Parallel Processing of Range Data Merging

Ryusuke Sagawa [1] **Ko Nishino** [1] **Mark D. Wheeler** [2] **Katsushi Ikeuchi** [1]
[1] Institute of Industrial Science, Univ. of Tokyo
4-6-1 Komaba Meguro-ku,
Tokyo, JAPAN 153-8505
{sagawa,kon,ki}@cvl.iis.u-tokyo.ac.jp
[2] CYRA Technologies, Inc.
8000 Capwell Drive
Oakland, CA 94621
mdwheel@cyra.com

Abstract

This paper describes a volumetric view-merging algorithm that generates a consensus surface of an object from its range images. Our original method merges a set of range images into a volumetric implicit-surface representation, which is converted to a surface mesh by using a variant of the marching-cubes algorithm. We propose a method that increases the computation and memory efficiency for computing signed distances and the method of parallel computing on a PC cluster. Since our method permits a reduction in the data amount allocated in memory, the closest point is searched efficiently; this allows us to increase the number of parallel traversals and to reduce the computation time.

In this paper, we describe the following two algorithms which are complementary in terms of the efficiency of CPUs and memory usage: distributed allocation of range data and parallel traversal of partial octrees. By adjusting them according to the system specifications, we can build the model efficiently by a PC cluster. We have implemented this system and evaluated its performance.

1 Introduction

Integration of multiple range images is important to enable the use of 3D data acquired from stereo systems, laser range finders, etc. It is also fundamental and essential for any algorithms which utilize the generated 3D models, for example, tracking, object recognition and so on.

We have been developing techniques for automatically creating virtual reality models through observation of real objects; we refer to these techniques as modeling-from-reality (MFR). In order to explore unforeseen technical difficulties and to further extend our MFR techniques by solving these difficulties, we have begun a project to model Japanese cultural heritage objects through the use of these MFR techniques[1].

Some Japanese cultural heritage objects are large and their shapes may be intricate. Thus, the models of these objects' shapes must contain huge amounts of data. In our previous experiments in modeling small, indoor objects, we did not have to consider the computation and memory requirements to build those models. However, building a model of a huge amount of data necessitates our taking these requirements into account. In this paper, we describe our proposed method for modeling the shape of huge, possibly intricate, objects.

After scanning the shape of an object by using a range sensor and then aligning all range images into the same coordinate system, our original method[2] converts a set of range images into a volumetric implicit-surface representation, It then obtains a surface mesh using a variant of the marching-cubes algorithm[3]. Unlike previous techniques[4, 5, 6] based on implicit-surface representations, our method estimates the signed distance to the object surface by finding a consensus of locally coherent observations of the surface.

Several approaches which are not based on implicit-surface representation have been proposed [7, 8, 9]. These algorithms perform poorly if the surfaces are slightly misaligned or if there is significant noise in the data.

There are some previous researches which implement the marching-cubes algorithm in a parallel manner [10, 11]. To reduce the computation time for merging range images, the signed distance should be also computed in a parallel manner.

The most costly part of the computation of our method is finding the consensus surface to compute the signed distance. To increase the computation and memory efficiency, we propose a method which reduces the amount of data to be searched, around which point the signed distance is computed.

We utilize octrees to represent volumetric implicit surfaces for effectively reducing the computation and memory requirements of the volumetric representation without sacrificing the accuracy of the resulting surface.

To further ease this size problem, we have developed

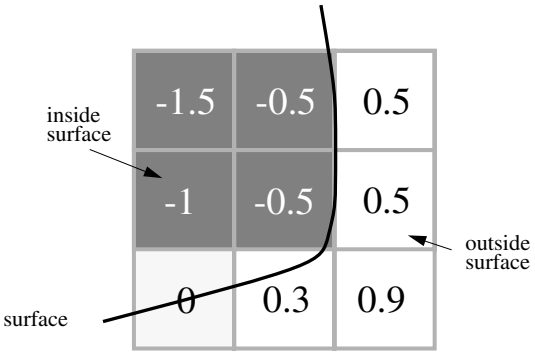

Figure 1: Zero-crossing interpolation from the grid sampling of an implicit surface

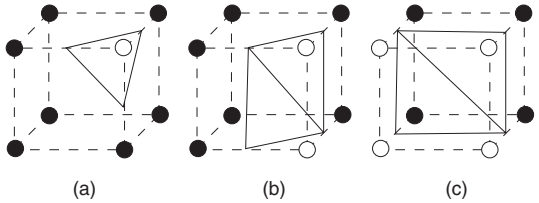

Figure 2: Marching Cubes: An implicit surface is approximated of by triangles. ○: voxels of outside surface. ●: voxels of inside surface.

parallel software that runs on a PC cluster to handle the huge amount of data. The parallel software consists of the following two components: 1. Distributed allocation of range data. 2. Parallel traversal of partial octrees.

In the following sections, Section 2 describes our original merging algorithm. Section 3 explains the method for increasing the computation and memory efficiency. In Section 4, the parallel merging algorithm is shown. Finally, the performance evaluation is shown in Section 5.

2 Data Merging

2.1 Volumetric Modeling and Marching Cubes

Recently, the marching-cubes algorithm[3] has propelled volumetric modeling beyond the confines of "blocky" occupancy grids. Instead of storing a binary value in each voxel to indicate whether the voxel is empty or full, the marching-cubes algorithm requires that the data in the volume grid are samples of an implicit surface. In each voxel, we store the signed distance, $f(\boldsymbol{x})$, from the center point of the voxel, $\boldsymbol{x}$, to the closest point on the object's surface. The sign indicates whether the point is outside, $f(\boldsymbol{x}) > 0$, or inside, $f(\boldsymbol{x}) < 0$, the object's surface, while $f(\boldsymbol{x}) = 0$ indicates that x lies on the surface of the object(See Figure 1).

The marching-cubes algorithm constructs a surface mesh by "marching" around the cubes while following the zero crossings of the implicit surface $f(\boldsymbol{x}) = 0$. The resulting surfaces are relatively smooth and their accuracy can be greater than the resolution of the volume grid due to subvoxel interpolation (See Figure 2).

Now we focus on a more easily solved problem: How do we compute $f(\boldsymbol{x})$? The real problem underlying our simple question is that we do not have a single surface; instead, we have many surfaces. Some elements of those surfaces do not belong to the object of interest but rather are artifacts of the image acquisition process or background surfaces. In the next subsection, we present an algorithm that answers the question and does so reliably in spite of the presence of noisy and extraneous surfaces in our data.

2.2 Consensus Surface Algorithm

This section describes the method for computing the signed distance function $f(\boldsymbol{x})$ for arbitrary points $\boldsymbol{x}$ when given N triangulated surface patches from various views of the object surface. We call our algorithm the consensus-surface algorithm.

We can break down the computation of $f(\boldsymbol{x})$ into two steps:

- Compute the magnitude: compute the distance, $|f(\boldsymbol{x})|$, to the nearest object surface from $\boldsymbol{x}$
- Compute the sign: determine whether the point is inside or outside of the object

The previous naive algorithm finds the nearest triangle from all views and uses the distance to that triangle as the magnitude $|f(\boldsymbol{x})|$. If the normal of the closest surface point is directed toward $\boldsymbol{x}$, then $\boldsymbol{x}$ must be outside the object surface. In Figure 3, the point chosen as the closest point from $\boldsymbol{x}$ does not belong to the real surface. Thus, based on the normal information from the closest point, the algorithm incorrectly considers that $\boldsymbol{x}$ is inside the surface.

Our solution to these problems is to estimate the surface locally by averaging the observations of the same surface. The trick is to specify a method for identifying and collecting all observations of the same surface.

Nearby observations are compared using their location and surface normal. If the location and normal are within a predefined error tolerance (determined empirically), we can consider them to be observations of the same surface. Given a point on one of the observed triangle surfaces, we can search that region of 3D space for other nearby ob-

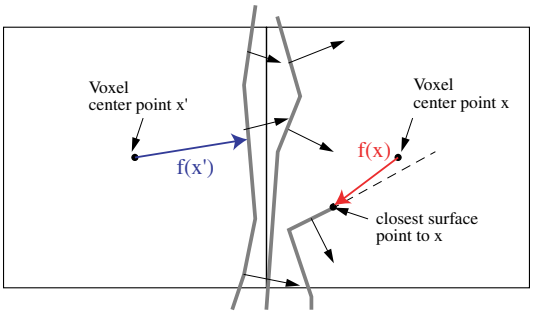

Figure 3: Naive algorithm: An example of inferring the incorrect sign of a voxel's value, $f(\boldsymbol{x})$, due to a single noisy triangle.

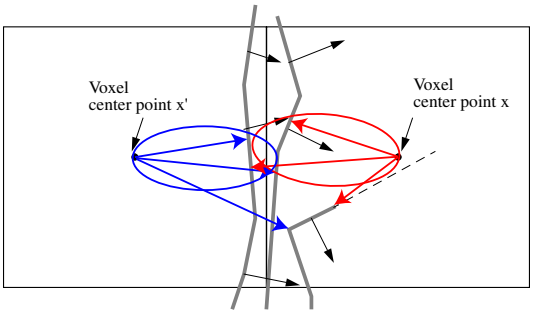

Figure 4: Consensus surface algorithm: The signed distance is chosen from circled consensus surfaces.

servations from other views which are potentially observations of the same surface. These searches are efficiently implemented using k-d trees[12].

The consensus-surface algorithm examines the closest point in each image's triangle set. If there are sufficient surfaces of other triangle sets which are regarded as the same surfaces of each closest point, the closest point is a consensus surface. The algorithm which determines whether two surface observations are sufficiently close in terms of location and normal direction is as follows:

$$\text{SameSurface}(\langle \boldsymbol{p}_0, \boldsymbol{n}_0 \rangle, \langle \boldsymbol{p}_1, \boldsymbol{n}_1 \rangle) = \begin{cases} \text{True} & (\| \boldsymbol{p}_0 - \boldsymbol{p}_1 \| \leq \delta_d) \wedge (\boldsymbol{n}_0 \cdot \boldsymbol{n}_1 \geq \cos\theta_n) \\ \text{False} & \text{otherwise} \end{cases} \quad (1)$$

where δ_d is the maximum allowed distance and θ_n is the maximum allowed difference in normal directions.

For example, consensus surfaces are circled in Figure 4. The algorithm chooses the closest one of them as the signed distance. In this case, it is correctly determined that $\boldsymbol{x}$ is the outside surface and $\boldsymbol{x}'$ is the inside surface.

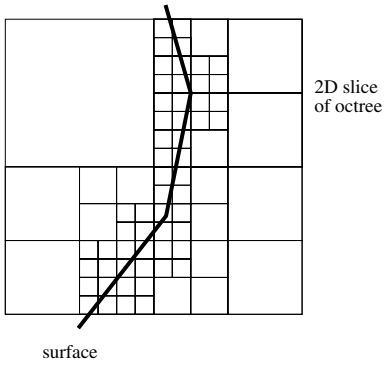

Figure 5: The adaptive resolution is high around the surface and low elsewhere

2.3 Adaptive Resolution by Octree Representation

Volumetric modeling involves a tradeoff between accuracy and efficiency. The octree representation[13] balances this problem while keeping the algorithm implementation simple. Instead of iterating over all elements of the voxel grid, we can apply a recursive algorithm on an octree that samples the volume more finely only when near the surface of the object (See Figure 5).

To interpolate the zero crossings properly, we will need the implicit distance for the voxel containing the surface (the zero crossing) and all voxels neighboring this voxel; these voxels must all be represented at the finest level of precision. This constraint means that, if we have a surface at one corner of an octant, the longest possible distance to the center of a neighboring octant is one and one-half diagonals of the voxel cube, which is a distance of $\frac{3\sqrt{3}}{2}$ cube units.

Given the current octant, we can compute the signed distance. If the magnitude of the signed distance, $|f(\boldsymbol{x})|$, is larger than $\frac{3\sqrt{3}}{2}$ of the octant width, then it is not possible for the surface to lie in the current or neighboring octant. If the surface is not in the current or neighboring octant, we do not care to further subdivide the current octant.

3 Increase the computation and memory efficiency

If the size of mesh data to be merged is huge, it is difficult to allocate all of that data to memory, Also, the computation time of the signed distance cannot be ignored. We propose the following method to increase the computation and memory efficiency by reducing the data allocated in the memory.

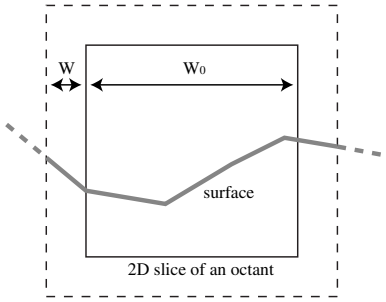

Figure 6: Load only the mesh data within the dotted rectangle into memory

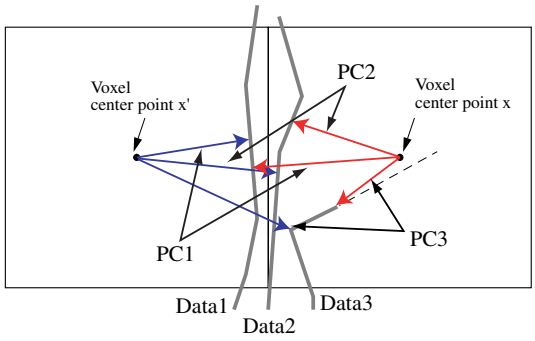

Figure 7: Parallel computation of signed-distances

When the algorithm traverses a part of the octree, the data searched for finding the closest surface is only the local area around the voxel. The data of the other area are never used for computing signed distances while traversing the sub-octree. Moreover, a closest surface is effectively searched using a k-d tree. However, it is inefficient when the k-d tree contains unnecessary data.

As described in Section 2.3, a octant is subdivided when its signed distance is less than $\frac{3\sqrt{3}}{2}$ cube units. Thus, the data farther than $\frac{3\sqrt{3}}{2}$ cube units is not necessary for finding the closest point of the voxel.

To load the necessary data into memory, we must read all of the data files. Since the overhead of reading files for the every finest octant is too costly, we read the data files for an ancestor octant. Where the width of an ancestor octant is W_0 and the width of the finest octant is W, the area of the mesh data to be loaded is inside the rectangle of a dotted line in Figure 6.

4 Parallel Computing of Signed Distances

In this section, we describe the algorithm for parallel computing of signed distances. There are two motivations for parallel computing signed distances. We now propose the parallel computing method for each motivation:

1. Handling range data of huge size: We distribute the allocation of range data to multiple PCs.

2. Fast merging: We divide the octree into sub-octrees and assign traversal of a sub-octree to each CPU.

4.1 Distributed Allocation of Range Data

Calculating a signed distance from a point requires consideration of all range data with respect to this point. When the number of the measurement increases, more data should be considered. It becomes difficult to allocate all the range data in a single processor.

We distribute that range data to multiple PCs and compute signed distances in a parallel manner. For example, in Figure 7, Data 1,2,3 are allocated to PCs 1,2,3, respectively. Signed distances from the point, x, to Data 1 are computed by PC1. In the same manner, signed distances to Data 2 are computed by PC2, and so on. Since finding the closest point of a mesh data is independent of the others, we can compute signed distances in a parallel manner.

However, the computation times are different among CPUs; After finding the closest points of all data, we have to choose the smallest magnitude of the signed distances. To synchronize, until the remaining CPUs finish computing the signed distances.

4.2 Parallel Traversal of an Octree

Dividing an octree into partial trees enables us to traverse the partial trees. We assign the partial space of an octree to each CPU and traverse partial trees in a parallel manner (See Figure 8). Since the traversals of partial trees are independent of one another, a traversal does not have to synchronize with others, and the computation time can be reduced according to the number of CPUs.

By the method described in Section 3, the area of range data which each process owns is only inside the voxel and its peripheral area. Thus, each process owns only the range data of the local area which it takes charge of in a traversal of a partial octree.

However, each machine must cache range data files in memory to read them efficiently and repeatedly. Since a PC cluster cannot share data as a shared-memory machine can, range data files have to be allocated redundantly; therefore, memory efficiency dwindles as this parallel traversal method is used.

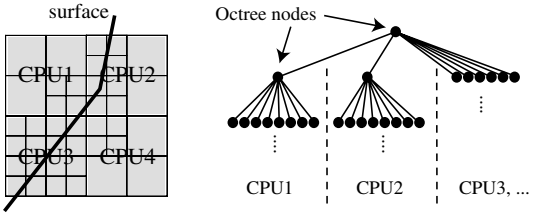

Figure 8: Assignment of partial space of the octree to each CPU and parallel traversal of partial trees

4.3 Combination of Parallel Methods

The above two methods are complementary in terms of the efficiency of CPUs and memory usage. In practice, they should be adjusted according to the system specification by combining those two methods with an appropriate condition. Two methods can be combined by allocating range data distributed in each parallel traversal.

The maximum number of traversals is determined by the system memory size. Thus, the combination strategy maximizes the number of traversals to deal with the memory. If the system has more CPUs than the parallel traversals, each traversal uses multiple CPUs by the method of distributed allocation.

5 Performance Evaluation

We have implemented these algorithms, and constructed one integrated digital Great Buddha of Kamakura. For this project, we have built a PC cluster that consists of eight PCs of dual PentiumIII 800MHz processors with 1GB memory for each PC. The machines are connected by 100BASE-TX Ethernet. Figure 9 shows the obtained geometric model of the Great Buddha; the model contains 3 million points and 5.5 million triangles.

We tested the merging program by changing parameters of the number of traversals and machines of each traversal. Raw data consists of 12 files; of those files, the average contains about 300 thousand points and 600 thousand triangles. The total size is about 150M bytes. The result is shown in Table 1.

Without reducing the data allocated in the memory, the maximum number of the traversals is four because of the system memory size. It takes 59 hours to build the model where it is computed by 4 traversals that are allocated and distributed to 4 PCs. It has been proven that the method of reducing the data allocated in the memory increases the computation and memory efficiency. After reducing data, we can compute the signed distances by a single machine; the computation time is 468 minutes.

The algorithm without parallel processing is equal to computing by one traversal using a machine (row A). If the system computes signed distances with the distributed allocation of memory (row C,E,F), the required memory for each machine is less than row A. The reciprocal of required memory of each machine is proportional to the number of machines in each traversal (See Figure 11). Next, if the system computes with parallel traversals (row B,D,F,G), the computation time is less than row A. The reciprocal of computation time is almost proportional to the number of parallel traversals(See Figure 10).

When the memory allocation is distributed to a small number of machines, it computes faster as the number increases. In this case, row C computes faster than row A, also row F faster than row D. However, when the memory allocation is distributed to a large number of machines, the computation becomes slower because of waiting synchronization. In this case, row E is slower than row C.

According to the combination strategy, the signed distances are computed by 16 parallel traversals that are allocated to each PC to minimize the computation time for our PC cluster. Now we consider the combination of systems of different memory size for computing signed distances of the Kamakura Buddha model: First, if the memory of each PC is less than 200MB, the number of distributed allocation must be larger than 2 machines, like row C and E. When each traversals is distributed to 4 machines, the number of traversals is determined to be 4 to minimize the computation time. Next, if the memory of each PC is 200-256MB, each traversal should be distributed to 2 machines. Then, the number of traversals is determined to be 2.

6 Conclusion

In this paper, we have proposed a method which increases the computation and memory efficiency of computing signed distances, along with a method for parallel computing using a PC cluster. First, since we reduce the data allocated in the memory, the closest point is searched efficiently. Thus, we can increase the number of the parallel traversals and reduce the computation time.

In addition, we have described two algorithms which are complementary in terms of the efficiency of CPUs and memory usage. By adjusting them according to the system specifications, we can build the model efficiently by using a PC cluster.

Now we can build models of huge size. In the future, we plan to scan more Japanese cultural heritage objects and build fine models with photometric attributes.

Figure 9: Merging result of Kamakura Great Buddha

Table 1: Results of different parameters of the number of traversals and machines of each traversal.

	Number of traversals	Number of machines in each traversal	Average required memory of each machine	Computation Time
A	1	1	200MB	468 min.
B	4	1	200MB	117 min.
C	1	4	50MB	215 min.
D	8	1	200MB	58 min.
E	1	8	20MB	256 min.
F	8	2	200MB	44 min.
G	16	1	250MB	23 min.

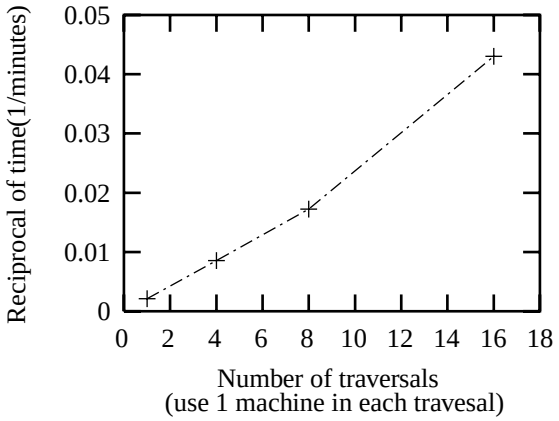

Figure 10: The reciprocal of computation time is proportional to the number of parallel traversals.

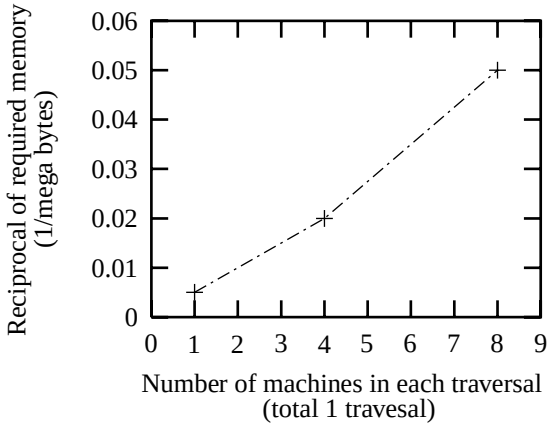

Figure 11: The reciprocal of required memory of each machine is proportional to the number of machines in each traversals.

References

[1] Daisuke Miyazaki, Takeshi Ooishi, Taku Nishikawa, Ryusuke Sagawa, Ko Nishino, Takashi Tomomatsu, Yutaka Takase, and Katsushi Ikeuchi. The great buddha project: Modelling cultural heritage through observation. In *Proceedings of 6th International Conference on Virtual Systems and MultiMedia*, pp. 138–145, Gifu, 2000.

[2] M.D. Wheeler, Y. Sato, and K. Ikeuchi. Consensus surfaces for modeling 3d objects from multiple range images. In *Proc. International Conference on Computer Vision*, January 1998.

[3] W. Lorensen and H. Cline. Marching cubes: a high resolution 3d surface construction algorithm. In *Proc. SIGGRAPH'87*, pp. 163–170. ACM, 1987.

[4] H. Hoppe, T. DeRose, T. Duchamp, J.A. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proc. SIGGRAPH'92*, pp. 71–78. ACM, 1992.

[5] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proc. SIGGRAPH'96*, pp. 303–312. ACM, 1996.

[6] A. Hilton, A.J. Stoddart, J. Illingworth, and T. Windeatt. Reliable surface reconstruction from multiple range images. In *Proceedings of European Conference on Computer Vision*, pp. 117–126, Springer-Verlag, 1996.

[7] M. Soucy and D. Laurendeau. A general surface approach to the integration of a set of range views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17, No. 4, pp. 344–358, April 1995.

[8] M. Rutishauser, M. Stricker, and M. Trobina. Merging range images of arbitararily shaped objects. In *Proceedings of 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 573–580, June 1994.

[9] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of SIGGRAPH'94*, pp. 311–318. ACM, 1994.

[10] D. Bartz and W. Straßer. Parallel construction and isosurface extraction of recursive tree structures. In *Proceedings of WSCG'98*, Vol. III, Plzen, 1998.

[11] P. Mackerras. A fast parallel marching-cubes implementation on the fujitsu ap1000. Technical report, Australian National University, TR-CS-92-10, 1992.

[12] Jerome H. Friedman, Jon Bentley, and Raphael Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, Vol. 3, No. 3, pp. 209–226, 1977.

[13] D. J. R. Meagher. *The octree encoding method for efficient solid modeling.* PhD thesis, Rensselaer Polytechnic Institute, 1980.