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.

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. These limitations 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. 2011] and cache oblivious data layouts [Yoon et al. 2005; 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. 2013; Jiang et al. 2014] as potential solutions to this problem of reducing seek time. In particular [Jiang et al. 2014] 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. Such an approach 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 [Jiang et al. 2014], seek time is simplistically modeled as counting the number of seeks, independent of the number of data units between them. This means that irrespective of whether the requested data blocks are adjacent to each other or far apart, this model would assign the same cost for both layouts. Our approach aims to address these issues.

3 Redundancy-based Cache Oblivious Data Layout Algorithm

3.1 Definitions

Let us assume that the walkthrough scene data, including all the levels of details of the model, are partitioned into equal sized data blocks (say 4KB) called data units. This is the atomic unit of data that is accessed and fetched from the disk. Typically, vertices and triangles that are spatially together (and belong to the same level of detail), have high chances of being rendered together, and hence can be grouped together in a data unit. All the data units required to render a scene from a viewpoint is labeled as an *access requirement*. In order to minimize the number of access requirements, the navigation space in the walkthrough scene, which defines the space of all possible view points, is partitioned into grids and all the viewpoints within each grid is grouped together to define one access requirement. Thus the number of grid partitions define the number of access requirements. Clearly, primitives in a data unit can be visible from many viewpoints, and hence that data unit will be part of many access requirements.

That was one example of data units and their access requirements. In general, the access requirements are determined by the application and are meant to be sets of data units that are likely to be accessed together. Given a linear ordering of data units that may eventually be the order in which they are stored in the hard drive, 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.~~

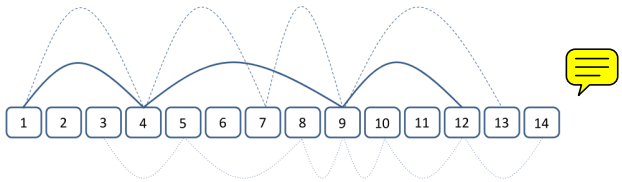

Figure 1: *Illustration of linear order of data units and three example access requirements. The lines connect data blocks that belong to the same access requirement and represent parts of the span of an access requirement.*

3.2 Seek Time Measure

Given a linear order of data units and the access requirements, and assuming that each access requirement is equally likely to be used, we would like to estimate the seek time for that application. For each access requirement the read head of the hard disk has to move from the first data block to the last irrespective of whether the intermediate blocks are read or skipped. Hence the span of an access requirement is a measure of seek time - time taken to seek the last data unit starting from the first data unit. Let I be the set of access requirements and A_i represent the span of the access requirement i. Then estimated total seek time EST is given by

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

It is interesting to note that [Yoon et al. 2005] used span to measure the expected number of cache misses. Typically, with every cache miss, the missing data will be sought in the disk and fetched, thus adding to the seek time. Hence using span to measure the seek time is justified.

3.3 Algorithm Overview

In [Yoon et al. 2005], 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. Using these operations, our goal is to minimize EST while keeping the number of redundant copies as low as possible. After constructing a cache oblivious layout of the data set to get an initial ordering of data units, we copy one data unit to another location, and reassign one or more of the access requirements that uses the old copy of the data unit to the new copy, such that the EST is reduced. If all the access requirements that used the old copy, now use the new copy of the data unit, then the old