

3D Surface Extraction using Incremental Tetrahedra Carving

Takayuki Sugiura, Akihiko Torii and Masatoshi Okutomi
Tokyo Institute of Technology
Tokyo, Japan

{ tsugiura@ok., torii@, mxo@ } ctrl.titech.ac.jp

Abstract

We propose a fully incremental and yet globally optimal surface extraction method for updating the 3D surfaces when new images, camera poses and 3D points are additionally registered into the existing 3D model. We extend the tetrahedra-carving-based surface extraction algorithm to the incremental fashion by efficiently detecting ray-tetrahedra intersections and formatting the graph to solve with the dynamic graph cut. The proposed method can result the surfaces identical to the one running from scratch, i.e. the global optimality of the extracted surface is guaranteed while improving the efficiency with fully incremental procedures. We compare the proposed method with state-of-the-art baseline methods and finally demonstrate the surface extraction of fairly large objects.

1. Introduction

After SIFT [23] and Photo Tourism(Bundler) [29], 3D reconstruction from imagery is not an infeasible task according to the advancement of SfM [2, 7, 30] followed by dense reconstruction [9, 8] and surface reconstruction [14, 16, 18, 33]. On top of these advances, static objects and scenes in the digital photos can be easily transformed to 3D models using publicly available codes [1, 5, 9, 29, 34] or online services [12, 25]. Some systems are even capable for large-scale reconstruction [2, 7, 8] thanks to the efficient image clustering based on similarity computation using image descriptors [15, 28]. These systems achieve accurate and complete 3D models by batch processing for the static input data.

If the input data grows with time, incremental 3D modeling and their effective visualization are valuable in practical applications, *e.g.* visual navigation of vehicle or UAV [11, 27, 36], AR [24], and city change detections [31]. To achieve this, 3D modeling frameworks suitable for incremental update are essential but no trivial method exists.

In principle, most of the components (algorithms) in SfM primarily suit for incremental processes, *e.g.* feature

detection and matching, image pair selection, and camera pose estimation [30]. The global bundle adjustment is the only exceptional but the recent works [13] accomplishes by taking care of updating variables. Typical approaches of dense reconstruction in the incremental fashion are to locally compute dense (depth) maps and to merge to a global map [24, 27]. This way does not guarantee the global consistency of the entire 3D model.

One of the successful approaches providing the accurate, complete, and globally consistent 3D models is: (1) densify 3D point clouds by guided matching expansion or plane sweeping [9, 33]; (2) extract surfaces from 3D Delaunay triangulation of the dense 3D points [18]; (3) refine the surfaces by using geometric and photometric information [14, 33]. This approach works even for very general input, *e.g.* photo collections or sparse image sequences taken by a single camera, but the extension to the incremental fashion is not fully studied. This is because the quality of the resulted models depends on the examination of the consistency of local patches [9] or the computation of surface patch visibility [18]. Therefore, their extension for incremental processes while preserving quality sounds contradictory.

In this paper, we propose a fully incremental surface extraction as an extension of [18]. We suppose the following situations (Figure 1): the camera poses and the point clouds are obtained by a large-scale SfM and they can be fixed as the base model, *i.e.* no further global bundle adjustment is required; the initial 3D surface is generated by [18] or our proposed method; new cameras and point clouds associated with them are additionally and locally registered to the base model, *e.g.* by [21]. We aim at performing efficient incremental updates for the existing large-scale 3D surface model while ensuring the global optimality of the extracted surfaces, *i.e.* our method produces the surfaces identical to the one running from scratch. The contributions of this paper are as follows.

- We propose the first algorithm that can efficiently update the surface model while keeping the optimality with all cameras and rays taken into account.

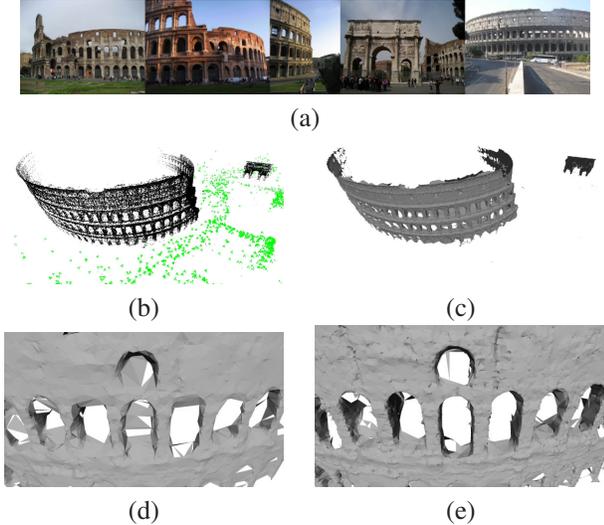


Figure 1. Incremental surface extraction on Rome Colosseum dataset. (a) Examples of input images. (b) 3D point clouds and camera poses obtained via SfM. (c) 3D surface extracted by the proposed method. (d) and (e) show the local part of the entire 3D models to improve the visibility. (d) The surface of the base model ($N = 500$). (e) The surface updated by the proposed method.

- We alternate two algorithms for detecting ray-tetrahedra intersections depending on the types of rays and tetrahedra in the updated graph.
- We adopt the dynamic graph cut [17] for the incremental surface extraction by introducing isolated nodes in the graph for handling the appearance and disappearance of tetrahedra.

The rest of the paper is organized as follows. Section 1.1 describes related works on multiple view stereo and surface reconstruction algorithms. In Section 2 we review the surface extraction with tetrahedra carving [33] for clarifying the differences and contributions of the proposed incremental algorithms. In Section 3 we describe our method for incremental surface extraction. Finally, we demonstrate and compare w.r.t. the state-of-the-art baselines in Section 4.

1.1. Related work

Multi view stereo is a well-studied problem. One of the popular methods is PMVS [9] which generates dense 3D point clouds by computing local patches in 3D space and filling the missing area using the neighbors of confident ones. The 3D surface mesh can be obtained by applying Poisson surface reconstruction [16] or its variant. PMVS can be applied for a large scale by segmenting the scenes (camera poses and sparse points) into pieces as neighboring ones have sufficient overlaps for giving seamless 3D models after integrating them [8]. Other recent works on multiple view stereo for large scale problem use some priors such as geometric primitives (planes or spheres) [19] and two or-

thogonal curves [35]. The use of priors improves the stability and understandability while enabling to reduce the data size according to the simplified description of the scene

As briefly summarized in Section 1, Vu *et al.* [33] proposed the surface reconstruction method based on 3D Delaunay triangulation and carving the 3D space. The method is capable for reconstructing the large-scale cluttered scenes without requiring some knowledge as approximate geometry and topology of the scene. Jancosek *et al.* [14] further improved the surface extraction so as to recover weakly supported surfaces by using the free-space cones based on visual-hull [20]. Note that since these methods focus on generating accurate and complete 3D models, the computational costs and extension to incremental procedures are not the critical issue.

On the other hand, there are several methods aiming at 3D reconstruction in incremental fashion. DTAM [24] is known as the real-time camera tracking and 3D reconstruction where the pixel-wise depth maps of reference cameras are generated. The work by Hoppe *et al.* [11] is closely related, which performs the large-scale incremental reconstruction for providing feedback of surface quality to manipulators. In [11], the surface is reconstructed for every keyframe by repeating the surface extraction [18] running from scratch. Therefore, the computation time increases as the reconstructed scene expands. Recently, Hoppe *et al.* proposed an incremental surface extraction method [10] which achieves online 3D reconstruction by integrating with the incremental SfM. For processing the incremental surface extraction in a limited computational time, they designed a new cost function which computes the local visibility of the surface patches whereas the original method [18] ensures the global visibility over the whole scene. Also, the use of dynamic graph cut is presented¹. Our work is also related to incremental free space carving with reasonable heuristics [22, 26, 36] but we are more interested in reconstructing large outdoor scenes from sparser input images. Also, note that the advantage of our method is to ensure the global optimality as the original tetrahedra carving method [33] while performing the surface extraction as efficient as the state of the arts.

2. Surface extraction with tetrahedra carving

We review the surface extraction by tetrahedra carving [33] to reveal the technical contributions of our proposed method in Section 3. Given the point cloud $\mathcal{P} = \{p\}$, cameras $\mathcal{C} = \{c\}$, and $\mathcal{R} = \{r\}$ where r is the line segment connecting a camera center c and a 3D point p (Figure 2), the surface extraction method [33] first segments the 3D space by 3D Delaunay triangulation of \mathcal{P} . Then, every tetrahe-

¹In Section 3.3, we also introduce how to adopt the dynamic graph cut for this problem. Note that the original submission of this paper was for ICCV 2013 and this was a parallel work with [10].

dron is labeled whether it is inside or outside of the objects in the scene. Finally, we obtain the surface as the border (set of facets) of tetrahedra separated to inside or outside. We next describe more details on each component in the surface extraction algorithm.

3D Delaunay triangulation. A 3D Delaunay triangulation (tetrahedralization) of a point cloud \mathcal{P} generates tetrahedra where all vertices correspond to the points in \mathcal{P} and the circumsphere of each tetrahedron does not include any other point of \mathcal{P} . The unique triangulation is ensured under the assumption that there exists only four points on the same circumsphere. Even if more than five points lie on the same circumsphere, the Delaunay triangulation generates the unique tetrahedra by using a symbolic perturbation scheme [6]. Regarding to the problem on merging duplicated 3D points, we assume that it is already trimmed in the SfM stage.

Formulation as a binary labeling problem. On top of the Delaunay triangulation, we build a finite directed graph $\mathcal{G} = (\mathcal{T}, \mathcal{E})$ where nodes $\mathcal{T} = \{t\}$ correspond to tetrahedra and edges $\mathcal{E} = \{E = (t_p, t_q)\}$ correspond to facets $\mathcal{F} = \{f\}$ shared by adjacent tetrahedra. Note that every facet is shared by a pair of tetrahedra except for the ones on the convex hull boundary of \mathcal{P} . The surface is extracted by classifying whether each tetrahedron t belongs inside or outside of objects. This is formulated as a binary labeling problem which is solved by minimizing the following cost function:

$$E(\mathcal{L}) = \sum_{t \in \mathcal{T}} \{U_t(l_t) + \sum_{v \in \mathcal{V}_t} B_{t,v}(l_t, l_v)\}, \quad (1)$$

where $l_t \in \mathcal{L}$ is the label of tetrahedron $t \in \mathcal{T}$ which indicates whether it exists inside ($l_t = \text{IN}$) or outside ($l_t = \text{OUT}$) of objects. The unary term U_t is the penalty for inconsistent evidences with the label l_t of the tetrahedron t

$$U_t(l_t) = \begin{cases} N_{\text{in}}(t) & \text{if } l_t \neq \text{IN} \\ N_{\text{out}}(t) & \text{if } l_t \neq \text{OUT} \end{cases} \quad (2)$$

$N_{\text{in}}(t)$ is the number of intersections of the extended rays of r and the tetrahedron t . More specifically, the ray r composed from p and c first reaches p and the extended ray of r skewers the tetrahedron t (Figure 2). $N_{\text{out}}(t)$ is the number of intersections of the rays r and the tetrahedron t which includes the cameras inside.

The binary term $B_{t,u}$ penalizes if two neighboring tetrahedra are labeled differently,

$$B_{t,v}(l_t, l_v) = \begin{cases} N_{\text{intrsect}}(t, v) & \text{if } l_t = \text{OUT} \cap l_v = \text{IN} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

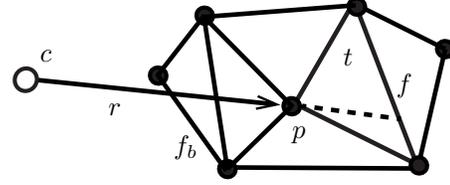


Figure 2. Intersection detection of ray and facet. The solid line with an arrow is the ray r (line segment) composed from the camera c and the point p . The dashed line is the extended ray of r . f is the facet of the tetrahedron t . f_b is the facet on the convex full boundary of \mathcal{P} .

$N_{\text{intrsect}}(t, v)$ is the number of intersections of $f(t, v)$ and the rays $r_{t \rightarrow v}$ where $f(t, v)$ is the facet shared by the tetrahedron t and its adjacent tetrahedron $v \in \mathcal{V}_t$ and $r_{t \rightarrow v}$ is the directed ray coming from t to v . See also [33] for more details.

Ray-facet intersection detection with ray-wise tracing.

The fundamental task for composing the two terms U_t and $B_{t,v}$ is to detect the intersections of the rays and the facets of tetrahedra, *i.e.* we detect tetrahedra skewered by a ray (line segment) composed from a point p and a camera center c (Figure 2). This is efficiently performed by tracing from the tetrahedron having p as a vertex to the tetrahedron including c inside. Note that if the camera is sitting outside the convex hull of \mathcal{P} , the tracing stops at f_b on the convex hull boundary. Also, the ray-wise tracing can naturally detect whether a ray passes a facet from front to back or vice versa. This is important for constructing N_{intrsect} in the binary term of the cost function. We refer this procedure to “ray-wise tracing” in order to discriminate the other method described in Section 3.2.

Optimization by graph cut. In the optimization step, a standard Max-flow/Min-cut algorithm [3] is used for finding the global optimum by s - t graph cut. For surface extraction, the graph consists of nodes corresponding to tetrahedra \mathcal{T} . The edges connecting adjacent tetrahedra via facets have the directed weights determined by $B_{t,v}$. Also, the edges connecting every tetrahedron to the source and sink have the directed weights determined by U_t . In detail, if facets of t are on the convex hull boundary of \mathcal{P} , the directed weights on edges between the source to the nodes are sum of $N_{\text{out}}(t)$ and $N_{\text{intrsect}}(t, v)$. Note that if all the weights associated to t are zero, the label of t is undetermined. In this case, we label t as inside because we prefer not carving t when no rays intersect with t at all. The surface is finally extracted according to the optimal labeling of \mathcal{T} , *i.e.* the global minimum of E .

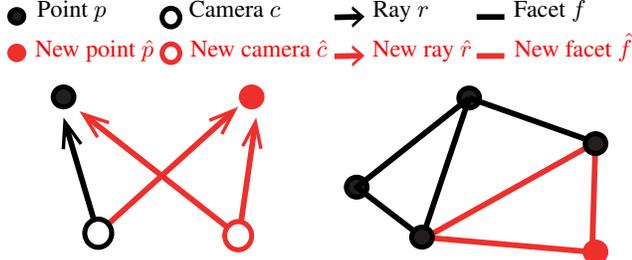


Figure 3. Points, cameras, rays, and facets composing the graph weights of \mathcal{G}' . See text for details.

3. Incremental tetrahedra carving

In this section, we extend the surface extraction algorithm to the incremental fashion while the optimality of the extracted surface is guaranteed, *i.e.* the proposed incremental algorithm should provide the identical surface computed by the method [33] running from scratch. We describe three important steps to be extended as follows. Hereafter, $\hat{c} \in \hat{\mathcal{C}}$ and $\hat{p} \in \hat{\mathcal{P}}$ denote the new camera and the 3D points newly reconstructed by incremental SfM by adding a new image I' (Figure 3).

3.1. Incremental Delaunay triangulation

The Delaunay triangulation step is rather trivial comparing to the others. When a new point $\hat{p} \in \hat{\mathcal{P}}$ is added to \mathcal{P} , we perform three processes to augment the tetrahedra segmentation.

- i ADDITION: If \hat{p} exists outside of the convex hull of \mathcal{P} , Delaunay triangulation is accomplished to the point \hat{p} and the points on the convex hull of \mathcal{P} . And then the convex hull is expanded to $\mathcal{P} \cup \hat{p}$.
- ii DIVISION: If \hat{p} exists inside the convex hull of \mathcal{P} , the tetrahedron including \hat{p} is divided into four tetrahedra.
- iii FLIP: If the ADDITION or DIVISION process generates some invalid tetrahedra which violate the second Delaunay rule. The re-divided tetrahedra are composed from the five (four + one) vertices by flipping the facets.

We denote the tetrahedra newly appeared by incremental Delaunay triangulation as $\hat{\mathcal{T}} = \mathcal{T}' \setminus \mathcal{T}$ where \mathcal{T}' is the Delaunay triangulation of $\mathcal{P} \cup \hat{\mathcal{P}}$. Regardless the order of adding new points, the Delaunay triangulation always generates the unique set of tetrahedra.

3.2. Efficient detection of new intersections

Due to addition of cameras and 3D points by incremental SfM, the graph \mathcal{G} have to be updated to \mathcal{G}' . Fully re-computing the weights of the graph \mathcal{G}' becomes a bottle neck in the tetrahedra carving as the number of points and cameras increase. The necessary and sufficient condition

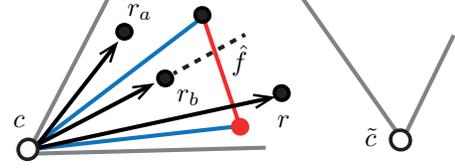


Figure 4. Facet-wise detection. Using the visibility of \hat{f} on each camera, we can immediately detect the rays emanated from the camera \tilde{c} as not intersecting. Then, using the visual cone composed from c and \hat{f} , r_a is quickly detected as not intersecting. Finally, we examine the rays r and r_b if they actually pass through the \hat{f} .

for ensuring the optimality is to detect the intersections of rays and facets related with the changes in \mathcal{G}' and to update the weights associated with them only.

To achieve this, we consider two types of rays, \mathcal{R} and $\hat{\mathcal{R}}$. $r \in \mathcal{R}$ is the ray composed from a camera $c \in \mathcal{C}$ and a 3D point $p \in \mathcal{P}$. $\hat{r} \in \hat{\mathcal{R}}$ is the newly appeared ray either composed from a new camera $\hat{c} \in \hat{\mathcal{C}}$ and a point in $\mathcal{P} \cup \hat{\mathcal{P}}$, or a camera $c \in \mathcal{C}$ and a new point $\hat{p} \in \hat{\mathcal{P}}$ (Figure 3). Similarly, there are two types of facets, \mathcal{F} and $\hat{\mathcal{F}}$. \mathcal{F} are the facets still existing as the Delaunay triangulation of \mathcal{P} after updating the Delaunay triangulation with new points $\hat{\mathcal{P}}$. $\hat{\mathcal{F}}$ are the facets of new tetrahedra $\hat{\mathcal{T}}$ (Figure 3). Obviously, we should avoid re-computing the intersections of \mathcal{R} and \mathcal{F} which have no change due to the addition of new camera and points.

For all the new rays $\hat{\mathcal{R}}$, we use the ray-wise tracing because it is necessary to detect the intersections with all the facets associated with them in $\mathcal{F} \cup \hat{\mathcal{F}}$. To ensure the completeness of updating the graph weights, it is also required to examine the intersections of the rays \mathcal{R} and the new facets $\hat{\mathcal{F}}$. This case is a bit tricky because the ray-wise tracing proceeds regardless the types of facets. Therefore, if we also apply the ray-wise tracing for the rays \mathcal{R} to ensure the same result, the computational cost with this ends up with the same as running from scratch.

To overcome this problem, we introduce facet-wise detection. This method detects ray-facet intersections via the projection of a facet to all cameras and verifying with the cameras actually viewing it. If we apply the facet-wise detection for every new facet \hat{f} , all the intersections of $\hat{\mathcal{F}}$ and \mathcal{R} can be efficiently detected because it disregards the only rays not contributing the ray-facet intersections at all.

In detail, the facet-wise detection is accomplished by the four steps (Figure 4). We first select the cameras in \mathcal{C} which have the rays potentially intersecting with a new facet \hat{f} . This is efficiently implemented using the view field of cameras. Next, in the rays associated to the selected camera c , we select the candidate rays which are inside the visual cone composed from the camera c and the facet \hat{f} . Then, we verify if the candidate rays actually intersect with the facet \hat{f} . Note that there are some rays not reaching \hat{f} since we con-

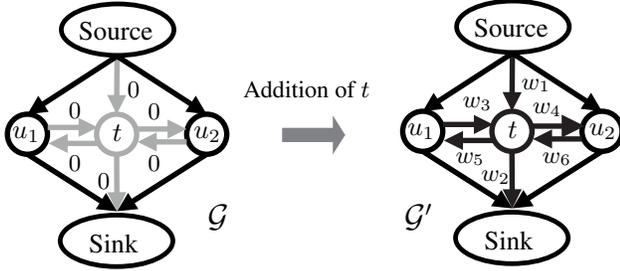


Figure 5. The isolation of a node t for adapting to the graph changes from \mathcal{G} to $\hat{\mathcal{G}}$.

consider the rays as the line segments as described in Section 2. Finally, for the facet \hat{f} , we distinguish the direction of the intersection, *i.e.* whether the ray passes from front to back or vice versa for constructing the binary term.

In summary, applying the ray-wise tracing to new rays $\hat{\mathcal{R}}$ and the facet-wise detection to new facets $\hat{\mathcal{F}}$ can accomplish all the required computation of the intersections comprehensively, without carrying out the duplicative intersections of \mathcal{R} and \mathcal{F} which are unnecessary to be recomputed.

3.3. Dynamic Binary Labeling

After the incremental Delaunay triangulation and computation of weights of the updated graph, the s - t graph cut should be performed as efficient as possible. Since the changes of our graph can be considered as a dynamic graph, there exists an efficient and beautiful algorithm to find the global optimum. Kohli, *et al.* [17] proposed the dynamic graph cut algorithm which detects the changes in the graph and compute the flow associated to them for achieving the global minimum cut without re-computing all the flows. This algorithm is particularly beneficial when the changes occur only locally in the entire graph.

In our problem, the graph \mathcal{G} is updated to \mathcal{G}' by adding a new image I' with producing two types of changes. If new tetrahedra $\hat{\mathcal{T}}$ are generated according to the addition of new points $\hat{\mathcal{P}}$, the new nodes $\hat{\mathcal{T}}$ appear in \mathcal{G}' . If the tetrahedra in \mathcal{T} are deleted while the FLIP or DIVISION process, the corresponding nodes in \mathcal{G} disappear in \mathcal{G}' . Note that tetrahedra can be deleted due to the changes of Delaunay structure even though we assume the disappearance of 3D points does not occur via SfM.

In order to adapt the changes from \mathcal{G} to \mathcal{G}' and to apply the dynamic graph cut algorithm [17] for finding the global minimum cut of the updated graph \mathcal{G}' , we use isolated nodes:

- In the graph \mathcal{G} (before the changes), we regard the new node t as an isolated node by setting zeros for every edge. When updating the graph to \mathcal{G}' , we assign the number of ray-facet intersections, which is computed in Section 3.2 to the edge weights of the isolated node in \mathcal{G} (Figure 5).

- Conversely, for deleting nodes in \mathcal{G}' , we set zero to the edge weights of node t in order to isolate it.

This algorithm is particularly efficient when the graph keeps growing but the changes in the graph is only limited, which is the typical situation in our incremental surface extraction.

4. Experiments

In this section we describe the experimental validation of our approach. First, we give the implementation details. Then we examine the performance of the proposed method with the baseline methods which repeats the surface extraction [33] from scratch as new inputs come. Finally, we demonstrate the surface extraction on the large scale datasets.

4.1. Implementation details

We implemented the algorithms in C++ on Linux 64bit OS on Intel Core i7, CPU 3.20GHz, and RAM 64GB Desktop PC. We used several libraries: OpenCV [4], CGAL [32], and Dynamic graph cut library [17]. In our implementation, the detections of ray-tetrahedra intersection for both ray-wise tracing and facet-wise detection can be multi-threaded per camera.

The two additional processes are performed in order to refine the extracted surface. One is that the large triangle patches are deleted based on the percentile of the longest edge of each patch. The other is that for smoothing the surface, every point on the surface is updated iteratively as follows: $p_{i+1} = p_i + \lambda \sum_{q_i \in V_p} w_{p_i q_i} (q_i - p_i)$, where V_p is the set of points which are neighboring p on the surface and λ is a parameter of smoothing. $w_{p,q}$ is the weight based on distance between two points: $w_{p,q} = \phi(p,q) / \sum_{k \in V_p} \phi(p,k)$, where $\phi(p,q)$ is the inverse distance of two points. These two simple refinements are not yet optimized for incremental surface extraction, we exclude from the evaluation but use for visualizing the surface model.

4.2. Evaluation of the proposed method

To purely evaluate the performance of the proposed surface extraction method, we use the pre-computed camera motions \mathcal{C} and point clouds \mathcal{P} for all the images in the datasets. Then, we sequentially input a camera pose and 3D points associated with it to the surface extraction algorithm.

We use **Brick Warehouse** dataset which consists of 978 images captured while walking around the building. 2296×1528 pixels images are taken by a single camera with a fixed local length. All the camera poses and 3D point clouds are computed by our implementation of incremental SfM. In the Brick Warehouse dataset (Figure 6(a)), the proposed method shows almost constant computational time compared to the baseline method which takes more time as the number of images increases. The computational

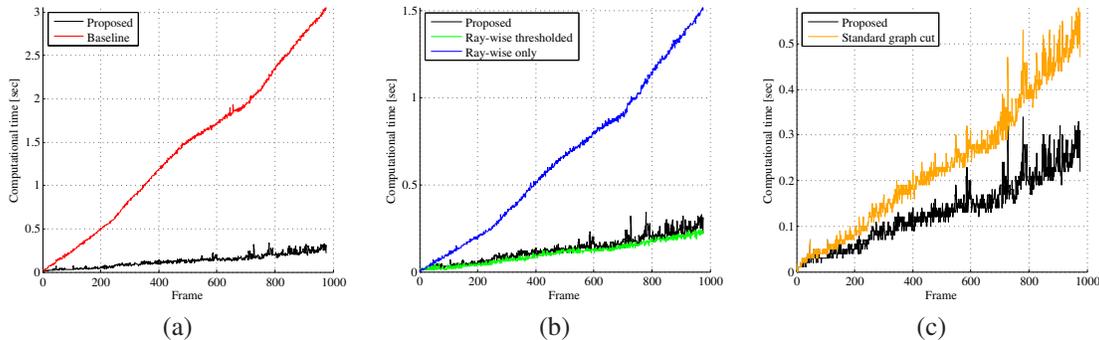


Figure 6. Comparison on the Brick Warehouse dataset. The graph shows the computational time of extracting surface at each frame. (a) The proposed surface extraction (black) and the baseline method (red). (b) The proposed method (black), the ray-wise tracing only (blue) and the ray-wise tracing applied to the subset of rays [36] (green). (c) The proposed method (black) and the method with a standard graph cut (orange).

time accumulated for the 977 updates is 127.67 (Proposed) and 1385.23 (Baseline) [sec]. We evaluated from the number of patches that the final surface of the proposed method is completely identical to the baseline method.

Impact of facet-wise detection. To evaluate the importance of the facet-wise detection, we compare the three different methods of the ray-facet intersection detection while keeping the other steps consistent. Figures 6(b) shows the computational time of each method. The method with ray-wise tracing only (blue) increases the computational time as the reconstructing scene grows. The ray-wise tracing applied to the subset of rays seeing from several latest cameras determined by a threshold [36] (green) is partly faster than the proposed (black), however, the resulted surface with thresholding is different to the baseline method with the errors 47%. Thanks to the facet-wise detection, the proposed method takes almost constant time compared to the ray-wise only method.

Impact of applying dynamic graph cut. Figures 6(c) shows the effect of applying dynamic graph cut [17]. Black and orange curves show the computational time at each frame by the proposed method with applying dynamic graph cut (Proposed) and with applying a standard graph cut [3] by computing the flows from scratch (Standard graph cut). Comparing to the other processes, the improvement by applying dynamic graph cut is rather small. Note that, however, the mean of fraction of improvement at every frame is 30 % which is significant towards real time applications.

Limitation for integrating with incremental SfM. The proposed surface extraction can be potentially integrated with the incremental SfM. However, there is a difficulty due to the distortion and perturbation yielded by global bundle adjustment while adding more and more images. In practice, the baseline method can be applied only after the global

Table 1. Number of points and patches on Brick Warehouse dataset.

	Proposed	[14] (SfM)	[14]	[9]
#points	124,216	196,994	1,797,387	5,141,170
#patches	276,367	227,355	2,503,592	-

bundle adjustment triggered while the proposed incremental method is used for all the other timings.

Also, we assume that the SfM does not remove or merge 3D points and cameras once they are generated. This assumption is required to ensure that the already detected intersections for the existing cameras \mathcal{C} , rays \mathcal{R} , and facets \mathcal{F} have no change. In practice, this assumption can be hold by proceeding SfM conservatively.

Qualitative evaluation w.r.t. the state of the arts. We qualitatively compare our 3D surface models to the state-of-the-art methods of 3D reconstruction. We use publicly available softwares, PMVS [9] and CPMVS without and with plane sweeping [14]². Figure 7 shows the 3D models of the four methods and Table 1 shows the numbers of the resulted points and patches. We show the colored point clouds as the result of PMVS since the simple point clouds can be sufficiently informative for a large cluttered scene. The quality of 3D model of the proposed method is not inferior to that of the other state-of-the-art methods.

4.3. Incremental surface extraction on large-scale datasets

We consider the situation that the base 3D model is given by a large-scale reconstruction, and we incrementally update the 3D surface model by adding several images newly acquired. We conducted the experiments on two publicly available datasets provided by [21]. **Rome Colosseum**

²The version of CPMVS without plane sweeping is provided by the courtesy of the authors of [14]



Figure 7. Qualitative evaluation with the state-of-the-art baselines on Brick Warehouse dataset. (a) The proposed method. (b) PMVS [9] (colored dense point clouds). (c) CMPMVS without plane sweeping, *i.e.* the same input as the proposed. (d) CMPMVS [14].

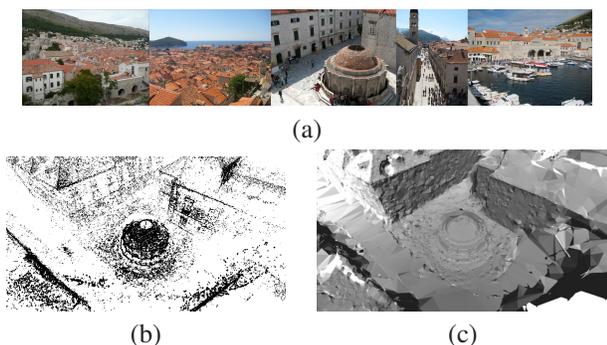


Figure 8. Results of the surface extraction on Dubrovnik dataset. (a) Examples of input images. (b) 3D point cloud obtained via SfM (only a local part is shown for visibility). (c) The surface extracted by the proposed method.

dataset consists of 2,043 images collected on the Flickr, all the camera poses and 3D point clouds computed by SfM [29]. **Dubrovnik** dataset in [21] is a larger dataset consisted of 6,844 images captured over the whole town.

We define the base 3D model by selecting N images, their camera poses and associated 3D points, and by running the surface extraction to obtain the initial 3D surface. Then, we randomly select k images from the rest, add their camera poses and 3D points, and use them as the input for the incremental surface extraction. We update the 3D surface models for t times until $N + t \times k$ reaches the total number of images of the dataset.

Table 2 shows the results for evaluating the computational efficiency of the proposed method. The “baseline time” is the computation time of extracting the surface from all the images in the dataset. Note that this computation time is almost equal to that of running the baseline method from scratch at every update. Since our method is capable for adding k images at once, we tested different k as $k = \{1, 5, 10\}$ shown in the “#images added at once” in Table 2. The “#points added at once” is the average number of points added at each update. The “computation time” shows the average of $t = 10$ updates using the proposed method.

Table 2. Experimental results for large scale datasets

	Dubrovnik			Colosseum		
#images	6,844			2,043		
#points	2,020,921			600,274		
baseline time [sec]	91.9			18.85		
#images added at once	1	5	10	1	5	10
#points added at once	142	1,010	3,060	246	2,057	3,826
computation time [sec]	3.8	10.7	26.9	1.95	8.68	15.35

Overall, the proposed method is more efficient than the baseline while producing the identical results. Especially, when adding one image for each update in the Dubrovnik dataset, the baseline method takes 91.9 seconds for every update whereas the proposed method takes 3.8 seconds which is 24.2 times faster. On the other hand, on the Rome Colosseum dataset, when 10 images are added at once, the improvement by the proposed method is not very significant. One of the reasons is that the current implementation of the facet-wise detection uses the view field of cameras and the bounding box of newly appeared facets for efficient computation. This is not very effective if the new images capture all the objects in the scene since the newly appeared facets appear all the cameras in the scene.

Figure 1 (e) shows the final surface extracted by the proposed method starting from the base model (d) ($N = 500$) on the Rome Colosseum dataset. Since the points increase by adding more and more images, the mesh of the updated surface is finer than the initial base model. Also, Figure 8 (e) shows the 3D surface extracted by the proposed method.

5. Conclusions

We have proposed a fully incremental surface extraction based on tetrahedra carving while ensuring the global optimality. We have introduced the two key ideas: the intersection detection adaptive to the types of rays and facets; the application of the dynamic graph cut for the growing data composed by cameras and 3D points. Although the

isolation of nodes is briefly mentioned in [17], its practical application is not described. We believe the real application and implementation of it have sufficient contribution. Further, the detailed experimental evaluations showed the improvement for each step of our method and the surface extraction was demonstrated on fairly large datasets. We left the system integration with SfM which should be accomplished near future.

Acknowledgement

This work was partially supported by the Grants-in-Aid for Scientific Research (no. 25240025, 24700161) from the Japan Society for the Promotion of Science.

References

- [1] 2d3 Ltd. Boujou: Automated camera tracking, 2003. <http://www.2d3.com>.
- [2] S. Agarwal, N. Snavely, I. Simon, S. Seitz, and R. Szeliski. Building Rome in a day. In *Proc. ICCV*, pages 72–79, 2009.
- [3] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *PAMI*, 26(9):1124–1137, 2004.
- [4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [5] P. Cignoni, M. Corsini, and G. Ranzuglia. Meshlab: an open-source 3d mesh processing system. *ERCIM News*, 2008(73), 2008.
- [6] O. Devillers and M. Teillaud. Perturbations and vertex removal in a 3d delaunay triangulation. In *ACM-SIAM, SODA '03*, pages 313–319, 2003.
- [7] J.-M. Frahm, P. Fite-Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y.-H. Jen, E. Dunn, B. Clipp, S. Lazebnik, and M. Pollefeys. Building rome on a cloudless day. In K. Daniilidis, P. Maragos, and N. Paragios, editors, *ECCV*, pages 368–381, 2010.
- [8] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. Towards internet-scale multi-view stereo. In *CVPR*, pages 1434–1441, 2010.
- [9] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *PAMI*, 32:1362–1376, 2010.
- [10] C. Hoppe, M. Klopschitz, M. Donoser, and H. Bischof. Incremental surface extraction from sparse structure-from-motion point clouds. In *BMVC13*, 2013.
- [11] C. Hoppe, M. Klopschitz, M. Rumpler, A. Wendel, S. Kluckner, H. Bischof, and G. Reitmayr. Online feedback for structure-from-motion image acquisition. In *BMVC*, pages 70.1–70.12, 2012.
- [12] A. Inc. Autodesk 123d, 2013.
- [13] V. Indelman, R. Roberts, C. Beall, and F. Dellaert. Incremental light bundle adjustment. In *BMVC*, 2012.
- [14] M. Jancosek and T. Pajdla. Multi-view reconstruction preserving weakly-supported surfaces. In *CVPR*, pages 3121–3128, 2011.
- [15] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, and C. Schmid. Aggregating local image descriptors into compact codes. *PAMI*, 2012.
- [16] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Eurographics, SGP '06*, pages 61–70, 2006.
- [17] P. Kohli and P. H. S. Torr. Efficiently solving dynamic markov random fields using graph cuts. In *ICCV*, pages 922–929, 2005.
- [18] P. Labatut, J.-P. Pons, and R. Keriven. Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In *ICCV*, pages 1–8, 2007.
- [19] F. Lafarge, R. Keriven, M. Brédif, and H.-H. Vu. Hybrid multi-view reconstruction by jump-diffusion. In *CVPR*, pages 350–357, 2010.
- [20] A. Laurentini. The visual hull concept for silhouette-based image understanding. *PAMI*, 16(2):150–162, 1994.
- [21] Y. Li, N. Snavely, and D. P. Huttenlocher. Location recognition using prioritized feature matching. In *ECCV*, pages 791–804, 2010.
- [22] D. Lovi, N. Birkbeck, D. Cobzas, and M. Jagersand. Incremental free-space carving for real-time 3d reconstruction. In *3DPVT10*, 2010.
- [23] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.
- [24] R. A. Newcombe, S. Lovegrove, and A. J. Davison. DTAM: Dense tracking and mapping in real-time. In *ICCV*, pages 2320–2327, 2011.
- [25] T. Pajdla, M. Havlena, A. Torii, M. J. Z. Kukulova, and J. Heller. CMP SfM Web Service, 2010.
- [26] Q. Pan, G. Reitmayr, and T. Drummond. ProFORMA: Probabilistic feature-based on-line rapid model acquisition. In *BMVC09*, 2009.
- [27] M. Pollefeys, D. Nistér, J.-M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S. J. Kim, P. Merrell, C. Salmi, S. N. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewénius, R. Yang, G. Welch, and H. Towles. Detailed real-time urban 3d reconstruction from video. *IJCV*, 78(2-3):143–167, 2008.
- [28] J. Sivic and A. Zisserman. Video Google: Efficient visual search of videos. In *CLOR*, pages 127–144, 2006.
- [29] N. Snavely, S. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. In *SIGGRAPH*, 2006.
- [30] N. Snavely, S. Seitz, and R. Szeliski. Modeling the world from internet photo collections. *IJCV*, 80(2):189–210, 2008.
- [31] A. Taneja, L. Ballan, M. Pollefeys, and M. Pollefeys. Image based detection of geometric changes in urban environments. In *ICCV*, pages 2336–2343, 2011.
- [32] CGAL. Computational Geometry Algorithms Library. <http://www.cgal.org>.
- [33] H.-H. Vu, P. Labatut, J.-P. Pons, and R. Keriven. High accuracy and visibility-consistent dense multiview stereo. *PAMI*, 34(5):889–901, 2012.
- [34] C. Wu. VisualSfM: A visual structure from motion system. <http://homes.cs.washington.edu/~ccwu/vsfm/>, 2011.
- [35] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz. Schematic surface reconstruction. In *CVPR*, pages 1498–1505, 2012.
- [36] S. Yu and M. Lhuillier. Incremental reconstruction of manifold surface from sparse visual mapping. In *3DIMPVT*, pages 293–300, 2012.