Eurographics Conference on Visualization (EuroVis) (2015)
E. Bertini, J. Kennedy and E. Puppo (Editors)

Dynamic Scheduling for Progressive Large-Scale Visualization

M. Flatken[1] and A. Berres[2] and J. Merkel[2] and I. Hotz[1] and A. Gerndt[1] and H. Hagen[2]
[1]German Aerospace Center, Germany
[2]University of Kaiserslautern, Germany

Abstract

The ever-increasing compute capacity of high-performance systems enables scientists to simulate physical phenomena with a high spatial and temporal accuracy. Thus, the simulation output can yield dataset sizes of many terabytes. An efficient analysis and visualization process becomes very difficult especially for explorative scenarios where users continuously change input parameters. Using a distributed rendering pipeline may relieve the visualization frontend considerably but is often not sufficient. Therefore, we additionally propose a progressive data streaming and rendering approach. The main contribution of our method is the importance-guided order of data processing for block structured datasets. This requires a dynamic scheduling of data chunks on the parallel post-processing system which has been implemented by using an R-Tree. In this paper, we demonstrate the efficiency of our implementation for view-dependent feature extraction with varying viewpoints.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.1]: Hardware Architecture—Parallel processing; Computer Graphics [I.3.2]: Graphics Systems—Distributed/network graphics;

1. Introduction

The evolution of high performance computing (HPC) hardware and software enables the simulation of physical phenomena with a very high resolution and accuracy. Simulation solvers carry out parallel algorithms in order to speedup the computation. For this purpose, the simulation domain has to be decomposed in many smaller data chunks, which can then be distributed among the available processors. After the simulation is done, the data chunks including simulation results are often stored in large scale multi-block datasets. Thus, a multi-block dataset consists of lots of files and each file includes a grid structure and the simulation results.

These massive datasets imply a significant challenge for efficient analysis and visualization. Since loading and processing of the data on local computers is no longer reasonable, remote filtering on HPC systems becomes more and more popular. Although processed in parallel, the time required to complete a filtering operation and to produce a final image is still often too high for interactive exploration. Therefore, as soon as parts of the extraction data are available on the back-end, they can be streamed to the front-end computer to reduce response times considerably. The requested data can be presented even faster when only data blocks relevant for the query are processed. And eventually, most useful information may be visualized first if the order of processing is importance-guided.

These streaming improvements require dynamic and efficient scheduling methods. In order to enable importance-guided processing, an importance function can be introduced to prioritize the data distribution to parallel processing elements. To retrieve related data blocks effectively, a query data structure should consider the following aspects (a) Data query: which data is relevant to the query, (b) Viewing position: which order allows for an efficient first inspection while data is progressively added to the view. Our dynamic scheduling approach is built on an R-Tree data structure. It supports a wide range of user queries typical for scientific data exploration. Furthermore, it is especially well-suited for applications processing large scale multi-block datasets.

2. Related Work

Several distributed post-processing applications and frameworks have been developed to facilitate interactive data exploration for large-scale data. The two most common tools are ParaView [HAL04] and VisIt [CBW*12]. Both of them rely on the Visualization Toolkit (VTK) [SML96] and make use of data parallelism to speedup the visualisation process.

DOI: 10.2312/eurovisshort.20151122

Viracocha is a framework which focuses on interactivity within virtual environments [GHW*04]. There are many examples for specific use cases and algorithmic challenges in distributed and parallel visualization [BCH12]. Ahrens et al. presented a modular and prioritized data streaming architecture for out-of-core data processing integrated into VTK and ParaView [ADM*07]. Their ideas are very similar to ours but do not deal with parallelism. Examples for view-dependent, occlusion efficient and parallel isosurface extraction can be found in [LH98] [CFSW01] [KcC*01] [ZBR02].

Besides data parallelism and out-of-core processing, I/O saving methods for query-driven visualization systems are investigated. Fisher et al. [Fis11] propose to use a database structure while Atasanov et al. [ASW12] adapt query sets to a given domain decomposition. Chauduri et al. [CLSP13] use an integral distribution stored in an octree to respond to range distribution queries.

Our work combines data parallel processing using distributed resources with an R-tree data structure to select relevant data and to define an optimized order for processing.

3. Dynamic Scheduling for Large-Scale Data

A common way to process large-scale datasets is to exploit data and task parallelism. However, parallelism alone is not sufficient to achieve interactive exploration. Dynamic scheduling approaches combined with progressive data streaming can help to improve the system response time (time expired until first results are visible). This gain in performance results from discarding unnecessary data and from processing data in an order which is dependent on an importance function. We achieved the described goals by integrating a dynamic scheduling approach into the Viracocha framework [GHW*04].

The Viracocha framework is based on the master/slave paradigm to distribute workload for parallel processing. The master process (referred to as scheduler) splits a job request into small independent work items. These work items are then distributed to slave (referred to as worker) processes. Until now this scheduling order has been static and did not change during a job request, e.g., change of viewing perspective. Enabling an efficient and dynamic scheduling order requires a data structure with the following features:

1. *Spatial filtering*: select the blocks of a multi-block dataset based on spatial information to decrease the I/O load, which is often a bottleneck when interactivity is required
2. *Importance filtering*: select the block of a multi-block dataset, which is most important (multi-dimensional) e.g. the block with the highest turbulence
3. *Efficiency*: fast construction for block counts typical in CFD simulations, and fast replies for queries to guarantee parallel efficiency and scalability

Data Structure For the dynamic scheduling approach, we need a data structure to manage the blocks of a multi-block dataset. Thereby, each data block is described by its 3-dimensional bounding box and optional importance values. These importance values are either given in the dataset (e.g. the scalar ranges) or can be calculated on-the-fly. Tree-based data structures are an efficient way to represent spatial partitions and hierarchies. Since typical block counts are in range of a few thousand the performance of different tree variants should be nearly identical. It is more important that the data structure can deal with overlapping bounding boxes and supports multiple dimensions for future enhancements.

Octrees [Mea82] are well-suited for axis-aligned bounding boxes, but they are not suited for a multi-dimensional space. In contrast, binary space-partitioning methods, such as kD-trees [OMSd87], are much more suited for multi-dimensional data. However, they are designed for point data and require adaptation to bounding boxes. Hence we employ a bounding volume hierarchy, namely an *R-Tree* (rectangle tree) [Gut84], as our data structure. An R-Tree consists of nested bounding volumes (rectangles in 2D, cuboids in 3D). Beckmann et al. [BKSS90] introduce R*-Trees, a variation of R-Trees with different insertion and splitting strategies. R*-Trees have improved query efficiency at slightly more costs for construction. R-Trees are dynamic data structures and can be constructed either gradually by progressive insertion of blocks, or en-bloc using the *sort-tile-recursive* (STR) algorithm [LLE97].

In addition of being perfectly suited to store the bounding boxes of a multi-block dataset, R-Trees are always height-balanced and support dynamic insertion and deletion of nodes. These features make them particularly well-suited for our dynamic scheduling. Additionally, it allows indexing of multi-dimensional information.

Workflow In a pre-processing step, an initial 3-dimensional R-Tree is generated for each time step of a multi-block dataset (Figure 1). These constructed R-Trees are then encoded and stored to disc for subsequent queries.

When a user query is received, all data blocks matching the spatial request are stored in a selection. Currently our system supports the following spatial requests:

- Intersection with a plane: All blocks whose bounding box intersect with the plane are stored in a worklist.
- Intersection with a volume: All blocks whose bounding box lie partially or fully inside the defined volume are stored in a worklist.

Based on the selection within the worklist, a new set of 3-dimensional R-Trees (one per time step) is generated on-the-fly. Optionally a set of importance values per block can be used to build multi-dimensional R-Trees. An example for importance values are the min. and max. pressure value per block. The scheduler now uses these R-Trees as data structure to distribute the workload among the workers for paral-

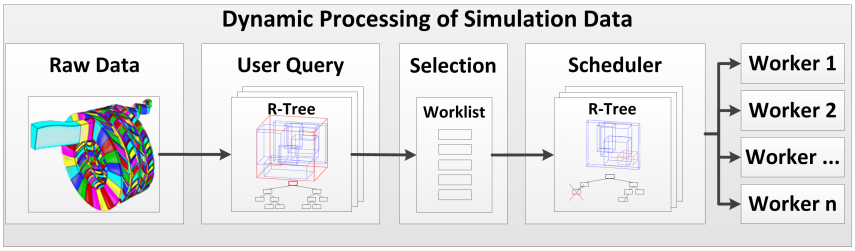

Figure 1: An initial set of R-Trees is generated once for the raw data. These R-Trees are used to select relevant data for a user query. From this selection, a new set of trees is generated and used by the scheduler to define a processing order.

lel processing. In case of view-dependent processing, it locates the block closest to the camera position and forwards data and algorithm parameters to the workers. The tree node is then dynamically removed and the scheduler continues with the closest block in the next tree. This round robin process continues until no nodes are left in the trees. Whenever, a worker has finished its algorithm on a data block, the result e.g. a portion of an isosurface is transferred to the front-end application to be progressively updated and rendered.

4. Results

To benchmark our dynamic scheduling approach we have used a small remote HPC system consisting of four workstations. Each of these workstations has two Intel®Xeon®X5670 hexa-core processors, and 48 GB main memory. The interconnect between the nodes is a 40 Gbit QDR-Infiniband network. The connection to the visualization front-end is 1 Gbit Ethernet.

We used a compressor turbomachinery computational fluid dynamics (CFD) dataset for our evaluation. This unsteady multi-block simulation dataset consists of 1792 time steps. Each of these time steps is composed of 1679 blocks. The data set is 2.2 TB large including all scalar and vector fields. For this work, 50 time steps were used resulting in 83950 data blocks to be scheduled for processing.

Tree Parameters Before comparing our dynamic processing to static-order processing, we conducted preliminary tests to optimize the tree parameters. As outlined in Section 3, different strategies for node insertion can be used. Therefore, R-Trees can be constructed en bloc using the STR algorithm, or through progressive insertion of nodes. We varied the node capacity, leaf capacity, node insertion methods, and construction methods. Finally, we measured the total time required to build all R-Trees, schedule and process all data, and render the extracted isosurface. Table 1 presents the results of our measurements. As depicted, the first row performs best among all combinations. Therefore, all further benchmarks utilize this configuration.

Plane and Volume Extraction When intersecting a cutting plane or region-of-interest with the entire dataset, it makes a significant difference whether all blocks or only a selection of relevant blocks is processed. Table 2 depicts a comparison between processing pre-selected blocks using the spatial filtering (see Figure 1) and processing all blocks. The number of selected blocks to be processed with our method was reduced to approximately 12% for the plane and 14% for the defined volume. This spatial filtering operation reduced the I/O load massively. The time required to extract a slice was reduced to less than 7% and for the volume to 16% of the time required to process all blocks.

node cap.	leaf cap.	insert	construction	duration
10	10	linear	STR	67.556s
100	10	linear	STR	69.941s
10	10	quad.	progr. insert	78.541s
10	10	quad.	STR	70.541s
5	10	R*	STR	70.687s
10	10	R*	progr. insert	69.072s
100	100	R*	progr. insert	106.223s

Table 1: Comparison of processing times for different insertion/construction strategies with varying node/leaf capacity.

	Slice Extraction	Volume Extraction
# data blocks	8507	12154
total time without pre-selection	43.73s	124.23s
total time with pre-selection	2.96s	20.39s

Table 2: Pre-selecting relevant blocks has a big impact on I/O and computation costs due to discarding unnecessary data.

Scalability Analysis We have evaluated how our scheduling approach scales with an increasing numbers of workers. In Table 3, we compared average assignment time and total computation time. The assignment time using dynamic scheduling is dominated by the time needed to locate the data blocks closest to the camera. The overhead for traversing the tree compared to gather blocks from a list (static scheduling) is in range of microseconds and does not significantly influence the scaling. As depicted, increasing the number of workers speeds up the computation, but the relation between number of workers and processing speed is not linear. The main bottleneck is caused by I/O performance as data is stored on a remote file system accessed via Ethernet.

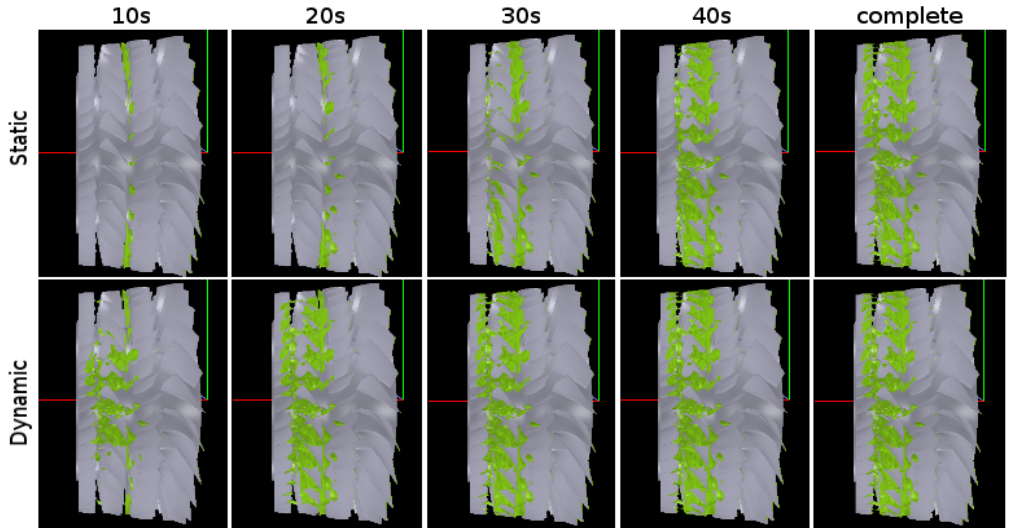

Figure 2: Comparison of the rendering progress of an isosurface extracted from a single time step (density 1.3, time step 0) using dynamic processing vs. static processing. Screenshots were taken at 10s, 20s, 30s, 40s, and of the completed surface. With dynamic scheduling (row 2), the user gets a detailed visualization more quickly than with static scheduling (row 1).

	avg assignment time		total time	
# processes	**static**	**dynamic**	**static**	**dynamic**
12	2.445ms	2.490ms	205.298s	209.732s
24	1.258ms	1.293ms	105.711s	109.270s
48	0.826ms	0.831ms	73.617s	70.506s

Table 3: Scalability test for static and dynamic scheduling.

Visual Analysis To visually evaluate our method, we query and render an isosurface of flow density inside the compressor. We compared the static and dynamic view-dependent scheduling approach. The computation was started from an initial perspective which was immediately changed to the opposite side of the compressor to cause maximal variation in processing order. To visually measure image progress, we stopped the animation at the first time step to compare rendering results. As depicted in Figure 2, the time until all visible parts of the isosurface are rendered is much shorter for dynamic than for static scheduling. Moreover, the order in which data blocks for the isosurface are processed and rendered lets the user examine the closest surface part more quickly. Table 4 depicts a quantitative comparison of the time-dependent progress each method makes in rendering through a pixel-wise comparison to the final image. The dynamic method finished rendering the visible parts at 42.5 s whereas the static method takes almost twice as long. Assuming a random order of blocks during static processing, this result was expected since approximately half of the data is occluded behind other surfaces. Additionally, the table depicts that 91% of the isosurface are calculated within 20s and the remaining 10% needs the same time. This is due to an suboptimal processing order caused by occlusion of blocks.

Strategy	**10s**	**20s**	**30s**	**40s**	**final**
static	8.7%	16.4%	46.4%	83.8%	81.0s
dynamic	45.5%	90.6%	99.4%	99.99%	42.5s

Table 4: Pixel-wise comparison between the final isosurface and the currently rendered part at different timings.

5. Conclusion and Future Work

We have presented a novel and efficient dynamic scheduling method for progressive large-scale data visualization. It is based on an R-Tree data structure, which incorporates an importance function to determine the processing order. The view-dependent streaming technique admits high performance exploration of relevant parts before the entire dataset has been processed. While the user can start almost immediately with the analysis of user-centred regions-of-interest, the surrounding area is progressively updated. The R-Tree data structure proves to be very efficient for the filtering of multi-block datasets. Spatial queries such as cutting planes or extracting sub-volumes results in a drastic speedups (6-14x) due to saving of I/O and computation costs. Using the view-dependent isosurface computation, all relevant parts of the data are displayed in less than half the time of a static approach.

In the future, we want to consider other simulation data types. In general, the R-Tree seems to be very promising. But there is still space for improvements of the current query data structure implementation. We also want to investigate and evaluate the scalability of our approach in greater detail. Therefore, we will analyze the runtime behavior of our application on larger cluster systems.

References

[ADM*07] AHRENS J. P., DESAI N., MCCORMICK P. S., MARTIN K., WOODRING J.: A modular extensible visualization system architecture for culled prioritized data streaming. In *Proceedings of the 2007 Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* (Jan. 2007), vol. 6495, pp. 64950I–1 – 64950I–12. 2

[ASW12] ATANASOV A., SRINIVASAN M., WEINZIERL T.: Query-driven parallel exploration of large datasets. In *Proceedings of the IEEE Symposium on Large Data Analysis and Visualization (LDAV)* (Seattle, Washington, USA, Oct. 2012), pp. 23 –30. 2

[BCH12] BETHEL E. W., CHILDS H., HANSEN C.: *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*, 1st ed. Chapman & Hall/CRC, 2012. 2

[BKSS90] BECKMANN N., KRIEGEL H.-P., SCHNEIDER R., SEEGER B.: The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)* (Atlantic City, New Jersey, USA, 1990), SIGMOD '90, ACM, pp. 322–331. 2

[CBW*12] CHILDS H., BRUGGER E., WHITLOCK B., MEREDITH J., AHERN S., PUGMIRE D., BIAGAS K., MILLER M., HARRISON C., WEBER G. H., KRISHNAN H., FOGAL T., SANDERSON A., GARTH C., BETHEL E. W., CAMP D., RÜBEL O., DURANT M., FAVRE J. M., NAVRÁTIL P.: Visit: An end-user tool for visualizing and analyzing very large data. In *High Performance Visualization - Enabling Extreme-Scale Scientific Insight*. Taylor and Francis, Oct 2012, pp. 357–372. 1

[CFSW01] CHIANG Y.-J., FARIAS R., SILVA C. T., WEI B.: A unified infrastructure for parallel out-of-core isosurface extraction and volume rendering of unstructured grids. In *Proceedings of the Parallel and Large-Data Visualization and Graphics* (Oct 2001), pp. 59–151. 2

[CLSP13] CHAUDHURI A., LEE T.-Y., SHEN H.-W., PETERKA T.: Efficient range distribution query in large-scale scientific data. In *Proceedings of the IEEE Symposium on Large Data Analysis and Visualization (LDAV)* (Atlanta, Ga, USA, Oct 2013), pp. 125–126. 2

[Fis11] FISHER D.: Incremental, approximate database queries and uncertainty for exploratory visualization. In *Proceedings of the IEEE Symposium on Large Data Analysis and Visualization (LDAV)* (Providence, RI, USA, Oct 2011), pp. 73–80. 2

[GHW*04] GERNDT A., HENTSCHEL B., WOLTER M., KUHLEN T., BISCHOF C.: Viracocha: An efficient parallelization framework for large-scale cfd post-processing in virtual environments. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing* (Washington, DC, USA, 2004), SC '04, IEEE Computer Society, p. 50. 2

[Gut84] GUTTMAN A.: R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM International Conference on Management of Data* (New York, NY, USA, 1984), SIGMOD '84, ACM, pp. 47–57. 2

[HAL04] HENDERSON A., AHRENS J., LAW C.: *The ParaView Guide*. Kitware Clifton Park, NY, 2004. 1

[KcC*01] KURC T., ÇATALYÜREK U., CHANG C., SUSSMAN A., SALTZ J.: Visualization of large data sets with the active data repository. *IEEE Comput. Graph. Appl. 21*, 4 (July 2001), 24–33. 2

[LH98] LIVNAT Y., HANSEN C.: View dependent isosurface extraction. In *Proceedings of visualization* (Research Triangle Park, NC, USA, 1998), IEEE Computer Society, pp. 175 – 180. 2

[LLE97] LEUTENEGGER S. T., LOPEZ M. A., EDGINGTON J.: STR: A simple and efficient algorithm for R-tree packing. In *Proceedings of the 13th International Conference on Data Engineering* (1997), IEEE, pp. 497–506. 2

[Mea82] MEAGHER D.: Geometric modeling using octree encoding. *Computer graphics and image processing 19*, 2 (1982), 129–147. 2

[OMSd87] OOI B. C., MCDONELL K. J., SACKS-DAVIS R.: Spatial kd-tree: an indexing mechanism for spatial databases. In *Proceedings of the IEEE International Computer Software and Applications Conference (COMPSAC)* (1987), vol. 11, IEEE Computer Society, pp. 433–438. 2

[SML96] SCHROEDER W. J., MARTIN K. M., LORENSEN W. E.: The design and implementation of an object-oriented toolkit for 3d graphics and visualization. In *Proceedings of the 7th Conference on Visualization* (Los Alamitos, CA, USA, 1996), VIS '96, IEEE Computer Society Press, pp. 93–100. 1

[ZBR02] ZHANG X., BAJAJ C., RAMACHANDRAN V.: Parallel and out-of-core view-dependent isocontour visualization using random data distribution. In *Proceedings of the Symposium on Data Visualisation* (Aire-la-Ville, Switzerland, Switzerland, 2002), VISSYM '02, Eurographics Association, pp. 9–ff. 2