An Efficient System for Multi-perspective Imaging and Volumetric Shape Analysis

Eugene Borovikov, Alan Sussman and Larry Davis
UMIACS and Department of Computer Science
University of Maryland, College Park, MD 20742
{yab,als,lsd}@cs.umd.edu

Abstract

We present a high performance system for efficient multi-perspective image analysis on very large image datasets, implemented as a customized extension of the Active Data Repository (ADR) object-oriented framework. The resulting system provides a flexible framework for handling multi-perspective video sequences in any parallel or distributed computing environment that can support ADR. We have employed the framework to produce an efficient volumetric shape analysis application implementation. Initial performance results show that using an effective data distribution algorithm to produce good load balancing allows the ADR implementation to achieve scalable high performance.

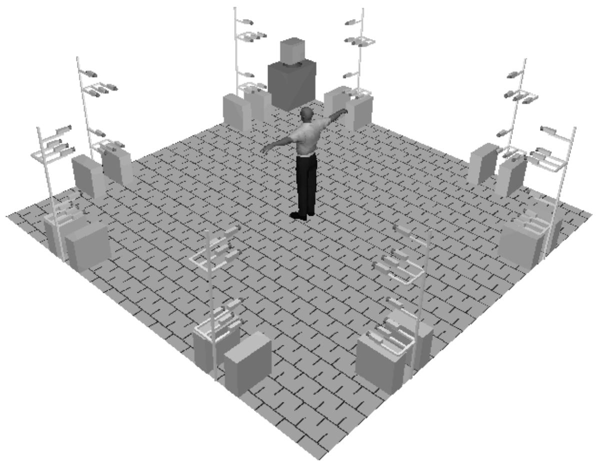
Figure 1. Keck Lab model

1. Introduction

Traditionally, research in image processing and computer vision systems has dealt with images or sequences of images shot from a single camera (mono-vision) or two cameras (stereo-vision). Digital video streams for those systems, although potentially large, do not require any special storage handling - generic file systems suffice. Falling hardware prices along with the modern growth of interest in vision-based interfaces, virtualized reality and ubiquitous visual computing gave a rise to *multi-perspective vision*, i.e. systems with multiple (more than two) cameras usually spread around a room, that shoot the scene from multiple perspectives. Multiple video streams produced this way generate very large amounts of image data that can become unmanageable unless there is an efficient way to store, catalog and process them.

Consider, for example, the Keck Lab (Figure 1) at the University of Maryland [5]. It consists of 64 cameras synchronously capturing video streams at a rate of up to 85 frames per second, with each frame being $644 \times 484 \times 1$ bytes. One minute of such multi-perspective video requires about 95 GBytes of storage. Managing such quantities of data on a single PC or workstation, while feasible, will have serious performance problems for the foreseeable future. To improve performance, the data should be stored and processed on a parallel machine or a cluster of workstations employing an efficient software system for providing the desired functionality.

To aid in efficiently implementing storage and processing of multi-perspective images, we employ an object-oriented framework called the Active Data Repository (ADR) [3, 4, 7, 8], that has been developed at the University of Maryland for managing and processing large amounts of scientific data on a parallel or distributed system. In this paper we describe how to customize ADR for building an efficient and flexible framework for storing multi-perspective image data and performing visual analysis. Further, we describe a volumetric shape recovery and analysis application that is based on this framework, and give some general guidelines on developing multi-perspective imaging applications using the system.

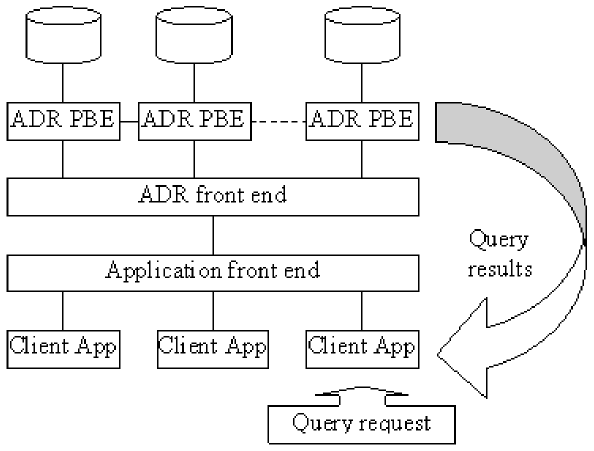

Figure 2. ADR-based application structure

2. Multi-perspective Imaging System

The core of our multi-perspective imaging framework is based on ADR customized for managing multi-perspective image sequences. As a tool, ADR is meant to be customized for a given application. With a number of other multi-perspective imaging applications in mind, we have built a relatively generic multi-perspective imaging framework based on ADR. A typical ADR-based application has the structure shown in Figure 2.

ADR is an object-oriented framework designed to efficiently integrate application-specific processing with the storage and retrieval of multi-dimensional datasets on a parallel machine or cluster of workstations with a disk farm. ADR consists of a set of modular services, implemented as a C++ class library, and a runtime system. Several of the services allow customization for user-defined processing. An application implemented using ADR consists of one or more *clients*, a *front-end process*, and a customized *back-end* The front-end interacts with clients, translates client requests into queries and sends one or more queries to the parallel back-end. Since the clients can connect and generate queries in an asynchronous manner, the existence of a front-end relieves the back-end from being interrupted by clients during processing of queries. The back-end is responsible for storing datasets and carrying out application-specific processing of the data on the parallel machine, eventually returning results to the clients.

The ADR framework has been used to provide support for storage and processing of large datasets in a wide range of scientific application areas. We have used ADR to develop applications in diverse fields, including coupling of multiple scientific simulation codes [10], analysis and processing of satellite datasets [13], analysis and visualization of microscopy data [1], and volume rendering and iso-surface rendering to support visualization of very large datasets [9].

The image analysis framework discussed in this paper customizes ADR's *parallel back-end* (PBE) and *front-end* (FE). The implementation also provides a tool for loading and cataloging multi-perspective sequences of image data. This leaves the image analysis application developer with the task of providing an application-specific front-end and a client. However, the application developer can also further customize the PBE and FE to meet the needs of his/her application that works on multi-perspective image data. An application developer is allowed to both provide additional application-specific image processing functions and also implement other additions to ADR's customizable services.

The customized PBE and FE for the framework are described in Sections 2.2 and 2.3, respectively, but first we describe the multi-perspective image datasets.

2.1. Multi-perspective image data and ADR

A multi-perspective video sequence constitutes a single image dataset. A data element (*chunk*) is a single image whose attributes are $<$*camera index*, *time index*$>$. To populate the database, a user employs the *data loader* tool supplied with the framework. The data loader takes multi-perspective data stored in flat files, computes chunk placement information for the disks available in the parallel system according to some user-supplied strategy (ADR supplies a default strategy based on space-filling curves), moves the chunks to their destination database files, and indexes them accordingly. Optionally, the user can move data files to the disks manually (if the chunks are already declustered across the data files) and provide a custom index. A multi-perspective data set also requires camera configuration data supplied as meta-data files. The configuration information is also supplied by the user.

Once image datasets are loaded into and registered with ADR, the system can satisfy requests from queries produced by client applications. Queries are processed by both an application FE and ADR FE before being executed on the PBE. The FE stores metadata information about datasets and reformulates user queries in terms of standard or custom registered indexes. In general, the FE acts as an intermediary between clients producing requests and the back-end that stores data and processes requests on stored datasets. The application FE is responsible for any application-specific communication protocols with the client, and translates client requests into a standard form that the ADR back-end requires. The ADR FE schedules translated requests onto the back-end nodes, and verifies that the requests are properly formed.

There is one important restriction on the user requests imposed by ADR: operations on data elements have to be *associative* and *commutative*. This is acceptable for many applications (e.g. volumetric reconstruction), since aggregation of multi-perspective data often complies with those requirements. We now describe the ADR customization provided by the image analysis framework.

2.2. The ADR Parallel Back-end

The image datasets are distributed on and managed by parallel back-end (PBE) nodes, which may be part of a distributed memory parallel machine or a cluster of workstations. The back-end is responsible for storing the datasets and carrying out the requested input data retrieval, projection to the output space, and aggregation operations. The default index service maps a four-dimensional query region (three for space, one for time) into a set of relevant chunks. This is accomplished via a camera index lookup computed for each query region as needed. The default projection maps the <camera-index, time-index> pair to the <time-index>, to allow aggregation across all cameras that view a spatial region at a given time. No default aggregation operation is supplied, since aggregation is application dependent. Any default services can be customized by the application developer.

2.3. Application and ADR Front-end

The application front-end interacts with clients via an application-specific communication protocol, and translates queries into a common format for the ADR front-end, which will send the reformulated query to the PBE. The application FE's purpose is to maintain meta-information about datasets (e.g., video sequence name, number of views, frame rate) and respond to queries coming from client applications. The ADR FE uses its *query interface* service to interact with application front-ends, and its *query submission* service to schedule multiple queries received from one or more application front-ends and submit them in batches to the PBE.

3. A Volumetric Shape Analysis Application

We illustrate the utility of the proposed framework for multi-perspective imaging applications via a parallel volume reconstruction and 3D shape analysis application implemented using the framework. The application uses multi-perspective image data to reconstruct the volumetric shape of a foreground object. The application then can render the volume from an arbitrary vantage point at any point in time and allow users to analyze the 3D shape by requesting region-varying resolution in the reconstruction. The volumetric reconstruction is based on the distributed volume intersection technique described in [2, 5].

The reconstructed volume is represented as an occupancy map encoded by an octree, whose space requirements are exponential in the depth of reconstruction. To compactly encode the octree, in order to minimize the network bandwidth required to send volumes to clients, it is encoded as a byte array in DFS-order, as was proposed in [2]. We have made an improvement to that encoding by noticing that it is sufficient to keep only leaf nodes, encoding them as bit-arrays whenever possible (similar approaches are discussed in [12] and [11]). As a result, a typical volumetric reconstruction of depth 8 requires about 50 KB for the octree, while the previously proposed system required about 200 KB. However, the octree handling routines have become more complex and expensive.

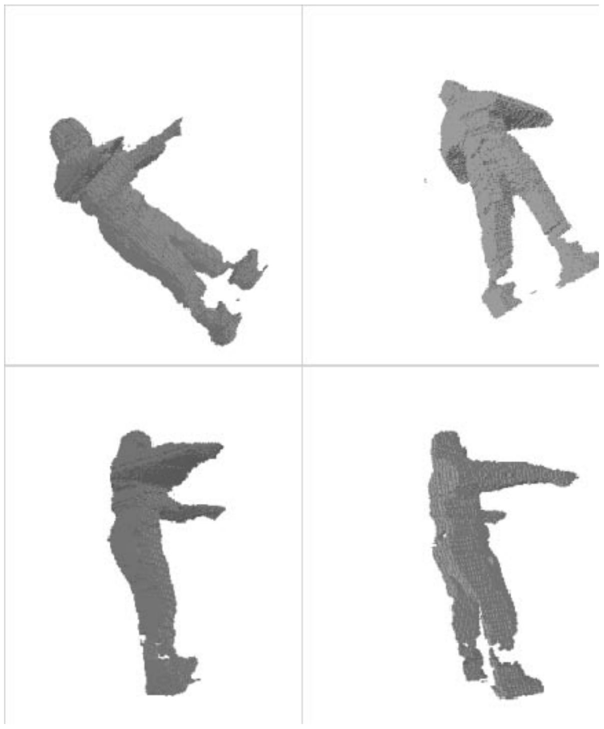
Figure 3. Reconstructed volume

A database inquiry involves specifying a 3D region, a time range and a volumetric reconstruction resolution. The query result is a multi-resolution reconstruction of the foreground object region lying within the query region. This information can be used for 3D shape analysis and 3D tracking.

The *client* application is a program run by a user. It generates queries based on user requirements, submits them to the application FE described earlier, and receives data from the PBE nodes for further application-specific post-processing. The volume reconstruction application implements a GUI client in C++. The client provides a user interface for specifying a user query and presents the query results via a volumetric viewer implemented in OpenGL. Figure 3 shows a typical real-world 3D reconstruction rendered from a number of view points.

A user query is specified by *Volume*(*time-range*, *space-range*, *reconstruction-resolution*). The result of a query is an occupancy map of the foreground object in the given 3D region at a given resolution over the specified time period. A series of such queries produces a union of volumetric reconstructions, each at a different resolution. The resulting shape is displayed in a client window from a specified perspective.

4. Experiments

Our initial experiments target the volumetric shape analysis application. We have considered scenarios varying the data distribution of the input image dataset across the disks of the parallel machine and varying the number of processors in the ADR parallel back-end to evaluate the system's overall performance and scalability.

The input attribute space is two-dimensional, consisting of <*camera-index*, *time-index*> pairs, and most of the datasets are likely to have $camera\_max\_index \ll time\_max\_index$, with the number of views comparable with the number of processors. With this observation in hand, we have considered various strategies for distributing data (images) onto the disks of the parallel or distributed machine.

It would seem "natural" to distribute data in a *view per processor* fashion. This trivial distribution might provide a reasonable workload balance for the cases where $views\_count = processors\_count$ and all views are involved in the query. It balances work poorly when only a few views are queried, for example when only a part of the scene is to be reconstructed and not all cameras can view that part of the scene. We therefore require a more sophisticated approach.

A *round robin distribution in a single data dimension* also runs the risk of distributing data similar to the *view per node* approach. The worst case for this distribution occurs when

dim_max_index mod $processors\_count = 0$,

but even in other cases, the method distributes data *periodically*, which leads to a poor workload balance for some query classes.

Consider the following example, where the time-index is increasing from left to right, and the camera-index is increasing from top to bottom:

```
0 4 3 2 1 0 4 3 ...
1 0 4 3 2 1 0 4 ...
2 1 0 4 3 2 1 0 ...
3 2 1 0 4 3 2 1 ...
```

This example represents video sequences from four cameras round-robin-distributed among five disks. Now, consider retrieving data for camera 0 and 2 for timesteps 0 through T taking every fifth frame: we will end up utilizing only two out of four disks (0 and 2), just as if we had a *view per node* distribution.

One method that does not have such problems is a *random* distribution, which relies on a good random number generator. Note that a pseudo-random generator for the 2D case would be difficult,or impossible, to use for higher-dimensional input datasets. Therefore this approach might not be practical for high-dimensional input data, and also is not guaranteed to always provide an even distribution of data across the disks.

Hilbert space-filling curves [6] are a more general approach for input data distribution. Space-filling curve techniques allow distributing data evenly among processors while declustering data items that are close to one another in multiple dimensions. This should provide a good workload distribution for the range queries that the ADR-based system will be executing.

Our preliminary experiments tested the random and Hilbert curve data distributions. Our dataset consists of 400-frame sequences captured synchronously by 13 color cameras in the Keck Lab. Both distributions (*random* and *Hilbert curve*) partitioned the data fairly evenly onto the available disks. Therefore, querying the full dataset should have a good workload balance. The ADR parallel back-end was run on a Linux PC cluster consisting of 16 dual-processor Pentium-II 450MHz nodes with 256MB memory and one disk, interconnected via switched Gigabit Ethernet.

The query universe is a $2 \times 2 \times 2$ meter cube centered at $(1000, 700, 800)$ [mm] with respect to the camera calibration origin. Figures 4 and 5 show the workload balance results and execution times for the Hilbert curve and random data distributions, respectively, for three test queries. The test queries all cover the entire 3D volume, with one query covering all 400 time steps (frames) from 0 through 399, the second skipping every other frame for a total of 200 frames, and the third selecting every fourth time step for a total of 100 frames. Results are shown for both four and eight processors.

The results show that Hilbert curve declustering provides a completely even distribution of the data across all the processors/disks. Hilbert curves also provide good declustering for the queries that access only part of the input image datasets, as seen by the low variance in the number of images accessed per processor. The random declustering does not provide a completely even distribution of the data across all processors, but actually shows even lower variance in the number of images accessed per processor than does the Hilbert curve declustering for partial dataset queries.

In looking at query execution times, we see that the Hilbert curve declustering always provides better overall performance than random declustering, most likely because of better local disk access patterns on the processors (fewer

	Images per processor		
proc#	100 frames	200 frames	400 frames
0	390	700	1300
1	310	632	1300
2	294	688	1300
3	306	600	1300
std. dev.	38	38	0
Execution time (sec)	204	415	830

(a) 4 processors

	Images per processor		
proc#	100 frames	200 frames	400 frames
0	232	390	650
1	152	276	650
2	110	294	650
3	156	340	650
4	158	310	650
5	158	356	650
6	184	374	650
7	150	260	650
std. dev.	35	44	0
Execution time (sec)	138	270	543

(b) 8 processors

Figure 4. Hilbert-curve declustering - images retrieved/processed per processor and query execution times, on four and eight processors

	Images per processor		
proc#	100 frames	200 frames	400 frames
0	315	655	1309
1	303	680	1263
2	293	651	1247
3	328	658	1310
std. dev.	25	11	38
Execution time (sec)	210	435	850

(a) 4 processors

	Images per processor		
proc#	100 frames	200 frames	400 frames
0	164	338	673
1	150	311	650
2	156	312	633
3	171	336	651
4	168	334	667
5	182	365	672
6	158	328	623
7	171	337	676
std. dev.	9	26	34
Execution time (sec)	155	314	621

(b) 8 processors

Figure 5. Random distribution declustering - images retrieved/processed per processor and query execution times, on four and eight processors

disk seeks). We will be investigating this effect further, and will report on those results at the workshop. Finally, the results show that the ADR implementation achieves reasonable speedups going from four to eight processors, with somewhat better speedup attained with the Hilbert curve declustering, especially on the largest query (400 frames). This shows that system performance looks promising for the much larger datasets that will be produced and processed in future applications within the framework. We will present results on sixteen processors at the workshop, and will also be running on much larger machine configuration in the future (several ADR applications have been run on up to 128 processors).

5. Conclusions

We have shown that ADR can be customized to be efficiently used for multi-perspective imaging and 3D shape analysis. We have introduced a multi-perspective imaging framework and used it to create a 3D shape analysis application. Basing the system on ADR ensures portability across parallel platforms and system robustness while handling large datasets. Our initial experiments show that we can achieve good workload balance from a straightforward Hilbert curve data declustering scheme, and that with a good workload balance the system scales (at least up to eight processors). Future work will investigate the behavior of the system on larger datasets and larger machine configurations.

We will also be investigating additional multi-perspective applications, including

- *3D object tracking* via adaptive queries and query pipelining,
- *view interpolation* via multi-view image rendering, and
- *smart environment management* via all of the above techniques with other application-specific processing

Acknowledgments

This research was supported by the National Science Foundation under Grants #ACI-9982087 and #ACI-9619020 (UC Subcontract # 10152408).

We would also like to acknowledge the assistance of Chialin Chang and Tahsin Kurc on various ADR implementation issues.

References

[1] A. Afework, M. D. Beynon, F. Bustamante, A. Demarzo, R. Ferreira, R. Miller, M. Silberman, J. Saltz, A. Sussman, and H. Tsang. Digital dynamic telepathology - the Virtual Microscope. In *Proceedings of the 1998 AMIA Annual Fall Symposium*. American Medical Informatics Association, Nov. 1998.

[2] E. Borovikov. A distributed system for real-time volume reconstruction. In *Computer Architectures for Machine Perseption*. IEEE Computer Society, Sept. 2000.

[3] C. Chang, R. Ferreira, A. Sussman, and J. Saltz. Infrastructure for building parallel database systems for multi-dimensional data. In *Proceedings of the Second Merged IPPS/SPDP Symposiums*. IEEE Computer Society Press, Apr. 1999.

[4] C. Chang, T. Kurc, A. Sussman, and J. Saltz. Optimizing retrieval and processing of multi-dimensional scientific datasets. In *Proceedings of the Third Merged IPPS/SPDP (14th International Parallel Processing Symposium & 11th Symposium on Parallel and Distributed Processing)*. IEEE Computer Society Press, Los Alamitos, Calif., May 2000.

[5] L. Davis, E. Borovikov, R. Cutler, D. Harwood, and T. Horprasert. Multi-perspective analysis of human action. In *Proceedings of Third International Workshop on Cooperative Distributed Vision*, November 19-20, 1999.

[6] C. Faloutsos and P. Bhagwat. Declustering using fractals. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*, pages 18–25, Jan. 1993.

[7] R. Ferreira, T. Kurc, M. Beynon, C. Chang, A. Sussman, and J. Saltz. Object-relational queries into multi-dimensional databases with the active data repository. *Parallel Processing Letters*, 9(2):173–195, 1999.

[8] T. Kurc, C. Chang, R. Ferreira, A. Sussman, and J. Saltz. Querying very large multi-dimensional datasets in ADR. In *Proceedings of the 1999 ACM/IEEE SC99 Conference*. ACM Press, 1999.

[9] T. Kurc, Ümit Çatalyürek, C. Chang, and J. Saltz. Exploration and visualization of very large datasets with the active data repository. Technical Report CS-TR-4208 and UMIACS-TR-2001-04, University of Maryland, Department of Computer Science and UMIACS, Jan. 2001. Submitted to IEEE Computer Graphics and Applications.

[10] T. M. Kurc, A. Sussman, and J. Saltz. Coupling multiple simulations via a high performance customizable database system. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, Mar. 1999.

[11] G. Lohmann. *Volumetric Image Analysis*. Willey and Teubner, 1998.

[12] H. Samet. *Applications of Spatial Data Structures*. Addison-Wesley Publishing Company, Inc., 1990.

[13] C. T. Shock, C. Chang, B. Moon, A. Acharya, L. Davis, J. Saltz, and A. Sussman. The design and evaluation of a high-performance earth science database. *Parallel Computing*, 24(1):65–90, Jan. 1998.