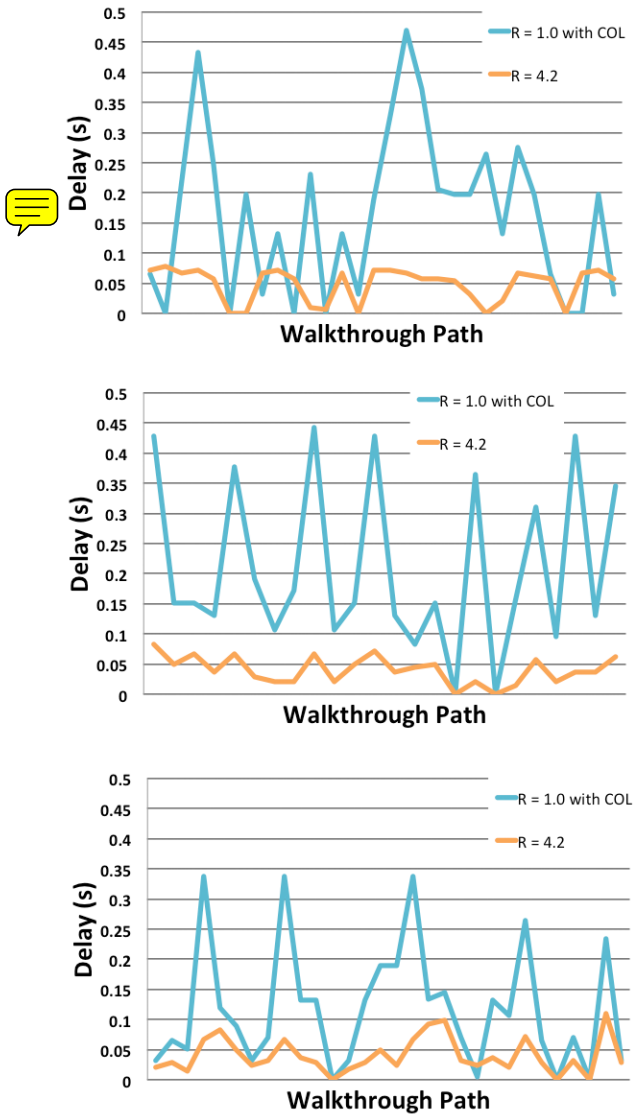

Figure 7: *Statistics of delays caused by the fetching processes for the City model (top), the Boeing model (center), and the Urban model (bottom), with and without redundancy.*

tor reaches a certain threshold. This helps us create a data layout with arbitrary redundancy factor without worrying about exceeding the capacity of secondary storage devices. We use this fact to test different redundancy factors and see their results. In Figure 8, 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. We can see that the rate of this decrement is not constant, and the benefits we gain at beginning are larger than the ones we get later. This implies that most of the performance improvement resides at the earlier phase of raising redundancy factor. This implies that it is worth it to limit the redundancy factor used because after a certain point you are using much more secondary storage space without improving seek time by much. It also implies that our algorithm dramatically reduces seek time in practice by using only small redundancy factors.

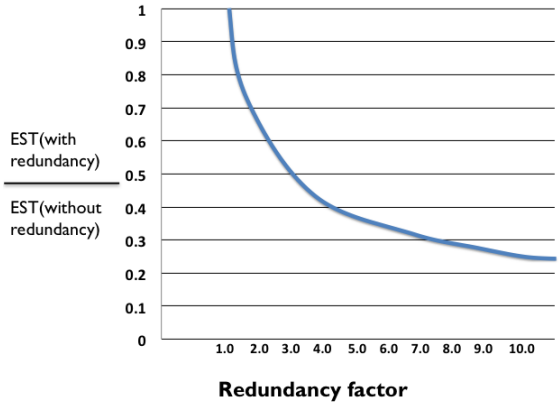

Figure 8: *Plot of the ratio of the EST of layout with redundancy over the EST of cache-oblivious mesh layout without redundancy.*

6 Analysis and Comparisons over Prior methods

In the algorithm, we make a list of data units that will reduce seek time by just moving them. We perform these moves first before ~~doing any copying~~. This initial step will produce a better solution than proposed by [Yoon et al.] without adding extra units. This is because our algorithm will consider cases where data units are close to each other but ~~in~~ hierarchically different blocks ~~with the cache oblivious layout~~. To show this, consider a case where we have two access requirements of 5 data units each. Figure 9 shows an example of that kind of layout. In the middle of that figure is the result of using the cache oblivious layout. Because hierarchically it would arrange the units in each block and then arrange the blocks, it does not detect that the units with the black access requirement can be grouped together. ~~When our algorithm is run, it~~ would ~~find that it can~~ shorten the black access requirements without adding redundancy ~~so it would do that first~~, as shown in the bottom of that figure.

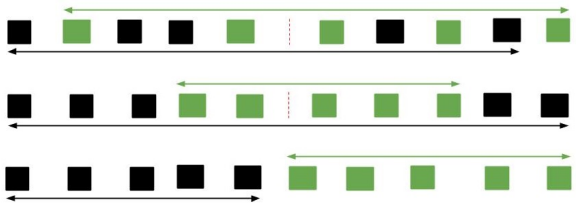

Figure 9: *Example of two access requirements of 5 data units each. The red line represents the boundary between blocks in the cache oblivious layout hierarchy. The original layout (top), cache-oblvious layout (middle), as well as the layout after running our algorithm (bottom) is shown.*

~~Existing algorithms for the data layout problem do not consider redundancy.~~ Even if we find a polynomial time algorithm that ~~solves~~ the data layout ~~problem, we can actually achieve a seek time better than the optimal one without redundancy~~. We have such an example with figure 10. As can be seen in the figure, the total seek time is 11 units originally. Without redundancy, the seek time is reduced to 9 units, as found through a brute-force search. With redundancy, the total seek time is the minimum required which is 8 units. While a reduction from 9 to 8 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.

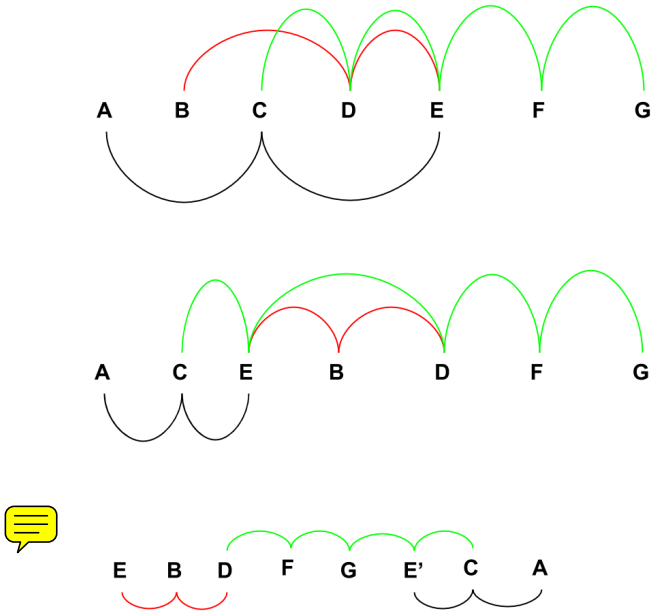

Figure 10: *Data Units with varying access requirements (top) as well as its optimal layout without redundancy (middle) and its optimal layout with redundancy (bottom)*

7 Conclusion and Future Work

~~We have shown that we have an algorithm with an efficient running time and storage space for the Data Layout Problem. It achieves significant results analytically and experimentally. When walking through an extremely detailed 3D model, this algorithm can be used to ensure that the performance will not suffer. If we give the algorithm the proper access requirements with this 3D model, then the performance will be even better.~~

~~This leads to a logical extension of this work. Since we have a good algorithm that takes over once we know the access requirements, we should figure out how to ensure there are good access requirements to begin with. One idea on how to ensure this is to check the usage history of an application and group data units together if they are accessed together with high probability. This could even be done dynamically in the sense that after a certain amount of usage and repeating on a regular basis, you recompute the optimal access requirements and then use that to recompute the optimal layout.~~

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 MEENAKSHISUNDARAM, G. Single-seek data layout for walkthrough applications. *SIBGRAPI 2013*.

JIANG, S., SAJADI, B., IHLER, A., AND MEENAKSHISUNDARAM, G. 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.

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 MEENAKSHISUNDARAM, G. Data management for ssds for large-scale interactive graphics applications. *ACM SIGGGRAPH 2011*.

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. Cache oblivious mesh layouts. *ACM SIGGGRAPH 2005*.