for our application storing data on disk in Z-order (Morton order) significantly improves the response time in comparison to conventional row-major layout.

2 Method

External memory algorithms and data structures [9] address the problem of data exceeding the size of available main memory. Out-of-core data storage on disk is required, which uses block-wise I/O. The general goal is to redesign the algorithms to run with minimal performance loss due to this non-uniform cost for memory accesses. The first step is to understand the data access patterns. Then, when possible, the algorithm should be redesigned to maximize data access locality, and to devise a data storage layout consistent with the access pattern, thus amortizing the cost of individual I/O operations over several memory access operations.

2.1 Storing data as a space-filling curve

The most common way to store data in a two-dimensional array of size N×M is in lexicographic order, i.e. either row-major or column-major. The following mappings are used to obtain the memory offset of element *(i,j)* with respect to the start of the array, for respectively row-major and column-major layout:

$$M_{rm}(i,j) = N \cdot i + j \quad M_{cm}(i,j) = i + M \cdot j$$

Accessing a row-major ordered array in a row-major order requires an optimal low number of disk block I/Os. However, data access in column-major order uses a worst number of I/Os (vice versa for column-major layout).

Storing data in Z-order (also known as Morton order) [5, 7] offers a compromise between these extremes. It is not biased towards a row-major or column-major access pat-

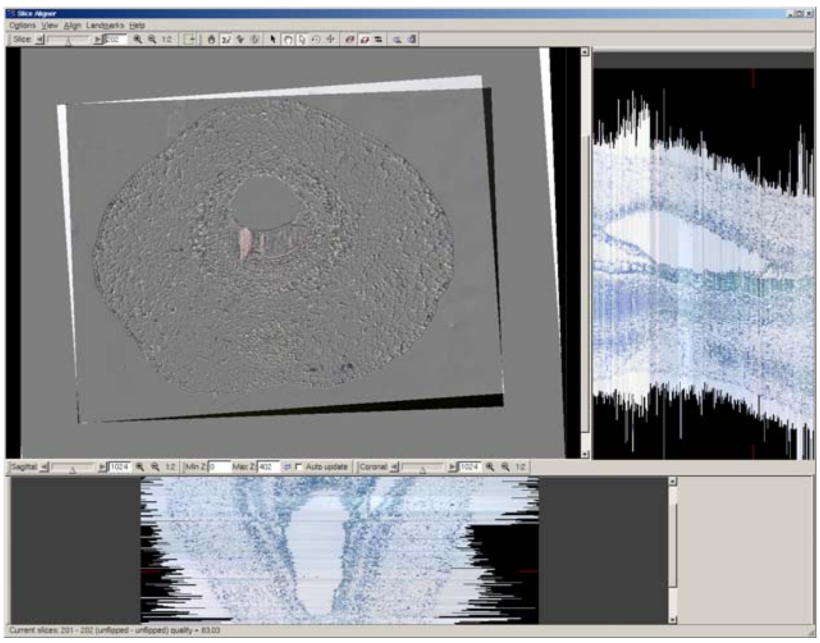


Fig. 1. Alignment application showing two slices currently being aligned (top left), sagittal (right) and coronal (bottom) cross-sections.

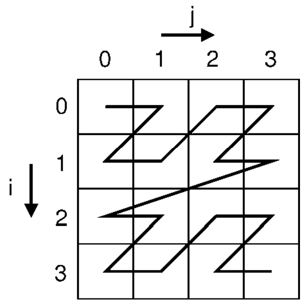


Fig. 2. Ordering of the elements of a 4×4 matrix in a Z-curve.

tern and it offers a great deal of spatial locality. The latter enables algorithms to avoid page faults independently of the actual page size. This *cache-oblivious* data layout [4] optimizes block transfers at all levels in the memory hierarchy at the same time.

A Z-curve is member of the class of space-filling curves, i.e. functions mapping a multi-dimensional array to a one-dimensional array [1, 3]. For the two-dimensional case this mapping has the signature:

$$P : N \times N \to \{1, \ldots, N^2\}$$

The index of point *(i,j)* of a two-dimensional array stored in Z-order can be calculated as follows. Let the bit-representations of i and j be $i = i_k...i_3i_2i_1i_0$ and $j = j_k...j_3j_2j_1j_0$. Then the bit-representation of the Z-index $Z(i,j)$ corresponds to the alternating bits of i and j:

$$Z(i, j) = i_k j_k \ldots i_3 j_3 i_2 j_2 i_1 j_1 i_0 j_0$$

The calculation of the Z-index is relatively expensive as it involves bit operations which are not implemented in hardware in today's CPUs. These costs can however be avoided when dilated integer arithmetic is used. We refer to [10] for a detailed discussion.

It may appear that Morton indexing wastes space when used for large arrays that are not square and do not have dimensions that are a power of two, because in those cases the space needs to be padded. This waste however only involves address space. In hierarchical memory only its margin will ever be swapped into cache, so only very little physical memory space is lost [10].

2.2 Approach

Our approach is characterized as follows. The data is stored in 2D Z-ordered slices. We do not use 3D Z-ordering because the alignment transformation of each slice is independent. Therefore, spatial locality in the z direction cannot be exploited.

By using the *mmap* system call (only available on Unix, *MapViewOfFile* is the equivalent Windows function) we leave all memory management to the operating system. All slices are '*mmap*ped' to a virtual address range which can be indexed through a pointer. This virtual address range is usually much larger than the available physical memory. The operating system performs the necessary paging and disk I/O to transparently provide the data at addresses accessed by the application. The virtual address range is limited by the number of bits per address. A 64-bit operating system is required if the data size exceeds approximately 1.5 GB. The whole geometry reconstruction pipeline is implemented in the visualization and data analysis software Amira [6].

3 Results

Two experiments were performed in order to measure the impact of Z-ordering the data on the update rates in comparison to row-major layout. In practice, a common scenario is to slice through the data in the sagittal (parallel to *yz*-plane) or coronal (*xz*) direction in order to confirm that the alignment result is satisfactory across the whole volume. This is simulated in the experiments. Starting with the Z-ordered stack of slices, the *yz*-plane is moved through the volume with one voxel stepsize, starting at $x=0$. This procedure is repeated for the coronal direction and with the data stored in row-major layout. In the second experiment, the same procedure was followed, but with randomly transformed data. Each slice was rotated around the image center with an arbitrary angle and subsequently translated with a random vector (T_x,T_y) where $T_x, T_y \in [-256, 256]$.

The data set used for the experiments consisted of a stack of 600 color (RGBA) images of size 2048×2048, yielding a total size of 9.6 GB. The experiments were performed on a Dual AMD Opteron 2 GHz 64-bit computer running Linux 2.4.21-211-smp. One gigabyte of physical memory was assigned to the process of which 65 MB was used for the application. The operating system's page size is 4 kB.

The result of the first experiment for the first 512 steps is shown in Fig. 3 (top). We see from Table 1 that the Z-ordered layout is approximately as fast as the optimal case (row-major, coronal), but performs much better than the worst case (row-major, sagittal) for untransformed data. Given 4 kB page size and the data set dimensions mentioned above, one page in row-major ordering corresponds to a 1024×1 subvolume (half an image line). Thus, for a sagittal cross-section, for each pixel to be displayed, one entire page needs to be retrieved, which means that half the volume needs to be loaded, although only 1/1024 is used. The amount of data exceeds the size of the main memory, resulting in a lot of page swapping (thrashing). This explains the long update times. Whenever a coronal cross-section needs to be displayed however, only a minimum number of pages needs to be loaded as all pixels from a loaded page are used. When the physical memory has been filled after approximately 200 slices, pages need to be swapped out, leading to increased update times. This also holds for the Z-ordered slices. With this layout, one page corresponds to a 32×32 subvolume. Thus, in order to display one line, 64 pages need to be loaded. For a whole cross-sectional slice this amounts to 64×600×4 kB ≈ 150 MB. As the memory is large enough so that those pages are not swapped out during the current screen update, the data required to display the next 31 slices is directly available. After this new disk I/O follows. This explains the peaks at 32-slice intervals. For the randomly transformed data, these effects disappear as the longer load times are spread out over all slices. In this case storing data in Z-order results in much faster update times.

4 Summary and Outlook

The experiments showed that a different data storage order makes a significant difference in the overall update rates of the orthogonal cross-sections. As the Z-order layout is approximately as fast as the best access pattern of the row-major storage, it offers a huge advantage over the worst case. In practice it proves however still too slow to provide interactive update rates. Current work focuses on a hierarchical storage and display scheme. This way, a subsampled version of the data can be retrieved and displayed very quickly. The resolution is gradually increased as more data is loaded. This way the user can still view the full-resolution cross-sections, without unnecessarily interrupting the workflow.

Acknowledgements

Special thanks go to Dr. S. Gubatz, Institute of Plant Genetics and Crop Plant Research, Gatersleben, Germany, for providing the data set.

Table 1. Average update time (msec.) per pixel. Average update time for one cross-section of the untransformed data is given in parenthesis.

	Untransformed data		Transformed data	
	sagittal	coronal	sagittal	coronal
Z-indexed	0.00160 (1.97s)	0.00109 (1.33s)	0.00606	0.00607
Row-major	0.213 (262s)	0.00161 (1.97s)	0.327	0.317

References

[1] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer, “Space-filling curves and their use in the design of geometric data structures,” *Theoretical Computer Science*, vol. 181, no. 1, pp. 3–15, 1997.

[2] A. Burger, R. A. Baldock, Y. Yang, A. Waterhouse, D. Houghton, N. Burton, and D. Davidson, “The Edinburgh mouse atlas and gene-expression database: A spatio-temporal database for biological research,” in *Proceedings of the 14th International Conference on Scientific and Statistical Database Management.* IEEE Computer Society, 2002, p. 239.

[3] S. Chatterjee, V. V. Jain, A. R. Lebeck, S. Mundhra, and M. Thottethodi, “Nonlinear array layouts for hierarchical memory systems,” in *International Conference on Supercomputing*, 1999, pp. 444–453.

[4] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, “Cache-oblivious algorithms (extended abstract),” *in Proc. Symp. Found. Comp. Sci. IEEE*, 1999, pp. 285–397.

[5] G. Morton, *A computer oriented geodetic data base and a new technique in file sequencing*. Ottawa, Ontario, IBM Ltd., 1966.

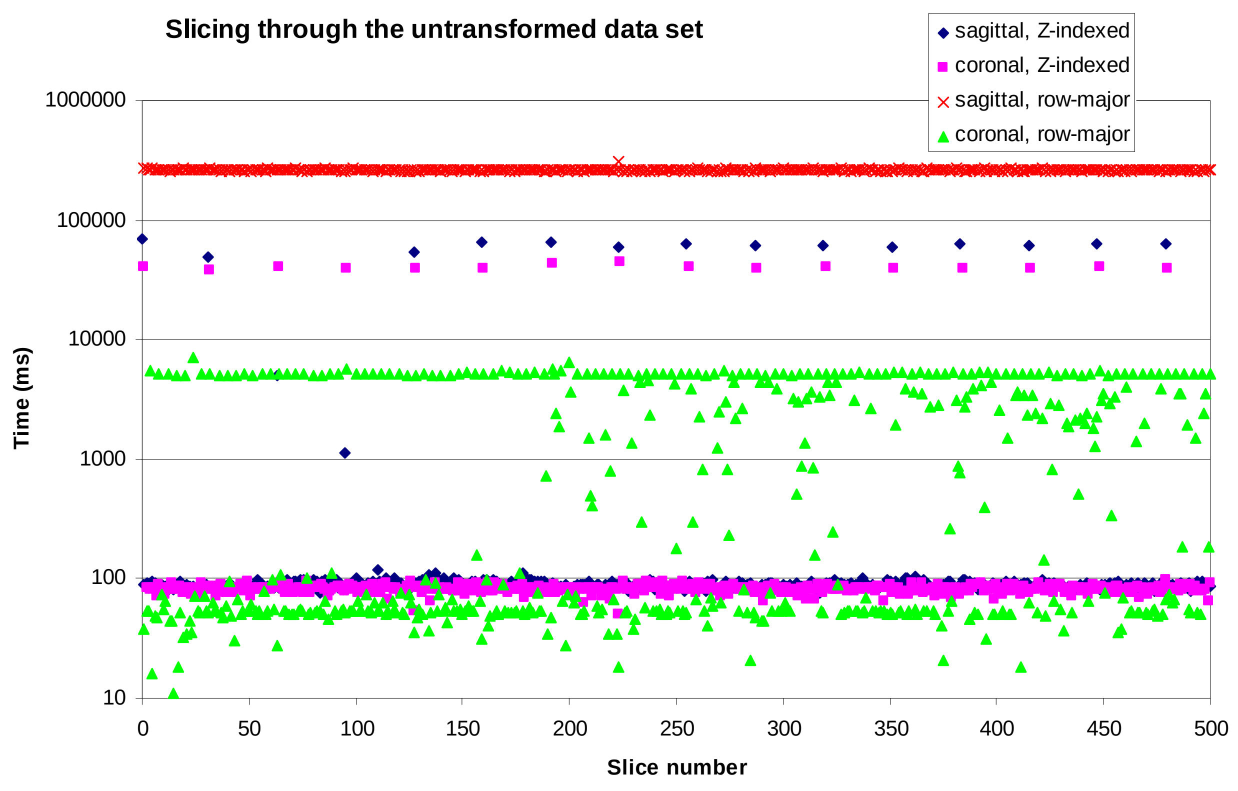
[6] D. Stalling, M. Westerhoff, and H.-C. Hege, “Amira: A Highly Interactive System for Visual Data Analysis,” 2004, in: C.D. Hansen and C.R. Johnson (eds.), The Visualization Handbook, Chapter 38, pp. 749–767, Elsevier, 2005.

[7] J. Thiyagalingam and P. Kelly, “Is morton layout competitive for large two-dimensional arrays,” Euro-Par, pp. 280–288, 2002.

[8] A. Toga and P. Thompson, “Maps of the brain,” *The Anatomical Record*, vol. 265, no. 2, pp. 37–53, 2001.

[9] J. S. Vitter, “External memory algorithms and data structures: Dealing with massive data,” *ACM Computing Surveys*, vol. 33, no. 2, pp. 209–271, 2001.

[10] D. S. Wise, “Ahnentafel indexing into morton-ordered arrays, or matrix locality for free,” in *Proceedings from the 6th International Euro-Par Conference on Parallel Processing*. Springer-Verlag, 2000, pp. 774–783



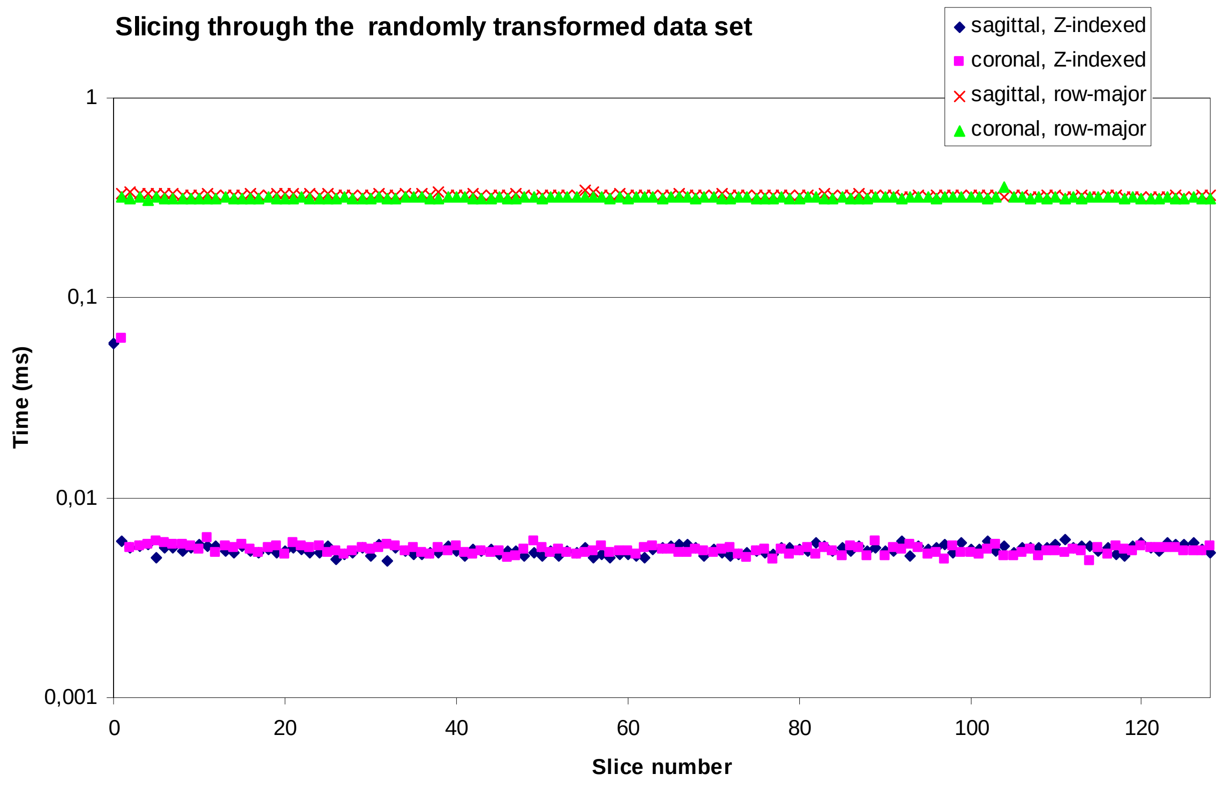


Fig. 3. The cross-sectional planes are moved along the *x*-axis (sagittal view) and the *y*-axis (coronal view). The time required to retrieve the data to update the orthogonal views is measured for Z-ordered data and for data stored in row-major order. The experiments were performed for both untransformed data (top) and randomly transformed data (bottom).