A Novel Page-Based Data Structure for Interactive Walkthroughs

Behzad Sajadi* Yan Huang* Pablo Diaz-Gutierrez* Sung-Eui Yoon+ M. Gopi*
*University of California, Irvine
+KAIST

Abstract

Given a data layout of a large walkthrough scene, we present a novel and simple spatial hierarchy on the disk-pages of the layout that has notable advantages over a conventional spatial hierarchy on the scene geometry. Assume that each disk-page consists of a set of triangles whose bounding boxes are computed. A spatial hierarchy of the walkthrough space is constructed, not with the given scene, but with the bounding boxes of disk-pages. The leaf nodes of the spatial-hierarchy refer directly to the page numbers of the pages of the bounding box it contains. We call this hierarchy on the pages as the disk-page hierarchy. We also propose a self-contained disk-page format that would suit this data structure well. Further, we present a new cache-oblivious graph-based data layout algorithm called the 2-factor layout that would preserve the proximity and orientation properties of the primitives in the layout. Walkthrough experiments have been conducted on a city scene consisting of over 110M triangles. Our system renders this scene on a laptop within a one pixel projection error at over 20 fps with simple texture substitution based simplification of distant objects, and with no explicit data/cache management.

Keywords: walkthrough systems, data layouts, spatial data structures, out-of-core algorithms

1 Introduction

In large scale walkthrough systems, much of the recent works mainly focus on the transfer bottleneck of massive data from hard-disk to the main memory. In this context, many out-of-core rendering methods, and a few algorithms for data layout on disks for efficient access have been proposed. The out-of-core rendering systems, in addition to their implementation of sophisticated techniques for culling, simplification, GPU-based rendering, etc., have to work on cache management of data for interactive out-of-core processing of massive data. The data layout methods assume a specific access pattern of the application and compute the layout of the data that would most likely suit the application. But most of the current rendering systems do not usually take data layout on the disk into account. For example, assume that the data layout scheme expects three blocks of data, say A, B, and C to be requested together by the renderer and hence has laid A, B, C, in that order, on the disk. But the renderer, although requires A, B, and C at the same time, oblivious of the actual layout of these data, requests in the order C, B, A. This increases the seek time of these blocks from the disk. Thus it is obvious that this cycle of symbiotic relationship between the rendering system and the data layout is not complete until the rendering system fine tunes its data access pattern based on the data layouts.

In this paper, we present a very simple data structure on any given (good) layout called the *disk-page hierarchies* which can be used directly by the rendering system. In other words, first we compute the data layout based on the application needs, then we compute the data structure used by the application on these layouts in order for the application to make the best use of the layout. There are two distinct goals of this data structure: to reduce the seek/transfer time of data from the disk, and to reduce the data management time of the operating system. Further, we also believe that this data structure will make the design of any walkthrough system extremely simple and elegant.

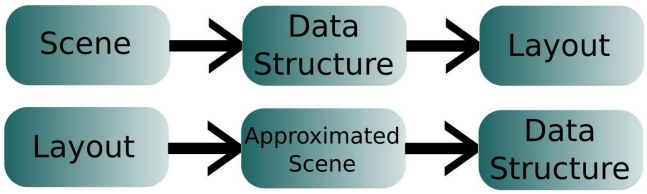

Figure 1: *Comparison of the workflow in traditional approach (top) and our approach (bottom)*

In essence, disk-page hierarchy can be built as follows: Let us assume that the primitives (triangles) of the walkthrough scene are clustered in terms of their spatial proximity, and they are laid linearly on the disk in multiple pages. Each page consists of a set of (related) triangles and hence information like bounding box can be computed for each page. A disk page K-d tree, for example, is a K-d tree on these bounding boxes of the disk pages. The leaf nodes of the K-d tree will have pointers to the disk-pages, and not individual triangles. So this data structure is not strictly on the original scene, but on the bounding boxes of the primitives contained in the disk-pages. A simple out-of-core walkthrough rendering system can be build easily around this disk-page K-d tree structure. This system will request multiple disk-pages worth of primitives, rather than any other grouping that may not have any relationship to how the primitives are stored on disk. Note that any data structure (e.g. an octree) that is used by the application can be converted to a disk-page hierarchy, without restrictions.

Since the requests are in the form of disk-page numbers, they can be ordered to achieve sequential access of the data, thus minimizing the data seek and transfer time. Second, since all the data in each page is requested (in other words, requests start and end in page boundaries), the operating system can copy the entire page to the data buffer requested by the application. Eliminating partial-page data transfers brings down the data/cache management cost of the operating system to the minimum.

Conceptually, the application dictates the spatial hierarchy of the scene, which is used by the disk-layout algorithms to find *layouts of the spatial data structures* used by the application. On the contrary, the proposed disk-page hierarchies are *spatial data structures on the layout* that is used by the application. This simple change in notion results in tremendous simplicity in the implementation of a walkthrough system and relieves the user from complex cache management and other system issues.

Following are the contributions of this paper:

- a novel data structure that uses disk-pages as primitives.
- a new disk-layout scheme called 2-factor layouts.
- a simple renderer that does not use very sophisticated simplification/primitive-reduction techniques or any cache management technique but just uses the disk-page hierarchies to achieve an interactive walkthrough performance of over 20 fps on a data set that has over 110 million triangles on a laptop.

2 Related Work

2.1 Layouts

Computing specific layouts for efficient rendering and processing has received considerable attention in computer graphics in recent years.

Rendering sequences: In the context of rendering, Deering [Deering 1995] pioneered the computation and use of layouts of triangles and vertices, called *rendering sequences or triangle strips*. Hoppe [1999] cast the computation of rendering sequences as a discrete optimization with a cost function. He particularly derived the cost function for a specific vertex buffer size used in a GPU. This layout can be classified as a cache-aware layout since the layout is optimized for a particular cache parameter. Diaz-Gutierrez et al. [2005a; 2006] presented a graph based algorithm for generalized cache oblivious layout of triangles for rendering and geometry processing. Recently, Sander et al. [2007] designed a fast triangle reordering method to compute a cache-aware layout for a vertex cache size depending on a GPU.

Processing sequences: Isenburg et al. [2003] proposed processing sequences as a generalization of rendering sequences to various kinds of large-data processing. This processing sequence can be stored as an indexed mesh, *streaming mesh*. Streaming meshes are represented as interleaved triangles and vertices that can be streamed through a small buffer [Isenburg and Lindstrom 2005]. These representations are particularly useful for off-line applications, e.g., simplification and compression, that can adapt their runtime computations to a fixed ordering. We present a similar indexed mesh storage format in which vertices may be duplicated across different pages, while a triangle is represented only once.

Cache-oblivious layouts: Many algorithms use space-filling curves [Sagan 1994] to compute cache-friendly layouts of grids. These layouts have been widely used to improve the performance of image processing [Velho and de Miranda Gomes 1991] and terrain visualization [Lindstrom and Pascucci 2001; Pascucci and Frank 2001]. However, space-filling curves are mainly used for uniform grid, image, and volumes that have uniform structures. Yoon et al. [2005; 2006] proposed a generalized cache-oblivious layout of a mesh or a graph for efficient rendering and processing of massive models. In this paper, we extend the quadrilateral/tetrahedral sequencing algorithm presented by [Diaz-Gutierrez and Gopi 2005] to present a graph based cache-oblivious data layout scheme called the 2-factor layouts.

2.2 Spatial Hierarchies

Various spatial hierarchies such as k-d trees and bounding volume hierarchies have been widely used to improve the performance of various applications including rendering, culling, and collision detection. K-d trees [Bentley 1975] have been widely used in ray tracing of static models due to its simplicity of construction and its high culling efficiency for ray-objects intersection tests. Bounding volume hierarchies are also used in visibility culling and collision detection [Lin and Manocha 2003; Teschner et al. 2005].

Various top-down and bottom-up construction methods for spatial hierarchies have been designed [Walter et al. 2008]. Also, fast construction methods for k-d trees and other spatial hierarchies have been proposed [Wald and Havran 2006]. However, the construction of any spatial hierarchies involves sorting and partitioning primitives of models and thus requires high time complexity [Samet 2006], which may be prohibitive for massive models.

Culling methods: Various culling methods have been designed to improve the performance of rendering massive models [Yoon et al. 2008]. Back-face and view-frustum culling techniques are ones of most widely used culling techniques. For models with high-depth complexity, visibility culling accelerated by graphics hardware have been proposed [Bittner et al. 2004]. These techniques typically require spatial hierarchies such as k-d trees and bounding volume hierarchies.

Multi-resolution hierarchies: Various multi-resolution hierarchies have been designed to improve the performance of rendering of massive models [Luebke et al. 2002]. For rendering massive models, multi-resolution hierarchies are constructed by clustering triangles and computing a representative simplification for the triangles. Therefore, constructing multi-resolution hierarchies is very similar to constructing spatial hierarchies since these include sorting and partitioning triangles into clusters. There are a few approach to use a dual hierarchy that can serve as a spatial hierarchy and multi-resolution hierarchy [Otaduy and Lin 2003].

2.3 Massive Model Rendering

Interactive massive model rendering has been widely researched. Most of these techniques use image-based [Losasso and Hoppe 2004; Gobbetti and Marton 2005] or geometry-based multi-resolution representations [Duchaineau et al. 1997; Cignoni et al. 2004] to support interactive performance of massive models. Also, these methods design explicit out-of-core data management techniques [Varadhan and Manocha 2002] and pre-fetching mechanisms [Corrêa et al. 2003] to efficiently deal with massive models. Although these techniques can provide interactive rendering performance at runtime, these methods typically require a long pre-computation time.

2.4 Organization of the paper

In Section 3, a file format that is a sequence of self-contained disk-pages is proposed. Our disk-page hierarchy data structure, that uses such a page format, is detailed in Section 4, followed by how this data structure is used actually in a walkthrough system in Section 5. Our data structure would work on any given layout. But we can improve the performance of the system using a data layout that would help any spatial clustering data structure over it. In Section 6, we propose one such data layout algorithm called the 2-factor layout scheme. Finally in Section 7, we give details of our implementation and present our results.

3 File Format

The input file to the walkthrough system consists of a sequence of disk pages, each of constant size. The primitives in the scene are grouped together in the form of disk-pages, in a specific format

(Figure 2). Each page consists of a set of triangles and only the vertices that are referred by those triangles. Other attributes like normal vector of each vertex in the page and color of each triangle in the page are also part of the page. Since a vertex may be shared by triangles in different pages, the shared vertex and its attributes may be repeated over different pages. But a triangle appears in only one page in the entire file. The page format is shown in Figure 2.

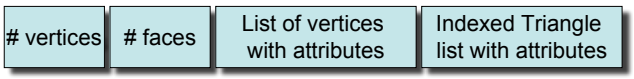

Figure 2: *Page format*

Our method can not be compared to a traditional K-d tree data structure on the triangles themselves because such a data structure might not fit into the main memory for large walkthrough models. However, it can be compared against a hierarchical data structure on the scene where only the references to the beginning of the clusters on the hard disk are kept in the main memory. The fundamental difference in our approach is in building our hierarchy where we do not use scene triangle clusters, but how they are laid out on the disk (Figure 1). Further, our page based representation has the following advantages over a hierarchical data structure over the original mesh:

- The layout and the spatial data structure are independent of each other in our approach. Therefore this approach can be combined with any existing layout algorithm.
- This approach uses fixed size pages (by changing the number of triangles in each page) which can be retrieved efficiently from the hard disk.
- Information within each page is self-contained. Every triangle in a page is well-defined without the need for data from any other page. This property can be made use of for parallel processing of disk-pages.
- Due to the locality of reference inside the pages usually one byte is enough for each vertex index.
- Compression can be applied effectively on each page since page is an atomic unit of access within which the data is accessed sequentially.

Note that, in our system, we do not use any page compression for the following two reasons. First, the model is still small compared to the typical hard-disk space available nowadays, so compression is not required for storage. Second, the potential advantage of less loading time with the compressed data might be lost in the time we have to spend decompressing this data.

This file format is dictated by the consideration to disk-pages rather than any processing sequence of the meshes. Thus this page format is conceptually different from other proposed mesh formats like streaming meshes [Isenburg and Lindstrom 2005]. Thus, the proposed page format is complementary to, and can be used along with other mesh formats like streaming meshes. This format can be used to organize any given set of (even unrelated) triangles into pages, and does not assume connectivity.

We used disk-pages of size 4KB in our implementation which is a typical page size used by many operating systems. This page size can be increased due to the high processing power of nowadays GPUs and high speed of new hard disks. However, there is a trade off between using larger page size and optimality of the bounding box of each page.

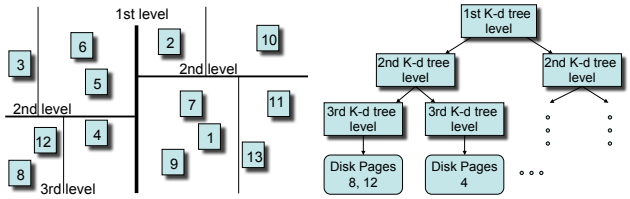

Figure 3: *An illustration of disk-page K-d tree hierarchy. It is same as the familiar K-d tree hierarchy on the walkthrough scene space, except that there are no more triangles on the scene but bounding boxes of the set of triangles in each disk-page. The leaf nodes of the K-d tree refer to these disk-page numbers.*

4 Disk Page Hierarchies

Consider a spatial data structure such as a K-d tree of a walkthrough scene consisting of a collection of triangles. A regular K-d tree of such an arbitrary collection of triangles should have at least one reference per triangle or a logical group of triangles in its leaf nodes. When we choose to render the primitives in a leaf node, we have to dereference the file locations in which these triangles or group of triangles are stored, retrieve them, process and render the required triangles. First, this K-d tree data structure may not fit in the main memory for large models. Second, the access pattern of these triangles will be based on the model or application, and not in the order it is stored in the hard-disk, which would lead to unacceptable *data access and transfer times* from disk to the main-memory. Finally, the dereferenced address may refer to the middle of a disk page. This will lead to an additional complexity in the time spent by the operating system to perform byte-level data management in order to return the data buffer containing just the requested partial data from the retrieved page. We call this the *data-management time*. Our goal is to reduce both the data access/transfer, and also management times.

Given a layout of the pages, we propose a new data structure called the disk-page K-d tree hierarchy, in which the leaf nodes do not refer to individual or group of triangles, but the disk-page numbers in which these triangles appear. This structure lets us directly organize our data in the form of pages, refer only to these pages, retrieve the entire page, and render all the triangles in these pages with minimal processing.

Computation of Disk Page Hierarchy: First, for each disk-page, the bounding box of the triangles in that page is computed. A K-d tree structure is built for the walkthrough scene that has these bounding boxes, and not individual triangles, as primitives. Given a layout of these pages, the nodes of the K-d tree will refer only to the page numbers of those pages whose bounding box it contains (or intersects). Refer to Figure 3 for an illustration of a 2-d disk-page hierarchy.

Note that bounding boxes of the disk pages might be overlapping and each bounding box might appear in multiple K-d tree nodes. However, considering each page to contain around 100 triangles number of disk pages will be considerably lower than the number of triangles. Therefore size of the K-d tree is not an issue for even large models.

Such a data structure can be considered to be a transparent interface between the application and the data in the hard-drive. Specifically, the application need not categorize the scene into objects, but just consider it as a collection of triangles. This may relieve the application from expensive pre-processing. On the storage side, the proposed disk-page hierarchy can work with any ordering of pages

(different layouts), although we will get better performance if the information in the pages is self-contained (as is in our file format) and if the data is grouped together based on proximity, which will result in fewer page references for every K-d tree node. Further, the memory required to store this disk-page K-d tree is very small since the reference is to just the disk-page numbers. Finally, since the unit of reference is disk-pages, the operating system spends less time in data management and transfer and can do efficient cache management. Since the bottleneck in a walkthrough system is not rendering, but memory access and cache management, the disk-page hierarchy directly addresses this issue by eliminating fragment accesses to disk pages and enables other improvements as described below.

5 Data Fetching Algorithm

The data that is in-core are the disk-page hierarchy, bounding box and the normal cone of the set of triangles contained in each page. In the rendering process, we use basic view-frustum culling of the disk-page K-d tree structure. The rest of the nodes have to be rendered. From these nodes, the set of page indices that needs to be rendered is collected and the duplicates (same page referenced by multiple nodes) are removed. From this resulting list, the bounding box of the every page is tested against the view-frustum for finer culling. The normal cone is tested against the viewing direction for quick page-level back-face culling. The page indices that remain are sorted for minimal seek time in the hard-disk, and the requests to fetch these pages are placed to the operating system. All triangles in these fetched pages are rendered without further processing. Notably, since the requested data start and stop at page boundaries, the overhead time required to manage the data by the operating system in terms of partial-page data transfer to the buffer of the application is completely eliminated.

The disk-page K-d tree data structure and the data fetching algorithm work with any given data layout. Clearly, their performance will be better if the layout is logically coherent and takes into account the data access pattern of the application. In other words, if the pages requested by a K-d tree node are as close as possible, then the access time can be minimized and the data transfer can be speeded up. Even from the perspective of cache management policies of the operating system, pages that are close to each other have a lower probability of collisions in the hashing function, and hence a higher probability of cache-hits in future accesses.

There are many data layout schemes, as detailed Section 2. In the following section, we present a new algorithm for data layout called the *2-factor layouts*, which is a graph based algorithm for a cache-oblivious layout.

6 Data Layout Algorithm

Consider a weighted graph in which the nodes represent a group of triangles and the edge weight between the nodes being the distance between the primitives of the nodes by some metric (like Euclidean distance or normal cone axis deviation). Any two nodes that may have to be adjacent to each other in the final layout (depending on the application) can be connected by an edgThe linear ordering of the nodes of this weighted graph would give the order in which these nodes have to be laid out on the disk. Such a linear order which minimizes the sum of the weights of the edges between the adjacent nodes in the order can be computed using the following algorithm, which is an extension of the algorithm to linearly order primitives in quadrilateral and tetrahedral meshes presented in [Diaz-Gutierrez and Gopi 2005].

In order to find the linear ordering of nodes in a weighted edge graph, we use a graph matching algorithm. A graph matching couples every node with at most one of its neighbors. If all nodes are matched, we call it a perfect matching. The subgraph induced by the perfect matching is a spanning 1-regular graph (all nodes in the original graph are present and every node has degree one), also called a 1-factor of the original graph. A 2-factor of a graph is a spanning 2-regular subgraph. Since every node has degree two, a 2-factor will consist only of cycles. Some graphs may not have a 2-factor. So our effort to compute a 2-factor in the given weighted graph may result in a mix of cycles and linear strips of nodes in the graph. These cycles can be cut, and along with other linear strips, they can be laid one after the other in sequence to get a linear ordering of the nodes of the graph. If nodes represent geometric primitives, this linear ordering would yield a disk layout of these primitives. If the 2-factor is computed to minimize the sum of the edge weights of the resulting subgraph, then the linear order would place two nodes that have smaller edge weight (distance) between them, close to each other. We compute the 2-factor using a 1-factor computing algorithm as given below.

The idea for the computation of partial linear ordering of the nodes in the original graph is to *inflate* the original graph by substituting a template node for every node in the original graph. On this inflated graph, we compute the 1-factor, which automatically gives a partial ordering on the original graph. This partial ordering can then be used to compute a total ordering of the nodes as done in several related papers [Gopi and Eppstein 2004; Diaz-Gutierrez et al. 2005b]. This idea of inflated graph is similar to the one proposed in [Diaz-Gutierrez and Gopi 2005] for a 4-regular graph. But now, we extend this method for general graphs. An illustration of this algorithm is given in Figure 4. With the original graph being weighted, a weight minimizing graph matching would directly give cycles of nodes that are most likely to be accessed in-order and hence have to be stored likewise in the disk to maximize the access coherence pattern.

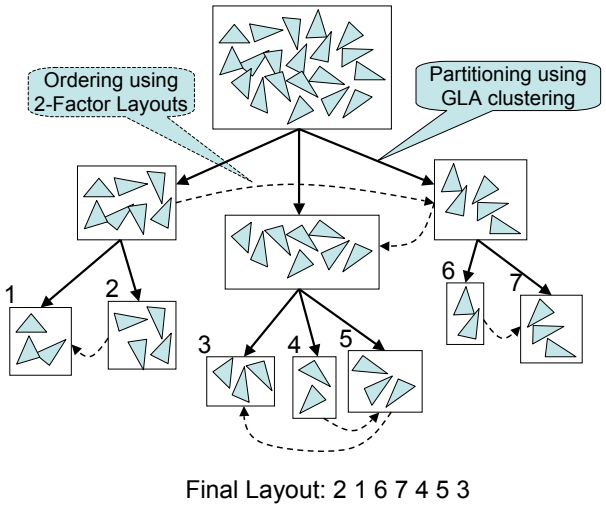

Figure 5: *Data Layout Algorithm*

The above algorithm is refined further for hierarchical ordering of large data sets. The proposed data layout algorithm recursively subdivides a data set into M partitions using the generalized Lloyd's algorithm (GLA) with the same clustering function used in the above graph algorithm for layouts. These partitions are ordered using the 2-factor layout algorithm presented above. The above two steps are repeated recursively on each partition until the number of primitives contained in a partition is below a predefined threshold. After the

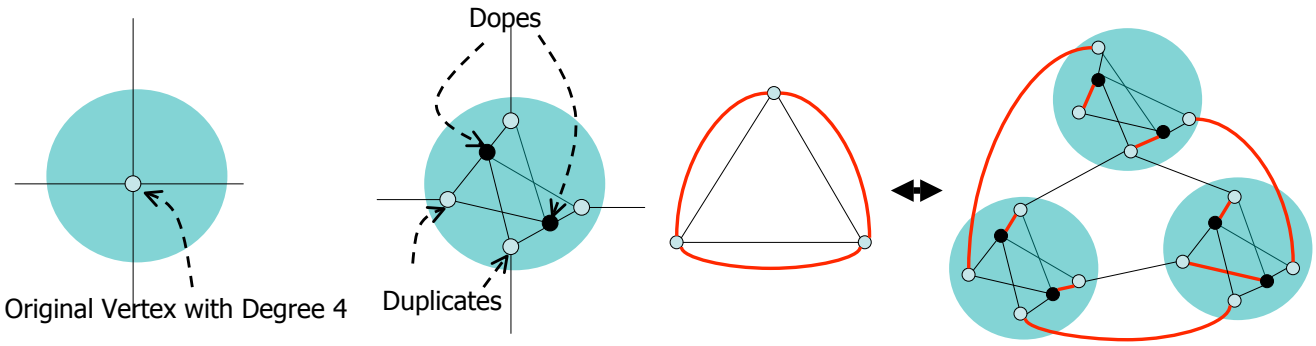

Figure 4: *Template substitution to find linear ordering: (Left) A node in the graph with degree k (4 in figure) is duplicated k times (see "Duplicates"), and $k-2$ more nodes called "dopes" are added. Every dope is connected to every duplicate. This forms a template for that particular node in the original graph. Right: An example inflated graph for a 4-regular 3-node original graph. An 1-factor (every node has exactly one selected edge) in the inflated graph means a 2-factor (every node has exactly two selected edges) in the original graph. This 2-factor imposes cycles in the original graph that gives a partial ordering, which can be converted to a total ordering with appropriate application dependent heuristics.*

tree has been fully expanded, all primitives are placed in a file using the orders calculated within and between the partitions as described.

The weighting functions between the nodes for clustering (GLA) as well as ordering (edge weights for our weighted graph) are dependent on the application. For our walkthrough application, the weighting function is basically the Euclidean distance between the primitives (or the bounding boxes of group of primitives). Once the number of triangles within each group is less than a threshold, the weighting function for linear ordering of primitives is switched to be the normal deviation between the primitives, producing a normal-based clustering for easy back-face culling.

7 Implementation and Results

Our data set is a city model consisting of 110 million triangles which originally consists of 90 million vertices and takes 4528 MB of disk storage. The converted model in page format consists of 115 million vertices and takes 3814 MB of disk storage without any compression due to the less space required for vertex index representation in the self contained page format. From this space 264 MB is used to store the vertex coordinates and normals, 690 MB is used to store the connectivity information with 2-byte vertex indices, and 345 MB is used to store color information for the triangles.

First, we partition the object space into small work-spaces called cells, such that the view positions from anywhere within that cell is equivalent with respect to the image generated by the distant objects from the viewpoints within that cell. Such distant objects are rendered to create textures which will be used during interactive walkthroughs. This fundamental technique of texture substitution for walkthrough systems has been used by researchers for a long time since [Aliaga and Lastra 1997]. In our experiment we create eight textures (45 degrees of viewing angle each) around each unit-square cell from its center, by rendering all primitives that are more than 10 units away (Figure 6). Obvious advantage of this technique over other simplification/primitive-reduction techniques is its generality: it can handle virtually any kind of primitives (triangles, points, etc.) of any topology (non-manifold/genus). The preprocessing time is proportional to the size of the model and is completely automatic with no manual intervention. In our system, we have 160 x 160 cells and a total of 160 x 160 x 8 textures with the total size of 640MB, and it took 20 hours (10 hours split on two typical desktop computers) to capture all the textures on the machine described below. Currently, we maintain these textures in-core. As seen from Figure 7, the textures form a very small part of the entire image; further, our cell sizes are also very small. So the projection error in our walkthrough system due to texture-based simplification is minimal.

For computing the textures, we used a desktop with Quad Core Intel Xeon Processor E5405 with 2GB memory, nVidia NVS 290 video card, and a 160GB SATA, 10,000 rpm hard drive.

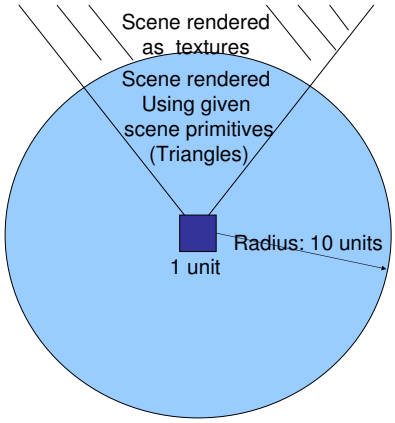

Figure 6: *Texture-Based Simplification*

Our disk-page hierarchy can work with any data layout. We tested our system with three layouts: a layout of the pages formed from just concatenating all the models in the scene in the specified page format (Section 3), a 2-factor layout of the same scene (Section 6), and the cache-oblivious mesh layout by Yoon et al. [2006]. Rewriting the entire scene (or a 2-factor ordering of the scene) in the appropriate page format took less than 15 minutes of processing (linear time in the scene size). Once this layout is given, we compute the bounding box and normal cone of the set of primitives contained in each disk-page, and construct disk-page K-d tree using the bounding box information. These computations are linear on the number of disk-pages, and took a total of less than 5 minutes

Figure 7: *Images from the walkthrough system: Left: Only the texture; Center: Only the nearby geometry; Right: Both geometry and texture.*

for roughly 930,000 pages of the entire scene on the machine described below.

For interactive walkthrough rendering and all the pre-processing operations, except for texture computation, we used a laptop with the following specifications: Intel Dual Core 2 T7500 2.2GHz, 64bit, 4MB cache, 1GB memory, 120GB 5400 rpm SATA drive with 8MB cache, and nVidia GeForce 8400 video card, running the Kubuntu 8.04 operating system. Note that considerable portion of the 1GB main memory is used to keep the textures and the K-d tree hierarchy and less than 264 MB of it is remained to be used by the operating system to cache the disk pages.

The in-memory data consists of the disk-page K-d tree hierarchy of 96 MB including the bounding box/normal cone of the disk-pages, and 160 x 160 x 8 textures with a total size of 640MB. In a laptop with the specifications mentioned above, we achieved on an average 20 frames per second (FPS) with the original data layout (a raster layout without any reordering) and 27 FPS using the cache oblivious layout by Yoon et al. [2006] and 28 FPS with the 2-factor layout. The average number of disk pages requested to the operating system per frame was 2365, while the page level cache management was done by the operating system. As mentioned before each disk-page has size 4KB. The renderer performed only view-frustum culling, limited back-face culling, and texture substitution for distant objects. With these simple primitive reduction techniques, the average number of triangles rendered per frame was around 240,000, not including the triangles required to render the textures that represent distant objects.

Note that the latest SAS disk drives are nearly three times faster (15,000 rpm) than the disk-drives in our laptop (5400 rpm); we did not use any GPU based acceleration technique like triangle strips, any sophisticated simplification algorithm or visibility culling methods; further, our 2-factor layout is not fully optimized for this walkthrough application. We also do not use data prefetching or multi-threading of data processing and rendering. With all these additional improvements, we expect to see a consistent frame-rate of over 35 fps on walkthrough scenes of similar characteristics and complexity.

8 Conclusion and Future Work

We presented a simple data structure on the page layouts of the data sets used for walkthrough application. We have shown that the data structure built on the layouts of the data, rather than on the data itself, has advantages in both implementation and performance of the system. In our walkthrough system, we used a K-d tree structure on the bounding boxes of the set of primitives in each disk-page. The preprocessing required for this walkthrough system involves texture generation, layout computation, page creation, and the bounding box/normal cone computation for each page. All these processes are completely automated since this system can accept virtually any kind of primitives and geometry in the scene. The walkthrough system itself is extremely simple – uses only texture substitution for distant objects as a form of simplification, and does no explicit cache-management, since this can be done more efficiently by the operating system.

As part of the future work, we have to analyze the disk-page hierarchy on other data data structures like Octree, and explore other applications that can make use of this data structure. We also have to improve our walkthrough system to include simple simplification techniques like [Luebke and Erikson 1997] (that again works just with triangle "soups" rather than objects) for better performance.

Note that in this work we do not do our own cache management. Adding a cache management system to this system and analyzing the number cache hits and cache misses can be considered as another future work.

Acknowledgements

M. Gopi is supported by the NSF grants CCF-0738401 and CCF-0811809.

Sung-eui Yoon was supported in part by KAIST & LG seed grants, MKE/IITA [2008-F-033-01], MKE/IITA u-Learning, MKE digital mask control, and DAPA & ADD contract(UD080042AD).

References

ALIAGA, D., AND LASTRA, A. 1997. Architectural Walkthroughs using Portal Textures. In *Proc. IEEE Visualization*, 355–362.

BENTLEY, J. 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM 18*, 9, 509–517.

BITTNER, J., WIMMER, M., PIRINGER, H., AND PURGATHOFER, W. 2004. Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum (Eurographics) 23*, 3, 615–624.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2004. Adaptive TetraPuzzles – efficient out-of-core construction and visualization of gigantic polygonal models. *ACM Transactions on Graphics 23*, 3.

CORRÊA, W. T., KLOSOWSKI, J. T., AND SILVA, C. T. 2003. Visibility-based prefetching for interactive out-of-core rendering. In *IEEE Symp. on Parallel and Large-Data Visualization and Graphics)*, 1–8.

DEERING, M. F. 1995. Geometry compression. In *ACM SIGGRAPH*, 13–20.

DIAZ-GUTIERREZ, P., AND GOPI, M. 2005. Quadrilateral and tetrahedral mesh stripification using 2-factor partitioning of the dual graph. *The Visual Computer 21*, 8-10, 689–697.

DIAZ-GUTIERREZ, P., BHUSHAN, A., GOPI, M., AND PAJAROLA, R. 2005. Constrained Strip Generation and Management for Efficient Interactive 3D Rendering. In *Proc. of Computer Graphics International Conference*, 115–121.

DIAZ-GUTIERREZ, P., GOPI, M., AND PAJAROLA, R. 2005. Hierarchiless simplification, stripification, and compression of triangulated two manifolds. In *Proc. of EUROGRAPHICS*.

DIAZ-GUTIERREZ, P., BHUSHAN, A., GOPI, M., AND PAJAROLA, R. 2006. Single Strips for Fast Interactive Rendering. *The Visual Computer 22*, 6, 372– 386.

DUCHAINEAU, M., WOLINSKY, M., SIGETI, D. E., MILLER, M. C., ALDRICH, C., AND MINEEV-WEINSTEIN, M. B. 1997. ROAMing Terrain: Real-time Optimally Adapting Meshes. In *Proc. IEEE Visualization*, 81–88.

GOBBETTI, E., AND MARTON, F. 2005. Far voxels: A multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics forum. In *Proc. of ACM SIGGRAPH*.

GOPI, M., AND EPPSTEIN, D. 2004. Single strip triangulation of manifolds with arbitrary topology. *Computer Graphics Forum (EUROGRAPHICS) 23*, 3, 371–379.

HOPPE, H. 1999. Optimization of mesh locality for transparent vertex caching. *ACM SIGGRAPH*, 269–276.

ISENBURG, M., AND LINDSTROM, P. 2005. Streaming meshes. *IEEE Visualization*, 231–238.

ISENBURG, M., LINDSTROM, P., GUMHOLD, S., AND SNOEYINK, J. 2003. Large mesh simplification using processing sequences. *IEEE Visualization*, 465–472.

LIN, M., AND MANOCHA, D. 2003. Collision and proximity queries. In *Handbook of Discrete and Computational Geometry*.

LINDSTROM, P., AND PASCUCCI, V. 2001. Visualization of large terrains made easy. *IEEE Visualization*, 363–370.

LOSASSO, F., AND HOPPE, H. 2004. Geometry clipmaps: Terrain rendering using nested regular grids. In *ACM SIGGRAPH*, 269–776.

LUEBKE, D., AND ERIKSON, C. 1997. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 199–208.

LUEBKE, D., REDDY, M., COHEN, J., VARSHNEY, A., WATSON, B., AND HUEBNER, R. 2002. *Level of Detail for 3D Graphics*. Morgan-Kaufmann.

OTADUY, M. A., AND LIN, M. C. 2003. CLODs: Dual hierarchies for multiresolution collision detection. *Eurographics Symposium on Geometry Processing*, 94–101.

PASCUCCI, V., AND FRANK, R. J. 2001. Global static indexing for real-time exploration of very large regular grids. In *Supercomputing*.

SAGAN, H. 1994. *Space-Filling Curves*. Springer-Verlag.

SAMET, H. 2006. *Foundations of MultiDimensional and Metric Data Structures*. Morgan Kaufmann.

SANDER, P. V., NEHAB, D., AND BARCZAK, J. 2007. Fast triangle reordering for vertex locality and reduced overdraw. *ACM Trans. on Graphics (SIGGRAPH) 26*, 3, 89.

TESCHNER, M., KIMMERLE, S., HEIDELBERGER, B., ZACHMANN, G., RAGHUPATHI, L., FUHRMANN, A., CANI, M.-P., FAURE, F., MAGNENAT-THALMANN, N., STRASSER, W., AND VOLINO, P. 2005. Collision detection for deformable objects. *Computer Graphics Forum 19*, 1, 61–81.

VARADHAN, G., AND MANOCHA, D. 2002. Out-of-core rendering of massive geometric environments. In *IEEE Visualization*.

VELHO, L., AND DE MIRANDA GOMES, J. 1991. Digital halftoning with space filling curves. In *ACM SIGGRAPH*, 81–90.

WALD, I., AND HAVRAN, V. 2006. On building fast kd-trees for ray tracing, and on doing that in O(N log N). In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, 61–69.

WALTER, B., BALA, K., KULKARNI, M., AND PINGALI, K. 2008. Fast agglomerative clustering for rendering. *IEEE Symp. on Interactive Ray Tracing*, 81–86.

YOON, S.-E., AND LINDSTROM, P. 2006. Mesh layouts for block-based caches. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization) 12*, 5.

YOON, S.-E., LINDSTROM, P., PASCUCCI, V., AND MANOCHA, D. 2005. Cache-Oblivious Mesh Layouts. *ACM Transactions on Graphics (SIGGRAPH) 24*, 3, 886–893.

YOON, S., GOBBETTI, E., KASIK, D., AND MANOCHA, D. 2008. *Real-Time Massive Model Rendering*. Morgan & Claypool Publisher.