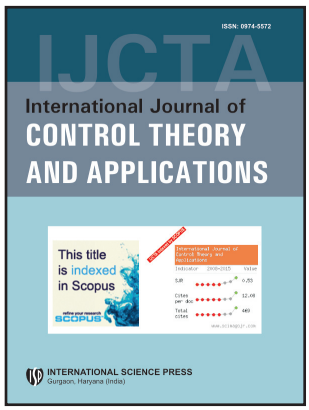

International Journal of Control Theory and Applications

ISSN : 0974-5572

Volume 9 • Number 44 • 2016

An Optimized Hybrid Data Structure For Sparse Volume Data

Rahul V. Cheeran[a] and Joby George[b]

[a-b]Department of Computer Science and Engineering, M.A College of Engineering, Kothamangalam, Kerala, India. Email: [a]rahulcheeran@gmail.com; [b]jobygeo@hotmail.com

Abstract: Processing of huge sparse volume dataset creates large overheads and computational complexities. Use of typical dense data structures for sparse dataset is an inefficient method. So we have to use dedicated sparse data structure for efficient representation of sparse dataset. The choice of the sparse data structure is based on criteria's such as sparsity of the dataset and memory availability. In case of lager sparse volume dataset the sparsity is not evenly distributed across the dataset. Since the sparsity is not evenly distributed we propose a hybrid data structure which adapts according to the local sparsity of the dataset. A hybrid data structure consist of several elemental data structure combined in to a single optimal structure. For data regions with very low sparsity Dense List representation are the most efficient. For data regions with medium sparsity Hashing can give better efficiency compared to other structures. In case of region with very high sparsity Coordinate List are the most efficient structures. GPU's have very limited memory resource which limits the user from loading large volume dataset into it. In such case this kind of memory efficient hybrid sparse data structure plays an important role.

Keywords: Hybrid Data Structure, Sparse Volume Dataset, Layering Approach.

1. INTRODUCTION

Volume dataset is a kind of three dimensional data set commonly represented as a regular grid of scalar or vector values. Most of such real world volume data sets are sparse in nature that is, most of the values in the dataset are null or zero values. Such a kind of data set is typically known as 'Sparse Volume Dataset'. Each of such dataset is characterized by its sparsity measure. Sparsity is the measure of percentage of zero values in the entire dataset. Usually the sparsity of the dataset is not evenly distributed across the data set. In case of most data set, sparsity may vary locally in between the data set.

Use of the common dense data structure to represent a sparse volume data set is not an efficient method. We have to make use of the dedicated sparse data structure's to represent such data sets. The main objective of such sparse data structures is to make the representation more memory efficient by eluding the storage of zero values in the dataset. Each of such individual sparse data structure has its own gains and drawbacks. We cannot make a single elemental global optimal data structure for any kind of sparse dataset. If we are using a Coordinate List representation to store a dataset with very high sparsity then it will be memory efficient. On the other hand if we

are using the same data structure for representing a dataset with very low sparsity, it will seriously degrade the memory efficiency. Typically the programmer has to pick an optimal data structure for each dataset considering the sparsity characteristics of the dataset. Choosing an optimal data structure form a set of candidate structures is not an easy task. The decision has to be taken based on several factors. The most difficult part is to find an optimal balance between memory efficiency and access performance.

2. RELATED WORK

The book 'Spatial Data Structures' which is written by Samet [2] gives a summary about different types of spatial data structures. In this work he differentiates the data structures for point data, collection 2D areas (rectangles), curvilinear data and volume data. In case of point data it always has static number of key which can be the coordinate value. Rectangle area data stores the bounding rectangles or else the bounding boxes. Curvilinear stores the boundary values of the object.

The list and grid are examples of non-hierarchical data structures that can be used store data set in non-hierarchical order. Retrieval process of a coordinate from the list of items involves O (n) executions, where n is the total number of elements in the list. The fixed-grid method divides the spatial domain into same sized grids. Each cell can either have a single value or a linked list of entries. For our structure, we are only concerned about a single value per cell. The fixed-grid method works well only if the data is evenly spread across the data set. Including the binary search mechanism in list or grid method can improve the retrieval method.

Octrees are the extension of Quadtrees to three dimensional spaces. The common property of quadtrees is that they recursively splits two dimensional space into four quadrants. Likewise the Octree splits the three dimensional space in to division of eight quadrant. This features are explained in [9] by Finkel.

Different types of Octrees are discussed in the survey by Knoll [11]. Besides the pointer octree, he explains octree hashing and discusses the full octree, and linear Octree. The full octree stores all of its leaves and if needed also all the internal nodes, therefore no pointers have to be stored, such an octree is also called pointer-less octree.

Bucket techniques were at first established to hide access latency to external storage. It divides the region of interest into buckets; this divided content can be loaded from disk on demand. Each bucket can hold a specified number of values in it; this measure is called bucket size. Bucket methods can be divided into non-hierarchical and hierarchical methods. The most simple non-hierarchical bucket method is the fixed grid method, which can store multiple entries per grid cell. A problem of this scheme is underflow and overflow as stated by Samet [2]. Many cells may remain empty; others can contain a large number of values.

The volume is divided into bricks of the equal size. An empty brick does not required to be stored. Hence the approach decreases the memory requirement. In addition, bricking results in a better memory locality when caching nearby values.

Alternative non-hierarchical techniques are linear and spiral hashing. In recent times hashing was introduced on the GPU as perfect spatial hashing [4]. The benefit of this methodology is the fast access. The generation of an appropriate hash function and offset table can be computationally challenging.

3. PROPOSED WORK

A. Hybrid Data Structure Design

A data set normally comprises of many sub-regions with dissimilar sparsity features. Each of these individual regions can be optimally encoded by one of the elemental data representation. The objective of our data structure

is to adapt according to the local sparsity of the sparse volume data-set. The basic technique is to store merely the non-empty values and to elude storing zero values in the data set.The best way to represent very sparse sub-regions are by using coordinate list structure; medium sparse sub-regions afford themselves best to hash tables, and the dense regions are most efficiently stored in the form of dense representation.

To split data into sub-regions we make use of layering approach. The advantage of layering, as a non-hierarchical method, is that it gives high performance access possible unlikely other subdivision methods. The main advantage of layering is that we are able to choose a memory efficient data structure for each of the individual layer in the data set.

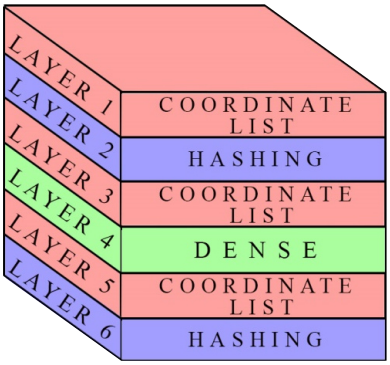

Figure 1: Layering approach applied on Volume Data Set

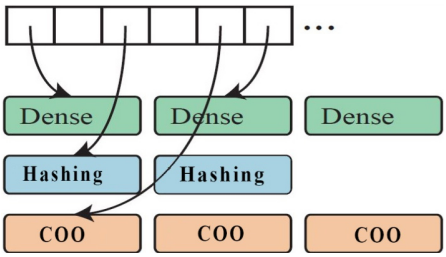

Figure 2: Hybrid Data Structure Layout

B. Elemental Data Structures

For the proposed hybrid data structure we chose a group of four different individual representations such as empty, coordinate list, hashing and dense structures. Each individual representation performs best for different level of sparsity in the dataset.

1. *Empty Volume:* Empty bricks are simply omitted when the data is loaded into the hybrid structure. Layers which represent such empty region hold a null pointer which indicates that it is an empty region. Such data are excluded from the hybrid structure.
2. *Dense Volume:* Dense volume bricks, colored green in Figure 3.3, are a good representation for regions of high density. They store all data values in one array. Empty values are also stored in this array. In our case a one dimensional array is used. Each three-dimensional spatial location is transformed to a one dimensional array position using the row-major order. In contrast to octree and voxel list, no additional information has to be stored per value to access the data structure. The dense volume representation performs very well for dense regions and fails for extremely sparse regions.

3. *Coordinate List:* COO stores a list of tuples. Tuples have the coordinate value *x* & *y* then the data value *v*. Ideally the entries are sorted to improve random access times. The sorting can be done in row major or column major order. This is another format which is good for incremental matrix construction. Coordinate lists are memory efficient for immensely sparse volumes. A major draw-back of coordinate lists is that access to the data requires O (*n*) steps, where n is the number of elements in the list. It is possible to de-crease the look-up time in a sorted coordinate list build using linked list structure.

4. *Hashing:* A hash function can be used to map data of random size to data of static size. The values given back by a hash function are called hash values, hash codes, hash sums, or simply hashes. One use is a data structure called a hash table, widely used in computer software for rapid data lookup. Hash table data structures can be used to store sparse data set in a memory efficient manner.

C. Layering Approach and Retrieval

The root of the data structure contains the list of layers obtained in the initial phase. Each individual node in the list has two fields, type and pointer. The type field can hold the elemental node types information. The pointer field is used to point into the location where data is stored.

The retrieval of data in the data set starts with an address translation which maps layers into corresponding node and then retrieves the type and pointer of the node. If the pointer is null then we can conclude that it is an empty region and retrieval for such empty nodes can be discarded at this point.

4. CONCLUSION

Through this work I introduced an optimised hybrid sparse data structure, which was construed from several individual data structure. I believe that such a kind of hybrid data structure paves wave for use of more such data structure, which is useful for different purpose. Our hybrid sparse data structure integrates several elemental data structure into a single efficient structure. The disadvantages in usage of single elemental sparse data structure will be leveraged by this Hybrid Structure. By making use of the proposed data structure we can efficiently manage the sparse volume data in the memory.

5. DATA SET

Figure 3: Present: An industrial CT scan of a Christmas present

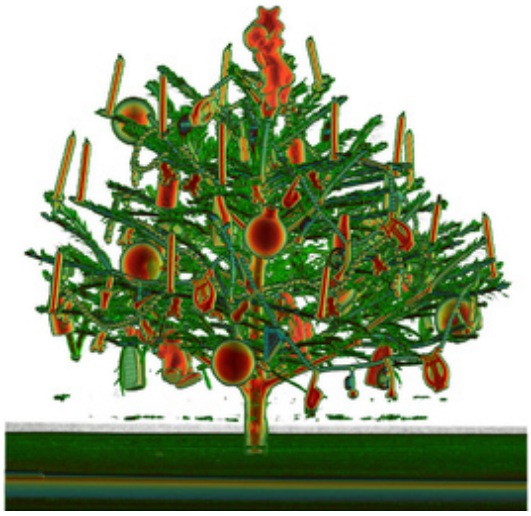

Figure 4: Christmas Tree: The model was scanned at the General Hospital in Vienna

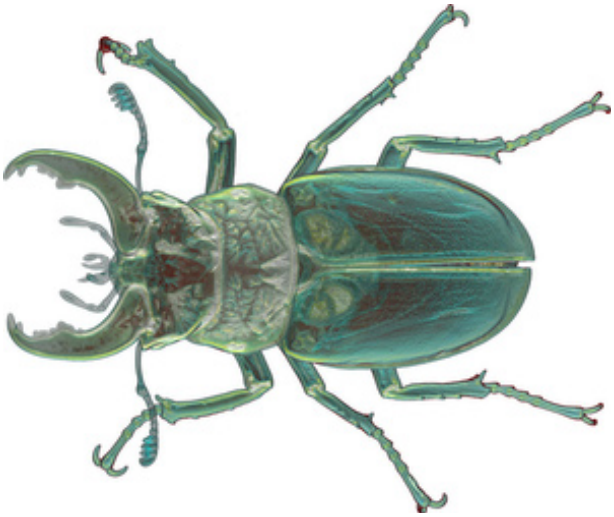

Figure 5: Stag Beetle: The model was scanned at Vienna University of Applied Arts, Austria

REFERENCES

[1] Matthias Labschutz, Stefan Bruckner, M. Eduard Groller, Markus Hadwiger, and Peter Rautek (2016) 'Jittree: A Just-in-Time Compiled Sparse GPU Volume Data Structure' IEEE Transactions on Visualization and Computer Graphics, Vol. 22.

[2] Hanan Samet, (1990) The Design and Analysis of Spatial Data Structures. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[3] Volker Gaede and Oliver Günther. (1998) Multidimensional access methods. *ACM Computing Surveys (CSUR)*, 30(2):170–231

[4] Sylvain Lefebvre and Hugues Hoppe. (2006) Perfect spatial hashing. *ACM SIGGRAPH*, pages 579–588.

[5] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. (2009) Giga voxels: Ray guided streaming for efficient and detailed voxel rendering. In Proceedings on Interactive 3D Graphics and Games, I3D '09, pages 15–22.

[6] Tim Foley and Jeremy Sugerman. (2005) Kd-tree acceleration structures for a GPU raytracer. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pages 15–22.

[7] Daniel Reiter Horn, Jeremy Sugerman, Mike Houston, and Pat Hanrahan. (2007) Interactive kd tree GPU raytracing. In Proceedings of symposium on Interactive 3D graphics and games, pages 167–174.

[8] Nadathur Satish, Mark Harris, and Michael Garland. (2009) Designing efficient sorting algorithms for manycore GPUs. In Proceedings of IEEE International Symposium on Parallel & Distributed Processing, IPDPS '09, pages 1–10.

[9] Raphael A. Finkel and Jon Louis Bentley. (1974) Quad trees a data structure for retrieval on composite keys. Acta informatica, 4(1):1–9.

[10] Jane Wilhelms and Allen Van Gelder. (1992) Octrees for faster isosurface generation. ACM Transactions on Graphics (TOG), 11(3):201–227.

[11] Aaron Knoll. (2006) A survey of octree volume rendering methods. Scientific Computing and Imaging Institute, University of Utah.