Distributed Volumetric Scene Geometry Reconstruction With a Network of Distributed Smart Cameras

Shubao Liu † Kongbin Kang † Jean-Philippe Tarel ‡
David B. Cooper †
†Division of Engineering, Brown University, Providence, RI 02912
{sbliu, kk, cooper}@lems.brown.edu
‡Laboratoire central des Ponts et Chaussées (LCPC), Paris, France
jean-philippe.tarel@lcpc.fr

Abstract

Central to many problems in scene understanding based on using a network of tens, hundreds or even thousands of randomly distributed cameras with on-board processing and wireless communication capability is the "efficient" reconstruction of the 3D geometry structure in the scene. What is meant by "efficient" reconstruction? In this paper we investigate this from different aspects in the context of visual sensor networks and offer a distributed reconstruction algorithm roughly meeting the following goals: 1. Close to achievable 3D reconstruction accuracy and robustness; 2. Minimization of the processing time by adaptive computing-job distribution among all the cameras in the network and asynchronous parallel processing; 3. Communication Optimization and minimization of the (battery-stored) energy, by reducing and localizing the communications between cameras. A volumetric representation of the scene is reconstructed with a shape from apparent contour algorithm, which is suitable for distributed processing because it is essentially a local operation in terms of the involved cameras, and apparent contours are robust to our-door illumination conditions. Each camera processes its own image and performs the computation for a small subset of voxels, and updates the voxels through collaborating with its neighbor cameras. By exploring the structure of the reconstruction algorithm, we design the minimum-spanning-tree (MST) message passing protocol in order to minimize the communication. Of interest is that the resulting system is an example of "swarm behavior". 3D reconstruction is illustrated using two real image sets, running on a single computer. ***The iterative computations used in the single processor experiment are exactly the same as are those used in the network computations.*** *Distributed concepts and algorithms for network control and communication performance are theoretical designs and estimates.*

1. An Overview of the System

1.1. Motivation

With the recent development of cheap and powerful visual sensors, wireless chips and embedded systems, cameras have enough computing power to do some on-board "smart" processing. These "smart cameras" can form a network to collaboratively monitor, track and analyze the scenes of interest. This area have drawn a lot of attention in both academia and industry over the past years (see [1] [11] and [13] for an overview). However compared with the maturity and availability of the camera network hardware, the software capable of fully utilizing the huge amount of visual information is greatly under-developed. This has become the bottleneck for the wide deployment of the smart camera network (also called visual sensor network, VSN). There is an obvious demand to synchronize the recent development of vision algorithms with the development of the visual sensor network hardware. Our paper presents a completely new and natural approach to 3D reconstruction within a smart camera network.

1.2. The Goal

Our goal is minimum-error 3D scene reconstruction based on edge information with N calibrated smart cameras through their collaborated distributed processing, as illustrated in Fig. 1. A Bayesian approach is taken to 3D reconstruction, where the surface is treated as a stochastic process modelling the smoothness of the surface. Thanks to the apparent contours' robustness to environmental factors, shape-from-apparent-contours is more suitable for outdoor distributed camera applications than the intensity-based multi-view reconstruction. The representation for the estimated surface is a discretized level set function defined on a grid of voxels. The cost function to be minimized is the sum of the area of the 3D surface and the integral of "consistency" between the apparent contour of the current surface estimate and the image edges. The object surface is to be reconstructed distributedly with N smart cameras

co-operating to minimize both the processing time and the consumed on-board battery energy. Computation and communication load-balancing are investigated to make battery usage roughly at the same level over all the cameras.

1.3. 3D Surface Reconstruction

The proposed surface estimation procedure is iterative through numerical solution of the first order variation of the energy functional (i.e., the cost function). It turns out that each iteration is a *linear* incremental change of the current estimated surface. All of the computation takes place within a thin band around the estimated surface. For each camera c, the data and information available for the $(t+1)$th iteration is: its projection matrix; the edges in its image; and a subset of voxels that this camera maintains. The incremental update is the sum of two increments. The first increment comes from the contribution of the image edge data. This increment attempts to align the contour generators of the estimated 3D surface with the edge-data in the observed image. The second increment is the contribution of the *a priori* stochastic model of the 3D surface. Hence, for each voxel on the estimated 3D surface at the start of an incremental surface update-iteration, the cameras contributing to the voxel update are the ones whose contour generators are close to that voxel. A voxel is in the primary responsibility set (PRS) of each camera whose image information contributes to the voxel's updating. Each contributes to the first update-increment. One of these cameras takes responsibility for computing the second update-increment. This group of cameras each has a record of the changes made, and therefore of the total update change made. For a 3D surface voxel not contributed by any camera at the start of an update-iteration, there is no first incremental-update, and one of the cameras takes responsibility for computing and communicating the second incremental-update. This voxel is in the second set of the responsible cameras (SRS). Fig. 2 illustrates these concepts on a sphere shape.

1.4. Distributed Processing

It is know that that the battery power for two wireless cameras to communicate is approximately proportional to the square of their distance. Hence, rather than two cameras communicating directly, the signals from the transmitting camera travels to the receiving camera through a sequence of relays where it travels from one camera to a camera that is close, then to another close camera, etc. In this way, communication power increases linearly with distance between cameras. This is a *camera communication network* (CCN) optimization problem that finds the best routing for each communication, i.e. how to send messages.

For our purpose, only the end-to-end communication in the application layer is considered. Inspired by the observation that the communication in the proposed algorithm works more like broadcast (although not exactly, which will be discussed later) than point-to-point ad-hoc communication, we can optimize the communication further by deciding *what to send* and *who to send to*, instead of only optimizing *how to send*. This results in an efficient message passing protocol based on the minimum-spanning-tree of the *camera reconstruction network* (CRN, the exact meaning will be discussed later.)

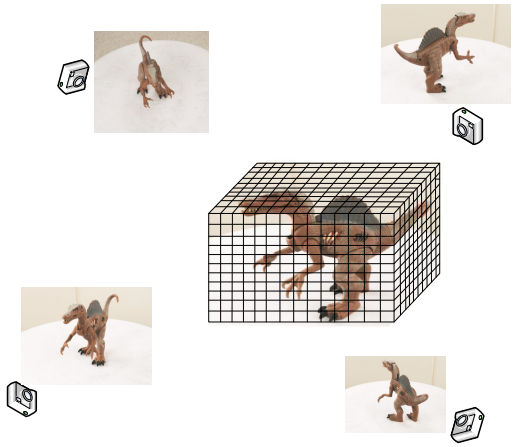


Figure 1. Volumetric world, smart cameras and their observations.

Distributing the voxel updating job among all the smart cameras to enable parallel processing is achieved by each camera processing those voxels in its primary and second responsibility sets. These sets for the various cameras are close enough in size such that the partition results in balanced parallel processing. A camera determines its secondary responsibility set through negotiating the boundaries with its neighbor cameras in the CRN. Also incurred is battery energy for the communications in determining the secondary responsibility set. Rough minimization of communications battery energy is achieved by routing communications over paths contained in an MST (Minimum Spanning Tree). Also some communication is required among cameras having primary sets that are close in order for the cameras to figure out their secondary responsibility sets.

2. Shape From Apparent Contours

A shape-from-apparent-contours algorithm is first developed to reconstruct the 3D shape from apparent edges in different views. The algorithm also incorporates the prior knowledge about the surface (e.g., surface smoothness) to produce a complete shape. The proposed algorithm combines the ideas in 2D active contoursand variational surface reconstruction [7, 6, 15, 9] based on implicit surface deformation. In active contour fitting, the best curve $C^\star$ is found by deforming a curve $C(s)$ to make it fit the object boundaries:

$$C^\star(s) = \arg\min_{C(s)} \mathcal{E}(C(s)).$$

The functional $\mathcal{E}(C(s))$ is usually defined as

$$\mathcal{E}(C(s)) = \mu \int_0^1 C(s)ds - \int_0^1 ||\nabla_G I(C(s)||ds \qquad (1)$$

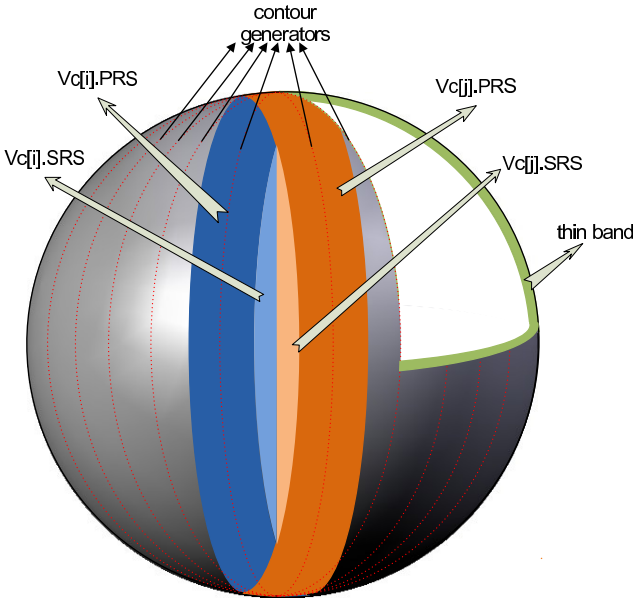


Figure 2. Illustration of the key concepts (including contour generators, band, PRS, SRS) in the distributed reconstruction algorithm, with a simple setting (a sphere shape and evenly distributed cameras around the equator of the sphere.

where ds is the infinitesimal curve length, $\nabla_G I(C(s)) = \nabla(G * I(C(s)))$ is the data term measuring the influence of the image intensity gradient along the fitted curve, $G * I$ is the convolution of the intensity image I with a Gaussian filter G, μ is a scalar value controlling the influence of the length of the curve.

Apparent contours are curves coming from contour generators on the surface through perspective projection. So instead of assuming that the contours can be deformed freely, we constrain them with a 3D surface:

$$C_i(s) = \Pi_i(\mathcal{G}_i(s)), \tag{2}$$

where Π_i is the ith camera's perspective projection, which maps a 3D point $\mathbf{X}$ to a 2D image point $\mathbf{x}_i$; $C_i(s)$ is the apparent contour in image i, $\mathcal{G}_i(s)$ is the corresponding contour generator on the surface, as shown in Fig. 2. Notice that

$$\int_0^1 ||\nabla_G I(C_i(s))||ds = \int_{\mathcal{S}} \mathbf{1}_{\mathcal{G}_i}(\mathbf{X})||\nabla_G I_i(\Pi_i(\mathbf{X}))||dA \tag{3}$$

where $\mathcal{S}$ is the surface. Equation (3) turns the line integral to a surface integral with the introduction of the contour generator indicator function $\mathbf{1}_{\mathcal{G}_i}(\mathbf{X})$, which is a impulse function. (In experiment, it is approximated with a Gaussian function.) Through the occluding geometry relationship between the surface normal $\mathbf{N}$ and the tangent plane $\bar{\mathbf{N}}_i$ (got from back-projecting of the tangent line of the apparent contours), we extend (3) to

$$\int_{\mathcal{S}} \mathbf{1}_{\mathcal{G}_i}(\mathbf{X})||\nabla_G I(\mathbf{x}_i)|| \cdot |\bar{\mathbf{N}}_i^T \mathbf{N}|dA \tag{4}$$

to further enforce the tangency constraint. The higher order term $|\bar{\mathbf{N}}_i^T \mathbf{N}|$ make sthe shape evolution converge faster and more accurate.

The surface to be reconstructed $\mathcal{S}^\star$ is the optimal surface that minimizes an energy functional in the form of a weighted area, with the weights depending on the M observed images as in (4) and a prior term:

$$\begin{aligned} \mathcal{E}(\mathcal{S}) &= \int_{\mathcal{S}} \Phi(\mathbf{X}, \mathbf{N}) dA \\ &= \int_{\mathcal{S}} \left(\sum_i^M \mathbf{1}_{\mathcal{G}_i}(\mathbf{X})||\nabla_G I(\mathbf{x}_i)|| \cdot |\bar{\mathbf{N}}_i^T \mathbf{N}| + \mu \right) dA, \end{aligned} \tag{5}$$

where dA is an infinitesimal area element, μ is a parameter controlling the smoothness of the surface. Interpreted in Bayesian language, the prior energy term $\int_{\mathcal{S}} \mu dA$ corresponds to a prior probability $\frac{1}{Z} e^{-\mu Area((S))}$, which is the energy representation of a 1st-order Continuous Markov Random Field, encouraging smooth surfaces instead of bumpy ones. The functional (5) is minimized through gradient descent methods by computing the first order variation. The gradient descent flow for (5) can be written as [7]:

$$\mathcal{S}_t = F\mathbf{N}, \tag{6}$$

$$F = 2\kappa\Phi - \langle \Phi_{\mathbf{X}}, \mathbf{N} \rangle - 2\kappa \langle \Phi_{\mathbf{N}}, \mathbf{N} \rangle. \tag{7}$$

where κ is the mean curvature of the surface $\mathcal{S}$. With the level set representation, $\mathcal{S} = \{\mathbf{X} : \phi(\mathbf{X}) = 0\}$, the above evolution equation can be rewritten as:

$$\phi_t = F||\nabla\phi||. \tag{8}$$

Through some calculus derivation, we get the speed function for (8) as

$$F = 2\mu\kappa - \sum_{i=1}^{M} \langle \Phi_{i\mathbf{X}}, \mathbf{N} \rangle. \tag{9}$$

3. Distributed Algorithm for Scene Reconstruction

In the above, we have briefly described a centralized algorithm for shape from apparent contours, where one central processor collects data from all cameras and processes them in batch. In the visual sensor network applications, distributed algorithms are preferred, where each smart camera runs identical programs but with different states and different image inputs. In this section we show that the proposed algorithm can be run distributedly on the smart camera network by augmenting the algorithm with a job division module and a communication module. Principally the algorithm can be extended distributedly because: (1) the algorithm reconstructs the contour generators, and the other part of the surface is interpolated through the prior energy, equivalently, a priori stochastic model for the 3D surface. (2) It has been shown that the contour generators can be reconstructed locally by studying the differential geometry of the apparent contour change [4, 3, 10].

$\mathcal{V}_c$	the set of voxels that the camera c maintains
$\mathcal{C}_v$	the set of cameras that maintains voxel v
F_v^c	a scalar representing camera c's contribution to voxel v's updating
PRS_c	the primary responsible set of camera c
SRS_c	the secondary responsible set of camera c, $PRS_c \cup SRS_c = \mathcal{V}_c$

Table 1. Main notation summary

voxel ID	voxel value	neighbor cameras in MST
1001	1.302	{2, 30}
2187	-2.630	...{10}
...	...	...
PRS: {1001, ...}		
SRS: {2187, ...}		
watching voxel list: {...}		
boundary voxel list: {...}		

Table 2. An example of the data structures that each camera maintains.

To highlight the structure of the reconstruction procedure, we summarize each voxel's updating with this formula:

$$\phi_v^{t+\Delta t} = \phi_v^t + (2\mu\kappa + \sum_{c\in\mathcal{C}_v} F_v^c)||\nabla\phi||\Delta t, \qquad (10)$$

where $\mathcal{C}_v$ is the set of cameras c that has $F_v^c \neq 0$ for voxel v, F_v^c is the speed contribution from camera c to voxel v:

$$F_v^c = -\langle \Phi_{i\mathbf{X}(v)}, \mathbf{N}(v)\rangle. \qquad (11)$$

Formula (10) describes the updating operation for each voxel. A naïve parallel implementation of the algorithm is to divide the entire set of voxels into M (the number of smart cameras) subsets, and each camera takes care of one subset of the voxels. The problem with this naïve approach is that (1) each camera needs to maintain a copy of all the other cameras' observed images. This implies a huge amount of data communication, which prevents the algorithm scaling up to a large camera network; (2) the contour generators dynamically change as the surface shape evolves. So fixing the set of voxels that each camera maintains requires distant cameras to exchange information about voxels' states and image observations. This prevents the communication between cameras from being localized.

Instead we build a camera-centric distributed algorithm, in which each camera c maintains a gradually changing dynamic subset $\mathcal{V}_c$ of voxels around the current estimated contour generators seen by this camera. Algorithm 1 describes the over-all procedure in a high level, with each subroutine being discussed in detail later in Algorithms 2 and 5. Through each camera maintaining a subset of voxels $\mathcal{V}_c$ and localizing the computation and communication, Algorithm 1 has good scalability with respect to the number of cameras and the resolution of the volumetric representation. In the following, we elaborate on different aspects of the distributed algorithm, including complete surface coverage,

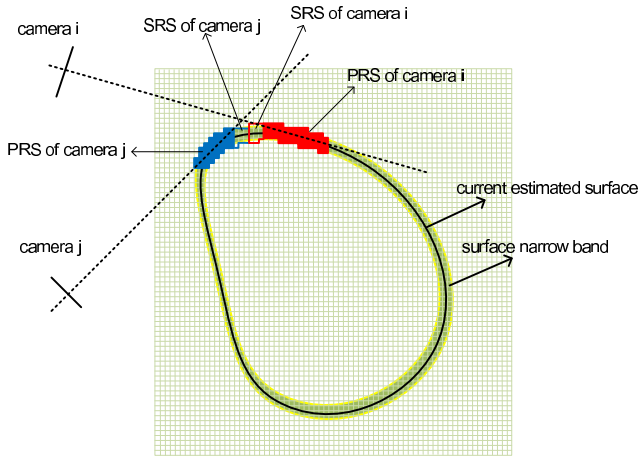


Figure 3. Illustration of job distribution scheme in 2D case. The light-green strip indicates the narrow band. The dark blue indicates the PRS of camera 1; the shallow blue indicates the SRS voxels of camera 1. The dark red indicates the PRS of camera 2; the shallow red indicates the SRS of camera 2.

computation load balancing among cameras, communication optimization, etc. For the sake of clarity, Table 1 summarizes the main notation used in the following discussion; And Table 2 shows the main data structures that each camera maintains to support the distributed algorithm. The usages of these data structures is discussed below.

Algorithm 1 Camera-centric distributed algorithm for scene geometry reconstruction

1: **for** each smart camera c, **do**
2: Compute the incremental updates F_v^c, $\forall v \in \mathcal{V}_c$, according to formula (11). If $\max_{v\in\mathcal{V}_c} |F_v^c| < \varepsilon$ (where ε is a stop criterion threshold), then terminate.
3: Send F_v^c to all the cameras in $\mathcal{C}_v$, $\forall v \in \mathcal{V}_c$ through a minimum-spanning-tree (MST) message passing protocol as described in Algorithm 5.
4: Update each voxel's level set value according to formula (10), after receiving messages from the other tree branches of this node in the MST, as described in Algorithm 5.
5: Update the voxel set $\mathcal{V}_c$ as described in Algorithm 2.
6: **end for**

3.1. Job Distribution Scheme

In the level set method ([12, 14]), a narrow band implementation is commonly used to save memory and computation. It is based on the fact that only the voxels around the surface (zero-level set) contribute to the shape evolution. So in the implementation, a band Ω around the surface $\mathcal{S}$ is defined with an interval $[DL, DH]$ on each voxel' level set function value and only the voxels inside the band are updated (see Fig. 2). The price for this is that after each iteration the band should be updated to keep the new surface always inside the band through keeping a *watching list* of voxels, which keep track of the boundary of the narrow band. As illustrated in Fig. 2 (3D version) and Fig. 3 (2D version), we need to further divide the band into patches so that each camera takes care of one patch. Each patch should contain at least all the "core voxels" — those voxels around its contour generator defined by the contour generator indicator function. The set of "core voxels" are called the *Primary Responsible Set* (PRS); (2) Each patch should include some "free voxels" — those voxels around the core voxels that are not taken care of by any other cameras. These "free voxels" hosted by camera c belong to the *Secondary Responsible Set* (SRS) of camera c. To effectively distribute the reconstruction job among the cameras, there are three criteria that the job division scheme should address:

$$PRS_c \in \mathcal{V}_c \quad \text{(correctness)} \tag{12}$$

$$\cup_c \mathcal{V}_c = \Omega \quad \text{(complete coverage)} \tag{13}$$

$$|\mathcal{V}_c| \text{ is approximately equal} \quad \text{(load balance)} \tag{14}$$

Eqn. (12) guarantees the correctness of the speed computation F_v^c; Eqn. (13) ensures that all voxels inside the narrow band Ω are updated. With the satisfaction of (12) and (13), the "free" voxels are distributed with the consideration of load balance among cameras with the Algorithm (4).

Algorithm 2 Update the voxel set $\mathcal{V}_c$ for each camera c

1: Update the PRS of camera c as described in Algorithm 3.
2: Update the SRS of camera c as described in Algorithm 4.

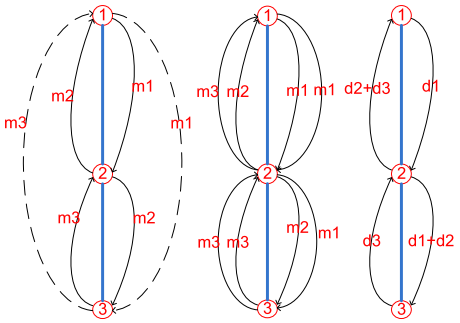
As described in Algorithm 1, after each iteration, for each camera c, its voxel set $\mathcal{V}_c$ (composed of PRS and SRS) should be updated. First each camera's new PRS can be computed easily, given the new detected contour generator, through narrow band updating, as described in Algorithm 3. Besides PRS, there are other portions of the surface that are not covered by any camera. To ensure that these "free" voxels are updated correctly, we need to assign them to some host cameras. These "free" voxels are put in the SRS of their corresponding host cameras. There are two considerations in these voxels' distribution: These voxels may belong to neighbor cameras' PRS in the next iterations, so if we could put these voxels to these potential cameras then we can save the communications later; Another concern is the load balance. Due to the non-uniformity of the surface and the distribution of the cameras, the size of the PRS for each camera is different. The existence of these "free" voxels provides us a leverage to balance the workload among cameras. The PRSs are fixed for the given surface and the cameras' locations; The SRSs are flexible as long as together with PRS they cover the whole surface. We can take advantage of this to assign these "free" voxels to the cameras that have relatively small PRS's. The workload balances are negotiated pairwisely by neighbor cameras that share boundaries, as described in Algorithm 4. The communications in Algorithm 4 happens in two steps: 1) communication between c and c' when checking $v \in PRS_{c'}$; 2) communication between c and c' when checking $|\mathcal{V}_{c'}| < |\mathcal{V}_c|$. Since this operation is performed for each boundary voxel, the communication cost is proportional to the number of boundary voxels.

Algorithm 3 Update the PRS

1: % update the narrow band
2: **for** each voxel v in the watching list, **do**
3: **if** its level set function value $\phi(v) \in [DL, DH]$ **then**
4: expand the boundary voxels by adding the neighbor voxels whose level set function's absolute values are greater than $|\phi(v)|$.
5: **else**
6: delete this voxel.
7: **end if**
8: **end for**
9: Update the contour generator indicator values $\mathbf{1}_{\mathcal{G}_c}(v)$, $\forall v \in \mathcal{V}_c$ for camera c. Put voxels whose indicator value is above a threshold T_G into the new PRS.

Algorithm 4 Update the SRS for camera c

1: % update the boundary list
2: **for** each boundary voxel v, **do**
3: **for** each $c' \in \mathcal{C}_v$, **do**
4: **if** $v \in PRS_{c'}$ **then**
5: delete v from $\mathcal{V}_c$; Add its neighbors to the boundary voxel list.
6: **end if**
7: **end for**
8: **end for**
9: % At this stage, each boundary voxel has only two hosts.
10: % Now start pairwise load balance.
11: **for** each voxel v in the boundary list, **do**
12: $c' = \mathcal{C}_v \backslash c$,
13: **if** $|\mathcal{V}_{c'}| < |\mathcal{V}_c|$ **then**
14: delete v from $\mathcal{V}_c$, and add its neighbors to the boundary voxel list.
15: **end if**
16: **end for**



(a) (b) (c)

Figure 4. Illustration of a simple communication case. (a) the virtual communication path in the naïve approach; (b) the physical communication path in the naïve approach; The communication cost is 8 units; (c) the physical communication path in the MST case; The communication cost is 4 units.

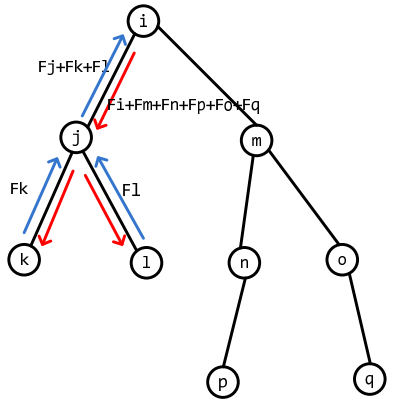


Figure 5. Illustration of the minimum spanning tree message passing. Each node sends a message to one of its edges given the message from the other edges have arrived.

3.2. Communication Optimization

As discussed above, cameras need to communicate with each other locally to share information about their common voxels and dynamically assign work loads among cameras. Here we examine the problem of optimizing the communications between these cameras. From the above description (especially in (10)), we know that each voxel's incremental update is composed of the summation of the participating cameras' contributions. So the basic communication job is: sending each camera c's incremental updating contribution F_c^v to all the other cameras in $\mathcal{C}_v$. Now let us analyze the communication cost of the naïve approach – each camera sends its own value F_v^c to all other cameras in the set $\mathcal{C}_v$ directly. Suppose the communication cost between neighbor cameras in the graph is 1 unit. For a random graph, the average communication complexity for one message passing is $O(D) = O(log(N))$, where D is the diameter of the communication graph of the network. Then, the total average communication complexity is $O(N^2 log(N))$. The worst case for one message passing is N, with the worst total communication complexity being N^3.

Instead of sending F_v^c directly to all the other cameras in C_v, there exists a more efficient way. Look at what each camera needs — the *summation* of F_c^v from all the participating cameras $c \in \mathcal{C}_v$. Based on this observation, our solution is the tree message passing protocol, as described in Algorithm 5 and illustrated in Fig. 5. We store this tree representation of the CRN distributedly, through each camera maintaining a list of directly connected camera nodes for each voxel, as shown in Table 2. Why does the message passing work correctly for the tree structure? This is because there is "no loop" in the tree, which guarantees that cutting each edge will separate the tree into two separate subtree. And the message sent through the edge is all the summed information from the subtree. In this way, each node's value is contributed to other nodes exactly once. Take the tree in Fig. 5 for example. For node j, it will receive message from k, l and i. And each message from k, l, i is the summation of the values in their subtrees $\{k\}$, $\{l\}$, and $\{i, m, n, o, p, q\}$.

Algorithm 5 Tree message passing protocol

1: **for** each node in the tree, **do**
2: Compute and send message to one edge if the messages from the other edges have been received;
3: Otherwise, wait.
4: **end for**

Next the communication cost of the tree message passing scheme is analyzed. For a tree with N nodes, there are $(N-1)$ edges and since we send information bidirectionally, the communication cost is $2(N-1)$ units. Given the set of cameras $\mathcal{C}_v$ for a fixed voxel v, there are many trees that can be constructed; which one is the best? Given a weighted undirected graph G, we define a *minimum spanning tree* (MST) as a connected subgraph of G for which the combined weight of all the included edges is minimized. In our case, the minimum spanning tree is the one that has the minimum communication cost. Since voxel updating is a key operation in the algorithm, the improvement on this operation will greatly speed up the algorithm. The tree message passing is very useful for distributed smart camera systems in general, since it is a common operation to summarize message in one subgraph and send it to the other branch. The tree message passing ensures that the protocol described above works correctly for tree topology structure. The MST can also be constructed and updated distributedly (See [2, 8, 5] for more details.)

With this MST protocol described in Algorithm 5, we can see that each camera updates its own copy of voxels only after receiving messages from all its neighbor cameras. By this, there is no need to synchronize among the cameras after each iteration. Each camera runs its own algorithm and updates its own state only after it receives all the information needed asynchronously. And the synchronization is implicitly controlled by the message passing.

Figure 6. Two sample images of the Toy Dinosaur dataset

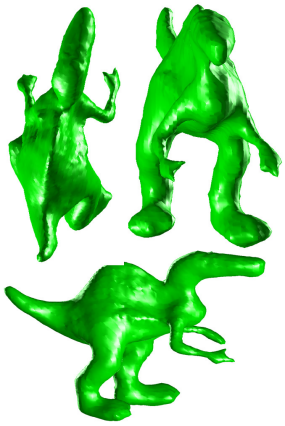


Figure 7. The reconstructed dinosaur shape

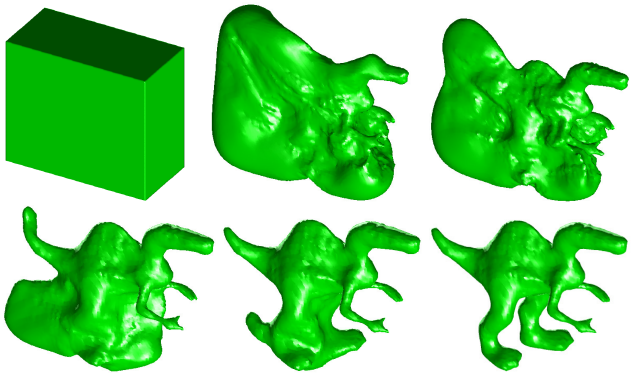


Figure 8. Shape evolution path of the Toy Dinosaur

4. Experimental Results

We first test the proposed algorithm on a public dataset, Toy Dinosaur [1]. In Fig. 6, two sample images out of a total of 23 images are shown. In this dataset, the background is relatively simple. The level set function is defined on a $56 \times 120 \times 96$ grid, μ is set as 0.01 (a small value to prevent smoothing out the dinosaur's high curvature parts). Fig. 7 shows the reconstructed shape after 200 iterations. From the results, we see that the overall shape is successfully reconstructed. Fig. 8 shows the whole shape evolution process, starting from a bounding rectangular box. It successfully converges to the concave parts, e.g. recovering the two hands, and separating two legs, etc. The reconstruction accuracy is measured by the projection error, which is defined as the distance between the projected apparent contour and the image apparant contour. The average projection error for this dataset is 0.21 pixels.

The next experiment is on the David bust dataset which consists of 20 calibrated images taken by one moving real camera. Fig. 9 shows samples of the image sequence.

[1]This dataset is available at `http://www-cvr.ai.uiuc.edu/ponce_grp/data/mview/`.

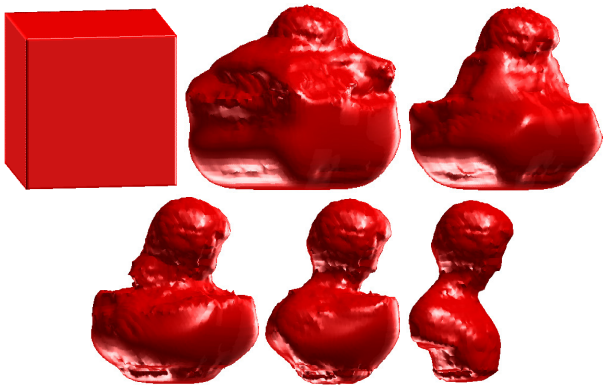


Figure 10. Shape evolution path of the David bust

This dataset is challenging in two aspects. First the object is textureless, non-Lambertian, and the illumination changes (due to flash light), which challenges most multi-view stereo algorithms based on intensity matching. Secondly the object is embedded in an natural indoor background. In the experiment on this dataset, the level set function is defined on a $64 \times 64 \times 64$ grid, and the parameter μ is set as 0.05. The projection error for the David dataset is 0.37 pixels. The projected 3D reconstructed apparent contours are shown in Fig. 9. Fig. 10 shows the whole evolution path, starting from a cubic bounding box. It can be seen that the shape evolution process does converge to the object even though the background in the image is complex.

A rough estimate of communication load and hence battery energy expenditure. As discussed previously, neighbor cameras communicate with each other 1) to figure out the ownership of the "free" voxels; and 2) to exchange information about the voxels' update values. In these two experiments, the total number of iterations is set as 200. The narrow band-width is 6, so the number of voxels in the narrow band is the surface area times the narrow band width (approximately 3×10^4 for the Dinosuar dataset). Each voxel is maintained by 3 cameras on average. So the total number of voxels that all the cameras take care of is about three times that number: 6×10^4. Since 20-23 cameras can cover the surface tightly, the number of "free" voxels is small compared to the size of the union of the PRS. So the message exchanged is dominated by the voxel updating messages. As shown in section 3.2, the total number of update values exchanged is $2(N-1) \approx 1.2 \times 10^5$. Each update value is 4 bytes (stored in single precision format), the total number of communication bytes for each iteration is $4.8 \times 10^5 = 48$KB. With 200 iteration, the total data exchanged is about 13.6 MB. This order of communication cost is affordable in VSNs.

5. Conclusions and Discussion

In this paper we define the problem and present a solution to the 3D reconstruction of an object, indoors or outdoors, from silhouettes in images taken by a network of *many randomly* distributed battery powered cameras hav-

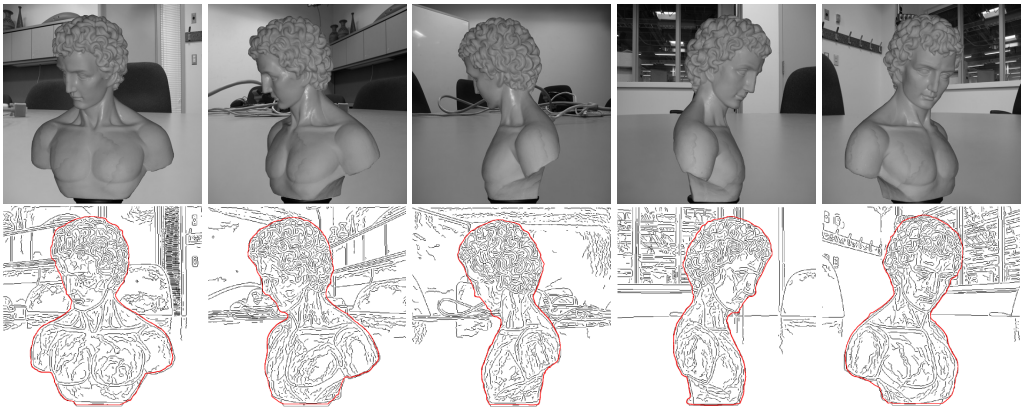


Figure 9. (top) Image sequence with indoor background; (bottom) Projected silhouette contours (in red) estimated by the algorithm, during the 3D reconstruction process, overlapped with the image edges

ing onboard processing and wireless communication. The goal is reconstruction to close to the achievable accuracy while roughly minimizing processing time and battery usage. (More generally, other constraints may be present, e.g., communication bandwidth limitation.) The challenge is to use few image pixels in an image, to communicate as little data as possible, and for each camera to communicate to as few other cameras as possible. Our solution involves maximum a posteriori probability estimation for achieving close to optimal accuracy, introducing and using a dynamically changing vision graph for assigning computation tasks to the various cameras for achieving minimum computation time, and routing camera communications over a minimum spanning tree (MST) for achieving minimum communications battery usage.

The main contribution of this work is the distributed processing of shape-from-contours, including the region-specific vision graph, job division schemes, and the MST message passing protocol. The region-specific vision graph and MST message passing developed in the paper can be applied to other distributed vision tasks generally. The job division scheme is linked to the shape-from-contours approach more tightly, but the principals developed here, including the three constraints (*correctness, complete coverage and load balance*), can be extended to other vision problems. For example, for shape from texture and contours, similar job division schemes can be developed by selecting most valuable image observations. The distributed algorithm proposed in the paper is not only applicable to smart camera network but also applicable to multiprocessor systems such as many-core CPUs and GPUs nowadays. We compute rough estimates of the amount of required computation and the required communication cost. The approach is appropriate for networks of very large numbers of cameras.

References

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38:393–422, 2002.

[2] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems. In *Proc. 19th Symp on Theory of Computing*, pages 230–240, May 1987.

[3] M. Brand, K. Kang, and D. Cooper. Algebraic solution to visual hull. In *CVPR*, 2004.

[4] R. Cipolla and P. Giblin. *Visual Motion of Curves and Surfaces*. Cambridge University Press, 2000.

[5] B. Das and V. Loui. Reconstructing a minimum spanning tree after deletion of any node. *Algorithmica*, 31:530–547, 2001.

[6] O. Faugeras, J. Gomes, and R. Keriven. *Geometric Level Set Methods in Imaging, Vision and Graphics. Osher and Paragios Eds.*, chapter Variational Principles in Computational Stereo. 2003.

[7] O. Faugeras and R. Keriven. Variational principles, surface evolution, PDE's, level set methods and the stereo problem. *IEEE Trans. Image Processing*, 7(3):336–344, 1998.

[8] R. Gallager, P. Humblet, and P. Spira. A distributed algorithm for minimum weight spanning tree. *ACM Trans. on Programming Languages and Systems*, 5(1):66–77, January 1983.

[9] P. Gargallo, E. Prados, and P. Sturm. Minimizing the reprojection error in surface reconstruction from images. In *ICCV*, pages 1–8, 2007.

[10] S. Liu, K. Kang, J.-P. Tarel, and D. Cooper. Free-form object reconstruction from occluding edges and texture edges: A unified and robust operator based on duality. *PAMI*, 30(1):131–146, January 2008.

[11] K. Obraczka, R. Manduchi, and J. Garcia-Luna-Aveces. Managing the information flow in visual sensor networks. In *5th Symp. Wireless Personal Multimedia Communications*, volume 3, pages 1177–1181, 2002.

[12] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, New York, 2002.

[13] B. Rinner and W. Wolf. An introduction to distributed smart cameras. *Proceedings of the IEEE*, 96:1565–1575, 2008.

[14] J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.

[15] A. J. Yezzi and S. Soatto. Stereoscopic segmentation. In *ICCV*, 2001.