~~A Greedy Heuristic using Redundancy for the Data Layout Problem~~

~~Zachary DeStefano*[1]~~, ~~Shan Jiang†[1]~~, ~~Gopi Meenakshisundaram‡[1]~~, ~~and Sung-Eui Yoon§[2]~~

[1]University of California, Irvine
[2]KAIST

Abstract

~~In this paper, we present a greedy heuristic that uses redundancy to try and optimize seek time in a cache oblivious mesh layout. This is an important problem that appears when you are attempting to render massive amounts of geometric data that must be accessed together but cannot fit into main memory. We take previous work on optimizing the data layout and attempt to further reduce the seek time by inserting redundant data into key places. We have an algorithm that inserts redundant data units individually and ensures that each insertion is the one that reduces seek time the most at that step. In this way, we can minimize seek time using a limit on the amount of redundancy that we are allowed. This will prove to be an improvement over existing algorithms both analytically and experimentally.~~

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types;

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

1 Introduction

~~When attempting to render geometric models that contain hundreds of millions of vertices, the geometric data is too large to fit into main memory so only a small part of the model can be accessed at a given moment. When rendering a walkthrough application in this context, there is thus a seek time penalty that arises from getting data from the hard disk. This leads us to the Data Layout Problem, which is how do we lay out the data on the hard disk in such a way that minimizes the seek time required for the geometric data. Yoon et al [Yoon et al.] described this problem and came up with a metric to find the average seek time given the data layout as well as the access requirements of the data. An access requirement is a set of data units that will likely be accessed together during run-time.~~

~~For the purposes of this paper, we are assuming that the relevant data units are laid out linearly on the disk. We are also assuming that the data units can be laid out in any order. Each data unit is assigned to one or more access requirements. With Yoon's definition, the average seek time ends up being the sum of the lengths of all the access requirements. We noticed that access requirements described in the paper can be very general and there could easily be data units that are far apart in the sequence but need to be accessed together. This led us to realize that if we copy data units and move them closer to other ones that share the same access requirement then we could save a lot of seek time without adding much storage space.~~

~~We decided to formalize this idea in the form of a greedy algorithm that at each step copies the data unit which will improve seek time the most. Each step in the algorithm will copy a single data unit. We can thus stop the algorithm when we no longer have storage space to store more copies. This algorithm is thus a heuristic for the problem of given a certain amount of storage space that we are limited by, how can we minimize the seek time.~~

2 Related Work

~~Massive model rendering has been a challenging field of research in computer graphics for decades. A large body of literature has been built on different aspects of solving this problem. Here we briefly discuss work in two main approaches in this topic, Out Of Core Rendering and Image-based Rendering.~~

~~2.1 Out-Of-Core Rendering~~

~~Large scale model rendering generally implies that the geometry data is so large that one must employ a secondary storage device during rendering. Thus, due to the nature of secondary storage devices and the architecture of modern computers, data fetching and data management for main memory can become performance bottlenecks during rendering. To remove or reduce these bottlenecks, research on out-of-core rendering aims at fetching, caching, and managing data in efficient ways. Coorea et al [Correa et al. 2002] introduced the iWalk system, in which octree-based spatialization is applied on geometry data, and this allowed only visible data to be retrieved from hard drives. Varadhan et al [Varadhan and Manocha 2002] also focused on isolating data required by each frame, but in a graph-based algorithm. A scene graph is generated in level of details and frame-to-frame visibility consistence. Parallel processing is also employed so that rendering the active part of the scene and fetching objects from the disk are done simultaneously. Sajadi et al [Sajadi et al. a] improved efficiency of out-of-core rendering by preprocessing the data set into a form of disk pages. The disk pages are self-contained data units with fixed size. This method avoids data management on the primitive level which reduces the time complexity by orders of magnitude. By utilizing a globally optimized data layout, caching can be further improved. Globally optimized data layout is an NP hard problem. Nevertheless, Yoon et al [Yoon et al.] provided a feasible method to try to compute it. Similar to other problems where an optimal permutation needs to be computed, they developed a hierarchical algorithm with a~~ heuristic based on edge spans to get an approximated solution quickly.

2.2 ~~Image-Based Rendering~~

~~Image-based rendering techniques reduce the geometry of a massive model to a view-independent mesh along with pre-rendered textures. The number of primitives to be rendered is much less than the original model. Therefore, both data transferring time and ren-~~

*zdestefa@uci.edu
†sjiang1714@gmail.com
‡gopi.meenakshisundaram@gmail.com
§toinsert

~~dering time can be greatly saved. The major drawback of image-based rendering is the lack of geometry information. Zhang et al [Zhang and Chen 2004] presented a survey of image-based rendering techniques. These techniques involve sampling the space using a plenoptic function in the first stage and then rendering the continuous plenoptic function in the next stage. It is similar to the technique in signal processing of sampling a set of numbers and then using those samples to get the continuous function and reconstructing the continuous function. The paper goes into detail on various applications of these techniques. Li et al [Li et al. 2000] explored various compression methods for image based rendering. Specifically, they explored the benefits of three compression algorithms. All three proved to be able to achieve real time rendering of large models and they vary in the complexity of the decoding and the compression ratio they are able to achieve.~~

2.3 Seek Time and Redundancy-based solutions

In real time rendering, time spent on one frame can be roughly divided into data fetching time, online processing time, and rendering time. Data fetching time can be further decomposed into seek time and data transfer time. Sajadi et al [Sajadi et al. b] explored the reasons for the performance advantage of Solid-State Drives (SSD) over Hard-Disk Drives (HDD). The result shows clearly constant seek time of SSDs is the major reason that fetching data on SSDs is generally faster and more consistent than HDDs. Jiang et al [Jiang et al. a] minimized disk seeks to one or less for each frame by storing multiple copies of same data at different locations of secondary storage devices. These extra copies will be referred to as redundancy. The paper successfully showed that limited seeks lead to improvement of performance. However, to keep number of seeks being one or less, a large amount of redundancy is necessary, which is not practical for many applications. Jiang et al [Jiang et al. b] generalized this approach by relaxing the number of seeks to a small threshold. This threshold is determined by time budget of each frame. In this way, the number of seeks required is relaxed to minimize amount of redundancy. This optimization is done through integer linear programming. This approach ended up reducing the amount of redundancy significantly. The algorithm however does not take data layout into account. It also does not take into account copies of the data that are not used and it does not take into account actual distances between these data units. For these reasons, it is not an optimal solution to our problem.

3 Greedy Redundancy-based Cache Oblivious Mesh Layout Algorithm

We can formulate this problem in the following way given Yoon's metric. We are given a linear sequence of data units as well as one or more access requirements that each data unit is assigned. In Yoon's paper, he is only allowed to move the data units around. In this paper, we are allowed to copy data units, move them, and delete old copies of data units. Given this input and these 3 operations, we need to figure out how to minimize the sum of the lengths of the access requirements using the least amount of extra space as possible. For this paper, we assume that we are given a certain amount of space and need to figure out how much we can minimize the seek time.

The algorithm in Yoon et al. computes a locally optimal solution without considering copying data units. Our algorithm takes over afterwards. W take a data unit and copy it to a place that will mean shortening at least one of the access requirements that is attached to it. If the new data unit shortens all the access requirements attached to it, then we delete the old data unit.

There are important issues to consider in order to make this idea into an algorithm. First of all, we need to know which data units in each access requirement we should consider. We then need to figure out which access requirements to take care of first. We need to know where to insert the copied data unit. Once we copy a data unit and find its location, we need to consider whether its other access requirements should use the old or new copy. Finally, we need to decide when to stop the algorithm.

3.1 Which data unit in each access requirement

Since we only care about the length of each access requirement and we can only copy a single data unit at one time, we will only be copying the data units that are on the endpoints of an access requirement. This will greatly reduce the search space of data units to consider for copying.

The figure below shows an example access requirement. As can be observed, if we move any of the interior units, the access requirement will stay the same length or become larger. However if we move the start or end unit, the access requirement will become shorter.

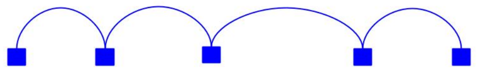
Figure 1: *Data Units for a single access requirement*

3.2 Where to locate data unit

Since we are only allowed to move one data unit at a time, in order to maximize our benefit to our current access requirement, we want to copy the start data unit to somewhere between the one after the first one and the last one. In the same manner, if the end data unit is better, we want to copy it to somewhere between the first one and the one before the last one. Below is the access requirement from the above figure with an arrow showing the locations where the start unit can be moved to.

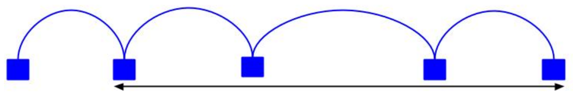
Figure 2: *Interval where the start data unit can be copied to*

By doing this we are guaranteed to reduce the seek time for the access requirement we care about. For our own access requirement, it won't matter where in that interval we place our data unit. However, for the other access requirements that overlap a potential place to insert our data unit, we are adding one unit of seek time. Therefore, we want to find which place will interrupt the least number of access requirements. We can assume that we have already precomputed the number of overlaps at each unit. We now have the problem of given a sequence of numbers and an arbitrary block of that sequence, what is the least number in that block. For our purposes, we will do a linear search through each entry in the block in order to find the ideal entry. While in theory there are more efficient solutions, they will be impractical for our purposes, as we will describe in the next section.