Performance Driven Redundancy Optimization of Data Layouts for Walkthrough Applications

Abstract

Performance of interactive graphics walkthrough systems depend on the time taken to fetch the required data to render from the secondary storage to the main memory. It has been earlier established that a large fraction of this fetch time is spent on seeking the data on the hard disk. In order to reduce this seek time, redundant data storage has been proposed in the literature, but the redundancy factors of those layouts are prohibitively high. In this paper, we develop a model for the seek time of a layout, and using this cost model, we propose an algorithm that would compute a redundant data layout with the redundancy factor that is within the user specified bounds while maximizing the performance of the system.

Keywords: Data Layout Problem, Out-Of-Core Rendering, Cache Oblivious Mesh Layout, Redundant Data Layout, Walkthrough Application

1 Introduction

1.1 Walkthrough Rendering

In typical walkthrough systems, data sets consisting of hundreds of millions of triangles and many gigabytes of associated data (e.g. walking through a virtual city) are quite common. Rendering such massive amounts of data requires out-of-core rendering algorithms that bring only the required data for rendering into main memory from secondary storage. In this process, in addition to the rendering speed, the data fetch speed also becomes critical for achieving interactivity, especially when we handle large-scale data. In general, data fetch speed depends on data seek time and data transfer time. Transfer time depends only on the amount of data that is transferred. Seek time is the time taken to locate the beginning of the required data in the storage device and depends on different factors depending on the storage medium.

For a hard disk drive (HDD), its seek time depends on the speed of the disk, and the relative placement of the data units with respect to each other, also called the data layout [Rizvi and Chung 2010]. For a solid state drive (SSD), this seek time is usually a small constant and is independent of the location of the data with respect to each other [Agrawal et al. 2008]. An earlier work utilized this difference between SSD and HDD and designed a data layout tailored for using SSDs with the walkthrough application [Sajadi et al.]. There have been many other techniques utilizing SSDs for various applications [Saxena and Swift 2009]. SSD, unfortunately, is not the perfect data storage and has its own technical problems, including limited number of data overwrites allowed, high cost, and limited capacity [Rizvi and Chung 2010].

On the other hand, the HDD technology – including disk technologies such as CDs, DVDs, and Blu-ray discs – has become quite reliable and inexpensive thanks to their extensive verifications and testing, and is thus in widespread use. Even for massive data sets hard disk drives (HDD) are still and will be the preferred medium of storage for the foreseeable future [Rizvi and Chung 2010], mainly because of its stability and low cost per unit. As an example, according to [Domingo 2014], as of 2014, an HDD can cost $0.08 per GB, while an SDD can cost $0.60 per GB. As a result, optimizing components of walkthrough systems with HDDs is critical. In particular, addressing the seek time, the main bottleneck of accessing data from HDDs, remains the main challenge for interactive rendering of massive data sets. In this paper, we leverage the inexpensive nature of HDDs to store redundant copies of data in order to reduce the seek time.

1.2 Redundancy based data layouts

In this paper, we leverage the inexpensive nature of HDDs to store redundant copies of data in order to reduce the seek time. Adding redundancy in order to improve the data access time is a classic approach, e.g., RAID [Patterson et al. 1988]. We are also not the first to consider redundancy for walkthrough applications. Redundancy based data layouts to reduce the seek time were introduced in a recent work [Jiang et al. a], in which the number of seeks for every access was reduced to at most one unit. However, in order to achieve this property, the redundancy factor – the ratio between the size of the data after using redundancy to the original size of the data was prohibitively high – around 80.

Another recent work [Jiang et al. b] related ~~a~~ transfer volume, which dictates the data transfer time, seek time, and redundancy, and proposed a linear programming approach to optimize the data transfer and seek time in order to satisfy the total data fetch time constraint. In the process, redundancy was a hidden variable that was minimized. Unfortunately, this approach does not directly model redundancy or seek time, and thus can have unnecessary data blocks and unrealistic seek times.

1.3 Main Contributions

In this paper, we propose a model for seek time based on the actual distance between the data blocks in the linear data layout. Using this model, and given the spatial proximity of the data set for a walkthrough application, we develop an algorithm to duplicate data blocks strategically to maximize the reduction in the seek time while keeping the redundancy factor within the user defined bound. We will show that our greedy solution generates both the extreme cases of data layout with redundancy, namely the no redundancy case (a simple cache oblivious mesh layout with low number of seeks) and the maximum redundancy case (a layout where seek time is at most one), as well as reasonable solutions for the redundancy factor constraints in between the extremes. We show that the implementation of our algorithm significantly reduces average delay between frames and noticeably improves the consistency of performance and interactivity.

2 Related Work

Massive model rendering is a well studied problem in computer graphics. Most of the early works focused on increasing the rendering efficiency. At that time the fundamental problem was not fitting the model into the main memory but the speed of the graphics cards. Hence these works provided solutions to reduce the number of primitives to be rendered while maintaining the visual fidelity. These solutions included level-of-detail for geometric models [Luebke et al. 2002], progressive level of detail [Hoppe 1998; Hoppe 1997; Hoppe 1996; Shaffer and Garland 2001], and image based simplification [Aliaga et al. 1999]. Soon thereafter the size of the main memory became the bottleneck in handling ever increasing sizes of the model. Hence memory-less simplification techniques [Lindstrom and Turk 1999] and other out-of-core rendering systems [Silva et al. 2002; Varadhan and Manocha 2002] emerged in which just the limited amount of required data that needs to be processed and rendered was brought from the secondary storage to the main memory.

Soon afterward, the researchers realized that the speed at which this data could be brought from the secondary to the main memory in these out-of-core algorithms is limited by the data bus speed, disk seek time, and data transfer time. They realized that these issues could be ameliorated to some extent by better cache utilization that would increase the utilization of data that is brought to the main memory and thus reduce the number of times the disk read is initiated. This meant that subsequent works focused on cache aware [Sajadi et al.] and cache oblivious data layouts [Yoon et al. ; Yoon and Lindstrom 2006] on the disk to reduce the data fetch bottleneck. Our work falls under this class of algorithms that reduces the data fetch time.

Redundancy based data layouts were mentioned in [Patterson et al. 1988; Jiang et al. a; Jiang et al. b] as potential solutions to this problem of reducing seek time. In particular [Jiang et al. b] presented an algorithm that limits the amount of redundancy required but there were major drawbacks. First, it provides a grouping of data units for each seek but it does not provide a data layout. This is because it does not relate one data group with another and it does not consider their relative layout. This could easily result in unnecessary data block duplications. The redundancy minimization is thus not modeled after physical representation of the data layout on the disk. The second major drawback is that the model for seek time is also not based on physical reality. Typically, seek time depends on the relative distance on the disk between the last data unit accessed and the data unit currently being requested. However, in the paper, seek time is simplistically modeled as number of seeks. This means that even if the requested data blocks are adjacent to each other and have no separate seek time to go from one block to another, this model would add them to the cost because it counts individual data blocks as one seek. Our approach seeks to address these issues.

3 Greedy Redundancy-based Cache Oblivious Data Layout Algorithm

3.1 Definitions

For the purposes of this paper, we are given uniform data units in a linear sequence and each data unit has one or more access requirements assigned to it. An individual data unit consists of geometric primitive data or other information that must be retrieved during walkthrough rendering. The access requirements are determined by the application and represent data units that will be accessed together. An example of this could be vertices that are spatially near each other.

For an access requirement A, the total span of A is the total number of data units between the first and last data units that use A. If a data unit is not required by A but lies between the first and last unit of A then it is still counted in the span of A. Figure 1 shows a set of data units and explicitly labels the blue access requirement in this set. The total number of data units between the first and last blue unit is 14 thus that is the total length of the blue access requirement.

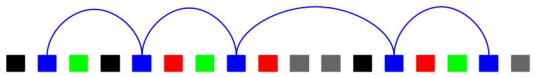

Figure 1: *Illustration of data units and a single access requirement. The blocks are data units and the colors represent access requirements. The lines connect data blocks with the same access requirement and represent parts of the span of an access requirement.*

Instead of trying to find the seek time explicitly while running, we use this one dimensional layout to compute a probabilistic estimate for the seek time. A cache oblivious measure was introduced to measure the expected number of cache misses [Yoon et al.]. Note that every time we have cache misses, we would have to seek data from the hard disk. We extend this idea by thinking of an individual cache miss as an individual seek. We define the Expected Seek Time (EST) as the total number of individual seeks that could be required. Let I be the set of access requirements and A_i represent the span of access requirement i. Because data units in an access requirement will be accessed together, we can assert the following definition:

$$EST = \sum_{i \in I} A_i$$

3.2 Algorithm Description

In [Yoon et al.], the only allowed operation on the data units is the move operation and the optimal solution is computed using only that operation. For our purposes, we are allowed to copy data units, move them, and delete them if they are not used. We want to figure out how to minimize EST while keeping the number of extra copies as low as possible. After constructing a cache oblivious layout to the mesh to get an initial optimization, we take a data unit and copy it to a place that will mean shortening at least one of the access requirements that use it. If the new data unit reduces the span of all the access requirements attached to it, then we delete the original data unit. We repeat the individual copying and possible deletion procedure until our redundancy limit has been reached.

Blocks to consider: Note that the span of an access requirement does not change by moving an interior data unit to another interior location. Cost can be reduced only by moving the blocks that are at the either ends of the access requirement. This will greatly reduce the search space of data units to consider for copying. For the sake of simplicity of the algorithm, we operate on only one data block at a time.

Location of copies: Based on the above observation, given an access requirement, we can possibly move the beginning or