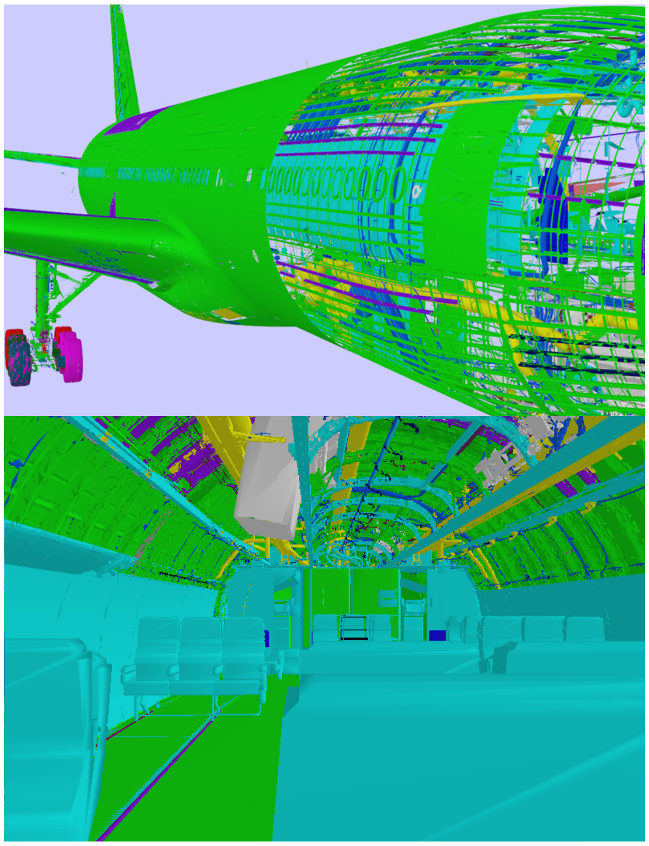


Figure 4: *Boeing model: 350 million triangles, 20 GB. Overview of model (top) and model detail (bottom).*

rate of the hard drive is 120 MB/s and the seek time is a minimum of 2 ms per disk seek.

Benchmarks: We use three models to perform our experiments, each model represents a use case or scenario. The City model (Figure 2) is a regular model that can be used in a navigation simulation application or virtual reality walkthrough. The Boeing model (Figure 4), on the other hand, represents scientific or engineering visualization applications. The Urban model (Figure 1) has texture attached to it, which is commonly used in games. By comparing performance of cache-oblivious layout without redundancy [Yoon et al. 2005] to our method using redundancy on these three models, our goal is to show that the redundancy based approach can achieve more stable and generally better performance on different real time applications.

To apply our method on these large-scale models, we had to find a proper set of access requirements. In general, that question is deep enough that it can be discussed as a separate research topic. Here, however, we had good performance using only simple schemes for creating access requirements. Each model ended up having a separate scheme. Nonetheless, an access requirement represents a set of data that is highly likely to be accessed together.

For the Boeing model, the predefined objects are used as a conceptual level to create access requirements. Samples of view positions are distributed across the model. For each sample, four fixed directions and four random directions are considered. Objects visible from this position in any one of these eight directions are added to access requirement for this specific sample. The density of these samples depends on the complexity of local ~~occulders~~ to reduce load of each access requirement, i.e. more samples are distributed to places with more complex geometry.

The Urban model is different from the previous two in a way that it involves textures. Building heavy redundancy of textures increases the total size of the dataset significantly, while keeping textures away from redundancy leads to inevitable long seek time, which is completely against the philosophy of this work. To solve this problem, we applied a spatial Lloyds clustering [Lloyd 1982] on objects. By moving centers of clusters, we look for a solution such that each cluster involves almost same amount of texture data. Between clusters, textures can be redundantly stored, but within each cluster, texture data are stored uniquely. In this way, each cluster is used as an access requirement.

For the City model, a 2D grid is used to divide the space into square cells. For each pair of adjacent cells, the difference of the data is considered as an access requirement. By propagating this rule, access requirements are created, and the number of them is determined by the resolution of the grid [Jiang et al. 2013].

The computation time to create redundancy layout is generally linearly correlated to the final redundancy factor for each model. For the examples we used in Figure 5, it took 16 minutes to create the redundancy layout from the cache-oblivious layout without redundancy for the City model. This number is 80 minutes and 38 minutes for the Boeing model and the Urban model, respectively.

~~**Results:**~~ Figure 5 shows the results of delays caused by fetching data on the experimental models we used. We compare the results of a cache-oblivious layout without redundancy [Yoon et al. 2005] and one with redundancy computed using the proposed method. For the layout with redundancy, we set the redundancy factor equal to 4.2. This factor was chosen because as can be seen in Figure 6, it had considerably better performance than lower redundancy factors and did not have significantly worse performance than higher redundancy factors. A factor of 4.2 is also still practical, as the largest model we tested, the Boeing model, becomes 84 GB which is still acceptable given the large capacity of modern secondary storage devices.

It is clear that the performance ~~of~~ the layout with redundancy has generally shorter delays than the cache-oblivious layout without redundancy. As can be observed from the results, although the layout with redundancy does not eliminate delays for most of sample points on the walkthrough path, it reduces delays to a small range and keeps the performance more consistent. Consistent as well as better performance is primarily the result of reduction of maximum as well as average data fetch delay as modeled by the EST measure (Section 3.2) using our optimization algorithm. Since the algorithm tends to eliminate seeks with longer seek time first, in practice the larger delays are avoided.

When we compare our method to the one in [Jiang et al. 2014] there is again a performance benefit and less redundancy required. The redundancy factor used for the linear programming method was 8.3 which was the ~~factor~~ that produced the best performance with that method. We used a redundancy factor of 4.2 for our method. ~~The performance benefit is shown in Figure 7.~~ Additionally in [Jiang et al. 2014], the user does not have any control over the final redundancy factor however in our proposed method, each time we duplicate one data unit, we can halt it if the redundancy factor reaches a certain threshold. This helps us to create data layouts with arbitrary redundancy factors.

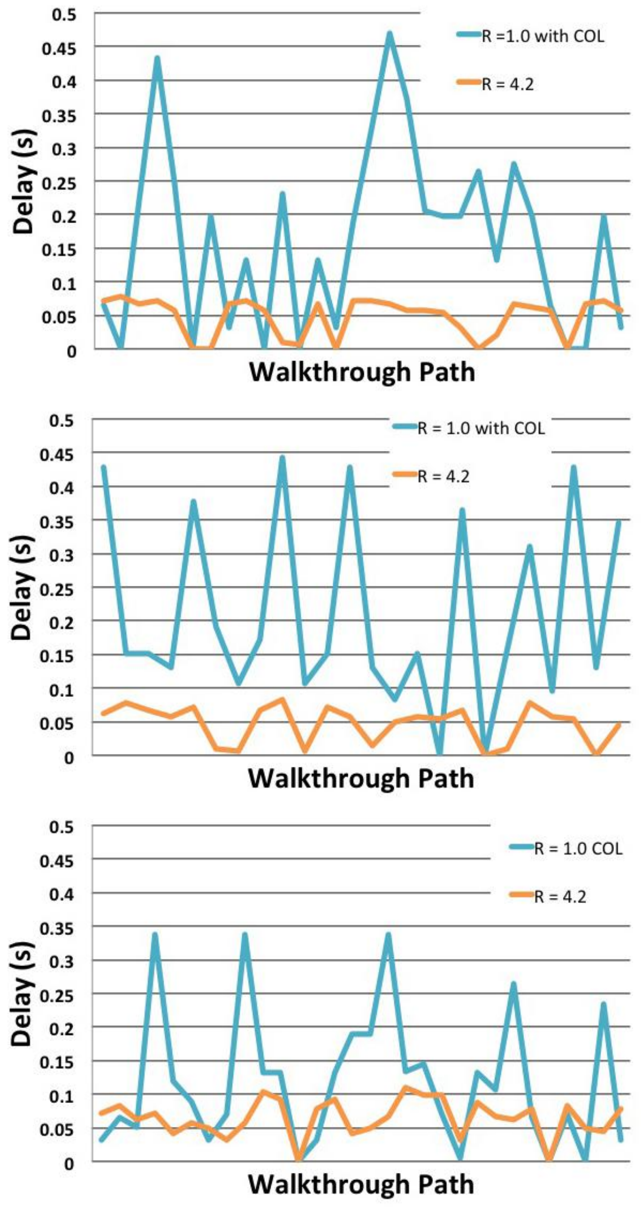


Figure 5: *Statistics of delays caused by the fetching processes for the City model (top), the Boeing model (center), and the Urban model (bottom), with and without redundancy. COL indicates a Cache-Oblivious Layout that does not use any redundancy. R indicates the redundancy factor.*

We use this fact to test the system with different redundancy factors. In Figure 6, we show the results of using layouts with redundancy factors that range from 1.0 to 10.0. The y axis in this figure is the ratio of the estimated seek time (EST) of the layout with redundancy over the EST of the layout without redundancy. This value starts at 1.0 where redundancy factor is 1.0, meaning no redundancy, and decreases as redundancy factor goes larger. The rate of decrease is exponential with larger benefits in the beginning meaning increasing the redundancy factor there reduces the seek time much more significantly than increasing the redundancy factor later. In other words, it is worthwhile to limit the redundancy factor used because after a certain point the cost of using redundancy

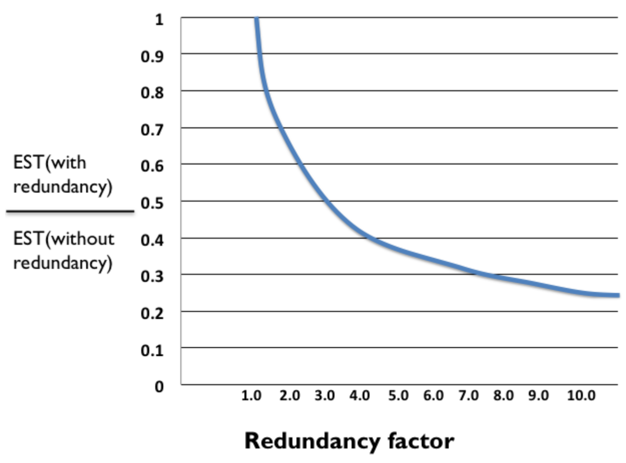


Figure 6: *Plot of the ratio of the EST of the layout with redundancy over the EST of the cache-oblivious mesh layout without redundancy for the Urban model. For the other models, the graphs look almost exactly the same.*

is very high – much more secondary storage space will be used without any significant improvement in seek time. It also implies that our algorithm dramatically reduces seek time in practice by using only small redundancy factors.

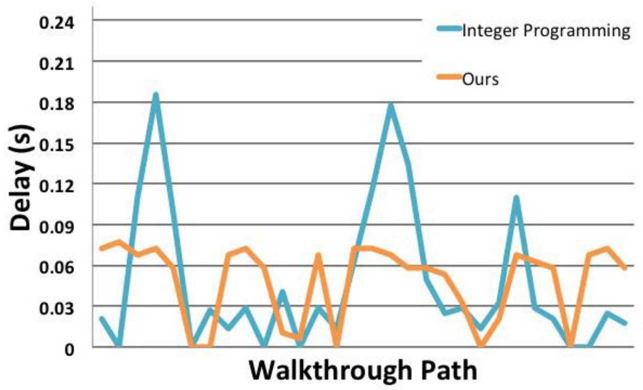
~~6 Analysis and Comparisons over Prior methods~~

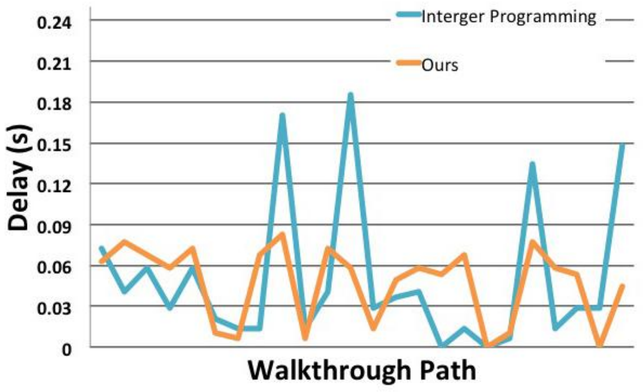
In the algorithm, we make a heap of data units that will reduce seek time by just moving instead of copying them. We perform these moves first before working with data units that need copying. This initial step will produce a better solution than proposed by [Yoon et al. 2005] without adding redundant units. This result is possible mainly because our optimization algorithm searches wider sets of potential locations for moving cases in an efficient manner. To show this, consider a case where we have two access requirements of 5 data units each. Figure 8 shows an example of that kind of layout. In the middle of that figure is the result of using the cache oblivious layout. Because it hierarchically constructs blocks and arranges the units in each block, it does not detect that the units with the black access requirement can be grouped together. On the other hand, the algorithm we propose would shorten the black access requirements without adding redundancy, as shown in the bottom of that figure.

The algorithm in [Yoon et al. 2005] did not necessarily produce the best cache oblivious ~~mesh~~ layout. However, even if we had the best layout without redundancy, we would actually achieve a better seek time ~~than it~~ using redundancy. We have such an example with Figure 9. As can be seen in the figure, the total seek time is 7 units which turns out to be the minimum possible seek time without redundancy, as found through a brute-force search. With redundancy, the total seek time is the minimum required which is 6 units. While a reduction from 7 to 6 units may not seem dramatic, when this result is scaled up to the hundreds of millions, this makes a big difference in seek time, which we saw in practice.

7 Conclusion and Future Work

Given the data units, access requirements, and the desired upper bound on redundancy factor, we have proposed an algorithm that would create a cache oblivious layout with the primary goal of





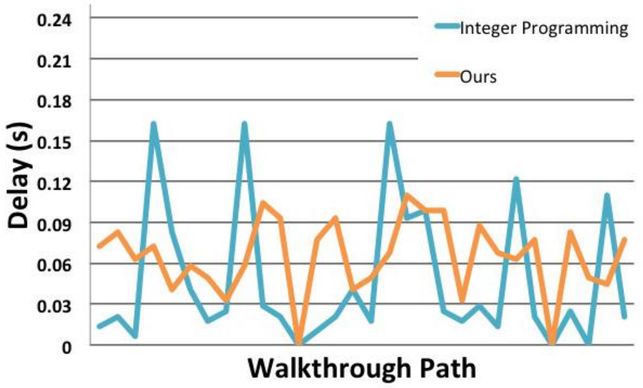


Figure 7: *Statistics of delays caused by the fetching processes for the City model (top), the Boeing model (center), and the Urban model (bottom), using integer programming and our method.*

reducing the seek time through duplicating the data units. We proposed a cost model for estimating the seek time, and in our algorithm we can move or copy data units in appropriate locations such that it reduces the estimated seek time. We have shown that such a layout significantly improves both the performance and consistency of interactivity in massive model walkthrough applications.

Our proposed redundant storage of data may limit editing and modification of data because the data has to be modified at all copies. However, we foresee no problem in recomputing and updating the layout due to this modification using our algorithm since every iteration in our algorithm just assumes a layout and improves on it. After data modification, we can delete/modify

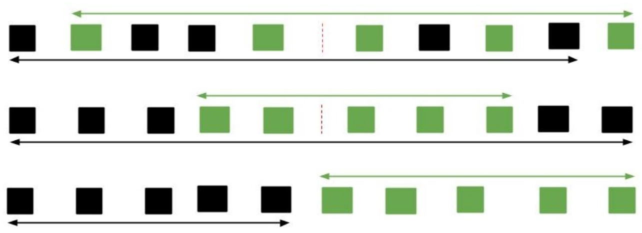


Figure 8: *Example of two access requirements of 5 data units each. The red line represents the boundary between blocks in the cache oblivious layout hierarchy. The original layout (top), cache-oblvious layout (middle), as well as the layout after running our algorithm (bottom) is shown.*

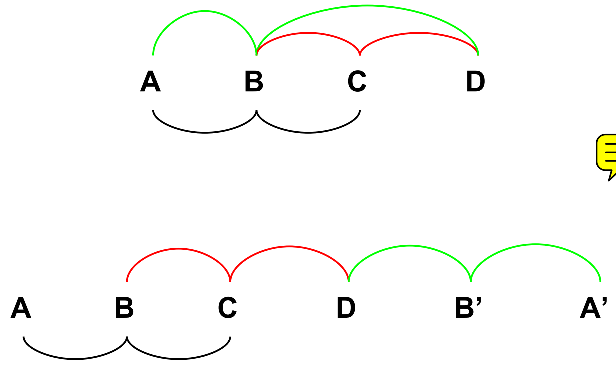


Figure 9: *Data Units with varying access requirements on the top. The letters represent data units and each color represents a different access requirement. It is laid out in its optimal layout without redundancy on top. A layout with redundancy and minimal EST is shown at the bottom*

the relevant data units, update the access pattern and run a few iterations of our algorithm to get a better layout. In other words, our algorithm is incremental and can be used for dynamic data sets, which also might be a result of scene editing and modification.

~~**Limitations and future work:**~~ Our cost model does not take account distance between access requirements. We only take into account distance between data units in the same access requirement and we do not consider the seek time between access requirements. If we do take this account in our model, then if we are given information as to which access requirement is more likely to be used before or after another access requirement we would have an even more accurate model for seek time.

References

AGRAWAL, N., PRABHAKARAN, V., WOBBER, T., DAVIS, J. D., MANASSE, M., AND PANIGRAHY, R. 2008. Design tradeoffs for ssd performance. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, USENIX Association, Berkeley, CA, USA, 57–70.

ALIAGA, D., COHEN, J., WILSON, A., BAKER, E., ZHANG, H., ERIKSON, C., HOFF, K., HUDSON, T., STUERZLINGER, W., BASTOS, R., WHITTON, M., BROOKS, F., AND MANOCHA,

D. 1999. MMR: An interactive massive model rendering system using geometric and image-based acceleration. In *Proceedings Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, 199–206.

DOMINGO, J. S. 2014. Ssd vs. hdd: What's the difference. *PC Magazine* (February).

HOPPE, H. 1996. Progressive meshes. In *Proceedings SIGGRAPH*, 99–108.

HOPPE, H. 1997. View-dependent refinement of progressive meshes. In *SIGGRAPH*, 189–198.

HOPPE, H. 1998. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings IEEE Visualization*, 35–42.

JIANG, S., SAJADI, B., , AND GOPI, M. 2013. Single-seek data layout for walkthrough applications. *SIBGRAPI 2013*.

JIANG, S., SAJADI, B., IHLER, A., AND GOPI, M. 2014. Optimizing redundant-data clustering for interactive walkthrough applications. *CGI 2014*.

LINDSTROM, P., AND TURK, G. 1999. Evaluation of memoryless simplification. *IEEE Transactions on Visualization and Computer Graphics 5*, 2 (April-June), 98–115.

LLOYD, S. 1982. Least squares quantization in pcm. *Information Theory, IEEE Transactions on 28*, 2 (Mar), 129–137.

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

PATTERSON, D. A., GIBSON, G., AND KATZ, R. H. 1988. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, ACM, SIGMOD '88, 109–116.

RIZVI, S., AND CHUNG, T.-S. 2010. Flash ssd vs hdd: High performance oriented modern embedded and multimedia storage systems. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, vol. 7, V7–297–V7–299.

SAJADI, B., JIANG, S., HEO, J., YOON, S., AND GOPI, M. 2011. Data management for ssds for large-scale interactive graphics applications. In *I3D '11 Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, 175–182.

SAXENA, M., AND SWIFT, M. M. 2009. Flashvm: Revisiting the virtual memory hierarchy. In *Proc. of USENIX HotOS-XII*.

SHAFFER, E., AND GARLAND, M. 2001. Efficient adaptive simplification of massive meshes. In *Proc. IEEE Visualization*, Computer Society Press, 127–134.

SILVA, C., CHIANG, Y.-J., CORREA, W., EL-SANA, J., AND LINDSTROM, P. 2002. Out-of-core algorithms for scientific visualization and computer graphics. In *IEEE Visualization Course Notes*.

VARADHAN, G., AND MANOCHA, D. 2002. Out-of-core rendering of massive geometric datasets. In *Proceedings IEEE Visualization 2002*, Computer Society Press, 69–76.

YOON, S.-E., AND LINDSTROM, P. 2006. Mesh layouts for block-based caches. *IEEE Trans. on Visualization and Computer Graphics ~~(Proc. Visualization)~~ 12*, 5, 1213–1220.

YOON, S., LINDSTROM, P., PASCUCCI, V., AND MANOCHA, D. 2005. Cache oblivious mesh layouts. *ACM SIGGGRAPH 2005*.