A High Performance Multi-Perspective Vision Studio*

Eugene Borovikov, Alan Sussman, and Larry Davis
Department of Computer Science
University of Maryland
College Park, Maryland 20742, USA

{yab,als,lsd}@cs.umd.edu
http://www.umiacs.umd.edu/~{yab,lsd}
http://www.cs.umd.edu/~als

ABSTRACT

We describe a multi-perspective vision studio as a flexible high performance framework for solving complex image processing and machine vision problems on multi-view image sequences. The studio abstracts multi-view image data from image sequence acquisition facilities, stores and catalogs sequences in a high performance distributed database, allows customization of back-end processing services, and can serve custom client applications, thus helping make multi-view video sequence processing efficient and generic. To illustrate our approach, we describe two multi-perspective studio applications, and discuss performance and scalability results.

Categories and Subject Descriptors

I.4 [**Image Processing And Computer Vision**]: Reconstruction, Applications; I.4.8 [**Image Processing And Computer Vision**]: Scene Analysis—*color, depth cues, object recognition, shape, time-varying imagery*; I.4.10 [**Image Processing And Computer Vision**]: Image Representation—*hierarchical, multidimensional, statistical, volumetric*; H.3.3 [**Information Search and Retrieval**]: Information Search and Retrieval—*clustering, query formulation*

General Terms

algorithms, management, measurement, performance, experimentation

Keywords

multi-perspective, vision, image processing, volumetric reconstruction, distributed system, high-performance, database

*This research was supported by the National Science Foundation under Grants #EIA-9901249, #EIA-0121161, #ACI-9619020 (UC Subcontract #10152408), and #ACI-9982087, and Lawrence Livermore National Laboratory under Grants #B517095 (UC Subcontract #10184497). The Linux cluster used for the experiments was obtained through NSF Grant #ANI-0123765 and a grant from IBM.

ICS'03, June 23–26, 2003, San Francisco, California, USA.

1. INTRODUCTION

Modern image analysis and computer vision systems often use *multi-perspective* imaging, which employs multiple cameras shooting the scene of interest from various perspectives. The basic idea is that more views deliver more information about the scene, and potentially allow recovery of interesting 3-dimensional features with more accuracy and less intrusion into the scene (e.g., no markers for tracking objects or people through the scene are required) . An exciting broad range of applications for such systems includes virtual view rendering, complex shape and movement analysis, multi-person tracking, virtualized reality, and smart environments [6, 7, 9, 20, 21, 22, 24].

The availability of affordable digital cameras along with the ever increasing computing capabilities of desktop computers has made such applications more attractive for use in various computer art and animation studios, medical facilities, business environments and people's everyday lives.

Small scale systems (involving 3 cameras or less) can be handled by a modern desktop computer, while larger scale applications could be quite challenging even for high performance computing machines. Performance issues arise because multi-perspective systems with large numbers of cameras can produce and process vast amounts of image data and video streams that can be very difficult to manage without an efficient way to store, catalog and process the data.

Figure 1 shows a VRML model of a multi-perspective laboratory at the University of Maryland [9]. One minute of multi-perspective video, shot in this facility, may require up to 95 GBytes of storage. If the desired processing of the video data cannot be performed in real time, as described in [4], the image data must persist in long-term storage for further off-line processing.

It is still not feasible to manage and process such large quantities of image data on a single PC or workstation, because of performance issues stemming both from accessing the stored data and the computational requirements of the subsequent processing. Therefore it is imperative to store and process those sequences on a parallel machine or a cluster of workstations, because much of the processing can be done in parallel, and there is an opportunity to distribute data across a disk farm to achieve high performance.

We propose a solution to the above problems in the form of a *multi-perspective vision studio* that abstracts multi-view image processing and vision algorithms away from the data

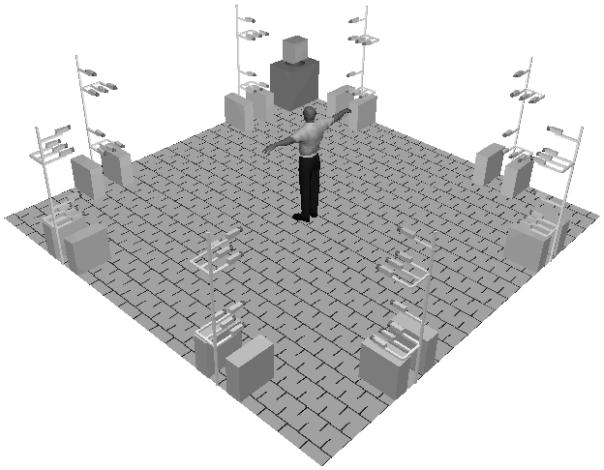

Figure 1: Keck Lab model

acquisition facilities, and provides a flexible and efficient way to store, retrieve and process the collected data. By developing such a studio we allow new multi-view image processing and vision algorithms to be implemented quickly and run efficiently through a mechanism of user requests or *queries*.

The studio is based on an object-oriented framework called the Active Data Repository (ADR) [15, 16], which provides the ability to process and manage large amounts of scientific data on a parallel machine or workstation cluster. The overall system design is logically partitioned into three parts:

- an ADR-based customizable and portable parallel back-end,
- an application front-end that acts as a control gateway for client applications, and
- a client application with a 3D viewer.

The studio design and implementation extends both the ADR back-end customization and the application front-end from a prototype [5] and provides a completely new OpenGL-based client application with extensive query specification and result management capabilities. We illustrate the utility of the multi-perspective vision studio for two tasks: (1) *volumetric shape reconstruction* and (2) human body shape estimation via *density model fitting*. Note that efficient volumetric shape reconstruction serves as the basis for implementing other interesting 3D vision algorithms not discussed in the paper, such as 3D tracking or surface recovery via a mesh approximation.

2. STUDIO CORE

With multi-perspective image capture facilities able to generate very large datasets, and with the development of powerful multi-view vision algorithms, it becomes very desirable to provide vision application developers with a highly customizable tool for efficient management and processing of the collected multi-perspective data. Our multi-perspective vision studio provides such a tool. As shown in Figure 2 the studio provides a *data loading facility* to import the captured video sequences into a high-performance data server that is managed by the ADR parallel *back-end* services. The data server can be queried by *client* applications via a set of *front-end* services, some of which are application-specific and some of which are common to all applications. Query results are returned to client applications either directly from the parallel back-end or through the application front-end (if, for example, a firewall separates the client application and the data server).

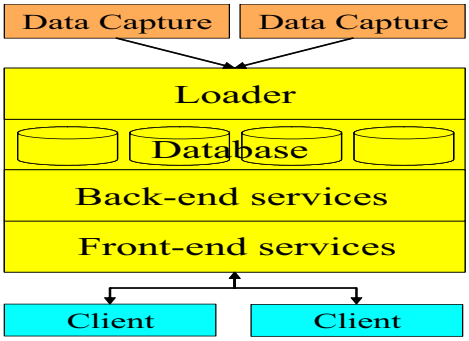

Figure 2: Multi-perspective Vision Studio architecture

2.1 Multi-view datasets

Because the studio data server is based on ADR, it follows ADR's basic design. A *dataset* consists of all images from a single multi-perspective video sequence. A *chunk* of data is a single image whose attributes include a *camera index* and a *time index*. Data chunks are declustered across the available disks in the data server according to a user-specified strategy. ADR supplies a default strategy based on Hilbert space-filling curves [18, 19]. Dataset *meta-information* (e.g., camera configuration) may also be supplied by the user when loading the data into the data server.

2.2 Customizable back-end

The image datasets are accessed through the parallel back-end (PBE). The back-end is responsible for storing the datasets and carrying out the requested input data retrieval, projecting input data from its coordinate space into the output coordinate space, and performing aggregation operations. All back-end services can be customized by the application developer, but default implementations are provided for most services by the studio implementation.

The default implementation of the index service maps a four-dimensional query region (x, y, z, t) into a set of relevant data chunks. A <camera-index, time-index> pair is mapped to the <time-index> via the default customization of the projection service, to allow aggregation across all cameras that view a given spatial region at a given time. No default aggregation operation is supplied, since aggregation is completely application dependent.

2.3 Application front-end

The application front-end (FE) interacts with clients via an application-specific communication protocol, and translates queries into a common format for the ADR front-end, which sends the query to the PBE. The application FE's duties also include maintenance of meta-information about datasets (such as dataset name, number of views/cameras and number of multi-perspective frames), responding to queries

coming from client applications, and (in some cases) sending query results back to clients. 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.

2.4 Graphical Client Application

The studio application front-end can accept both individual queries and query batches. Query batches are more convenient for command line-based client applications, while graphical clients are more likely to produce single queries. The default GUI client provided by the studio allows both approaches via the concept of a query *session*.

A session is a collection of queries that a user specifies via the graphical interface of the 3D OpenGL client, which also renders the session dataset in a flexible fashion. Sessions can be saved to a file from the client, and loaded into the client for later viewing.

A query request from the client into the data server includes the name of a dataset, a specification of the 3D region of interest within the dataset, a frame index range with a constant stride, and a set of cameras to use. For applications that employ volumetric reconstruction, a query also specifies a resolution for the reconstruction (which corresponds to the depth of the octree used to represent the volume). The volumetric results (one per frame) are returned as a set of occupancy maps, represented as octrees. The new results are then merged with previously obtained results, if any, for viewing in the client. The user controls which volumetric sequences participate in the merge by selecting the ones of interest through a GUI control. The *merge* operation always preserves the most detailed sections of the participating occupancy maps.

The client renders the 3D data a frame at a time, allowing the user to browse the reconstructed sequences of 3D data. A typical rendering of a 3D scene is shown in Figure 3. As with most 3D renderers, the scene can be examined from any point of view by dragging the mouse across the window. A particular 3D frame is selected via the slider control.

3. VOLUME RECONSTRUCTION

The volumetric shape reconstruction procedure is based on visual cone intersection, and was described in detail in [4, 5]. The main idea is to efficiently build a 3D *occupancy map* of the foreground object using silhouette data from all available cameras (for example, see Figure 8) [23]. At each step, the algorithm determines the occupancy of the space bounded by a cube, and if the occupancy is undetermined, it is decided for each of the cube's eight sub-cubes recursively, producing an *octree* approximating the object's 3D shape.

```
OccupancyMap(cube,depth)
  if depth=0, return;
  label=Occupancy(cube);
  if label=undetermined,
    for subcube in Sub-cubes(cube),
      OccupancyMap(subcube,depth-1);
  else output label;
```

The prototype design and implementation did not perform well for several reasons, mainly related to the algorithms used to perform cube projections, the visual cone intersection algorithm, and the method used to merge query

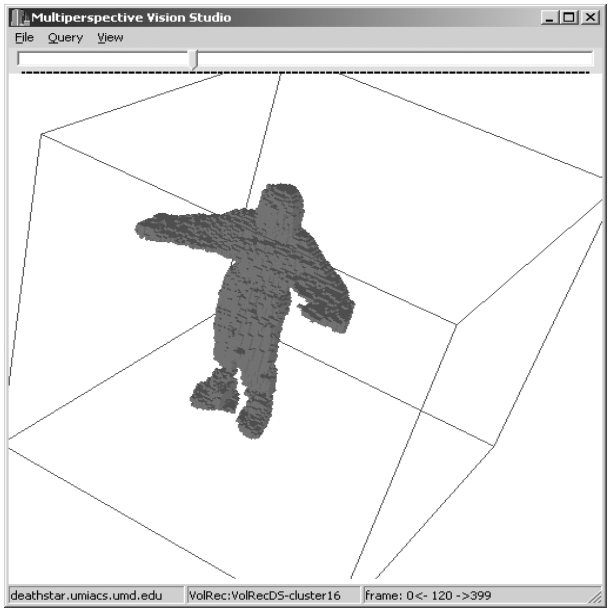

Figure 3: Client GUI

results from multiple sources. We describe a set of potential improvements to the problematic algorithms and provide experimental results to evaluate their performance benefits.

3.1 Cube projection strategies

The efficiency of our occupancy map building procedure strongly depends on the speed of the routine that determines a cube's occupancy, which involves computing projections of the given cube into each of the available image planes. The computation to project a 3D point $\mathbf{X}$ to a 2D camera image plane is given by $\mathbf{x} = P\mathbf{X}$, where P is a 3×4 camera projection matrix. Therefore, projecting a cube requires 8 matrix-vector multiplications. The straightforward prototype implementation used *naive vertex projections*, i.e. project vertices as needed, which led to redundant projections of the same vertices, because multiple cubes of interest may have common vertices. We therefore consider two alternative cube projection strategies: *vertex caching* and *approximate cube projection*. We discuss the relative performance of the three strategies in Section 3.4.

3.1.1 Vertex caching

To reduce inefficiencies due to redundant vertex projections, we employed a hash table that, for each silhouette, stores the projected coordinates of already visited vertices. This method has average constant-time behavior for insert and look-up operations, but (when utilized for very fine-resolution occupancy maps) uses a significant amount of memory to store vertex projections, which is a limited resource employed by many other ADR system components. Therefore the storage allocated for the projected vertex cache might not be enough to store all vertex projections, resulting in cache misses that would require redundant projection operations.

3.1.2 Approximate cube projection

Because of the recursive nature and expected sparsity of the resulting occupancy maps (octrees), the volumetric re-

construction method can afford to be conservatively approximate at estimating the cube projections. We can prove that a cube projection always lies within the projection of the cube's circumscribed sphere. Estimating the bounding box of a sphere projection (requiring the equivalent of two vertex projections) is much less expensive than projecting eight vertices of a cube. However, this procedure produces an approximation of the true cube projection and, therefore, might require a deeper set of recursive calls in the octree construction procedure.

3.2 Visual cone intersection

A *visual cone* is the portion of 3D space the camera sees through a given silhouette. As was described in [5], the final occupancy map is produced by intersecting visual cones, also represented via octrees and produced from each silhouette view. The visual cone intersection procedure has two phases: local and global. In the local phase, each ADR PBE node constructs a *partial occupancy map* as an intersection of silhouette visual cones (one per image) local to that node. The global phase intersects all partial octrees across nodes to produce the final volumetric estimate.

3.2.1 A local phase improvement

In the prototype implementation of the local phase, each node sequentially constructed local visual cones and then intersected them to produce its partial occupancy map. This was inefficient because the local visual cone creating procedure always constructed the whole visual cone for a given silhouette image, postponing the intersection step until all of the visual cones are ready. The current implementation optimizes this operation by restricting the visual cone creator with the current partial result, if one exists.

For example, given a number of local silhouettes, we first build a visual cone for image 1. It will contain a superset of all occupied voxels. Then we consider the silhouette image 2, and prune the existing partial result accordingly, thus reducing the partial occupancy map, still resulting in a superset of all occupied voxels. The third and all subsequent steps of the local phase are the same as the second step.

The partial occupancy map reduction technique substantially reduces the amount of work compared to the amount done in the prototype implementation. More specifically, the performance improvement of the local phase is proportional to the number of local silhouettes (images) a node must process.

3.2.2 A global phase improvement

The prototype system's global phase was designed to process frames one at a time, thus imposing an unnecessary synchronization restriction, and resulting in an inefficient interprocessor communication pattern that required $(p-1)$ communications per frame, where p is the number of processors. In most cases, queries request reconstruction of multiple frames, leading us to consider the idea of *frame grouping*.

Suppose the parallel back-end processes frames in groups of k. With this design, the local phase constructs k partial occupancy maps. In the global phase, each of the p processors is responsible for combining at most $\lceil \frac{k}{p} \rceil$ frames with an expense of at most $\frac{2(p-1)}{k}$ communications per frame per processor. While frame grouping may require higher peak network bandwidth, it results in a better distribution of the global phase workload (compared to the extreme case of the prototype, where a single processor would do the global combine). We would expect query execution across processors to become less synchronous (because of smaller communication polling times per frame), and also to have a better overall workload balance (because of the smoothing effect from processing frames in groups). Our experimental results presented in Section 3.4.3 support the performance benefits of frame grouping.

3.3 Merging multiple query results

The graphical client application has to manage multiple octree sequences returned from one or more queries. For each frame, a user might have several query results for various 3D regions in the scene, resulting in aligned octrees, usually of different depths (corresponding to different resolutions). The client application has to merge and render all the parts of the octrees, preserving the greatest reconstruction resolution available in each query region.

The *merge* operation performed on binary-valued occupancy maps (octrees) is different from union or intersection. Given two maps, merge keeps all opaque nodes from both trees, always preferring the deepest branches in the octree. Therefore, merge is both *inclusive* (like set union) and *fine-detailed* (like set intersection). In the current system implementation, merge is done at the client, because only the client stores query results. In future work, we will investigate the utility of computing and caching merged results in the application front-end, or possibly in the parallel back-end, to improve query response times.

3.4 Experiments

We ran our experiments on a Linux cluster with sixteen Pentium III 650MHz nodes, each of which has 768MB of RAM and 160GB disk storage. The nodes are interconnected via channel-bonded Fast Ethernet (effectively 200Mb/sec per node). Our test dataset was a multi-perspective sequence of 2400 frames generated by 13 synchronized color cameras, each producing 640×480 images at 30 Hz. Even though all of the experiments were run on a PC Linux cluster, we did not attempt to optimize single node performance with platform-specific coding techniques (SSE, etc.), because we wanted the source code to be portable across different platforms.

3.4.1 Different projection approaches

In the volumetric reconstruction procedure, the most computationally intensive task is constructing silhouette-based visual cones. This operation involves numerous 3D$\rightarrow$ 2D projection operations per silhouette (order of millions for an octree of depth 8).

We queried a sample sequence of 256 frames, each frame reconstructed to a depth of 8, and measured the time it takes to do all necessary projections for a visual cone, using the three different strategies described in Section 3.1.

On average, 85% of the vertex projections made by the naive cube projection method were redundant. The vertex caching method stores computed projections in a hash table, which eliminates most of the redundant vertex projections to produce a considerable performance gain over the naive approach. The hash table was made quite large, using a memory buffer that was 50% of the octree buffer size. The approximate projection method, in turn, ran about 2 times faster than the vertex caching method. This can be ex-

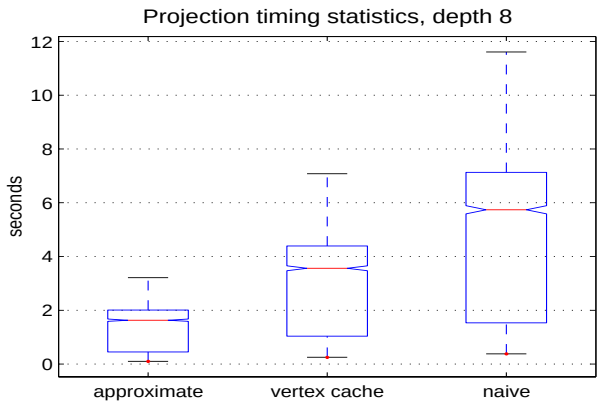

Figure 4: Voxel projection timing statistics. The plot presents sample distributions as notched-boxes with lines at the samples' lower quartile, median, and upper quartile values. The whiskers show the samples' extrema.

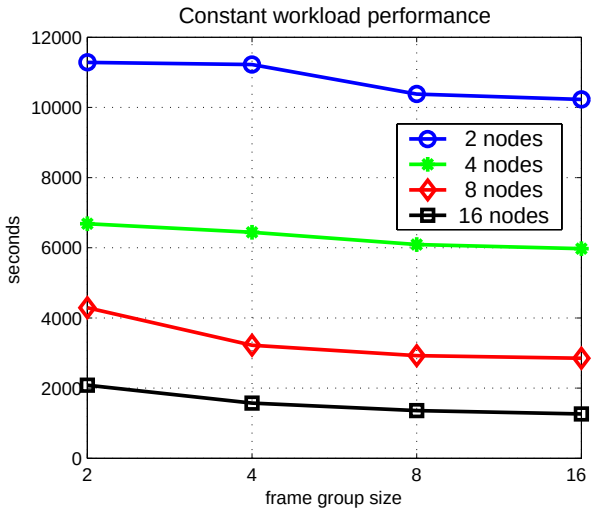

Figure 6: Major factors affecting server performance: number of nodes and frame grouping

plained by the fact that in any resulting occupied cube set, the number of cube centers is always strictly less than the total number of (shared) vertices.

Our timing experiments show that with respect to the *naive* projection method, using *vertex caching* can speed up the visual cone construction by an average factor of 1.6, and using *approximate* projections by an average factor of 3.5. Figure 4 shows the timing results of the three different cube projection strategies for a sample sub-sequence of 256 frames, with each volumetric frame reconstructed to depth 8. The processing time statistics were aggregated over all frames and all back-end nodes. Notice that not only did the approximate projection method perform the best, it also resulted in the lowest standard deviation for the cube projection running times.

3.4.2 System scaling

In our scaling experiment we used queries that require processing a fixed number of images (3900 for the largest query, 500 for the smallest), and ran on 2, 4, 8, and 16 processors. The upper two plots in Figure 5 show that the system achieves close to linear speedups on contiguous time range queries (step=1), while the lower two plots illustrate suboptimal system scaling for non-contiguous time range queries (step=3) due to workload imbalance introduced by the choice of the time stride as well as by the set of cameras used. More time stride related experiments are discussed in Section 3.4.4.

3.4.3 Frame grouping schemes

As was mentioned in Section 3.2.2, the straightforward approach of processing one multi-view frame at a time [5] suffered from (a) excessive communication polling time, and (b) significant workload imbalance. Our experiments show that with multiple processing nodes, performance improves by processing several multi-view frames at a time. A typical eight-node query (third curve in Figure 6) for a sequence of 600 frames with the octree depth parameter set to 9, clearly reveals the advantages of frame grouping. Notice, however, that it does not make practical sense to make the frame group size too large, because the performance benefits diminish and the bandwidth and memory requirements for interprocessor communication greatly increase beyond a fairly small value for the frame group size.

We summarize the performance effects of the parallel machine size and the frame grouping in Figure 6. By far, the most important factor affecting the system performance is the number of processors it runs on. Large frame group size increases performance in all considered cases, but its effect is fairly limited.

3.4.4 Varying time stride

Data declustering based on Hilbert space filling curves should result in close to optimal workload balance for range queries on contiguous regions in the multi-dimensional space. However, that is not necessarily true for more complex query patterns, such as volumetric reconstruction queries with varying time strides, where the user requests, for example, every other or every third frame within a specified set of frames. Figure 7 illustrates exactly that problem. We produced statistics for a 16-node parallel back-end servicing 220-frame queries into a fixed 3D region viewed by 13 cameras, varying the time stride.

Although we used an adequate frame grouping parameter (16), time strides of 4 and 8 resulted in a poor workload balance. For the time strides 2 and 10 the frame distribution is not perfect but reasonably good, and for 1, 5 and 7 the workload balancing is close to ideal.

By design, the data declustering algorithm based on Hilbert curves results in good workload balance for contiguous range queries (i.e. over time-contiguous regions). We also expected Hilbert curve declustering to produce a good workload balance for other time strides, but as our experiments show the workload may be poorly balanced for some non-unit time strides queries. This result is not completely surprising because Hilbert curve based declustering emphasizes local separation, ensuring that data chunks that are adjacent in the underlying attribute space (in this case, the time/frame number) are distributed to different disks. In the case of non-unit strided queries, the workload bal-

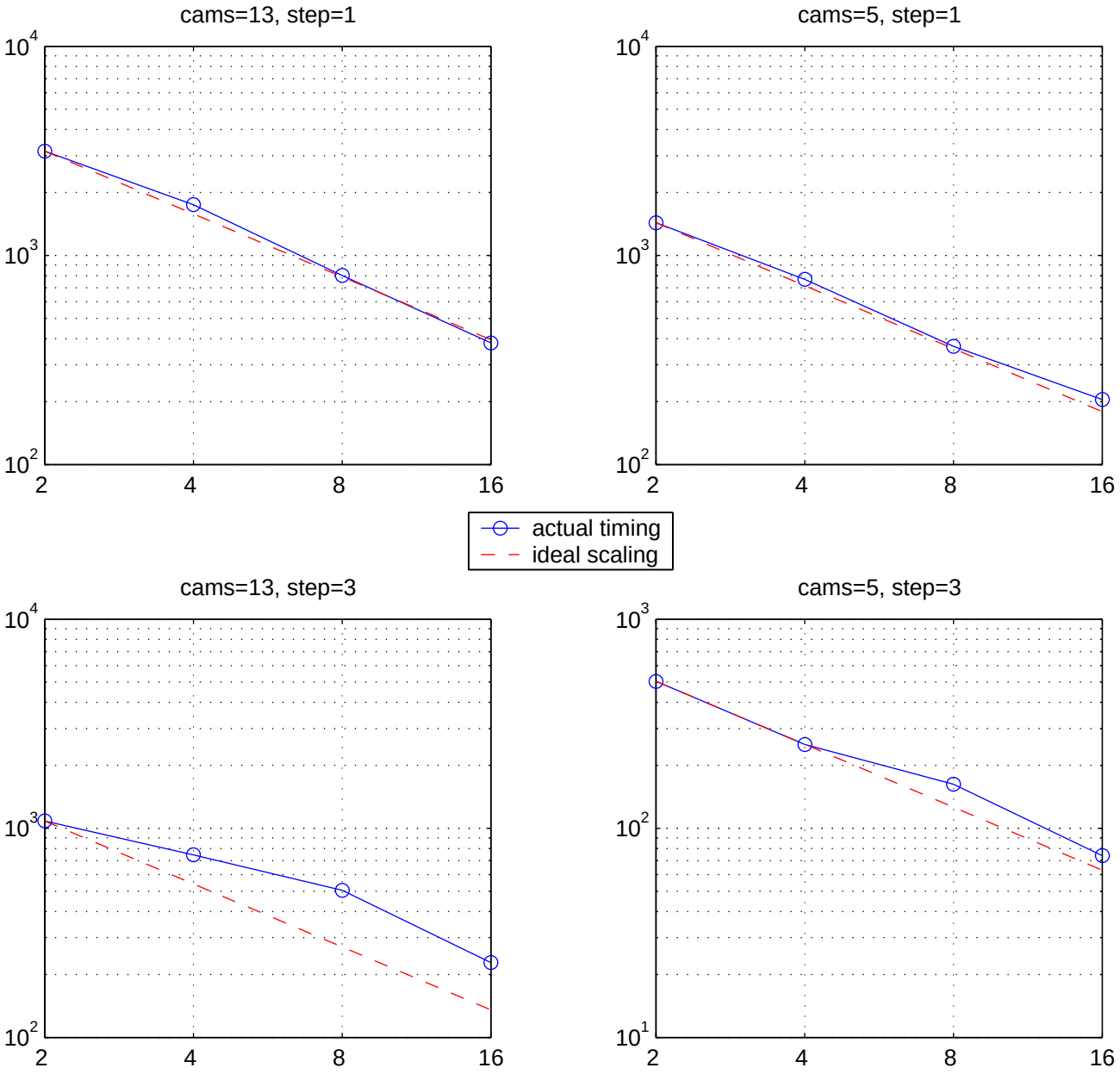

Figure 5: System speedup for fixed workload queries. The log-log plot captions indicate the number of cameras and the time step used. The horizontal axes show the number of processors, and the vertical axes show the total query time in seconds.

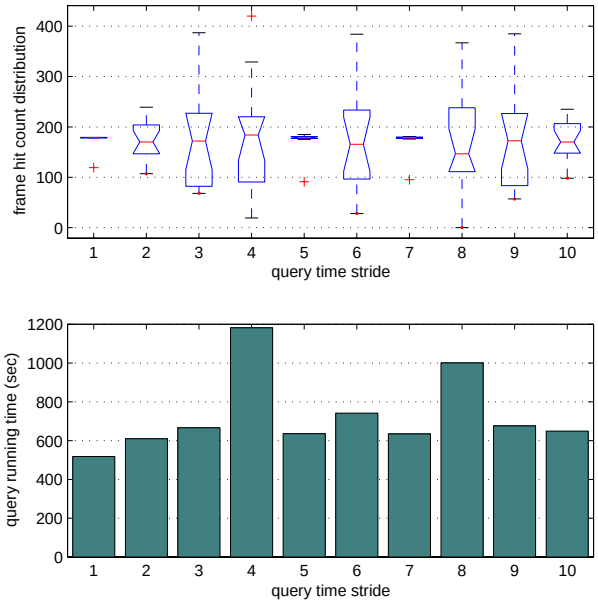

Figure 7: Load balancing depends on the time stride

ance becomes very unpredictable. Fortunately, most Multi-perspective Vision Studio queries are expected to specify unit time strides, so that a good workload balance is expected.

4. DENSITY BASED MODEL FITTING

3D shape estimation is one of the complex tasks related to multi-perspective vision [10, 11, 12, 14]. Having an efficient method for acquiring 3D occupancy maps is not all that is needed for shape recognition. One needs a method for analyzing an occupancy map to reason about the foreground object's shape. One good way to analyze the object's shape is to fit a model into the segmented image [3, 13, 17, 25]. Such a model fitting procedure can be incorporated in the Multi-perspective Vision Studio to help analyze the target object's shape.

4.1 Density-based shape recognition method for articulated objects

The model fitting method we present can be used for analysis of any 3D articulated non-rigid objects. In this work we narrow its application to analyzing human figures. We have an efficient way of constructing the volumetric shape estimate of a human body. Using such a 3D shape estimate we would like to locate the body and its constituent parts, and estimate their shapes.

We model real world objects via parametric *density fields* that are similar to potential field techniques [8], but are more flexible in allowing for a wider class of density functions. An object's density field is a non-negative scalar function showing how much this object differs from its environment at each point in space. Given a density field, one can derive many geometric and topological properties of the object it represents. For example, a field's iso-surface approximates the object's surface, and the field's ridges correspond to the object's skeleton. Within the model fitting framework, density fields can be utilized in two ways: as a target shape representation, and as a driving force for model fitting.

4.1.1 Formalism behind the method

We characterize the model of the object by an integrable and bounded *density* function $f(\mathbf{x}, \theta)$. For any model configuration characterized by a parameter vector $\theta \in R^m$, $f(\mathbf{x}, \theta)$ gives a non-negative density value at any $\mathbf{x} \in R^n$.

For an occupancy map $V \subset R^n$, we define a measure of fitness

$$F(\theta) = \int_V f(\mathbf{x}, \theta)\, dV_{\mathbf{x}} \tag{1}$$

and solve the following optimization problem:

$$\text{maximize}_\theta [F(\theta)] \text{ subject to some constraints on } \theta$$

For most practical problems, this optimization calls for a numeric solution. Suitable numerical integration and optimization packages should be used for the problem at hand. In our experiments, we used a midpoint quadrature rule and a publicly available implementation of the Levenberg-Marquardt non-linear minimization algorithm, commonly known by the name *MinPack*.

4.1.2 Modeling 3D articulated objects

Complex articulated objects such as human or animal bodies consist of many sub-parts and have a lot of non-rigid tissue. Therefore, when dealing with articulated objects it is often very desirable to utilize a *composite density function* by combining densities of the components, and a *hierarchical model* capturing both coarse (few parts) and fine (elaborate sub-parts) object details.

Since a target object may consist of several moving parts and sub-parts, we would like its *cumulative density* field to consist of several *component density* fields and dependent sub-fields:

$$f(\mathbf{x}, \theta) = \sum\nolimits_i f_i(\mathbf{x}, \theta)$$

Component densities usually represent individual body parts or the sub-parts thereof. They are usually composed out of some well known *basic densities*:

$$f_i(\mathbf{x}, \theta) = g_i(A_i \mathbf{x}; \theta)$$
$$g_i(\mathbf{x}; \theta) = \prod\nolimits_j h_j(\mathbf{x}_j; \theta)$$

where h_j are some basic densities (often one-dimensional), and A_i is a component-specific scale-rotate-shift operator.

4.1.3 Hierarchical model fitting

Occupancy maps for foreground objects, represented by octrees, can be used for hierarchical model fitting. Different octree levels can be matched by corresponding levels in the model. At some octree level, the foreground object becomes well localized and its major dimensions become evident. At the next level, one can start distinguishing the object's gross body parts. At finer levels, the major body parts and joints are well localized and can be reliably detected. The higher (finer) levels of the hierarchy can supply additional information for detailed recovery of small body parts such as hands and fingers.

At its first level the model aims only at the torso. Once the torso is localized, the model at its second level additionally fits legs and arms. When locations of the gross body

parts are estimated, on the next level the model inherits the previous estimations and refines the detail of the foreground object by splitting parts into sub-parts. In this way, the most detailed level of the hierarchy will have a good initial guess and thus less chance to get stuck at local minima in the optimization process.

4.2 Experiments

In our current implementation, density based model fitting runs at the client, because it relies on the results of multiple volumetric queries merged by the client application for a given volumetric frame. Since we are interested in estimating human body figures, we modeled human body parts as follows. For the head we used a product of three *bell-shaped* densities given by $e^{-(cx)^2}$, for legs and arms we used tube-like densities composed as a product of two bell-shaped densities and one *plateau-shaped* density given by

$$p(x;r) = \begin{cases} \exp(x+r)^2 & x < -r \\ 1 & -r \le x \le r \\ \exp(x-r)^2 & x > r \end{cases}$$

while the torso was modeled as a product of two plateau-shaped and one asymmetric bell-shaped densities.

We were able to fit density based models to volumetric images of a person performing pointing gestures. Figure 8 exhibits a subset of the source imagery for a single frame that produces a typical multi-perspective scene shot. The volumetric reconstructions shown in Figure 9 are octrees of depth 8, as are the renderings of the density models restricted by some iso-surfaces. The iso-surface renderings appear to be smoother than the source occupancy maps because the density models are based on smooth density functions.

On a 1.4GHz Pentium4 PC one round of hierarchical fit takes from 3 to 4 minutes depending on the scene complexity and the initial guess. The most expensive computation occurs in evaluating the objective function, which requires 3D numeric integration. We are working on implementing the entire optimization procedure in the multi-perspective vision studio parallel back-end.

5. CONCLUSIONS AND FUTURE WORK

We have presented a multi-perspective vision studio as a flexible high performance framework for solving non-trivial image processing and machine vision problems on multi-view image sequences. The studio makes multi-view video sequence processing efficient and generic by

- abstracting multi-view image data from image sequence acquisition facilities,
- storing and cataloging sequences in a high performance distributed database,
- allowing customization of back-end processing services, and
- serving multi-view data to various custom client applications.

We have described how the multi-perspective studio can be used for complex 3D shape analysis applications that include distributed volume reconstruction and density based model fitting.

Our experimental results show that high performance comes from

- an effective distribution of the multi-perspective images across the disk farm,
- frame grouping to improve workload balance, and
- an efficient strategy for projecting between the volume and the images.

Our ongoing work includes

- the introduction of color and texture mapping on reconstructed surfaces,
- employing a more flexible filter-based data server based on the DataCutter Grid infrastructure [2, 1], and
- implementing the density-based model fitting procedure in the data server.

Texture mapping will improve the presentation quality of the reconstructed 3D image and will be based on building a mesh given an occupancy map. The texture and color can then be re-projected back from the original imagery into the mesh patches.

DataCutter is a component-based framework for implementing and efficiently executing data analysis applications in the distributed, heterogeneous Grid environment. Employing a filter-based data server will allow for easier implementation of additional image processing and vision applications, and and also allow parts of the application to be run on processors other than the ones where the data is stored so that additional computational resources can be applied if needed. The ADR implementation requires that the computations be performed on the processors that store the data. Fortunately, the DataCutter framework also enables us to reuse much of the application-specific code from the ADR-based data server implementation.

Finally, the model fitting process is currently quite slow because it is implemented as a sequential routine in the client application. Model fitting can be parallelized and executed in the data server, enabling interactive response times for these complex tasks.

In the future, we hope that the multi-perspective vision studio will become a useful research and development tool for many interesting multi-sensor applications, not limited to just optical sensors (e.g. video cameras), but also involoving other non-optical imagery.

6. REFERENCES

[1] M. Beynon, C. Chang, U. Catalyurek, T. Kurc, A. Sussman, H. Andrade, R. Ferreira, and J. Saltz. Processing large-scale multidimensional data in parallel and distributed environments. *Parallel Computing*, 28(5):827–859, May 2002. Special issue on Data Intensive Computing.

[2] M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz. Distributed processing of very large datasets with DataCutter. *Parallel Computing*, 27(11):1457–1478, Oct. 2001.

[3] J. Bloomenthal and C. Lim. Skeletal methods of shape manipulation. In *Proc. of International Conference on Shape Modeling and Applications*, pages 44–47, 1999.

[4] E. Borovikov and L. Davis. A distributed system for real-time volume reconstruction. In *Proc. of Computer Architectures for Machine Perception*. IEEE Computer Society, Sept. 2000.

Figure 8: A multi-perspective scene shot with source images and the corresponding silhouettes

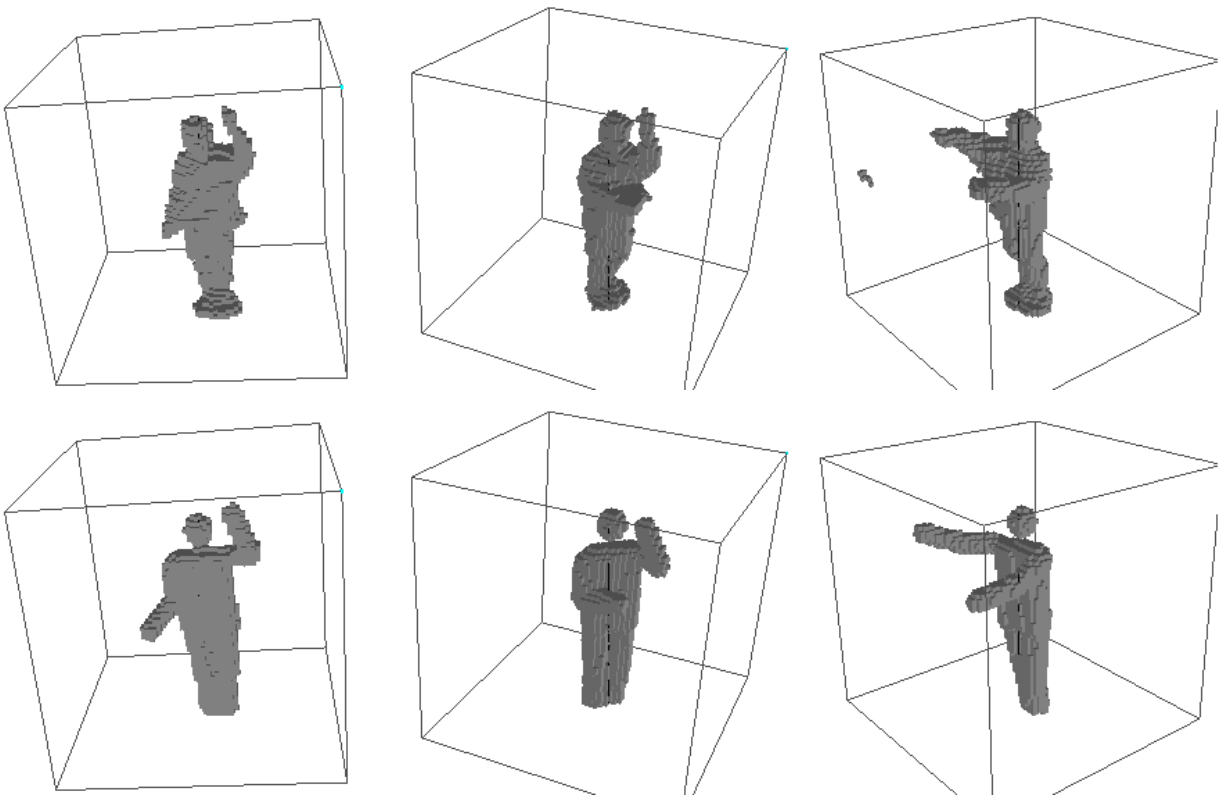

Figure 9: Fitting density models (bottom row) to real images (top row)

[5] E. Borovikov, A. Sussman, and L. Davis. An efficient system for multi-perspective imaging and volumetric shape analysis. In *Proceedings of Workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia*, April 2001.

[6] A. Bottino and A. Laurentini. A silhouette-based technique for the reconstruction of human movement. *Computer Vision and Image Understanding*, 83, 2001.

[7] M. Cavazza, R. Earnshaw, N. Magnenat-Thalmann, and D. Thalmann. Motion control of virtual humans. *IEEE Computer Graphics and Application*, 18(5):24–31, 1998.

[8] J.-H. Chuang, C.-H. Tsai, and M.-C. Ko. Skeletonization of three-dimensional object using generalized potential field. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[9] 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.

[10] D. Dion, Jr., D. Laurendeau, and R. Bergevin. Generalized cylinder extraction in range images. In *Proc. of International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, 1997.

[11] F. Ferrie, J. Lagarde, and P. Whaite. Darboux frames, snakes, and super-quadrics: Geometry from the bottom up. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(8), 1993.

[12] D. M. Gavrila and L. S. Davis. 3-D model-based tracking of humans in action: A multi-view approach. In *Proc. of IEEE International Conference on Computer Vision and Pattern Recognition*, 1996.

[13] P. Golland and W. E. L. Grimson. Fixed topology skeletons. In *Proc. of IEEE International Conference on Computer Vision and Pattern Recognition*, 2000.

[14] P. Havaldar and G. Medioni. Full volumetric descriptions from three intensity images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):540–545, May 1998.

[15] T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz. Visualization of large datasets with the Active Data Repository. *IEEE Computer Graphics and Applications*, 21(4):24–33, July/August 2001.

[16] 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, Nov. 1999.

[17] J. Lee and T. L. Kunii. Model-based analysis of hand posture. *IEEE Computer Graphics and Applications*, 15(5):77–86, 1995.

[18] B. Moon, H. Jagadish, C. Faloutsos, and J. Saltz. Analysis of the clustering properties of the Hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, January/February 2001.

[19] B. Moon and J. H. Saltz. Scalability analysis of declustering methods for multidimensional range queries. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):310–327, March/April 1998.

[20] E.-J. Ong and S. Gong. Tracking hybrid 2D-3D human models from multiple views. In *Proceedings of the IEEE International Workshop on Modeling People*. IEEE Computer Society Press, Los Alamitos, Calif., 1998.

[21] H. Saito, S. Baba, M. Kimura, S. Vedula, and T. Kanade. Appearance-based virtual view generation of temporally-varying events from multi-camera images in the 3D room. Technical Report CMUCS99127, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, Apr. 1999.

[22] S. Stillman, R. Tanawongsuwan, and I. Essa. A system for tracking and recognizing multiple people with multiple cameras. In *Proc. Second Int'l Conf. Audio- and Video-based Biometric Person Authentication*, pages 96–101, 1999.

[23] R. Szeliski. Rapid octree construction from image sequences. *Computer Vision Graphics and Image Processing: Image Understanding*, July 1993.

[24] A. Utsumi, H. Mori, J. Ohya, and M. Yachida. Multiple-human tracking using multiple cameras. In *Proc. of the third IEEE Int'l Conf. Automatic Face and Gesture Recognition (Nara, Japan)*. IEEE Computer Society Press, Los Alamitos, Calif., 1998.

[25] M. Zerroug and R. Nevatia. Part-based 3D descriptions of complex objects from a single image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):835–848, September 1999.