Incremental Free-Space Carving for Real-Time 3D Reconstruction

David Lovi, Neil Birkbeck, Dana Cobzaş, Martin Jägersand
University of Alberta
Edmonton AB., Canada, T6G 2E1
{dlovi,birkbeck,dana,jag}@cs.ualberta.ca

Abstract

Almost all current multi-view methods are slow, and thus suited to offline reconstruction. This paper presents a set of heuristic space-carving algorithms with a focus on speed over detail. The algorithms discretize space via the 3D Delaunay triangulation, and they carve away the volumes that violate free-space or visibility constraints. Whereas similar methods exist, our algorithms are fast and fully incremental. They encompass a dynamic event-driven approach to reconstruction that is suitable for integration with online SLAM or Structure-from-Motion.

We integrate our algorithms with PTAM [12], and we realize a complete system that reconstructs 3D geometry from video in real-time. Experiments on typical real-world inputs demonstrate online performance with modest hardware. We provide run-time complexity analysis and show that the per-event processing time is independent of the number of images previously processed: a requirement for real-time operation on lengthy image sequences.

1. Introduction

Reconstructing 3D geometry from multiple images is well studied. The accuracy of recent algorithms based on offline batch-processing of images is impressive [20]. However, comparably little work is published on incremental real-time 3D reconstruction from video. Yet this is useful in many applications in robotics, where continuous input video is processed, but on-board CPU power is limited.

Typically, real-time reconstruction proceeds in two stages. First, cameras and 3D points (or patches) are incrementally computed for each new video frame. Second, a triangulated 3D model is computed from the geometric primitives. For the first step, powerful SLAM [4, 6] and SFM-based systems (*e.g*. PTAM [12]) have been published. We focus on the second step, 3D real-time model construction, and we use either mono-SLAM or PTAM as inputs.

Incremental model reconstruction can be done by merging partial geometry (*e.g*. depth maps from video frames). Merrell *et al*. and Pollefeys *et al*. published a real-time method based on fusing quick and noisy depth maps together into a free-space consistent mesh [16, 18]. The results are impressive, and the quality is competitive with offline methods [20]. However, extensive hardware was used including an inertial navigation system (for camera calibration) and a powerful CPU and GPU. In contrast, we use only a moderate mobile (laptop) CPU.

Hilton describes a method that incrementally reconstructs a 3D model from sparse features with strong run-time guarantees [10]. His algorithm performs a 2D Delaunay triangulation in each view, and it back-projects and merges the triangulations into a free-space consistent mesh. Hilton proves that its run time is practically constant per frame. However, his method only considers keyframe addition. Our algorithms handle a superset of events (including keyframe addition, outlier deletion, geometric refinement, *etc*.). Thus, our event-driven approach is more dynamic and can integrate with more SLAM and SLAM-like systems.

An alternative to merging partial geometries is volumetric carving. Related 3D volumetric methods are that of Faugeras *et al*. [8] and Gargallo [9]. They compute a model from a set of sparse or quasi-dense features via the Delaunay triangulation and free-space carving. Our proposed method builds on these works. Recently, Labatbut *et al*. developed methods that combine, via graph cuts, the 3D Delaunay triangulation and free space with photo-consistency and regularization [13, 23]. They achieve impressive results. Unfortunately however, all of these related methods are for offline batch use. Our algorithms achieve real-time performance by exploiting the incremental structure of the Delaunay algorithm to complement it with fully incremental carving.

Independently and very recently, Pan *et al*. developed ProFORMA, a system for online reconstruction [17], with similar goals as ours. ProFORMA constructs a 3D Delaunay triangulation, and it carves free-space via a probabilistic voting scheme which obtains a smoother mesh. Our system, however, has at least two key advantages. First, ProFORMA is only capable of reconstructing isolated objects;

we support complex scenes. Second and most notably, ProFORMA's reconstruction is not incremental; they start over at every keyframe. We remark in Section 3 that, unlike our method, ProFORMA's run-time complexity suffers from free-space constraint accumulation and scales poorly in the number of keyframes. Thus it is not suited for online operation on long image-sequences.

In this work, we present three main contributions:

1. Efficient incremental algorithms for online 3D reconstruction via event handling for adding views, adding or deleting visibility information, and adjusting the model in response to adjusted geometry.
2. Complexity analysis characterizing the method's run-time properties.
3. A complete real-time system showcasing our algorithms: the result of integrating them with PTAM, a SLAM-like system [12].

Our method discretizes space via the 3D Delaunay triangulation of sparsely reconstructed point features, and it exploits free-space constraints: If a camera with optic center O observes a 3D feature P, then the tetrahedra on the line between O and P are carved away. This yields an approximate model, but it is adequate for many applications where view-dependent texturing is used to provide detail. Coarse geometry proxies are in fact often used for image-based modeling and rendering [7, 14, 15, 19]. A target use for our models is photo-realistic predictive display for robotics [3, 19]

Our algorithms are incremental in the following sense. Having computed a Delaunay-discretization and a carving, we handle an incremental change to the input data not by starting over, but by reconciling the previous result to agree with the change. Such a change comes from the underlying process that produces the inputs, *e.g.* from online SLAM or Structure-from-Motion. It can be thought of as one of a set of events, be it keyframe addition or outlier deletion.

The incremental nature alone is not enough to guarantee suitability for online use. As the number of images increases, the number of free-space constraints increases; a naive algorithm will swiftly get bogged down. We prove that real-time performance can be attained by selectively forgetting redundant free-space constraints (Section 3), and we integrate our algorithms with PTAM [12] to produce a complete real-time modeling system (Section 4).

2. Algorithms

Given as input a set of reconstructed 3D point features $\{P\}$, camera estimates $\{O\}$, and visibility lists describing which points P are visible in each view O, our method connects and interpolates the point set to produce a triangulated mesh that approximates the scene. We assume the inputs are readily available from online SLAM [4, 6] or Structure-from-Motion [12]. Therefore, the estimates of $\{P\}$, $\{O\}$, and the visibility lists are continuously changing.

This implies two requirements for any real-time reconstruction method that utilizes this information. First, the method itself must produce intermediate outputs. Specifically, our approach should be fully incremental and generate new meshes in real-time. Second, the method must support fine-grained event handling. Different types of updates to the input data trigger different algorithms for tailored processing.

In this section, we present a set of algorithms that satisfy these requirements. They handle five main events that are common to SLAM and SLAM-like systems:

1. New-keyframe events, where newly initialized features are also added to $\{P\}$
2. Data-association events, entailing the addition of new visibility information
3. Data-dissociation events, entailing the deletion of erroneous visibility information
4. Deletion events, where outliers are removed from $\{P\}$
5. Refinement events, where subsets of $\{P\}$ and $\{O\}$ are reestimated and moved

Because the algorithms all operate on the same data structures, we first discuss their commonalities. What follows is a description of each event handler. Finally, we propose a heuristic that brings our method to real time.

2.1. Preliminaries / Commonalities

Our algorithms encompass a volumetric approach: we adaptively [8] discretize space via the 3D Delaunay triangulation of the input point set $\{P\}$. A triangulation of $\{P\}$ is a partition of the convex hull of $\{P\}$ into a connected set of tetrahedra. A Delaunay triangulation is one that satisfies the empty circumsphere property, *i.e.* the interior of every tetrahedron's circumsphere contains no vertices [22].

Each event handlers' goal is to maintain and update a carving of this discretized space. The algorithms prune entire tetrahedra by marking them as empty if they intersect a free-space constraint, as illustrated in Figure 1.

Given a current carving, it is straightforward to output the reconstruction as a triangulated mesh (since tetrahedral facets are triangles): we simply extract the isosurface as the facets between the carved and uncarved volumes.

When an event updates a subset of the point set $\{P\}$, the Delaunay descritization of a related subset of space changes in turn. Some tetrahedra are deleted, holes are thus created, and new tetrahedra fill the holes and take their place. Therefore, our algorithms must efficiently determine which of the newly created tetrahedra to mark as free space.

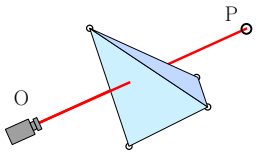


Figure 1. A free-space constraint. The camera, O, observes a point P. The tetrahedron is carved by the constraint: it intersects $\overline{OP}$ and thus would occlude the point.

For this purpose, we associate with each tetrahedron a set of all the free-space constraints that intersect it. The sets from the old tetrahedra that span the hole(s) determine the minimal constraint set that we must test against. In contrast to a batch method that would redo the triangulation from scratch and then retest all constraints, our algorithms save time by minimally processing only the new tetrahedra and relevant constraints.

However, this minimal processing is not enough to ensure real-time operation. As views are processed, free-space constraints quickly accumulate in the tetrahedra's constraint sets. The size of these sets balloon, and so does memory consumption and computation time. Thus, in Section 2.6, we propose a heuristic for keeping the size in check.

2.2. Keyframe Insertion

Upon insertion of a new keyframe, we must handle two changes. First, any point features that are initialized in this keyframe add to $\{P\}$ and thus change the discretization. Second, this keyframe's visibility list carves the resulting discretization.

To preserve the empty circumsphere property, when a new point is inserted, the triangulation algorithm deletes all tetrahedra that contain the point within their circumspheres and then retriangulates the resulting hole. The hole is always a connected set of tetrahedra, and a valid retriangulation can be performed by staring off the hole [22]; see Figure 2.

After retriangulating holes, our algorithm determines which of the new tetrahedra to mark as free space. The minimal constraint set to test against is the union of the constraint sets from the old tetrahedra that spanned the holes.

Finally, the free-space constraints induced by the current view are applied. The entire process is summarized in Figure 2 and in Algorithm 1.

2.3. Data Association and Dissociation

Data association and dissociation events only change visibility information. Association events add visibility rays, and dissociation events remove them. For example, if a point is initialized in some keyframe and then later matched to an earlier keyframe, this raises an association event. If, *e.g.* after a refinement event, a point's reprojection error in some keyframe is large, the corresponding visibility ray is likely erroneous. This may raise a dissociation event: that ray's carving must then be undone.

Algorithm 1 New-Keyframe Event Handler

$U \leftarrow \emptyset$, the empty constraint set
for all $Q \in \{P\}.NewPoints$ **do**
  $C \leftarrow \{$Cells whose circumspheres conflict with $Q\}$
  for all $T \in C$ **do**
    $U \leftarrow U \cup T.ConstraintSet$
  Insert Q into the triangulation and star off the hole
Apply all constraints $\in U$ as described in § 2.3
Apply constraints from the current visibility list

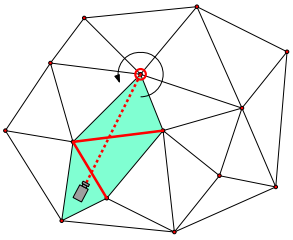


Figure 3. The traversal algorithm for processing a free-space constraint (dashed line). First, the cells adjacent to the highlighted vertex are tested for intersection with the constraint. Then, the algorithm hops from cell to adjacent cell via the red facets (bolded) until the camera center is reached. The carved cells are shaded.

To handle an association event, we only need to carve. To carve via a given free-space constraint $\overline{OP}$ with optic center O and feature point P, we adopt Gargallo's algorithm [9]. (This is our only event handler that is not novel.)

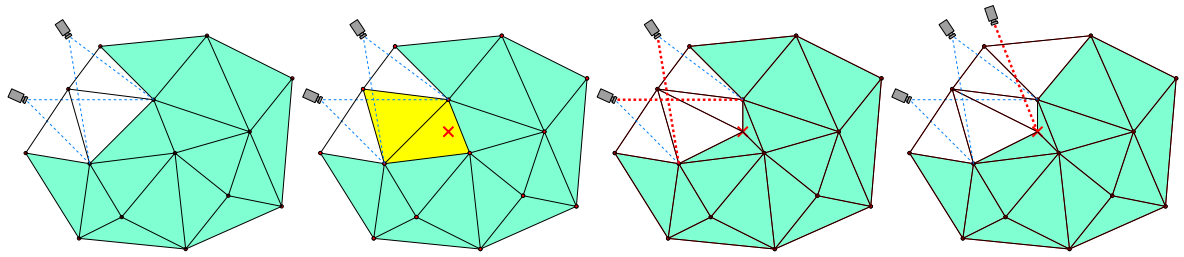
The algorithm is depicted in Figure 3. The line segment $\overline{OP}$ stabs a connected set of tetrahedra through their shared facets (red and bolded in the figure). This makes traversal possible: begin at vertex P, and perform a series of facet-segment intersection tests to determine which adjacent tetrahedron to traverse to. To find the initial tetrahedron, iterate over all the tetrahedra incident to P, testing only the facet opposite to P for intersection with $\overline{OP}$. If at any time the current tetrahedron contains point O in its interior, the traversal is complete.

We add $\overline{OP}$ to all the crossed tetrahedra's constraint sets.

To initially find vertex P, we maintain a vector of vertex pointers as we construct the triangulation. We represent $\overline{OP}$ as a pair of indices, and index the vector to find the vertex (in $O(1)$ time).

To ensure that the optic center is contained inside some tetrahedron in the triangulation, eight artificial vertices are initially inserted before all events. These vertices represent a loose bounding cube on the features and optic centers.

For dissociation events, our event handler simply traverses all tetrahedra in the triangulation to erase the selected



(a) Initial (b) Conflicts (c) Retriangulation (d) Final Carving

Figure 2. A 2D illustration of Algorithm 1 (keyframe insertion). (a) The initial triangulation. The blue dashed lines are free-space constraints currently in the triangulation. Shaded aquamarine cells have not yet been carved. (b) An incoming point (red cross). The yellow cells are in conflict with the point because it falls inside their circumcircles. (c) The yellow cells were deleted, and the hole stared off. The red free-space constraints (bolded) belonged to the deleted tetrahedra; they are now used to carve away two of the four new tetrahedra. (d) Finally, the new free-space constraint(s) from the current view are applied.

constraint from their intersection sets. See Algorithm 2.

Algorithm 2 Dissociation Event Handler

$\overline{OP} \leftarrow$ The free-space constraint to remove
for all Tetrahedra T **do**
  $T.ConstraintSet \leftarrow T.ConstraintSet \setminus \{\overline{OP}\}$

2.4. Outlier Deletion

When a point Q is marked for deletion, we must remove it from both $\{P\}$ and the triangulation, but we must also undo the carving of all visibility rays induced by it.

Therefore, Algorithm 3 first traverses all tetrahedra to remove any constraints incident to Q from their intersection sets.

Deletion of Q from a Delaunay triangulation rediscretizes space: a hole is formed and retriangulated. The hole is precisely the set of tetrahedra adjacent to Q [5]. This is because point removal is the inverse operation of point addition; a stared off hole around Q forms the set of tetrahedra adjacent to Q (*c.f.* Figure 2 and assume that Q was inserted last). A robust procedure for retriangulating the hole is given in [5].

Algorithm 3 therefore collects a set of free-space constraints from incident cells before removing Q, to apply this set afterwards.

2.5. Refinement

The refinement event handler is invoked whenever a set of points and cameras are moved. This happens continuously in SLAM, and it corresponds to partial or full bundle adjustments in online Structure-from-Motion.

Algorithm 4 summarizes. It is essentially a series of point deletions and insertions (akin to Sections 2.2 and 2.4), with free-space constraints collected and applied only when necessary.

Algorithm 3 Outlier-Deletion Event Handler

$U \leftarrow \emptyset$, the empty constraint set
$Q \leftarrow$ Point marked for deletion
// Remove all constraints that reference Q
for all Tetrahedra T **do**
  $T.ConstraintSet \leftarrow T.ConstraintSet \setminus \{\overline{OP} \mid P = Q\}$
// Collect constraints from incident cells
$C \leftarrow \{$Cells incident to $Q\}$
for all $T \in C$ **do**
  $U \leftarrow U \cup T.ConstraintSet$
Delete Q from the triangulation (this retriangulates)
Apply all constraints $\in U$ as described in § 2.3

For efficiency, we implement this handler slightly incorrectly: we partially ignore the movement of keyframes, and thus of some visibility constraints. This works due to two realistic assumptions. First, we assume that keyframes are initialized accurately such that they only move slightly. Second, we rely on future events to re-discretize the remainder of space; constraints then will be reprocessed using the moved optic centers. This implementation choice allows for a better run-time complexity that is independent of the number of keyframes in $\{O\}$; see Section 3.

2.6. Forgetting Heuristic

The heuristic is simple: retain the K least similar constraints in each tetrahedron's constraint set. As shown in Sections 3 and 5, the efficiency greatly improves with insignificant impact on reconstruction quality.

We define the similarity measure by (the inverse of) the sum of the areas spanned between the two constraints; see Figure 4. By retaining the K most spatially distinct constraints that intersect each tetrahedron, we hope to cover as much volume as possible so that, when a hole is retriangu-

Algorithm 4 Refinement Event Handler

// Collect constraints from incident cells
$U \leftarrow \emptyset$, the empty constraint set
for all $Q \in \{P\}.MovedPoints$ **do**
  $C \leftarrow$ {Cells incident to Q}
  for all $T \in C$ **do**
    $U \leftarrow U \cup T.ConstraintSet$
// Remove the vertices (this retriangulates)
for all $Q \in \{P\}.MovedPoints$ **do**
  Delete Q from the triangulation
// Insert moved vertices while collecting constraints
for all $Q \in \{P\}.MovedPoints$ **do**
  // Q_{moved} **refers to** Q**'s new coordinates.**
  $C \leftarrow$ {Cells Delaunay-conflicting with Q_{moved}}
  for all $T \in C$ **do**
    $U \leftarrow U \cup T.ConstraintSet$
  Insert Q_{moved} into the triangulation
Remove all constraints from the triangulation that reference moved points, like in Algorithm 3
Apply all constraints $\in U$ as described in § 2.3, using moved point and camera locations

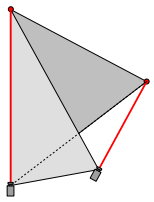


Figure 4. The similarity of the left-hand segment w.r.t. the right-hand segment is equal to 1 over the sum of areas spanned by the gray triangles.

lated, space is sampled well enough that the new tetrahedra can be carved.

This is not a true similarity measure as it is asymmetric, but it is only a heuristic. The asymmetry arises because the areas depend on the base segment's length $||\overline{OP}||$. This weights longer segments as more different, which usually is desirable.

When a set is full, an incoming constraint is inserted iff it is less similar to all the constraints than their most similar constraint in the set. This knocks out the constraint with the highest (asymmetric) similarity score.

In the case $K = 1$, we simply retain the first inserted constraint and reject all others. For $K = \infty$, the set is never full, so no similarity measures need to be computed.

3. Complexity

The run-time bounds derived in this section are not intended to be tight. Instead, in conjunction with the experiments in Section 5, they serve to illustrate the real-time quality of our algorithms.

First, we analyze the version that employs the forgetting heuristic ($K < \infty$). Let N be the number of input points, and M the number of views.

Theorem 1. *The worst-case run-time complexity of Algorithm 1 is $O(N^4)$, for $K < \infty$.*

Proof. First, we cite some relevant properties of the 3D Delaunay triangulation. In the worst case, the number of tetrahedra in the triangulation, as well as the number of tetrahedra that a line can intersect, is of order $O(N^2)$ [11, 21]. Because we have a structured point set, these bounds are loose. (Several papers suggest or prove tighter bounds for point sets sampled from smooth surfaces [2, 8].)

The algorithm can be split into two phases: point insertions plus retriangulation with recarving, and carving via the current view's free-space constraints.

For the first phase, there are at most N vertices to insert. For each insertion, at worst all $O(N^2)$ tetrahedra conflict, and thus are deleted and stared off in $O(N^2)$ time. Locating the conflicting cells takes no more than $O(N^2)$ time, since even a naive enumeration of all $O(N^2)$ tetrahedra suffices. Thus, inserting all the vertices takes $O(N^3)$ time.

To recarve the new tetrahedra, the constraints from the old tetrahedra are collected into a single set and then applied. Since $K < \infty$ and since the number of deleted tetrahedra is $O(N^2)$, the number of constraints to reprocess is $O(N^2)$. Therefore, inserting them into a set can be done in $O(N^2 \log(N^2)) = O(N^2 \log(N))$ time using a red-black tree. Theorem 2 shows that applying a single constraint takes $O(N^2)$ time. Since there are $O(N^2)$ constraints, the total time for the first phase of an iteration is $O(N^4)$.

For the second phase of an iteration, because at most N points can be observed in a single view, there are at most N free-space constraints to apply. Therefore, the second phase takes $O(N^3)$ time.

Thus, the complete algorithm takes $O(N^4 + N^3) = O(N^4)$ time per iteration. □

Theorem 2. *The worst-case run-time complexity of carving a free-space constraint $\overline{OP}$ (§ 2.3) is $O(N^2)$, for $K < \infty$.*

Proof. Because segment $\overline{OP}$ can intersect at most $O(N^2)$ tetrahedra [21], and because the number of tetrahedra incident to P is bounded by $O(N^2)$ [11], the traversal takes $O(CN^2)$ time. Here C refers to the cost of inserting a free-space constraint into a tetrahedron's constraint set. C depends on K, and because K is a constant, $O(C) = O(f(K)) = O(1)$. Thus a single traversal takes $O(N^2)$ time. □

The worst-case complexities of Algorithms 2, 3, and 4 are $O(N^2)$, $O(N^4)$, and $O(N^4)$ respectively, for $K < \infty$.

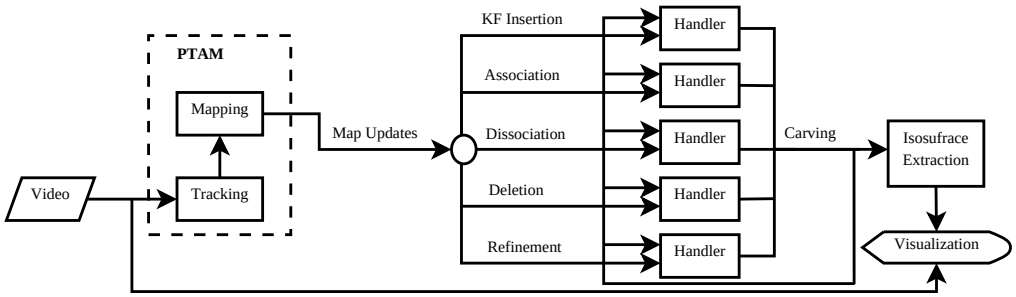


Figure 5. Reconstruction System.

Proofs for these bounds are very similar, straightforward, and therefore omitted. We cite one used fact: deletion of a point Q from a Delaunay triangulation takes $O(fd) \rightarrow O(N^3)$ time, where f is the number of tetrahedra that retriangulate the hole, and d is the degree of vertex Q [5].

For a closed bounded scene, we can assume that the average number of features observed in a given view, N_{avg}, is proportional to N [10]. Therefore, the average case complexity of all our event handlers is $O({N_{avg}}^4)$. Because N_{avg} depends on system properties such as the camera's field of view and the spatial density of the tracked point set, the per-event time complexity is practically constant [10], which is desirable for an online algorithm.

Without the forgetting heuristic ($K = \infty$), the situation is worse. In this case, there are at most $N * M$ constraints in the triangulation. If, upon a point insertion, all tetrahedra are deleted and retriangulated, then just collecting the constraints into a set takes $O(NM \log(NM))$ time, which is superlinear in M.

We do not provide a complete analysis for $K = \infty$. It suffices to show that the complexity is dependent on M, and thus not suitable for online use. Our experimental results support this.

We remark that ProFORMA [17], a similar free-space carving system, processes $O(NM)$ constraints every keyframe. Therefore their per-frame processing time is $\Omega(NM)$. This dependence on M shows that ProFORMA must eventually slow below real-time capability.

4. System Description

By integrating our algorithms with PTAM [12], we have devised a complete system that reconstructs 3D meshes from video.

Figure 5 shows the system's components and the flow of data. PTAM consists of two main threads: the tracker and the mapper. The mapper is responsible for producing the information that our algorithms operate on. Our routines accept this information in the form of events.

The current integration is far from optimized; event handlers operate in the mapper thread, and they parse argument strings to extract the event type and data. Ideally, the handlers should operate in their own thread to benefit from *e.g.* quad-core processors. The string parsing is a relic from logging and offline testing. In spite of this, we still attain real-time tracking, mapping, modeling, and rendering on a two-year-old laptop (Intel Core2 Duo 1.83 GHz processor and 3 GB of RAM).

The visualization is also preliminary. It only projects and blends textures from the four most recent keyframes onto the model for rendering.

Our implementation makes use of the CGAL software package for Delaunay triangulations [1]. CGAL is numerically robust and fast, and it provides all the necessary operations, such as point insertion and efficient traversals.

5. Experiments

Our method was tested on numerous datasets; we present three different results in Figure 6. Sample input images along with shaded and textured output models are shown. We refer to the datasets in the figure, from top to bottom, as "Shelves", "Fireplace", and "House" respectively. These results were generated in real-time using the system described in Section 4, and the heuristic parameter $K = 1$.

The results are promising. Even a complex cluttered scene like "Shelves" reconstructs well. "Fireplace" contains specular surfaces to no detriment, *e.g.* the metal fireplace door and glass-framed portraits. This demonstrates the robustness of our feature-based approach. Even though our method carves the convex hull of the features using sparse visibility information, the recovered geometries closely resemble the highly concave scenes, *e.g.* "House".

Reconstruction quality is however not perfect. As our method employs no noise model and no regularization, the meshes are noisy and include some stray uncarved tetrahedra. However, the method is intended for fast, approximate reconstruction. We accomplish this goal: the meshes are adequate geometry proxies for image-based rendering.

Figure 7 compares timings, and the number of free-space constraints retained, for $K = 1$, 5, and ∞ on a typical dataset from our system. For $K = \infty$, as the number

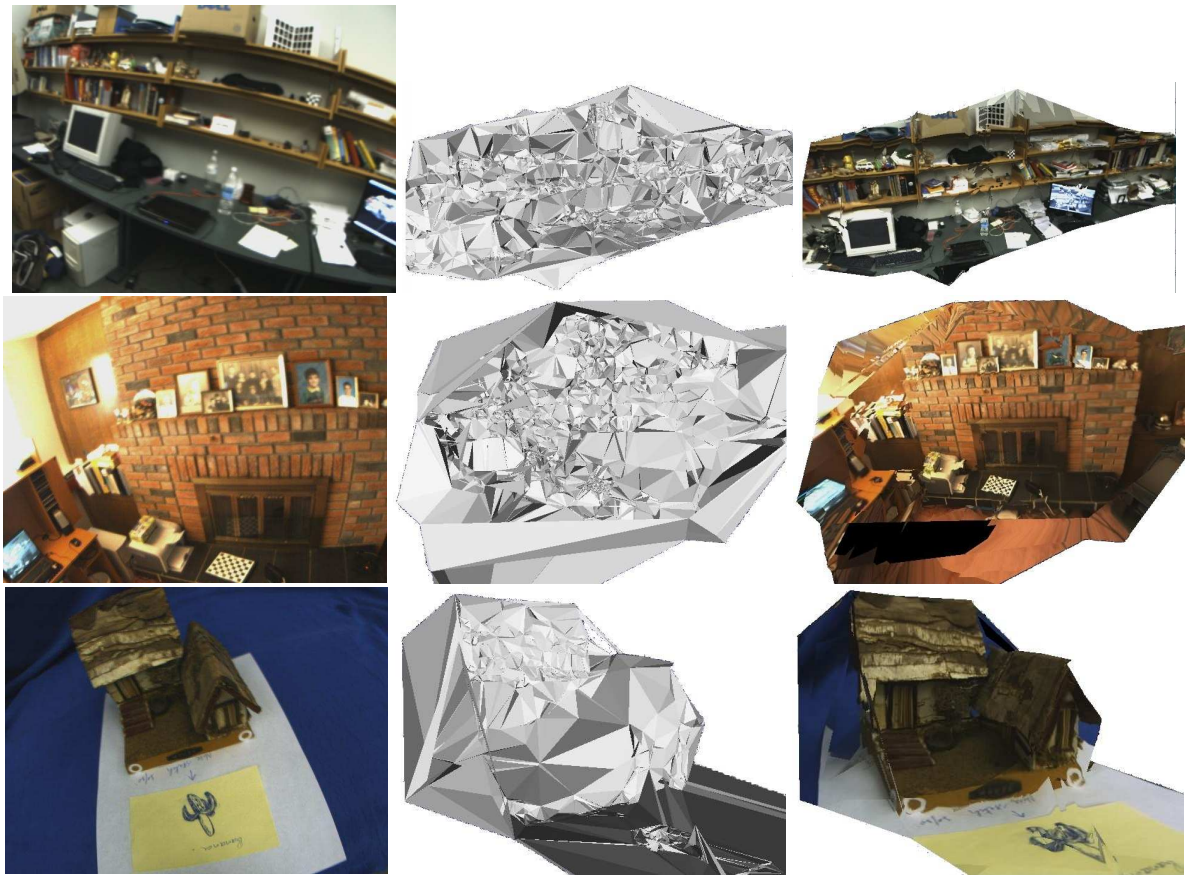


Figure 6. Our algorithm's results on three data sets. Top row: "Shelves" dataset. Middle row: "Fireplace." Bottom row: "House." Left: A sample input image from the dataset. Middle: Shaded final models. Right: Offline view-dependent texture rendering.

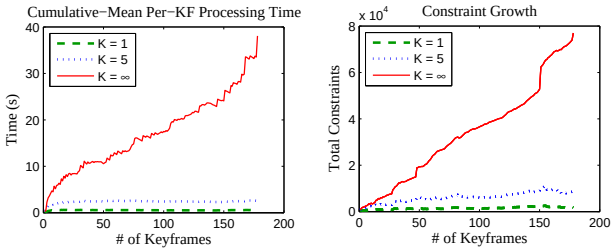


Figure 7. Efficiency for $K = 1, 5$, and ∞ on a representative dataset. Left: Mean per-keyframe processing time as a function of the number of views processed. Right: Number of free-space constraints in the triangulation (a function of the same). Our heuristic effectively bounds computation time and memory usage.

of views increases, the per-keyframe processing time and memory consumption grow. For finite K however (*i.e.* the forgetting heuristic), they are tightly bounded and quickly level off, which supports the complexity results of Section 3. We conclude that with finite K, our algorithms can operate in real time on image sequences of arbitrary length.

All timings were collected by running our algorithms offline on an event log. The per-keyframe times count the time spent handling all events raised between adjacent keyframes. We average over 30 independent runs before computing the means, except $K = \infty$ was averaged over only 10 due to lengthy run-time. The challenging dataset contained 2887 points in 178 keyframes. This is large compared to typical output of SLAM-type systems [4, 6].

Finally, Figure 8 shows that the outputs for $K = 1$ and $K = \infty$ are almost identical. Thus the difference in carving-quality when using the forgetting heuristic is almost negligible. Note that this result compares old datasets produced by offline Structure-from-Motion: we artifically simulated a set of new-keyframe events to feed to our method. Online results, however, support the conclusion: carving is effective with $K = 1$ for all event-types, see Figure 6.

6. Conclusions

This paper presented a novel incremental method for 3D scene reconstruction by free-space carving, along with a complete real-time system that implements it. Experiments

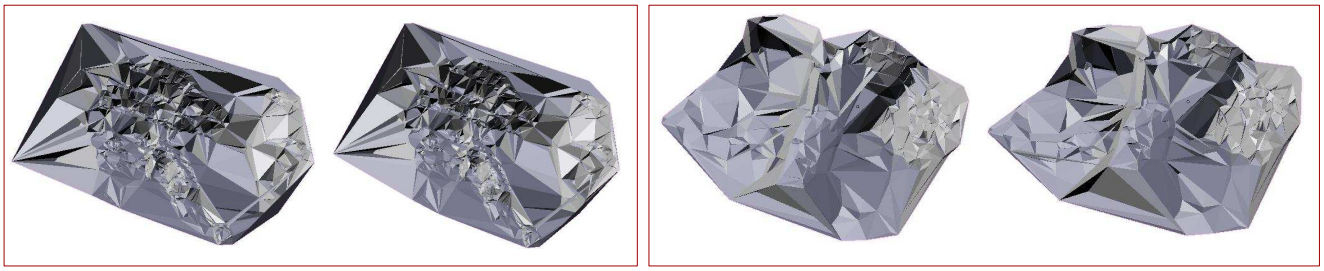


Figure 8. Results for $K = \infty$ (left) and $K = 1$ (right) on different data sets. The meshes are similar for extremely different K.

demonstrate that the system generates approximate models that well resemble the scene. Texturing with input-imagery provides novel-view visualization, both online and offline.

There are several directions for future work. First, we plan to apply our work to predictive display for telerobotics, and also to simultaneous capture and visualization. We will assess the benefits that our method provides in both of these contexts. Second, we will investigate ways to improve the mesh quality. Our isosurface extraction is minimal; combining it with regularization will improve results. Yet another direction is to see if our outputs can be used to initialize geometric refinement algorithms, *e.g*. variational multi-view stereo. Often, good initializations for open scenes are not available—our algorithms might provide them at low expense.

Acknowledgements

The authors wish to acknowledge funding and in-kind support from NSERC, CFI, CSA and Barrett. We also thank Adam Rachmielowski for data and for helpful discussions.

References

[1] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org. 6

[2] D. Attali, J. Boissonnat, and A. Lieutier. Complexity of the delaunay triangulation of points on surfaces the smooth case. In *ACM Symposium on Computational Geometry*, pages 201–210, 2003. 5

[3] T. Burkert, J. Leupold, and G. Passig. A Photorealistic Predictive Display. *Presence: Teleoperators & Virtual Environments*, 13(1):22–43, 2004. 2

[4] A. Davison. Real-time simultaneous localisation and mapping with a single camera. In *ICCV*, pages 1403–1410, 2003. 1, 2, 7

[5] O. Devillers and M. Teillaud. Perturbations and vertex removal in a 3D Delaunay triangulation. In *ACM-SIAM Symposium on Discrete algorithms*, pages 313–319, 2003. 4, 6

[6] E. Eade and T. Drummond. Scalable Monocular SLAM. In *CVPR*, volume 1, pages 469–476, 2006. 1, 2, 7

[7] M. Eisemann, B. D. Decker, M. Magnor, P. Bekaert, E. de Aguiar, N. Ahmed, C. Theobalt, and A. Sellent. Floating Textures. *Computer Graphics Forum (Proc. Eurographics EG'08)*, 27(2):409–418, 4 2008. 2

[8] O. Faugeras, E. Le Bras-Mehlman, and J. Boissonnat. Representing Stereo Data with the Delaunay Triangulation. *Artificial Intelligence*, 44(1-2):41–87, July 1990. 1, 2, 5

[9] P. Gargallo. Modélisation de Surfaces en Vision 3D. Master's thesis, INRIA Grenoble Rhône-Alpes, 2003. 1, 3

[10] A. Hilton. Scene modelling from sparse 3D data. *Image and Vision Computing*, 23(10):900–920, 2005. 1, 6

[11] V. Klee. On the complexity of d-dimensional Voronoi diagrams. *Archiv der Mathematik*, 34(1):75–80, 1980. 5

[12] G. Klein and D. Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *ISMAR*, pages 1–10, 2007. 1, 2, 6

[13] P. Labatut, J. Pons, and R. Keriven. Efficient Multi-View Reconstruction of Large-Scale Scenes using Interest Points, Delaunay Triangulation and Graph Cuts. In *ICCV*, 2007. 1

[14] M. Levoy and P. Hanrahan. Light Field Rendering. *SIGGRAPH*, pages 31–42, 1996. 2

[15] W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan. Image-Based Visual Hulls. *SIGGRAPH*, pages 369–374, 2000. 2

[16] P. Merrell, A. Akbarzadeh, L. Wang, P. Mordohai, J. Frahm, R. Yang, D. Nistér, and M. Pollefeys. Real-Time Visibility-Based Fusion of Depth Maps. In *ICCV*, 2007. 1

[17] Q. Pan, G. Reitmayr, and T. Drummond. ProFORMA: Probabilistic Feature-based On-line Rapid Model Acquisition. In *BMVC*, London, September 2009. 1, 6

[18] M. Pollefeys, D. Nistér, J. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S. Kim, P. Merrell, et al. Detailed real-time urban 3d reconstruction from video. *IJCV*, 78(2):143–167, 2008. 1

[19] A. Rachmielowski. Concurrent acquisition, reconstruction, and visualization with monocular video. Master's thesis, University of Alberta, 2009. 2

[20] S. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms. In *CVPR*, pages 519–526, 2006. 1

[21] J. Shewchuk. Stabbing Delaunay Tetrahedralizations. *Discrete and Computational Geometry*, 32(3):339–343, 2004. 5

[22] K. Sugihara and H. Inagaki. Why is the 3D Delaunay triangulation difficult to construct? *Information Processing Letters*, 54(5):275–280, 1995. 2, 3

[23] H. Vu, R. Keriven, P. Labatut, and J.-P. Pons. Towards high-resolution large-scale multi-view stereo. In *CVPR*, 2009. 1