A Distributed System for Real-time Volume Reconstruction

Eugene Borovikov and Larry Davis
Department of Computer Science
University of Maryland
College Park, Maryland 20742, USA
mailto: {yab,lsd}@cs.umd.edu
http://www.umiacs.umd.edu/users/{yab,lsd}

Abstract

We present a distributed system for constructing volumetric image sequences in real time. Each volumetric image depicts a moving object (e.g. a person) using an octree representation. The object's volume is reconstructed via visual cone intersection using multi-perspective view of the scene.

1 Introduction

We present work that is part of ongoing research on the problem of distributed motion capture and gesture recognition. We describe a distributed system for real-time volume reconstruction and volumetric image sequence production. The system reconstructs the volume occupied by a moving object (e.g. a person) in the scene being viewed from multiple perspectives, produces a volumetric image represented by an octree, and then streams the volumetric images into a sequence.

Over the last decade, the computer vision community has introduced a number of volume reconstruction techniques [4, 9, 8, 10, 6]. Some of them were more efficient than the others, but due to the large amounts of data involved none could produce multiple volume reconstructions per second without using specialized hardware.

Volume reconstruction via visual cone intersection is a classical procedure. Szeliski [10] introduced a very efficient algorithm for rapid octree construction from image sequences, but did not describe a real-time implementation. Saito [6] presented an interesting multi-camera approach for volume reconstruction in projective grid space. This approach required only "weakly" calibrated cameras, but was utilizing the CMU VR Lab's distributed capability only for image capturing. The rest of the system was neither distributed nor real-time. Davis et al.[1], introduced a distributed system for volume reconstruction, but it could not handle large volumes of data from multiple cameras due to the system's poor scalability.

The work being presented in this paper extends [1]. Here we present an improved distributed system for volume reconstruction that can run in real-time on a parallel machine based on the Keck cluster (section 2.1), and is capable of scaling up and down very well presenting flexibility in the number of processors. The following are the key ideas of the system:

- specialized lookup tables for voxel projections
- octree encoding optimized for streaming
- highly scalable visual cone intersection algorithm

These and some auxiliary issues will be discussed in detail in the paper.

2 System design

The distributed volume reconstruction system design overview covers two topics: a high-level description of the approach to the volume reconstruction problem, and the setup of the Keck computing cluster providing a parallel machine for the volume reconstruction system. We shall first give an overview of the Keck Lab architecture pointing out its strengths and weaknesses as an environment for distributed visual computing. We then shall introduce our approach to the volume reconstruction problem, and discuss its adaptation to the Keck Lab environment.

2.1 The Keck Lab architecture

The Keck Laboratory for the Analysis of Visual Motion is intended for multi-perspective imaging. It is a 24'×24'×10' room equipped with 64 digital, progressive-scan cameras organized into 16 groups as shown in Figure 1.

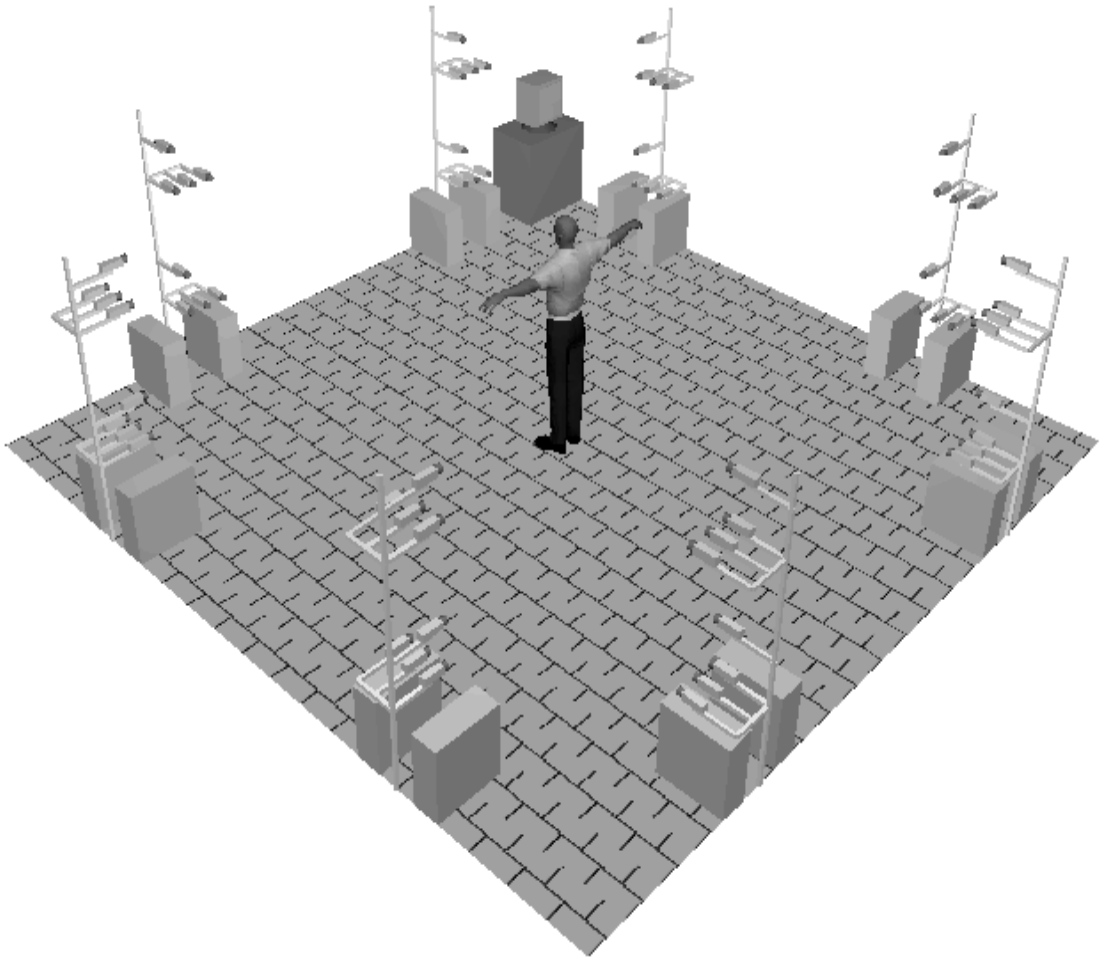

Figure 1. The Keck Lab architecture

Each group constitutes a short baseline stereo rig and consists of four cameras: three gray scale and one color. Each quadranocular rig is connected to a dedicated PC equipped with four identical digital frame grabbers. The 16 networked PC's form the Keck cluster that is controlled by a separate computer, the Keck domain controller. More detailed specifications of the Keck cluster are found in [1].

Being equipped with low-noise, high shutter speed cameras, and a high bandwidth network, the Keck cluster provides an ideal environment for high performance multi-perspective imaging and is used in a variety of projects as

- a high speed multi-view frame capture system capable of capturing up to 85 frames per second, and/or
- a parallel machine that allows high performance distributed processing of the multi-view sequences.

As capable as it is, the Keck cluster has limitations. The most important factor affecting the performance of any distributed system run on the Keck cluster is its network bandwidth. Suppose the Keck cluster (16 nodes) is to run a distributed program that we expect to capture and process 30 frames per second. With the given (theoretical) network bandwidth of 100Mbit/s, each node in the system would be able to transmit no more than $\frac{100}{30*16} = .20833$MBits ($\frac{.20833*1024}{8} = 26.666$KBytes) per frame. Thus, any distributed application utilizing all 16 nodes of the Keck cluster and requiring each node to transmit (on average) more than 25KB of data per frame has to give up any hope to run at the NTSC frame rate (30Hz).

2.2 Distributed volume reconstruction system

Algorithmically (as described in section 3), our volume reconstruction system does not put any restriction on the number of processors (network nodes). The system scales up (or down) very well allowing a great deal of flexibility in choice of the parallel (or distributed) system to be run

on. Our implementation of the system for the Keck cluster uses all the 16 *worker* nodes (the nodes capable of capturing video sequences), dedicating the Keck controller to be the *manager* node. The system is implemented in Microsoft Visual C++ using DSG-DEI-UC's adaptation of Message Passing Interface for Windows (WMPI).

The volume reconstruction procedure utilizes a multi-perspective view of the scene, and consists of the following steps:

- camera calibration
- silhouette extraction
- volume reconstruction via silhouette visual cone intersection
- volumetric data interpretation and visualization

Notice that not all of the above steps have to be done in real-time. Some preliminary steps (e.g. camera calibration) can be done off-line.

For calibrating cameras we are using the *non-coplanar* camera calibration procedure described by Tsai [11]. We use a calibration frame that provides up to 25 non-coplanar calibration points. For our implementation of Tsai's algorithm, the average estimated error in object space is about 3 mm. The accuracy can be improved somewhat by (i) increasing the image resolution and/or (ii) computing the projections of the feature points with sub-pixel precision.

Another off-line procedure is the background modeling. The volume reconstruction procedure relies on accurate foreground object silhouette extraction, which uses background subtraction. In our system, we employed recently developed silhouette extraction methods by Horprasert et al.[5] (for color images) and by Haritauglu et al.[3] (for gray-scale images). Both methods build statistical pixel-wise models of the scene background off-line. During execution of the volume reconstruction program, the background models are efficiently used for robust object silhouette segmentation (Figure 2). The background subtraction methods we use have the following advantages, they

- work well for general backgrounds (i.e. do not require the background to have a uniform color or texture),
- are tolerant to noise, and
- detect and eliminate shadows cast by the foreground object

Volume reconstruction via object silhouette visual cone intersection produce an estimate of the foreground object's volume which needs then to be visualized. Volumetric data rendering and analysis are usually time consuming tasks and also can be (and often are) done off-line.

In Figure 3, we show some snapshots of the volumetric data recovered by the volume reconstruction procedure from the multi-perspective sequence called "Conductor". This is an eight level deep octree representation of a human body shape recovered using six views. The volumetric reconstruction is not entirely accurate. The holes and discontinuities in the volumetric image are due to imperfect silhouette extraction (Figure 2). Some extra parts of the volume (e.g. "wings") that have not been carved out are due to the finite number of view points. There are also some aspects of the 3D shape (e.g. concavities) that the visual cone intersection method can not recover. On the positive side, the procedure has proven to be resistant to background noise due to its multi-perspective nature.

3 Volume reconstruction

This section describes the details of the volume reconstruction system. We start with a description of the entire distributed program. We then elaborate on the system's details that include the methods for visual cone construction and intersection.

3.1 Octree construction from multiple views

Once all the cameras are calibrated and background models are created, the system can proceed with the on-line volume reconstruction phase. The high-level algorithm looks as follows:

Algorithm 1 (Volume Reconstruction)

1. *loop*
2. *capture next frame*
3. *extract object's silhouette*
4. *compute silhouette's visual cone*
5. *intersect visual cones*
6. *render reconstructed volume*
7. *repeat*

The program's input is a synchronized multi-perspective video stream (live or pre-captured). The output is a sequence of volumetric images. Volumetric data in the system is represented via *octrees*, data structures for hierarchical representation of spatial 3D data. Octrees were shown [2, 7] to be

- *storage space efficient* taking memory proportional to the object's surface area (not to the object's volume), and

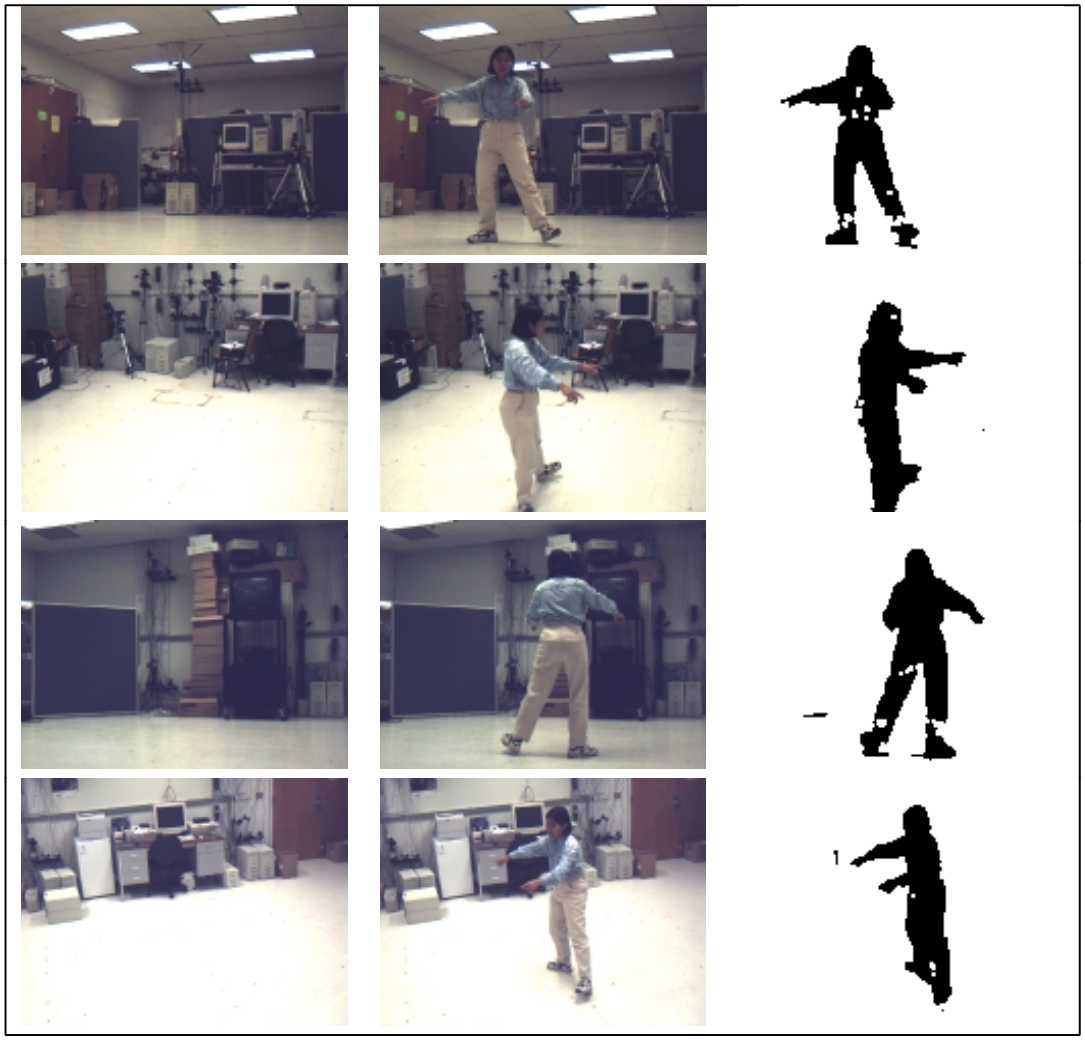

Figure 2. A multi-perspective snapshot of the background (the left column), a person (the middle column), and the extracted silhouette (the right column)

- *traversal time efficient* due to their hierarchical structure.

There are multiple ways to implement octrees. We optimized octrees for our system to make them cheap both to allocate/deallocate and to transmit over the network (section 3.3.2).

When Algorithm 1 runs on a parallel machine, steps 2, 3, and 4 are executed independently on each processor with no interprocessor communication involved. Step 5 involves a collective effort of all processors, and most of the communication takes place here. Finally, step 6 is executed solely at the manager node; it merely streams the volumetric data to a rendering or storage device, and involves no interprocessor communication.

3.2 Visual cone construction

The visual cone is defined to be the volume viewable through the object's silhouette. Given the foreground object silhouette, each node constructs its visual cone representing the volumetric data via an octree. The octree is constructed in a hierarchical fashion by testing the occupancy of volume elements (voxels). If the voxel in question is entirely transparent/opaque, it is labelled as such and the procedure returns; otherwise the voxel is subdivided into eight child subvoxels and the procedure is recursively invoked on each child.

The efficiency of the above method depends on the two subroutines: voxel projection, and voxel transparency test. The voxel transparency test can be done efficiently using the algorithm suggested by Szeliski [10], which utilizes

Figure 3. The reconstructed volume of a person's body viewed from different virtual points

half distance transform (HDT) maps. Voxel projection can be optimized by either usage of specialized (usually parallel) hardware or by using lookup tables precomputed in advance. In our system we take the latter approach.

We notice that the voxel transparency test does not use an explicit voxel's projection. It employs the bounding square of a voxel's projection to test it against the HDT map. Once a camera is calibrated and the initial bounding box is fixed, all voxel projections to the image plane become fixed and so do their bounding boxes. We take advantage of this situation by precomputing the bounding squares of all voxel projections (up to some maximum octree depth) and storing them in a lookup table.

A *voxel's projection bounding square lookup table* (VPBSLT) is a finite depth *full* octree spatially associated with the given initial box (e.g. a room). It is computed for every view off-line. Each node of that octree stores the bounding square of the corresponding voxel's projection to the image plane. During the visual cone construction phase, the VPBSLT is traversed in the same manner as the octree of the volume being reconstructed, and the precomputed values from VPBSLT are used in the voxel transparency test.

Using the VPBSLT (versus redundant projection of voxel vertices) significantly improves the speed of the voxel transparency test. Our experiments show that looking up the bounding box vs computing it via voxel projection is roughly ten times as fast.

Since the VPBSLT is a full octree, its size grows exponentially with its depth d as $O(8^d) = O(2^{3d})$. In our implementation, a lookup table of depth 8 takes about 28MB of space. This makes it impractical to share VPBSLTs over the network, and thus requires the visual cones to be computed

locally at each node.

3.3 Final volume reconstruction

The system estimates the object's volume by intersecting the visual cones constructed for each of the views. The visual cone intersection operation is an interprocessor effort, and therefore it involves a great deal of interprocessor communication. This section explains how the visual cones are intersected and what needs to be done to make this operation efficient, yet economical and scalable.

3.3.1 Distributed visual cone intersection

The parallel visual cone intersection procedure is based on hierarchical flow of data. Without loss of generality, suppose our parallel machine has $n = 2^k$ processors with ID's range from 0 to $(n-1)$. Let each processor with an odd ID send its visual cone to its predecessor; then each recipient intersects the received visual cone with the one it constructed. The senders can proceed with the next frame, while $n_1 = 2^{k-1}$ recipients being renumbered from 0 to $(n_1 - 1)$ continue with the visual cone intersection: the odd ID's send the octrees they've accumulated to their new predecessors, and the recipients intersect octrees they receive with ones they have. The process terminates after k steps with the final octree, equal to the intersection of all visual cones, accumulated at the root processor with ID=0. More formally the algorithm looks as follows:

Algorithm 2 (Distributed Visual Cone Intersection)

1. *step = 0*
2. *k = 1*
3. *while k $\leq$ n, do*
4. *if divisible(ID,k) and odd($\frac{ID}{k}$), (* a sender *)*
5. *send local octree to (ID-k)*
6. *break (* source does not need to continue *)*
7. *else if (ID+k) < n (* a recipient *)*
8. *receive external octree from (ID+k)*
9. *intersect local and external octrees*
10. *step = step + 1*
11. *k = 2k*

The above distributed procedure runs on each node in parallel. Given a collection of visual cones as input, it intersects them and produces a final estimate of the object's volume. Notice that the information travels in a binary tree fashion having the node with ID=0 as its ultimate destination and the accumulator of the final result. Here are the major points of the above procedure:

- it works for any n, not necessarily a power of 2
- it allows for pipelining of the visual cones
- its local memory and network bandwidth requirements are much lower than ones of the ring configuration described in [1]

These features make the visual cone intersection procedure highly parallel and scalable.

3.3.2 Octree encoding

The need for compact representation of volumetric data arises from having the system transmit it through the network. The most general way of constructing an octree would utilize a tree data structure, where each node stores the information about the node itself and the pointers to its children. In the case of an octree, most of the memory is taken for storing pointers. This organization is flexible but memory inefficient, and is often processing time inefficient especially when it comes to streaming octree data through the network, a process involving active (and redundant) node allocation/deallocation. Fortunately, there are better (application specific) ways to organize octrees.

We noticed that all our octree processing routines traverse them in a fixed manner, i.e. the depth-first-search (DFS) order. The system takes advantage of this fact by encoding the octrees in a special way. With the expected upper bound on the octree size, it is possible to allocate memory for each octree as a continuous byte buffer. The octree nodes are stored sequentially in the DFS-order, one node per byte. Each octree node records its level (4 bits), the occupancy attribute (3 bits) and the leaf-flag (1 bit). This representation is

- memory efficient (memory is allocated/deallocated once for the whole tree)
- compact (node-a-byte and no need for pointers)
- network transmission friendly (no need for serialization)
- disk file streaming friendly (same reason)
- adapted for DFS-order octree processing procedures (nodes are stored in the DFS-order)

The above features of the octree encoding make this data structure very efficient to use in a real-time volume reconstruction system.

4 Experimental results

The distributed system for volume reconstruction runs on the parallel machine provided by the Keck cluster consisting of 16 Pentium II 450MHz PC's connected via 100MBit/s bandwidth TCP/IP network. We used 14 color cameras to capture and process 400-frame sequences of a person moving within a $2\times2\times2$ meter cube. On 40-frame subsequences we observed the following average performance.

octree depth	voxel size (cm)	frame rate (Hz)
6	3.125	10
7	1.563	8
8	0.781	4

The system's frame rate depends on many factors. The most important one is evidently the octree depth. It affects primarily the visual cone construction/intersection and data transmission time. For an octree of depth 8, the typical time for the visual cone construction is 78ms, the cone intersection usually takes less than 30ms, and the octree transmission time is usually no greater than 32ms.

There are also static factors, independent of the reconstruction depth but dependent on the source image resolution and quality. They are frame acquisition (from board/disk, color filtering) and preprocessing (silhouette extraction, and HDT map construction). It turns out that for the coarse octrees (e.g. of depth 6), they may be significant, compared to the rest of the steps. The typical time for acquiring a frame (from a disk file) is about 35ms, while frame preprocessing time varies (depending on the size of the silhouette and noise) between 45 and 65 ms.

There is also a latency term due to the pipelining of the volume intersection procedure. For the reconstruction depth of 8, it is typically about 188ms.

5 Conclusions

Using the Keck Lab capabilities, we have built a distributed system for volume reconstruction. We have demonstrated that given a synchronized multi-perspective video stream, our system is capable of producing volumetric sequences of the foreground static and moving objects at the rate of an interactive application. It is possible to slightly increase the average frame rate of the system by refining the system's software, but to achieve a frame rate comparable to the NTCS's standard of 30Hz, the Keck cluster's hardware is insufficient. On a more powerful parallel machine, however, the system can be expected to run at 30Hz.

Acknowledgment The support of MURI under grant N00025149510521 and Keck Foundation are gratefully acknowledged.

References

[1] 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.

[2] C. Faloutsos, H. Jagadish, and Y. Manolopoulos. Analysis of the n-dimensional quadtree decomposition for arbitrary hyper-rectangles. Technical Report UMIACS-TR-94-13, UMIACS, University of Maryland at College Park, December 1994.

[3] I. Haritaoglu, D. Harwood, and L. Davis. W4: Who? when? where? what? a real-time system for detecting and tracking people. In *Proc. the thrid IEEE Int'l Conf. Automatic Face and Gesture Recognition (Nara, Japan)*, pages 222–227. IEEE Computer Society Press, Los Alamitos, Calif., 1998.

[4] H.Noborio, S. Fukuda, and S. Arimoto. Construction of the octree approximating three-dimensional objects by using multiple views. *PAMI*, 10(6):769–782, November 1988.

[5] T. Horprasert, D. Harwood, and L. Davis. A statistical approach for real-time robust background subtraction and shadow detection. In *Proc. IEEE ICCV'99 FRAME-RATE Workshop*, 1999.

[6] H. Saito and T. Kanade. Shape reconstruction in projective grid space from large number of images. In *Proc. the Computer Vision and Pattern Recognition*, June 23-25, 1999.

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

[8] P. Srinivasan, P. Liang, and S. Hackwood. Computational geometric methods in volumetric intersection for 3d reconstruction. *PR*, 23:843–857, 1990.

[9] S. Srivastava and N. Ahuja. Octree generation from object silhouettes in perspective views. *CVGIP*, 49:68–84, 1990.

[10] R. Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Understanding*, July 1993.

[11] R. Tsai. An efficient and accurate camera calibration technique for 3d machine vision. In *Proc. the Computer Vision and Pattern Recognition*, 1986.