N° d'ordre: D.U. 2356
EDSPIC: 610

UNIVERSITÉ BLAISE PASCAL - CLERMONT II

ÉCOLE DOCTORALE
SCIENCES POUR L'INGÉNIEUR DE CLERMONT-FERRAND

Thèse

présentée par

Shuda YU

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

Spécialité: Vision pour la Robotique

Modélisation 3d automatique d'environnements: une approche éparse à partir d'images prises par une caméra catadioptrique

Automatic 3d modeling of environments: a sparse approach from images taken by a catadioptric camera

Soutenue publiquement le 3 juin 2013 devant le jury:

Mme Luce Morin	PR INSA de Rennes	Président du jury
Mme Raphaëlle Chaine	MCF Université Lyon 1	Rapporteur
M. Helder Araújo	PR University of Coimbra	Rapporteur
M. Luc Robert	Manager Autodesk	Examinateur
M. Thierry Chateau	PR Institut Pascal	Examinateur
M. Maxime Lhuillier	CR CNRS Institut Pascal	Directeur de thèse

Acknowledgements

First of all, I would like to thank Luce Morin, Raphaëlle Chaine, Helder Araújo, Luc Robert and Thierry Chateau who agreed to be part of my thesis committee and take time to evaluate my work.

I would like sincerely thank my advisor Maxime LHUILLIER who has proposed this thesis to me. And I thank him also for his guidance, leadership and ongoing support during my studies.

I do not forget of course all members of the laboratory who have greatly helped me solve problems during my research. I refer in particular to Baptiste Charmette and Pierre Lébraly, and other colleagues for their helpful advises.

Finally I thank my family and friends who were able to bring me all the support that I needed during the production of this thesis.

Résumé

La modélisation 3d automatique d'un environnement à partir d'images est un sujet toujours d'actualité en vision par ordinateur. Ce problème se résout en général en trois temps: déplacer une caméra dans la scène pour prendre la séquence d'images, reconstruire la géométrie, et utiliser une méthode de stéréo dense pour obtenir une surface de la scène. La seconde étape met en correspondances des points d'intérêts dans les images puis estime simultanément les poses de la caméra et un nuage épars de points 3d de la scène correspondant aux points d'intérêts. La troisième étape utilise l'information sur l'ensemble des pixels pour reconstruire une surface de la scène, par exemple en estimant un nuage de points dense.

Ici nous proposons de traiter le problème en calculant directement une surface à partir du nuage épars de points et de son information de visibilité fournis par l'estimation de la géométrie. Les avantages sont des faibles complexités en temps et en espace, ce qui est utile par exemple pour obtenir des modèles compacts de grands environnements comme une ville.

Pour cela, nous présentons une méthode de reconstruction de surface du type sculpture dans une triangulation de Delaunay 3d des points reconstruits. L'information de visibilité est utilisée pour classer les tétraèdres en espace vide ou matière. Puis une surface est extraite de sorte à séparer au mieux ces tétraèdres à l'aide d'une méthode gloutonne et d'une minorité de points de Steiner. On impose sur la surface la contrainte de 2-variété pour permettre des traitements ultérieurs classiques tels que lissage, raffinement par optimisation de photo-consistance ... Cette méthode a ensuite été étendue au cas incrémental: à chaque nouvelle image clef sélectionnée dans une vidéo, de nouveaux points 3d et une nouvelle pose sont estimés, puis la surface est mise à jour. La complexité en temps est étudiée dans les deux cas (incrémental ou non).

Dans les expériences, nous utilisons une caméra catadioptrique bas coût et obtenons des modèles 3d texturés pour des environnements complets incluant bâtiments, sol, végétation... Un inconvénient de nos méthodes est que la reconstruction des éléments fins de la scène n'est pas correcte, par exemple les branches des arbres et les pylônes électriques.

Mots-clefs: Reconstruction de 2-variété, triangulation de Delaunay 3d, sommets de Steiner, analyse de complexité, nuage de points épars, Structure-from-Motion.

Abstract

The automatic 3d modeling of an environment using images is still an active topic in Computer Vision. Standard methods have three steps: moving a camera in the environment to take an image sequence, reconstructing the geometry of the environment, and applying a dense stereo method to obtain a surface model of the environment. In the second step, interest points are detected and matched in images, then camera poses and a sparse cloud of 3d points corresponding to the interest points are simultaneously estimated. In the third step, all pixels of images are used to reconstruct a surface of the environment, e.g. by estimating a dense cloud of 3d points.

Here we propose to generate a surface directly from the sparse point cloud and its visibility information provided by the geometry reconstruction step. The advantages are low time and space complexities; this is useful e.g. for obtaining compact models of large and complete environments like a city.

To do so, a surface reconstruction method by sculpting 3d Delaunay triangulation of the reconstructed points is proposed. The visibility information is used to classify the tetrahedra in free-space and matter. Then a surface is extracted thanks to a greedy method and a minority of Steiner points. The 2-manifold constraint is enforced on the surface to allow standard surface post-processing such as denoising, refinement by photo-consistency optimization ... This method is also extended to the incremental case: each time a new key-frame is selected in the input video, new 3d points and camera pose are estimated, then the reconstructed surface is updated. We study the time complexity in both cases (incremental or not).

In experiments, a low-cost catadioptric camera is used to generate textured 3d models for complete environments including buildings, ground, vegetation ... A drawback of our methods is that thin scene components cannot be correctly reconstructed, e.g. tree branches and electric posts.

Key-words: 2-Manifold Reconstruction, 3d Delaunay Triangulation, Steiner Vertices, Complexity Analysis, Sparse Point Cloud, Structure-from-Motion.

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Notations

Here we give the notations which are frequently used in this dissertation.

- $\mathbb{R}$ is the set of reals.
- $\mathbb{R}^+$ is the set of positive reals.
- $\mathbb{R}^3$ is the 3d space, i.e. the set of triples in which each coordinate is a real.
- $\mathcal{O}$ is the "big o" notation used in the complexity studies, which should not be confounded with the letter O.
- I^i is the image with index i in an image sequence.
- $\mathbf{c}^i$ stands for a reconstructed camera pose (rotation and location in 3d) corresponding to the image I^i.
- $\mathbf{t}^i$ is the location of camera pose $\mathbf{c}^i$.
- $\mathbf{q}_j$ is a reconstructed 3d point with index j.
- $C = \{\mathbf{c}^1, .., \mathbf{c}^m\}$ is the list of reconstructed camera poses.
- $T = \{\mathbf{t}^1, .., \mathbf{t}^m\}$ is the list of reconstructed camera viewpoints locations.
- $Q = \{\mathbf{q}_1, .., \mathbf{q}_n\}$ is the list of reconstructed 3d points.
- I_3 is the 3×3 identity matrix.
- 3×3 matrix $[\mathbf{a}]_\times$ is the skew-symmetric matrix for the 3d vector $\mathbf{a}$: let $\mathbf{a}, \mathbf{b}$ be two vectors in 3d, the cross product meets $\mathbf{a} \wedge \mathbf{b} = [\mathbf{a}]_\times \mathbf{b}$.

For 3d geometry reconstruction (Chap. 3), we have also

- $\mathbf{p}^i_j$ is the matched 2d point of the 3d point $\mathbf{q}_j$ in the image I^i.
- $[\mathbf{t}^i \mathbf{q}_j)$ is the ray (half-line) in 3d which originates from camera location $\mathbf{t}^i$ and goes through the reconstructed 3d point $\mathbf{q}_j$.
- $\mathbf{D}^i_j$ is the direction vector of ray $[\mathbf{t}^i \mathbf{q}_j)$ in the camera coordinate system.

- $[\mathbf{t}^i\mathbf{p}^i_j)$ is the ray (half-line) in 3d which originates from camera location $\mathbf{t}^i$ and goes through the matched 2d point $\mathbf{p}^i_j$.
- $\mathbf{d}^i_j$ is the direction vector of ray $[\mathbf{t}^i\mathbf{p}^i_j)$ in the camera coordinate system.

For surface reconstructions (Chap. 4 and Chap. 5), we have also

- $\mathbf{t}^i\mathbf{q}_j$ is the visibility constraint (line segment) between camera viewpoint location $\mathbf{t}^i$ and reconstructed 3d point $\mathbf{q}_j$.
- V_j is the visibility constraint list of point $\mathbf{q}_j$.
- D is a 3d Delaunay triangulation.
- Δ is a tetrahedron of D.

Introduction

Image-based Automatic 3d Modeling

Human beings use eyes to acquire visual information of environments. By analyzing the visual information, our brains are able to perceive 3d information of environments. However, human eyes cannot store images and the acquired information can be quickly lost due to the forget function of human brains. The digital camera is invented to solve this problem. Visual information of an environment can be stored in 2d images by using a digital camera. Then by interpreting several images of a same environment taken by a digital camera with the help of a computer, 3d information of the environment can be obtained and a 3d model of the real environment can be estimated. The process of automatically estimating 3d models of environments using 2d images is called the image-based automatic 3d modeling. Here, the word "automatically" should be understood in a sense that images are manually taken by cameras and the 3d modeling using these images is automatically done by a computer.

Numerous applications can be realized on the estimated 3d models. The models can be simply rendered for an interactive visualization, and the related applications cover variant fields, such as virtual visit in games and entertainment, prototyping in manufacturing, visualization and simulation in geology, diagnosis in medicine etc. The estimated models can also be used for further processing such as robot auto-navigation, 3d printing, computer aided animation and film etc.

There are several categories of models used for image-based automatic 3d modeling. The simplest model is the 3d point cloud: a set of independent 3d points reconstructed from images and each 3d point has 3d coordinate–supposing there exists a Cartesian world coordinates system–and eventually other information such as the color, the orientation etc. The 3d point cloud is sufficient for some applications such as localization for robot auto-navigation or the visualization in case that the point cloud is very dense. However, more sophisticated models like depth maps, voxels, surfaces, are preferred for most of the applications. A very common model used by the image-based 3d modeling methods is the surface. A surface can be defined as a function which maps a 2d parameter domain into the 3d space, and we talk about the parametric surface. It can also be defined as the zero set of a scalar function, e.g. the zero set of a distance function which maps every node of a Cartesian grid to its signed distance to the surface, and in this case we talk about the implicit surface.

The image-based automatic 3d modeling is challenging in a lot of aspects. First, the input is a set of images which contain rich information, e.g. one image contains often millions of

8 bits (gray level) or 24 bits (color) information. A lot of the information is redundant at the most of the time, so it should be efficiently interpreted to provide 3d information of environments. Second, the estimated models of environments should be close to the real environments as much as possible. In case that only images are available, the estimated models should be consistent as much as possible to the input images, which contain noise due to the sensor and the image taking conditions. Third, the estimated models should be adequate to the aimed application, e.g. many Computer Graphics applications require the models to verify some topology properties. Fourth, more and more applications need the modeling process be incremental or real time. It means that the estimated models of environments should be progressively updated when new images of the environment are provided and the computation time of each model updating process should be small enough to satisfy the real time performance.

Objective and Our Methods

The objective of this thesis is to develop new image-based automatic 3d modeling methods which deal with complete environments such as part of a city. In our case, we assume that objects of environments are rigid and we do not deal with 3d modeling for deformable scenes. Besides, our methods do not use active sensors such as a range scanner. We believe that the passive camera is more adequate for a large-scale environment modeling: most of the active sensors are limited to indoor scene modeling or are very expensive. Furthermore, a camera can deal with both indoor and outdoor environments and has a low price. In our work, we use the catadioptric (omni-directional) camera, which provides wide field-of-view images. Indeed, a wide field of view is required to reconstruct every point of the environment.

To estimate models of the environment, most of previous works involve all or most of the image pixels to estimate the model. Some of them directly estimate a 3d model by satisfying the consistency between the estimated model and input images, e.g. multi-view stereo methods surveyed in [124]. Others firstly use dense matching to estimate the geometry of the environment, which is easily converted to a dense 3d point cloud, then reconstruct a surface to fit the reconstructed geometry, e.g. [23, 76]. In contrast to methods above, our methods reconstruct few pixels of input images in 3d to form a sparse 3d point cloud by using Structure-from-Motion (SfM), and reconstruct a surface using a 3d Delaunay triangulation of the estimated point cloud and visibility information between camera viewpoints and estimated points. Thus our modeling methods can be seen as surface reconstruction methods with a geometry reconstruction pre-processing step, i.e. the sparse point cloud computation.

Many Computer-Aided Design methods uses 2-manifolds for object modeling [68]. Besides, a lot of the surface reconstruction methods in the Computer Graphics field (Sec. 2.2.2) also enforce manifold property on the output surface. Such a surface has a lot of advantages in post-processing (see Sec. 1.4.2). Our methods also generate a manifold surface which can have multi genus, by alternating a greedy algorithm and a topology extension process. The surface denoising post-processing of our methods is improved thanks to the manifold property of the reconstructed surface. Besides, the reconstructed manifold surface is a compact surface model of environments, which can also be a good initialization for a

more sophisticated dense stereo method.

Finally, two image-based automatic 3d modeling methods have been developed in our work: a batch and an incremental method. Both methods have features described above. The batch modeling method assumes that all input images are available at a time. The geometry reconstruction uses all images to firstly estimate a sparse point cloud, which is then used to reconstruct the environment surface. Regarding the incremental method, the input images are assumed to be progressively provided. A sparse point cloud and a surface are initialized using the first three images. Then for each image, new points are reconstructed to update the sparse point cloud and the environment surface is incrementally reconstructed. The incremental surface reconstruction here is not a complete surface reconstruction using the current point cloud, but an update based on previous surface results. In the ideal case, the processing time per image of an incremental surface reconstruction does not increase with time.

Contributions and Document Structure

Now we summarize the major contributions of this thesis. These works also lead to several publications.

- A batch image-based automatic 3d modeling method has been proposed [1, 3, 6]. The manifold constraint and the visibility information are combined to estimate a multi-genus manifold surface of environments, by using sparse SfM points reconstructed from image or video sequences. Besides, in order to improve the surface quality, spurious handles existing on the reconstructed 2-manifold surface are detected and removed using a "detect, force and repair" process [5].

- An incremental environment modeling method [4, 6] is also proposed. It is an incremental version of the batch method. Manifold surfaces of environments are incrementally reconstructed and both our batch and incremental environment modeling methods are able to deal with long image or video sequences.

- Complete and detailed time complexity analyses of our surface reconstruction methods are given, under different assumptions on the input data sets.

- Various experiments are done on quite large image sequences of complete environments. Our surface reconstruction methods take only a few seconds/minutes for hundreds/thousands images.

The dissertation is organized as follows. Chap. 1 gives preliminary notions on 2-manifold, simplicial complex, triangulated 2-manifold and 3d Delaunay triangulation. Chap. 2 investigates the existing image-based automatic 3d modeling methods. Chap. 3 presents the geometry reconstruction methods used in our modeling methods. Chap. 4 and Chap. 5 show respectively our batch and incremental surface reconstruction methods. The time complexity analyses of both methods are in Chap. 6. Chap. 7 describes experiments of our modeling methods. Last, we conclude in Chap. 8.

LIST OF TABLES

Chapter 1

Preliminaries

1.1 2-Manifold

This section presents the definition of 2-manifold. A 2-manifold is a surface that can be defined in $\mathbb{R}^3$ and it is a special case of topological space [99]. Now we begin by the definition of topological space.

Definition 1.1 (Topological space) *Let X be a set and T be a set of subsets of X. The pair (X, T) is a topological space if*

- $\emptyset \in T$ *and* $X \in T$,
- *every union of elements of T is also an element of T,*
- *every finite intersection of elements of T is also an element of T.*

The elements of T are called open sets, and T is called the topology for X. Intuitively, we can "think of a topology as the knowledge of the connectivity of a space X: each point in X knows which points are near it, that is, in its neighborhood" [152].

The Euclidean spaces $\mathbb{R}^k$ are examples of topological spaces. Indeed, the Euclidean distance d of $\mathbb{R}^k$ define open balls which generate a topology for $\mathbb{R}^k$. Here we omit the notation of the topology for $\mathbb{R}^k$. An open ball has center $\mathbf{c} \in \mathbb{R}^k$, radius $r > 0$, and it is the set of points

$$B_k(\mathbf{c}, r) = \{\mathbf{x} \in \mathbb{R}^k, d(\mathbf{x}, \mathbf{c}) < r\}. \tag{1.1}$$

Let $M \subseteq X$ and $T_M = \{V \cap M, V \in T\}$. Then (M, T_M) is a topological space. We say that (M, T_M) is a subspace of (X, T) and T_M is the induced topology of T by M. In our case, $X = \mathbb{R}^3$, T is the topology generated by d, and the topological subspace (M, T_M) becomes a 2-manifold once we describe a tool to assert that (M, T_M) has dimension 2. Here is this tool.

Definition 1.2 (Homeomorphism) *Let (X_1, T_1) and (X_2, T_2) be two topological spaces. Function $f : X_1 \to X_2$ is a homeomorphism between these spaces if*

- *f is bijective*
- *f is continuous, i.e. $\forall V \in T_2, f^{-1}(V) \in T_1$*
- *f^{-1} is continuous, i.e. $\forall V \in T_1, f(V) \in T_2$.*

In this case, we say that spaces (X_1, T_1) and (X_2, T_2) are homeomorphic. For example, it can be shown that $B_k(\mathbf{0}, 1)$ and $\mathbb{R}^k$ are homeomorphic. Now we can define a k-manifold.

Definition 1.3 (k-manifold in $\mathbb{R}^n$) *Let $M \subseteq \mathbb{R}^n$ and $k \in \mathbb{N}$ such that $1 \leq k \leq n$. We say that (M, T_M) is a k-manifold in $\mathbb{R}^n$ if every $\mathbf{x} \in M$ is contained in an open set $V \in T_M$ such that V is homeomorphic to $B_k(\mathbf{0}, 1)$.*

As mentioned in [152], "Intuitively, a k-manifold is a topological space that locally looks like $\mathbb{R}^k$." In other words, we can parameterize a small enough neighborhood in M of every point $\mathbf{x} \in M$ by k real parameters. In our work, we say that M is a k-manifold omitting the induced topology T_M.

See Fig. 1.1 for examples of 2-manifolds: a two-dimensional Euclidean sphere, border of a cube, border of tetrahedron (a manifold does not need to be "smooth"). Furthermore, these three 2-manifolds are homeomorphic.

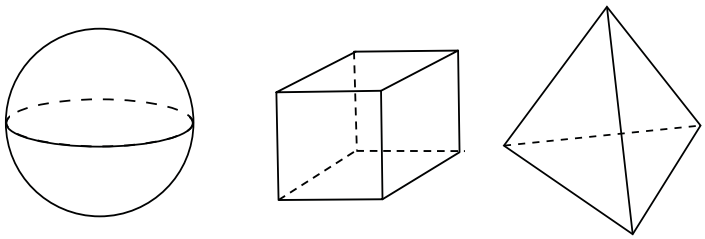


Figure 1.1: Examples of homeomorphic 2-manifolds. From left to right: a two-dimensional Euclidean sphere, border of a cube and border of a tetrahedron.

1.2 2-Manifold Classification

The homeomorphisms define equivalence classes among the manifolds. The manifolds in a same class are homeomorphic and they have the same topological properties: same dimension k (Section 1.1), same number of connected components (see Definition 1.4) and same genus (see definition in Theorem 1.1).

Definition 1.4 (Connected Component) *Let $M \subseteq \mathbb{R}^n$ be a k-manifold and $\mathbf{x} \in M$. The connected component of $\mathbf{x}$ in M is the set of points $\mathbf{y} \in M$ such that there exists a path linking $\mathbf{x}$ and $\mathbf{y}$ in M, i.e. there exists a continuous function $f : [0, 1] \to M$ such that $f(0) = \mathbf{x}$ and $f(1) = \mathbf{y}$.*

Here we confound connected and path-connected components since they are equivalent for a manifold [56]. The latter is preferred in the definition since it is more intuitive than the former. We say M is connected if it has a single connected component.

Theorem 1.1 (2-Manifold Classification in $\mathbb{R}^3$) *Let M be a connected and compact 2-manifold in $\mathbb{R}^3$. There is an unique $h \in \mathbb{N}$ such that M is homeomorphic to a 2-sphere with h handle(s). More precisely, M is homeomorphic to*

- *sphere* $\mathbb{S}^2 = \{(x, y, z) \in \mathbb{R}^3, x^2 + y^2 + z^2 = 1\}$ *if* $h = 0$,
- *torus* $\mathbb{T}^2 = \{(x, y, z) \in \mathbb{R}^3, z^2 + (\sqrt{x^2 + y^2} - 1)^2 = 1/9\}$ *if* $h = 1$,
- *a 2-manifold defined by h tori $\mathbb{T}^2$ joined by $h - 1$ tubes as in Fig. 1.2 if $h \geq 2$.*

Integer h is also called the genus of M, or the number of handles of M.

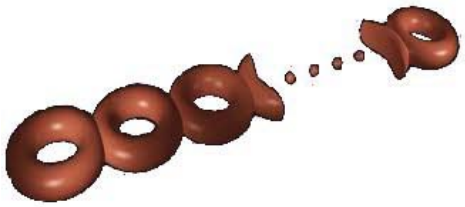


Figure 1.2: Example of h-holed *torus*[1].

The original 2-manifold classification Theorem 4 in p. 161 of [99] is more general: it deals with 2-manifolds which can have a cross-cap as the Klein's bottle embedded in $\mathbb{R}^4$. However, this generality is not useful in our work where the 2-manifolds are embedded in $\mathbb{R}^3$ (see also p. 6 of [39]). Last, we remind that a subset M of $\mathbb{R}^3$ is compact if M is bounded and $\mathbb{R}^3 \setminus M$ is an open set of $\mathbb{R}^3$.

1.3 Simplicial Complex

Curves, surfaces and volumes can be approximated by piecewise linear discretizations defined from a finite number of points. Here we give the definition of a single mathematical tool which can do all these approximations: the simplicial complex [55, 99]. Now we begin by introducing the definition of simplex.

Definition 1.5 (k-simplex) *Let $V = \{\mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_k\}$ be a set of $k + 1$ points in general position in $\mathbb{R}^n$. The k-simplex σ_V is the convex hull of V, i.e.*

$$\sigma_V = \{\sum_{i=0}^{k} \lambda_i \mathbf{v}_i, \lambda_i \in \mathbb{R}^+, \sum_{i=0}^{k} \lambda_i = 1, \}. \tag{1.2}$$

We also note σ_V as $\mathbf{v}_0\mathbf{v}_1 \cdots \mathbf{v}_k$, and k is called the dimension of σ_V.

[1] Image source: http://www.ualberta.ca/dept/math/gauss/fcm/topology/AlgbrcTop/gennsurf.htm

Remind that "V is in general position" means that k is the dimension of the linear space generated by $\mathbf{v}_1 - \mathbf{v}_0, \cdots, \mathbf{v}_k - \mathbf{v}_0$ (which implies that $k \leq n$).

Let V' be a subset of V with k' ($k' < k$) elements. Then simplex $\sigma_{V'}$ is called a k'-face of σ_V. Here, we use the notation $\sigma_{V'} < \sigma_V$ to say that simplex $\sigma_{V'}$ is a k'-face of σ_V.

There are four kinds of simplices in $\mathbb{R}^3$: vertices (0-simplices), edges (1-simplices), triangles (2-simplices) and tetrahedra (3-simplices).

Definition 1.6 (simplicial complex) *A simplicial complex K is a finite set of simplices such that*

- *if $\sigma \in K$ and $\tau < \sigma$, then $\tau \in K$*
- *if $\sigma, \tau \in K$ and $\sigma \cap \tau \neq \emptyset$, then $\sigma \cap \tau$ is a face of both σ and τ.*

The dimension of K is the largest dimension of its simplices.

We say that K' is a subcomplex of K if K' is a simplicial complex such that $K' \subseteq K$. Two simplices of K are incident if they intersect. Two simplices of K are adjacent if they have the same dimension k and their intersection is a simplex of dimension $k-1$.

1.4 Triangulated 2-Manifold

1.4.1 Notions & Definitions

Let K be a simplicial complex. The union of the simplices of K is noted $\{K\}$. $\{K\}$ is called a polyhedron. According to Section 1.1, $\{K\}$ is a topological subspace of $\mathbb{R}^n$ using the topology induced by the topology of $\mathbb{R}^n$. Here we introduce the definition of triangulated k-manifold (p. 5 of [99]) as follows,

Definition 1.7 (Triangulated k-Manifold) *A triangulated k-manifold is a simplicial complex K such that $\{K\}$ is a k-manifold.*

In case of $k = 2$, we can see that a triangulated 2-manifold is a simplicial complex K such that $\{K\}$ is 2-manifold. In this dissertation, "triangulated 2-manifold" is shortened to "2-manifold".

Now we review two methods which check that K is a 2-manifold: a triangle-based test (p. 723 of [58]) and an edge-based test (p. 87 of [55]).

Property 1.1 (Triangle-Based Test) *A two-dimensional simplicial complex is a 2-manifold if and only if every edge is incident with two triangles, and the triangles around a vertex can be ordered as $f_0, \cdots, f_{k-1}$ so that there is exactly one edge incident with both f_i and f_{i+1} (indices modulo k).*

Before presenting the edge-based test, we should remind what is a simple closed polygon and what is the link of a vertex in a simplicial complex.

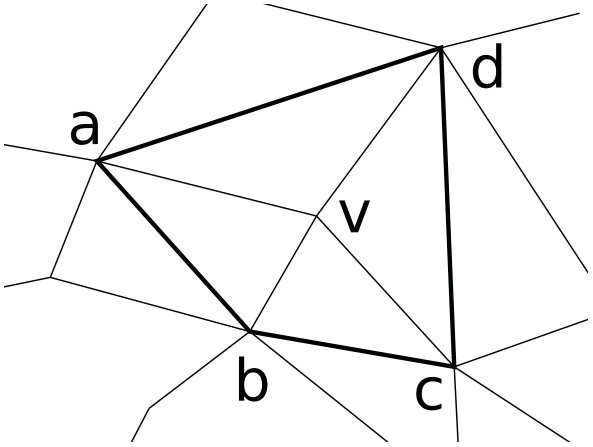


Figure 1.3: Star $St(\mathbf{v})$ and link $L(\mathbf{v})$ of a vertex $\mathbf{v}$ on a 2-manifold simplicial complex. $St(\mathbf{v})$ contains triangles $\mathbf{vab}, \mathbf{vbc}, \mathbf{vcd}, \mathbf{vda}$ and all their faces. Furthermore, $L(\mathbf{v}) = \{\mathbf{ab}, \mathbf{bc}, \mathbf{cd}, \mathbf{da}, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$.

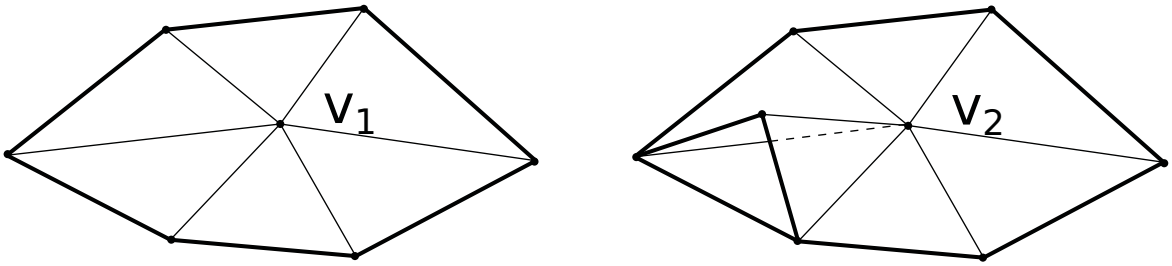


Figure 1.4: A regular and a singular vertex. The link of a vertex is shown in bold. We see that $\mathbf{v}_1$ is regular since its link is a simple closed polygon and $\mathbf{v}_2$ is singular since its link is not a simple closed polygon.

Definition 1.8 (Simple Closed Polygon) *A simple closed polygon is a one-dimensional simplicial complex with k ($k \geq 3$) distinct vertices $\mathbf{v}_1, \mathbf{v}_2, \ldots \mathbf{v}_k$, and the same number of edges which are $\mathbf{v}_1\mathbf{v}_2, \mathbf{v}_2\mathbf{v}_3, \cdots, \mathbf{v}_{k-1}\mathbf{v}_k, \mathbf{v}_k\mathbf{v}_1$.*

Definition 1.9 (Star and Link of a Vertex) *Let $\mathbf{v}$ be a vertex in a simplicial complex K. Let $St(\mathbf{v})$ be the set of simplices of K containing $\mathbf{v}$, and their faces. Then $St(\mathbf{v})$ is the star of $\mathbf{v}$ in K and it is a subcomplex of K. The link $L(\mathbf{v})$ of $\mathbf{v}$ is the set of simplices of $St(\mathbf{v})$ which do not contain $\mathbf{v}$.*

See Fig. 1.3 for an example of the star and link of a vertex in a 2-manifold simplicial complex. Based on the definitions above, the edge-based test can be presented as follows,

Property 1.2 (Edge-Based Test) *A two-dimensional simplicial complex is a 2-manifold if and only if the link of every vertex is a simple closed polygon.*

If the link of a vertex is a simple closed polygon, then this vertex is called a regular vertex. Otherwise, it is called a singular vertex. Fig. 1.4 shows examples of a regular and a

singular vertex.

Once we known that the simplicial complex K is a (connected) 2-manifold in $\mathbb{R}^3$, one can classify K thanks to Theorem 1.1 since K is compact.

1.4.2 Using a triangulated 2-manifold for scene modeling

Estimating a triangulated 2-manifold surface to model scenes is a very popular choice in the image-based modeling methods [23, 70, 76]... More details can be found in Chap. 2.

Advantages exist by enforcing 2-manifold property for an estimated surface which models the scene. Assuming that the true scene surface is a smooth 2-manifold, the continuous differential operators of normal and curvature can be well defined. Now if the estimated surface which approximates the true surface is also a 2-manifold, these operators can be extended to the discrete case [27, 97]. Then the discrete operators can be used to enforce constraints on the computed surface, in surface fairing methods or dense methods which minimize a global cost function involving a regularization term (e.g. [64]). More generally, a lot of Computer Graphic algorithms do not apply if the estimated surface is not a 2-manifold [27]. Here is a simple example shown in Fig. 1.5: the oriented normal of a surface is not well defined for a non-manifold surface. In our work, the manifold property is used to constrain a surface interpolating a sparse point cloud and to improve surface denoising.

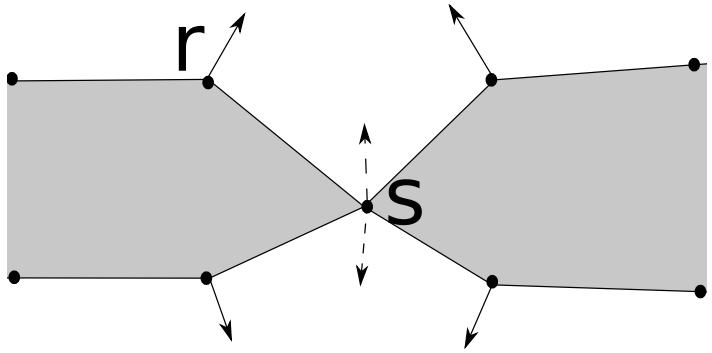


Figure 1.5: Oriented normals of a non-manifold surface in the 2d case. r has a well defined normal but s has not.

1.5 3D Delaunay Triangulation

The 3d Delaunay Triangulation is frequently used in the problem of surface reconstruction from a point cloud [31, 39]. This tool is talked about in this section.

Definition 1.10 (3D Delaunay triangulation) *Let $Q = \{\mathbf{q}_1, \cdots, \mathbf{q}_n\}$ be a set of $n \geq 4$ points in $\mathbb{R}^3$. A 3D Delaunay triangulation of Q is a three-dimensional simplicial complex K such that*

- *Q is the set of vertices of K*

- *the union of the tetrahedra of K is the convex hull of Q*
- *the circumscribing sphere of every tetrahedron of K does not contain any vertex in its interior.*

A 3D Delaunay triangulation always exists for Q. Furthermore, it is unique if Q is in general position, i.e. if Q does not contain 5 cospherical points and does not contain 4 coplanar points.

1.5.1 "Good" Surface Reconstruction

Assume that the set Q of points is a sampling of the (unknown) scene surface S_0. On the one hand, the 3D Delaunay triangulation D is a simplicial complex which, intuitively, stores the nearest neighbors of every vertex in every direction using edges. On the other hand, a surface reconstruction method can try to approximate S_0 by choosing a list S of triangles which connect adjacent point triples in Q. As a result, the idea of searching the surface S as a two-dimensional subcomplex of the 3D Delaunay triangulation comes naturally to mind.

It can be shown that, under assumptions on S_0 and Q, there exists a subcomplex S of D which is a "good" approximation of S_0 [12, 25]. We summarize this fact by the Theorem 1.2 presented in the following paragraph. Details are omitted (definitions of ε-sample, small enough ε, and restricted Delaunay triangulation) since they are not required in our work.

Theorem 1.2 ("Good" Surface Reconstruction) *Let S_0 be a compact and $\mathcal{C}^2$ continuous 2-manifold in $\mathbb{R}^3$. Let Q be an ε-sample of S_0 where $\varepsilon > 0$ (the smaller ε, the higher density of Q in S_0. The Q density is adapted to the S_0 curvature). Let D be the 3D Delaunay triangulation of Q.*

If ε is small enough, there is a simplicial subcomplex S' of D such that

- *S' is homeomorphic to S_0*
- *the Hausdorff distance between S' and S_0 is $\mathcal{O}(\varepsilon^2)$.*

A solution S' can be computed from S_0 as the Delaunay triangulation of Q restricted to S_0.

Note that the approximation is "good" in both topological sense (S' is a triangulated 2-manifold homeomorphic to S_0 [12]) and geometrical sense (the Hausdorff distance between S' and S_0 [25] is small).

1.5.2 Infinite Vertex

A 3D Delaunay triangulation D of point set Q can be seen as the adjacency graph of its tetrahedra:

- a graph vertex is a tetrahedron,
- a graph edge is a triangle between two tetrahedra.

All tetrahedra have 4 neighbors, except a few tetrahedra which have (at least) a triangle on the border δC of the convex hull C of Q. To make easier both design of algorithms and implementation, a standard method [7] completes this graph such that all tetrahedra (without exception) have exactly 4 neighbors.

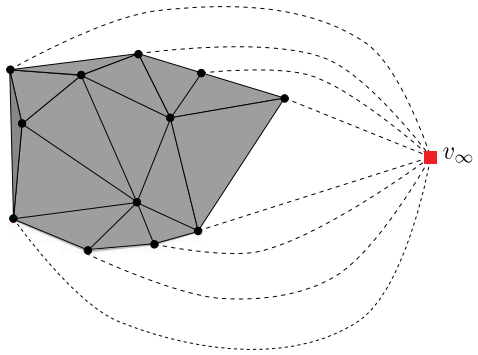


Figure 1.6: Delaunay triangulation D with infinite vertex $\mathbf{v}_\infty$. We have finite vertices of D (round black points), finite edges of D (solid lines), the infinite vertex $\mathbf{v}_\infty$ (squared red point), infinite edges of D (dashed curves), finite cells (gray), infinite cells (white).

The infinite vertex $\mathbf{v}_\infty$ is introduced such that we define a virtual tetrahedron joining every δC triangle and $\mathbf{v}_\infty$. See Fig 1.6 for an example. The virtual tetrahedra (and $\mathbf{v}_\infty$) do not exist in $\mathbb{R}^3$, but they are vertices of the adjacency graph. Then, the adjacency graph becomes a 4-regular graph.

1.5.3 Space and Time Complexities

Here we only describe properties [58] (p. 516) that we need. We use the standard "big o" notation $\mathcal{O}$ and we assume that the number of vertices in a 3d Delaunay triangulation D is n.

First, the number of tetrahedra $|D|$ in D can vary from $\mathcal{O}(n)$ to $\mathcal{O}(n^2)$. In the worst case, $|D|$ is $\mathcal{O}(n^2)$ and its maximum vertex degree d is $\mathcal{O}(n)$ [18]. Here, d is the maximum number of tetrahedra incident to a vertex of D except $\mathbf{v}_\infty$. [36] shows that $|D|$ can be $\mathcal{O}(n \log(n))$ if Delaunay vertices sample a smooth surface under some uniform sampling condition. However our Delaunay vertices are sparse points without uniformity guarantee, thus we prefer to use $\mathcal{O}(n^2)$ as the theoretical complexity of $|D|$.

Second, the 3d Delaunay triangulation can be computed in $\mathcal{O}(n^2)$. In our work, we use the CGAL implementation [7] of the algorithm [38] and it has a complexity of $\mathcal{O}(n^2)$ in the worst case [26]. Besides, the complexity of adding a 3d point in D is also $\mathcal{O}(n^2)$ in the worst case, where all tetrahedra of D are destroyed and D is reconstructed.

Chapter 2

State of the Art of Automatic Image-based 3d Modeling

Automatic modeling of scenes from images is a well studied subject during the recent decades. This chapter surveys the existing methods in the literature. Here, we investigate only methods using passive visual sensors like a camera and objects in scenes are assumed to be non-deformable. Methods using active vision or dealing with deformable scenes are out of the scope of our survey.

One category of automatic image-based 3d modeling methods, e.g. [48, 74, 81, 147], directly model scenes from images by using tools such as voxels, level-sets, meshes or depth maps [124]. Another category of methods firstly estimate a 3d point cloud from images, and then apply a surface reconstruction method, e.g. [82, 149], to model scenes. In our work, we roughly classify the automatic image-based 3d modeling methods in these two categories. Sec. 2.1 presents methods which model directly from images and Sec. 2.2 presents methods which model via a point cloud. Notice that Sec. 2.1 is relatively short compared to Sec. 2.2 since methods in Sec. 2.1 are quite different from ours. Besides, we survey the incremental and real time image-based modeling methods in Sec. 2.3. At last, a conclusion of our survey is given in Sec. 2.4.

2.1 Modeling directly from Images

Many image-based modeling methods, known as multi-view stereo methods, directly estimate a model of scenes from images. In most cases, the input images are calibrated before modeling. It means that not only camera intrinsic parameters but also the camera orientations and locations where images are taken are known. The calibration of camera parameters from multi images is a standard problem in Computer Vision that we do not present here. One might refer to [63] or Chap. 3 for further reading. Now we suppose that all input images are calibrated.

The work [124] gives a survey and classification of multi-view stereo methods. Inspired by this work, we classify the methods which model scenes directly from images in four categories, according to their representation structures: voxels, level sets, meshes and depth maps.

Before going into details, we remark that our classification criterion is on the representation structure used by the final estimated model. One structure might be converted to another one using a suitable algorithm, e.g. the voxels and level sets can be efficiently converted to a mesh using marching-cube algorithm [92]. The classification criterion that we used is a common one, and there are also other criteria [124] such as reconstruction algorithm.

2.1.1 Voxels

Some works [28, 81, 123, 129, 139, 146]... model scenes as a voxel-occupancy function. The majority of these methods [81, 123, 129, 139]... measure a compatibility score of the voxel with input images and decide if this voxel is occupied. This score is called the photo-consistency [81]. These methods are usually known as "space-carving" approaches.

Let us take the work [81] as an example. This work initializes with a volume V (a Cartesian grid) containing the object to model and removes the non-photo-consistent voxels from V. To do so, each voxel is back-projected to visible images and the photo-consistency score of the voxel is computed. This score is calculated based on the deviation of colors of the back-projected pixels and compared to a threshold to decide if the voxel is photo-consistent or not. In practice, voxels of grid are swept in an order such that the occluder is swept before voxels that they occlude.

2.1.2 Level Sets

Some works [48, 72, 91, 115]... model scenes as a level set of a scalar function. The scalar function is sampled on a regular Cartesian grid and the implicit surface which models scenes is located at the set of points where the scalar (interpolated) function values are zero.

Generally, level sets methods are variational approaches. They start at an initial guess of the surface and then the surface evolves (shrinks or expands) to a final surface which is consistent to input images. We take the work [48] as an example. Assume that $C_0 = \{x \in \mathbb{R}^3, f_0(x) = 0\}$ is the initial surface and $C(t) = \{x \in \mathbb{R}^3, f(x,t) = 0\}$ is the deformed surface at time t. Here, f_0 and f are two scalar functions in class C^2 such that $f_0(x) = f(x,0)$. The evolution of surface $C(t)$ is described by partial differential equations of f which can be obtained as follows. A mobile point $x(t)$ in $C(t)$ meets $f(x(t),t) = 0$, which also means $\frac{\partial f}{\partial t} + \nabla f \frac{dx}{dt} = 0$. By noting $n = \frac{\nabla f}{|\nabla f|}$ and $\beta = -n\frac{dx}{dt}$, we have $\frac{\partial f}{\partial t} = \beta|\nabla f|$. Here $\beta(x)$ is the evolution speed of point x in the surface in direction of its normal to the surface. The idea is to choose β in function of a criterion of global matching to minimize. For $x \in C(t)$, the more the neighbors of projections of x in different images are similar, the more $\beta(x)$ should have a magnitude near to zero, i.e. the less the surface must move in x. The criterion of global matching is the integral of a correlation coefficient of the neighborhood of all points x in $C(t)$ between well-chosen image pairs. β is chosen as the component along the normal of $C(t)$ of Euler-Lagrange equations associated in the problem of criterion minimization.

At each instant, f is defined in nodes of a Cartesian grid containing objects to model. Values of f at $t+1$ are defined by values at t thanks to the equation $\frac{\partial f}{\partial t} = \beta|\nabla f|$. Surfaces

defined by level sets are suitable for modeling objects of complex shapes: the surface $C(t)$ can be separated to several connected components or merge without specific treatment.

2.1.3 Meshes

A mesh is an object connecting vertices by edges and polygon faces. It is specially efficient for store and render. Therefore many image-based modeling methods [52, 64, 70, 71, 84, 138, 146]... rely on it to represent the scene.

A lot of these methods [70, 138, 147]... convert a Cartesian grid (which contains objects to model) to a graph, and use volumetric graph cuts techniques to extract a mesh which optimizes global cost functions e.g. photo-consistency of grid voxels. To do so, each voxel of the grid is a node in the graph and each face separating a pair of neighbor voxels is a graph edge. Each graph edge is weighted by a cost function. Then the minimum (cost) cut of the graph, which is a polygon mesh, can be computed as the final model. Notice that this mesh separates all grid voxels into two regions thus these methods can also be seen as voxels based methods presented in Sec. 2.1.1.

Let us take the work [146] as an example to present the graph-cuts framework for image-based modeling. In this work, a base surface is firstly computed. This base surface can be either the visual hull of the object to reconstruct, by intersecting cones of the silhouettes of the scene, or a surface triangulating sparse correspondences in case that the scene is not circumnavigated. Besides, an inner surface is also computed by shrinking the base surface. Now, a Cartesian grid containing both surfaces is used, which is considered as a graph. Voxels between the base surface and the inner surface are projected to images and photo-consistency scores are calculated using normalized cross-correlation of the projections. The weight of an edge in the graph is defined by involving the photo-consistency scores of both graph nodes (voxels) linked by this edge. Finally, by considering the base surface as the source and the inner surface as the sink, the minimum cut of the graph (a mesh) can be obtained. The principal advantage of the graph cuts based methods is that the resulting mesh is a minimal surface for a cost function which combines the global photo-consistency score and the surface smoothness.

Some mesh based methods [51, 52, 64, 71]... are variational approaches like level sets methods in Sec. 2.1.2. They deform an initial mesh to converge to a final mesh by minimizing a cost functional.

Let us take the work [52] as an example. This work combines the multi-view stereo and shape-from-shading techniques to deform a (hexagonally connected) triangulated mesh. It relies on an initial regular mesh obtained by applying an approximative dense stereo algorithm and refines the mesh by moving vertices of the mesh such that a cost functional is minimized by a conjugate gradient descent method. This cost functional is a weighted sum of three terms: a smoothing term over mesh vertices, a multi-image intensity correlation term by taking into account occlusions for each triangle, and a smoothing term of albedo coefficient (ratio of reflected to incident light) for each pair of adjacent triangles. The smoothing term over mesh vertices play a double role. It makes the functional more convex so that the minimization convergence is more rapid, especially at the beginning. Besides, the surface estimation is less sensible to image noise. The terms of multi-image intensity correlation and

albedo are complementary: the former brings accurate information in textured regions of images in contrast to the latter. By using this cost functional, various kinds of information obtained from multiple images are combined and poorly textured objects such as human faces can be successfully modeled.

2.1.4 Depth Maps and Disparity Maps

The depth map is an image on which each pixel has a depth information related to the distance between the surface of the scene and the camera viewpoint. The core problem of depth maps computation is the dense stereo matching, which consists in finding correspondent pixels (pixels correspondent to a same 3d point) in different images. Once a pixel is matched to another pixel in a different image, the disparity, i.e. a 2d vector from the pixel to its correspondent pixel in the image space can be calculated. By doing so for all pixels, a disparity map is obtained and the depth value for each pixel of the disparity map can be subsequently estimated [47], thanks to the calibrated camera parameters.

We briefly summarize some tools frequently used by dense stereo matching. First, an image rectification step is usually applied to a pair of images in order to facilitate the subsequent matching steps, i.e. warping both images such that their epipolar lines are horizontal. Remember that an epipolar line is the intersection between an image plane and a plane which contains both camera centers. Thanks to image rectification, the searching zone of the correspondent point is reduced to a horizontal segment and the 2d disparity vector becomes 1d. Note that there exist some methods which do not need a pre-rectification step, e.g. plane sweep methods [34, 148] and [90].

Now assume that an image pair is rectified and we want to match pixels of an image to pixels of another image. One can measure the similarity of pixels or regions in two images by calculating correlation scores (e.g. Zero Normalized Cross Correlation) or measure the dissimilarity by calculating a matching cost function, e.g. Sum of Squared Difference. Instead of matching a single pixel which is very erroneous, standard approaches match a window of pixels around the pixel in question [141]. However, only using similarity measures is not sufficient [41]. Some matching constraints can be imposed to reduce false positive matches. Here we list the most important ones,

- **disparity limit** Suppose that the distance between camera locations of image pair are small compared to the distance between camera locations and the surface of observed scenes. The disparity of two correspondent points in the image space has a small magnitude compared to the dimension of image.

- **disparity continuity** Disparities of correspondent points should have a smooth variation in the image space (except at depth discontinuities).

- **uniqueness** Each pixel admits at most one correspondent point in another image.

- **ordering constraint** Pixels of an epipolar line and their correspondent pixels in the conjugate epipolar line are in the same order.

- **disparity gradient limit to 1** The magnitude of disparity gradient, i.e. variation of disparity for two neighbor pixels, is limited to 1 if normals of their corresponding 3d points in the real surface of scenes are near to the observation directions of camera.

Many dense stereo matching methods exist in the literature [121]. Some methods [22, 73, 74, 110] calculate a matching cost function for each pixel and estimate the disparity of each pixel by choosing the one which locally minimizes the matching cost. Many other methods [11, 21, 29, 95, 118, 132] formulate the dense matching as a global energy function minimization scheme. Disparities are computed by minimizing an energy function which involves the global matching costs and the matching constraints previously listed. Various minimization tools are used such as PDE (e.g. [11, 132]), graph cuts (e.g. [21, 118]), dynamic programming (e.g. [29, 95]) etc. Besides, a coarse-to-fine multi resolution scheme can be used to improve both processing time and matching quality (e.g. [17, 117] They start matching from the highest (coarsest) level to the lowest (finest) level of a pyramid of images, which are generated from an original image pair.

We note that depth maps are not a "good" final solution for modeling scenes because they are independent, redundant and not suitable for visualization. As a result, many image-based modeling methods, either merge depth maps to estimate an unorganized dense point cloud or directly take depth maps as input, to reconstruct a more sophisticated model (the surface) of scenes. These methods will be further presented in Sec. 2.2.2.

2.2 Modeling via a Point Cloud

Many automatic image-based 3d modeling methods take a 3d point cloud as input to reconstruct a surface fitting the point cloud. The point cloud is reconstructed from images and the reconstructed surface is the estimated model of scenes. We called these methods the surface reconstruction methods. Depending on how the 3d point cloud is computed, these methods can be divided in two classes: the dense methods and the sparse methods. The dense methods use a dense 3d point cloud which corresponds to all or almost all pixels of (calibrated) images. They rely uniquely on the dense point cloud (and also the underlying structures of the point cloud if it is organized), i.e. informations other than the point cloud, e.g. the photo-consistency, are not used. The sparse methods uses a sparse point cloud (or edges in some cases), which corresponds to features detected in images such as corners [62] or edges. In contrast to the dense methods, the sparse surface reconstruction methods do not use only the sparse point cloud but also other informations, notably the visibility information between a 3d point and several camera locations.

2.2.1 How to compute a Point Cloud ?

Now we talk about how to compute a dense and a sparse point cloud from images. To limit the scope of the survey, we briefly introduce them and do not investigate these methods. After this section, we suppose that the point cloud is known and we focus on the survey of surface reconstruction methods.

Dense point cloud generation As we have already mentioned in Sec. 2.1.4, depth maps estimated from images by a dense stereo matching method (e.g. [74]) can be directly used to reconstruct a surface. These depth maps, also called range images, are organized point sets such that by projecting pixels of range images to 3d space, an organized 3d point cloud is formed and connectivities of 3d points are known. Some surface reconstruction methods exploit the underlying structures of the organized point cloud to facilitate the surface reconstruction step. These methods are surveyed in Sec. 2.2.2.2. Note that most of these methods can also use range images obtained using other depth sensors e.g. laser scanner.

An unorganized point cloud can be obtained by registering different depth maps in a same coordinate system (range image registration) then followed by a point cloud simplification. Many surface reconstruction methods rely on an unorganized point cloud to estimate a surface model of scenes. The point cloud is said "unorganized" in a sense that points are completely independent. Only 3d coordinates (and a 3d orientation vector in some cases) of each point are given and connectivities between points are unknown. The range image registration consists in estimating Euclidean transformations (rotations+translations) which map points of all range images in a same coordinate system. The standard approach is the Iterative Closest Point (ICP) algorithm [20]. To register a range image A to another one B, the algorithm iteratively refines the transformation mapping A to B by minimizing the sum of distances between registered points of A to their closest points in B. An unorganized point cloud is obtained after this step, however it is very redundant due to the common points of different range images. Thus a subsequent point cloud simplification step is usually needed to reduce the redundancy [27].

Sparse point cloud generation For sparse point cloud, only feature points such corners in images are reconstructed in 3d. A well known approach to compute the sparse point cloud of an image sequence is the Structure-from-Motion (SfM) [63]. It begins by detecting and matching feature points in images, reconstructs the feature points and also camera parameters, and then refines the reconstruction results (e.g. bundle adjustment). Details about SfM can be found in Chap. 3. Different from the dense methods and also the modeling methods in Sec. 2.1, the SfM can be applied to a non calibrated image sequence, i.e. the camera parameters are not known. And visibility information between reconstructed feature points and cameras is also provided. By using visibility information together with the sparse point cloud (a 3d point is observed in several views), the subsequent surface reconstruction can still reconstruct a surface approximating scenes, in spite of the poor density of the point cloud.

2.2.2 Dense Methods

As already mentioned in Sec. 2.2.1, dense methods can be regrouped in two classes: methods using an unorganized dense point cloud and those using an organized one. Remember that the difference between them is that the latter uses the underlying known structure in the point cloud, i.e. adjacencies between points, to help the surface reconstruction.

2.2.2.1 Methods using an unorganized point cloud

The surface reconstruction methods with an unorganized dense point cloud are firstly presented. According to the output format, some methods reconstruct a parametric surface to model the scene and other methods use an implicit surface. Now to facilitate the presentation, we note Q for the input dense unorganized point cloud, and S for the reconstructed surface.

Parametric surface The majority [12, 14, 15, 23, 32, 40, 45, 145]... of the methods using a parametric surface are based on the Voronoi diagram or its dual, the Delaunay triangulation (see also [31]). These methods rely on the principle that a "good" approximation of the scenes surface is embedded in the 3d Delaunay triangulation of Q, if Q is a ε-sample with ε small enough (see Sec. 1.5.1).

Some Delaunay-based methods [12, 14] exploits the fact that, if Q is sufficiently dense, vertices of Voronoi cells of Q are generally far from the (unknown) surface of scenes in a direction perpendicular to the surface. The Crust algorithm [12] builds a 3d Delaunay triangulation using Q and poles of Q. Here, poles of a point $\mathbf{q} \in Q$ are the two farthest vertices of the Voronoi cell of $\mathbf{q}$. The Delaunay triangles which have at least one pole as vertex are eliminated. Next, a second triangle filtering process is applied to remove irregular triangles (e.g. a triangle is irregular if angles between its normal and pole vectors of triangle vertices are large) and a set of triangles are selected such that each edge is incident to at least two triangles. Finally, a 2-manifold is grown by a breadth-first search over the triangle set. The Crust algorithm is further simplified by Cocone algorithm [14] which directly uses the 3d Delaunay triangulation of Q and selects a set of Delaunay triangles whose dual Voronoi edges intersect cocone zones of points of Q. Here, a cocone zone of a point $\mathbf{q} \in Q$ is the complement of the double cone which has $\mathbf{q}$ as apex and the pole vector of $\mathbf{q}$ as axis. The aperture of a cone is slightly smaller than π, i.e. the cocone zone is a small zone containing the tangent plane of $\mathbf{q}$. At last, a 2-manifold is extracted from the selected triangle set as in Crust [12].

Works [15, 40, 80] consider that the surface of objects can be approximated by using a union of balls (Delaunay circumspheres), called polar balls, centered at poles of points in Q. For example, [15] builds a *power diagram* of polar balls, which is a generalized Voronoi diagram partitioning the space into polyhedral cells. A polyhedral cell for a given polar ball B, contains all points in $\mathbb{R}^3$ such that its power distance to B is smaller than to other polar balls. The surface is reconstructed by selecting faces in the *power diagram* which separate cells of inner polar balls from those of outer polar balls. These faces are recognized using an heuristic [13] such that two polar balls of adjacent cells have different labels (inner or outer) if they do not intersect deeply. Another work [80] uses spectral graph partitioning techniques to help separate Delaunay tetrahedra (via their dual Voronoi vertices) in two sections. The idea is to build a pole graph based on the 3d Delaunay triangulation T of Q by using all poles as graph nodes and each edge $\mathbf{v_i v_j}$ of T is translated to four graph edges: $\mathbf{v_i^+ v_j^+}, \mathbf{v_i^+ v_j^-}, \mathbf{v_i^- v_j^+}, \mathbf{v_i^- v_j^-}$, where $\mathbf{v_i^+}, \mathbf{v_i^-}$ and $\mathbf{v_j^+}, \mathbf{v_j^-}$ are respectively poles of $\mathbf{v_i}, \mathbf{v_j}$. The heuristic of [13] is used to assign weights for graph edges. Each graph edge is weighted positive (resp. negative) if their polar balls (resp. do not) intersect deeply. Then a spectral

graph (bisection) partitioning method is used to cut graph nodes (so their dual Delaunay tetrahedra) into two sections. At last, a second graph is built and cut based on previous results. This graph cut labels remaining tetrahedra whose dual Voronoi vertices are not poles and a surface is extracted from triangles in border of two sections.

Another well known family of Delaunay-based methods are sculpture methods [23, 32, 45, 145]. The first sculpture method [23] is proposed by Boissonnat in 1984. This method constructs firstly a 3d Delaunay triangulation of Q, then sculpts it until all point of Q appear on the surface S while keeping the boundary of the Delaunay be always a polyhedron (2-manifold with genus 0). Here, "sculpture" means that tetrahedra of the 3d Delaunay triangulation are removed one by one in an order based on a priority criterion. The priority criterion of a tetrahedron Δ is the maximum distance between faces of Δ on S and the circumsphere of D. After that, several sculpture approaches similar to [23] are proposed. [145] sculpts the 3d Delaunay triangulation by computing $\gamma-$indicators for boundary Delaunay tetrahedra as the sculpture priority criterion. Let Δ be a tetrahedron such that one of its triangles is on the boundary of Delaunay. Let r be the radius of the circumcircle of the boundary triangle and R be the radius of the circumsphere of Δ. Let $d = 1 - \frac{r}{R}$ (note that $0 < d < 1$). The $\gamma-$indicator for Δ is $-d$ (resp. d) if the circumsphere of Δ is centered inside (resp. outside) the boundary of Delaunay. This method can deal with variable point density however, like [23], this method cannot model objects with multi genus. Further, the work [37] improves [23] by sculpting a connected set of tetrahedra each time instead of only one tetrahedron each time. Consequently it improves the genus of the reconstructed objects.

In 2003, Chaine [32] proposes a geometric convection scheme which translates the surface convection method [150] over the 3d Delaunay triangulation of Q. To do so, the method takes the convex hull of Q as the initial pseudo-surface and shrinks (sculpts tetrahedra) it by always maintaining the Gabriel property of each oriented half-facet, until the pseudo-surface locally fits the input points. Remember that for an oriented half-facet, the Gabriel property is met if the half diametrical ball of the half-facet does not contain any point in Q. Here the term "pseudo-surface" comes from that both oriented half-facets of a same (non-oriented) triangle can exist in the surface. Further, this geometric convection method has been used by [9] which reconstructs the surface from points in a streaming scheme. The work [9] can be seen as an incremental surface reconstruction method that we discuss in Sec. 2.3.

Some other approaches [16, 45] reconstruct an alpha shape of Q and consider the boundary of the alpha shape as the final surface S. A 3d alpha shape or α-shape can be seen as a sub-complex of a 3d Delaunay triangulation. For a given value of alpha, a tetrahedron in the 3d Delaunay triangulation can be labeled as outside or inside of the boundary of the alpha shape depending on the radius of its circumsphere and the value of alpha. In 1994, Edelsbrunner and Mucke give the mathematical definition of alpha shapes in their work [45] and present in details how to obtain alpha shapes based on 3d Delaunay triangulation to reconstruct the surface of scenes.

A minority [19, 59] of parametric surface methods directly establish connectivities of points in Q in forms of triangles. And a triangulated surface is extracted from these triangles. [19] proposes a ball-pivoting algorithm (BPA). Starting from a triangle seed, a size-fixed ball is pivoting around edge of triangles (endpoints of the pivoting edge are on the ball). Each

time a point in Q is reached by the ball, a new triangle is formed by this point together with endpoints of the pivoting edge and the process continues to try other edges. Finally, BPA is tried several times until all points of Q are considered. Another work [59] firstly estimates an oriented tangent plane for each point $\mathbf{q} \in Q$ and then creates a 2d Delaunay triangulation for $\mathbf{q}$ by using projections of neighbors of $\mathbf{q}$ in its tangent plane. At last, a manifold surface S is obtained by stitching a set of (3d) triangles which are obtained such that three input points of Q form a triangle if they are mutual neighbors in the 2d Delaunay triangulations. Notice that both methods presented above require that Q is sufficiently dense and uniform or locally uniform to correctly reconstruct S.

Implicit surface Another category of surface reconstruction methods (using a dense unorganized point cloud) reconstruct surfaces in an implicit form. These methods estimate an implicit function $f : \mathbb{R}^3 \to \mathbb{R}$ to interpolate or approximate points of Q. Each point $\mathbf{q} \in Q$ constrains the function f such that $f(\mathbf{q}) = 0$ (interpolation) or $f(\mathbf{q}) \simeq 0$ (approximation). And the function f is usually the sum of some base functions (global approaches) or a list of local functions (local approaches).

Some methods [10, 24, 69] define the implicit function as a signed distance function. Let us take the work [69] proposed by Hoppe et al. as an example. This method estimates the signed distance for a point (grid node) in 3d as the distance between this point and the local oriented tangent plane of the nearest point in Q. The tangent plane of each point $\mathbf{q}$ in Q is locally computed by using the k-nearest points to $\mathbf{q}$ in Q. Then consistent orientations of the tangent planes are obtained using minimum spanning tree optimization techniques such that neighbor points of Q have similar orientations.

Some other methods [30, 101, 108, 120, 143] use radial basis functions (RBFs) to fit the point cloud. A radial basis function is a real function ϕ whose value depends uniquely on the distance from a point $\mathbf{c}$, called the center, i.e. $\phi(\mathbf{x}, \mathbf{c}) = \phi(|\mathbf{x} - \mathbf{c}|)$ ($|\cdot|$ is the Euclidean norm in $\mathbb{R}^3$). The implicit function f to estimate is the sum of some weighted RBFs and each point $\mathbf{q}$ in Q gives a constraint $f(\mathbf{q}) = 0$. Here, centers of RBFs are usually (but not necessarily) points in Q, and the RBFs weights need to be computed. Global fitting methods [120, 143] translate constraints provided by points of Q in a linear equation system (LES) and weights of RBFs can then be resolved. To avoid the trivial solution (all weights are zero), work [30] proposes to add some off-surface constraint points in LES whose f values are non-zero. Those points can be generated along the normals of points in Q. Besides, the solution matrix of global RBFs methods is dense and ill-conditioned if all points in Q are centers of RBFs. [30] iteratively reduces the number of RBF centers and uses a fast Multipole method to accelerate the RBFs evaluation. There exist also local fitting methods [101, 108] using RBFs. These methods use compactly supported RBFs models (a RBF is locally supported in a neighborhood of the center) to locally fit the points in Q. The solution matrix is much sparser but the difficulty of these methods is to choose suitable support size for RBFs in case of non-uniform point cloud.

Some local fitting methods [109, 137] are based on the partition of the space containing points of Q, which are known as the partition of unity methods. These methods partition the space containing Q in subspaces where a local function is estimated using points (in

Q) of each subspace. In the work [109], an adaptive octree grid is used to partition input points Q in a set of cells based on the local density of Q. Normals of Q are pre-estimated, and a quadratic or piecewise quadratic polynomial is adaptively chosen (depending on local curvatures) and fitted to points of each cell. Finally, a global implicit function is obtained by blending polynomial of each cell. This method can deal with large point sets thanks to the adaptive subdivision of Q.

Another category of local fitting approaches [8, 79, 85, 111, 126] are Moving Least Squares (MLS) methods. We take the work [85] as an example, which defines the surface using a projection operator. For each point $\mathbf{r}$ in or near to the real (unknown) surface S, the method firstly estimates a reference plane which locally approximating S. Then a polynomial locally approximating S is initialized over the reference plane. Next, points in Q are projected on the polynomial via a projection operator and least squared errors between points in Q and their projections in the polynomial are iteratively minimized. Finally, the local surface of point $\mathbf{r}$ is defined as the set of stationary points of the final projection operator. The MLS methods naturally handle uniform noise and generate smooth surfaces, but are sensible to outliers.

Another choice of implicit function is the indicator function used by methods [75, 76], which has value one inside objects and zero outside objects. Let us take the work [76] as an example. This method, known as the Poisson surface reconstruction, firstly computes a vector field $\overrightarrow{V}$ from the input oriented point cloud Q by smoothing (applying a Gaussian filter to) normals of Q. And then the goal is to determine χ such that gradient of χ best fits $\overrightarrow{V}$, which is formulated as resolving a Poisson equation: $\Delta\chi = \nabla \cdot \overrightarrow{V}$, where Δ is the Laplace operator and $\nabla\cdot$ is the divergence operator. To do so, the sum of some weighted compactly supported base functions (centered at points of Q) is used to represent χ. The weights are globally resolved by minimizing the sum of squared errors between projections of $\Delta\chi$ and $\nabla \cdot \overrightarrow{V}$ on each base function. This method is robust and provides excellent results however, like many other implicit methods [30, 69, 109], normals of Q should be provided or computed.

Level-set methods (Sec. 2.1.2) can also be applied to fit a point cloud. Here, remember that the idea of a level-sets method is to deform an initial surface by minimizing an energy functional defined by an integral over the surface. The work [150] proposes to minimize the functional: $E(S) = (\int_S d^p(\mathbf{x})ds)^{\frac{1}{p}}, 1 \leq p \leq \infty$. Here, $d(\mathbf{x})$ is the distance between a point $\mathbf{x}$ in the surface to the nearest point in Q. This function can be seen as a surface area weighted by the distance between the surface and Q, and it is minimized by using a gradient descent method. The evolution speed of a point $\mathbf{x} \in S$ depends on $d(\mathbf{x})$ and $\nabla d(\mathbf{x})$. In this method, in order to efficiently perform the evolution process, an initial surface roughly fitting Q is used as input of the level sets process. This initial surface is computed in a convection scheme by iteratively refining an arbitrary surface (e.g. a bounding box) along the direction of $-\nabla d$ of points in the surface.

2.2.2.2 Methods using an organized point cloud

Some surface reconstruction methods [33, 35, 53, 66, 67, 113, 131, 142, 149] directly estimate a surface using range images (organized point sets) to model scenes. Each range image defines a partial surface and the goal of these methods is to merge or fuse the partial surfaces into a global surface. Thus these methods are also called "fusion methods". The underlying connectivities between points in a range image can be used to facilitate the surface reconstruction step. For example, a range image can be easily triangulated to produce a triangulated mesh as follows. Four adjacent points on a range image are joined to form 0, 1 or 2 triangles and for each point, it has at most eight adjacent triangles. If the difference of depths between two adjacent points is too large (superior to a threshold), then these two points will not be connected.

Some methods [33, 113, 131, 142] generate triangulated meshes from range images and directly merge meshes of all range images. A representative work is the Mesh Zippering [142] proposed by Turk and Levoy. In this work, a variant of ICP algorithm is firstly used to register range images one after another and for each range image, a triangulated mesh is generated as described in the previous paragraph and zippered to a global one. To zipper two meshes, redundant triangles (a triangle is redundant if its vertices lie within a tolerance distance to another mesh) of both meshes are firstly removed. Now if meshes are intersected, then vertices on boundaries of both meshes which have intersected triangles are re-triangulated. If not, the boundary of a mesh will be thickened to connect to the other mesh. At last, small triangles due to the meshes zippering, together with their vertices, are removed and the resulting holes are filled by using constrained triangulation.

Other methods [35, 53, 66, 67, 149], instead of merging triangulated meshes of range images, represent an implicit function over a Cartesian grid to fit depth maps and extract finally an isosurface. Let us take the work [35] as an example. This work defines a weighted signed distance function over a Cartesian grid. The function sums up for each grid voxel, its weighted signed distance to each range surface, which is measured along the line of sight to the sensor. Here, a range surface is the triangulated mesh of a range image. And the weight depends on the uncertainty of the intersection between the line to sensor and the range surface. A point in boundaries of a range surface should have a high uncertainty thus a small weight. Finally, an isosurface can be extracted at the zero level set of the signed distance function.

2.2.3 Sparse Methods

Now we present the image-based modeling methods which reconstruct a surface model of the scene by using sparse feature points (or edges in some cases) estimated from images. The sparsity comes from the fact that only interest points are detected and matched in images such as Harris [62], which have uneven distributions and low densities in images. In our sparse case, about 1 pixel over 200-300 is reconstructed by using SfM methods [87, 103].

Generally speaking, surface reconstruction using sparse data is more difficult than using dense data due to the poor density of points. Sparse methods are also less popular and are a minority in the bibliography. However there are several particular advantages by modeling

scenes using sparse features.

Now the motivation of developing sparse methods is talked about in Sec. 2.2.3.1. Then by examining the data structures encoding the potential adjacencies between points, sparse methods are classified in two classes: works using a 3d Delaunay triangulation which are presented in Sec. 2.2.3.2; and works using several 2d Delaunay triangulations (one per image) which are presented in Sec. 2.2.3.3.

2.2.3.1 Why a sparse method ?

There are several advantages to model scenes by using sparse features.

First, in a computational viewpoint, building a 3d surface using sparse features is obviously more efficient than that from a dense point cloud. Thus it is especially interesting for developing on line surface reconstruction applications of large-scale scenes, e.g. it is suitable for reconstructing compact models for cities. Second, the surface reconstructed from sparse features could be a good initial surface for dense surface reconstruction methods [105, 114, 146] etc. Third, there are some scenes where texture is so poor that only a sparse (not dense) point cloud can be reliably reconstructed. Last, the accuracy of a point in SfM cloud is expected to be better than that of a point in dense stereo cloud, thanks to the SfM machinery [63] involving bundle adjustment.

One might directly apply some dense methods to sparse features, e.g. the sculpture methods [23, 37, 145]. However these works are different from the methods specially designed for sparse features. Sparse methods using a SfM point cloud have an additional visibility knowledge: several viewpoints which reconstruct each 3d point by SfM. And all sparse methods reconstruct a surface which is consistent to these visibility constraints in a sense that, every line segment linking the 3d point and one of its viewpoints does not intersect the surface. Such a line segment is called a "ray". Some sparse methods [49, 93, 112] essentially use the rays to sculpt an initial volume to build a visibility consistent scene representation. Thanks to this additional knowledge, a surface can be reconstructed with a smaller number of points than dense methods investigated in Sec. 2.2.2.

2.2.3.2 3d Delaunay-based methods

Works [49, 82, 93, 112] use 3d Delaunay triangulation as the geometry structure for the surface reconstruction. Here, [93] is an incremental approach that we will talk about in Sec. 2.3.

The work [49] of Faugeras et al. starts by constructing a constrained 3d Delaunay triangulation with sparse feature edges. Then the objective of the method is to label tetrahedra either *free-space* or *matter*, and the reconstructed surface is the collection of triangles separating *free-space* and *matter* tetrahedra. To do so, it firstly marks tetrahedra which are intersected with visibility rays as *free-space*. Next, it applies several times a region-grown algorithm [23] in the rest (non-marked) tetrahedra such that the border of each grow region is a 2-manifold, until all tetrahedra are considered. Tetrahedra inside each region have the same label which should be decided either *free-space* or *matter*. A heuristic can be used to do so: if a region has a large number of vertices then its tetrahedra are labeled *matter*,

otherwise they are labeled *free-space*. This work guarantees the reconstructed surface to be a list of 2-manifolds, however limitations exist. This method is experimented with only very simple data sets and the reconstructed surface is a set of genus-0 2-manifolds. Thus it is not suitable to model large-scale scenes. A simple object like a torus may not be topologically correctly reconstructed. Besides, the use of the heuristic described above may provide false decisions.

In 2007, Labatut et al. [82] introduce a 3d Delaunay triangulation based surface reconstruction method. Firstly, a point cloud is computed by matching a great number of interest points. Next the 3d Delaunay triangulation is built after removing redundant points. At last, a surface is obtained by using graph cuts techniques which minimize an energy function involving the photo-consistency of Delaunay triangles, the visibility of sparse feature points and the surface smoothness (area of the surface). Results of this work show that the reconstructed surface is a good approximation of the scene. However compared to classical sparse point clouds obtained by SfM methods, the point cloud used for this work is denser but less accurate. Besides, the resulting surface is not guaranteed to be 2-manifold.

In 2009, Pan et al. implement a probabilistic on-line 3d surface reconstruction pipeline [112], which builds a textured 3d model of an object with the help of human interaction. During the on-line modeling, user should turn the object in order that all parts of object could be captured by camera. The reconstruction pipeline begins by tracking 2d feature points in images and reconstructs them in 3d whenever there is a sufficient rotation. Next, a 3d Delaunay triangulation is built using the reconstructed 3d points and also the previous reconstructed ones. Then tetrahedra which are not consistent with the visibility rays of feature points are carved away by using a probabilistic approach. To do so, for each triangle and the visibility rays which intersect this triangle, a confidence score is calculated to decide if this triangle is consistent to these visibility rays. The score is low if the intersected points of the triangle with the visibility rays are close to 3d points. Tetrahedra which have one or more than one inconsistent triangle are carved away and the final surface is then the boundary of the rest tetrahedra. Results show that a simple surface model can be efficiently generated by this method. However, this method is an on-line application while it is not incremental: the surface is completely recomputed for each key-frame. Thus the processing time will increase rapidly when the image sequence becomes longer thus this method is adequate for small object acquisition but not for large-scale scenes. Besides, the reconstructed surface is not guaranteed to be a 2-manifold.

2.2.3.3 2d Delaunay-based methods

Other sparse methods [100, 119, 136] build 2d Delaunay triangulation(s) in 2d image(s) and back-project them to 3d in order to generate 2.5d surfaces.

In 2000, Morris and Kanade present an image-consistent surface reconstruction method [100]. Given a set of images, this method begins by computing an initial 2d Delaunay triangulation of a reference image and back-projects it to 3d to generate a 2.5d triangulation. Then this triangulation is progressively refined to be consistent to other images by edge flipping. And the image consistency of the triangulation is measured by the sum for all triangles of the luminance covariance in the re-projected images. The reconstructed surface

is a 2-manifold (genus 0) which is image-consistent. However whenever changing an edge, a predicted image should be processed and a likelihood score should be calculated to check if the novel triangulation is better. The method is experimented using only simple data sets. Besides, the resulting 2-manifold has genus 0.

In [136], a feature based stereo algorithm is applied to clusters of images, and edges are reconstructed. These edges are inserted in the constrained 2d triangulations, then the back-projected 2.5d triangulations are merged by computing the union of the free-space defined by the 2.5d triangulations. The resulting implicit surface is converted to 2-manifold mesh by the marching cube method, which requires a regular subdivision of space. The reconstructed surface is a visibility consistent 2-manifold. However, compared to other sparse methods, a limitation of this method is: it uses a regular subdivision of space which is not suitable for large-scale scenes, as mentioned in [82]. In fact, the subdivision should be sufficiently dense to not degrade the quality of the reconstructed surface,

In [119], another 2d Delaunay triangulation based surface reconstruction method is proposed. The input is a set of reconstructed 3d points and their 2d tracks in calibrated images. After a pre-processing (merging, filtering and smoothing) of the input 3d point cloud, this work detects contours in the 2d images and selects 3d points whose projections in 2d images are near to contours. These points are called ctracks in the work, and edges (in contours) connecting ctracks are inserted to a constrained 2d Delaunay triangulation which is based on ctracks and projections of other 3d points. By doing so on all images, a soup of 3d triangles is obtained. It is further filtered by taking into account the visibility rays of input 3d points, normals and sizes of 3d triangles. And at last, a triangulated mesh is extracted from the triangle soup by using a Delaunay refinement meshing process [25] (it is briefly introduced at the end of this paragraph). This approach differs from other sparse method in the sense that the output surface is an approximation (not interpolation) of sparse feature points. Now we introduce the Delaunay refinement meshing process used at the end of [119]. Given an initial surface S (triangle soup in case of [119]), this process performs firstly an initial sampling of S, then it progressively enriches the samplings and finally constructs a Delaunay triangulation restricted to S. The obtained restricted Delaunay triangulation is a triangulated mesh which is close to S and provides good approximations of normals, areas and curvatures of S. It is a 2-manifold if the sampling is sufficiently dense, i.e. ε-samples with a sufficiently small ε. The meshing process relies only on an oracle that, given a segment, testing if the segment intersects the surface (triangle soup in the case of [119]) and if so, calculating the intersection points. This feature makes [25] applicable to mesh generation from a triangle soup or an implicit surface (e.g. [10]).

2.3 (Real time) Incremental Modeling Methods

More and more computer vision applications need and achieve real time performance, thanks to the well developed parallel calculating technologies. Recently, significant advances [9, 65, 93, 105, 106, 114, 134] have been made on real time or incremental image-based modeling. Almost all of these methods (except [114]) are adequate for small scenes but not for large-scale scenes. Regarding the modeling process, works [65, 93] presented in Sec. 2.3.1 use firstly

SfM algorithms to reconstruct the sparse geometry of scenes, and then directly reconstruct surface models of scenes. So they can be considered as sparse methods. Works [9, 105, 114] presented in Sec. 2.3.2 estimate 3d models of scenes using dense points. Thus these methods can be considered as dense methods.

2.3.1 Sparse Methods

In 2005, Hilton [65] describes a 2d Delaunay based incremental surface reconstruction method interpolating sparse feature points and edges (extension of the work [96]). In each view, the method builds a constrained 2d Delaunay triangulation of sparse features on the image and back-projects it to the space and merges them to update a mesh. The mesh should be maintained to be consistent to visibility constraints of feature points during the whole process. Up to our knowledge, this is the first method which computes incrementally 3d models using sparse SfM features. However, the reconstructed surface is non-manifold which has holes and self-intersections. Besides, experiments are done with very short sequences.

In 2010, Lovi et al. propose an incremental 3d modeling method [93] based on space-carving techniques. This method builds incrementally a 3d Delaunay triangulation of sparse points obtained from SfM, then several heuristics are developed to update the triangulation, including key frame addition, outlier deletion etc. Furthermore, it updates also incrementally the carved/un-carved information for each tetrahedron of the triangulation. This method is integrated with a SfM method PTAM [77] to realize a real time system which builds a 3d model from video. But the reconstructed surface is not 2-manifold. In addition, several heuristics are used to handle the instability of points due to the standard refinement step of SfM algorithms. However the unstable points are usually erroneous or even outlier, and it might be more interesting and easier to only use the stable points, which are more accurate and reliable.

2.3.2 Dense Methods

[9] proposes a streaming surface reconstruction scheme using geometric convection method [32]. The input is a set of dense points organized in slices along an axis and the surface is incrementally reconstructed by slice of input points along the axis. The idea of this method is to incrementally construct the Delaunay triangulation of points slice by slice, and guarantees that the Delaunay tetrahedra D_i of each slice S_i are not in conflict with any point of another slice S_{i+3} by splitting large tetrahedra (adding extra points in Delaunay). By doing so, D_i of slice S_i and also the surface which is subsequently generated from D_i remain stable when further slices $S_{i+4}, S_{i+5}, ...$ are added in Delaunay. Thus D_i (except tetrahedra which have triangles in the surface) can be destroyed and the algorithm continues to deal with new slice S_{i+1} and so on. Here, to generate the surface from D_i, the algorithm applies iteratively the geometric convection which shrinks from outside or from inside of the surface through several levels like a onion peeling process. This method achieves the surface reconstruction in a streaming framework. It is able to deal with large amount of data (several millions of points) without large memory consumption. However, the input points should be pre-estimated and organized in a set of slices along an axis.

A reconstruction system based on a costly hardware mounted on a car was also developed [114]. It involves several perspective cameras pointing in several directions, accurate GPS/INS (global positioning system + inertial navigation system). The approach is briefly summarized as follows. Firstly, successive poses are estimated using Kalman fusion of visual reconstruction, INS and GPS. Secondly, interest points are detected and tracked in the images; then a sparse cloud of 3d points is obtained. Third, this cloud is used to select plane normals in 3d, which are used to drive a denser reconstruction. Fourth, the obtained 3d depth maps are merged by blocks of consecutive images. Last, a list of triangles which approximate the dense 3d information is generated. This approach is incremental and real-time, it allows reconstruction of very long video sequences. However, the surface is not 2-manifold (the triangles are not connected). At first glance, this problem could be corrected by using a merging method such as [35] followed by a marching cube [92]. However it is not adequate to large-scale scene since it requires a regular subdivision of space into voxels. For this reason (and other reasons mentioned in [82]), an irregular subdivision of space into tetrahedra is more interesting for large-scale scene.

In 2010, Newcombe and Davison [105] propose a live dense 3d modeling method. This approach uses a Radial Basis Functions (RBFs) based method (similar to RBFs methods described in Sec. 2.2.2.1) to continuously fit sparse points provided by a real time SfM method. A mesh, called the base mesh, is automatically updated by polygonizing the implicit surface of RBFs. Another process parallel to the base mesh generation, uses the base mesh and input images to generate a dense surface. This process selects bundles of consecutive cameras with a partial overlapped visible surface and each bundle has a reference camera and several neighbor cameras. A dense depth map is estimated for the reference image, by back-projecting all pixels of reference image to the base mesh and iteratively deforming the base mesh into a photo-consistent dense model. Here, the displacement of a sampling in the base mesh is calculated based on differences of the pixel in the reference image and its matched pixels in neighbor images. The dense matching of two images is achieved by projecting textured base mesh to cameras of the bundle to produce synthetic predictive images, and applying a variational optical flow method [149] between synthetic and true images. A local model is then obtained by triangulating the depth map, and finally fused to the global surface model. This method provides high quality reconstructed surface and a global surface model can be incrementally generated. However the reconstructed surface may contain holes due to the occlusion and the method is not suitable for large-scale scenes. Besides, the processing time of each local model computation and fusion is long. According to the on line video provided by authors, it takes about 1.5s using two GPUs.

2.4 Conclusion

In this chapter, we have surveyed the image-based automatic 3d modeling methods in the bibliography and roughly classified them in two categories. The first category contains the Multi-view stereo methods which directly estimate models of scenes from calibrated images. The second category contains the surface reconstruction methods which reconstruct a surface by using a pre-estimated point cloud, either dense or sparse, depending on that if all pixels

or only feature pixels of images are reconstructed. At last, we have also investigated the real time or incremental approaches.

In our work, two methods are developed to automatically model scenes: a batch image-based modeling method and its incremental version. Our methods reconstruct firstly sparse points from an image sequence using SfM, and reconstruct surface models of scenes. The surface reconstruction is a 3d Delaunay triangulation-based sculpture process inspired by the work [23]. And the reconstructed surface is a 2-manifold with multi genus.

Compared to most of the Multi-view stereo methods (Sec. 2.1), our methods use sparse feature points thus the processing time is lower and the estimated models are more compact. Besides, most of the dense surface reconstruction methods (Sec. 2.2.2) are designed to deal with points acquired by range scanners, which provide usually a "good"-quality point cloud. A lot of these works rely on a dense, nearly uniform, and noise/outlier free point cloud. Thus these methods are not widely used in the Computer Vision community, where it is difficult to obtain such a point cloud. Now if we compare our methods to other sparse approaches, our methods have several advantages which can be described as follows. First, our methods generate multi genus 2-manifold. The manifold property has a lot of advantages in post-processings (see Sec. 1.4.2), however it is ignored by many sparse works. Only [49, 100, 136] generate (one or several genus-0) 2-manifolds, but these works are not suitable for large-scale scenes, e.g. a camera moving around a building. Second, up to our knowledge, our incremental method is the first incremental or real time image-based modeling method which provides a 2-manifold from sparse SfM points only. And we deal with large-scale scenario such as a camera mounted on vehicle/robot/human exploring an unknown and large scene. This is different to the scenario of a camera moving in a limited workspace such as desk-like/indoor scenes [105].

Chapter 3

Pre-processing: Geometry Reconstruction

Given a sequence of 2d images taken by a moving camera in an environment, the pre-processing step of our 3d surface reconstruction methods is to reconstruct the geometry of the environment. The catadioptric camera is used thanks to its wide field of view, which is strongly required for an efficient acquisition and reconstruction of a complete environment (not a small object). The geometry reconstruction consists in estimating parameters of the camera which has taken the 2d images, a sparse 3d feature point cloud which describes well the geometry of the scene and visibility information between the points and the camera. This problem of geometry reconstruction with a moving camera is also called Structure-from-Motion (SfM) in the Computer Vision field. Here, the purpose is not to do an exhaustive survey of all available methods. We shortly present the tools that we use and basic principles of SfM. A state of the art of SfM can be found in [104].

Generally, SfM methods begin by detecting and matching feature points in images, e.g. corners. Then 3d coordinates of matched features points and camera parameters are calculated. Finally, a refining step, e.g. bundle adjustment, is usually applied to these 3d points and camera parameters in order to minimize the reprojection errors.

Now we show firstly the principles and notions of each step of SfM in Sec. 3.1. Then, we present our batch SfM method [87] in Sec. 3.2 and our incremental SfM method [104] in Sec. 3.3. The "batch" SfM means that the whole image sequence of the scene is already known before the geometry reconstruction process and the geometry of the scene is reconstructed all at once. Regarding the incremental SfM, the 2d matched interest points are progressively available, and the geometry of the scene is progressively updated each time new 2d matched points are available.

3.1 Principles and Notions

3.1.1 Camera Model

A digital camera is an optical system which focuses light on an image pickup device and records images to acquire visual information of environments. The 2d image formation is a projection process such that a 3d point in the space is mapped to a pixel on a 2d map of the camera. A camera model mathematically describes this projection process.

3.1.1.1 Perspective Camera Model

The most common camera model is the perspective or pinhole camera model. It describes the projection of a 3d point on to the image plane of the camera without taking into account the distortions caused by lenses or discreteness of the signal. The principle of a perspective camera model can be shown in Fig. 3.1. There is an optical center $\mathbf{t}$, an image plane I of the camera and an optical axis which goes through $\mathbf{t}$ and is perpendicular to I. It intersects the image plane on a point $\mathbf{t}'$ which is called the principal point. A 3d point $\mathbf{q}$ is projected on I in such a way that there is a ray (line segment) which originates from $\mathbf{q}$ and goes through $\mathbf{t}$, and this ray intersects with I on $\mathbf{p}$.

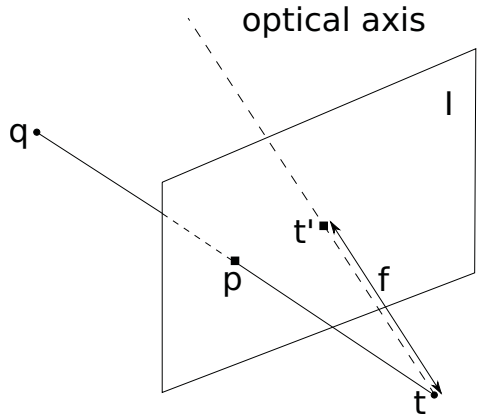


Figure 3.1: Perspective camera model. $\mathbf{q}$ is a 3d point, $\mathbf{p}$ is the projection of $\mathbf{q}$ on image plane, f is the focal length, $\mathbf{t}$ is the center and $\mathbf{t}'$ is the principal point.

Assume that there is a known world coordinate system, a 2d image coordinate system on the image plane and a camera coordinate system which has the origin at point $\mathbf{t}$. Besides, the image x,y axis and the optical axis are respectively the x,y and z axis of the camera coordinate system. The homogeneous coordinates of the 3d point $\mathbf{q}$ expressed in the camera coordinate system is $\mathbf{q}_c = (X_c\ Y_c\ Z_c\ 1)^T$. The homogeneous coordinates of the projected point $\mathbf{p}$ expressed in the image coordinate system (in pixels) is $\mathbf{p} = (x\ y\ 1)^T$. We have,

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{3.1}$$

Here "≡" means equals up to a (non-zero) real scale. f_x and f_y are respectively the focal length measured in width and height of the pixels. c_x, c_y are coordinates of the principal point $\mathbf{t}'$. s is a factor accounting for the skew due to non-rectangular pixels which is almost zero for most cameras. They are parameters specific to the camera which do not depend on camera motion, so we call them intrinsic camera parameters. And the process of calculating intrinsic parameters is also called the camera calibration. If we use K to name the intrinsic parameter matrix, Eq. 3.1 is,

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv K \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{3.2}$$

Note that $(X_c, Y_c, Z_c)^T$ is a non-normalized direction of the ray which originates from the optical center and goes through the 3d point $\mathbf{q}$. It can be calculated up to scale by simply inversing K, if $\mathbf{p}$ is known

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \equiv K^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.3}$$

Then the normalized ray direction, denoted $\mathbf{d}$, is $\frac{1}{\sqrt{X_c^2+Y_c^2+Z_c^2}}(X_c\ Y_c\ Z_c)^T$.

We can see that the projection pixel of a 3d point on an image can be calculated if camera intrinsic parameters are known. However from a pixel of an image, one can only determinate the ray which goes through the 3d point.

Let the homogeneous world coordinates of $\mathbf{q}$ be $\mathbf{q}_w = (X_w\ Y_w\ Z_w\ 1)^T$. Assume that the transformation of world coordinates to camera coordinates, i.e. the camera pose $(R,\ t)$ is known. Then we have,

$$\mathbf{p} = KR^T[I_3 \mid -t]\mathbf{q}_w \tag{3.4}$$

The camera pose is also called extrinsic camera parameters and the process of estimating the camera pose $(R,\ t)$ is called the camera pose estimation.

3.1.1.2 Catadioptric Camera

In our work, we use a catadioptric camera which is an omni-directional camera composed by a perspective camera and a convex mirror in front of the camera as shown in Fig. 3.2. Our catadioptric camera has a wide field of view. It has 360 degrees in the horizontal direction and approximately 110 degrees in the vertical direction (in a half-plane): about 50° above and 60° below the horizontal plane.

Several camera models for the catadioptric camera exist, e.g. [54, 57, 87, 98]. Describing all these models is outside the scope of this dissertation. Here we present an "exact" non-central model and an approximative central model introduced by [87]. The latter is selected as the camera model used for our 3d geometry reconstruction methods. We give comparisons of both models and explain why the latter is selected.

Non Central Catadioptric Camera Assume that there is a known world coordinate system and a mirror coordinate system which has the origin at the mirror apex. The convex

Figure 3.2: A catadioptric camera and an image taken by this camera.

mirror has a symmetry axis which is the z-axis of the mirror coordinate system. The non central catadioptric camera model is defined by three parts:

1) the rotation matrix R and the location $\mathbf{t} \in \mathbb{R}^3$ of mirror coordinate system expressed in world coordinate system,

2) the perspective camera intrinsic parameters K_p, the rotation matrix R_p and location $\mathbf{t}_p$ of the perspective camera both expressed in the mirror coordinate system,

3) and the known mirror profile.

Let $\mathbf{X}$ be homogeneous world coordinates of a 3d point, then its 3d coordinate in mirror coordinate system is $R^T(\pi_3(\mathbf{X})-\mathbf{t})$. Here π_3 is a function such that $\pi_3((x\ y\ z\ t)^T) = (\frac{x}{t}\ \frac{y}{t}\ \frac{z}{t})^T$.

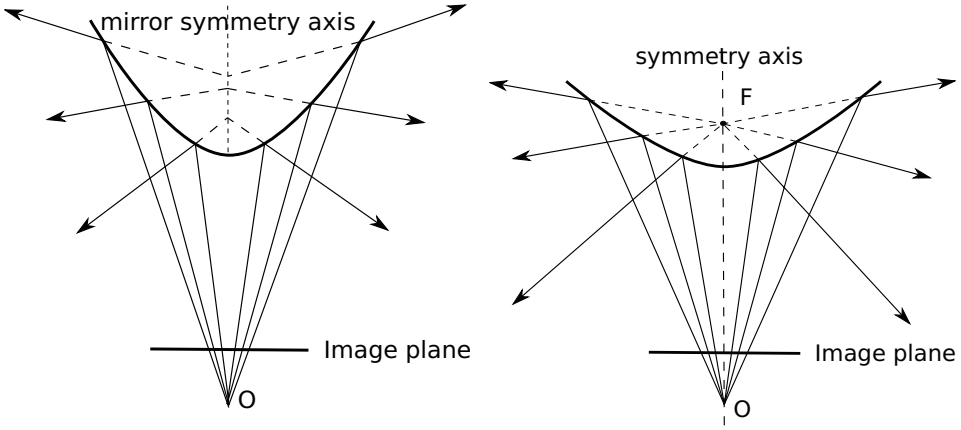
We assume that there is one and only one ray passing through the 3d point $\mathbf{X}$, reflects on the mirror and then passes finally through the perspective camera projection center $\mathbf{t}_p$. Then the reflection point can be calculated by the reflection law if the mirror profile is known. Details about how to calculate the reflection point from the mirror profile are shown in [87]. Here we assume that the mirror profile is known and the function $M(\mathbf{a}, \mathbf{b})$ calculates the reflection point for $\mathbf{a}, \mathbf{b}$ where $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$. Here, $\mathbf{a}, \mathbf{b}$, and $M(\mathbf{a}, \mathbf{b})$ are all in the mirror coordinate system.

We can express the projected pixel $p_{nc}(\mathbf{X})$ as follows,

$$p_{nc}(\mathbf{X}) = \pi_2(K_p R_p^T(M(\mathbf{t_p}, R^T(\pi_3(\mathbf{X}) - \mathbf{t})) - \mathbf{t}_p)) \tag{3.5}$$

where the function π_2 is defined as $\pi_2((x\ y\ t)^T) = (\frac{x}{t}\ \frac{y}{t})^T$

Central Catadioptric Camera Different from non central model, our central model makes an approximation which assumes that all extended rays reflecting on the mirror go through a common point F, called the center. See Fig. 3.3 for schema of both models. We



(a) Non central **(b)** Central

Figure 3.3: Non central and central catadioptric camera models

suppose also that the central model has a symmetry axis. The upper border of the mirror and the border of the mirror support are projected on the image to form two concentric circles (see Fig. 3.4b). Note that the mirror profile is not necessarily a conic. It makes this model more general than other works such as the one of [54] which assumes the mirror profile to be conic and the center F to be one of the conic foci.

Now suppose that there is a known world coordinate system and a camera coordinate system which has the origin at F and the symmetry axis as its z-axis. The central model can be defined by two parts:

1) the camera pose, which is composed by a rotation matrix R and location $\mathbf{t} \in \mathbb{R}^3$ of the center F, both expressed in the world coordinate system,

2) and an intrinsic projection function, $C : \mathbb{R}^3 \to \mathbb{R}^2$ which is expressed in the camera coordinate system.

Supposing $\mathbf{X}$ is the homogeneous world coordinates of a finite 3d point, there is a ray passing through this point and is reflected on the mirror then projected on image of the perspective camera. The projection is $p_c(\mathbf{X})$. We have,

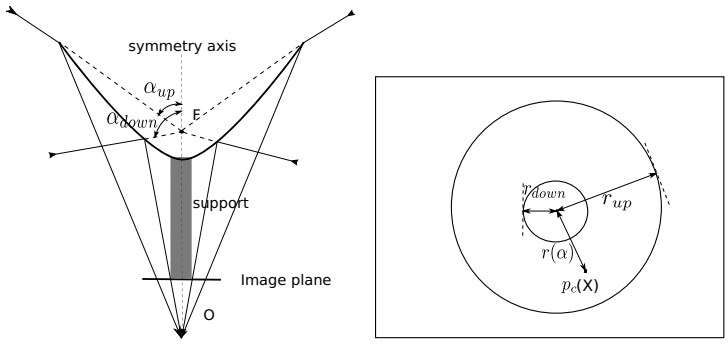
$$p_c(\mathbf{X}) = C(\mathbf{d}), \quad \mathbf{d} = R^T[I_3| - \mathbf{t}]\mathbf{X} \tag{3.6}$$

$\mathbf{d}$ is the direction (up to scale) of the ray which originates from the center F and goes through the 3d point, expressed in the camera coordinate system. We suppose that the origin and z-axis of the camera coordinate system are respectively F and the symmetry axis. The projection function C can be defined as,

$$C(\mathbf{d}) = C(x, y, z) = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + r(\alpha(x, y, z)) \frac{1}{\sqrt{x^2 + y^2}} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.7}$$

Here $(x_0\ y_0)^T$ is the image coordinates of center's projection, $\mathbf{d} = (x\ y\ z)^T$ and $\alpha(x, y, z)$ is the angle between z-axis and $\mathbf{d}$. We have $\alpha(x, y, z) = acos(\frac{z}{\sqrt{x^2+y^2+z^2}})$. The function $r(\alpha)$ is

a positive and decreasing polynomial function which maps angle α to the distance between $p_c(\mathbf{X})$ and the projection of center F. Thus we can see that by computing reverse function of $r(\alpha)$, a 2d point on image can be mapped to a ray whose direction is $\mathbf{d}$.



(a) Vertical field of view of the catadioptric camera

(b) $r_{down}, r(\alpha)$, and r_{up}

Figure 3.4: Vertical field of view of catadioptric camera.

As shown in Fig. 3.4, r_{up} and r_{down} are respectively radius of the concentric circles which are projections of the upper border of the mirror and the border of the mirror support on image plane. These borders define the vertical field of view of the catadioptric camera, and the corresponding angles $\alpha_{up}, \alpha_{down}$ are respectively the lower bound and the upper bound of α, $\alpha_{up} \leq \alpha \leq \alpha_{down}$. Thus we have, $r_{down} = r(\alpha_{down}) \leq r(\alpha) \leq r(\alpha_{up}) = r_{up}$. If r is linear, we say that the catadioptric camera is equiangular. The coefficients of the r polynomial are our intrinsic parameters of the catadioptric camera.

Comparisons Now if we compare the central and non central model, we can see that the central model is based on the assumption that all extended rays pass through a common point and the model has a symmetry axis. The non central model has not these assumptions.

In the general case, the catadioptric camera is non central in the reality. Thus the central model is an approximative catadioptric camera model and the non central model can be seen as an exact model. It might be natural to choose the non central model for our 3d geometry reconstruction because it approximates better to the reality. However, according to exhaustive experiments provided by [87] which compare both models for our camera with respect to robustness, accuracy and uncertainty estimations, the use of non-central model does not provide significant improvement on the 3d reconstruction results. Moreover, the geometry reconstruction using non central model is more complicated than using the central model. As a result, we choose the central model for our geometry reconstruction methods.

3.1.2 Detection and Matching of Interest Points

We have seen previously that a 2d image pixel can be mapped to a 3d ray which goes through the 3d point in question, however this 3d point can not be reconstructed by using only one ray. Now if we know that pixels of an image are matched to pixels in another image taken

at a different camera location, then not only 3d points but also the relative camera pose between two camera poses can be reconstructed. Here a pixel in an image is "matched" to a pixel in another image means that the two pixels are considered to be projections of a same 3d point on different images. In our work, we do not match all pixels but some interest points and reconstruct these interest points in 3d, i.e. our geometry reconstruction methods are sparse methods. Now we present how interest points are detected and matched in our batch and incremental SfM methods.

3.1.2.1 In the Batch SfM (Still image sequence)

We assume that the image sequence used for our batch SfM is composed of still images, i.e. the user alternates one shot and one step in the scene. Firstly, the Harris point detector [62] is used to detect interest points all images of the sequence. The Harris point detector is a standard interest point detector based on curvatures of auto-correlation function of pixels. It is selected as our interest point detector thanks to its detection stability and its invariance to rotation and illumination [122]. Other detection and matching methods exist, e.g. the SIFT detector [94]. But a review of these methods is outside the scope of this dissertation. Second, a two-view matching method is applied to each pair of consecutive images of the sequence. Here, each Harris point in one image is matched with Harris points in a corresponding search area of the other image by calculating the ZNCC score (Zero Mean Normalized Cross Correlation). And pairs of Harris points with high ZNCC scores are added to a list of matches. In the work [87], this list is further enlarged by a quasi-dense match propagation process [90] which progressively matches image pixels using a 2d-disparity gradient limit and the uniqueness constraint. In our work, in order to provide a sparse point cloud, only Harris points matched by [87] are used for subsequent steps. At last, the two-view matching is chained throughout the sequence. By doing so, the two-view matching is extended to n-view matching and a sequence of matched 2d points lists is obtained.

Note that our batch SfM can also deal with a video sequence by including an additional key-frame selection process described in the following section.

3.1.2.2 In the Incremental SfM (video sequence)

We assume that a video sequence is used for our incremental SfM. In this case, the distance between each pair of consecutive camera locations is not large enough which would make the subsequent 3d geometry reconstruction strongly erroneous. To solve this problem, we use an additional key-frame selection process [102]. This process uses also the two-view matching method in Sec. 3.1.2.1 to match Harris points in pairs of consecutive video frames, except that the match propagation process is not applied since the matching between two consecutive video frames is easy. The first frame of the sequence is always chosen as a key-frame, noted I^1. The second key-frame I^2 is chosen as the farthest frame in the sequence under the condition that there is at least M matched points between I^1 and I^2. Whenever key-frames I^1 to I^i $(2 \leq i)$ are chosen, we choose I^{i+1} such that at least M matched points exist between I^{i+1} and I^i and at least N matched points between I^{i+1} and I^{i-1}. In practice, $M = 600$ and $N = 400$. At last, only the lists of matched points between key-frames are

retained and used for our subsequent geometry reconstruction steps.

3.1.3 Camera Pose Estimation and 3d Point Reconstruction

This section shows methods of the 3d geometry (camera poses and 3d points) reconstruction under different conditions. We should note that tools to be presented here are basic methods used for initializing the scene geometry. They are usually combined with a robust estimation method (RANSAC) and their results are refined by an optimization process (bundle adjustment) to improve the reconstruction quality. Both RANSAC and bundle adjustment will be presented in further sections (Sec. 3.1.4 and Sec. 3.1.5).

3.1.3.1 Essential Matrix and Epipolar Constraint

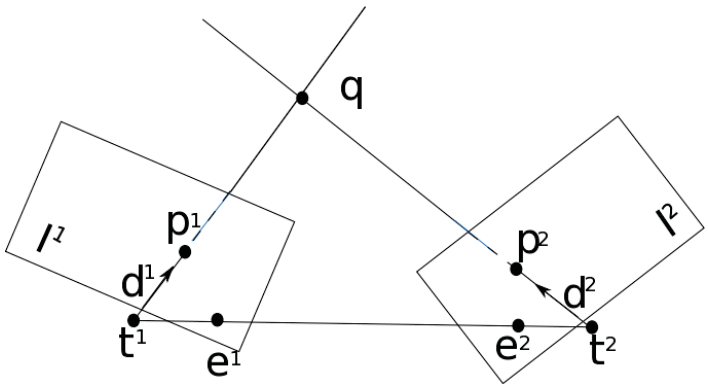


Figure 3.5: Epipolar geometry. $\mathbf{t}^1, \mathbf{t}^2$ are two camera locations, $\mathbf{p}^1, \mathbf{p}^2$ are respectively projections of a same 3d point $\mathbf{q}$ onto the images I^1, I^2 taken at $\mathbf{t}^1, \mathbf{t}^2$. The camera baseline $(\mathbf{t}^1\mathbf{t}^2)$ intersects I^1, I^2 on e^1, e^2 respectively, which are called epipoles. The ray $[\mathbf{t}^1\mathbf{p}^1)$ intersects with ray $[\mathbf{t}^2\mathbf{p}^2)$, and the intersected point is $\mathbf{q}$.

Let I^1, I^2 be two perspective images and $\mathbf{t}^1, \mathbf{t}^2$ be respectively two camera locations (in world coordinates) where I^1, I^2 are taken. Assume that two camera coordinate systems exist which have respectively their origins at $\mathbf{t}^1, \mathbf{t}^2$. Let $\mathbf{c}^1 = (R^1, \mathbf{t}^1)$ (resp. $\mathbf{c}^1 = (R^2, \mathbf{t}^2)$) be camera pose in world coordinates of I^1 (resp. I^2). Remember that a camera pose is the transformation of world coordinates to camera coordinates.

Now a 3d point $\mathbf{q}$ is projected on I^1, I^2, and the projections are respectively $\mathbf{p}^1$ and $\mathbf{p}^2$ (see Fig. 3.5). According to the perspective camera model (Sec. 3.1.1.1, both back-projected rays (half-lines) $[\mathbf{t}^1\mathbf{p}^1), [\mathbf{t}^2\mathbf{p}^2)$ should go through $\mathbf{q}$. Let $\mathbf{d}^1, \mathbf{d}^2$ be directions of $[\mathbf{t}^1\mathbf{p}^1), [\mathbf{t}^2\mathbf{p}^2)$ in camera coordinates. The coplanarity of $R^1\mathbf{d}^1$, $R^2\mathbf{d}^2$ and $\mathbf{t}^1 - \mathbf{t}^2$ gives the epipolar constraint which can be expressed as follows,

$$(\mathbf{d}^1)^T (R^1)^T [\mathbf{t}^2 - \mathbf{t}^1]_\times R^2 \mathbf{d}^2 = 0 \tag{3.8}$$

It can be further simplified as,

$$(\mathbf{d}^1)^T E \mathbf{d}^2 = 0 \tag{3.9}$$

where $E = (R^1)^T[\mathbf{t}^2 - \mathbf{t}^1]_\times R^2$ is called the essential matrix. It is a 3×3 rank-two matrix and it can be shown that its two nonzero singular values are equal.

According to Eq. 3.9, if $\mathbf{d}^1$ and $\mathbf{d}^2$ are known, then we have a linear constraint on E. If $\mathbf{d}^1$ and E are known, then we have a linear constraint on $\mathbf{d}^2$. In this case, the epipolar constraint can also be used as a matching constraint. Here, $\mathbf{p}^1 \in I^1$ defines line $(\mathbf{e}^2\mathbf{p}^2)$ on which the potential matches of $\mathbf{p}^2 \in I^2$ should lie.

We should note that Eq. 3.8 and Eq. 3.9 are also met for our central catadioptric camera model. In this case, $\mathbf{t}^1$(resp. $\mathbf{t}^2$) is the center F^1 (resp. F^2). According to the central catadioptric camera model (Sec. 3.1.1.2), projection points $\mathbf{p}^1, \mathbf{p}^2$ can be mapped to two back-projected rays which both go through $\mathbf{q}$. The ray direction is respectively $\mathbf{d}^1, \mathbf{d}^2$. At last, the coplanarity of $\mathbf{d}^1$ and $\mathbf{d}^2$ gives Eq. 3.8 and Eq. 3.9.

3.1.3.2 Camera Pose Estimation

Camera poses can be estimated using epipolar constraints of pairs of matched points. Assume that we know a pair of matched points $\mathbf{p}^1, \mathbf{p}^2$ in images I^1, I^2. Let camera poses of I^1, I^2 be respectively $\mathbf{c}^1, \mathbf{c}^2$. If the catadioptric camera is calibrated, then $\mathbf{p}^1, \mathbf{p}^2$ can be respectively mapped to two rays $[\mathbf{t}^1\mathbf{p}^1)$, $[\mathbf{t}^2\mathbf{p}^2)$ whose directions are $\mathbf{d}^1$, $\mathbf{d}^2$ in camera coordinates. Furthermore Eq. 3.9 is met.

Now the five-point algorithm [107] is used. This algorithm uses five pairs of matched points to establish five constraints defined by Eq. 3.9 and the relative camera pose between $\mathbf{c}^1$ and $\mathbf{c}^2$ is efficiently calculated. We do not go to details for this algorithm (also seven-point or three-point algorithm below), check [107] for more details. An alternative and simple method estimates E using a linear (least squares) method from seven point correspondences [63]. Then R^1, R^2 and $\mathbf{t}^1 - \mathbf{t}^2$ can be estimated by singular value decomposition [63].

Now assume that the catadioptric camera is calibrated and some 3d points are also reconstructed. In this case, given at least three 3d points and their 2d matched points in one image I, the camera pose $\mathbf{c}$ of I can be estimated. In our work, we use the three-point algorithm of Grunert [61] based on trigonometry calculations (law of cosines) in the tetrahedron composed by the camera location of $\mathbf{c}$ and the three 3d points.

3.1.3.3 3d Point Reconstruction

In reality, $[\mathbf{t}^1\mathbf{p}^1)$ and $[\mathbf{t}^2\mathbf{p}^2)$ in Fig. 3.5 do not intersect due to image noise. In our work, we use a classical solution to reconstruct (initialize) $\mathbf{q}$: the mid-point method [47].

The principle of mid-point method is illustrated by Fig.3.6. In the non degenerate case ($\mathbf{d}^1$ and $\mathbf{d}^2$ are not parallel), we calculate points $\mathbf{q}^1, \mathbf{q}^2$ such that $\mathbf{q}^1, \mathbf{q}^2$ lies respectively on ray $[\mathbf{t}^1\mathbf{p}^1)$ and $[\mathbf{t}^1\mathbf{p}^1)$. Besides, $\mathbf{q}^1\mathbf{q}^2$ should be perpendicular to $\mathbf{d}^1$ and $\mathbf{d}^2$. The reconstructed 3d point $\mathbf{q}$ is then the middle point of $\mathbf{q}^1\mathbf{q}^2$.

Here are detailed formulas to calculate $\mathbf{q}^1, \mathbf{q}^2$ and $\mathbf{q}$:

$$\begin{aligned} \mathbf{q} &= \frac{\mathbf{q}^1 + \mathbf{q}^2}{2} \\ \mathbf{q}^1 &= \mathbf{t}^1 + \alpha\mathbf{d}^1 \\ \mathbf{q}^2 &= \mathbf{t}^2 + \beta\mathbf{d}^2 \end{aligned} \tag{3.10}$$

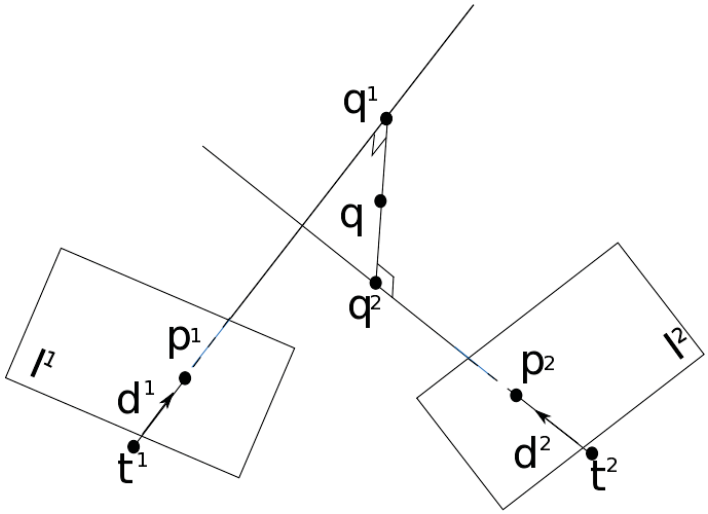


Figure 3.6: Mid-point method. $\mathbf{t}^1, \mathbf{t}^2$ are two camera locations, I^1, I^2 are respectively two images taken at $\mathbf{t}^1, \mathbf{t}^2$. $\mathbf{p}^1, \mathbf{p}^2$ are respectively two matched points corresponding to a same 3d point in I^1, I^2. $\mathbf{d}^1$, $\mathbf{d}^2$ are respectively directions of rays $[\mathbf{t}^1\mathbf{p}^1)$, $[\mathbf{t}^2\mathbf{p}^2)$. There is a line perpendicular to $\mathbf{d}^1$ and $\mathbf{d}^2$, and intersects the ray $[\mathbf{t}^1\mathbf{p}^1)$ and $[\mathbf{t}^2\mathbf{p}^2)$ on $\mathbf{q}^1$, $\mathbf{q}^2$. The reconstructed 3d point $\mathbf{q}$ is the middle point of $\mathbf{q}^1\mathbf{q}^2$.

and,

$$\begin{aligned}\alpha &= \frac{((\mathbf{t}^2-\mathbf{t}^1)\cdot\mathbf{d}^2)(\mathbf{d}^1\cdot\mathbf{d}^2) - ((\mathbf{t}^2-\mathbf{t}^1)\cdot\mathbf{d}^1)\|\mathbf{d}^2\|^2}{(\mathbf{d}^1\cdot\mathbf{d}^2)^2 - \|\mathbf{d}^1\|^2\|\mathbf{d}^2\|^2} \\ \beta &= \frac{((\mathbf{t}^2-\mathbf{t}^1)\cdot\mathbf{d}^1)(\mathbf{d}^1\cdot\mathbf{d}^2) - ((\mathbf{t}^2-\mathbf{t}^1)\cdot\mathbf{d}^2)\|\mathbf{d}^1\|^2}{(\mathbf{d}^1\cdot\mathbf{d}^2)^2 - \|\mathbf{d}^1\|^2\|\mathbf{d}^2\|^2}\end{aligned} \tag{3.11}$$

3.1.4 Robust Estimation

It is important to estimate robustly parameters in the geometry estimation process because the measures are not only noised but sometimes aberrant. These aberrant measures, called also outliers, are usually due to erroneous measurements such as change of lighting, occlusion, or matching failure. A *robust* estimation method is a method which can still correctly estimate parameters despite the presence of outliers.

The robust estimation problem can be defined as follows. Given a set of n observations $\{s_i\}, i = 1..n$, we know that $\forall i, e(x^*, s_i) = 0$ where x^* are some unknown parameters, and $e(,) \geq 0$ is an error function involving x^* and $\{s_i\}$. We also know that an estimation x of x^* can be calculated by at least $m(m \leq n)$ observations. Now the objective is to find an estimation x which is close to the real parameters x^* despite the outliers.

The robust estimation method that we used in our work is the RANdom SAmple Consensus (RANSAC) algorithm [50]. This algorithm performs a fixed number N of x estimations and at last chooses the best one. In each tentative, it selects randomly m observations to determinate x and calculates the number of observations which fit x. To check if an observation s_i fits x, i.e. s_i is inlier to x, the error function $e(,) \geq 0$ and a threshold $e_{\max}$ are used:

s_i is inlier to x if $e(s_i, x) < e_{\max}$. Here, the definition of $e(,)$ and how to choose $e_{\max}$ in our case will be given in Sec. 3.1.5. At last, the estimation x which has the largest number of inliers is chosen as the solution.

The number N of tentatives should be chosen high enough to ensure a probability p (0.99 in practice) that there is at least one x to which the m observations are inlier. Let ε be the percentage of inlier observations, then we have,

$$N \geq \frac{\log(1-p)}{\log(1-\varepsilon^m)}$$

Note that the solution of RANSAC may be not the optimal estimation of x^*: as we limit the number of iterations, there may be other estimation which has more inliers. Nevertheless, RANSAC is able to provide high precision estimation even if many observations are outliers.

3.1.5 Bundle Adjustment

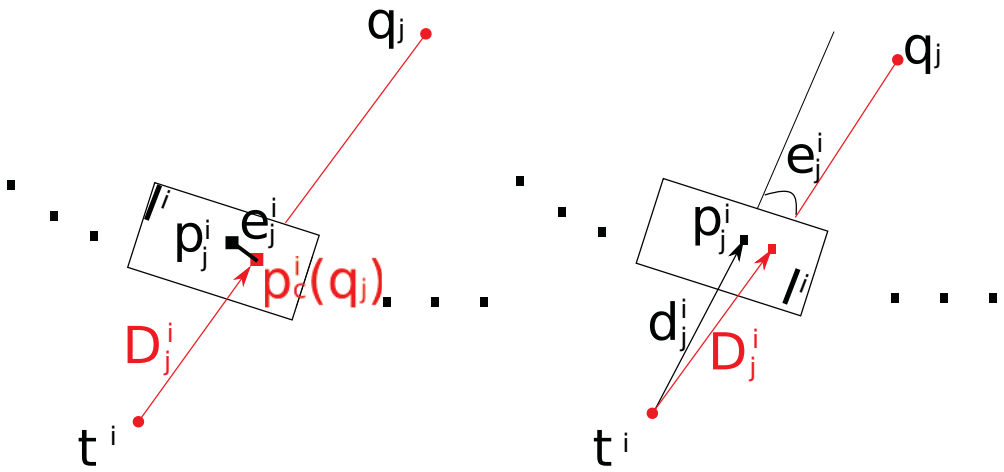
Bundle Adjustment (BA) is a process which refines the geometry reconstruction results including 3d points, camera poses (and sometimes camera intrinsic parameters as in [87]), by minimizing reconstruction errors. The reconstruction errors involving the matched 2d points, the reconstructed 3d points and camera poses, are expressed as squared Euclidean norms of many non linear functions,

$$E = \sum_{i,j} \|e_j^i\|^2 \tag{3.12}$$

Here $\|\cdot\|$ is the Euclidean norm. e_j^i is the error vector of the reconstructed camera pose $\mathbf{c}^i$ and the reconstructed 3d point $\mathbf{q}_j$, with respect to the matched 2d point $\mathbf{p}_j^i$ of $\mathbf{q}_j$ on image of $\mathbf{c}^i$. The sum of these errors is globally optimized by BA thus the problem turns to be a non linear optimization problem. Different forms of e_j^i exist, which will be discussed in the next paragraph.

Errors for Bundle Adjustment Bundle adjustment refines the reconstructed geometry by minimizing the sum of squared Euclidean norms of error values. Standard BA uses image error which measures the discrepancy between the projections of reconstructed 3d points and the matched 2d pixels [140], see Fig. 3.7a. In our approach, the error e_j^i proposed by [87] is used. Here, e_j^i measures the angle between two rays originating from the same reconstructed camera location $\mathbf{t}^i$. One ray, whose direction is $\mathbf{d}_j^i$, goes through the detected matched 2d point $\mathbf{p}_j^i$. The other ray, whose direction is $\mathbf{D}_j^i$, goes through the reconstructed 3d point $\mathbf{q}_j$ (see Fig. 3.7b). Note that both $\mathbf{d}_j^i$ and $\mathbf{D}_j^i$ are expressed in the camera coordinate system. Suppose that the world coordinates of reconstructed $\mathbf{c}^i$ (camera pose) is $(R^i, \mathbf{t}^i)$. Then error e_j^i is defined as follows,

$$e_j^i = \pi_2(R_j^i \mathbf{D}_j^i), \text{where } R_j^i \text{ is a rotation matrix such that } R_j^i \mathbf{d}_j^i = (0\ 0\ 1)^T \tag{3.13}$$



(a) Image Error. The ray with direction $\mathbf{D}_j^i$ originates from $\mathbf{t}^i$ and goes through $\mathbf{q}_j$. It has a reprojected pixel $p_c^i(\mathbf{q}_j)$ on I^i. The image error is $e_j^i = p_c^i(\mathbf{q}_j) - \mathbf{p}_j^i$.

(b) Angular Error. Rays of $\mathbf{d}_j^i$ and $\mathbf{D}_j^i$ both originate from $\mathbf{t}^i$. $\mathbf{d}_j^i$ goes through $\mathbf{p}_j^i$ and $\mathbf{D}_j^i$ goes through $\mathbf{q}_j$. $\|e_j^i\|$ is the tangent of the angle between $\mathbf{d}_j^i$ and $\mathbf{D}_j^i$.

Figure 3.7: Image error and angular error. I^i is the image taken by $\mathbf{t}^i$, $\mathbf{q}_j$ is a reconstructed 3d point, $\mathbf{p}_j^i$ is the 2d matched matched point of $\mathbf{q}_j$ in I_i and e_j^i is the error vector.

and $\pi_2(x\ y\ z)^T = (\frac{x}{z}\ \frac{y}{z})^T$. Let $\mathbf{X}_j$ be the homogeneous world coordinates of $\mathbf{q}_j$, then we have

$$e_j^i = \pi_2(R_j^i(R^i)^T[I_3| - t^i]\mathbf{X}_j) \tag{3.14}$$

We note that $\|\pi_2(x\ y\ z)\|^2 = \frac{x^2+y^2}{z^2}$ is the squared tangent of the angle between $(0\ 0\ 1)^T$ and $(x\ y\ z)^T$. Thus we have

$$\|e_j^i\|^2 = \tan^2(\alpha_j^i) \text{ with } \alpha_j^i = \text{angle}(\mathbf{d}_j^i, \mathbf{D}_j^i) \tag{3.15}$$

This error model guarantees the C^2 continuity even when the 3d point is in the infinite plane (this is not the case when catadioptric image error is used) thus the local minimum of the cost function can be efficiently reached [140]. Another advantage is that this error model is generic: it does not involve the camera model during minimization. It is suitable to catadioptric cameras but also works successfully with perspective cameras [86].

The error function defined here is also used for the RANSAC process described in Sec. 3.1.4. In our case, the error function e_j^i measures the angle between $\mathbf{d}_j^i$ and $\mathbf{D}_j^i$, which involve the 2d matched points (observations) together with 3d reconstructed points and camera poses (parameters). Besides, in RANSAC, a 2d matched point (observation) $\mathbf{p}_j^i$ is inlier to parameters x if the error $e(\mathbf{p}_j^i, x)$ is smaller than a threshold $e_{\max}$. Here, $e(\mathbf{p}_j^i, x) = \|e_j^i\|$ and it is calculated as the tangent of the angle e_j^i. In practice, $e_{\max}$ corresponds to an image error of 2 pixels.

Levenberg-Marquardt The BA in our work is based on the Levenberg-Marquardt algorithm (LM), which estimates a numeric solution for problem of non linear function minimization.

The reconstructed camera poses and 3d points (and intrinsic parameters in case of the batch Structure-from-Motion method) form the parameter vector X for LM. Now LM begins by an initial guess of X, adds a suitable δX to X in each iteration to reduce E (Eq. 3.12) and finally converges to a locally optimal solution $\hat{X}$ such that the sum E of reconstruction errors $\|e_j^i\|^2$ reaches the (local) minimum. LM combines the Gauss-Newton algorithm which converges rapidly and the method of gradient descent which can converge (slowly) even if the initial guess is far from the optimal solution. Details about the Levenberg-Marquardt algorithm can be found in appendix A.

3.2 Our Batch Structure-from-Motion Method

Until now, we have presented all principles and notions that we use for our SfM methods and we can now describe our SfM methods. Our batch SfM method is almost the same to the method in [87]. Both estimate the geometry of a scene in a batch manner. Besides, camera intrinsic parameters are also estimated and adjusted. The slight difference is that method in [87] only deals with still image sequence but we can also deal with video sequence thanks to a key-frame selection process (see Sec. 3.1.2.2).

In the first step, we initialize the intrinsic parameters of the central catadioptric camera model (Sec. 3.1.1). To do so, large and small circles in each catadioptric image are firstly detected or given. Then the projection function $r(\alpha)$ is defined as a linear function such that $r(\alpha_{up}) = r^i_{up}$ and $r(\alpha_{down}) = r^i_{down}$. r^i_{up}, r^i_{down} are respectively the radii of large, small circles of each image and angles α_{up}, α_{down} are initialized using values provided by the mirror manufacturer. So we assume that the camera is equiangular for the initialization.

Second, Harris points are detected and matched for each pair of consecutive images (Sec. 3.1.2) and for each image, rays are generated for each matched 2d point in the local camera coordinate system.

Third, essential matrices for pairs of matched 2d points are estimated using the 7-point algorithm [63] and RANSAC (Sec. 3.1.4), and refined by Levenberg-Marquardt (Appendix. A). Next, 3d points are also reconstructed using pairs of rays and the mid-point method. As many 3d points are tracked in three consecutive images, the 3d scale factor between every two consecutive image pairs is calculated using triples of matched points. At the end of this step, all points in each three views are reconstructed, in other words, partial geometry is reconstructed for every triple of consecutive images.

Fourth, partial geometries of triples of consecutive images are refined and fused in a hierarchical framework [63]. To do so, the sequence is divided in a binary tree. Sub-sequences of the same parent node have two consecutive common images and sub-sequences of the lowest level are thus triples of consecutive images whose partial geometries are already reconstructed. See Fig. 3.8 for an example of hierarchy of a sequence of 10 images. Now, to fuse two sub-sequences of the same parent node, the geometry of one sub-sequence is mapped to the coordinate system of the other thanks to the common camera poses and the result is the geometry of the node which is refined by BA after the fusion. By repeating the fusion and BA, the geometry of the whole sequence is finally reconstructed.

At last, the linear function $r(\alpha)$ which defines an approximative calibration is replaced by

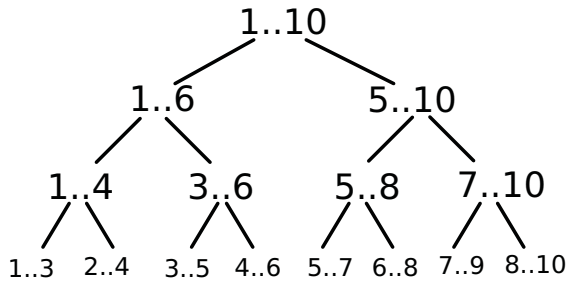


Figure 3.8: Hierarchy framework of a sequence of 10 images. The sequence is subdivided in a binary tree. For each node, $i..j$ ($1 \leq i < j \leq 10$) means that the subsequence of the node contains image I^i to image I^j (I^i, I^j included). Subsequences of a same parent node have two consecutive common images.

a cubic polynomial. The four coefficients of the polynomial, together with the geometry of the sequence are refined by an additional BA. Here, $4+6m+3n$ parameters are refined, where m is the number of camera poses and n is the number of 3d points. In some cases, loops exist in the camera trajectory, i.e. images are taken in a same position at different times. However due to the accumulation of reconstruction errors, called SfM drift, the reconstructed camera viewpoints at the beginning and the end of a loop are not the same. To deal with the SfM drift, in our work, a loop closure method [88] is applied to remove the SfM drift. Details about the loop closure can be found in [88].

3.3 Our Incremental Structure-from-Motion Method

Now we summarize the incremental SfM algorithm that we used, which is almost the same to the one [104] in the central generic case. The input is a video sequence and our method selects key-frames (Sec. 3.1.2.2) for the further steps of environment modeling. Note that still image sequences can also be directly used by our incremental SfM without key-frame selection.

Our incremental geometry reconstruction method begins by a three-view initialization step based on RANSAC algorithm. We assume that the catadioptric camera is already calibrated. Three key-frames are firstly selected, and then camera poses of the first and the third key-frames $\mathbf{c}^1, \mathbf{c}^3$ are initialized using the five-point algorithm. Next, 3d points are reconstructed using mid-point method, and $\mathbf{c}^2$ is initialized using Grunert three-point algorithm (Sec. 3.1.3). At last, the three camera poses and reconstructed 3d points are refined by bundle adjustment (BA).

After the three-view initialization, an incremental step is applied whenever the matched point lists of a new frame are available. New key-frames are progressively selected based on numbers of matches between the current frame and previous key-frames. For each new key-frame, the corresponding camera pose $\mathbf{c}^i$ is firstly computed using the 3-point algorithm

and RANSAC. Then new 3d points are reconstructed for the new key-frame using RANSAC. At last, a local BA is applied to refine the local reconstructed geometry.

Here we present the local BA mentioned in the paragraph above. The local BA refines the local geometry of n_l ($3 \leq n_l$) most recent key-frames by minimizing its errors involving N_l ($N_l \geq n_l + 2$) most recent key-frames. More precisely, the parameters to be refined by local BA are the n_l last camera poses $C^i = \{\mathbf{c}^{i-n_l+1}, .., \mathbf{c}^i\}$ and the 3d points Q^i projected on key-frames of C^i. Regarding the reconstruction errors, local BA minimizes errors of rays which go through a point of Q^i and whose originating camera poses are in $\{\mathbf{c}^{i-N_l+1}, .., \mathbf{c}^i\}$. In practice [104], n_l is set to 3 and N_l is set to 10.

3.4 Comparison of Our Two SfM Methods

In this chapter, we have presented a batch and an incremental SfM methods, which are respectively the pre-processing of our batch (Chap. 4) and incremental (Chap. 5) surface reconstruction methods. Both SfM methods use common tools for the geometry reconstruction such as the central catadioptric camera model, the Harris point detection and matching, the camera pose estimation, the RANSAC robust estimation and the Levenberg-Marquardt algorithm for bundle adjustment (BA). However, there are several differences between them.

First, our batch SfM method accurately calibrates intrinsic parameters of the camera but our incremental SfM method requires that the camera intrinsic parameters are pre-calibrated.

Second, our batch SfM reconstructs the entire geometry of the environment in an hierarchical framework, while the incremental SfM reconstructs progressively the environment geometry. Besides, our batch SfM has different hierarchical levels of BA while our incremental SfM has only $n_l = 3$ local BAs per key-frame.

Third, if the camera trajectory contains loops, our batch SfM has a loop closure process which deals with the SfM drift. Our incremental SfM does not have the loop closure.

Chapter 4

Batch Surface Reconstruction

4.1 Introduction

Once the geometry of an environment is reconstructed, a 3d model can be estimated based on the reconstructed geometry. Our batch surface reconstruction method reconstructs a 2-manifold surface to model the environment, using the geometry reconstructed by the batch SfM method presented in Sec. 3.2.

The surface is a widely used model in computer graphics and computer vision applications. Suppose that the environment to model is opaque, the 3d space of the environment can be considered to be composed by two regions: the *inside* region and the *outside* region. The former is the region inside objects of the environment which is invisible if one observes the model at the camera viewpoints. The latter is the region outside objects of the environment which are transparent to the observer. We assume that the surface of the real scene between *inside* and *outside* regions is a 2-manifold. The objective of our method is to reconstruct a 2-manifold which approximates the real surface.

The batch SfM algorithm is used as the pre-processing of our batch surface reconstruction. It provides the geometry information of environments for our surface reconstruction, which includes a list of estimated camera viewpoints $T = \{\mathbf{t}^i\}$, a sparse cloud of reconstructed 3d points $Q = \{\mathbf{q}_j\}$, and for each point $\mathbf{q}_j$, a list of visibility constraints V_j.

Here, a visibility constraint in V_j is a line segment which has $\mathbf{q}_j$ and one of the camera viewpoints $\mathbf{t}^i$ as extremities such that $\mathbf{q}_j$ should be visible to $\mathbf{t}^i$. Giving more details, in our geometry reconstruction process (see Chap. 3), a 3d point $\mathbf{q}_j$ is first initialized by using rays originated from different camera viewpoints. After bundle adjustment, only inlier rays are retained. Here, V_j corresponds to the list of these inlier rays. Thus a visibility constraint is also called a visibility ray or simply a ray in the following scope of this dissertation.

Our batch surface reconstruction has six steps described as follows:

1. At first, a 3d Delaunay triangulation D is created by using points in Q as vertices. Therefore the space is *discretized* into tetrahedra. (Sec. 4.2).

2. Then using visibility rays, tetrahedra of D are classified into two categories: *free-space* and *matter*. (Sec. 4.3).

3. Until now, a triangulated surface can already be obtained by selecting triangles separating *free-space* and *matter* tetrahedra. However this surface is non-manifold which is unfavorable for further processings like surface refinement or surface smoothing. In this step, a greedy algorithm is applied to generate a 2-manifold as the surface model of environments, which at the same time maximizes the visibility consistency obtained at the SfM step. (Sec. 4.4).

4. The 2-manifold reconstructed by step 3 has genus 0 i.e. it has a ball topology. It is problematic if the camera trajectory contains closed loop(s). To solve this problem, a topology extension process is applied. (Sec. 4.5).

5. Spurious handles can exist on the reconstructed 2-manifold due to the step 3 and 4. Two methods are used to deal with these spurious handles and try to remove them off. (Sec. 4.6).

6. At last, to improve the model quality and to facilitate the model visualization, post-processings are done on the 2-manifold including peak removal, surface denoising, sky triangle removal, and surface texturing. (Sec. 4.7).

4.2 3d Delaunay Triangulation

The first step of our batch surface reconstruction is to obtain a discretization of the space. The 3d Delaunay triangulation is chosen for this step, thanks to its advantages presented in Sec. 1.5.1.

The 3d Delaunay triangulation D is built by taking points in Q as its vertices but not all points in Q are added in D. A point $\mathbf{q}_j$ has poor accuracy if it is reconstructed in a degenerate configuration: if all rays of V_j are nearly parallel [63]. Remember that V_j is the list of inlier rays which reconstruct $\mathbf{q}_j$ in SfM (Sec. 4.1). The size of V_j is greater than two (two is the theoretical minimum but it is insufficient for robustness). This degenerate configuration occurs when the reconstructed points are close to part of the camera trajectory which is a straight line. In our work, the point cloud is filtered (as in [42]) to avoid the degenerate configuration: a point $\mathbf{q}_j$ is added in D if and only if there is at least one aperture angle $\widehat{\mathbf{t}^i\mathbf{q}_j\mathbf{t}^k}$ of $\mathbf{q}_j$ which meets $\epsilon \leq \widehat{\mathbf{t}^i\mathbf{q}_j\mathbf{t}^k} \leq \pi - \epsilon$. Here, $\mathbf{t}^i\mathbf{q}_j$ and $\mathbf{t}^k\mathbf{q}_j$ are in V_j, and $\epsilon > 0$ is a threshold.

In practice, ϵ is 5 or 10 degrees. Our 3d Delaunay triangulation is then built using the filtered Q. By doing so, many inaccurate points are ignored for our subsequent surface reconstruction. Besides, we obtain another consequence thanks to the point filter. It is described in Lemma. 4.1 below.

Lemma 4.1 *The length of a visibility ray in V_j is bounded by $\frac{\max\|\mathbf{t}^i-\mathbf{t}^k\|}{\sin(\epsilon)}$, where $\mathbf{t}^i\mathbf{q}_j$ and $\mathbf{t}^k\mathbf{q}_j$ are in V_j.*

Now we firstly present this lemma in the 2d case. Here, we remind a property of circle. Assume that there are a circle and three points: a mobile point $\mathbf{c}$, and two fixed points $\mathbf{a}$,$\mathbf{b}$ lying on the circle which separate the circle on two arcs. Then the angle $\widehat{\mathbf{acb}}$ is always

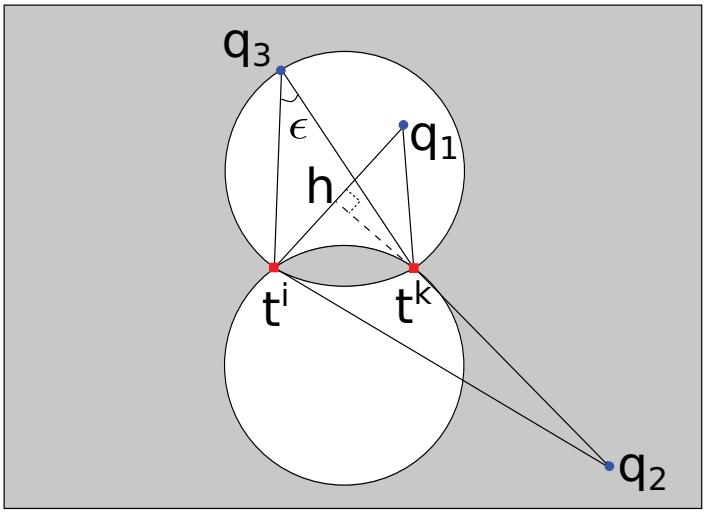


Figure 4.1: An example of point filter for a pair of camera viewpoints in 2d. $\{\mathbf{t}^i, \mathbf{t}^k\}$ is a pair of two camera viewpoints (red squared points), $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3$ are 3d points (blue round points), $\mathbf{t}^i\mathbf{q}_j$ and $\mathbf{t}^k\mathbf{q}_j$ are in V_j ($j \in \{1, 2, 3\}$). $\mathbf{h}$ is the projection of $\mathbf{q}_k$ on $\mathbf{t}^i\mathbf{q}_1$. For $\{\mathbf{t}^i, \mathbf{t}^k\}$, the non degenerate region is in white, and the degenerate region is in gray. $\mathbf{q}_1, \mathbf{q}_3$ are in the non degenerate region of $\{\mathbf{t}^i, \mathbf{t}^k\}$, thus can be added in D. $\mathbf{q}_2$ is outside this region thus cannot be added in D according to $\{\mathbf{t}^i, \mathbf{t}^k\}$. Thus $\mathbf{q}_2$ should be tested with other pairs of camera viewpoints.

constant if $\mathbf{c}$ lies on the circle and moves in the same arc. $\widehat{\mathbf{acb}}$ becomes bigger (resp. smaller) if $\mathbf{c}$ moves towards (far from) the circle center.

With this property in mind, we can see that for a pair of camera viewpoints $\{\mathbf{t}^i, \mathbf{t}^k\}$, a point $\mathbf{q}_j$ is in non degenerate configuration if it is in a so-called "non degenerate" region (see Fig. 4.1). The aperture angle $\widehat{\mathbf{t}^i\mathbf{q}_j\mathbf{t}^k}$ in this region meets $\epsilon \leq \widehat{\mathbf{t}^i\mathbf{q}_j\mathbf{t}^k} \leq \pi - \epsilon$. Thus this region is formed by two disks of same radius which both have $\mathbf{t}^i$ and $\mathbf{t}^k$ on their border (a circle), with their intersected part removed. We have $\sin(\widehat{\mathbf{t}^i\mathbf{q}_j\mathbf{t}^k}) \geq \sin(\epsilon)$ since $\epsilon \leq \widehat{\mathbf{t}^i\mathbf{q}_j\mathbf{t}^k} \leq \pi - \epsilon$.

As illustrated in Fig. 4.1, let $\mathbf{h}$ be the projection of $\mathbf{t}^k$ on $\mathbf{t}^i\mathbf{q}_1$, then we have,

$$\|\mathbf{q}_1 - \mathbf{t}^k\| = \frac{\|\mathbf{h} - \mathbf{t}^k\|}{\sin(\widehat{\mathbf{t}^i\mathbf{q}_1\mathbf{t}^k})} \leq \frac{\|\mathbf{t^i} - \mathbf{t}^k\|}{\sin(\widehat{\mathbf{t}^i\mathbf{q}_1\mathbf{t}^k})} \leq \frac{\|\mathbf{t^i} - \mathbf{t}^k\|}{\sin(\epsilon)} \tag{4.1}$$

It means that the length of the ray in the non degenerate region defined by $\{\mathbf{t}^i, \mathbf{t}^k\}$ is bounded by $l_{\max} = \frac{\|\mathbf{t^i} - \mathbf{t}^k\|}{\sin(\epsilon)}$.

Now by rotating the 2d non degenerate region around axis $\mathbf{t}^i\mathbf{t}^k$, we extend this region to 3d. Notice that $l_{\max}$ does not change. We can finally conclude the lengths of all rays are bounded by $l_{\max} = \frac{\max \|\mathbf{t}^i - \mathbf{t}^k\|}{\sin(\epsilon)}$.

At last, we also note that this point filter is just a simple way to select accurate points for the 3d Delaunay triangulation. Other and more sophisticated criteria [43, 89] exist.

4.3 *Free-space/matter* Labeling by Ray-tracing

As we have already mentioned in Sec. 4.1, the objective of this step is to divide the space in two regions: *free-space* region and *matter* region. The 3d Delaunay triangulation D discretizes the space, thus each tetrahedron of D should be classified to be either *matter* or *free-space*.

Let Δ be a tetrahedron of D. If Δ is intersected with at least one ray of a point in Q, then Δ should be transparent to an observer who observes the model at the camera viewpoint of the intersected ray. Δ is then labeled *free-space*. A simple algorithm can then be designed by checking for each tetrahedron Δ of D if Δ intersects any ray of any point in Q. If yes then it is *free-space*, otherwise it is *matter*. This is a brute force approach which has a high complexity. In our work, a more efficient method is preferred, called the ray-tracing algorithm (as in [93]). Complexity analyses in Sec. 6.3.2.2 compare the complexity of both approaches.

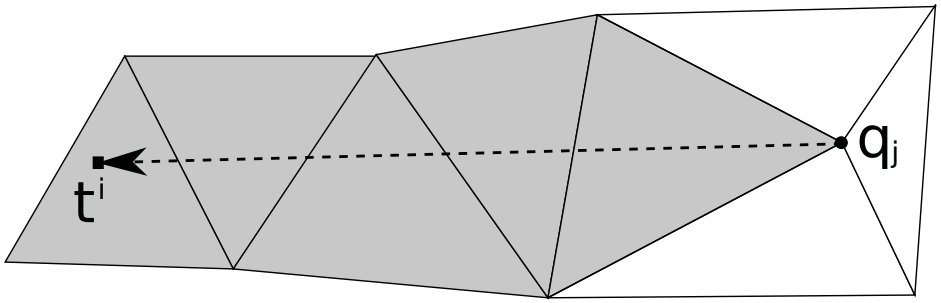


Figure 4.2: Ray-tracing in the 2d case. The process begins at $\mathbf{q}_j$ and traces towards $\mathbf{t}^i$. It continuously finds the tetrahedra intersected by $\mathbf{t}^i\mathbf{q}_j$ and labels them *free-space*. Here, we see also Delaunay triangles (solid lines) and the visibility constraint $\mathbf{t}^i\mathbf{q}_j$ (dashed line).

Ray-tracing All tetrahedra are initialized as *matter*, including infinite tetrahedra. Remember that according to Sec. 1.5.2, D is a graph and infinite tetrahedra are formed by triangles on convex hull of Q and the infinite vertex, which is outside the convex hull of Q. In our work, Q is reconstructed in almost all directions around camera viewpoints (we use a omni-directional camera). Thus generally, rays (and also all camera viewpoints) are in the convex hull of Q and do not intersect infinite tetrahedra. As a result, these infinite tetrahedra are labeled *matter*.

Saying that a tetrahedron Δ is intersected by a ray $\mathbf{t}^i\mathbf{q}_j$ means that they have common points, and at least one of the common points should be interior of Δ. In other words, if all common points of Δ and $\mathbf{t}^i\mathbf{q}_j$ are in faces, edges, or vertices of Δ, then we do not consider that Δ is intersected by $\mathbf{t}^i\mathbf{q}_j$.

Now the ray-tracing process is applied to each ray $\mathbf{t}^i\mathbf{q}_j$ which forces all tetrahedra intersected by $\mathbf{t}^i\mathbf{q}_j$ to be *free-space*. As D is a graph (Sec. 1.5.2), tracing a ray $\mathbf{t}^i\mathbf{q}_j$ is a walk in the graph, as illustrated in Fig. 4.2. It starts from a tetrahedron incident to $\mathbf{q}_j$, moves to another tetrahedron which is adjacent to the starting tetrahedron through the triangle

intersected by the line segment $\mathbf{t}^i\mathbf{q}_j$, and stops to the tetrahedron which contains $\mathbf{t}^i$ (the inverse walk is also possible). Ray $\mathbf{t}^i\mathbf{q}_j$ is traced if and only if $\mathbf{q}_j$ is a vertex of D. Here, remember that the two neighborhood definitions ("incident" and "adjacent") are already given in Sec. 1.3.

At last, after applying ray-tracing to all rays, tetrahedra which have at least one intersected ray are *free-space*, the rest tetrahedra remain *matter*. The algorithm also saves for each *free-space* tetrahedron, the number of rays which intersect this tetrahedron. It is used as the priority score in the further 2-manifold generation step (Sec. 4.4.3).

Special Case A special case might occur in the ray-tracing process: if (at least) a camera viewpoint is located outside the convex hull of the Delaunay triangulation D. In this case, ray-tracing for some rays cannot be correctly applied (the infinite vertex is not geometrically defined).

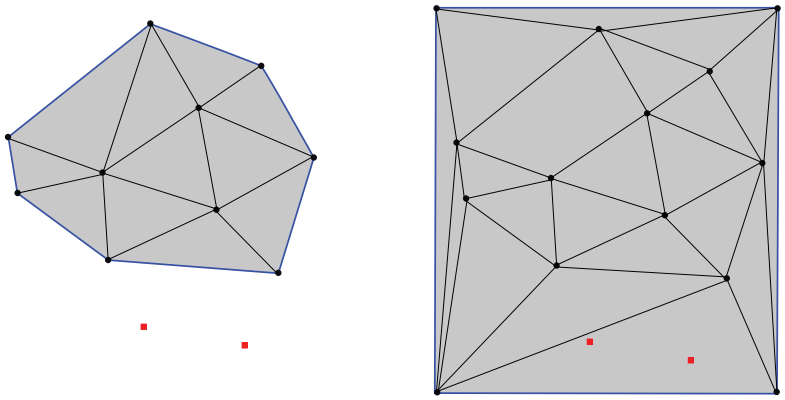


Figure 4.3: Delaunay triangulation D using a point cloud Q before and after adding a bounding box (2d case). Left (resp. Right): D of Q before (resp. after) adding bounding box vertices. Here, we see 3d points Q (black round points), camera viewpoints T (red squared points), triangles of D (black and blue lines), convex hull triangles of D (blue lines), finite tetrahedra of D (grey) and the infinite tetrahedra of D (white)

This special case occurs rarely in our work because camera viewpoins (T) are generally inside the convex hull of Q, as mentioned in the previous **Ray-tracing** paragraph. Nevertheless, to be independent to the acquisition sensor for reasons of generality (e.g. if we use a perspective camera), an extra step can be added to our surface reconstruction in order to avoid this special case. Before ray-tracing, we check that all camera viewpoints of T are inside the convex hull of Q. If it is not the case, a box bounding Q and T is calculated, and the 8 vertices of this bounding box are inserted to D. Fig. 4.3 shows in 2d case D before and after adding bounding box vertices. By doing so, we ensure that all camera viewpoints are located in finite tetrahedra and the ray-tracing can be correctly applied.

4.4 2-manifold Generation

4.4.1 Original (non-manifold) Surface

Until now, one can already reconstruct a surface which is consistent to the visibility information provided by SfM by simply collecting triangles between *free-space* and *matter* tetrahedra. This surface is called the original surface in our work and some previous 3d surface reconstruction methods [93, 112] based on sparse feature points and 3d Delaunay triangulation consider such a surface as the final surface model of their methods.

Notations & Definition Let L be a list of finite tetrahedra of the 3d Delaunay triangulation D. The border δL is the list of triangles of D such that each triangle is included in only one tetrahedron of L. The original surface is δF, where F is the list of all *free-space* tetrahedra in D.

Characteristics of original surface Let us analyze characteristics of the original surface δF. First, as all triangles of δF are embedded in the 3d Delaunay triangulation D, δF is a surface which does not have self-intersections elsewhere at vertices and edges of the surface triangles. Furthermore, δF is closed and watertight, i.e. it encloses the tetrahedra of O.

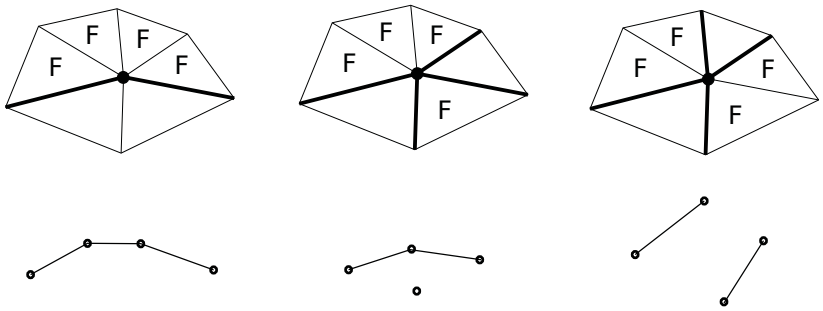


Figure 4.4: Tetrahedra incident to edge e in the original surface δF and the adjacency graph g_e in the 2d case. In the 2d case, edge e is a vertex and tetrahedra are triangles. In top, we see *free-space* tetrahedra (triangles labeled F) and *matter* tetrahedra (triangles with no label). All these tetrahedra contain edge e (round point). We see also triangles (bold segments) in δF which contain e. In bottom, we see the corresponding adjacency graphs of F_e.

Second, an edge e in δF is always incident to two or more than two triangles. This remark can be proved as follows. Let F_e be the list of tetrahedra in F which contain e. Let g_e be the adjacency graph of the tetrahedra in F_e. Here, remember that in an adjacency graph of Delaunay tetrahedra, a graph node represents a tetrahedron and a graph edge represents a triangle separating two adjacent tetrahedra (Sec. 1.5.2). Note that the graph of all tetrahedra around edge e is a cycle, and g_e is included in the cycle. Here, g_e cannot be a complete cycle because tetrahedra containing e cannot be all *free-space*. g_e has (at least) a g_e vertex without edges, or (at least) two g_e vertices with one edge... In all cases, we have

at least two lacking edges in g_e, which correspond to triangles in δF. Fig. 4.4 shows several 2d examples of F_e and g_e.

Regular and singular vertices We distinguish two kinds of vertices in the original surface by using their stars and links (see Definition. 1.9 in Sec. 1.4): the regular and singular vertices.

A vertex $\mathbf{v}$ in a surface is regular if its star in the surface is homeomorphic (see Definition. 1.2 in Sec. 1.1) to a disk. Otherwise, this vertex is singular. If every vertex of a surface is regular, then this surface is a 2-manifold (see Sec. 1.1). Fig. 4.5 shows examples of a regular and two types of singular vertices.

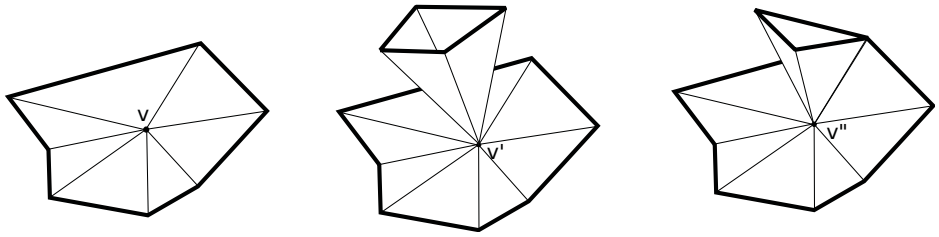


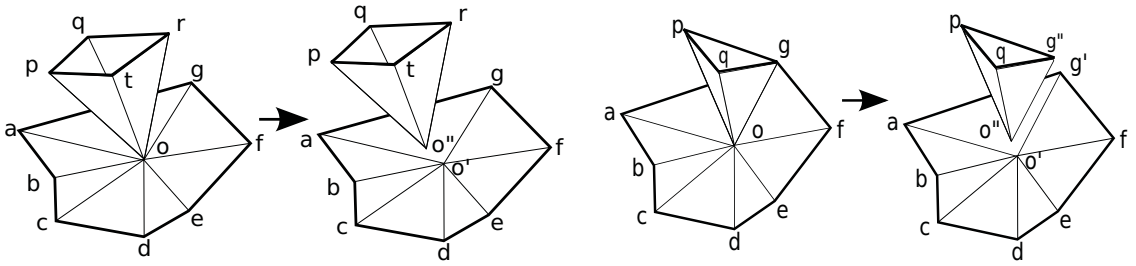
Figure 4.5: Examples of a regular vertex and two singular vertices. $\mathbf{v}$ is regular, $\mathbf{v}'$ and $\mathbf{v}''$ are singular vertices. Here, we see links of vertices (bold solid lines) and edges incident to vertices (standard solid lines).

An important drawback of the original surface δF is that no care has been taken to the topology of the surface model: it can be non-manifold. This is not favorable for additional processing of the surface such as surface fairing or texturing. In contrast, our work generates a 2-manifold instead of the original surface as the final reconstructed surface model. The discussion of why enforcing the reconstructed surface to be 2-manifold is detailed in Sec. 1.4.2.

4.4.2 Heuristics for 2-manifold Generation

4.4.2.1 Tetrahedra Status Inverse

We have tried several heuristics to generate a 2-manifold. A first approach is to try to inverse the tetrahedra labels to reduce the number of singular vertices. If a tetrahedron Δ in D has singular vertices, the method calculates the number of its singular vertices, and inverses the label (*free-space* to *matter* or *matter* to *free-space*) of Δ if this number is reduced. We do this for all tetrahedra which have singular vertices and repeat the process until no more singular vertices could be removed. In practice, in order to reduce the maximum number of singular vertices, tetrahedra having the largest number of singular vertices are tried first. This approach is simple and very easy to implement however its default is obvious: not all singular vertices are guaranteed to be removed and consequently the surface might not be 2-manifold. Furthermore, this approach can easily violate the visibility constraints provided by SfM, and generate absurd results.



(a) Left: first case of a singular vertex, right: O is split to O' and O''

(b) Left: second case of a singular vertex, right: O is split to O' and O'', g is split to g' and g''

Figure 4.6: Examples of vertices splitting.

4.4.2.2 Singular Vertices Splitting

Another method tries to generate the 2-manifold by splitting each singular vertex into several new vertices which are regular. Each new vertex has its star in surface homeomorphic to a disk and its coordinates are the same or close to those of its original singular vertex. Here, two types of singular vertices exist:

1) the singular vertex whose opposite edges form several disconnected polygons, as shown in Fig. 4.6a,

2) the singular vertex whose opposite edges form several polygons which share a common vertex, as shown in Fig. 4.6b.

This idea of splitting singular vertices is also used by other works in bibliography such as [60] and [128], to extract 2-manifold surfaces from a simplicial complex like the original surface δF. However, this approach adds a lot of additional vertices. Furthermore, splitting singular vertices of δF will artificially increase the number of connected components (2-manifolds after splitting).

4.4.3 Region-Growing

We now present the finally chosen 2-manifold generation method: the region-growing.

Remember that our objective is to generate a 2-manifold S based on results of *free-space/matter* labeling and S should separate the *matter* and *free-space* tetrahedra "as much as possible".

Optimization Formulation In fact, our 2-manifold generation can be seen as an optimization problem described as follows. Let $r : F \to \mathbb{R}^{+*}$ be a scalar and positive function. Now imagine another tetrahedron list $O \subseteq F$ and all tetrahedra contained in O are labeled *outside*, i.e. outside the *matter*. All tetrahedra in its complement list $D \setminus O$ are labeled *inside* Now we extend r to O by,

$$r(O) = \sum_{\Delta \in O} r(\Delta). \tag{4.2}$$

In general, the border δO of O is a triangle list and a two-dimensional closed surface (sub-complex of D) as the original surface δF, but it is not a 2-manifold.

Now we would like to find $O \subseteq F$ maximizing $r(O)$ subject to the constraint that δO is a 2-manifold, i.e.

$$O = \arg \max_{\begin{cases} O' \subseteq F \\ \delta O' \text{ is 2-manifold} \end{cases}} r(O'). \tag{4.3}$$

The target surface S is then δO.

Let us summarize the different tetrahedra lists used in our 2-manifold generation step, we have: F=the *free-space* tetrahedra, $D \setminus F$=the *matter* tetrahedra, O=the *outside* tetrahedra and $D \setminus O$=the *inside* tetrahedra. In addition, $O \subseteq F$ thus $D \setminus F \subseteq D \setminus O$.

The optimization problem is difficult due to the manifold constraint. In our work, Eq. 4.3 is solved by a greedy region-growing algorithm: tetrahedra of F are progressively added in O as long as δO remains always a 2-manifold, then $r(O)$ increases and the final O is an approximation of the exact solution of Eq. 4.3. Fig. 4.7 illustrates this region-growing process.

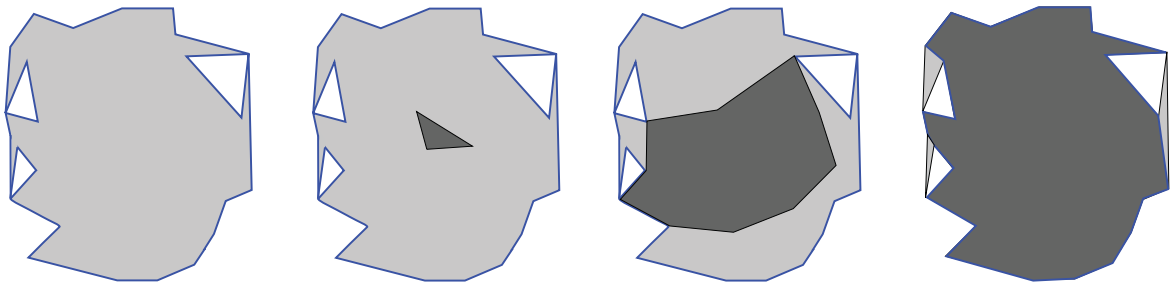


Figure 4.7: The principle of greedy region-growing in the 2d case. Region growing is illustrated from left to right. Here we see, original surface (blue lines), *free-space* region (light gray), *matter* region (white), *outside* region (dark gray) and *inside* region (light gray+white).

Choice of r The number of rays which intersect tetrahedron Δ is chosen as $r(\Delta)$. Thus our region-growing process can be seen as a process which optimizes the visibility consistency provided by SfM, i.e. the number of intersections between tetrahedra and rays. A *free-space* (resp.*matter*) tetrahedron is visibility consistent if it is finally *outside* (resp.*inside*). Other choices of r are also experimented in Sec. 7.2.3.

4.4.3.1 2-Manifold tests

Before presenting our greedy region-growing approach in details, methods which check that a surface S (a list of triangles of the 3d Delaunay triangulation D) is a 2-manifold are presented. A surface S is 2-manifold if and only if all vertices in it are regular (Sec. 4.4.1). This topic is somewhat technical but very important in practice for the computation time of our method.

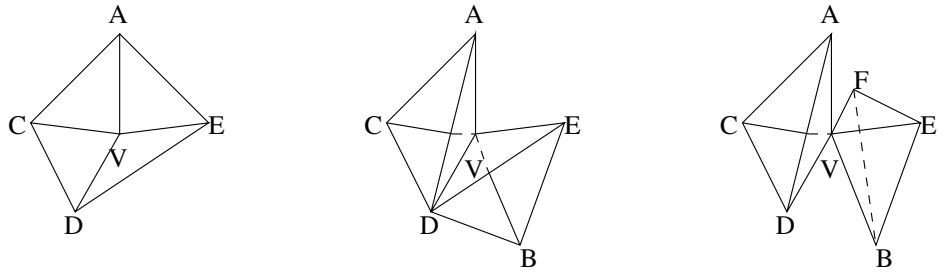


Figure 4.8: Edge-based test of regular and singular vertices. Left: **v** is regular since the edges opposite to (the link of) **v** define a simple polygon **acdea** on the surface. Middle: **v** is singular since polygon **dacdbed** has multiple vertex **d**. Right: **v** is singular since **acda** − **befb** is not connected.

Edge-Based vertex test This test is based on the "Edge-based test" [23] which has already been presented in Sec. 1.4. Vertex **v** in a surface S is regular if and only if the link of **v** is a simple closed polygon, i.e. every vertex is regular. See Fig. 4.8 and also Fig. 4.5 for examples.

Tetrahedron-based vertex test Here we propose another test [1] to check that **v** is regular. Let $g_{\mathbf{v}}$ be the graph of the tetrahedra incident to **v** ($g_{\mathbf{v}}$ is a sub-graph of D). Appendix. B shows that the link of **v** in S above forms a simple polygon if and only if all *inside* tetrahedra of $g_{\mathbf{v}}$ are connected and all *outside* tetrahedra of $g_{\mathbf{v}}$ are connected. See Fig. 4.9 for examples. Thus we check that **v** is regular thanks to a simple traversal of graph $g_{\mathbf{v}}$ where the edges between *inside* and *outside* are removed.

In our implementation, this test is fast since the graph is already encoded in the Delaunay triangulation implementation of CGAL. We do not need to compute the edges/faces from a tetrahedra graph as in the Edge-based vertex test.

Single tetrahedron test A third 2-manifold test is based on [23]. It checks that a single tetrahedron Δ can be added in O such that the border δO of O remains 2-manifold. Assume that δO before adding Δ is 2-manifold. Then Δ can be added in O if a condition based on the neighborhood configuration of Δ is met. Appendix. C gives details of this condition. This 2-manifold test is also very efficient because it only requires reading once the lists of tetrahedra incident to the four Δ-vertices (without graph traversal). The single tetrahedron test cannot be applied if more than one tetrahedron are added at once in O.

4.4.3.2 Greedy region-growing

According to the beginning of Sec. 4.4.3, the solution of Eq. 4.3 is approximated by a greedy approach: an *outside* region O grows from $\emptyset$ by adding *free-space* tetrahedra one-by-one such that δO remains 2-manifold (Sec. 4.4.3.1). The result of this step is the final δO. Each time, the single tetrahedron test in Sec. 4.4.3.1 is used to check if a tetrahedron Δ can be added in O or not.

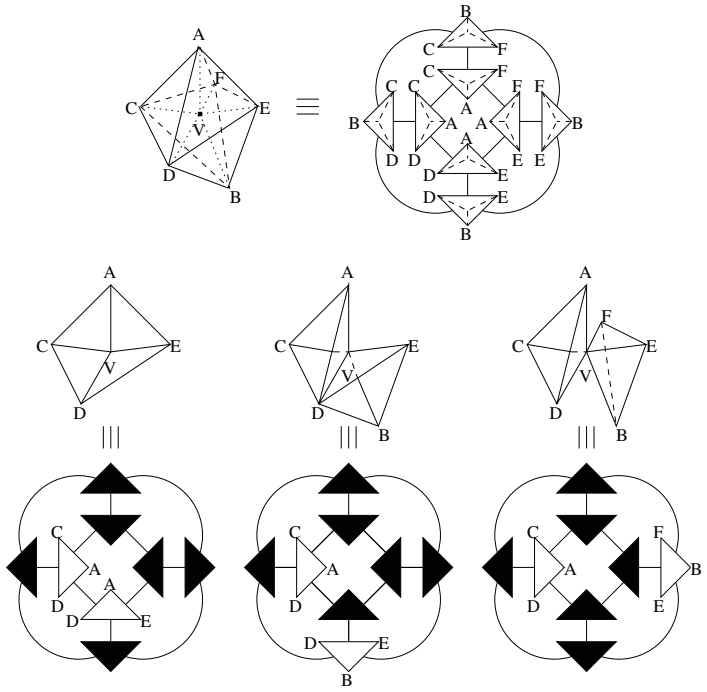


Figure 4.9: Relation between topology and graph. Top: 8 tetrahedra incident to **v** and their adjacency graph $g_{\mathbf{v}}$. Bottom: three cases of tetrahedra labeling (*inside* is black, *outside* is white). Middle: the resulting topology around **v**. Middle and bottom: **v** is regular if and only if the *inside* tetrahedra are connected and the *outside* tetrahedra are connected.

The final δO depends on the adding order of tetrahedra to O. To approximate O of Eq. 4.3, r is used to define a priority for the tetrahedra of F: the Δ with the largest $r(\Delta)$ is added in O before the others. Remember that r is chosen as the number of rays intersected by Δ. Besides, to avoid that the greedy algorithm gets stuck too easily in a bad solution, the selected Δ is required to have at least one triangle shared with δO. So finally, the tetrahedra in the neighborhood of O are stored in a heap (priority queue) for fast selection of the tetrahedron with the greatest r. By doing so, our method tries to propagate the *outside* region from the most visibility consistent tetrahedra to the least ones. A similar propagation algorithm can be found in [90], but used for a very different problem (point matching).

A detailed algorithm (Algorithm. 1) is also given. In this algorithm, O is the *outside* region, F is the set of *free-space* tetrahedra, Q is the priority queue and the function r maps a tetrahedron Δ to the number of rays intersected with Δ. Q_0 is the set of tetrahedra to initialize Q (useful for Topology Extension presented in Sec. 4.5).

Note that a *free-space* tetrahedron Δ can be tried to be added to O for several times, but there is no infinite loop in our region-growing process. This is shown in Sec. 6.2.2.4 (complexity analysis).

Algorithm 1: Greedy region-growing

Inputs : O, F, Q_0
Outputs: O
Step1: Initialization
$Q = \emptyset$;
if $O == \emptyset$ **then**
  Let $\Delta \in F$ be such that $r(\Delta)$ is maximum;
  $Q \leftarrow Q \cup \{\Delta\}$;
else
  //Used by topology extension
  For each tetrahedron $\Delta \in Q_0 \cap F$ **:**
    if $\Delta \notin O$ *and* Δ has an adjacent tetrahedron in O **then**
      $Q \leftarrow Q \cup \{\Delta\}$;

Step2: Region-growing by adding tetrahedra one-by-one
while $Q \neq \emptyset$ **do**
  Pick from Q the Δ which has the largest $r(\Delta)$;
  if $\Delta \in O$ **then**
    continue;
  $O \leftarrow O \cup \{\Delta\}$;
  if all vertices of Δ are regular **then**
    For each tetrahedron Δ' adjacent to Δ **:**
      if $\Delta' \in F$ && $\Delta' \notin O$ **then**
        $Q \leftarrow Q \cup \{\Delta'\}$;
  else
    $O \leftarrow O \setminus \{\Delta\}$;

4.4.3.3 Drawbacks of greedy region-growing

Our greedy region-growing algorithm grows an *outside* region O by adding tetrahedra one by one under the condition that the border δO of O is 2-manifold. A drawback of our algorithm is that the resulting 2-manifold has always genus 0. As tetrahedra are added one-by-one in O, the genus of the resulting 2-manifold δO cannot be changed, i.e. δO has the ball topology [23]. This is problematic if the true *outside* does not have the ball topology, e.g. if the camera trajectory contains closed loop(s) around building(s). In the simplest case of one loop, the true *outside* has the toroid topology and the computed *outside* O cannot close the loop (see Fig. 4.10).

Another drawback is the shelling stuck problem. We assume that there is an *outside* region $\widehat{O}$ that we want to reach using our greedy region-growing algorithm. In discrete computational geometry field, our algorithm and also the sculpture algorithm of [23] can

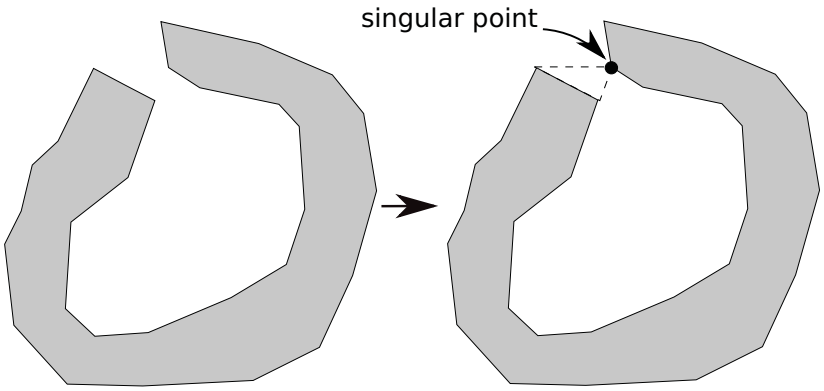


Figure 4.10: Greedy region-growing one-by-one. We can see that growing O by adding tetrahedra one-by-one cannot change the topology of O.

be seen as a process to find a shelling of $\widehat{O}$ [44]. Here, a shelling means an ordering of the tetrahedra such that every prefix defines a 3-ball, i.e. the border of every prefix is connected 2-manifold with zero genus. [151] shows that some 3-ball $\widehat{O}$ embedded in a 3d Delaunay triangulation does not have any shelling, i.e. $\widehat{O}$ can never be reached by a greedy algorithm whatever the order of tetrahedron adding.

4.5 Topology Extension

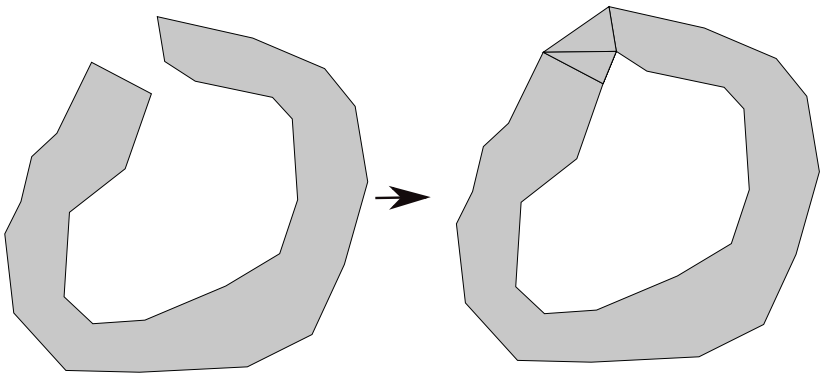


Figure 4.11: Topology extension. We can see that growing O by multi tetrahedra incident to the same vertex can change the topology of O.

Dealing with the drawback of the greedy region-growing, a topology extension process is proposed to extend the topology of the *outside* region O. It can be described as follows.

1. For a vertex in δO, if all *inside* tetrahedra incident to this vertex are *free-space*, then let these tetrahedra be A and force them to be *outside*, i.e. we add A in O. Now we check that all vertices of A regular by using the tetrahedron-based vertex test (Sec. 4.4.3.1).

If there is at least one singular vertex, A are restored to *inside*, i.e. we remove A from O.

2. If Step 1 succeeds, the greedy region growing algorithm (Algorithm. 1) in Sec. 4.4 is applied. Here, Q_0 in Algorithm. 1 in this case is the list of tetrahedra adjacent to A.

In practice, Step 1 and Step 2 are alternated several times for all vertices of δO. One can see that thanks to this topology extension process, the topology types of 2-manifold can be changed and it makes our environment modeling method more flexible. Besides, when the greedy region-growing process gets stuck, the topology extension may restart the greedy region-growing. As a result, the *outside/free-space* ratio is improved. However, it should be noted that we cannot theoretically guarantee that all possible 2-manifold surfaces embedded in the 3d Delaunay triangulation can be generated.

4.6 Spurious Handle Removal

Topology Extension (Sec. 4.5) calculates a 2-manifold without genus limitation and improves our approximation of the Eq. 4.3 solution, but it has one drawback: it can generates spurious handles.

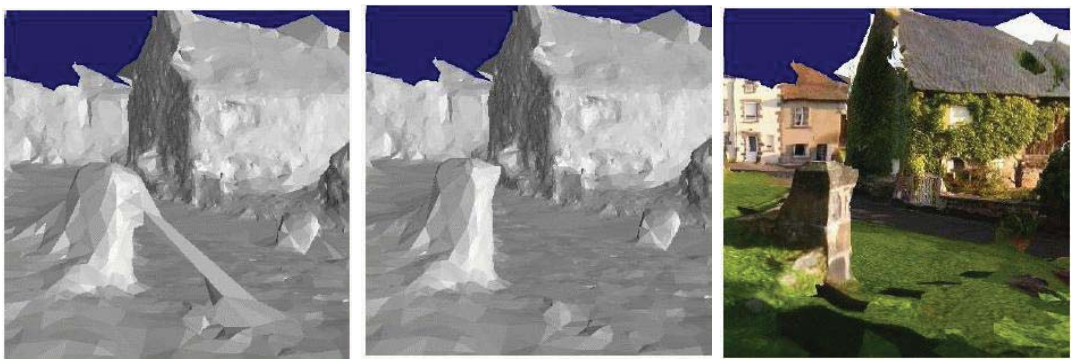


Figure 4.12: Spurious handle (left) and its removal (middle and right).

Fig. 4.12 shows an example in a real case: the oblique handle on the left connects a small wall to the ground. This handle is spurious: it does not exist on the true scene surface and it should be removed while the manifold property should still be maintained.

In our work, the spurious handles removal deals only with handles which are both "visually critical" and due to "incomplete" *outside* growing in the *free-space*. "Visually critical" means that we ignore the handles that are too small to be easily noticeable by an observer (virtual pedestrians) located at all camera viewpoints reconstructed by SfM. "Incomplete" means that the handles only contain *free-space* tetrahedra that are *inside* and which should be forced to *outside*. These conditions are used to localize the spurious handles and to obtain a final 2-manifold which meets the visibility constraints provided by SfM. For the remainder of the reading, a "spurious handle that is visually critical and due to incomplete growing" is shortened by "spurious handle".

Two spurious handles removal methods are presented here. Both methods use Steiner vertices, i.e. extra points added in D. These points are not in the original (SfM) input, thus no ray is attached to them.

4.6.1 Method 1: Reduce Size of Tetrahedra

A simple method is used to reduce the risk of spurious handles in the beginning of our work [3]. In the 3d Delaunay step (Sec. 4.2), Steiner points are added in D such that the long tetrahedra potentially involved in spurious handles are split in smaller tetrahedra. The Steiner points are added in the critical region for visualization: the immediate neighborhood of the camera trajectory. For each camera location $\mathbf{t}^j$, a fixed and small number (2 in practice) of Steiner vertices are randomly added inside a ball region centered at $\mathbf{t}^j$. The radius of the ball is defined as a multiple (e.g. 10) of $\text{mean}_j||\mathbf{t}^{j+1} - \mathbf{t}^j||$.

Note that the addition of Steiner vertices increases the number of tetrahedra, while the number of ray is unchanged. The risk is to obtain a ratio #tetrahedra/#rays (#L means the number of tetrahedra in L) which is locally too large such that *matter* tetrahedra can appear. However this risk is very low since we add Steiner vertices in the area which has a high density of rays.

4.6.2 Method 2: Detect, Force and Repair

The first method is a simple way to avoid spurious handles in critical regions, however spurious handles can still exist. Another more efficient removal method is developed in our latter work [5]. It has three steps: Detect, Force and Repair which are described in the following sections.

4.6.2.1 Detect

Let e be an edge with finite vertices $\mathbf{a}_e$ and $\mathbf{b}_e$. Let $\alpha > 0$ be a threshold. Edge e is "visually critical" if the three following conditions are met:

1. every tetrahedron including e is *free-space*
2. at least one *inside* tetrahedron includes e
3. there exists a view point $\mathbf{t}^j$ such that angle $\widehat{\mathbf{a}_e\mathbf{t}^j\mathbf{b}_e}$ is greater than α.

As showed in Fig. 4.13, a spurious handle may have critical edges both on its border and its interior. All visually critical edges are stored in a list L_α. The larger α, the smaller size of L_α and also the greater lengths of the edges in L_α. Finally, a moderated number of handles are selected thanks to $\alpha > 0$.

4.6.2.2 Force

This step needs a mesh operator called "Edge Splitting". It splits an edge of D by adding a Steiner vertex at the middle of the edge, and every tetrahedron including the edge is split in

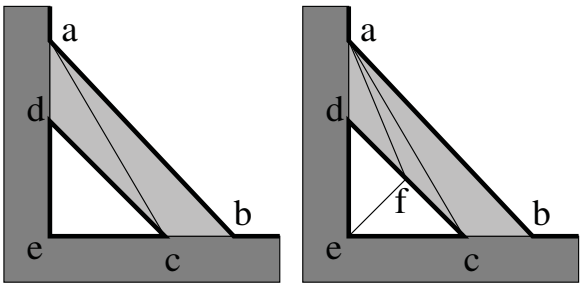


Figure 4.13: Spurious handle (left) and edge splitting (right) in the 2d case. Here we see, manifold edges (bold lines), Delaunay edges (simple lines), *outside* region (white), *inside-freespace* region (light gray), and *inside-matter* (dark gray). Left: edges **ab**, **cd**, **ac** are visually critical. Right: Steiner point **f** is inserted on **cd**, this splits triangle **cde** and **adc**.

two tetrahedra whose labels (*free-space*, *matter*, *inside*, *outside*) are the same as the original tetrahedra. See an example in Fig. 4.13. The resulting triangulation D is not 100% Delaunay after edge splitting, but the surface which separates *outside* and *inside* tetrahedra is still a 2-manifold (the set of surface points is the same).

The edge splitting is applied to every edge in the list L_α. This changes the graph of tetrahedra and provides new tetrahedra for the next step, which is a local region growing of *outside* O in *free-space* tetrahedra F. For each vertex $\mathbf{v}_i$ on (split) critical edges, a list L_i is defined which collects all *free-space* and *inside* tetrahedra incident to $\mathbf{v}_i$. Now a first round of (Force,Repair) is applied to the list L_i. Function "Force" adds tetrahedra in O without checking that δO is a 2-manifold, and function "Repair" (Sec. 4.6.2.3) tries to grow O such that δO becomes a 2-manifold. If the first round fails, another round of (Force,Repair) is tried for tetrahedra of L_i one-by-one.

Algorithm. 2 shows the Force and Repair process.

Algorithm 2: Force and Repair

```
For each vertex v_i on every edge e ∈ L_α :
    Let L_i be the list of inside and free-space tetrahedra incident to v_i;
    G = L_i;
    O ← O ∪ G; // Force
    if !Repair(G, O) then
        For each tetrahedron Δ ∈ L_i :
            G = {Δ};
            O ← O ∪ G; // Force
            Repair(G, O);
```

4.6.2.3 Repair

The Repair step is a local growing of O in F such that the number n_s of singular vertices of surface δO decreases and every regular vertex is maintained regular. At the beginning, $n_s > 0$ is due to the addition of G in O (Force). Then a *free-space* and *inside* tetrahedron Δ is chosen in the neighbors of G, and it is added in O. If n_s increases, Δ is removed from O and another tetrahedron is tried. The process stops when every Δ candidate increases n_s. If the final n_s is 0, the local region growing succeeds. Otherwise it fails and O is restored to its value before the Force step.

Algorithm. 3 shows the Repair algorithm. The inputs are O, G and g_{max} (an upper bound to limit the local growing complexity). The output is O and a boolean which asserts that Repair is successful or failed.

Note that this algorithm looks like the greedy region-growing algorithm Algorithm. 1 in Sec. 4.4. Its main differences compared to Algorithm. 1 are: (1) the 2-manifold tests are replaced by non-increase tests of the number of singular vertices (2) the process can fail.

Algorithm 3: Repair

Inputs : O, G, g_{max}
Outputs: boolean b, O
Let n_s be the number of singular vertices of G;
Create a priority queue $Q = \emptyset$;
For each tetrahedron $\Delta \in G$ **:**
  For each tetrahedron Δ' adjacent to Δ **:**
    if $\Delta' \in F$ && $\Delta' \notin O$ **then**
      $Q \leftarrow Q \cup \Delta'$;
while $Q! = \emptyset$ **do**
  Pick from Q the tetrahedron Δ which has the largest $r(\Delta)$;
  if $\Delta \in O$ **then**
    continue;
  Let b_i^0 be true if and only if the i-th vertex of Δ is singular in δO;
  $n_0 = \sum_{i=1}^{4} b_i^0$; *//number of singular Δ-vertices*;
  $O \leftarrow O \cup \{\Delta\}$;
  Let b_i^1 be true if and only if the i-th vertex of Δ is singular in δO;
  $n_1 = \sum_{i=1}^{4} b_i^1$; *//number of singular Δ-vertices*;
  if $n_0 \geq n_1$ && $b_1^0 \geq b_1^1$ && $b_2^0 \geq b_2^1$&& $b_3^0 \geq b_3^1$ &&$b_4^0 \geq b_4^1$ **then**
    $G \leftarrow G \cup \{\Delta\}$; *//G can be used at the growing end*
    $n_s \leftarrow n_s + n_1 - n_0$; *// fast n update*
    if the size of G is g_{max} **then**
      break; *// too large computation: stop*
    For each tetrahedron Δ' adjacent to Δ **:**
      if $\Delta' \in F$ && $\Delta' \notin O$ **then**
        $Q \leftarrow Q \cup \{\Delta'\}$;
  else
    $O \leftarrow O \setminus \{\Delta\}$;
if $0 == n_s$ **then**
  return **true**;
else
  $O \leftarrow O \setminus G$;
  return **false**;

4.7 Post-processing

The reconstructed surface S has still several weaknesses which can be easily noticed during visualization. This section shows a list of post-processings used in our work, dealing with these weaknesses.

Here is a list of these post-processings:

1. Peak removal.

   Sharp vertices which are not physically plausible are removed. It can also reduce effects of bad (false positive) SfM points, which will be discussed in Sec. 4.8.

2. Surface denoising.

   The noise of the estimated 2-manifold surface is reduced.

3. Removal of "sky" triangles.

   Part of the surface corresponding to the sky (assuming outdoor sequences) is removed. It facilitates the model visualization.

4. Texturing.

   Assign textures extracted from input images to the reconstructed surface.

4.7.1 Peak Removal

On the reconstructed 2-manifold, some peaks can exist. Fig. 4.14 illustrates several examples of this kind of peaks. These peaks are not physically plausible, i.e. they do not recover any part of the scene, and should be removed.

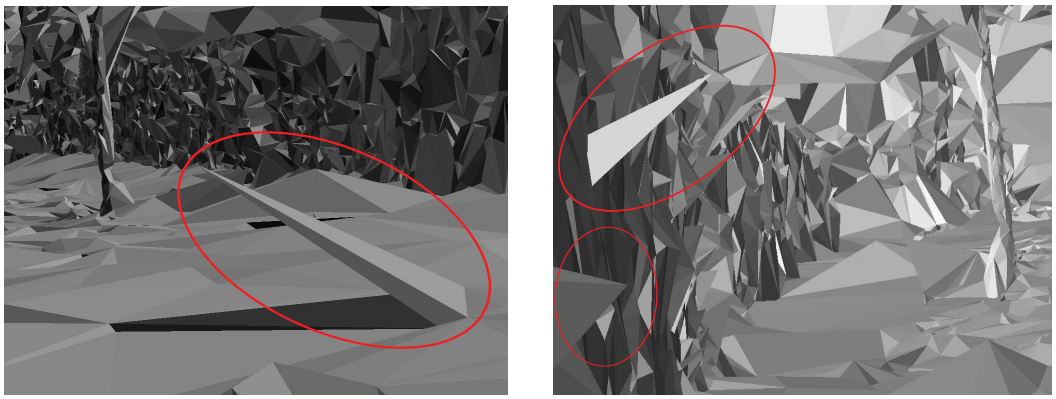


Figure 4.14: Several examples of peaks (surrounded by red lines).

A peak is a vertex $\mathbf{v}$ in S such that the ring of its incident triangles in S defines a solid angle w which is too small to be physically plausible, i.e. $w < w_0$ where w_0 is a threshold. Let L be the list of the tetrahedra in the acute side of the peak ring. All tetrahedra of L have the same label: *outside* or *inside*. Then w can be calculated by summing the solid angle of

$\mathbf{v}$ in each tetrahedron of L. Appendix. D shows how to calculate the solid angle of a vertex in a tetrahedron.

Now if $\mathbf{v}$ is a peak, i.e. $w < w_0$, then it can be removed by inversing the labels of tetrahedra in L: *inside* becomes *outside*, and vice versa. The inversing is successful if all vertices of the tetrahedra in L remain regular. Otherwise, the L labels are restored to original values and another peak is tried. In practice, the peak removal is done by going through several times the list of S vertices. Note that this step does not take into account the *free-space/matter* labeling of tetrahedra.

4.7.2 Surface Denoising

Surface denoising is usually a necessary process for a 3d modeling. Whatever the acquisition captor is (laser, Kinect, camera etc.), and whatever the reconstruction method is (dense stereo or structure from motion), the 3d points have always noise. Considering the surface parameterized by function $f : \Omega \to \mathbb{R}^3$, where $\Omega \subset \mathbb{R}^2$, the denoising process of a mesh is generally a process to remove high-frequency noise of f. Thanks to the 2-manifold property of our surface model, the surface denoising can be easily processed according to Sec. 1.4.2.

Uniform Laplacian Flow Operator A standard polygonal mesh denoising method is used to denoise the reconstructed 2-manifold S: the uniform Laplacian flow operator [135] [78]. It should be noted that there exist other more sophisticated operators such as Laplace-Beltrami operator [27] etc. Nevertheless, the uniform Laplacian operator is preferred in our work thanks to its simpleness and stability.

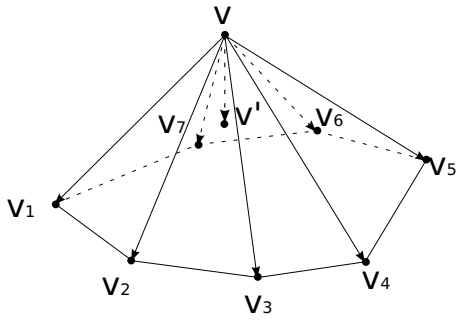


Figure 4.15: Umbrella operator. $\mathbf{v}$ is the vertex of the previous iteration, $\mathbf{v_1}, \cdots, \mathbf{v_7}$ are one-ring neighbor vertices of $\mathbf{v}$ in the previous iteration, $\mathbf{v}'$ is the updated vertex of $\mathbf{v}$ in the current iteration

Let $\mathbf{v}$ be the original vertex in S, $\mathbf{v}'$ be the denoised vertex of $\mathbf{v}$, $\mathcal{N}(\mathbf{v}) = \{\mathbf{v_1}, \mathbf{v_2}, \cdots, \mathbf{v_n}\}$ be the list of one-ring neighbor vertices (neighbor vertices which are connected to $\mathbf{v}$ with one edge) of $\mathbf{v}$ on S and n is its size. We have,

$$\mathbf{v}' = \mathbf{v} + \lambda \Delta \mathbf{v} \tag{4.4}$$

where λ is a constant (0.8 in practice) and $\Delta\mathbf{v}$ is the displacement vector, also called the

umbrella operator (see Fig. 4.15). $\Delta\mathbf{v}$ is defined as follows,

$$\Delta\mathbf{v} = \frac{1}{n} \sum_{\mathbf{v_i} \in \mathcal{N}(\mathbf{v})} (\mathbf{v_i} - \mathbf{v}) \tag{4.5}$$

Advantages and Drawbacks There are several advantages by using uniform Laplacian flow operator for surface denoising. It is very simple to use and triangle forms are also improved: minimal angles of "thin" triangles become generally greater after surface denoising and their forms become more regular [27].

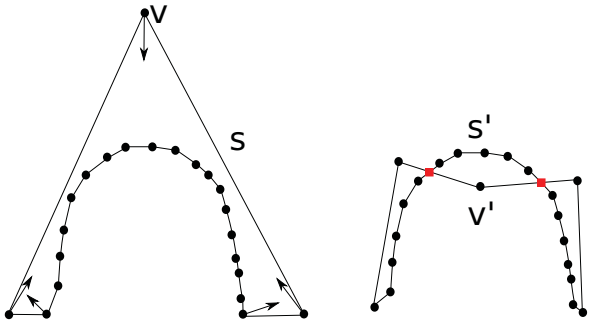


Figure 4.16: An example of non-manifold after surface denoising in 2d case. Left and right are respectively 2-manifold surface S before denoising and the denoised surface S'. We can see the vertices (round black points) in S, S', the displacement vectors (arrows) of several vertices, and self-intersections (red squared points) of S'. $\mathbf{v}$ is a vertex in S which is very distant to its connected vertices, $\mathbf{v}'$ is the denoised point of $\mathbf{v}$.

Regarding the drawbacks, firstly surfaces with high curvatures are over-smoothed. Indeed, the uniform Laplacian flow operator is a low-pass filter. Furthermore, as denoised vertices are displaced, the 2-manifold might become non manifold. See Fig. 4.16 for an example. In this example, the surface obtained after surface denoising has self-intersections thus it is non-manifold. Nevertheless, we want the reconstructed surface to be 2-manifold in our work so that post-processings such as surface denoising can be applied. Thus we only guarantee the surface to be 2-manifold before surface denoising.

4.7.3 Removal of "Sky" Triangles

Our surface reconstruction method reconstructs the whole scene including the "sky" in case of outdoor scenes. The "sky" part of the reconstructed surface can be seen as hallucinations (which does not correspond to real surfaces) or highly erroneous part of the reconstructed surface. In fact, the sky is usually far from the camera and the corresponding reconstructed 3d points are highly erroneous. Besides, as our final surface is a closed mesh, removing part of the mesh allows a better visualization of the model in a bird view point, e.g. see Fig. 4.17. So in our work, the "sky" part is removed off the reconstructed surface. Saying about "remove", the "sky" triangles are not really removed from the surface. They are only detected and turned to be invisible for visualization.

The "sky" triangles removal is performed as follows:

Figure 4.17: Bird view of textured models with and without sky. Left: model with sky. Right: model without sky

1. We use the z-axis of our catadioptric camera (Sec. 3.1.1.2) to calculate the sky direction. Let $\mathbf{n}^i$ be world coordinates of the z-axis of the catadioptric camera for camera pose $\mathbf{c}^i$. Then the sky direction is simply: $\frac{1}{m}\sum_{i=1}^{m}\mathbf{n}^i$.

2. A list of open rectangles are defined by the finite edge $\mathbf{t}^i\mathbf{t}^{i+1}$ and the two infinite edges (half lines) starting from $\mathbf{t}^i$ (or $\mathbf{t}^{i+1}$) with sky direction. A triangle of S which is intersected by an open rectangle is marked as "sky" triangle.

3. "Sky" triangles are iteratively propagated on the S in a fixed number of iterations. In each iteration, for each "sky" triangle, its neighbor triangle t will be marked as "sky" triangle under condition that t is on S and the angle between the "sky" direction and the normal direction of t is smaller than a threshold β. The number of iterations is limited and in practice, 5 iterations are sufficient.

4.7.4 Texturing

Last, the texture should be defined for each triangle of S. A simplified version of [83] is used here, which gets "as is" the texture of a well chosen view point $\mathbf{t}^j$ for each triangle s_i of S. In our case where the image sequence has hundreds or even thousands of images (not dozens), a list of candidates for s_i are pre-selected. A camera viewpoint $\mathbf{t}^j$ is selected as a candidate if it meets the following conditions: 1) the triangle is entirely projected in the image taken at $\mathbf{t}^j$ (the triangle is split if it is too large); 2) s_i is not occluded by another triangle of S; 3) $\mathbf{t}^j$ provides one of the k-largest solid angle for s_i, where k is a threshold.

4.8 Discussion on Bad (False Positive) SfM Points

SfM can reconstruct bad point $\mathbf{q}$ due to repetitive texture or image noise. Now we discuss the consequences and give solutions for this problem. If $\mathbf{q}$ is in the true matter of the scene, some tetrahedra which should be *matter* are labeled *free-space* due to bad ray of $\mathbf{p}$. If these tetrahedra are in O, δO can be corrupted, i.e. a wall with a spurious concavity. In practice,

the risk of bad **p** is low thanks to the SfM machinery (bundle adjustment, RANSAC, robust matching, interest point detectors).

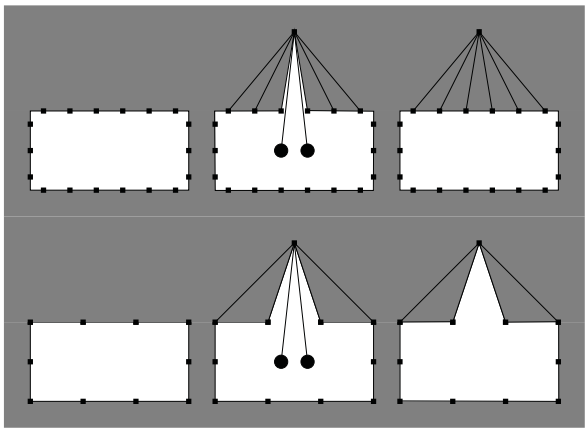


Figure 4.18: Peak removal can remove spurious concavity (2d case). Outside is white, inside is gray. Points and camera locations are black squares and disks, respectively. The density of good points on the surface is high (top) or low (bottom). Left: no bad point. Middle: one bad point adds a spurious concavity due to its two rays. Right: Peak Removal removes the concavity (which is a peak) if it has a small solid angle.

Nevertheless, our method also reduces this risk or even repairs spurious concavities. First, the risk is reduced thanks to the choice of ϵ in the point selection step (Sec. 4.2): the larger ϵ, the more accurate points and their rays used by Ray-Tracing (Sec. 4.3), the lower risk of spurious concavity. Second, the spurious concavities created by the manifold constraint in the growing steps (Sec. 4.4 and 4.5) cannot be worse (larger) than those of the *free-space* in the *matter*. Indeed, a concavity is a list of *outside* tetrahedra which are included in *free-space*. Third, Fig. 4.18 explains how Peak Removal (Sec. 4.7.1) can remove a spurious concavity due to bad point **p**. Here **p** carves a peak due to its rays, the peak apex is **p** and the acute side of the peak is the spurious concavity. The larger density of good points, the smaller solid angle of the peak, the better success rate of the spurious concavity removal.

4.9 Conclusion

In this chapter, we have presented a batch surface reconstruction method using SfM output: a sparse cloud of reconstructed Harris points and their visibilities in the images. The potential adjacencies between the points are encoded in a 3d Delaunay triangulation, then the point visibilities are used to label the tetrahedra, at last a 2-manifold is extracted thanks to a greedy method and the use of Steiner vertices.

Our contributions include,

- a region-growing method [3, 6] which generates a 2-manifold of environments, and optimizes the visibility consistency provided by SfM.

- two methods [5] which remove artifacts called "spurious handles" in the estimated 2-manifold surface,
- an efficient tetrahedron-based test [1] which checks if a surface is 2-manifold (Sec. 4.4.3.1).

Chapter 5

Incremental Surface Reconstruction

5.1 Introduction

The incremental surface reconstruction method estimates incrementally a 2-manifold surface of the environment using sparse 3d points progressively provided by our incremental SfM method.

Given a video sequence, the incremental SfM algorithm described in Sec. 3.3 continuously selects key-frames and estimates the current geometry information of environments each time a key-frame is selected. The estimated geometry for each key-frame includes the camera pose (rotation+translation) corresponding to the current image, new 3d points and new visibility rays between camera viewpoints and 3d points. To simplify, an "image" means a key-frame in the following scope of this chapter. Each time the geometry of the current image is computed, it is then used by our incremental surface reconstruction to update the current reconstructed surface of environments. Here the index of image t is used to represent the time.

In our work, "incremental" means that a surface obtained before time $t+1$ is locally updated by using geometry provided at time $t+1$ to obtain the surface model at $t+1$. In the ideal case, the processing time for each surface update at t should be independent to t, i.e. $\mathcal{O}(1)$ in terms of time complexity. Specifically, when t increases, the processing time of each surface update should not increase.

Here our incremental surface reconstruction method is briefly summarized. Like other standard real time or incremental algorithms, it begins by a short initialization phase and then passes to its principal incremental phase. In the initialization phase, the surface is estimated as in the batch case (Chap. 4) for the first three images. And in the incremental phase, the following steps are sequentially applied each time the geometry of a new image is available,

1. incremental 3d Delaunay triangulation (Sec. 5.2)
2. local *free-space/matter* labeling (Sec. 5.3)
3. incremental 2-manifold extraction (Sec. 5.4)

4. incremental topology extension (Sec. 5.5)

5. incremental post-processing (Sec. 5.6)

Now details of these steps are described in the followings sections.

5.2 Incremental 3d Delaunay triangulation

The first step of our incremental surface reconstruction is the incremental 3d Delaunay triangulation of 3d points. At time $t = 1$, 3d points of the first three images are used to build the initial Delaunay. Note that in our work, time t starts from 1 (not 0), i.e. $t = 1$ means the beginning of the sequence. Then for other images, the Delaunay at time t ($t > 1$) is obtained by updating Delaunay at $t - 1$ with the SfM points which become stable at t. We use an incremental Delaunay triangulation implementation [7], which adds points one by one to build the Delaunay triangulation.

5.2.1 Point Selection

In contrast to the work [93], our incremental Delaunay triangulation step does not add 3d points newly reconstructed at the current time t but the 3d points which become stable at t (see description below). Indeed, the Delaunay triangulation at time t is not totally synchronized with the current geometry, however the update of Delaunay triangulation is much more efficient from a computational viewpoint.

What means "a point becomes stable at t"? A 3d point $\mathbf{q}$ becomes stable at t if and only if $\mathbf{q}$ is no longer modified by the incremental SfM at and after t. Assume that the Local Bundle Adjustment (LBA) of incremental SfM refines the local geometry of the $n_l = 3$ last images (as in [104]). It implies that $\mathbf{q}$ should be the point which is no longer detected/tracked starting from $t - 1$. See Fig. 5.1 for examples of stable points and instable points at t.

Method Here we explain why the Delaunay should be updated with stable points. During the incremental SfM algorithm (Sec. 3.3), new 3d points are initialized and refined at time t. Remember that at the same time, some old points which are initialized in previous images $(t - 1, t - 2, ..)$ are also refined by LBA and some of them may become outliers after LBA. Let Q_t be all these unstable points (old refined points and outliers). Now if the Delaunay D_{t-1} at $t-1$ is updated to the Delaunay D_t at t with Q_t, then the old refined points should be removed and re-added, and the outliers should be removed from D_{t-1}.

Furthermore, it will be problematic for the subsequent ray-tracing process. Each tetrahedron Δ should hold a list $L(\Delta)$ of rays intersected with Δ. If a ray of $L(\Delta)$ is modified after LBA, then we should recheck that this ray intersects Δ. The computation time for large-scale scene can be high and experiments of work [93] show that if the size of $L(\Delta)$ is not manually bounded, the processing time for each image increases when t increases.

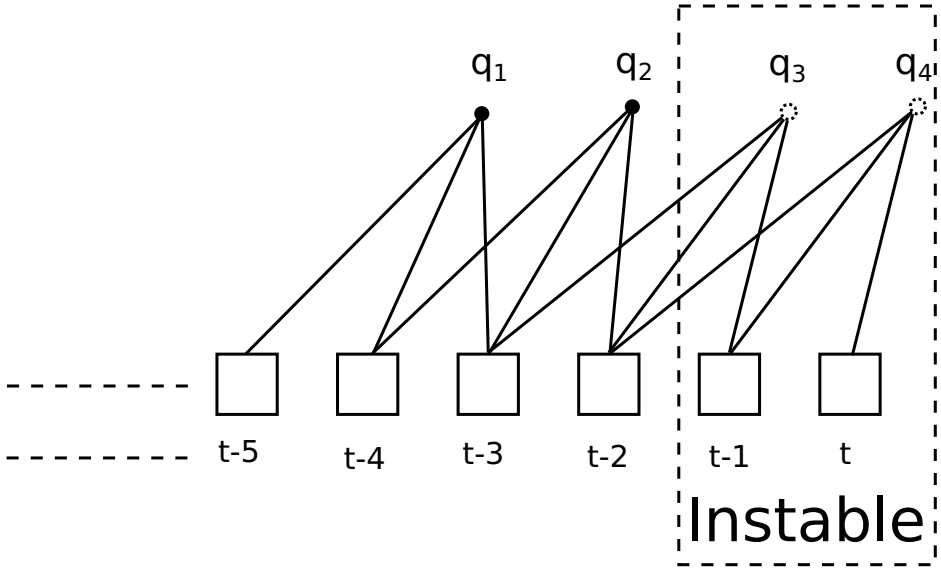


Figure 5.1: Stable and instable points at time t. At time t, Local Bundle Adjustment (LBA) is applied at $t, t-1, t-2$. Then $\mathbf{q}_1$ has already become stable at $t-1$, $\mathbf{q}_2$ becomes stable at t, $\mathbf{q}_3, \mathbf{q}_4$ are instable at t (they will be refined by LBA at time $t+1$).

After selecting the points which become stable at t, a point filter is applied to the selected points as in the batch case (Sec. 4.2): a point $\mathbf{q}$ will not be added to Delaunay if angles between all pairs of rays associated to $\mathbf{q}$ are less than a threshold ϵ (5 or 10 degrees in practice) or greater than $\pi - \epsilon$. Besides, a fixed number (2 in practice) of Steiner points are added in the neighborhood of the current camera viewpoint $\mathbf{t}^t$ to reduce the risk of spurious handles, as in the batch case Sec. 4.6.1.

5.2.2 Dating

A creation date is assigned to each tetrahedron and each vertex of the current Delaunay D_t. The dating process is necessary for all further steps (local ray-tracing, incremental 2-manifold generation...) and can be easily done thanks to the Delaunay implementation of CGAL [7].

When a point $\mathbf{q}$ is added to the Delaunay at time t, a list $L_d(\mathbf{q})$ of tetrahedra are destroyed and a list $L_c(\mathbf{q})$ of tetrahedra are created. So the date t is assigned to $\mathbf{q}$ and every tetrahedron of $L_c(\mathbf{q})$. By doing so, the Delaunay is finally partitioned to a list of temporal layers of tetrahedra created at different times. Here, we also note the smallest creation date of tetrahedra in L_d as d_t, which will be useful in our further 2-manifold generation step (Sec. 5.4).

By adding 3d points progressively along the camera trajectory and dating the Delaunay triangulation, a direct link is made between the spatial and the temporal information of 3d points and tetrahedra in Delaunay. We have:

1. From a global viewpoint, 3d points and tetrahedra with similar creation dates are

usually incident.

2. From a local viewpoint, incident tetrahedra and points have often similar creation dates.

There exist exceptions. When loops exist in the camera trajectory, the first observation is still valid however the second is false. More precisely, in the area where a loop is closed, a point or tetrahedron with late creation date (loop end) can be neighbor to points or tetrahedra created early (loop beginning).

5.3 Local Ray-tracing

In the batch version, a ray-tracing algorithm is used to label *free-space* and *matter* tetrahedra. It is still used by our incremental surface reconstruction method, except that not all rays are used. In fact, using all rays of each image is too time consuming and the complexity is at least linear to the number of rays, which is also linear to the time.

A process called the local ray-tracing is used in the incremental case. Only part of rays are used by this process: at time t, the ray-tracing is only applied to the rays with creation dates in $\{t-k+1, \cdots, t-1, t\}$ (the creation date of a ray is the one of the point to which this ray is attached), where k is a threshold.

We can show that results of all local ray-tracings are almost the same to results of global ray-tracing. The justification is given in following paragraphs.

Justification Let D_{new} be the list of tetrahedra which are created at t (their creation date is t). The other tetrahedra in D are $D \setminus D_{new}$, noted as D_{old}. We note that all tetrahedra of D_{old} have a creation date less than t. Let R_{new} be the list of rays used for local ray-tracing (their creation dates are in $\{t-k+1, \cdots, t-1, t\}$), and other rays are R_{old}, which have creation dates less than $t-k+1$.

1. If the batch ray-tracing is used, all tetrahedra, i.e. $D_{new} \cup D_{old}$ are tested with all rays $R_{new} \cup R_{old}$ to check their intersections. It means D_{new} should be tested with R_{new} and R_{old}, and D_{old} should also be tested with R_{new} and R_{old}.

2. Now if local ray-tracing is used, the ray-tracing is only applied to R_{new}, i.e. D_{new} and D_{old} are checked with R_{new}. Besides, D_{old} are already tested with R_{old} in previous local ray-tracings.

Finally, the only difference between the local and global ray-tracing is that D_{new} are not tested with R_{old}.

In general, it is useless to test D_{new} with R_{old}. In fact, if the camera trajectory does not contain loops, R_{old} have almost no intersection with D_{new} if k is large enough. As showed in Fig. 5.2, when $k = 1$, D_{new} have intersections with R_{old}. The number of intersections decreases when k increases to 2 and becomes 0 when $k = 3$. In practice, $k = 40$ is used in the local ray-tracing, and almost the same results to the batch ray-tracing are obtained.

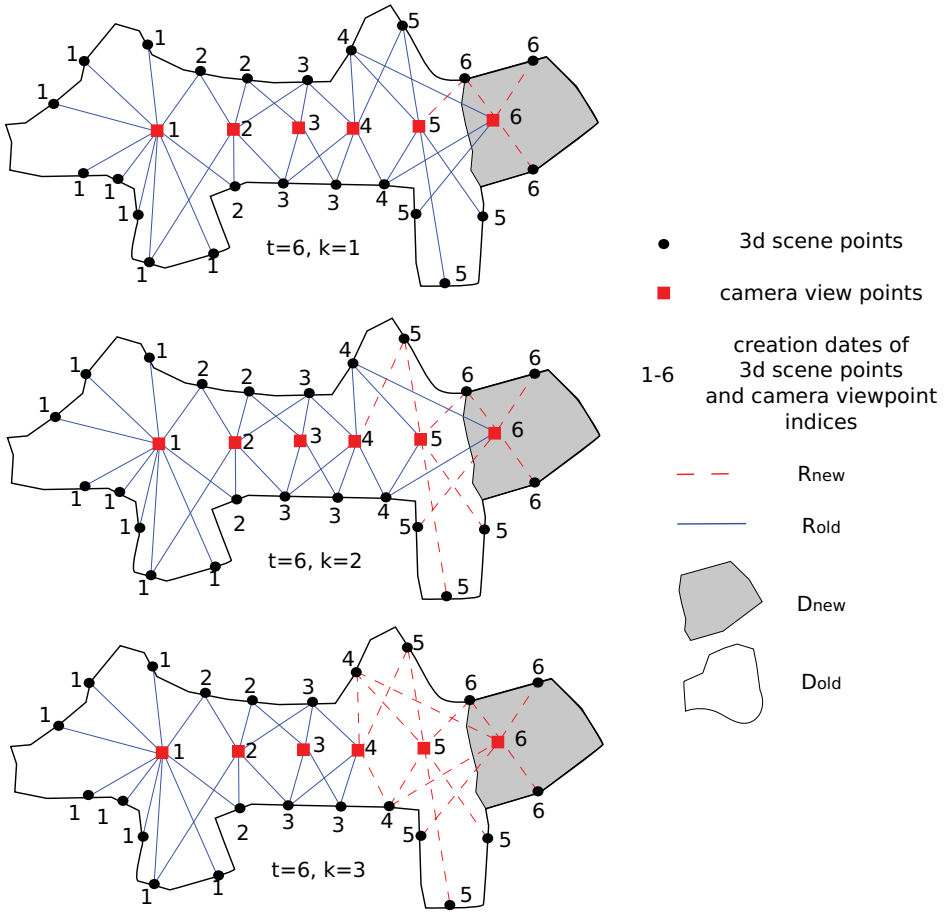


Figure 5.2: Examples of local ray-tracing at time $t = 6$ with different k. $k = 1$ (top), $k = 2$ (middle) and $k = 3$ (bottom)

In case of loops in the camera trajectory Assume that loops exist in the camera trajectory. In the area where a loop is closed, the tetrahedra D_{new} recently created are near to 3d points with old creation dates. As tetrahedra of D_{new} are not tested with R_{old}, there might exist un-carved tetrahedra. However, these un-carved tetrahedra are rare. In fact, as we use an omni-directional camera, new images acquired at the closed loops area are similar to the old images acquired at this area. Under some conditions (we talk about them in the next paragraph), almost the same 3d points are reconstructed by SfM. It implies that the space covered by R_{new} is almost the same to the space covered by R_{old} in the closed loop area. Consequently, we do not test D_{new} with R_{old}.

The proof above is valid under some conditions. The camera moving directions should be roughly parallel at the beginning and at the end of the loop in the camera trajectory. This is due to the point filter used for selecting points to add in the Delaunay triangulation D. Remember that for a pair of camera viewpoints $\{\mathbf{t}^1, \mathbf{t}^2\}$, a 3d point $\mathbf{q}$ can be selected in

D only if $\mathbf{q}$ is in a so-called non degenerate region such that $\varepsilon < \widehat{\mathbf{t}^1\mathbf{q}\mathbf{t}^2} < \pi - \varepsilon$ (Sec. 4.2). The proof in the paragraph above requires that almost the same 3d points are reconstructed at the beginning and the end of the loop. That means the non degenerate regions of camera viewpoint pairs at the beginning and the end of the loop should be almost the same (overlapping). Assume that $\{\mathbf{t}^a, \mathbf{t}^b\}$ and $\{\mathbf{t}^c, \mathbf{t}^d\}$ are respectively at the beginning and the end of a loop. As illustrated by Fig. 5.3, we can see that the non degenerate zones of $\{\mathbf{t}^a, \mathbf{t}^b\}$ and $\{\mathbf{t}^c, \mathbf{t}^d\}$ are almost overlapping only if $\mathbf{t}^a\mathbf{t}^b$ and $\mathbf{t}^c\mathbf{t}^d$ are roughly parallel and close.

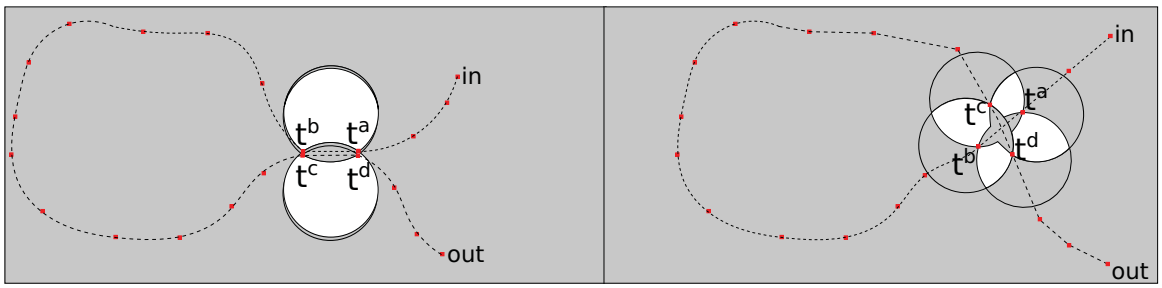


Figure 5.3: Overlapping of non degenerate regions for camera viewpoints at the beginning and the end of a loop in the camera trajectory. Camera viewpoints (red squared points) $\mathbf{t}^a, \mathbf{t}^b$ and $\mathbf{t}^c, \mathbf{t}^d$ are respectively at the beginning and the end of a loop in the camera trajectory (dashed line). The overlapping of non degenerate regions of camera pairs $\{\mathbf{t}^a, \mathbf{t}^b\}$ and $\{\mathbf{t}^c, \mathbf{t}^d\}$ is in white. Left: $\mathbf{t}^a\mathbf{t}^b$ and $\mathbf{t}^c\mathbf{t}^d$ are roughly parallel and close, thus their non degenerate regions are almost overlapped. Right: $\mathbf{t}^a\mathbf{t}^b$ and $\mathbf{t}^c\mathbf{t}^d$ are not parallel thus only part of their non degenerate regions are overlapped.

5.4 Incremental 2-manifold Generation

5.4.1 Problem

The third step of our incremental surface reconstruction is 2-manifold generation by applying an incremental region-growing process. It is an incremental version of the region-growing algorithm used in the batch version (Sec. 4.4). At each time t, it grows an *outside* region O_t in the *free-space* tetrahedra and the border of O_t is a 2-manifold S_t. It is obvious that growing O_t from $\emptyset$ like in the batch case cannot satisfy the incremental performance. As the region-growing is a progressive process, an intuitive idea of growing the current region based on previous regions comes naturally into mind: if the *outside* region O_{t-1} is already grown at $t-1$, one can grow from O_{t-1} with *free-space* tetrahedra obtained at t to obtain the *outside* region O_t at t.

However there exists a problem: O_{t-1} will probably be modified at t due to the add of new points to D_{t-1} and the surface S_{t-1}, which is the border of O_{t-1}, will be no longer 2-manifold. Consequently, the region-growing from O_{t-1} to O_t cannot apply. Fig. 5.4 illustrates an example of this problem. The add of new 3d points to D_{t-1} at t destroy some tetrahedra of O_{t-1}. The tetrahedra destruction is uncontrolled and S_{t-1} becomes probably non-manifold.

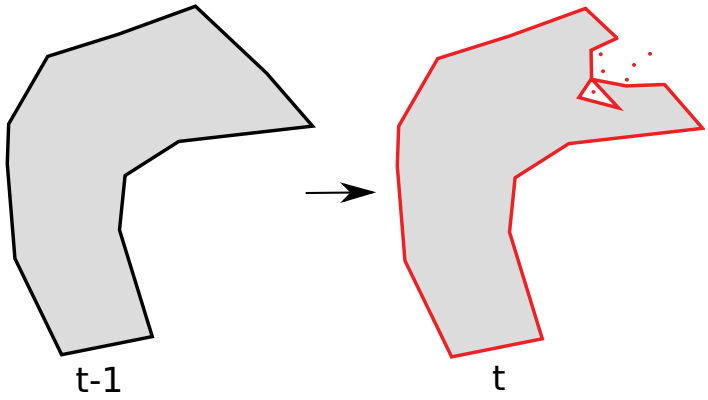


Figure 5.4: Problem of region-growing. Left: 2-manifold at $t-1$. Right: surface at t. In both figures, we see *outside* region (gray), *inside* region (white), S_{t-1} (black line), new 3d points added at t (red points), S_{t-1} after adding new 3d points at t (red lines).

5.4.2 Principle of our Solution

The proposed solution still takes the idea of growing *outside* region from previous *outside* regions and at the same time always maintains the 2-manifold property. This solution is based on temporal layers of tetrahedra of the Delaunay. The idea is that: *outside* regions created at all instances are stored. Then the *outside* region O_t at time t can be grown from a stored *outside* region $O_{i_0}(i_0 < t)$ such that the border of O_{i_0} is 2-manifold.

Now assume that the border of the *outside* region $O_{i_0}(i_0 < t)$ is 2-manifold and O_t is grown from O_{i_0}. The region-growing is done layer by layer. For each temporal layer i, a new *outside* region $O_i(i_0 \leq i \leq t)$ corresponding to this layer is generated and stored. Only *free-space* tetrahedra created before or at layer i can be added into O_i. As a result, for each temporal layer, a 2-manifold can be generated as the border of O_i and all tetrahedra in O_i have a creation date $\leq i$. The region-growing stops when $i = t$ and the border of the last O_i is then the target 2-manifold. This process of region-growing layer by layer is named the layer-wise region-growing.

How to choose the starting temporal layer i_0 is showed as follows. We recall that when new 3d points are added to the Delaunay D_{t-1} to create D_t, some tetrahedra are destroyed and some new tetrahedra are created. Let d_t $(d_t < t)$ be the smallest creation date of destroyed tetrahedra. We can see that the border of the *outside* region at the temporal layer $d_t - 1$ (and also other temporal layers before $d_t - 1$) is a valid 2-manifold. Thus $d_t - 1$ can be chosen as the beginning temporal layer i_0 of the incremental region-growing.

Unfortunately, as showed by the experiment in Sec. 7.3.4, the layer-wise region-growing is not efficient from a computational viewpoint. At time t, assume that D_t has two incident tetrahedra Δ_a, Δ_b which have respectively creation dates a, b such that $i_0 \leq a < b \leq t$. Assume also that Δ_b can be added in the outside region O_b by region-growing at layer b. From layer a, Δ_a is tried to be added in O_i $(i \in \{a, \cdots, b\})$. However depending on neighbor configuration of Δ_a, sometimes it cannot be added in O_i if Δ_b is not in O_i. Thus redundant tentatives of adding Δ_a in O_i are done, until Δ_b is added in O_i.

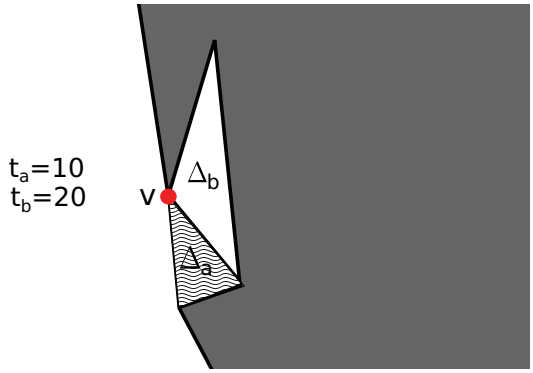


Figure 5.5: An example (in the 2d case) showing why the layer-wise region-growing is inefficient. At time $t = 20$, we see *outside* region (gray zone), *inside* region (white zone), a tetrahedron Δ_a (waved triangle) of creation date 10, a tetrahedron Δ_b (white triangle) of creation date 20 which is adjacent to Δ_a. The vertex $\mathbf{v}$ (red round point) will be singular if Δ_a is added in the *outside* region before Δ_b. Thus $b - a = 10$ redundant tentatives have been done for Δ_a before region-growing at layer 20 where Δ_b and Δ_a are successively added in the *outside* region.

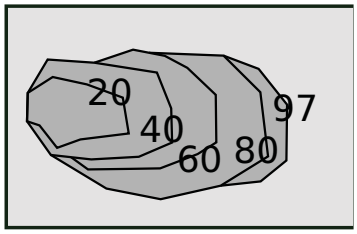
Let us see an example in Fig 5.5 for better understanding. At time $t = 20$, a tetrahedron Δ_a of creation date $a = 10$ and another tetrahedron Δ_b of creation date $b = 20$ are adjacent. Due to vertex $\mathbf{v}$, Δ_a cannot be added in O if Δ_b is *inside*. As a result, $b - a = 10$ redundant tests are done for Δ_a before adding it to O.

5.4.3 Region-Growing by Pack of Temporal Layers

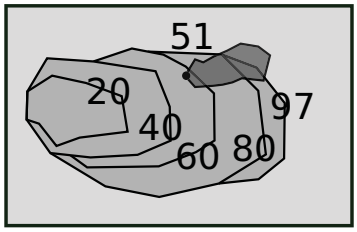
In our final incremental region-growing algorithm, a region-growing method by pack of temporal layers is used to generate incrementally the 2-manifold. Let $\mathcal{L}_i$ be the free-space tetrahedron list of each layer t_i. At each time t, several lists of *outside* tetrahedra are hold which correspond to particular creation dates (multiples of a parameter l): O_l, O_{2l}, ... $O_{i_t l}$ where l is a fixed integer and i_t is the largest integer such that $i_t l \leq t$. These lists and also O_t of *outside* tetrahedra meet

$$\begin{aligned} &O_l \subseteq O_{2l} \subseteq \cdots \subseteq O_{(i_t-1)l} \subseteq O_{i_t l} \subseteq O_t \\ &\forall t' \in \{l, 2l, \cdots, i_t l, t\}, O_{t'} \subseteq F_{t'} = \mathcal{L}_1 \cup \mathcal{L}_2 \cup \cdots \cup \mathcal{L}_{t'} \\ &\text{and the border of } O_{t'} \text{ is 2-manifold.} \end{aligned} \tag{5.1}$$

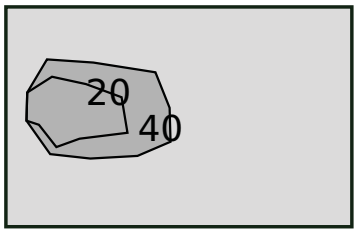
The border of O_t is the target 2-manifold S at time t. If $t \leq l$, we apply the batch region-growing in all *free-space* tetrahedra from O_0 to obtain O_t ($O_0 = \emptyset$). Now assume that $t > l$. The algorithm works from Eq. 5.1 at $t - 1$ to Eq. 5.1 at t. Let i_0 be the largest integer such that $i_0 l < d_t$. If $i \leq i_0$, O_{il} is unchanged and its border is still manifold. If $i_0 < i$, tetrahedra may be destroyed in O_{il}, its border may be non manifold and O_{il} should be recomputed. Time starts from 1, thus $1 \leq d_t$, $0 \leq i_0$ and $O_{i_0 l}$ exists. Then our method grows from $O_{i_0 l}$ to obtain $O_{(i_0+1)l}$, then from $O_{(i_0+1)l}$ to obtain $O_{(i_0+2)l}$ and so on, until $O_{i_t l}$ is obtained. At last, our method grows from $O_{i_t l}$ to obtain O_t if $i_t l < t$. An example is shown in Fig. 5.6. In this example, $t = 98, l = 20, d_t = 51$.



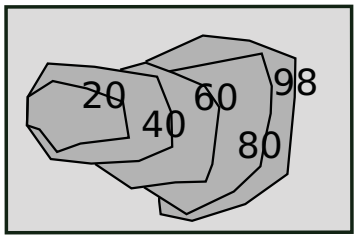
(a) Before region-growing for image 98, the 2-manifolds of layers 20, 40, 60, 80 and 97 are already computed.



(b) The points additions destroy tetrahedra. The earliest creation date of tetrahedra which are destroyed due to point additions to the Delaunay is $d_t = 51$.



(c) The 2-manifolds of layers 60, 80, and 97 become invalid so they are destroyed.



(d) Region-growing from layer $i_0 l = 40$ to layer 98 by pack of $l = 20$ layers.

Figure 5.6: An example of incremental region-growing. The current time t is 98, the number of temporal layers contained in a pack is $l = 20$, and the smallest creation date of destroyed tetrahedra d_t is 51.

In summary, our incremental region-growing method for time t, can be considered as a series of successive region-growings for fixed special times $\{i_1 l, \cdots, i_t l, t\}$. Each successive region-growing is based on results of the previous region-growing (except for the first one which performs based on the *outside* region of time $i_0 l$).

Algorithm of Successive Region-growing Let t be the current time and let us consider the successive region-growing from layer il to layer $(i+1)l \leq t$. Algorithm. 4, which is a modification of Algorithm. 1, shows the detailed algorithm of the successive region-growing.

Here, the inputs are O_{il} and $F_{(i+1)l}$, where O_{il} is the *outside* region at il. $F_{(i+1)l}$ is the set of *free-space* tetrahedra created before or at $(i+1)l$. For reasons of theoretical complexity, we use $F_{(i+1)l} = \mathcal{L}_{(i_0-1)l} \cup \cdots \cup \mathcal{L}_{(i+1)l}$ (not those of $\mathcal{L}_1 \cup \cdots \cup \mathcal{L}_{(i+1)l}$). In addition, the list Q_0 of tetrahedra used to initialize the priority queue Q is $\mathcal{L}_{il-b_0} \cup \mathcal{L}_{il-b_0+1} \cup \cdots \cup \mathcal{L}_{(i+1)l}$. Here, b_0 is a threshold and $b_0 \in \mathbb{N}$.

Algorithm 4: A Successive Region-growing from O_{il} to $O_{(i+1)l}$

Inputs : O_{il}, Q_0, $F_{(i+1)l}$
Outputs: $O_{(i+1)l}$
Step1: Initialization
Create a priority queue Q;
if $O_{il} == \emptyset$ **then**
//the case $d_t < l$
Let $\Delta \in F_{(i+1)l}$ be such that $r(\Delta)$ is maximum;
$Q \leftarrow Q \cup \{\Delta\}$;
else
For each tetrahedron $\Delta \in Q_0 \cap F_{(i+1)l}$ **:**
if Δ has at least one adjacent tetrahedron in O_{il} && $\Delta \notin O_{il}$ **then**
$Q \leftarrow Q \cup \{\Delta\}$;

Step2: Region-growing (same as Step 2 of Algorithm. 1, rewritten here for clarity)
$O_{(i+1)l} = O_{il}$;
while $Q \neq \emptyset$ **do**
Pick from Q the Δ which has the largest $r(\Delta)$;
if $\Delta \in O_{(i+1)l}$ **then**
continue;
$O_{(i+1)l} \leftarrow O_{(i+1)l} \cup \{\Delta\}$;
if all vertices of Δ are regular **then**
For each adjacent tetrahedron Δ' of Δ **:**
if $\Delta' \in F_{(i+1)l}$ *and* $\Delta' \notin O_{(i+1)l}$ **then**
$Q \leftarrow Q \cup \{\Delta'\}$;
else
$O_{(i+1)l} \leftarrow O_{(i+1)l} \setminus \{\Delta\}$;

5.5 Incremental Topology Extension

Each time after a region-growing from $O_{(i-1)l}$ to O_{il} (including O_t), the *outside* region O_{il} obtained at the previous step is improved by an incremental version of "Topology Extension" (Sec. 4.5) . The improved O_{il} still meets Eq. 5.1. Here, the *free-space* tetrahedron list F_{il} used for region-growing from $O_{(i-1)l}$ to O_{il} is also used in the topology extension.

For a vertex $\mathbf{v}$ in the current 2-manifold δO_{il}, the topology extension of $\mathbf{v}$ in the incremental case applies two steps as follows:

1. it tries to switch the set A of *inside* tetrahedra incident to $\mathbf{v}$ to *outside*, i.e. O_{il} is updated. Here, all tetrahedra in A should be in F_{il}.

2. if the Step 1 succeeds, the successive region-growing algorithm (Algorithm. 4) is ap-

plied. The inputs O_{il}, F_{il} are the same as in the incremental manifold generation (Sec. 5.4.3) and only Q_{il} is different. In this case, Q_{il} is the list of tetrahedra adjacent to A.

For reasons of computational costs, not every vertex in δO_{il} is tried but only vertices with the most recent creation dates to layer il. More precisely, both steps above are only applied to the vertices of δO_{il} which have creation dates in $\{il - b_1, \cdots, il - 1, il\}$ where $b_1 \in \mathbb{N}$ is a threshold such that $b_1 < l$. In addition, these vertices are tried only once.

5.6 Incremental Post-processing

Incremental Surface Denoising An incremental version of the surface denoising process in Sec. 4.7.2 is applied to reduce the noise of the incremental 2-manifolds obtained.

Let S_{t-1} and S_t be the 2-manifolds obtained at time $t-1$ and t respectively. Let S'_{t-1} be the denoised 2-manifold obtained at time $t-1$. Now the objective is to calculate the denoised 2-manifold S'_t at time t. Assume that both S_{t-1} and S_t have a vertex $\mathbf{v}$,

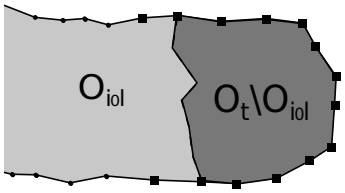


Figure 5.7: Vertices which should be denoised in incremental surface denoising for time t. *Outside* region O_t (light and dark gray) obtained at t is grown from O_{i_0l} (light gray) and the grown volume is $O_t \setminus O_{i_0l}$ (dark gray). The border of O_t is the 2-manifold S_t for time t. In vertices (round and squared points) of S_t, those (squared points) which should be denoised are on border or one-ring neighbors to the border of the grown volume $O_t \setminus O_{i_0l}$.

and the $\mathbf{v}$ neighborhood (one ring neighbors) in S_{t-1} is the same as the one in S_t, i.e. $\mathcal{N}(\mathbf{v}, S_{t-1}) = \mathcal{N}(\mathbf{v}, S_t)$. Then the denoised vertex $\mathbf{v}'$ in S'_t is the same to the one in S'_{t-1}, thus it is useless to recalculate it. More precisely, remember that in the batch case, we have $\mathbf{v}' = \mathbf{v} + \lambda\Delta\mathbf{v}$ (Eq.4.4). The denoised vertex $\mathbf{v}'$ depends on $\mathbf{v}$ and $\mathcal{N}(\mathbf{v})$ which is the list of one-ring neighbor vertices of $\mathbf{v}$ on the 2-manifold. In the incremental case, $\mathbf{v}'$ should be (re)-calculated only if $\mathbf{v}$ is a new vertex in S_t or if $\mathcal{N}(\mathbf{v}, S_{t-1}) \neq \mathcal{N}(\mathbf{v}, S_t)$.

The incremental region-growing step (Sec. 5.4.3) and the incremental topology extension step (Sec. 5.5) grow from an *outside* region O_{i_0l} to obtain O_t. $O_t \setminus O_{i_0l}$ is the *outside* volume newly grown. So the vertices in S_t which should be denoised are those on the border or one-ring neighbors to the border of $O_t \setminus O_{i_0l}$. In other words, $\mathbf{v}'$ will be (re)-calculated if $\mathbf{v}$ is in S_t and $\mathcal{N}(\mathbf{v}, S_t) \cup \{\mathbf{v}\}$ contains at least one vertex of the border of $O_t \setminus O_{i_0l}$. See Fig. 5.7 for an example.

5.7 Conclusion

In this chapter, we have presented an incremental surface reconstruction method using results of an incremental SfM algorithm. Sparse SfM points are progressively provided and a 2-manifold surface modeling the environment is incrementally updated, i.e. each time new SfM points are available, a new 2-manifold surface is reconstructed by updating the surface previously reconstructed with these new SfM points. Each surface update firstly incrementally updates a 3d Delaunay triangulation and labels *free-space* tetrahedra with new SfM points and point visibilities. Then it grows a new *outside* region from a previous *outside* region (whose border is 2-manifold) in the Delaunay triangulation. Finally, the border of the new *outside* region is the updated 2-manifold.

Up to our knowledge, this surface reconstruction method [4, 6] which incrementally estimates a 2-manifold using sparse (SfM) points is novel in the Computer Vision field. Note that it can be seen as the incremental version of our batch surface reconstruction method presented in Chap. 4, although the incremental one is not 100% complete compared to the batch version. In the incremental version, the second method of spurious handle removal and several post-processings (peak removal, sky removal and texturing) are lacking.

Chapter 6

Time Complexity Analyses

In this chapter, the time complexities in the worst case of all steps of our batch and incremental surface reconstruction methods are analyzed. The standard "big o" notation $\mathcal{O}(x)$ is used and $\mathcal{O}(1)$ means "bounded". Note that the font $\mathcal{O}$ used for the "big o" notation is different from the font O, which is used to represent the list of *outside* tetrahedra.

In Sec. 6.1, some notations and data structures used for complexity analyses are presented. In Sec. 6.2, loose analyses about the worst case time complexity of our surface reconstruction methods are shown. These analyses are done with very few lemmas deduced from our SfM methods. In Sec. 6.3, a Steiner grid bounding the input point cloud is added to the Delaunay triangulation for complexity analysis. Compared to Sec. 6.2, more assumptions can be made and tight complexities, which are more consistent to the experimental complexities, can be obtained. At the same time, the Steiner grid has a very low density so that the reconstructed surface quality and the process time are almost unchanged. At last, Sec. 6.4 summarizes results of the time complexity analyses.

6.1 Notations and Data Structures

Here we introduce some notations and data structures used for the complexity analyses.

The 3d Delaunay triangulation is D. $|L|$ is the number of elements of a list L; $|D|$ is the number of tetrahedra of D. n is the number of SfM points and m is the number of camera viewpoints. d is the maximum vertex degree of D, i.e. the maximum number of tetrahedra incident to a vertex of D except the infinite vertex $\mathbf{v}_\infty$.

A list of informations frequently used are attached to each tetrahedron and each vertex of D:

1. For every tetrahedron Δ, we store: the index list of 4 Δ vertices, the index list of 4 neighbor tetrahedra of Δ, the number $r(\Delta)$ of intersected rays, a label *outside* or *inside*. Here, the label *free-space* or *matter* is not stored because $r(\Delta) > 0$ means *free-space* and $r(\Delta) = 0$ means *matter*. In the incremental case, informations above are updated each time when necessary. Besides, we store also other informations like the creation date $\mathrm{dc}(\Delta)$ and the grown date $\mathrm{dg}(\Delta)$ of Δ. Here the grown date of Δ is the index of

the layer from which Δ is grown, i.e. it is il if Δ becomes *outside* in the region-growing from layer $(i-1)l$ to il.

2. For every vertex $\mathbf{v}$, we store: the list of indices of its incident tetrahedra, the list of its visibility rays. Besides, in the incremental case, informations above are updated each time when necessary and the creation date of $\mathbf{v}$ is also stored.

Note that in the incremental case, we use the creation date $\mathrm{dc}(\Delta)$ to indicate if Δ is in F_t or F_{il}. And the grown date $\mathrm{dg}(\Delta)$ can be used to indicate if Δ is in O_t or O_{il}, e.g. $\mathrm{dg}(\Delta) = il$ implies that $\Delta \in O_{il}$ and $\Delta \notin O_{(i-1)l}$ (assuming that $(i-1)l \geq 0$).

6.2 Loose Complexity Analysis

6.2.1 Assumptions and Lemmas

Here are some assumptions and lemmas which are obtained thanks to properties of our SfM methods. They are used for the loose complexity analyses.

- H1: the size of the ray list V of a 3d point $\mathbf{q}$ is $\mathcal{O}(1)$.
- H2: in the incremental case, $\mathcal{O}(1)$ reconstructed points are added to the 3d Delaunay D_t at time t.

We suppose that a detected Harris point is tracked in a limited number of consecutive images. It implies that the size of V is bounded (H1).

In the pre-processing step, for each image, the incremental SfM detects and matches a limited number of Harris points (Sec. 3.1.2). Besides, a detected Harris point is tracked in a limited number of consecutive images(H1). Thus given an image t, the number of Harris points which stop to be tracked at $t-2$, i.e. the number of 3d points which become stable at t, is also bounded. Remember that a 3d point $\mathbf{q}$ is added to the Delaunay triangulation D_t at t only when $\mathbf{q}$ becomes stable at t (Sec. 4.2). As a result, we can conclude that : the number of 3d points added to the Delaunay D_t at t is always $\mathcal{O}(1)$ (H2).

6.2.2 Batch Surface Reconstruction

Loose complexities of the batch surface reconstruction are analyzed in this section. They are expressed by using the number of camera viewpoints locations m and the number of 3d points n. In practice, we have $m < n$.

6.2.2.1 3d Delaunay triangulation

The first step is the point filtering. For each SfM point $\mathbf{q}_j$, checking that $\mathbf{q}_j$ can be added in the Delaunay triangulation D consists in calculating aperture angles using all pairs of rays in ray list V_j of $\mathbf{q}_j$. The size of V_j is $\mathcal{O}(1)$ (H1), thus the check for one SfM point has a complexity of $\mathcal{O}(1)$. There are n SfM points, so the complexity of the point filtering is $\mathcal{O}(n)$.

Now given $\mathcal{O}(n)$ points, according to Sec. 1.5.3, the Delaunay triangulation construction of D using these points has a time complexity of $\mathcal{O}(n^2)$ in the worst case, and D has $\mathcal{O}(n^2)$ tetrahedra (Sec. 1.5.3).

6.2.2.2 *Free-space/matter* labeling

The ray-tracing algorithm is applied to every ray to label *free-space/matter* tetrahedra. The worst case of ray-tracing for a ray is that this ray goes through all tetrahedra of D [127]. Note that D contains $\mathcal{O}(n^2)$ tetrahedra and there are $\mathcal{O}(n)$ 3d points. Besides, the number of rays of a point is bounded (H1). As a result the worst time complexity of this step is $\mathcal{O}(n^3)$.

6.2.2.3 2-manifold tests

Here we analyze complexities of different 2-manifold tests presented in Sec. 4.4.3.1. These tests are used in steps of 2-manifold generation (Sec. 4.4), topology extension (Sec. 4.5) and the second method of spurious handle removal (Sec. 4.6.2). They check that the surface δO is still 2-manifold after adding the tetrahedron list A in O, i.e. all vertices of tetrahedra of A are regular. Here, A contains a single finite tetrahedron or several finite tetrahedra sharing a same surface vertex. In the second case, the shared vertex cannot be the infinite vertex $\mathbf{v}_\infty$ since all $\mathbf{v}_\infty$-incident tetrahedra are *matter* and every surface vertex is incident to a *free-space* tetrahedron. Thus, $|A| \leq d$, i.e. $|A| = \mathcal{O}(d)$, and the number of vertices of the tetrahedra in A is $\mathcal{O}(d)$.

In case of edge-based or tetrahedron-based vertex tests, A is firstly added in O and we check that every vertex of tetrahedra in A is regular. For the edge-based vertex test, checking that a vertex $\mathbf{v}$ is regular consists in calculating the link of $\mathbf{v}$ and checking that the link is a simple polygon. The complexity is $\mathcal{O}(d^2)$. For the tetrahedron-based vertex test, it consists in a simple travel in the graph of tetrahedra incident to $\mathbf{v}$. The complexity is $\mathcal{O}(d)$. The latter has a better complexity than the former, thus we prefer to use the latter to check that a vertex is regular in practice. If A contains a single tetrahedron, the complexity of checking all vertices of A using the tetrahedron-based vertex test is $\mathcal{O}(d)$. If A contains more than one tetrahedra, the complexity is $\mathcal{O}(d^2)$.

In case of single tetrahedron test, the test checks the incident tetrahedra configuration of each vertex of a tetrahedron Δ to decide if Δ can be added in O. Each vertex has $\mathcal{O}(d)$ incident tetrahedra. According to the proof of the single tetrahedron test in Appendix C, one possible configuration to add Δ in O involves incident tetrahedra lists of two vertices. In this case, the complexity is $\mathcal{O}(d)$ if the lists are ordered, or $\mathcal{O}(d^2)$ if the list are not ordered. In our implementation, the incident tetrahedron lists are ordered to reduce the complexity. Thus finally, the complexity of our single tetrahedron test is $\mathcal{O}(d)$.

6.2.2.4 *2-manifold generation*

Now the time complexity of the greedy region-growing algorithm introduced in Sec. 4.4.3.2 is analyzed. One might refer to Algorithm. 1 for better reading. Let q_0 be the number of

tetrahedra in list Q_0. Let g be the number of grown tetrahedra, i.e. the difference between the number of tetrahedra in the output O and in the input O of this algorithm. A tetrahedron Δ is definitely added at most once into O. In this case, at most four tetrahedra are added to Q. There is no other addition to Q, except at the initialization of Q where there are at most $\mathcal{O}(q_0)$ additions to Q. Thus the number of "while" iterations in Algorithm. 1 is $\mathcal{O}(g + q_0)$.

In one "while" iteration, picking the best tetrahedron in heap Q is $\mathcal{O}(\log(g + q_0))$. Then the single tetrahedron 2-manifold test is used to check that if a finite tetrahedron Δ can be added in O. According to Sec. 6.2.2.3, it has a complexity of $\mathcal{O}(d)$. As a result, the complexity of one "while" iteration is $\mathcal{O}(\log(g + q_0) + d)$.

The complexity of the greedy region-growing algorithm is then $\mathcal{O}((g+q_0)(\log(g+q_0)+d))$. The 2-manifold generation process uses the greedy region-growing process by starting O from $\emptyset$. Thus, $g = \mathcal{O}(n^2)$ and $q_0 = 1$. Besides, Q should be initialized by choosing the tetrahedron with the largest r, which has a complexity of $\mathcal{O}(|D|)$. Finally, the 2-manifold generation process has a complexity of $\mathcal{O}(|D| + (g + q_0)(\log(g + q_0) + d))$. Then it is $\mathcal{O}(n^3)$ since $|D| = \mathcal{O}(n^2)$, $g + q_0 = \mathcal{O}(n^2)$ and $d = \mathcal{O}(n)$.

6.2.2.5 *Topology extension*

Remember that "Topology Extension" alternates two steps:

1. Switching from *inside* to *outside* tetrahedra incident to a surface vertex $\mathbf{v}$, and check that the resulting surface is still 2-manifold.

2. Region growing in F starting from the neighborhood of reversed tetrahedra of step 1.

If Step 1 is successful, Step 2 occurs. Otherwise, the switching of Step 1 is reversed (from *outside* to *inside*). This process is tried a finite number of times for every surface vertex.

In Step 1, we consider firstly one vertex $\mathbf{v}$ in the surface. Let $L'(\mathbf{v})$ be the list of *inside* tetrahedra of $\mathbf{v}$. The goal is to check that all vertices of tetrahedra in $L'(\mathbf{v})$ are regular if $L'(\mathbf{v})$ is added in O. The tetrahedron-based test is used here. Here, we see that the size of $L'(\mathbf{v})$ can be larger than one. So according to Sec. 6.2.2.3, the tetrahedron-based test in this case has a complexity of $\mathcal{O}(d^2)$. Now let s be the number of vertices in the reconstructed 2-manifold S, thus doing Step 1 for all vertices on S is $\mathcal{O}(d^2 s)$. As $s = \mathcal{O}(n)$ and $d = \mathcal{O}(n)$, finally the complexity of all Steps 1 is $\mathcal{O}(n^3)$.

The i-th growing from the i-th vertex (step 2) has complexity $\mathcal{O}((g_i+q_i)(d+\log(g_i+q_i)))$ (refer to the complexity analysis of the greedy region-growing algorithm in Sec. 6.2.2.4). g_i is the number of grown tetrahedra and q_i the number of tetrahedra in Q_0. We have $\sum_i g_i = \mathcal{O}(n^2)$ since the total number of tetrahedra is $\mathcal{O}(n^2)$. As $q_i = \mathcal{O}(d)$, we have $\sum_i q_i = \sum_i \mathcal{O}(d) = \mathcal{O}(nd)$. Then $\sum_i q_i = \mathcal{O}(n^2)$. The complexity of all Steps 2 is,

$$\begin{aligned} \sum_{i\in I}(g_i + q_i)(d + \log(g_i + q_i)) &\leq (\sum_{i\in I}(g_i + q_i))(d + \log(\sum_{i'\in I} g_{i'} + \sum_{i'\in I} q_{i'})) \\ &= \mathcal{O}(n^2(d + \log(n^2)) \\ &= \mathcal{O}(n^3). \end{aligned} \tag{6.1}$$

In summary, the topology extension has a complexity of $\mathcal{O}(n^3)$.

6.2.2.6 *Spurious handles removal*

The first handle removal method (Sec. 4.6.1) adds to D a number of Steiner vertices which is linear to m. The number of Steiner vertices per camera location is $\mathcal{O}(1)$ and they act as reconstructed vertices with empty ray lists. Since $m < n$, the complexities of all steps of the surface reconstruction do not change.

The second handle removal method (Sec. 4.6.2) has steps "Detect", "Force" and "Repair". D has n vertices, and each vertex has maximum $\mathcal{O}(n)$ edges, thus D has $\mathcal{O}(n^2)$ edges. The complexity of "Detect" for every edge is $\mathcal{O}(m)$. So the complexity of "Detect" is $\mathcal{O}(mn^2)$.

Now we analyze the complexity of "Force" and "Repair". A Steiner vertex is added in each critical edge, thus there are three vertices in each critical edge. Let L be the list of tetrahedra incident to one vertex. The size of L is $\mathcal{O}(d)$. Now two rounds of "Force" and "Repair" are applied for each vertex. Both rounds force a list A of of tetrahedra to be *outside*, and apply a greedy region-growing algorithm to increase O using a heap Q (as the Algorithm. 1 in Sec. 4.4). As already shown in Sec. 6.2.2.4, the number of "while" iterations is $\mathcal{O}(g + q_0)$, where g is the number of grown tetrahedra and q_0 the number of tetrahedra in the initialization of heap Q. Furthermore, the complexity of one "while" iteration is $\mathcal{O}(d + \log(g + q_0))$.

In the first round, the list A contains all *inside* and *free-space* tetrahedra of L. Thus its size is $\mathcal{O}(d)$. The initialized heap Q contains all *inside* and *free-space* tetrahedra which are adjacent to A. Thus $q_0 = \mathcal{O}(d)$. Besides, $g \leq g_{\max}$ where $g_{\max}$ is the constant to limit the computation time for one "Repair". So the complexity of the first round is $\mathcal{O}(g_{\max} + q_0)(d + \log(g_{\max} + q_0))$, which is $\mathcal{O}(d^2)$.

In the second round, the list A contains respectively each *inside* and *free-space* tetrahedron of L. The size of A is 1 and the second round is applied $\mathcal{O}(d)$ times. For each time, $q_0 = \mathcal{O}(1)$ since the number of *inside* and *free-space* tetrahedra adjacent to A is less or equal to 4. Besides, $g \leq g_{\max}$. Thus the complexity of the second round is $\mathcal{O}(d(g_{\max} + q_0)(d + \log(g_{\max} + q_0))$, which is $\mathcal{O}(d^2)$.

In summary, the "Force" and "Repair" for one vertex in a critical edge is $\mathcal{O}(d^2)$. Since there are w critical edges and each critical edge has three vertices, the complexity of all "Force" and "Repair" steps is $\mathcal{O}(wd^2)$, which is $\mathcal{O}(n^2d^2)$. We conclude that the complexity of the second method of spurious handle removal is $\mathcal{O}(mn^2 + n^4)$. As $m < n$, so the final complexity is $\mathcal{O}(n^4)$.

6.2.2.7 *Post-processings*

Complexities of the post-processings are shown in the following paragraphs except the texturing which is not our contribution.

Peak Removal We consider at first one vertex $\mathbf{v}$ in the 2-manifold S. Let $L(\mathbf{v})$ be the list of tetrahedra incident to $\mathbf{v}$. $|L(\mathbf{v})|$ is $\mathcal{O}(d)$, thus calculating the solid angle for $\mathbf{v}$ has a complexity of $\mathcal{O}(d)$. Now if $\mathbf{v}$ is a peak, then the peak removal of $\mathbf{v}$ consists in inversing labels of $L(\mathbf{v})$ and checking that vertices of all tetrahedra in $L(\mathbf{v})$ are regular using tetrahedron-based test in Sec. 4.4.3.1. The former has a complexity of $\mathcal{O}(d)$. The complexity of the latter

is $\mathcal{O}(d^2)$ since the number of vertices of all tetrahedra in $L(\mathbf{v})$ is $\mathcal{O}(d)$ and the tetrahedron-based test has a complexity of $\mathcal{O}(d)$ (Sec. 6.2.2.3). As a result, the complexity of peak removal for one vertex is $\mathcal{O}(d^2)$, which is $O(n^2)$. The 2-manifold S has $\mathcal{O}(n)$ vertices thus finally, the complexity of the peak removal is $\mathcal{O}(n^3)$.

Surface denoising The 2-manifold S has $\mathcal{O}(n)$ vertices and for each vertex $\mathbf{v}$, the Laplacian operator is applied to $\mathbf{v}$ and its one-ring neighbor vertices $\mathcal{N}(\mathbf{v})$. $|\mathcal{N}(\mathbf{v})|$ is $\mathcal{O}(d)$. Thus the complexity of surface denoising is $\mathcal{O}(nd)$, thus $\mathcal{O}(n^2)$.

Sky removal The sky direction is calculated using all z-axis of catadioptric camera, thus it has a complexity of $\mathcal{O}(m)$. The number of triangles of S is less than $4 \times |D|$. Thus it is $\mathcal{O}(n^2)$. So the rectangle-triangle intersections are $\mathcal{O}(n^2 m)$ since S has $\mathcal{O}(n^2)$ triangles and there are m open rectangles. The border growing is bounded by the number of S triangles ($\mathcal{O}(n^2)$). We conclude that the complexity of Sky Removal is $\mathcal{O}(n^2 m)$.

In summary, the complexity of the post-processings is $\mathcal{O}(n^3 + n^2 + n^2 m)$. As $m < n$, thus the complexity is $\mathcal{O}(n^3)$.

6.2.3 Incremental Surface Reconstruction

Let t be the current time and let n_t be the current number of points. Let d_t be the smallest creation date of the tetrahedra which are destroyed due to the addition of 3d points at t. D_t is the 3d Delaunay triangulation at time t. H2 implies that at each time, a bounded number of points are added to the Delaunay, thus we have $n_t = \mathcal{O}(t)$. So at time t, the current Delaunay triangulation has $\mathcal{O}(t)$ vertices and $\mathcal{O}(t^2)$ tetrahedra. Besides, it is obvious that the number of camera viewpoint locations $m = t$. Thus we use t and d_t instead of n and m to parameterize complexities of our incremental surface reconstruction method.

6.2.3.1 Incremental 3d Delaunay triangulation

First, the complexity of checking if a SfM point can be added in the Delaunay triangulation is $\mathcal{O}(1)$ thanks to H1. Second, D_{t-1} has $\mathcal{O}(t)$ vertices. According to Sec. 1.5.3, adding a 3d point in a Delaunay triangulation of $\mathcal{O}(t)$ vertices has a complexity of $\mathcal{O}(t^2)$. At last, dating new tetrahedra after adding a 3d point has also a complexity of $\mathcal{O}(t^2)$ in the worst case where all tetrahedra of D_{t-1} are destroyed and rebuilt. As a result, the incremental 3d Delaunay triangulation for one 3d SfM point has a complexity of $\mathcal{O}(t^2)$. Since the number of new SfM points available at t is $\mathcal{O}(1)$ (H2), the complexity of the incremental 3d Delaunay triangulation is $\mathcal{O}(t^2)$.

6.2.3.2 Local ray-tracing

In the local ray-tracing step, only rays of 3d points whose creation dates are in the k most recent dates are used $(t-k+1, t-k+2, ..., t)$. Besides, H1 and H2 imply that the number of rays associated to each date is bounded. Thus, the number of rays used in this step is

$\mathcal{O}(1)$. Furthermore, as already shown in the batch case (Sec. 6.2.2.2), ray-tracing for one ray has $\mathcal{O}(t^2)$ complexity. As a result, the complexity of local ray-tracing is $\mathcal{O}(t^2)$.

6.2.3.3 Incremental 2-manifold generation

The incremental 2-manifold generation is composed by several successive region-growing in

$$F_t = \mathcal{L}_{(i_0-1)l} \cup \cdots \cup \mathcal{L}_t : \tag{6.2}$$

from $O_{i_0 l}$ to $O_{(i_0+1)l}$, from $O_{(i_0+1)l}$ to $O_{(i_0+2)l}$, ... and at last from $O_{i_t l}$ to O_t. i_0 is the largest integer such that $i_0 l < d_t$ and i_t is the largest integer such that $i_t l < t$. Let $I = \{i_0, i_0+1, .., i_t\}$, we have $|I| = \mathcal{O}(t - d_t)$.

According to results of the previous complexity analysis in Sec. 6.2.2.4, we know that i-th region-growing has a complexity of $\mathcal{O}((g_i + q_i)(\log(g_i + q_i) + d))$ where $i \in I$. g_i is the number of tetrahedra grown by this region-growing iteration, q_i is the number of tetrahedra in the initialized heap and d is the maximum vertex degree.

Note that $\sum_{i\in I} g_i$ are tetrahedra grown from layer $i_0 l$ to t and $\sum_{i\in I} g_i = \mathcal{O}(t^2)$. Indeed, one tetrahedron can be added in the *outside* region only one time in an incremental surface reconstruction. The heap is initialized by using $b_0 + l + 1$ layers of tetrahedra, and each layer contains $\mathcal{O}(t^2)$ tetrahedra, thus $q_i = \mathcal{O}(t^2)$. So $\sum_{i\in I} q_i = \mathcal{O}((t - d_t)t^2)$.

Now the complexity of the incremental manifold generation is then,

$$\begin{aligned}
\sum_{i\in I}(g_i + q_i)(\log(g_i + q_i) + d) &\leq (\sum_{i\in I} g_i + \sum_{i\in I} q_i)\log(\sum_{i'\in I} g_{i'} + \sum_{i'\in I} q_{i'}) \\
&\quad + d(\sum_{i\in I} g_i + \sum_{i\in I} q_i) \\
&= \mathcal{O}\{(t^2 + (t - d_t)t^2)(\log(t^2 + (t - d_t)t^2) + t)\} \\
&= \mathcal{O}(t^3(t - d_t))
\end{aligned} \tag{6.3}$$

6.2.3.4 Incremental topology extension

The topology extension is applied each time after the successive region-growing (Algorithm. 4) presented in Sec. 5.4.3.

Consider that one of a successive region-growing is already done which grows from $O_{(i-1)l}$ to O_{il} ($0 \leq (i-1)l$ and $il \leq t$). In order to extend the topology of O_{il}, two steps are alternated for each $\mathbf{v}$ of δO_{il} such that the creation date of $\mathbf{v}$ should be in $\{il - b_1, \cdots, l - 1, il\}$ where $b_1 \in \mathbb{N}$ and $b_1 < l$. These two steps are:

1. switch the set A of tetrahedra incident to $\mathbf{v}$ from *inside* to *outside*. Check that if the resulting 2-manifold is still valid using the tetrahedron-based test and reverse the switching if it is not the case. Here, all tetrahedra in A should be in F_{il}.

2. region-growing in F_{il} starting from a neighborhood of $\mathbf{v}$ (if Step 1 is successful).

In Step 1, switching tetrahedra A from *inside* to *outside* for one vertex $\mathbf{v}$ has the complexity of $\mathcal{O}(d)$. As A has generally more than one tetrahedra, checking that all vertices of tetrahedra in A are regular has a complexity of $\mathcal{O}(d^2)$ (Sec. 6.2.2.3). For the i-th incremental topology extension, the switching is applied to vertices of creation dates between $il - b_1$ and il. Thanks to H2, the number of vertices with the same creation date is bounded. Thus there are $\mathcal{O}(1)$ switching vertices for the i-th topology extension. As there are $|I|$ incremental topology extensions and $|I| = \mathcal{O}(t - d_t)$ (Sec. 6.2.3.3), the total number of switching vertices, which is also the number of Steps 1, is $\mathcal{O}(t - d_t)$. As a result, the complexity of all Steps 1 is $\mathcal{O}(d^2(t - d_t))$, so $\mathcal{O}(t^2(t - d_t))$.

The complexity analysis for Steps 2 is similar to the one of incremental 2-manifold generation in Sec. 6.2.3.3. The differences are the list I and the heap initialization for each Step 2.

Each time a vertex is successfully switched in Step 1, the Step 2 will be applied. So the list I contains indices of all these vertices. In the worst case (all Steps 1 succeed), $|I|$ is equal to the number of Steps 1 which is $\mathcal{O}(t - d_t)$. Besides, for each vertex $\mathbf{v}$, the heap in Step 2 is initialized by using the tetrahedra which are adjacent to A. Thus for each $i \in I$, $q_i = \mathcal{O}(d)$ and $\sum_{i \in I} q_i = \mathcal{O}((t - d_t)d)$, which is $\mathcal{O}(t(t - d_t))$.

g_i is the number of tetrahedra grown by each Step 2. One tetrahedron can be added in the *outside* region only one time, so $\sum_{i \in I} g_i = \mathcal{O}(t^2)$. Thus $\sum_{i \in I} g_i + \sum_{i \in I} q_i = \mathcal{O}(t^2)$.

Finally the complexity of all Steps 2 is,

$$\begin{aligned}\sum_{i \in I}(g_i + q_i)(\log(g_i + q_i) + d) &\leq (\sum_{i \in I} g_i + \sum_{i \in I} q_i)\log(\sum_{i' \in I} g_{i'} + \sum_{i' \in I} q_{i'}) \\ &\quad + d(\sum_{i \in I} g_i + \sum_{i \in I} q_i) \\ &= \mathcal{O}(t^2 \log t) + \mathcal{O}(t^3) \\ &= \mathcal{O}(t^3)\end{aligned} \tag{6.4}$$

In summary, the incremental topology extension has a complexity of $\mathcal{O}(t^2(t - d_t)) + \mathcal{O}(t^3) = \mathcal{O}(t^3)$.

6.2.3.5 Incremental post-processing

Incremental surface denoising The number of vertices on the border of $O_t \setminus O_{i_0 l}$ is $\mathcal{O}(t)$. For each vertex, the complexity of applying the Laplacian operator is same to the one in the batch case (Sec. 6.2.2.7). It is $\mathcal{O}(d)$, which is $\mathcal{O}(t)$. As a result, the complexity of incremental surface denoising is $\mathcal{O}(t^2)$.

6.3 Tight Complexity Analysis

The loose complexity analyses presented previously give theoretical complexities in the worst case. These results are obtained with few assumptions (H1 and H2 in Sec. 6.2.1) on the input data sets. However, the complexities are high and far from results of experiments in

Sec. 7.2.7 and Sec. 7.3.2. To better fit the complexities in experiments, a Cartesian grid bounding the input point cloud is introduced: we add vertices of the Cartesian grid in the Delaunay triangulation. Then tighter theoretical complexities which are close to complexities of experiments can be obtained thanks to these grid vertices. These vertices are not in the input SfM points thus can be seen as Steiner vertices and the Cartesian grid can also be called the Steiner grid. The density of this grid is guaranteed to be very low. It slightly changes the input data sets while results of the surface reconstructions, including both reconstructed surfaces and processing times, are almost not changed in experiments. In our work, the Steiner grid has essentially a theoretical interest: it guarantees tight theoretical complexities of our methods while it almost does not change the results in the reality.

6.3.1 Assumptions and Lemmas

6.3.1.1 Additional point assumptions

First, we assume that the density of the reconstructed points in 3d is bounded: there are $p > 0$ and $q > 0$ such that every p-ball contains at most q points. A p-ball is a ball with radius p. This could be justified as follows:

(1) only interest points are reconstructed and

(2) the scene surface has texture such that the interest points, which are detected due to gray-level 2d variations in their neighborhood, have a bounded density (e.g. no fractal-like texture).

Second, a Cartesian Steiner grid bounding the input points is added to the Delaunay triangulation. The adding of Steiner vertices is a standard method in Computational Geometry to generate meshes with good properties, e.g. to guarantee a linear-size 3d Delaunay triangulation with bounded vertex degree [18]. Every Steiner vertex has empty visibility list. In practice, the number n_s of Steiner vertices is quite smaller than the number n of reconstructed vertices.

The Steiner grid is added before the surface reconstruction, both in batch and incremental cases. Appendix E.3 shows that this can be done in $\mathcal{O}(n_s)$. In the incremental case, it would be better to add the Steiner vertices incrementally and during the surface reconstruction process. However the complexity analyses will be more complicated.

The voxel size is large enough such that:

1) the initialization above is not a problem from the computational viewpoint,

2) the number n_s of Steiner vertices is much smaller than n,

3) the effect of Steiner vertices on the reconstructed surface is negligible.

The voxel size can be chosen as a multiple (e.g. 10) of the expected mean of the camera step between two consecutive camera locations.

6.3.1.2 Bounded density of tetrahedra and bounded vertex degree

Sec. 6.3.1.1 implies that the tetrahedron density is bounded, i.e. there are $p' > 0$ and $q' > 0$ such that every p'-ball intersects at most q' tetrahedra.

Here is the proof. Since D is Delaunay, every tetrahedron has a circumsphere such that the interior does not contain a Delaunay vertex (Sec. 1.5). These vertices include the Steiner

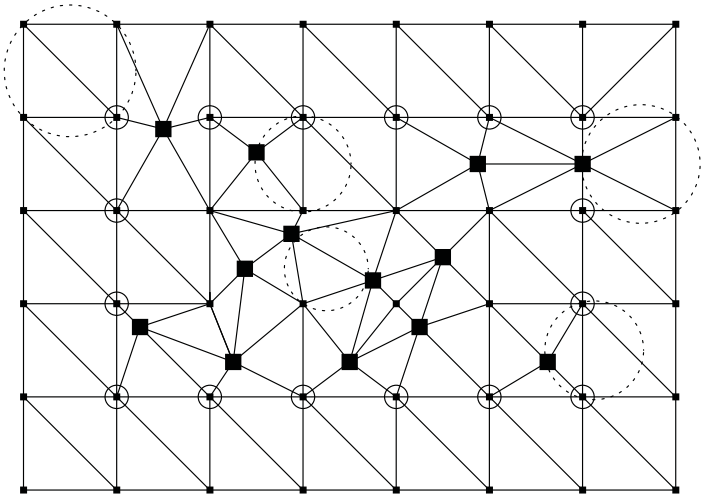


Figure 6.1: Delaunay mesh D (in the 2d case) for tight complexity analysis. Steiner and reconstructed points are small and large black squares, respectively. The dotted circles are circumcircles of triangles. The surrounded points by small circle are the border of the bounding box. This box is increased by an additional layer of Steiner points.

vertices of the grid. Then we show that the circumsphere diameter is less than (or equal to) the diagonal length l of the grid voxels in Appendix E.1. The proof requires that the bounding box of the grid is increased by an additional layer of voxels/Steiner vertices as shown in Fig. 6.1. Thus, the tetrahedron diameter is less than (or equal to) l.

Let $p' > 0$ and $L(\mathbf{z})$ be the list of tetrahedra which intersect the p'-ball centered at point $\mathbf{z}$. Then we show that the tetrahedra of $L(\mathbf{z})$ are in the $p' + l$-ball centered at $\mathbf{z}$, thanks to the triangular inequality. Here, let Δ be a tetrahedron in $L(\mathbf{z})$ and $\mathbf{a}$ is a vertex in Δ. Then there exists a point $\mathbf{b}$ in the p'-ball such that $\|\mathbf{b} - \mathbf{a}\| \leq l$ because Δ intersects the p'-ball and the diameter of Δ is less than (or equal to) l. Besides, $\|\mathbf{b} - \mathbf{z}\| \leq p'$ since $\mathbf{b}$ is in p'-ball, thus the triangular inequality implies that $\|\mathbf{a} - \mathbf{z}\| \leq \|\mathbf{a} - \mathbf{b}\| + \|\mathbf{b} - \mathbf{z}\| \leq p' + l$.

In $\mathbb{R}^3$, we can cover the $p' + l$-ball with a finite number b of p-balls (b does not depend on $\mathbf{z}$). According to Sec. 6.3.1.1, each p-ball contains at most q vertices. Thus the $p' + l$-ball contains at most bq vertices. Since a tetrahedron is a 4-tuple of vertices, we see that the number of tetrahedra in $L(\mathbf{z})$ is bounded. We conclude that the tetrahedron density is bounded.

If we choose $\mathbf{z}$ at a finite vertex of the Delaunay triangulation, we see that the maximum vertex degree d is bounded. It implies that the number of tetrahedra is also bounded, i.e. $|D| = \mathcal{O}(n)$.

6.3.1.3 List of assumptions and lemmas

The assumptions and lemma used in the tight complexity analyses are listed as follows.

- H1: the density of the point cloud is bounded.

- H2: the size of ray list of a 3d scene point is bounded.
- H3: the ray length is bounded.
- H4: in the incremental case, $\mathcal{O}(1)$ new 3d points are added to D at each time.
- H5: before the surface reconstruction, D is initialized by Steiner grid vertices in $\mathcal{O}(n_s)$. Here, n_s is the number of Steiner vertices which is much smaller than n.
- H6: the density of tetrahedra in the Delaunay triangulation is bounded.
- H7: the vertex degree is bounded.
- H8: adding one reconstructed point to D has $\mathcal{O}(1)$ complexity.
- H9: the number of tetrahedra in D is $\mathcal{O}(n)$.

H1, H5, H6, H7, H9 are already introduced (or shown) in Sec. 6.3.1.1 and Sec. 6.3.1.2.
H2 and H4 are lemmas already used in loose complexity analyses (Sec. 6.2.1).
H3 is the Lemma. 4.1 which is already shown in Sec. 4.2.
H8 is compatible with the fact that the incremental addition algorithm for 3d Delaunay triangulation is almost linear in practice [49]. In our work we conjecture that H5 and H6 imply H8. Details are given in Appendix E.3.

6.3.2 Batch Surface Reconstruction

Tight complexities of the batch surface reconstruction are analyzed. As in the loose case (Sec. 6.2.2), the number n of input SfM points and the number m of camera viewpoint locations are used to parameterize the complexities.

6.3.2.1 3d Delaunay triangulation

The complexity analysis of point filter is the same as in the loose case. Checking that if n points can be added in the 3d Delaunay triangulation D has a complexity of $\mathcal{O}(n)$. Thanks to H5 and H8, the complexity of 3d Delaunay triangulation construction is $\mathcal{O}(n_s + n)$. As $n_s < n$, the complexity becomes $\mathcal{O}(n)$.

6.3.2.2 *Free-space/matter* labeling

Before presenting the complexity of ray-tracing algorithm, we analyze the complexity of the brute force approach introduced in Sec. 4.3, to show that ray-tracing is indeed more efficient in terms of time complexity.

The brute force approach of *free-space/matter* labeling tries to check if each ray intersects each tetrahedron, thus it has a complexity of $\mathcal{O}(n_r n_D)$ with n_r the number of all rays and n_D the number of all tetrahedra in D. We know that the size of ray list of a scene point is bounded (H2), so $\mathcal{O}(n_r) = \mathcal{O}(n)$. Besides, n_D is $\mathcal{O}(n)$ (H9). Finally, the complexity of the brute force approach is $\mathcal{O}(n^2)$.

Regarding the ray-tracing approach, we first estimate the complexity of tracing one ray (line segment) $\mathbf{t}_i\mathbf{q}_j$. The ray is covered by a number of p'-balls which is linear to the Euclidean distance $||\mathbf{t}_i - \mathbf{q}_j||$. Since every p'-ball intersects at most q' tetrahedra (H6), $\mathbf{t}_i\mathbf{q}_j$ intersects $\mathcal{O}(||\mathbf{t}_i - \mathbf{q}_j||)$ tetrahedra. Furthermore, $\mathbf{t}_i\mathbf{q}_j$-tracing is a walk (Sec. 4.3) in graph D restricted to these tetrahedra and started from $\mathbf{q}_j$, which is a D vertex. Thus it has a complexity of $\mathcal{O}(d + ||\mathbf{t}_i - \mathbf{q}_j||)$. As $||\mathbf{t}_i - \mathbf{q}_j|| = \mathcal{O}(1)$ (H3) and $d = \mathcal{O}(1)$ (H7), tracing a ray has finally a complexity of $\mathcal{O}(1)$. Now ray-tracing is applied for all 3d points which have a ray list. The number of these points is $\mathcal{O}(n)$ and the size of the ray list is bounded (H2). So the complexity of the ray-tracing approach is $\mathcal{O}(n)$. Compared to the one of the brute force approach ($\mathcal{O}(n^2)$), we conclude that the ray-tracing approach is more efficient in terms of time complexity.

6.3.2.3 2-manifold generation

The complexity of the greedy region-growing algorithm is $\mathcal{O}((g + q_0)(\log(g + q_0) + d))$ (Sec. 6.2.2.4). Remember that g is the grown tetrahedra, q_0 is size of the tetrahedra list to initialize the heap Q and d is the maximum vertex degree. Now in the tight case, this complexity becomes $\mathcal{O}((g + q_0)\log(g + q_0))$ since $d = \mathcal{O}(1)$ (H7).

The 2-manifold generation is the greedy region-growing algorithm by starting the *outside* region O from $\emptyset$. Thus $q_0 = 1$ and the complexity of initializing Q should be added. Its complexity becomes $\mathcal{O}(|D| + g\log(g))$. In the tight case, $|D| = \mathcal{O}(n)$ and $g = \mathcal{O}(n)$. So finally, the complexity of the 2-manifold generation is $\mathcal{O}(n\log n)$.

6.3.2.4 Topology extension

Topology extension alternates two steps:

1. switch from *inside* to *outside* tetrahedra around a surface vertex $\mathbf{v}$ and check if vertices of all tetrahedra incident to $\mathbf{v}$ are regular.

2. region-growing in F starting from the neighborhood of these tetrahedra.

Step 2 only occurs if Step 1 is successful. This process is tried a finite number of times for every surface vertex.

There are $\mathcal{O}(n)$ surface vertices, and H7 implies that Step 1 is $\mathcal{O}(1)$ for one vertex. So the complexity of all steps 1 is $\mathcal{O}(n)$.

According to the result of Sec. 6.3.2.3, the i-th greedy region-growing from the i-th vertex (Step 2) has a complexity of $\mathcal{O}((g_i+q_i)(\log(g_i+q_i)))$ where g_i is the number of grown tetrahedra and q_i the number of tetrahedra to initialize the heap. We have $\sum_i g_i = \mathcal{O}(n)$ since the total number of tetrahedra is $\mathcal{O}(n)$. We have $\sum_i q_i = \sum_i \mathcal{O}(1) = \mathcal{O}(n)$ since $q_i = \mathcal{O}(1)$ thanks to H7. Thus,

$$\sum_{i\in I}(g_i + q_i)\log(g_i + q_i) \leq \sum_{i\in I}(g_i + q_i)\log(\sum_{i'\in I} g_{i'} + \sum_{i'\in I} q_{i'}) = \mathcal{O}(n(\log n)) \tag{6.5}$$

In summary, the complexity of topology extension is $\mathcal{O}(n\log n)$.

6.3.2.5 Spurious handles removal

The first method (Sec. 4.6.1) in the tight case has the same complexity as in the loose case: adding Steiner points along the camera trajectory has a complexity of $\mathcal{O}(m)$, which does not change complexities of other steps. Note that these Steiner points are those of the handle removal but not those of the Cartesian grid.

Now we study the second method (Sec. 4.6.2). Thanks to H7, D has $\mathcal{O}(n)$ edges and the complexity of step "Detect" for every edge is $\mathcal{O}(d+m) = \mathcal{O}(m)$. Then "Detect" is $\mathcal{O}(nm)$. Besides, the number w of critical edges is $\mathcal{O}(n)$.

In "Force" and "Repair", a Steiner vertex is added for each critical edge detected. And for each vertex $\mathbf{v}$ of a critical edge, including the Steiner vertex, two rounds of "Force" and "Repair" are applied to *free-space* and *inside* tetrahedra incident to $\mathbf{v}$. According to Sec. 6.2.2.6, both rounds have a complexity of $\mathcal{O}(d^2)$, which is $\mathcal{O}(1)$ in the tight case (H7). As there are $w = \mathcal{O}(n)$ critical edges, and each critical contains three vertices, the complexity of "Force" and "Repair" is $\mathcal{O}(n)$. We conclude that the the complexity of the second removal method is $\mathcal{O}(nm + n + n) = \mathcal{O}(nm)$.

6.3.2.6 Post-processing

Peak removal According to Sec. 6.2.2.7, the complexity of peak removal for each vertex is $\mathcal{O}(d^2)$, which is $\mathcal{O}(1)$ since $d = \mathcal{O}(1)$ (H7). The manifold S has $\mathcal{O}(n)$ vertices thus the complexity of the peak removal is $\mathcal{O}(n)$.

Surface denoising Remind that the denoised vertex $\mathbf{q}'$ of a vertex $\mathbf{q}$ in the surface S is calculated using $\mathbf{q}$ and $\mathcal{N}(\mathbf{q})$. Here, $\mathcal{N}(\mathbf{q})$ is the list of vertices which are one-ring neighbors of $\mathbf{q}$ in S. Thanks to H7, the calculation of $\mathcal{N}(\mathbf{q})$ is $\mathcal{O}(1)$. As S has $\mathcal{O}(n)$ vertices, the complexity of Surface denoising is $\mathcal{O}(n)$.

Sky removal Same to the loose complexity analysis in Sec. 6.2.2.7, the sky direction computation has a complexity of $\mathcal{O}(m)$ in the tight case. The complexity of rectangle-triangle intersection is $\mathcal{O}(nm)$ since S has $\mathcal{O}(n)$ triangles thanks to H6 and there are m open rectangles. The border growing is bounded by the number of S triangles ($\mathcal{O}(n)$). So the complexity of Sky Removal is $\mathcal{O}(nm)$.

In summary, the complexity of the post-processings is $\mathcal{O}(n + n + nm) = \mathcal{O}(nm)$.

6.3.3 Incremental Surface Reconstruction

Now we look at the tight complexity our incremental surface reconstruction method. Results are given using parameters t and $t - d_t$ as in Sec. 6.2.3. Remember that t is the time and d_t is the smallest creation date of the tetrahedra which are destroyed due to the addition of 3d points at time t.

6.3.3.1 Incremental 3d Delaunay triangulation

H2 and H4 imply that the point filter of input SfM points at time t is $\mathcal{O}(1)$. H4 and H8 imply that updating D_{t-1} to D_t by adding selected points and dating has a complexity of $\mathcal{O}(1)$. Finally, the complexity of the incremental 3d Delaunay triangulation is $\mathcal{O}(1)$.

6.3.3.2 Local ray-tracing

In the local ray-tracing step, our method uses only rays of 3d points whose creation dates are in the k most recent dates. The number of rays associated to points which have a same creation date is bounded thanks to H2 and H4. Thus the number of rays used for local ray-tracing is $k\mathcal{O}(1) = \mathcal{O}(1)$. As ray-tracing of one ray has $\mathcal{O}(1)$ complexity according to the analysis in Sec. 6.3.2.2, we can see that the complexity of the local ray-tracing step is $\mathcal{O}(1)$.

6.3.3.3 Incremental 2-manifold generation

The incremental 2-manifold generation is composed by several successive region-growings which grow in

$$F_t = \mathcal{L}_{(i_0-1)l} \cup \cdots \cup \mathcal{L}_t : \tag{6.6}$$

from $O_{i_0 l}$ to $O_{(i_0+1)l}$, from $O_{(i_0+1)l}$ to $O_{(i_0+2)l}$, ... from $O_{i_t l}$ to O_t. i_0 is the largest integer such that $i_0 l < d_t$ and i_t is the largest integer such that $i_t l < t$. Let $I = \{i_0, i_0 + 1, .., i_t\}$, we have $|I| = \mathcal{O}(t - d_t)$.

One region-growing has a complexity of $\mathcal{O}((g_i+q_i)\log(g_i+q_i))$ (already shown in Sec. 6.2.2.4 using $d = \mathcal{O}(1)$). g_i is the number of tetrahedra grown by this region-growing iteration, q_i is the number of tetrahedra in the initialized heap.

The complexity of the incremental 2-manifold generation is the complexity of the sum of successive region-growings: $\mathcal{O}(\sum_{i\in I}(g_i + q_i)\log(g_i + q_i))$. In Appendix E.2, we show for large values of $t - d_t$ that,

$$\sum_{i\in I} g_i = \mathcal{O}(t - d_t) \tag{6.7}$$

and

$$\sum_{i\in I} q_i = \mathcal{O}(t - d_t) \tag{6.8}$$

Then we have,

$$\begin{aligned}\sum_{i\in I}(g_i + q_i)\log(g_i + q_i) &\leq (\sum_{i\in I} g_i + \sum_{i\in I} q_i)\log(\sum_{i'\in I} g_{i'} + \sum_{i'\in I} q_{i'}) \\ &= \mathcal{O}((t - d_t)\log(t - d_t)).\end{aligned} \tag{6.9}$$

As a result, the complexity of the incremental 2-manifold generation is $\mathcal{O}((t-d_t)\log(t-d_t))$.

6.3.3.4 Incremental topology extension

The topology extension is applied each time after the successive region-growing. The complexity analysis is similar to the one in the loose case (Sec. 6.2.3.4).

Suppose that a region growing is already done which grows $O_{i-1}l$ to O_{il}. Then the incremental topology extension applies two steps to vertices of δO_{il}:

1. switch the set A of tetrahedra incident to a vertex $\mathbf{v}$ from *inside* to *outside*. Check if the resulting 2-manifold is still valid and reverse the switching if it is not the case. Here, all tetrahedra in A should be in F_{il}.

2. region-growing in F_{il} starting from a neighborhood of $\mathbf{v}$ (if Step 1 is successful).

The complexity of one Step 1 is $\mathcal{O}(d^2)$, which is $\mathcal{O}(1)$ in the tight case. The switched vertices are those of creation dates between $i_0 l$ and t. Let I be the indices of these vertices. Thanks to H4, $|I| = \mathcal{O}(t - i_0 l) = \mathcal{O}(t - d_t)$. So the complexity of all Steps 1 is $\mathcal{O}(t - d_t)$.

Step 2 is applied each time Step 1 succeeds. Thus the number of Steps 2 is $\mathcal{O}(t - d_t)$ in the worst case (all Steps 1 succeed). Besides, Step 2 is a region-growing algorithm whose complexity is $\mathcal{O}((g_i + q_i)\log(g_i + q_i))$ (Sec. 6.3.2.3). The complexity of all Steps 2 is thus $\sum_{i\in I}(g_i + q_i)\log(g_i + q_i)$.

Each time Step 2 is applied to a vertex $\mathbf{v}$, the heap is initialized by using adjacent tetrahedra of the tetrahedra which are incident to $\mathbf{v}$. Thus $q_i = \mathcal{O}(d)$, which is $\mathcal{O}(1)$ in the tight case. We have $\sum_{i\in I} q_i = \mathcal{O}(t - d_t)$. g_i is the number of tetrahedra grown in F_t by each Step 2. We have $\sum_{i\in I} g_i \leq |F_t|$. $F_t = \mathcal{O}(t - d_t)$ according to Eq. E.12 in Appendix E. Thus the complexity of $\sum_{i\in I} g_i$ is $\mathcal{O}(t - d_t)$.

Finally the complexity of all Steps 2 is,

$$\begin{aligned}\sum_{i\in I}(g_i + q_i)(\log(g_i + q_i)) &\leq (\sum_{i\in I} g_i + \sum_{i\in I} q_i)\log(\sum_{i'\in I} g_{i'} + \sum_{i'\in I} q_{i'}) \\ &= \mathcal{O}((t - d_t)\log(t - d_t))\end{aligned} \tag{6.10}$$

In summary, the incremental topology extension has a complexity of $\mathcal{O}((t - d_t)\log(t - d_t))$.

6.3.3.5 Incremental post processing

Surface denoising The tetrahedra of $O_t \setminus O_{i_0 l}$ are in F_t and $|F_t| = \mathcal{O}(t - d_t)$. Thus, the number of vertices on the border of $O_t \setminus O_{i_0 l}$ is less than $4|F_t|$, which is $\mathcal{O}(t - d_t)$. The calculation of the discrete Laplacian for one vertex is $\mathcal{O}(1)$ thanks to H7. As a results, the time complexity of the incremental surface denoising is $\mathcal{O}(t - d_t)$.

6.4 Summary of Complexities

Table 6.1 summarizes the results of our complexity analyses. The batch surface reconstruction has a loose complexity of $\mathcal{O}(n^4)$ and a tight complexity of $\mathcal{O}(n\log n + nm)$. The incremental surface reconstruction has a loose complexity of $\mathcal{O}(t^3(t - d_t))$ and a tight complexity of $\mathcal{O}((t - d_t)\log(t - d_t))$.

	Loose Batch	Tight Batch	Loose Incremental	Tight Incremental
DT	n^2	n	t^2	1
RT	n^3	n	t^2	1
MG	n^3	$n \log n$	$t^3(t - d_t)$	$(t - d_t) \log(t - d_t)$
TE	n^3	$n \log n$	t^3	$(t - d_t) \log(t - d_t)$
HR2	n^4	nm	-	-
PP	n^3	nm	t^2	$t - d_t$
Global	n^4	$n \log n + nm$	$t^3(t - d_t)$	$(t - d_t) \log(t - d_t)$

Table 6.1: Summary of the worst case time complexities. DT: Delaunay Triangulation construction including the first method of spurious handle removal, RT: Ray-Tracing, MG: 2-Manifold Generation, TE: Topology Extension, HR2: second Handle Removal method, PP: Post-Processing, Global: the sum of all steps. The sign $\mathcal{O}()$ is omitted and "-" means does not exist.

Globally, both our surface reconstruction methods have high theoretical complexities in the loose case where very few assumptions are made on the input data sets. Now in order to obtain tight theoretical complexities, a Steiner grid bounding the input point cloud is added to input points which allows our methods to generate a linear-size Delaunay triangulation. Additional assumptions are made thanks to the grid and we obtain tight complexities of our surface reconstruction methods which fit better the experimental results in Sec. 7.2.7 and Sec. 7.3.2.

The tight complexity analysis of the batch method is also done in [1]. We think that the complexities of our methods could be improved by using hierarchical bounding boxes to accelerate the detection of HR2 and the rectangle-triangle intersection test of the sky triangle removal post processing. We do not investigate this in depth since this is far from the core of our contribution (2-manifold enforcement).

Chapter 7

Experiments

Our experiments are done in three steps. First, image or videos sequences are taken in the environment that we want to model. Second, sparse SfM points and camera poses are reconstructed by the geometry reconstruction. At last, the surface reconstruction is applied to reconstruct the surface model of the environment.

Equipments used in our experiments are firstly introduced in Sec. 7.1. Then detailed results of our batch and incremental environment modeling methods are respectively presented in Sec. 7.2 and Sec. 7.3. At last, we provide other experiments without details in Sec. 7.4.

In this chapter, the suffix k means $\times 1000$. Besides, we provide also a lot of figures showing experiments results. Readers are recommended to view them in the PDF version of this dissertation or the version printed in colors.

7.1 Equipments

The necessary equipments for our image-based environment modeling methods are very simple in order to be adequate for the 3d modeling of complete environments at low cost. An equiangular catadioptric camera is needed for image sequence acquisition and both 3d geometry and surface reconstruction are done using a standard PC.

The catadioptric camera is hand/helmet-held, and it is composed by a convex mirror and a standard perspective camera. Our convex mirror is a 0-360 mirror [1] mounted in front of the perspective camera thanks to an additional adapter ring. The perspective camera is pointing roughly oriented to the sky when our image sequence is taken. It has a view field of 360° in the horizontal plane and about $50^\circ - 60^\circ$ above and below the horizontal plane. Here, two catadioptric cameras are used in experiments. One is a 0-360 mirror with a Nikon Coolpix 8700, which is showed on the left of Fig. 7.1. It is used to take still 3264×2448 JPEG images sequences used for our batch modeling method. The other is a 0-360 mirror with a Canon Legria HFS10, which is showed on the right of Fig. 7.1. It is used to take 1920×1080 AVCHD (MP4) video sequences used for the both batch (Sec. 7.2.10) and incremental modeling methods. Both still image and video sequences are down-sampled

[1] http://www.0-360.com/

by 2 to accelerate the subsequent SfM steps. In addition, the AVCHD video is interlaced, so each video frame is read one line in two.

Figure 7.1: Catadioptric cameras used in experiments. Left: 0-360 mirror + Nikon Coolpix 8700; Right: 0-360 mirror + Canon Legria HFS10

The PC used in experiments is an Intel Core^{TM} Duo E8500 at 3.16GHz. Our methods are implemented without multi-threading or other parallel techniques. Only one CPU core is used during the program execution.

7.2 Batch Environment Modeling

Sec. 7.2.1 provides an overview of our batch environment modeling results in case of a still image sequence. It includes also a summary of the pre-processing geometry reconstruction (SfM) results. Next, we experiment and discuss the manifold constraint in Sec. 7.2.2. Then we show and discuss the surface reconstruction results by varying different parameters, algorithms or inputs: in Sec. 7.2.3, we use different choices of function r, i.e. the function which calculates the priority score in region-growing; in Sec. 7.2.4 and 7.2.5, we vary respectively densities of Steiner points and those of the reconstructed SfM 3d points; and in Sec. 7.2.6, we compare two methods of spurious handle removal. After that, the processing time of our batch surface reconstruction is analyzed in Sec. 7.2.7. And then, a comparison with the Poisson surface reconstruction [76] is done in Sec. 7.2.8. A quantitative evaluation using a synthetic image sequence is also given in Sec. 7.2.9 and finally, we show results of the environment modeling using a 1.4 km long video sequence in Sec. 7.2.10.

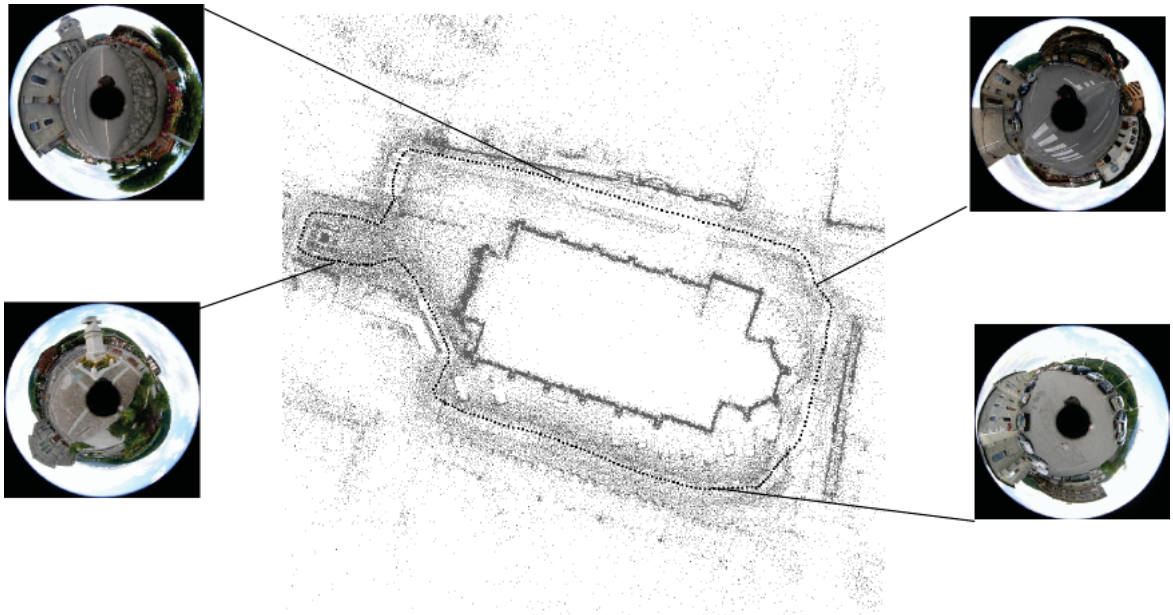


Figure 7.2: Overview of the batch SfM. Top view of the 136 k 3d points and 343 camera poses reconstructed by SfM and several images of the sequence. 3d points are in gray and camera poses are in black.

7.2.1 Overview of Results

Except in Sec. 7.2.9 and 7.2.10, we use a still JPEG image sequence of 343 images during a complete walk around a church of a commune in France called Grand Bornand. The trajectory length is about $(30 \pm 5 \text{ cm}) * 343 = 103 \pm 17$ m. The exact step lengths between consecutive images are unknown. The radii of large and small circles of the catadioptric images are 574 and 103 pixels.

Batch Geometry Reconstruction (SfM) Fig. 7.2 shows top view of our SfM, which reconstructs 343 camera poses and 136k 3d points from 343 images. The radial function $r(\alpha)$ (Sec. 3.1.1.2) in our central catadioptric camera model, which maps the angle between a ray direction and the mirror symmetry axis to a 2d image pixel, is $1.40278 - 0.618567\alpha + 0.0394814\alpha^2 - 0.00308602\alpha^3$ after the final bundle adjustment. 872k Harris (inlier) points are detected and 136k 3d points are reconstructed. In average, the number of detected Harris points is 2.5k per image and the ratio between the number of 3d points and images is 397 points/image. A 3d point has in average 6.4 tracks in images, i.e. 6.4 rays. A ray (and also its corresponding 2d point) is considered as outlier if its reprojection error is greater than 2 pixels. The RMS (Root-Mean-Square) error of the final bundle adjustment is 0.74 pixels.

Batch Surface Reconstruction Fig. 7.3 shows the surface estimated using $\epsilon = 10°$ (Delaunay Step), without the Steiner vertices of the tight time complexity (Sec. 6.3), with the second Handle Removal method R2 (Sec. 4.6.2) and $\alpha = 5°$, and using post-processing

Figure 7.3: Overview of the batch surface reconstruction. Top and oblique views of the reconstructed surface (152k triangles), including triangle normals (gray) and trajectory (black).

thresholds: $w_0 = \pi/2$ steradians (Peak Removal), $\beta = 45°$ (Sky Removal), and $k_t = 10$ (Texturing). The VRML model has 152k triangles, which are few for a scene like this. It is done in 35 s including 7.6 s for Delaunay Step, 2.9 s for *Free-space/matter*, 2.5 s for 2-Manifold Generation, 2.9 s for Topology Extension, 16.8 s for Handle Removal and 2.5 s for Post-Processing (ignoring Texturing). The 3d Delaunay is constructed with 81k selected vertices, and it has 500k tetrahedra (without $\mathbf{v}_\infty$). 47.1% of these tetrahedra are *free-space*, 88.6% of the *free-space* tetrahedra are *outside* by the 2-Manifold Generation, and 92.0% of the *free-space* tetrahedra are *outside* at the end of the process. More details are given in the following sections.

7.2.2 2-Manifold Constraint

Here are additional informations on the 2-manifold constraint in the experiment of Sec. 7.2.1. First, there are several methods to test if a surface is manifold (Sec. 4.4.3.1). We use the single tetrahedron test for the 2-Manifold Generation step (adding the tetrahedra one-by-one in O), and the tetrahedron-based manifold test for the Topology Extension step (adding the tetrahedra several-at-once in O). In our implementation, the edge-based test is slower: if we use it, it multiplies the computation time of the 2-Manifold Generation step by 1.4 and that of the Topology Extension step by 1.9. We also check that the different tests provide the same surface.

Now, we study the advantage of the 2-manifold constraint. We start from two closed surfaces: the original surface, i.e. the border of the *free-space* tetrahedra which is non-manifold, and the 2-manifold surface, i.e. the border of the *outside* tetrahedra generated by the 2-Manifold Generation step which is manifold. In the original surface, 24.62% of the surface vertices are singular. Then the denoising step is applied to both surfaces, and finally we have four surfaces: denoised original surface, non-denoised original surface, denoised manifold, and non-denoised manifold. For the remainder of Sec. 7.2.2, we do not apply the further steps of our method: Topology Extension, Spurious Handle Removal, Post-processings except texturing.

Fig. 7.4 shows that the manifold constraint helps surface denoising: the denoised manifold surface is smoother than the denoised original surface. The use of several denoisings does not change this observation.

Remind that our denoising method is based on the uniform discrete Laplacian operator [135], where a denoised vertex is a sum of the weighted vertices in its immediate neighborhood on the surface. Although this method is simple, it improves the texturing step. Indeed, Fig. 7.4 also shows that the texturing of the denoised 2-manifold is better than the texturing of the non-denoised 2-manifold.

7.2.3 Varying Function r in Region-growing

Remind that our target surface depends on function r: the 2-Manifold Generation (Sec. 4.4) is a best-first region growing of the *outside* tetrahedra in the *free-space* tetrahedra, and the growing order is defined by r. In this section, we compare the performance of the 2-Manifold Generation for several choices of r. Let Δ be a *free-space* tetrahedron. We try

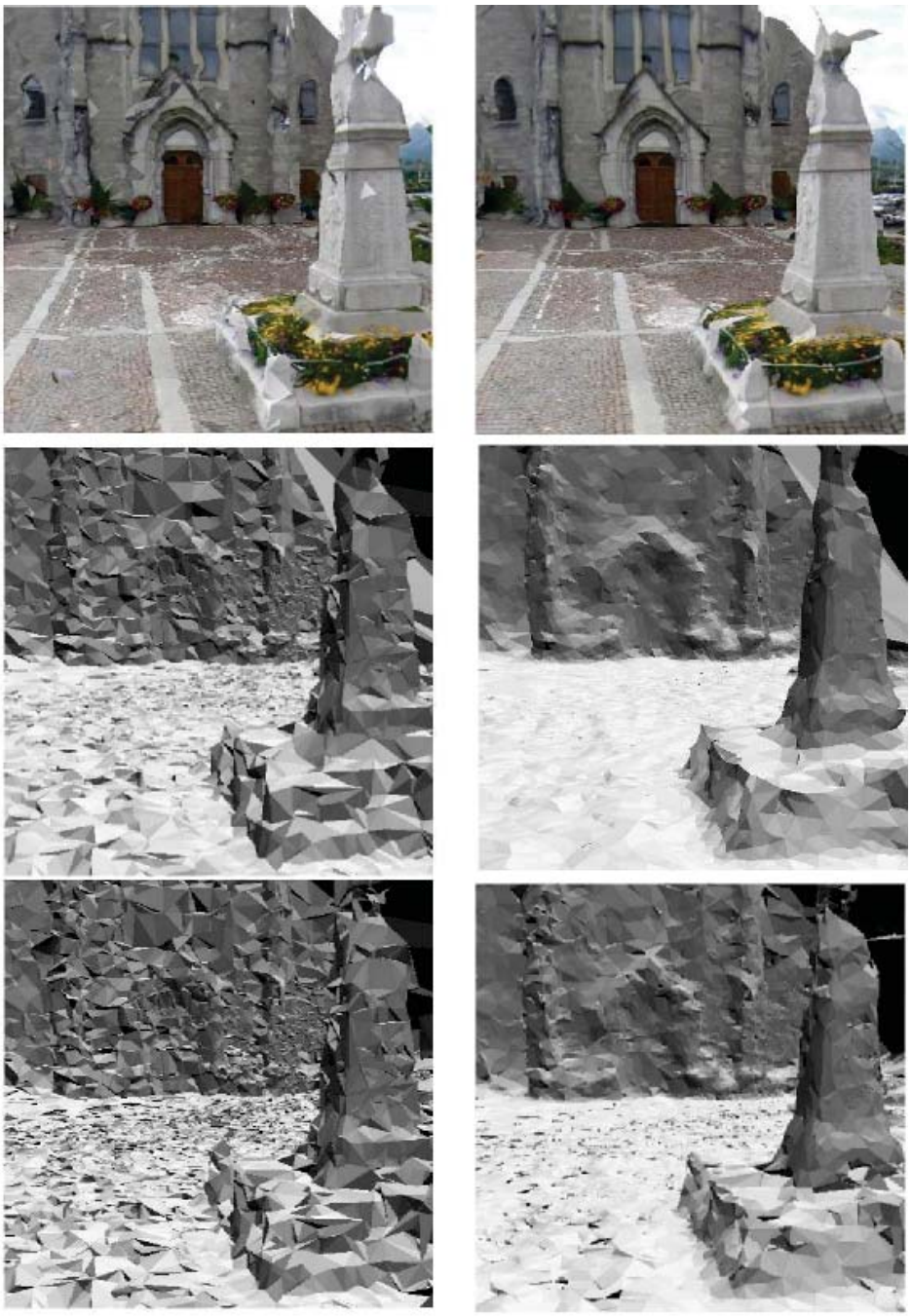


Figure 7.4: Importance of the 2-manifold constraint and surface denoising. With (top and middle) and without (bottom) the manifold constraint. Without (left) and with (right) surface denoising.

- $r_{random}(\Delta)$, a random number in interval $]0, 1[$
- $r_{edge}(\Delta)$, the largest length of the 6 edges of Δ
- $r_{volume}(\Delta)$, the volume of Δ.
- $r_{circumradius}(\Delta)$, the radius of the circumsphere of Δ
- $r_{visibility}(\Delta)$, the number of rays which intersects Δ (default choice).

A quantitative comparison is done for the input data in Sec. 7.2.1 using *free-space/outside* ratio. Here, remember that the list O of *outside* tetrahedra grows in the list of *free-space* tetrahedra, and the ratio between the numbers of *outside* and *free-space* tetrahedra, called *free-space/outside* ratio, can be used to compare the performances of the growing steps (2-Manifold Extraction and Topology Extension). The five r-choices are ordered from worse to best: the percentages are $p_{random} = 75.02\%$, $p_{edge} = 86.77\%$, $p_{volume} = 87.03\%$, $p_{circumradius} = 87.74\%$, $p_{visibility} = 88.62\%$. Here we do not use the further steps of our method. If we use them, the final results are similar (except for "random" which is slightly worse).

7.2.4 Varying Density of Steiner Grid Vertices

Here we study the effects of the Steiner grid introduced in Sec. 6.3.2 if it is added to the input point cloud. Remember that the Steiner grid is a Cartesian grid of Steiner points where the Steiner vertices are located at the corners of the grid. The length of voxel edges of the grid is a multiple sz of the camera step between two consecutive images (the length is sz $\text{mean}_j||\mathbf{c}_{j+1}-\mathbf{c}_j||$). We use the CGAL [7] implementation of the 3D Delaunay triangulation, which deals with the degenerate configurations of vertices as those of the grid. However, the computation of the degenerate cases is costly and we prefer to perturb slightly and randomly the grid vertex locations to accelerate the computation. This does not compromise the bounded size of the Delaunay tetrahedra.

sz	s/m	$\|T\|/v$	d	DT(s)	RT(s)	ME(s)	TE(s)	HR2(s)
10	0.079	6.11	386	9.0	2.7	2.4	2.2	15.1
20	0.016	6.16	550	7.7	3.0	2.5	2.9	15.8
∞	0	6.19	296	7.6	2.9	2.5	2.9	16.8

Table 7.1: Statistical results of surface reconstruction using several grids. From left to right: edge length coefficient, ratio between the numbers of Steiner grid vertices and reconstructed vertices, ratio between numbers of tetrahedra and all vertices, vertex degree, time in seconds of Delaunay/Ray-tracing/2-Manifold Generation/Topology Extension/Handle Removal Method 2 (Detect-Force-Repair).

Fig. 7.5 shows views for $sz \in \{2, 5, 10, 20\}$. We see that the surface quality is degraded if sz becomes smaller. When sz is 10 or 20, the reconstructed surface is almost the same. The negative effect of the Steiner vertices becomes noticeable when sz is under 10 in the areas

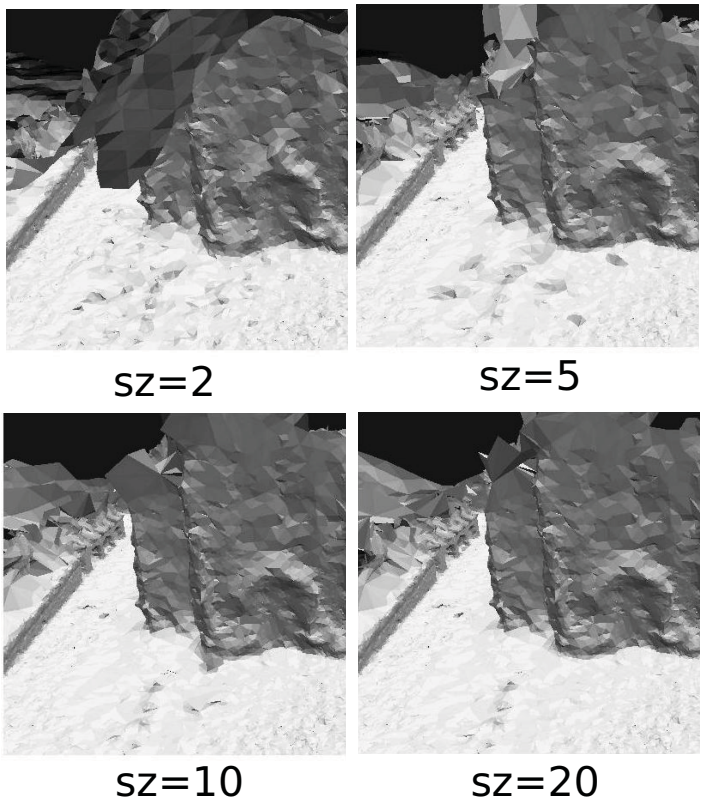


Figure 7.5: Effect of grids of Steiner vertices.

where the density of reconstructed vertices is low (sky and parts of ground in our example). In the sky, there are too much tetrahedra for a fixed number of rays, then small *matter* tetrahedra appears if sz is small. Thus sz should not be too small.

Tab. 7.1 shows the experimental computation times for large enough sz ($sz = \infty$ means the Steiner grid is not used). We can see that the grid does not really have complexity advantage in practice, and we conclude that the grid has essentially a theoretical interest: it guaranties the worst case complexities derived in Sec. 6.3.2. The fact that the smallest d in Tab. 7.1 is obtained without the grid might be surprising, but this is not in contradiction to the fact that every finite grid size sz guaranties an upper bound for d (Sec. 6.3.1.2). Note that d can be smaller than $d(sz = \infty)$, e.g. $d(sz = 5) = 264$.

7.2.5 Varying Density of SfM 3d Points

Since our goal is the 2-manifold estimation from a small number of SfM points (and their visibility), it would be interesting to experiment on the same data as in Sec. 7.2.1, but with a yet smaller number of points. Thus we reduce the size of the images by several coefficients. Then for each resulting sequence, the whole process, including both SfM and surface reconstruction, is applied with the same parameter settings. This reduces the number of reconstructed points as if we use a different camera to reconstruct the same scene. The

image reduction also increases the image noise impact on the 3d result. We think that this experiment is more interesting than the surface calculations from random selections of the SfM points using the original image sequence in Sec. 7.2.1.

rc	SfM	m	$\|T\|/v$	d	$\|\delta O\|$	Time (s)
1	136k	81k	6.19	296	152k	35
1.5	65k	40k	6.21	620	76k	20
2	37k	23k	6.22	386	44k	12
2.5	23k	14k	6.23	501	28k	9.8

Table 7.2: Statistical results on reduced sequences. From left to right: reduction coefficient, number of reconstructed points, number of Delaunay vertices, ratio between the numbers of tetrahedra and vertices, vertex degree, number of triangles, surface computation time in seconds.

Tab. 7.2 provides statistical results using reduction coefficients $rc \in \{1, 1.5, 2, 2.5\}$. The number of reconstructed points is roughly linear to $1/(rc)^2$, the ratio between the numbers of tetrahedra and vertices of the Delaunay is almost constant, and the calculation times are very small for large rc (a few seconds). Fig. 7.6 shows the surfaces for different rc values. We can see the progressive reduction of the level of details when rc increases.

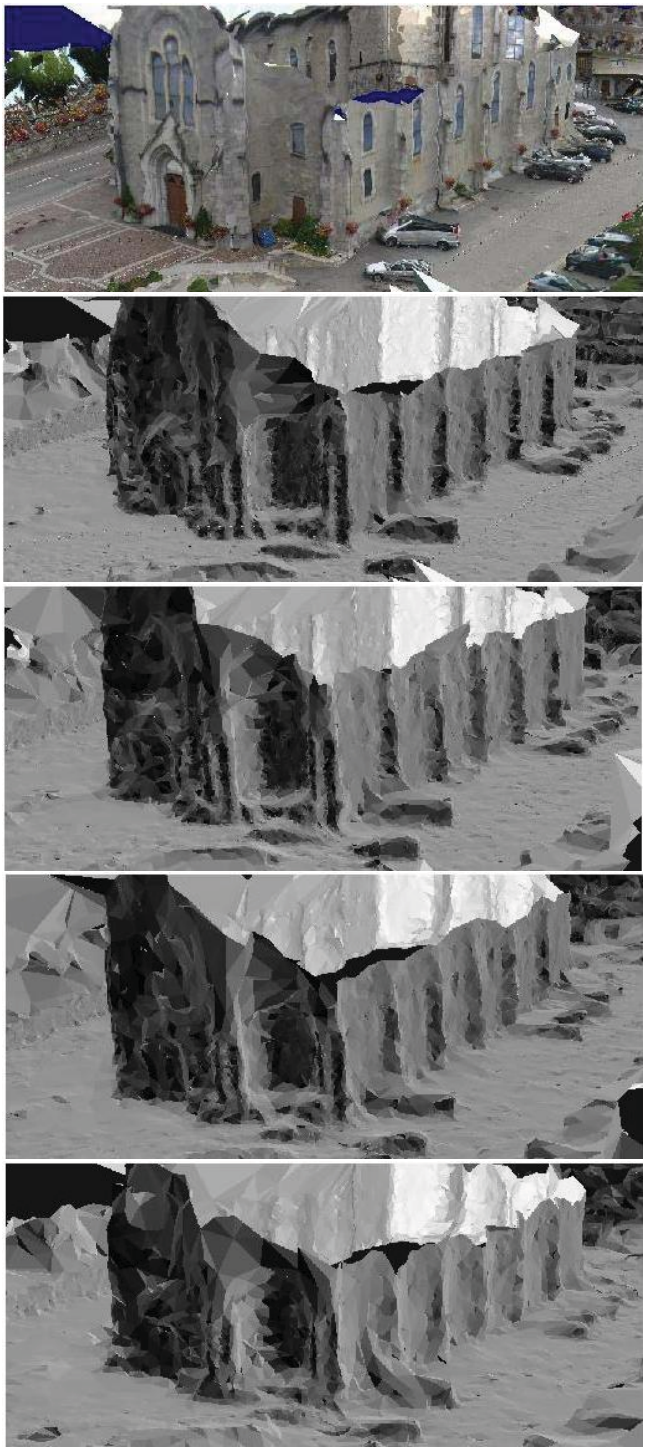


Figure 7.6: Surfaces for several densities of reconstructed points. The density is defined by coefficient rc. From top to bottom: texture and normals for $rc = 1, 1, 1.5, 2, 2.5$.

7.2.6 Varying Handle Removal Method

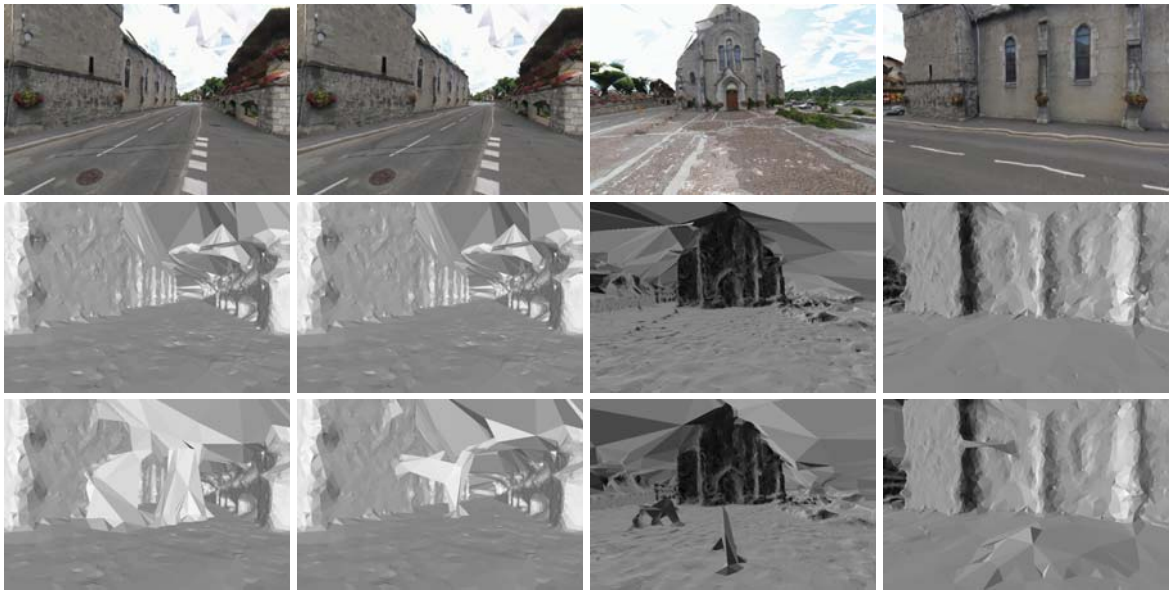


Figure 7.7: Comparison of the two handle removal methods. Results using R1 are at the bottom and results using R2 are at the top and the middle. Each column shows the results for given number of Steiner points per camera pose for the first method (from left to right: 0, 1, 5, 10).

Fig. 7.7 compares the two handle removal methods. Method $R1(ns)$ (Sec. 4.6.1) adds randomly a given number ns of Steiner points in the ball neighborhood of every camera location. The radius coefficient is 10. Method $R2$ (Sec. 4.6.2) detects, forces and repairs the spurious handles. We compare $R2$ to $R1(ns)$ with several values of $ns \in \{0, 1, 5, 10\}$ ($ns = 0$ means that the handle removal is not applied). The right column shows the largest spurious handles of the surface if $ns = 0$. These handles are reduced by $R1(1)$ and are removed by $R2$ and $R1(ns), ns \in \{5, 10\}$. However, $R1(ns), ns \in \{1, 5, 10\}$ are not able to remove spurious handles at other locations (two examples are shown in the two columns on the right). Furthermore, we notice that ns should not be too large for the same reasons as sz should not be too small in Sec. 7.2.4. $R2$ gives the best results and removes the large handles at the locations viewed by Fig. 7.7 (and others). Note that the sky is not removed in these experiments since we would like to examine the Handle Removal step only. Indeed, Sky Removal also removes spurious handles in a special case: if they are above the camera locations (they are intersected by the open rectangles of Sec. 4.7.3).

Remind that the main parameter of $R2$ is the angle α, which selects the minimal size of critical edges where the computations are done. We found that $\alpha = 5°$ is a good compromise: (1) $R2$ misses visually noticeable handles if α is larger than 5 and (2) the calculation times increases while the improvement is negligible if α is smaller than 5. For this image sequence, 1.5% of the 581k Delaunay edges are critical edges using $\alpha = 5$.

7.2.7 Experimental Complexities

In Sec. 7.2.5, 3d points of different densities are reconstructed by SfM with images of different resolutions, and these points are used for batch surface reconstruction. Their processing time results can be used to analyze the experimental time complexities of our method. Note that the number of camera viewpoints m does not change in experiments since we vary only input image resolution.

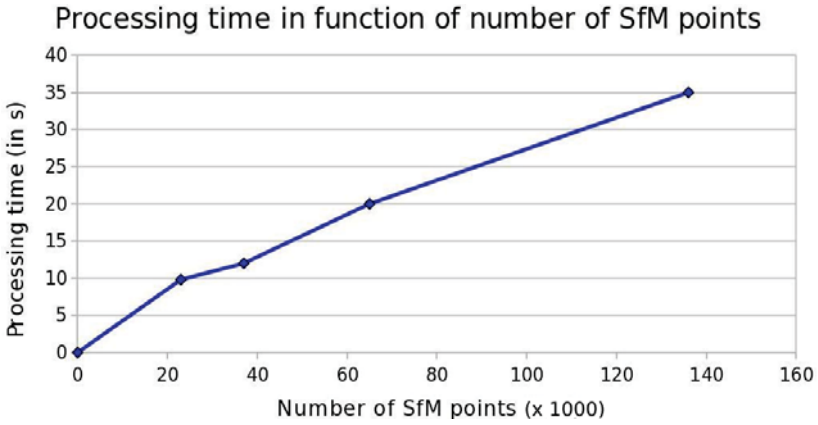


Figure 7.8: Processing time of batch surface reconstruction as a function of the number of input SfM points.

Fig. 7.8 illustrates the relationship between the processing time of the batch surface reconstruction and the number n of SfM points. We see that the experimental time complexity of our method is roughly $\mathcal{O}(n)$. It is slightly better than the tight theoretical complexity in the worst case of our batch surface reconstruction $\mathcal{O}(n \log n + nm)$ (Sec. 6.3.2), which is $\mathcal{O}(n \log n)$ in this case since m does not change.

7.2.8 Comparison with Poisson Surface Reconstruction

Our method is also compared to the Poisson Surface Reconstruction [76]. A summary of this method is already given in Sec. 2.2.2.1. Remind that it takes a set of points and their oriented normals as input, and the reconstructed surface is a triangulated 2-manifold approximating input points, whose size is moderated thanks to an octree-based sampling. We use the implementation[1] of the authors with parameter depth = 12 to obtain a large enough resolution.

Several surfaces are reconstructed by using different methods to compute the oriented normal $\mathbf{n}_i$ of reconstructed point $\mathbf{q}_i$:

- S_{ray}: $\mathbf{n}_i$ is the mean of directions of all $\mathbf{q}_i$'s rays.
- S_{6nn}: $\mathbf{n}_i$ is the normal of the plane Π minimizing the sum of the squared point-plane distances between Π and the 6-nearest neighbors of $\mathbf{q}_i$.

[1] http://www.cs.jhu.edu/ misha/Code/PoissonRecon/

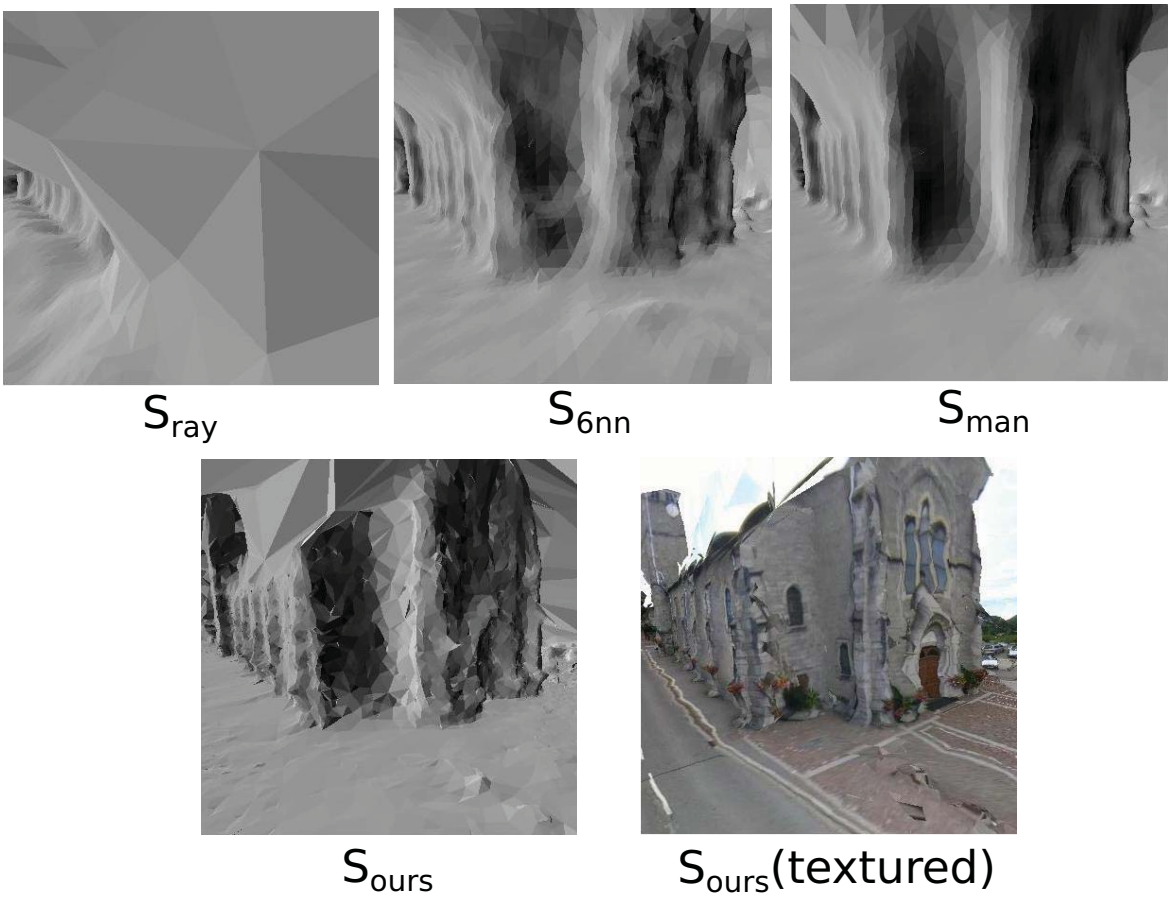


Figure 7.9: Comparison of surfaces reconstructed by our method and Poisson. Different surfaces are viewed from the same viewpoint.

- S_{man}: $\mathbf{n}_i$ is a weighted sum of normals of the $\mathbf{q}_i$-incident triangles separating *outside* and *inside* tetrahedra.

Besides, let S_{ours} be the surface of our method. Notice that the normal estimation of S_{man} is obtained using the reconstructed surface of our method. Furthermore, the normal weight is the incident triangle angle at $\mathbf{q}_i$ [27], and $\mathbf{q}_i$ is not used by S_{man} if its normal cannot be computed due to the $\mathbf{q}_i$-tetrahedra labeling. To calculate $S_{\text{ray}}, S_{6\text{nn}}, S_{\text{man}}$ and S_{ours}, the point filtering of Sec. 4.2 is applied and there is no Sky Removal step.

Fig. 7.9 shows views of the four surfaces. As shown by the experiments, the better accuracy of normal computation, the better quality of the Poisson surface: S_{man} is better than $S_{6\text{nn}}$, which is better than S_{ray}. The quality of S_{ours} is between those of $S_{6\text{nn}}$ and S_{man}, although S_{man} is oversmoothed compared to S_{ours}. Regarding the processing time, S_{ours} has 154k triangles computed in 35 s and S_{man} has 178k triangles computed in 128 s, whose computation time is about 3.5 times of that of S_{ours}.

7.2.9 Quantitative Evaluation

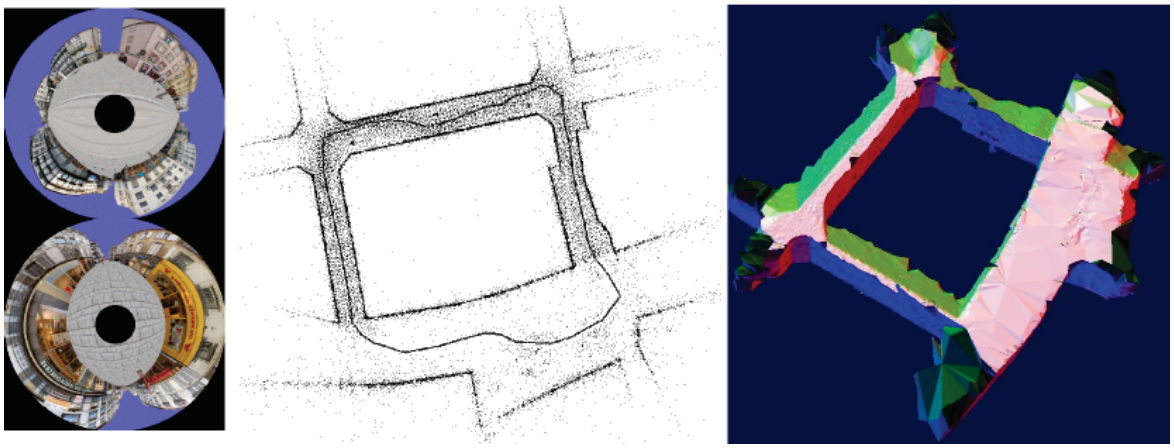


Figure 7.10: Reconstruction results of synthetic image sequence. From left to right: two synthetic catadioptric images, top view of sparse point cloud (and camera trajectory) by SfM, top view of the reconstructed surface (the colors encode the triangle normal).

The quantitative evaluation of our surface reconstruction methods is done by using a synthetic image sequence. The evaluation process can be summarized as follows.

- First, a synthetic scene is manually generated using images taken in a real environment.
- Second, a synthetic catadioptric image sequence is generated using the synthetic scene.
- Third, the batch SfM and surface reconstruction are applied to the synthetic image sequence.
- Fourth, the reconstructed surface is registered to the ground truth surface, i.e. the manually generated scene in the first step.
- At last, an error function is defined and used to measure errors between the reconstructed surface and the ground truth surface.

Firstly, the synthetic scene should be manually generated. In our work, we use a synthetic model which is provided by CRISTAL project[1] for the quantitative evaluation. The advantage of this model is that it uses textures and dimensions of a real environment thus the scene and the subsequent image sequence acquisition are more realistic. To generate this model, a perspective camera is used to take real images in a city and a "simple" 3d model of part of the city is manually generated and textured using data of GPS-RTK and a telemeter. Notice that this 3d model is not necessarily accurate since our objective is to reconstruct a 3d surface which approximates this 3d model, but not the real environment. For reminder

[1] An innovation project about the future public transport conducted by LOHR Industry, Transitec, INRIA, Vulog, UTBM and Institut Pascal.

of this section, the ground truth in the evaluation process refers to this 3d model in a metric coordinate system.

The synthetic catadioptric image sequence generation is a computer graphic method. It uses ray-tracing techniques and takes into account the ray reflection on the mirror of catadioptric camera. The trajectory of the sequence is a 230 m long closed loop around a building including several shops. The large circle, which contains the scene projection in the image, has a 600 pixel radius. Fig. 7.10 (left) shows two images of the synthetic sequence.

The synthetic image sequence is then used by the batch SfM to reconstruct the geometry of the scene, which includes 600 camera poses and a sparse cloud of 257k 3d points (middle in Fig. 7.10). Our batch surface reconstruction method is subsequently applied. The reconstructed surface (right in Fig. 7.10) has 120k vertices and 241k triangles. And the ratio of *free-space/outside* tetrahedra is 90.96%.

Now the reconstructed surface is registered to the ground truth surface. Let T_e be $[\mathbf{t}_e^0\ \mathbf{t}_e^1\ ...\ \mathbf{t}_e^{599}]$ and T_g be $[\mathbf{t}_g^0\ \mathbf{t}_g^1\ ...\ \mathbf{t}_g^{599}]$. Here, $\mathbf{t}_e^i$ and $\mathbf{t}_g^i$ are respectively the estimated locations and the ground truth locations of the camera i ($\mathbf{t}_e^i$ is at the camera center and $\mathbf{t}_g^i$ is at the mirror apex). We firstly estimate transformation matrix Z using a linear method based on the singular value decomposition [46]. Then by using Levenberg-Marquardt algorithm, Z is refined such that $E(Z) = \sum_{i=0}^{599} ||Z(\mathbf{t}_e^i) - \mathbf{t}_g^i||^2$ is minimized. We found $\sqrt{E(Z)/600} = 5.1$ cm and use Z to map the reconstructed surface in the ground truth coordinate system.

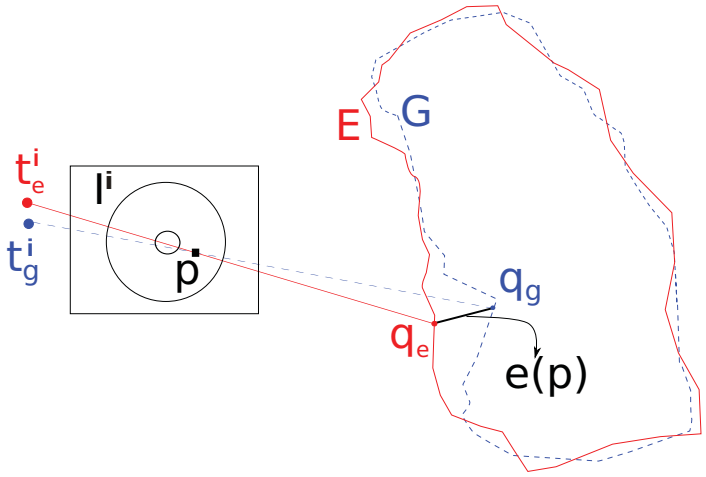


Figure 7.11: An example of error function. $\mathbf{t}_e^i$, $\mathbf{t}_g^i$ are respectively the i-th reconstructed camera location and ground truth camera location, $\mathbf{p}$ is a pixel of the i-th image, E, G are respectively the reconstructed surface and the ground truth surface, and $\mathbf{q}_e, \mathbf{q}_g$ are respectively intersected point of the back projected ray of $\mathbf{t}_e^i$, $\mathbf{t}_g^i$ with E, G. Error $e(\mathbf{p})$ is the distance between $\mathbf{q}_e$ and $\mathbf{q}_g$.

The error function e to measure the difference between the reconstructed surface and the ground truth surface should be defined. Traditional error function $e(\mathbf{q})$ [124] measures the distance between the ground truth surface and vertex $\mathbf{q}$ of the reconstructed surface. However it is not suitable in the sparse case since this error is biased in favor of reconstructed areas which have the largest densities of reconstructed points. Furthermore, the closest point in

the ground truth surface does not necessarily correspond to the same point $\mathbf{q}$.

We prefer an error function based on ray tracing, as seen in Fig. 7.11. Let $\mathbf{p}$ be a pixel in an image of the sequence. Let $\mathbf{q}_e$ be the intersection of the reconstructed surface and the back-projected ray of $\mathbf{p}$ from the reconstructed camera location. Let $\mathbf{q}_g$ be the intersection of the ground truth surface and the back-projected ray of $\mathbf{p}$ by the ground truth camera location. In both cases, if there are several intersections, we take the intersection which is the closest to the camera location. Then we use error $e(\mathbf{p}) = ||\mathbf{q}_e - \mathbf{q}_g||$. If $\mathbf{q}_g$ does not exist or $e(\mathbf{p}) > \mu_0$ (where $\mu_0 = 2$ m), we assume that the point matching $(\mathbf{q}_e, \mathbf{q}_g)$ is outlier (e.g. pixels corresponding to the sky). In practice, the statistic of $e(\mathbf{q})$ is estimated by uniform sampling in all images of the sequence. 6×10^6 pixels are sampled in the sequence. 75.7% of sampled pixels are inliers, the error median of inliers is 8 cm and the 90% quantile of inliers is 55 cm.

Lastly, the same experiment (both SfM and surface estimations) is re-done for the same images down-sampled by 2. We found $\sqrt{E(Z)/600} = 56$ cm, which implies that the SfM drift is larger than in the previous case. 73.9% of sampled pixels are inliers, the error median of inliers is 64.3 cm, the 90% quantile of inliers is 103 cm.

Other quantitative evaluations in different contexts are provided in Sec. 7.4.3.

7.2.10 Video as Input

At last, we test our batch surface reconstruction method by using a video as input. The video sequence contains 24700 frames and the trajectory length is about 1.4 km long. The camera is mounted on an helmet worn by a person who rides a bike. The radii of large and small circles of the catadioptric images are 297 and 59 pixels.

Fig. 7.12 shows a top view of the SfM result. As already mentioned in Sec. 3.1.2, a key-frame selection process is applied in case of using a video as input. Here, 2504 key-frames are selected from 24700 frames such that about 600 Harris points are matched in three consecutive key-frames. We obtain 385k 3d points reconstructed from 2230k Harris (inlier) points.

Fig. 7.13 shows views of the VRML model with the parameters in Sec. 7.2.1. The 416k triangles of the surface are estimated in 166 s, including 18.7 s for the Delaunay Step, 6.9 s for Ray-tracing, 6.7 s for 2-Manifold Generation, 5.9 s for Topology Extension, 120 s for Handle Removal and 7.7 s for Post-Processing. The 3d Delaunay is initialized with 213k selected vertices and it has 1335k tetrahedra (without $\mathbf{v}_\infty$). 51.41% of these tetrahedra are *free-space*, 87.33% of the *free-space* tetrahedra are *outside* by the 2-Manifold Generation, and 91.82% of the *free-space* tetrahedra are *outside* at the end of the process. The most costly step is Handle Removal; 3% of the 1548k Delaunay edges are critical edges.

Every key-frame adds about 85 points in the 3d Delaunay, and the distance between two consecutive key-frames is about 55 cm. This implies a very simplified and compact model of the scene. The thin details such as electric posts, window carriers and trees with low density foliage cannot be modeled. Note that the point density of the still image sequence (Sec. 7.2.1) is higher than the one of the video sequence: every key-frame adds about 213 points in the 3d Delaunay and the distance between two consecutive key-frames is about 30 cm. Both image resolution and number of points contribute to the surface quality.

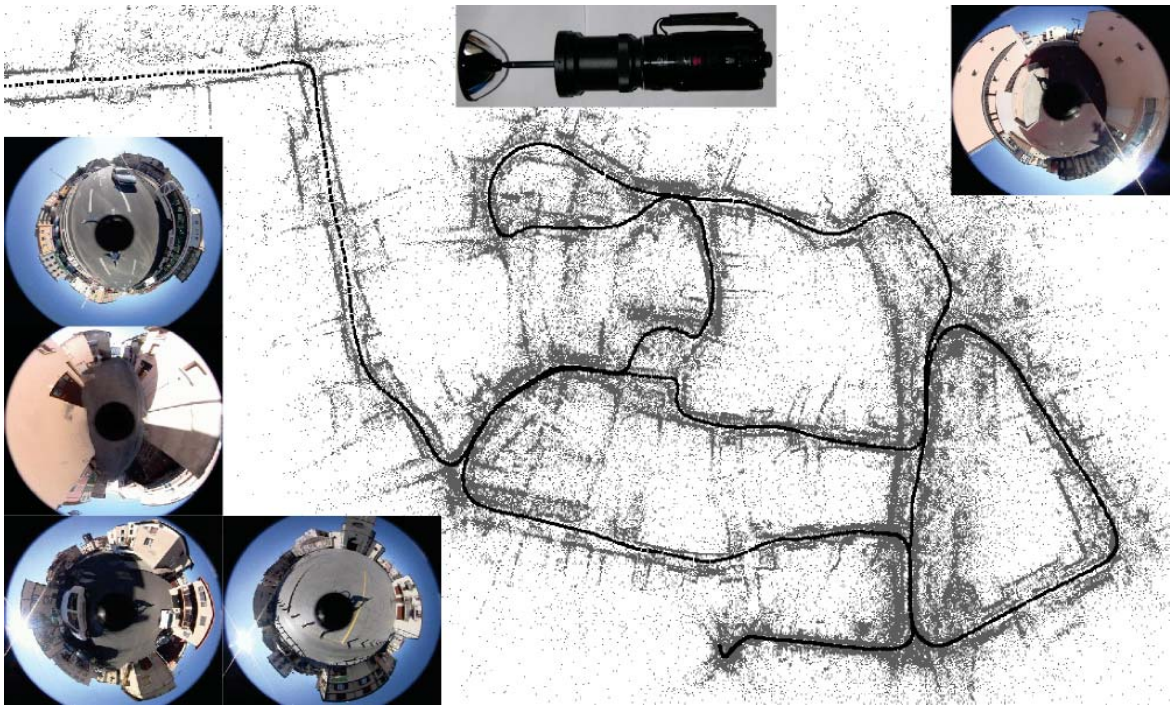


Figure 7.12: Catadioptric camera, images of input video and SfM results.

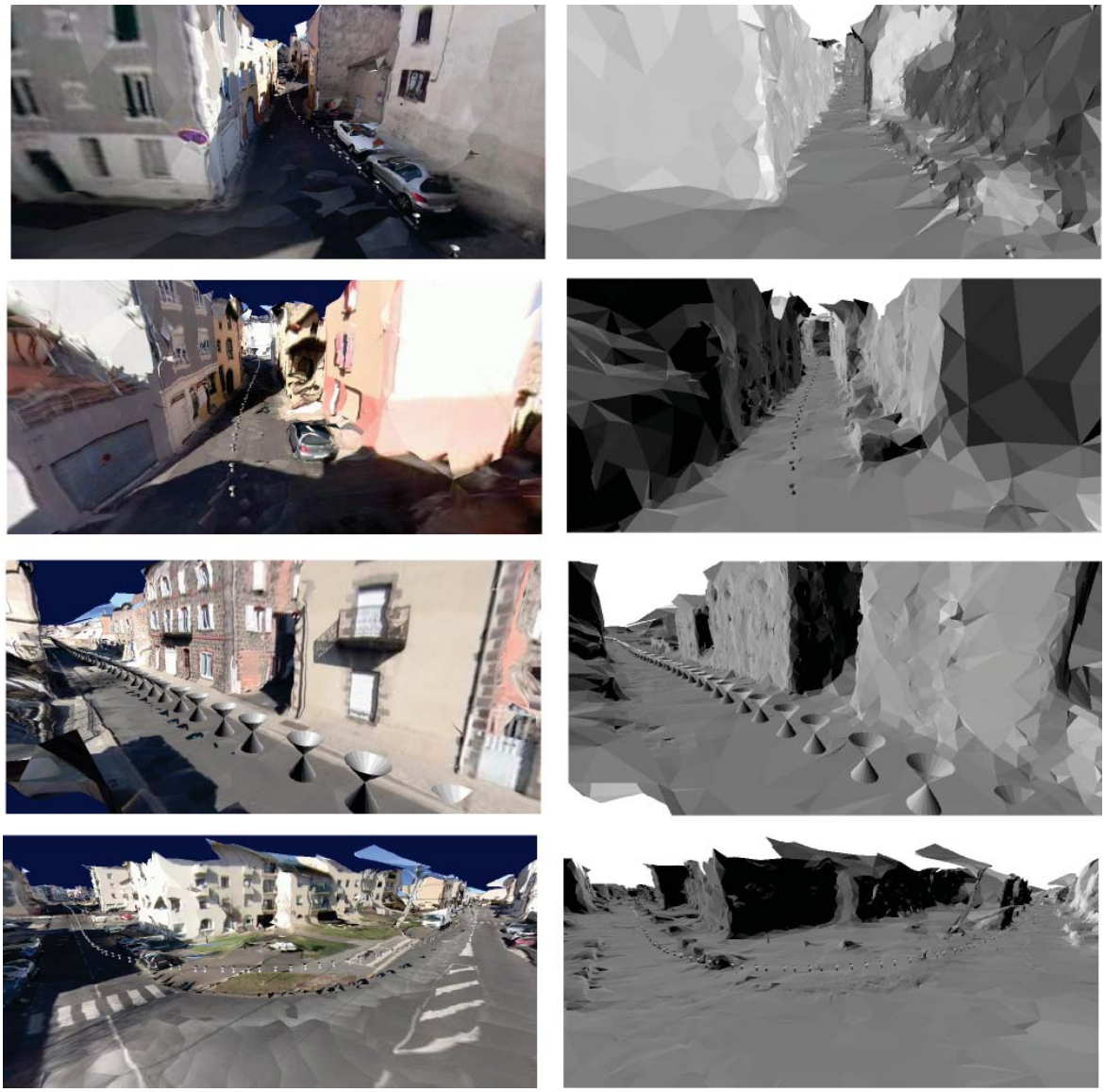


Figure 7.13: Reconstructed models using video as input. Images are taken in viewpoints which are not in the camera trajectory. Camera poses are represented by cones.

7.3 Incremental Environment Modeling

Now we present the experiments of our incremental environment modeling method. Sec. 7.3.1 shows the overview of our experiments results. Sec. 7.3.2 analyzes the experimental complexities. In Sec. 7.3.3 and Sec. 7.3.4, we vary respectively parameters k (in local ray-tracing) and l (in incremental 2-manifold generation) of our method and show the results. In Sec. 7.3.5, we give a quantitative evaluation of our incremental surface reconstruction method and a comparison to evaluation results of our batch method.

7.3.1 Overview of Results

The input is a AVCHD video (MP4) of 25278 frames taken by walking in a city during 505 seconds. Ground truth is not available, but we know that the trajectory length is about 800 m. Fig. 7.14 shows our camera and several images of the sequence. The horizontal and vertical radii of the large ellipse, which contains the scene projection in the images, are 700 and 693 pixels, respectively.

Incremental Geometry Reconstruction (SfM) The incremental SfM method in Sec. 3.3 is firstly applied and 1033 key-frames are progressively selected from 25278 frames. In average, about 600 (inlier) Harris points are matched by correlation in three consecutive key-frames and the ratio between the number of reconstructed 3d points and key-frames is 182 point/key-frame. The decision of inlier/outlier for a 2d point or a ray is based on our angular error threshold, which is equivalent to a reprojection error of 2 pixels. The average RMS error is equivalent to a reprojection error of 0.73 pixels. Finally, 1033 camera poses and 188k 3d points are reconstructed for the complete sequence. Fig. 7.14 shows these reconstructed points and camera poses. Thanks to an aerial photography, we can see the drift our incremental SfM. It is unused by our method.

Incremental Surface Reconstruction The incremental surface reconstruction is done with the following parameter setting: ϵ for point filter (Sec. 5.2) is 10°, the number k of layers used in local ray-tracing (Sec. 5.3) is 40 and the number l of layers contained in a pack in incremental 2-manifold generation (Sec. 5.4) is 60. Besides, threshold b_0 used in the priority queue initialization for a successive region-growing in incremental 2-manifold generation is 10, and threshold b_1 used in the incremental topology extension (Sec. 5.5) is 10.

1031 incremental surfaces are successively reconstructed which correspond to times from 3 to 1033. Fig. 7.15 shows some of them in a top viewpoint without sky removal. And we show also local views of some incremental surfaces in Fig. 7.16, which are extracted from our on-line video[1]. Here, the observer moves in the scene such that he/she is observing the most recent part of the surface at a (roughly) constant distance. At time t, this part is mainly within a ball whose center is located at camera pose $\mathbf{c}^{t-2}$. The observer is located at $\mathbf{c}^{t-20}$ and is looking towards $\mathbf{c}^{t-2}$. The observer and the surface end come forward simultaneously. The reconstructed incremental surfaces are posteriorly textured for better visualization.

[1] http://www.youtube.com/watch?v=4QZFgfMeG4E

In average, to update a reconstructed surface to a new surface, about 118 3d points are selected from 182 SfM points and added in the Delaunay triangulation. The average percentage of *free-space*/all tetrahedra is 44.9% and the one of *outside*/*free-space* tetrahedra is 88.2%. The final Delaunay triangulation contains 121k points selected from 188k SfM points. The final reconstructed surface, showed in Fig. 7.17, contains 233k triangles and 117k vertices. The processing time of each incremental model computation is bounded by 300 ms (see Fig. 7.18), except at the end of the sequence where a small (incomplete) loop exists.

Figure 7.14: Overview of incremental SfM. At the top, we see from left to right: the catadioptric camera, two sequence images and aerial view of trajectory. At the bottom, we see the final reconstructed SfM points and camera poses.

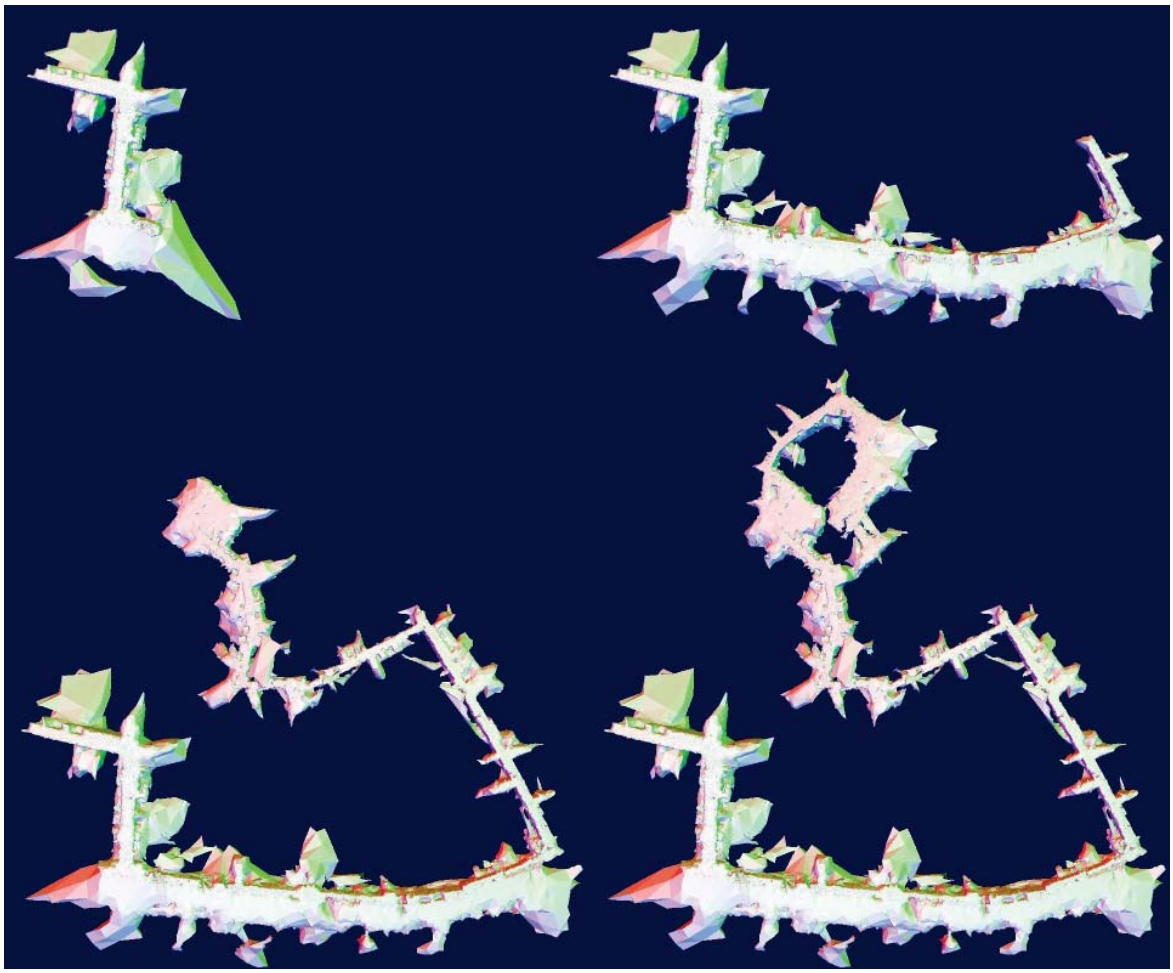


Figure 7.15: Bottom views of incremental surfaces. We show respectively bottom views of the incremental surfaces reconstructed at time 100 (top left), 400 (top right), 800 (bottom left) and 1033 (bottom right).

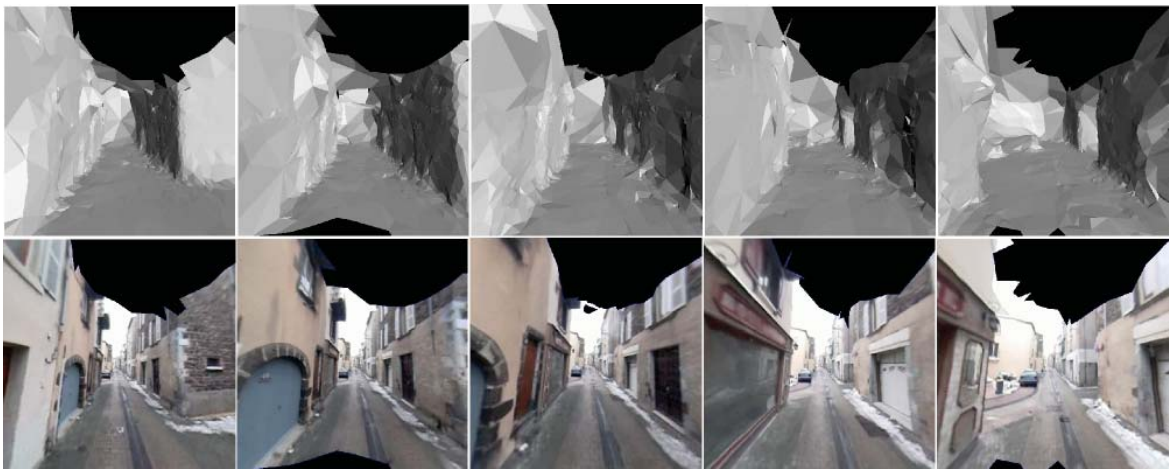


Figure 7.16: Local view of incremental surfaces obtained at different time t. Top: gray levels encode the triangle normals. Bottom: one omnidirectional image is used for texture mapping. The black areas are due to triangles without texture in this image.

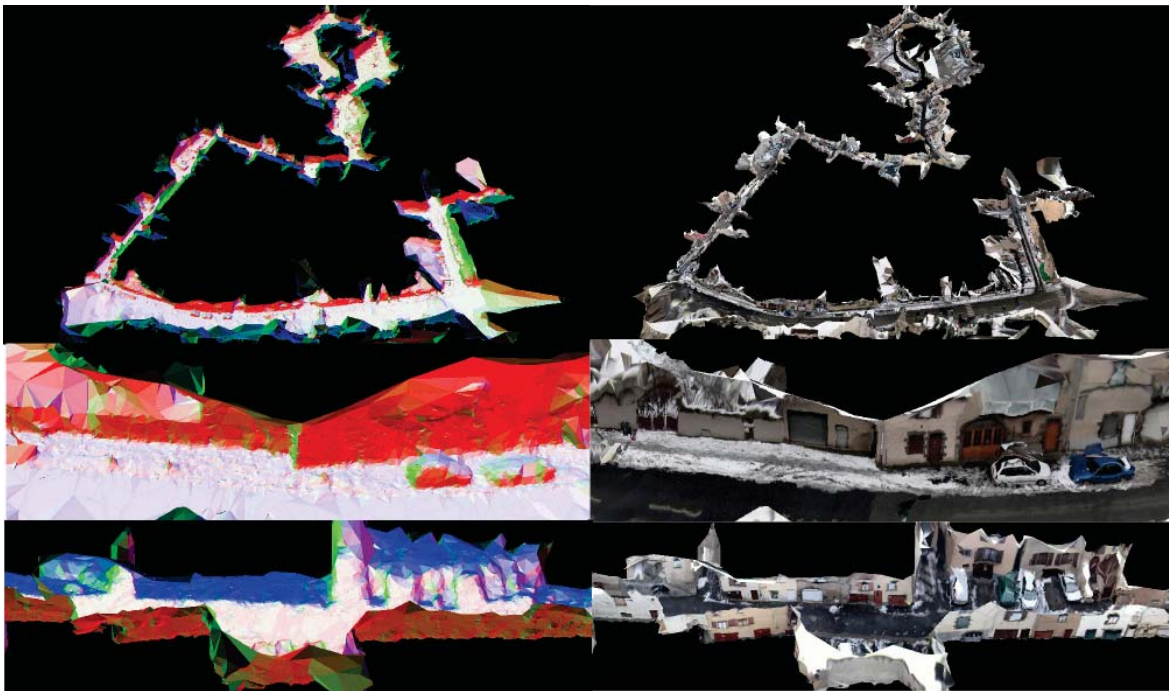


Figure 7.17: Final incremental surface. We show the final incremental surface ($t = 1033$) with "sky" triangles removed. The left ones are models where colors encode triangle normals and the right ones are models with texture.

7.3.2 Processing Time and Experimental Complexities

Now we present and analyze the results of processing time of our method. Fig. 7.18 illustrates the computation times of the different steps at each time t: "Delaunay" in yellow (3d Delaunay Triangulation+Dating), "Ray-tracing" in blue (Ray Tracing), "Manifold" in red (2-Manifold Extraction+Topology Extension), "Post-processing" in green (Surface Denoising) and "Total" in black.

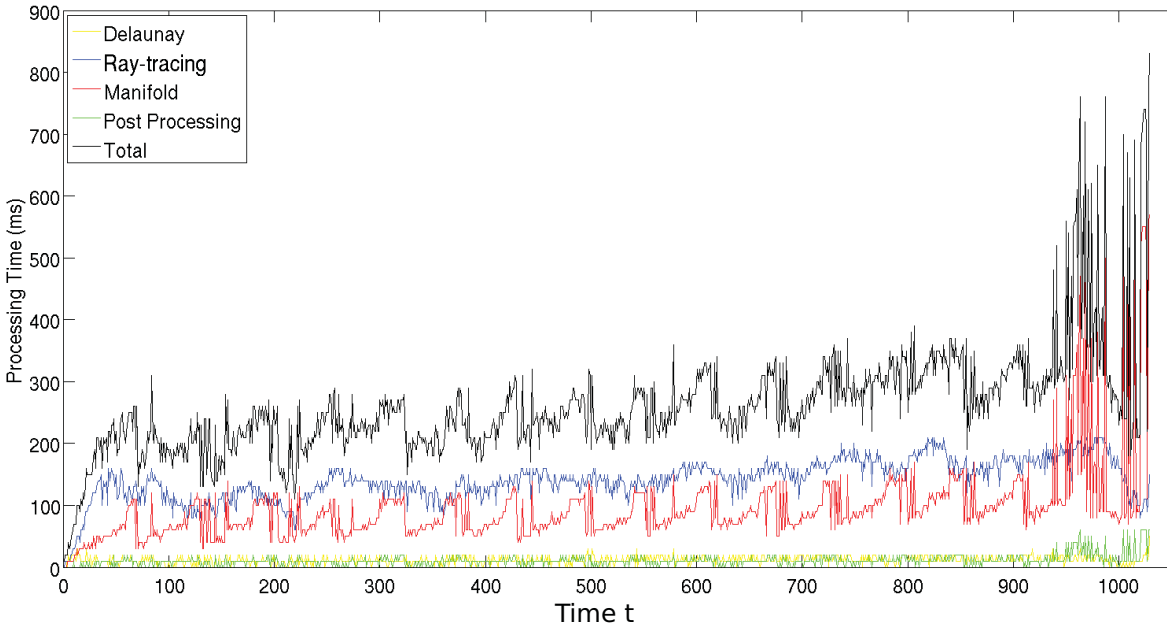


Figure 7.18: Processing time of incremental surface reconstruction as a function of time t.

"Delaunay" and "Post-processing" have almost negligible computation times compared to the other steps. "Ray-tracing" is less than 190 ms. If $t \in [0, 925]$, "Manifold" is less than 200 ms. In the other cases, "Manifold" is between 50 and 600 ms.

Thanks to Fig. 7.19, we see that the computation times of "Manifold" and "Post-Processing" globally increase if $t - d_t$ increases and $t - d_t > 50$. Furthermore, $t - d_t < 280$ in the whole sequence. Remember that d_t is the smallest date of all *outside* tetrahedra destroyed at time t by "Delaunay". These results are consistent with those of the tight theoretical time complexity study (Sec. 6.3.3): Delaunay and Ray-tracing are $\mathcal{O}(1)$, Manifold is $\mathcal{O}((t - d_t)\log(t - d_t))$, Post-Processing is $\mathcal{O}(t - d_t)$.

We note that the processing time of Manifold in Fig. 7.18 has a zig-zag form with perturbations. It can be explained as follows. Assume firstly that $t - d_t$ is constant; $t - d_t$ is about $30 = l/2$ in practice. If $t - i_t l > 30$, then $i_0 = i_t$ and we have one region-growing from $O_{i_t l}$ to O_t. Otherwise, we have $i_0 = i_t - 1$ and there are two region-growings: from $O_{(i_t-1)l}$ to $O_{i_t l}$, then from $O_{i_t l}$ to O_t. In both case, the number of tetrahedra in O_t increases with t (these tetrahedra are dated $i_t l + 1, \cdots, t - 1, t$). This implies the sawtooth form of Manifold processing time. This form is disturbed by variations of $t - d_t$ around its mean, especially

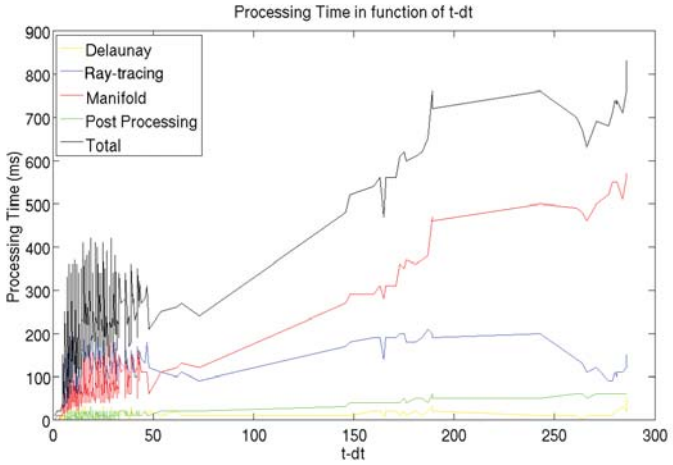


Figure 7.19: Processing time of incremental surface reconstruction as a function of $t - d_t$.

when the $t - d_t$ variations change the number of region-growings from 1 to 2 or from 2 to 1.

We now explain the large values of "Manifold" if $t \in [925, 1032]$. In a complete trajectory loop, vertices added at the loop end (at time t) destroy *outside* tetrahedra created at the loop beginning (at time d_t) since these vertices and tetrahedra have similar 3d locations. The larger the loop, the larger $t - d_t$, and the larger the computation time of "Manifold" (and "Post-Processing"). Fig. 7.14 shows that the reconstructed trajectory has two incomplete (about 75%) loops: a large one on the top and a small one on the bottom. Here both loops are incomplete but the principle presented above still applies for the small loop which is 75% closed if $t \in [925, 1032]$: there are times in $[925, 1032]$ such that added vertices destroy *outside* tetrahedra created at the loop beginning. However, it does not apply in case of the large loop. It is because that the added vertices and *outside* tetrahedra are in a tubular neighborhood of the camera trajectory, and the neighborhood radius is small enough compared to the distance between the beginning and the end of the loop. Fig. 7.17 shows the neighborhood and its size and we see the small loop on the top and the large loop on the bottom of the figure.

7.3.3 Varying Parameter k in Local Ray-tracing

We vary the parameter k of the local ray-tracing step (Sec. 5.3) in $\{ 10, 40, 80, 2000\}$ and compare results of the incremental surface reconstruction. Here, $k = 2000$ means that the ray-tracing is applied to all rays up to t since t is in $\{3, 4, ..., 1033\}$. Thus the local ray-tracing at t using $k = 2000$ is same to the ray-tracing step of batch surface reconstruction using all available rays at t.

According to Fig. 7.20, the processing time of ray-tracing depends on k. Using all rays, i.e. $k = 2000$, has a complexity of $\mathcal{O}(t)$ and using $k = 10, 40$ or 80 has a complexity of $\mathcal{O}(1)$. Besides, according to statistic results showed in Tab. 7.3, if $k = 10$, the average ratio of *free-space/all* tetrahedra of the incremental surface reconstruction is 40.81%. It is smaller than 44.95% which is the one by using all rays. Now if $k = 40$, the average *free-space/all*

k	*free-space*/all	*outside/free-space*	processing time
10	40.81%	86.16%	37 ms
40	44.87%	88.12%	105 ms
80	44.95%	87.94%	209 ms
2000	44.95%	87.97%	1279 ms

Table 7.3: Incremental surface results using different k. k is in $\{10, 40, 80, 2000\}$. We show from left to right: value of k, the average ratio of *free-space*/all tetrahedra, the average ratio of *outside/free-space* tetrahedra and the average processing time of local ray-tracing.

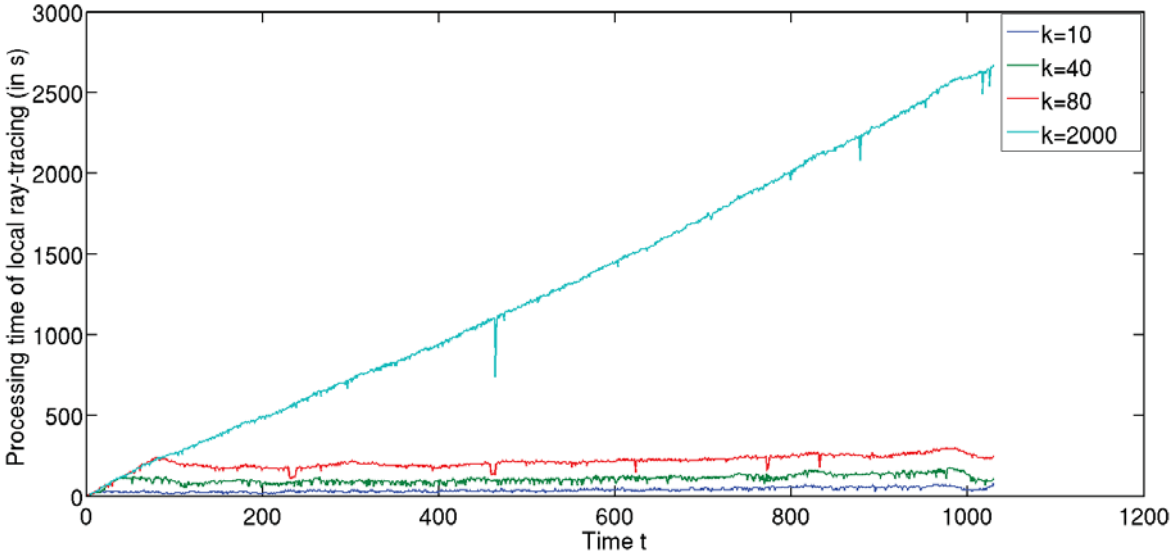


Figure 7.20: Processing time of local ray-tracing as a function of time t.

ratio is 44.87% which is close to 44.95%. As the number of all tetrahedra of the Delaunay triangulation does not change when k is varied, so the *free-space/all* ratio indicates the completeness of the local ray-tracing, which is the best if all rays are used. By using a suitable k, here 40 in our case, the local ray-tracing results are almost the same to those of ray-tracing using all rays. It proves the remark made in Sec. 5.3.

7.3.4 Varying Parameter l in Incremental 2-Manifold Generation

In the incremental 2-manifold generation step 5.4, the region-growing is applied by pack of l layers. We now vary l in $\{20, 60, 100, 2000\}$ to see its impact on the results of incremental surface reconstruction. Here, $l = 2000$ means that the incremental 2-manifold generation at time t contains only one region-growing which grows from $\emptyset$. Thus results of the incremental 2-manifold generation using $k = 2000$ at t are close to results of batch 2-manifold generation (Sec. 4.4) using all points available at t.

Tab. 7.4 illustrates the results of "Manifold" using different l. Here, "Manifold" includes

l	*outside/free-space*	|*outside* tetrahedra|	processing time
1	66.61%	225k	187 ms
20	86.95%	295k	77 ms
60	88.12%	298k	98 ms
100	88.03%	298k	125 ms
2000	88.58%	299k	621 ms

Table 7.4: Incremental surface results using different l. l is varied in $\{20, 60, 100, 2000\}$. We show from left to right: value of l, the average ratio of *outside/free-space* tetrahedra, the number of *outside* tetrahedra in the final Delaunay triangulation and the average processing time of "Manifold".

steps of 2-manifold generation and topology extension. Remember that the *outside/free-space* ratio is an indicator of the performance of our surface reconstruction. If $l = 1$, i.e. the region-growing is done layer by layer, we see that the *outside/free-space* ratio is lower and the processing time is higher compared to region-growing results using other values of l. In fact, the incremental region-growing have both dating and manifold constraints: $O_{i_0 l} \subseteq O_{i_1 l}, \cdots, O_{i_t l} \subseteq O_t$ and their borders $\delta O_{i_0 l}, \cdots, \delta O_t$ are always 2-manifold. The impact of dating constraints becomes important if the number of *outside* regions is large, i.e. if l is small. We found that if l is sufficiently large, e.g. 60, the *outside/free-space* ratio is close to the one of region-growing using $k = 2000$. In addition, from $l = 20$, the average processing time of "Manifold" is higher if l is larger. Finally, $l = 60$ is a good compromise between the *outside/free-space* ratio and the "Manifold" processing time in our experiments.

7.3.5 Quantitative Evaluation and Comparison to Batch Surface Reconstruction

In this section, the quantitative evaluation of our incremental surface reconstruction method is performed by using synthetic data, and the evaluation results are compared to those of the batch surface reconstruction. The same evaluation process and synthetic image sequence are used as in Sec. 7.2.9. Here, to reduce the impact of the geometry reconstruction, we do not use the incremental SfM presented in Sec. 3.3. Instead, we progressively provide the SfM points and camera poses which are reconstructed by batch SfM from synthetic images as follows. The camera pose $\mathbf{c}^i$ obtained by batch SfM is available for the incremental surface reconstruction at time i. For a 3d point $\mathbf{q}_j$ obtained by batch SfM, if its last 2d track is in the image of camera pose $\mathbf{c}^i$, then $\mathbf{q}_j$ will be available for the incremental surface reconstruction at time $i + 2$. The incremental surface reconstruction is performed using parameters in Sec. 7.3.1 and finally 598 surfaces are successively reconstructed. We use the final incremental surface ($t = 600$) for the quantitative evaluation and comparison.

Qualitative comparison The results of the batch and the incremental surface reconstruction methods are compared. The input SfM points are the same for both methods. The batch method uses the second handle removal method (R2) while the incremental method uses the

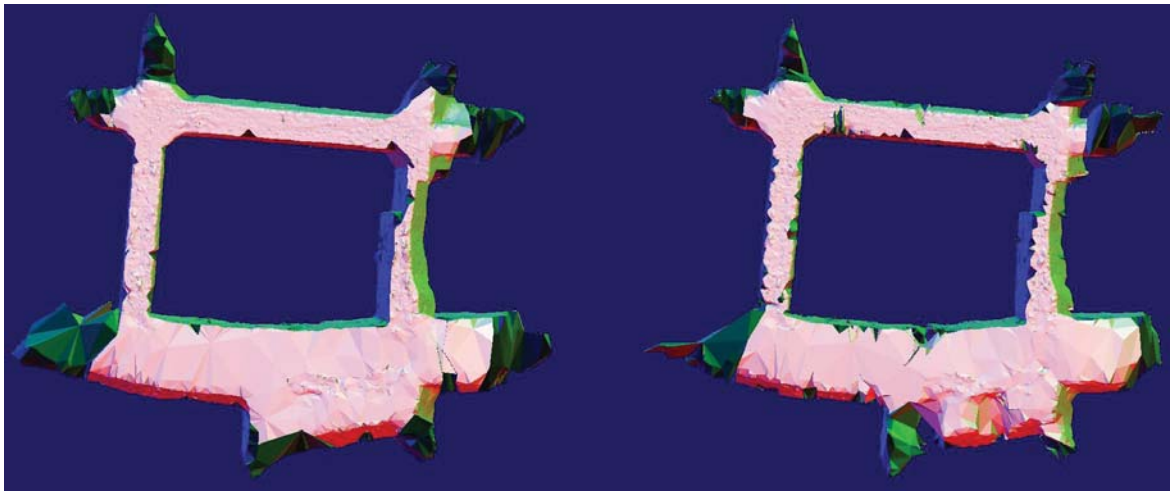


Figure 7.21: Surfaces reconstructed by batch and incremental surface reconstruction methods using synthetic data . We show respectively surfaces reconstructed by batch and incremental methods on left and right. Colors encode triangle normals of surfaces. "sky" triangles are removed using the sky triangle removal method in Sec. 4.7.3.

first one (R1)—see Sec. 4.6 for more details about R1 and R2. Thus the final triangulations of both methods contain slightly different number of points. We have 129k vertices and 782k tetrahedra in the triangulation of the batch method, and 123k vertices and 750k tetrahedra in the final Delaunay triangulation of the incremental method. The resulting surfaces of batch and incremental methods have respectively 241k and 232k triangles. Here, "sky" triangles of both surfaces are also included. We have a slightly better *free-space/outside* ratio in the batch case than in the incremental case. The former is 90.96% while the latter is 85.8%. This result can be explained as follows: the incremental growing is more constrained than the batch growing. In the incremental case, both dating and manifold constraints are used however the batch method only uses manifold constraints. We have already seen in Sec. 7.3.4 that the dating constraints become important if l is small and the reconstruction results become worse. Additionally, as shown in Sec. 7.2.6, R2 used by the batch method is more efficient than R1 used in the incremental method and improves the *free-space/outside* ratio.

Quantitative comparison Errors between the final incremental surface and the ground truth surface are evaluated using the same method described in Sec. 7.2.9. Note that the transformation matrix Z which registers the incremental surface to the ground truth surface is the same to the one in the batch case since the camera poses used in both cases are the same. The same experiment (both SfM and surface calculations) is re-done for the same synthetic images down-sampled by 2. In this case, we found $\sqrt{E(Z)/600} = 56$ cm, which implies that the SfM drift is larger than in the previous case where $\sqrt{E(Z)/600} = 5.1$ cm.

According to Tab. 7.5, we see that the batch method has better results than the incremental method. It is because that the batch method uses the second spurious handle removal

method	inliers (%)	median (cm)	90% quantile (cm)
batch	77.5 (74.1)	7.7 (64.0)	30.7 (101)
increm.	72.4 (69.7)	8.6 (71.0)	49.7 (106)

Table 7.5: Errors of batch and final incremental surfaces using ground truth surface. The numbers between parentheses are obtained for down-sampled images.

method, the peak removal post processing and does not have dating constraints. We also note that errors of the reconstructed surfaces depend mainly on the SfM drift. It is shown in Tab. 7.5 that the magnitude order of the median error is roughly the same to the average SfM drift.

Fig. 7.22 gives more details on the distribution of errors. We see that the batch surface has slightly more errors distributed in low error interval than the incremental surface. In addition, the majority of errors of reconstructed surfaces using the original image sequence are in interval (0 m, 0.1 m). When the down-sampled image sequence is used, the image resolution is smaller thus the SfM drift is larger. It implies that the surface accuracies are degraded where most errors are in interval (0.5 m, 1 m).

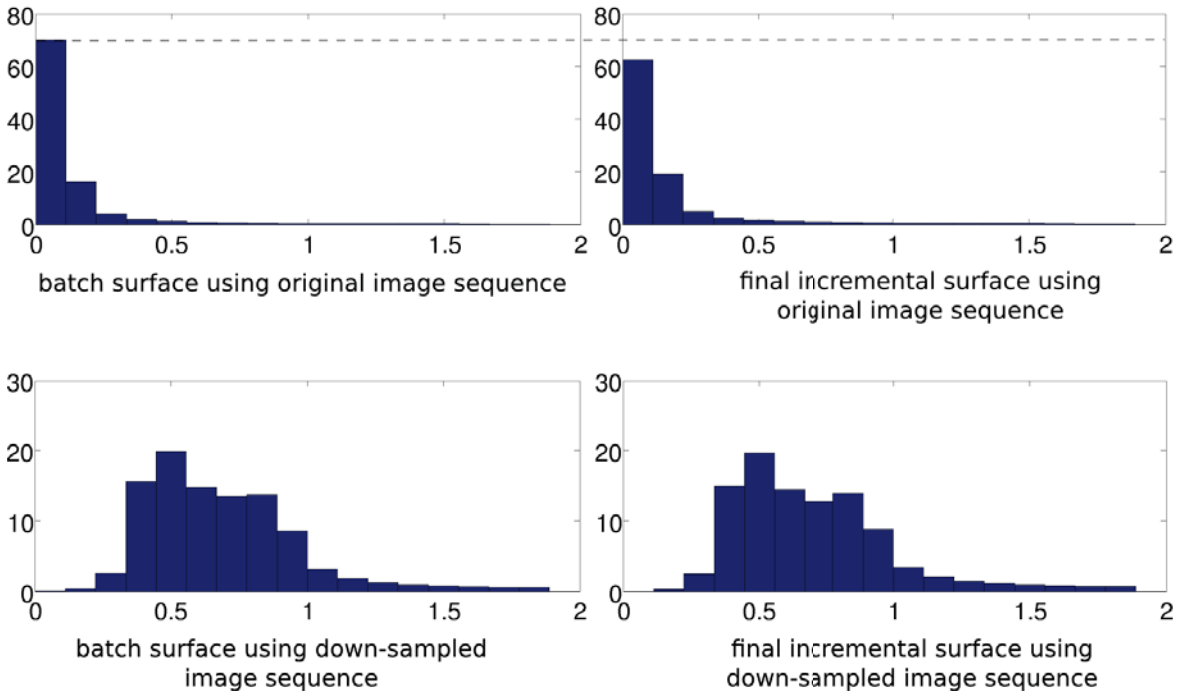


Figure 7.22: Error distributions of reconstructed surfaces in histograms. For each sub figure, the horizontal axis is the error axis (in meter) containing 20 uniform intervals, and the vertical axis is the error distribution axis which indicates the percentage of inlier samples in an interval in all inlier samples.

7.4 Other Examples

7.4.1 Laschamps Church

This sequence is a still JPEG image sequence of 208 images and the camera trajectory is a complete loop around a church at Laschamps in France. It is also used in our first publication [3] where our batch environment modeling method (except the second method of handle removal) is presented. A video showing the reconstructed surface is on line[1]. Most of the parameters used here are same to those in Sec. 7.2.1, except that ϵ used in point filter is 5° instead of 10° and R1 is used instead of R2 for the spurious handle removal. The incremental environment modeling method is also applied to this image sequence by using most of the parameters in Sec. 7.3.1 except that ϵ is set 5°.

Fig. 7.23 illustrates results of batch and incremental SfM. We see that the SfM drift is removed in the batch SfM, thanks to the loop closure. Fig. 7.24 and Fig. 7.25 show respectively surfaces reconstructed by the batch and incremental surface reconstruction methods.

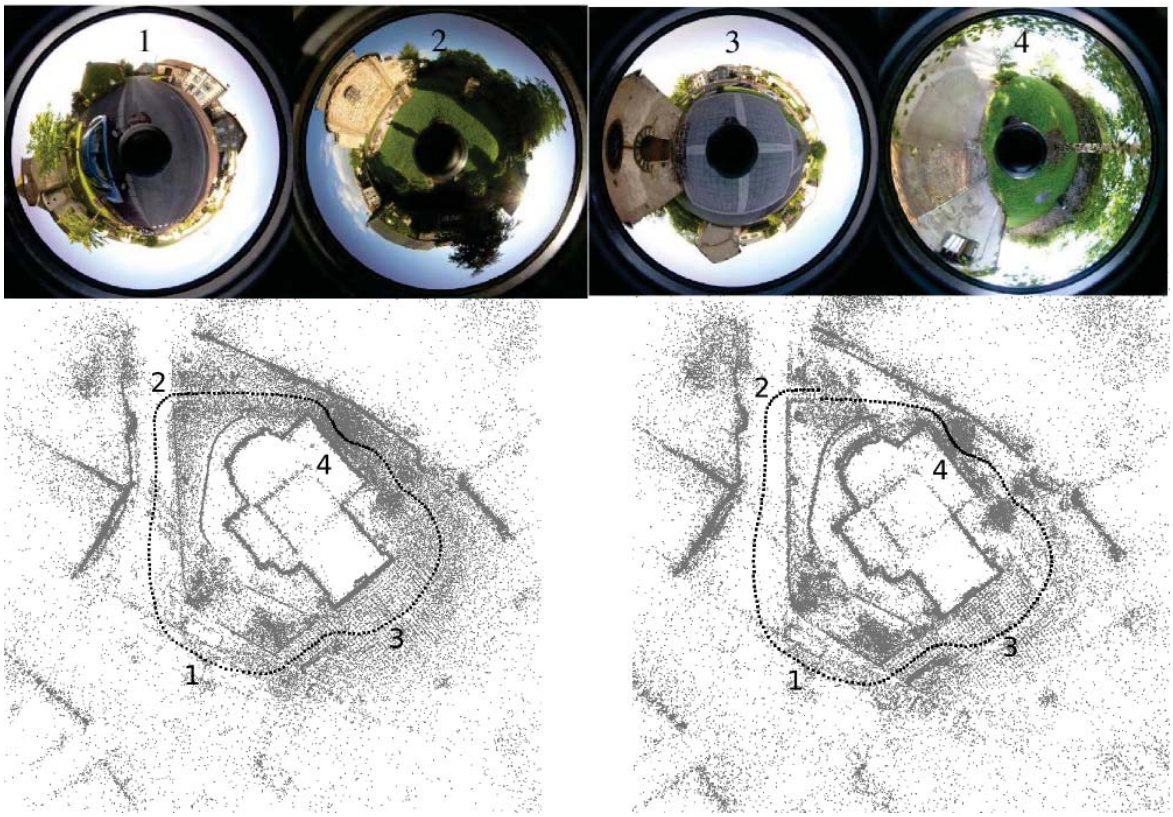


Figure 7.23: Top view of 3d points and camera poses reconstructed by batch and incremental SfM using Laschamps Church image sequence. Results of batch SfM are on the left and those of incremental SfM are on the right. Points are in gray and camera poses are in black.

[1] http://www.youtube.com/watch?v=mNcbWhvNftk

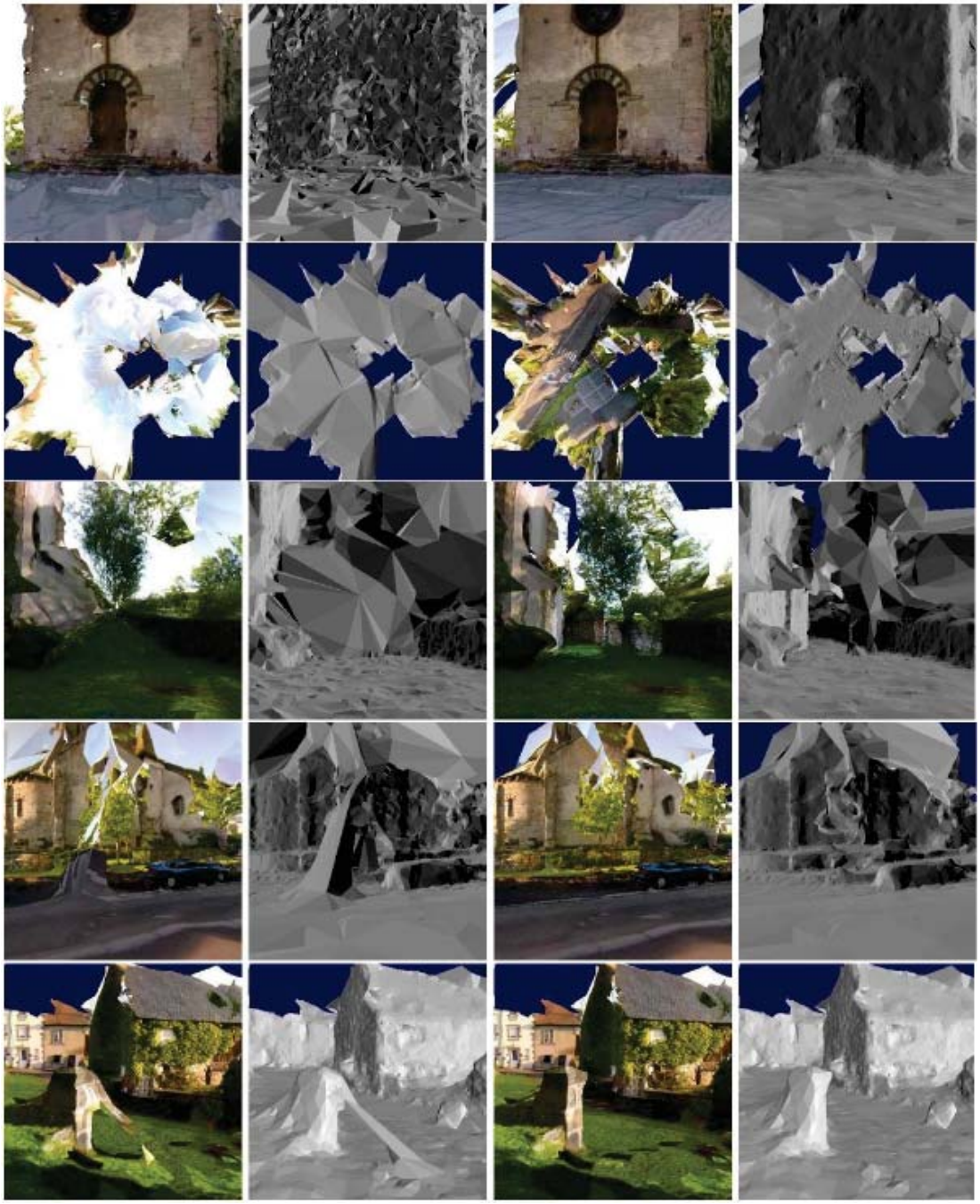


Figure 7.24: Results of batch surface reconstruction for Laschamps Church image sequence. We show effects of some steps of our method. The two left columns show results without using these steps and the right two columns show results using complete steps of our method. Row 1 shows effects of two post-processings: peak removal and surface denoising. Row 2 shows effects of the post-processing: sky removal (top views). Row 3 and 4 show effects of the "Topology Extension". Row 5 shows effects of the first method of spurious handle removal.

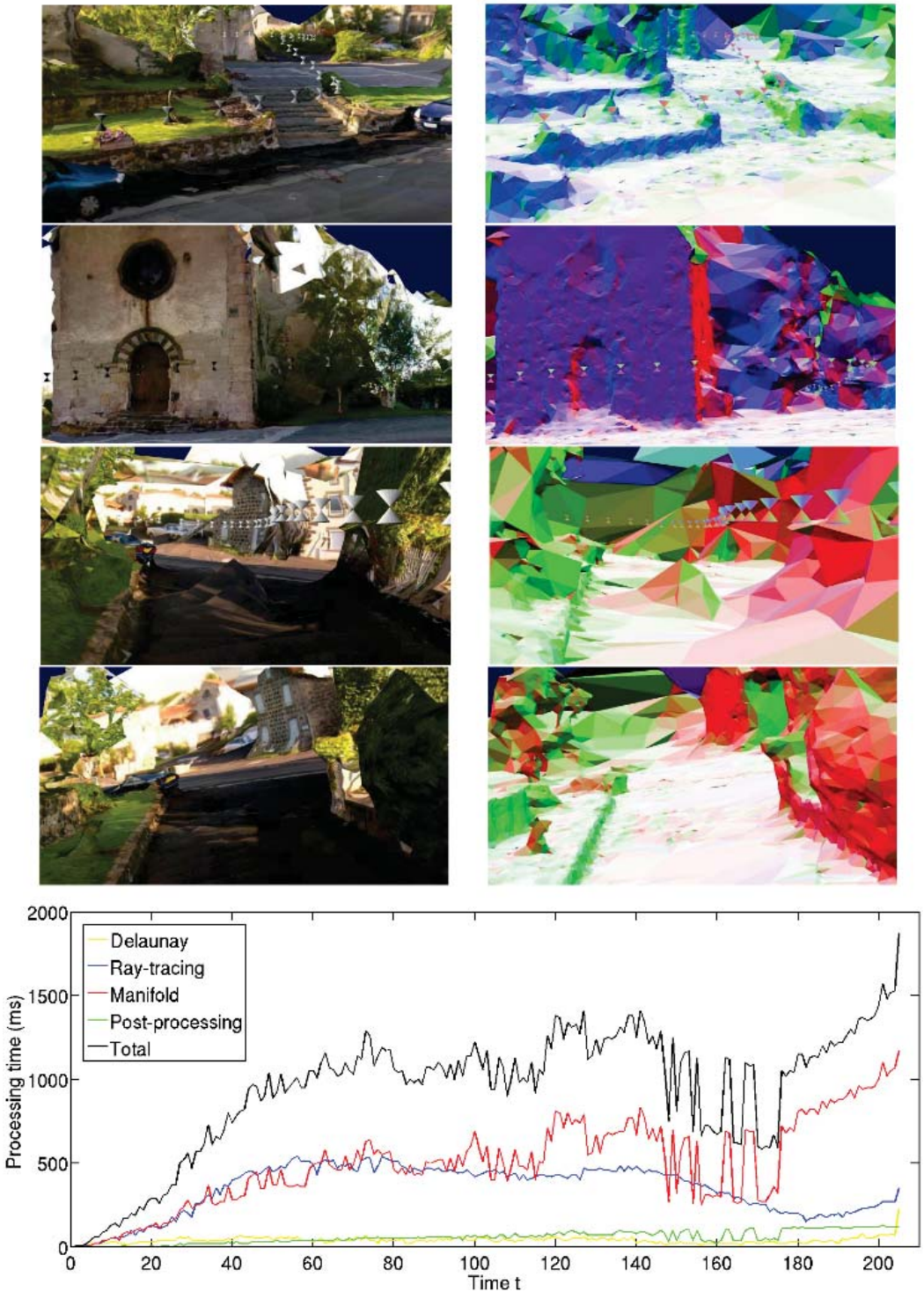


Figure 7.25: Results of incremental surface reconstruction for the Laschamps image sequence. Row 1 and 2 show local views of the final incremental surface, Row 3 and Row 4 show respectively local views at the end of the camera trajectory loop of the final incremental surface and the batch surface. The left column shows views of textured models and the right column shows views of models colored in function of triangle normals. The processing time in function of time t is shown in the bottom. Here, about 300 points are added in the Delaunay at each time.

7.4.2 Clermont-Ferrand Downtown

The sequence is a still JPEG image sequence of 354 images taken in the downtown of Clermont-Ferrand. It is also used by our incremental environment modeling method in our publication [6], by using parameters in Sec. 7.3.1. Here, we reuse the experiment results of [6]. Besides, the batch environment modeling method is also applied to this image sequence. Most of the parameters in Sec. 7.2.1 are used except that R1 is used instead of R2 for spurious handle removal.

Fig. 7.26 presents results of batch and incremental SfM. Fig. 7.27 and Fig. 7.28 show respectively final surfaces reconstructed by batch and incremental surface reconstruction.

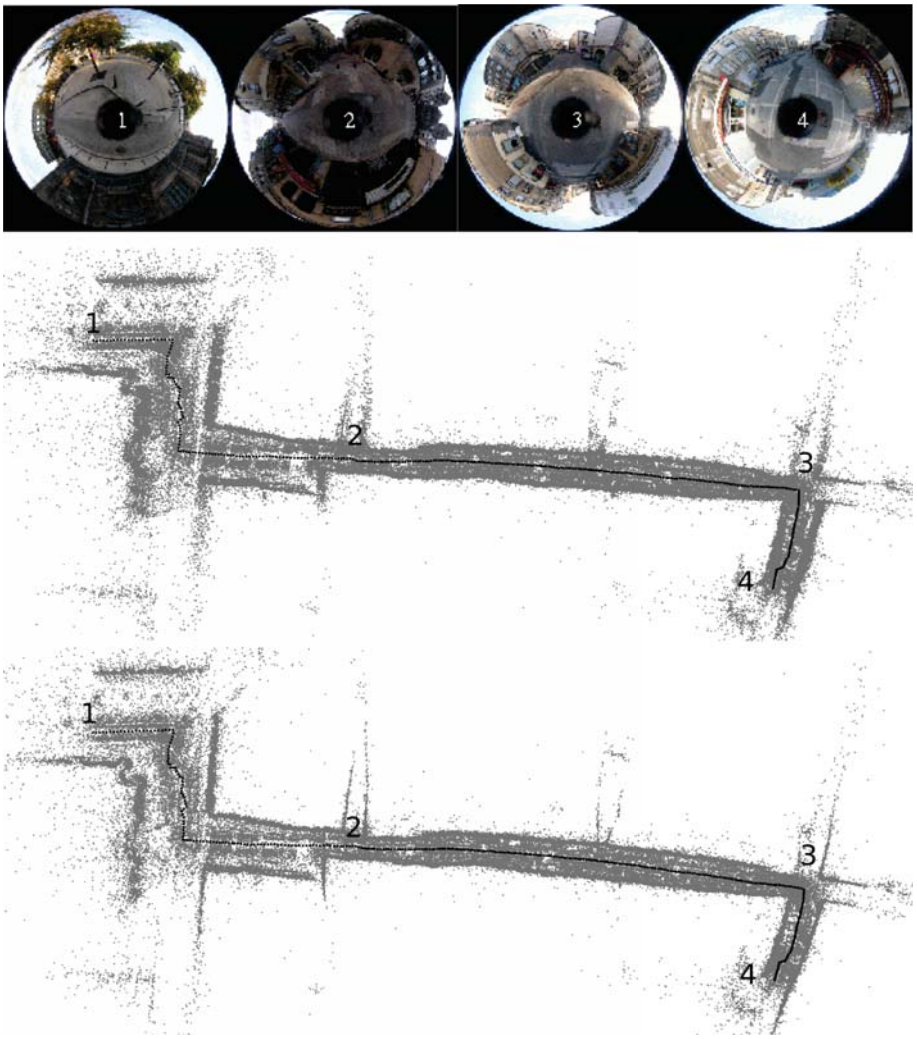


Figure 7.26: Top view of 3d points and camera poses reconstructed by batch and incremental SfM using Clermont-Ferrand Downtown image sequence. We see respectively four images of the sequence on the top, top view of the batch SfM in the middle and top view of the incremental SfM on the bottom.

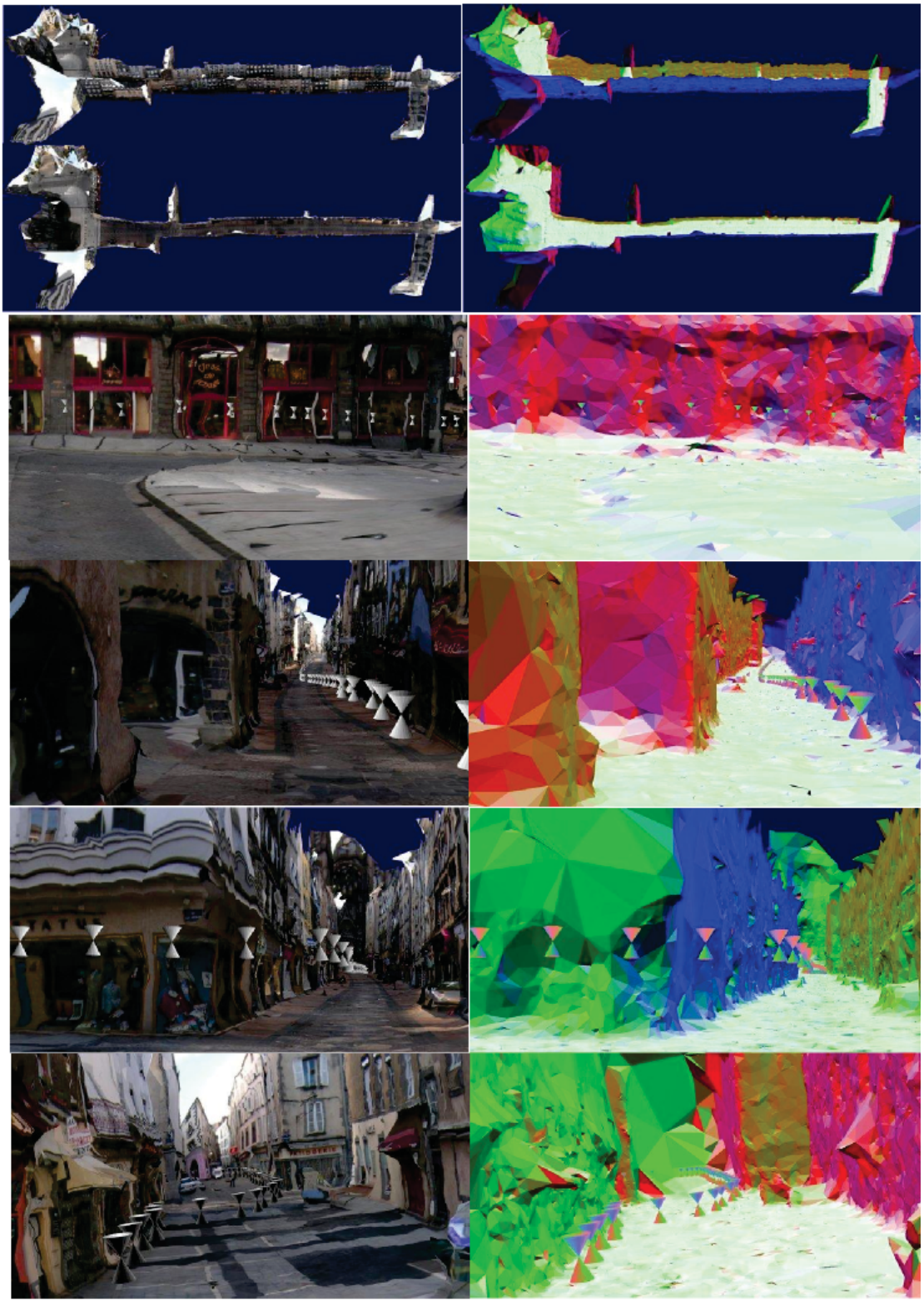


Figure 7.27: Batch surface for Clermont-Ferrand Downtown sequence. Row 1,2 show two bird views of the surface. Row 3-6 show local views of the surface at position 1-4 of Fig. 7.26. The left column shows views of the textured model and the right column shows views of the model colored in triangle normals.

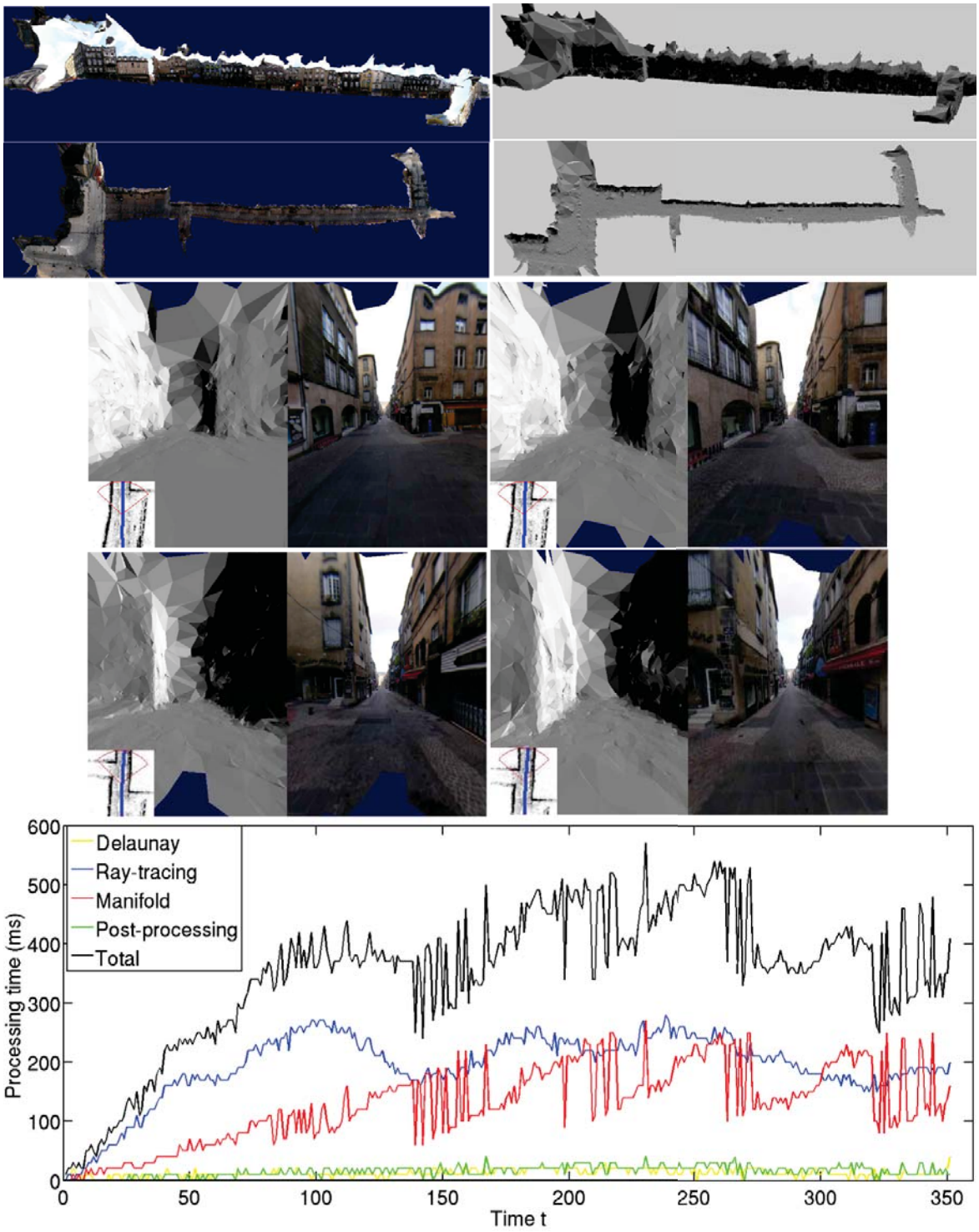


Figure 7.28: Incremental surfaces of Clermont-Ferrand Downtown sequence. Row 1,2 show two bird views of the final surface. Row 3,4 show local views of the surface at four times. Row 5 shows the processing time in function of time. Here, about 200 points are added in the Delaunay each time.

7.4.3 Standard Multi-view Stereo Data Sets (Perspective Camera)

Our batch surface reconstruction method is also applied to 3d points reconstructed from three standard multi-view stereo data sets with available ground truths: "Temple" [124], "Fountain-P11" and "Herzjesu-P8" [133]. The reconstructed surfaces are further quantitatively evaluated. The surface reconstruction is done by using most of the parameters in Sec. 7.2.1 except that R1 is used for the spurious handle removal. Here, we give only a summary of the experiments and details can be found in [2].

Camera poses are given by the input data sets and are used to reconstruct interest points detected and matched in input images. Curves in input images can also be reconstructed and points sampling curves are integrated in the reconstructed point cloud [2]. Finally our batch surface reconstruction method is applied. Note that our environment modeling methods assume that the camera is omni-directional, while images of these data sets are taken by the perspective camera with small field of view. Thus we slightly modify our methods to be able to deal with these data sets (Sec. 4.3). Statistical results of the batch surface reconstruction for all three data sets are given in Tab. 7.6. Note that we do not show results using curves for Fountain-P11 and Herzjesu-P8 because adding curves provides a very minor improvement.

	Temple	Fountain-P11	Herzjesu-P8
number of images	312	11	8
image resolution	640×480	3072×2048	3072×2048
number of 3d points (curves)	32k (56k)	67k	29k
number of surface triangles (curves)	49k (82k)	121k	52k
time: 3d points (+curves) reconstruction	13 s (+18 s)	100 s	64 s
time: surface reconstruction	18 s (+10 s)	23 s	10 s

Table 7.6: Statistical results of batch surface reconstruction for "Temple", "Fountain-P11" and "Herzjesu-P8".

Temple The "Temple" data set contains 312 separate views of a plaster replica. The tight bounding box of the "Temple" model is about $0.1 \times 0.16 \times 0.07$ m^3. Two surfaces are computed and sent to data set creators[1] for evaluation. The first one, named the point-only surface, uses only points. The second one, named the point-curve surface, uses both points and curves. Evaluation results show that the point-curve surface is slightly better than the point-only surface. The point-only surface computation takes 31 s and provides an accuracy of 0.66 mm (for 90% of reconstructed points) and a completeness of 93% (for an error less than 1.25 mm). The point-curve surface computation takes 59 s, and provides an accuracy of 0.59 mm and a completeness of 95.4%. Note that spurious handles exist on both surfaces (R1 is used for spurious handle removal). They can be removed by using R2 with a small value of α. Fig. 7.29 shows the reconstructed surfaces.

[1] http://vision.middlebury.edu/mview/eval/

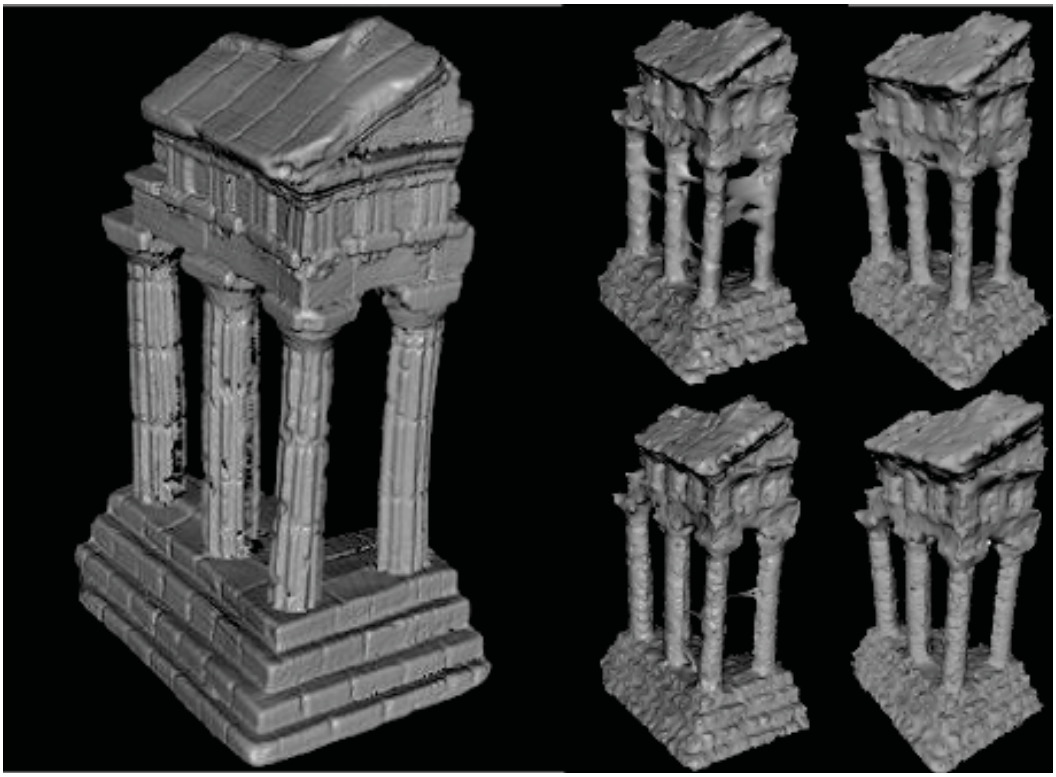


Figure 7.29: Ground truth and reconstructed surfaces of "Temple". Left: ground truth. Middle: reconstructed surfaces using R1. Right: reconstructed surfaces using R2. For Middle and Right, points-only surfaces are at the top and points-edges surfaces are at the bottom.

"Fountain-P11" and "Herzjesu-P8" The "Fountain-P11" and "Herzjesu-P8" data sets contain respectively 11 and 8 high-resolution images. Our reconstructed surfaces are quantitatively evaluated against ground truths provided by data set creators. The evaluation consists in measuring errors between each vertex of our surface and the nearest point in the ground truth surface. We find that 80% of reconstructed surface vertices have an error smaller than 6 cm for "Fountain-P11" and smaller than 7 cm for "Herzjesu-P8" respectively. The computations of "Fountain-P11" and "Herzjesu-P8" take 123 s and 74 s respectively. Surfaces are reconstructed using R1 and spurious handles exist. Most of them can be removed by using R2 using a small value of α. Fig. 7.30 shows the reconstructed surfaces.

Comparison to dense multi-view stereo methods According to the comparison of our reconstructed surfaces with the ground truths, the precision achieved by our sparse environment modeling method cannot yet compete with the dense multi-view stereo methods, but we think that it is sufficient to be used as initialization of dense stereo. Furthermore, given the achieved speed, the algorithm is indeed a good choice not only for this purpose but also for the applications where the speed is at least as important as the precision.

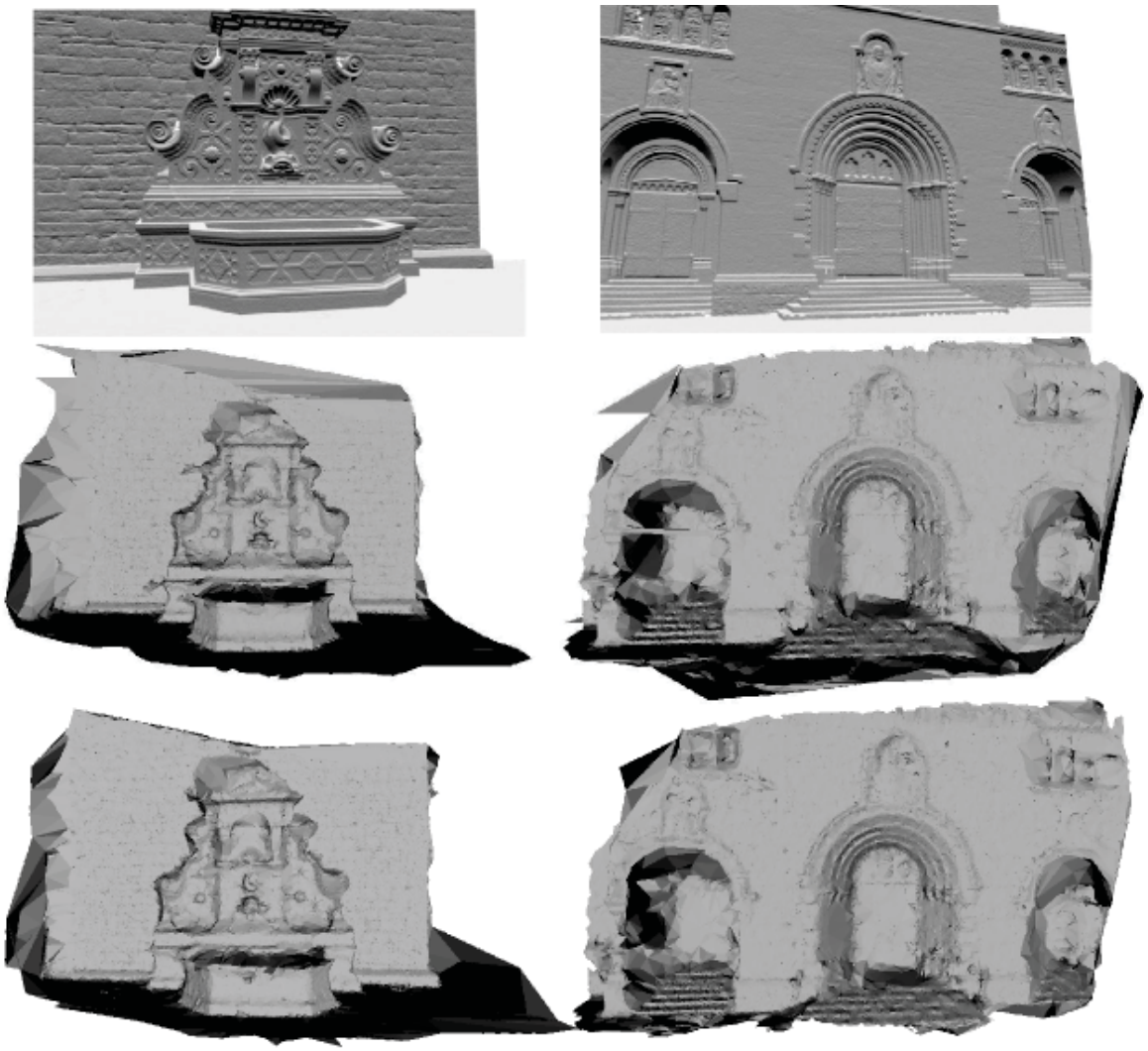


Figure 7.30: Ground truths and reconstructed surfaces of "Fountain-P11" and "Herzjesu-P8". We show "Fountain-P11" on the left and "Herzjesu-P8" on the right. We show from top to bottom: one image of the ground truth, reconstructed surfaces using R1 and reconstructed surfaces using R2.

7.5 Conclusion

Experiments of our batch and incremental environment modeling methods are presented in details in this chapter. A low-cost catadioptric omni-directional camera is used to take images of the environment. Then with the help of a standard PC, we apply our environment modeling methods: a batch method which consists of a geometry reconstruction step (batch SfM) and a subsequent surface reconstruction step, and an incremental method which alternates an incremental geometry reconstruction step (incremental SfM) and an incremental surface reconstruction step. In experiments of the batch method, we have shown the interest of the manifold constraint, results by varying different parameters or steps of the method, results of the experimental time complexity, a comparison to the Poisson surface reconstruction method, a quantitative evaluation, and results using both image and video sequences. In experiments of the incremental method, we have shown results by varying different parameters of the method, the experimental complexities, a quantitative evaluation and comparison of our incremental surface reconstruction method to our batch method. At last, other examples which have also been used in our publications are also provided, including an image sequence of a complete-loop camera trajectory, a video sequence of a canyon-like environment and standard multi-view stereo data sets with ground truths. Besides, our batch surface reconstruction is also used in the work [2]. In this work, SfM points are reconstructed from images taken by a Ladybug omni-directional camera and surface models of both indoor and outdoor environments are reconstructed.

Through these experiments, we see that our environment modeling methods are able to efficiently estimate 2-manifold surface models of complete environments using sparse SfM points in a short time. The extended incremental method can incrementally reconstruct 2-manifold surfaces of environments and the processing time of each incremental reconstruction does not increase with time in case that no loop exists in the camera trajectory. Despite the achievements above, we can see some defects on results of the experiments: incomplete reconstruction of thin objects, high time consumption for the second method of spurious handle removal, high processing time for the incremental surface reconstruction at the end of loops. These drawbacks need to be removed in future works.

Chapter 8

Conclusion

Summary

This thesis deals with the problem of image-based automatic 3d modeling by reconstructing surfaces using results of Structure-from-Motion.

A batch environment modeling method is proposed. The method reconstructs a 2-manifold surface from the SfM output: a sparse cloud of reconstructed Harris points and their visibilities in the images. The potential adjacencies between the points are encoded in a 3d Delaunay triangulation, then the point visibilities are used to label the tetrahedra. A multi-genus 2-manifold is extracted thanks to a greedy region-growing method and a topology extension process. At last, a spurious handle removal process and several post-processings including surface denoising, peak removal, "sky" triangles removal and texturing, are applied to improve the surface quality and visualization. Compared to other environment modeling methods, our method combines both manifold and visibility constraints and the reconstructed surface is a multi-genus 2-manifold surface using sparse SfM points only. The theoretical time complexity analysis of the method is given and various experiments are presented. Our experiments show: the interest of manifold constraint which is neglected by most of the previous Computer Vision works based on sparse SfM points only, results by varying different parameters or steps of the method, results of the experimental time complexity, a comparison to the Poisson surface reconstruction method, quantitative evaluations, and results using both image and video sequences.

The batch environment modeling method is also extended to the incremental case. Video frames of the environment are progressively provided and key-frames are selected. Each time a key-frame is selected, new Harris points are reconstructed. They, together with their visibilities in images, are subsequently used by an incremental surface reconstruction method to incrementally update the current surface of the environment. The processing time of each update does not increase with time (except at the end of a camera trajectory loop). At all times, the current surface of the environment is always a multi-genus 2-manifold surface. To our knowledge, this method is the first system with four features: *incremental* reconstruction for triangulated *manifold* surface from *sparse* point cloud generated by *SfM*. Detailed theoretical time complexity analysis of the method is given. At last, we provide experiments,

including results by varying different parameters of the method, the experimental complexities, a quantitative evaluation and comparison of our incremental surface reconstruction method to our batch method.

An omni-directional catadioptric camera is used in our experiments. But our surface reconstruction methods are also able to deal with data sets using other cameras. In the PhD work of Vadim Litvinov [2], our batch surface reconstruction method is used for images taken by a Ladybug camera. Besides, we also provide quantitative evaluation results using standard multi-view stereo data with ground truths.

Our sparse environment modeling methods could be used in a lot of applications. For example, we plan to use these methods in robotics, especially for autonomous navigation. We expect to extend the volume of cameras to localize thanks to the reconstructed surface of environments. Our methods could also be used for visualization after improving the surface using a dense stereo method, or be directly used for virtual visit using a low-performance device such as a tablet or a smartphone. Our reconstructed surfaces could also be used in augmented reality which handles occlusions.

Limitations

Here we discuss on the limitations of our environment modeling methods.

We remark that thin objects in environments such as tree branches, window concavities in a wall and electric posts cannot be correctly reconstructed due to the lack of points.

The second handle removal method is the slowest step of our batch method (see Sec. 7.2.1) and should be accelerated. Furthermore, although most of the spurious handles can be detected and removed, this process may fail in removing certain handles.

Results of the incremental surface reconstruction method are not as good as those of the batch method (Sec. 7.3.5). In addition, as shown in Sec. 7.3.2, the processing time of the incremental surface reconstruction at the end of a camera trajectory loop depends on the size of the loop, i.e. it is not limited to a constant.

The surface denoising operator that we used is an uniform Laplacian operator. As discussed in Sec. 4.7.2, surfaces with high curvatures are over-smoothed and the resulting surface is no more manifold.

The greedy region-growing process of our surface reconstruction methods has a shelling stuck problem, as already introduced in Sec. 4.4.3.3. In our work, the topology extension and also the second method of spurious handle removal can loosen up and restart the greedy region-growing. However it still remains an open theoretical problem for us: is our environment modeling method able to reach every *outside* region (whose border is 2-manifold) embedded in the 3d Delaunay triangulation ?

Perspectives

SfM Drift

The SfM drift due to the propagated geometry reconstruction errors in our incremental SfM is an important error source of our reconstructed surfaces. A priori informations can be used to improve the SfM accuracy. For example, as our camera is mounted on a car or on a helmet, the symmetric axis of our catadioptric camera can be assumed vertical, and the distance between a camera viewpoint and the ground can be considered constant. This distance can be measured and used as a constraint in the local bundle adjustment to limit the reconstruction error. More precisely, the drift of the scalar factor in SfM can be reduced.

Edges or Curves

More SfM points can be reconstructed in order to improve the completeness of our environment modelings. For example, the work [2] reconstructs curves in input images to enrich the reconstructed SfM points and computes surfaces of environments using our batch method. The resulting surfaces (see Sec. 7.4.3) have a better completeness and accuracy than those using only interest points, especially for indoor environments. In the future works, we plan to also use edges and curves in the incremental surface reconstruction.

Using SfM points of Unorganized Images

SfM points used by our surface reconstruction methods are reconstructed from sequences of consecutive images. We plan to check if our surface reconstruction methods are still applicable to SfM points reconstructed from unstructured collection of images such as images in [130] (Photo Tourism).

Visibility Score

The work [112] also reconstructs a surface which is consistent to visibility rays obtained by SfM. It is a probabilistic approach which considers that the intersection between a visibility ray and a triangle is less confident if the intersected point is close to the 3d point of the ray. By doing so, the effects of false SfM points, which have already been discussed in Sec. 4.8, are reduced. Inspired by this idea, in our work, we could assign a visibility score to each tetrahedron. And the score depends on the rays which intersect the tetrahedron and the confidence measure of each intersection. Then, *free-space* tetrahedra with a low visibility score can be removed from the *free-space* tetrahedra list.

Photo-consistency Driven Region-growing

The photo-consistency measurement might be helpful in our visibility optimization process. In our current work, an *outside* region O is grown under a rigid manifold constraint on the border δO of O. And we optimize the visibility information, i.e. the number of intersections between tetrahedra in O and visibility rays. We could also measure the photo-consistency

our surface and combine it in the optimization process. For example, for each *free-space* tetrahedron Δ adjacent to O, we could measure the photo-consistency of its triangle(s) in O and combine it with the visibility score of Δ to define its priority score. Δ has low priority score if its triangle(s) have high photo-consistent score(s). The photo-consistency score is particularly interesting to tetrahedra which have poor visibility information. We expect that the resulting surface would be more consistent to the input images than the current one. Note that the work [82] optimizes also the visibility information and the photo-consistency measures of Delaunay triangles. The difference is that its optimization process is realized using graph cuts techniques and the resulting surface is not guaranteed to be 2-manifold.

Loop Closure

A drawback of our incremental surface reconstruction approach is the complexity when the camera closes a trajectory loop: the time complexity is $\mathcal{O}(l_s)\log(l_s)$, where l_s is the size of the loop. In the PhD work of Vadim Litvinov, an alternative approach of incremental surface reconstruction is studied, which does not have this drawback. It would be based on a local update of the *outside* volume near the current location of the camera.

Improve Surface Denoising

Our reconstructed surfaces interpolate sparse SfM points which have usually a non-uniform point distribution. The surface denoising process of our environment modeling methods may be improved by applying an operator such as Laplacian-Beltrami operator [27]. By doing so, the denoising process could better deal with non-uniform data. Furthermore, we should also avoid self intersections of triangles to maintain the 2-manifold property in the post-processing step.

Image-based Automatic Modeling Pipeline

Remind that experiments of our environment modeling process are done in three separating steps: image or video sequence acquisition, geometry reconstruction and surface reconstruction. Output of the first (resp. second) step is input of the second (resp. last) step and communications between different steps are currently realized via ASCII files. For our incremental environment modeling method, it would be interesting to combine the separating three steps in a complete pipeline. By doing so, our environment modeling pipeline could be used in real time to reconstruct the surface model of environments.

Appendix A

Levenberg-Marquardt

This appendix presents the Levenberg-Marquardt (LM) algorithm [63, 116] used for bundle adjustment and its implementation in our work.

Remind that LM is an iterative numeric algorithm to minimize the non linear function error function $\|E\|^2$, where $E = (.., e^i_j, ..)^T$ and e^i_j is an error function involving the reconstructed camera pose $\mathbf{c}^i$ and 3d point $\mathbf{q}_j$. The parameter vector X of $\|E\|^2$ is $(\mathbf{c}^1, .., \mathbf{c}^m, \mathbf{q}_1, .., \mathbf{q}_n)^T$, which is simplified to $X = (X^c \; X^q)^T$

The algorithm begins at an initial guess of the solution X_0 and finally converges to a locally optimal solution. Now assume that the current parameter vector is X_k. In the next LM iteration, X_k is replaced by $X_{k+1} = X_k + \delta X$ to approximate the local minimum. Writing the first order Taylor expansion of the error function $E(X_{k+1})$, we have

$$E(X_k + \delta X_k) \approx E(X_k) + J_k \delta X_k \tag{A.1}$$

where, J_k is the Jacobian of E at X_k. A local minimum of $\delta X_k \mapsto \|E(X_k + \delta X)\|^2 \approx \|E(X_k) + J_k \delta X_k\|^2$ exists when the gradient of this function reaches zero. That means:

$$J_k^T J_k \delta X_k = -J_k^T E(X_k) \tag{A.2}$$

Finding δX_k by resolving the linear equation system A.2 is the principle of Gauss Newton algorithm. It converges rapidly if the initial guess is sufficiently close to the optimal solution. The contribution of Levenberg-Marquardt is to replace $J_k^T J_k$ by $J_k^T J_k + \lambda diag(J_k^T J_k), \lambda > 0$. Here, $diag(J_k^T J_k)$ is a matrix filled by zero except that its diagonal is same to the one of $J_k^T J_k$. We have,

$$(J_k^T J_k + \lambda_k diag(J_k^T J_k))\delta X_k = -J_k^T E(X_k) \tag{A.3}$$

λ_k is a positive parameter (initialized to 10^{-3} in practice) which evolves as follows. If the cost is reduced with the current λ_k, i.e. $E(X_{k+1}) < E(X_k)$, then λ_k is reduced in the next iteration (divided by 10) to accelerate the convergence. In this case the algorithm behaviors like a Gauss-Newton algorithm. Otherwise, λ_k is increased (multiplied by 10) and the algorithm behaviors more like a gradient descent algorithm to guarantee a decreasing cost function. We consider that the algorithm converges if $\|E(X_{k+1})\| < \|E(x_k)\| \leq \frac{\|E(X_{k+1})\|}{\sigma}$, where σ is a positive parameter which is set to be 0.9999 in practice.

Linear System Resolution The main implementation problem for LM which is the resolution of linear equation system in Eq. A.3. We suppose that X_k is composed of m camera poses and n 3d points (and eventually 4 intrinsic parameters which are not included in this example). Let matrix $H_k = J_k^T J_k + \lambda_k diag(J_k^T J_k)$, then H_k has a dimension of $(6m + 3n) \times (6m + 3n)$. In general, image sequences have hundreds or thousands of camera poses and from thousands to hundreds of thousands points. Directly inversing such a matrix is extremely expensive in terms of computation time.

An efficient solution to resolve this problem is to exploit the sparse structure of the matrix $J_k^T J_k$. In fact, J_k, the Jacobian matrix of $E(X_k)$, is composed of partial derivations of $E(X_k)$ with respect of all points and camera poses. Fortunately, all points are not viewed by all camera poses, thus J_k contains a majority of zero values. In other words, J_k is a sparse matrix. Consequently, $J_k^T J_k$ and $H_k = J_k^T J_k + \lambda diag(J_k^T J_k)$ are also sparse matrices.

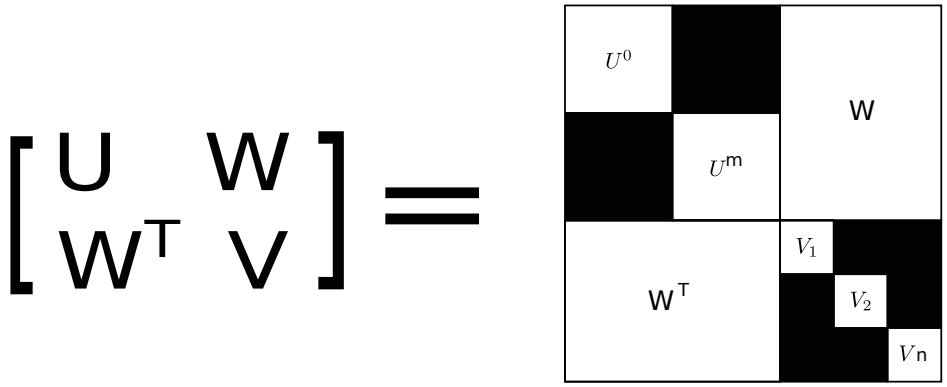


Figure A.1: Structure of sparse matrix H_k. black means zero values.

Fig. A.1 shows the structure of the sparse matrix H_k which is composed of four submatrices: U, W, W^T and V. U has a dimension of $6m \times 6m$ and is composed of m blocks of 6×6 in diagonal and null coefficients in other blocks. The i-th diagonal block contains product of partial derivatives of $E(X_k)$ with respect of the 6 extrinsic parameters of $\mathbf{c}_i$. The matrix W, has a dimension of $6m \times 3n$, and is composed of $m \times n$ blocks of 6×3. It represents the inter-correlations between points parameters and camera extrinsic parameters. The (i,j)-th 6×3 block $W(i, j)$ of W is 0 if the j-th point is not in the i-th image. V is a block-wise diagonal matrix which is invertible. The j-th diagonal block in V contains a sum of products of partial derivatives of $E(X_k)$ with respect to the 3 parameters of $\mathbf{q}^j$.

Then the linear equation system Eq. A.3 can be rewritten as follows,

$$\begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta X_k^c \\ \delta X_k^q \end{bmatrix} = \begin{bmatrix} E_k^c \\ E_k^q \end{bmatrix} \tag{A.4}$$

δX_k^c and δX_k^q are respectively steps for camera poses and 3d points in the iteration k. E_k^c and E_k^q are respectively $-\frac{1}{2}\times$ gradients of $\|E\|^2$ for camera poses and 3d points in the iteration k. As V is invertible, we can calculate the schur complement of U in H_k: $U - WV^{-1}W^T$ by

multiplying Eq. A.4 on the left by $\begin{bmatrix} I & -WV^{-1} \\ 0 & I \end{bmatrix}$. Then we have,

$$\begin{bmatrix} U - WV^{-1}W^T & 0 \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta X_k^c \\ \delta X_k^q \end{bmatrix} = \begin{bmatrix} E_k^c - WV^{-1}E_k^q \\ E_k^q \end{bmatrix} \tag{A.5}$$

Now the linear equation system (Eq. A.5) can then be resolved in two steps:

1. Calculate the increment of camera parameters by solving the equation:

$$(U - WV^{-1}W^T)\delta X_k^c = E_k^c - WV^{-1}E_k^q \tag{A.6}$$

2. Calculate the increment of point parameters

$$\delta X_k^q = V^{-1}(E_k^q - W^T \delta X_k^c) \tag{A.7}$$

We can see that directly solving Eq. A.4 consists in inversing the matrix H_k and it is computationally expensive. Now, by using schur complement, we solve subsequently Eq. A.6 and Eq. A.7 instead of directly solving Eq. A.4. And the process is much more efficient in terms of computation costs: $U - WV^{-1}W^T$ is a $6m \times 6m$ matrix (much more smaller than H_k) and Eq. A.6 can be solved by using Cholesky factorization; V is a block-wise diagonal matrix thus Eq. A.7 can be efficiently solved by block.

Appendix B

Proof of Tetrahedron-based Vertex 2-Manifold Test

Here we use theory on planar graphs and duality in [125] to show that the tetrahedron-based and edge-based tests in Sec. 4.4.3.1 are equivalent. This proof is also submitted in [1].

First, we introduce graphs G and G^*. Let $\mathbf{v}$ be a vertex on surface δO. Let V^* be the list of $\mathbf{v}$-incident tetrahedra of the Delaunay T. Let E^* be the list of triangles between adjacent tetrahedra in V^*. Graph $G^* = (V^*, E^*)$ has vertices V^* and edges E^*. Let V be the list of vertices of the tetrahedra of V^*, except $\mathbf{v}$. Let E be the list of edges of the tetrahedra of V^* without endvertex $\mathbf{v}$. Graph $G = (V, E)$ has vertices V and edges E.

Fig. B.1 shows an example. There are eight $\mathbf{v}$-incident tetrahedra whose union is an octahedron centered at $\mathbf{v}$. G has the vertices and edges of the octahedron, and G^* has the vertices and edges of a cube. Note that G^* is also the adjacency graph of the $\mathbf{v}$-opposite triangles in the $\mathbf{v}$-incident tetrahedra.

Second we detail the duality between G and G^*. Graph G is planar since it lies on the border of a $\mathbf{v}$-star-shaped volume which is homeomorphic to a 2-sphere. Remind that a planar graph (e.g. G) is embedded in $\mathbb{R}^2$, it cuts $\mathbb{R}^2$ into regions called faces such that every edge has two faces (one on the left and one on the right). The dual graph of G is defined as follows: its vertices are the faces of G, its edges connect the left and right faces of an edge in G. Now we see that G^* is the dual graph of G.

Third we explicit the duality between edges due to the *inside-outside* labeling of tetrahedra. We have a nontrivial bipartition of V^*: V_I^* (*inside* tetrahedra) and V_O^* (*outside* tetrahedra). This implicitly defines a bipartition of *inside* and *outside* ($\mathbf{v}$-opposite) triangles. Let F^* be the edges of graph G^* having an endvertex in V_I^* and an endvertex in V_O^*. Let F be the edges of graph G having an *inside* triangle and an *outside* triangle as its two faces. The edges F^* of graph G^* are the duals of edges F of graph G (one example in Fig. B.1).

Last we show that tetrahedron-based and edge-based tests give the same result. According to the tetrahedron-based test, $\mathbf{v}$ is regular if and only if V_I^* and V_O^* are the connected components of the graph $(V^*, E^* \setminus F^*)$, i.e. if and only if F^* is a minimal edge-cut of G^*. According to the edge-based test, $\mathbf{v}$ is regular if and only if F is a cycle in graph G. Now we use proposition 7 in [125]: "Let $G = (V, E)$ be a connected plane multigraph. A set F

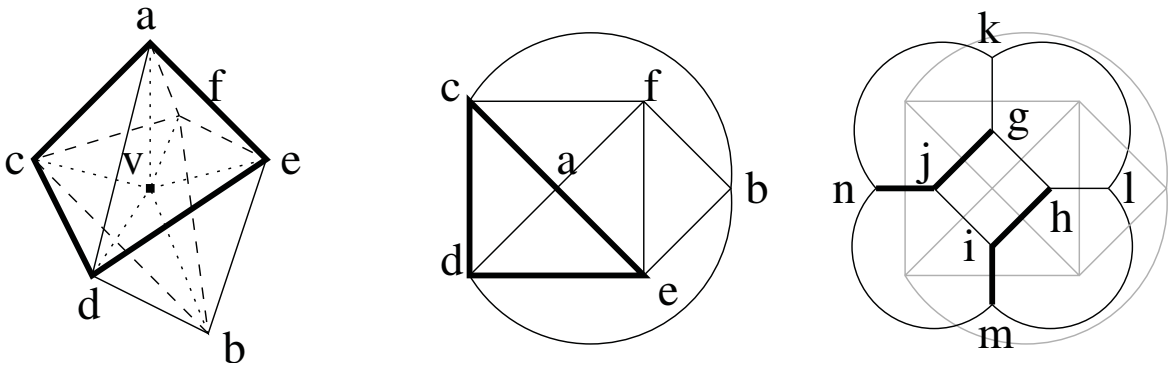


Figure B.1: Duality of planar graphs. Left: the eight **v**-adjacent tetrahedra (octahedron volume), and border (bold edges) between *inside* and *outside* tetrahedra. Middle: planar graph G and bold edges F. Right: planar graph G^* and bold edges F^*. The dual of G (octahedron) is G^* (cube). The dual of $F = \{ac, cd, de, ea\}$ is $F^* = \{jg, jn, im, ih\}$.

included in E is a cycle in G if and only if F^* is a minimal edge-cut in G^*". We obtain the result. Note that a multigraph is a graph allowing loop edges and parallel edges, that our graphs have not.

Appendix C

Proof of the Single Tetrahedron 2-Manifold Test

In this part, T is the Delaunay triangulation, O is a list of tetrahedra of T, $O_{\mathbf{v}}$ is the list of the tetrahedra in O which are incident to vertex $\mathbf{v}$. The border δO of O is the list of triangles of tetrahedra in O such that every triangle is included in exactly one tetrahedron of O.

We can add a tetrahedron Δ in O if and only if the condition of the following Theorem is met.

Theorem C.1 (Add One Tetrahedron) *Assume that δO is 2-manifold. Let Δ be a tetrahedron in $T \setminus O$ and f be the number of Δ-triangles included in δO. Then, the border $\delta(O \cup \{\Delta\})$ of $O \cup \{\Delta\}$ is 2-manifold if and only if one of the following conditions is meet:*

- *if $f = 0$ and every Δ-vertex $\mathbf{v}$ meets $O_{\mathbf{v}} = \emptyset$*
- *if $f = 1$ and the Δ-vertex $\mathbf{v}$ which is not in δO meets $O_{\mathbf{v}} = \emptyset$*
- *if $f = 2$ and the Δ-edge which is not in δO has end-vertices $\mathbf{v}$ and $\mathbf{w}$ such that $O_{\mathbf{v}} \cap O_{\mathbf{w}} = \emptyset$*
- *if $f = 3$*
- *if $f = 4$.*

In practice, the lists $O_{\mathbf{v}}$ are ordered (e.g. by increasing indices of tetrahedra) so that the computation of $O_{\mathbf{v}} \cap O_{\mathbf{w}}$ has linear complexity in the sizes of $O_{\mathbf{v}}$ and $O_{\mathbf{w}}$.

The following proof is submitted in [1]. Although this theorem is derived from Figures of [23], it should be stressed that there is no formal proof in [23] or even in other previous work (up to our knowledge).

C.1 Principle of the Proof

Let L be a list of tetrahedra of T. A path in L between Δ_a and Δ_b is a tetrahedron series $\Delta_i \in L, i \in \{0, 1, \cdots k\}$ such that $\Delta_0 = \Delta_a$, $\Delta_k = \Delta_b$, and Δ_i and Δ_{i+1} are adjacent (i.e. $\Delta_i \cap \Delta_{i+1}$ is a triangle) for every $i \in \{0, 1, \cdots k-1\}$. We say that L is connected if and only if there is a path in L between every pair of tetrahedra in L. We use the following notation: $T_{\mathbf{v}}$ is the list of all tetrahedra which are incident to vertex $\mathbf{v}$, T_e is the list of all tetrahedra including edge e, g_e is the adjacency graph of all tetrahedra of T_e (g_e is a cycle).

In almost all cases, our proof is based on the Tetrahedron-based Test that we rewrite here for both surfaces δO and $\delta(O \cup \{\Delta\})$:

Lemma C.1 *A vertex* $\mathbf{v}$ *of* δO *is regular in* δO *if and only if* $O_{\mathbf{v}}$ *is connected and* $T_{\mathbf{v}} \setminus O_{\mathbf{v}}$ *is connected. Assume that* $\mathbf{v}$ *is a vertex of* Δ *and* $\delta(O \cup \{\Delta\})$, $\mathbf{v}$ *is regular in* $\delta(O \cup \{\Delta\})$ *if and only if* $O_{\mathbf{v}} \cup \{\Delta\}$ *is connected and* $T_{\mathbf{v}} \setminus \{O_{\mathbf{v}} \cup \{\Delta\}\}$ *is connected.*

The principle of the proof is the following: for every vertex $\mathbf{v}$ of Δ, we check that $\mathbf{v}$ is regular (or singular) in surface $\delta(O \cup \{\Delta\})$ by studying the connectivity of $O_{\mathbf{v}} \cup \{\Delta\}$ and $T_{\mathbf{v}} \setminus \{O_{\mathbf{v}} \cup \{\Delta\}\}$ using the fact that δO is 2-manifold (i.e. $O_{\mathbf{v}}$ and $T_{\mathbf{v}} \setminus O_{\mathbf{v}}$ are connected). Then $\delta(O \cup \{\Delta\})$ is 2-manifold if and only if every Δ-vertex is regular in $\delta(O \cup \{\Delta\})$.

The other vertices of $\delta(O \cup \{\Delta\})$ does not need to be checked since their incident triangles are the sames in both surfaces δO, which is 2-manifold, and $\delta(O \cup \{\Delta\})$

The proof depends on f, the number of adjacencies between Δ and the tetrahedra in O, i.e. the number of triangles which are in $(\delta\Delta) \cap (\delta O)$.

C.2 Δ is not adjacent to a tetrahedron in O, i.e. $f = 0$

Let $\mathbf{v}$ be a Δ-vertex. If $O_{\mathbf{v}} = \emptyset$, $O_{\mathbf{v}} \cup \{\Delta\}$ is connected and $T_{\mathbf{v}} \setminus \{O_{\mathbf{v}} \cup \{\Delta\}\}$ is connected (since $\mathbf{v}$ is regular in $\delta\Delta$). Then $\mathbf{v}$ is regular in $\delta(O \cup \{\Delta\})$.

If $O_{\mathbf{v}} \neq \emptyset$, there are (at least) two tetrahedra Δ and Δ' in $O_{\mathbf{v}} \cup \{\Delta\}$. Since $f = 0$, Δ is not adjacent to the tetrahedra in $O_{\mathbf{v}}$. Then, there is no path in $O_{\mathbf{v}} \cup \{\Delta\}$ between Δ and Δ'. Thus $O_{\mathbf{v}} \cup \{\Delta\}$ is not connected and $\mathbf{v}$ is singular in $\delta(O \cup \{\Delta\})$.

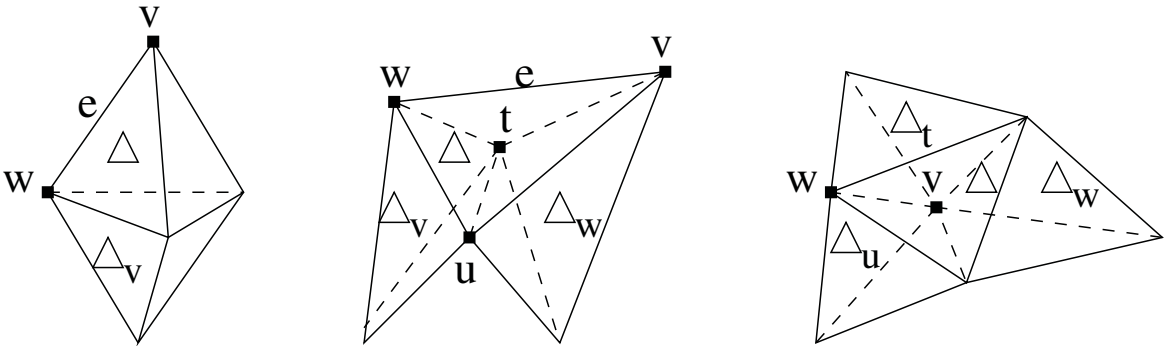


Figure C.1: Single tetrahedron 2-manifold test in several cases. Cases $f = 1$ (left), $f = 2$ (middle) and $f = 3$ (right) Tetrahedron Δ has vertices $\mathbf{t}, \mathbf{u}, \mathbf{v}, \mathbf{w}$ and edge e. We have $\Delta \notin O$. Tetrahedra $\Delta_{\mathbf{t}}, \Delta_{\mathbf{u}}, \Delta_{\mathbf{v}}, \Delta_{\mathbf{w}}$ are in O and adjacent to Δ ($\Delta_{\mathbf{v}}$ is opposite to $\mathbf{v}$ for Δ).

C.3 Δ is adjacent to one tetrahedron in O, i.e. $f = 1$

Let $\mathbf{v}$ be the Δ-vertex opposite to the triangle of $\delta\Delta$ which is in δO, as illustrated in Fig. C.1 (left). This and $f = 1$ imply that Δ is not adjacent to the tetrahedra in $O_{\mathbf{v}}$. According to Section C.2, $\mathbf{v}$ is regular in $\delta(O \cup \{\Delta\})$ if and only if $O_{\mathbf{v}} = \emptyset$. Thus $\delta(O \cup \{\Delta\})$ is 2-manifold implies that $O_{\mathbf{v}} = \emptyset$.

Conversely, we assume that $O_{\mathbf{v}} = \emptyset$ and show that every other vertex $\mathbf{w}$ of Δ is regular in $\delta(O \cup \{\Delta\})$. To do this, we show that $O_{\mathbf{w}} \cup \{\Delta\}$ is connected and $T_{\mathbf{w}} \setminus \{O_{\mathbf{w}} \cup \{\Delta\}\}$ is connected.

We have $\mathbf{w} \in \delta O$ and δO is 2-manifold, then $O_{\mathbf{w}}$ is connected and $T_{\mathbf{w}} \setminus O_{\mathbf{w}}$ is connected. Since $f = 1$, Δ is adjacent to one tetrahedron Δ' of $O_{\mathbf{w}}$. Now we see that $O_{\mathbf{w}} \cup \{\Delta\}$ is connected.

Let Δ_a, Δ_b be in $T_{\mathbf{w}} \setminus \{O_{\mathbf{w}} \cup \{\Delta\}\}$. Since $T_{\mathbf{w}} \setminus O_{\mathbf{w}}$ is connected, there is a path P_1 defined by $\Delta_i \in T_{\mathbf{w}} \setminus O_{\mathbf{w}}, i \in \{0, 1 \cdots k\}$ between Δ_a and Δ_b. If there is i_0 such that $\Delta_{i_0} = \Delta$, $\Delta_{i_0-1} \cap \Delta \cap \Delta_{i_0+1}$ is an edge e of Δ. This edge has end-vertex $\mathbf{w}$ (since $\Delta_i \in T_{\mathbf{w}}$) and end-vertex $\mathbf{v}$ (otherwise $\Delta' \in \{\Delta_{i_0-1}, \Delta_{i_0+1}\}$, which is impossible since $\Delta' \in O$ and $\Delta_i \notin O$). Thus $e = \mathbf{vw}$, which implies $T_e \cap O \subseteq O_{\mathbf{v}} = \emptyset$ and then $T_e \subseteq T_{\mathbf{w}} \setminus O_{\mathbf{w}}$. Furthermore g_e is a cycle: there is a path P_2 in $T_e \setminus \{\Delta\}$ between Δ_{i_0-1} and Δ_{i_0+1}. In P_1, we replace $(\Delta_{i_0-1}, \Delta, \Delta_{i_0+1})$ by P_2 and obtain a path in $T_{\mathbf{w}} \setminus \{O_{\mathbf{w}} \cup \{\Delta\}\}$ between Δ_a and Δ_b. Thus $T_{\mathbf{w}} \setminus \{O_{\mathbf{w}} \cup \{\Delta\}\}$ is connected.

C.4 Δ is adjacent to two tetrahedra in O, i.e. $f = 2$

As showed in Fig. C.1 (middle), the two triangles which are both in $\delta\Delta$ and δO have two vertices $\mathbf{t}$ and $\mathbf{u}$. Let $\mathbf{v}$ and $\mathbf{w}$ be the two other Δ-vertices.

First we show that $\mathbf{u}$ (and similarly $\mathbf{t}$) is regular in $\delta(O \cup \{\Delta\})$. Since $\mathbf{u} \in \delta O$ and δO is 2-manifold, $O_{\mathbf{u}}$ is connected and $T_{\mathbf{u}} \setminus O_{\mathbf{u}}$ is connected. Since $f = 2$, Δ is adjacent to two tetrahedra of $O_{\mathbf{u}}$. We see that $O_{\mathbf{u}} \cup \{\Delta\}$ is connected. Let Δ_a and Δ_b be in $T_{\mathbf{u}} \setminus \{O_{\mathbf{u}} \cup \{\Delta\}\}$. Since $T_{\mathbf{u}} \setminus O_{\mathbf{u}}$ is connected, there is a path $\Delta_i \in T_{\mathbf{u}} \setminus O_{\mathbf{u}}, i \in \{0, 1 \cdots k\}$ between Δ_a and Δ_b.

If there is i_0 such that $\Delta_{i_0} = \Delta$, we have $\Delta_{i_0-1} = \Delta_{i_0+1}$ since Δ is adjacent to exactly one tetrahedron of $T_{\mathbf{u}} \setminus O_{\mathbf{u}}$ (indeed, Δ is adjacent to 3 tetrahedra in $T_{\mathbf{u}}$ and 2 tetrahedra in $O_{\mathbf{u}}$). Then we remove (Δ, Δ_{i_0+1}) from the path and obtain a path in $T_{\mathbf{u}} \setminus \{O_{\mathbf{u}} \cup \{\Delta\}\}$ between Δ_a and Δ_b. We see that $T_{\mathbf{u}} \setminus \{O_{\mathbf{u}} \cup \{\Delta\}\}$ is connected and $O_{\mathbf{u}} \cup \{\Delta\}$ is connected, then $\mathbf{u}$ is regular in $\delta(O \cup \{\Delta\})$.

Second we examine $\mathbf{v}$ (and similarly $\mathbf{w}$). We assume $O_{\mathbf{v}} \cap O_{\mathbf{w}} = \emptyset$ and show that $O_{\mathbf{v}} \cup \{\Delta\}$ is connected and $T_{\mathbf{v}} \setminus \{O_{\mathbf{v}} \cup \{\Delta\}\}$ is connected, which imply that $\mathbf{v}$ is regular in $\delta(O \cup \{\Delta\})$. We have $\mathbf{v} \in \delta O$ and δO is 2-manifold, then $O_{\mathbf{v}}$ is connected and $T_{\mathbf{v}} \setminus O_{\mathbf{v}}$ is connected. Since $O_{\mathbf{v}}$ is connected and Δ is adjacent to one tetrahedron of $O_{\mathbf{v}}$, we see that $O_{\mathbf{v}} \cup \{\Delta\}$ is connected. Let Δ_a, Δ_b be in $T_{\mathbf{v}} \setminus \{O_{\mathbf{v}} \cup \{\Delta\}\}$. Since $T_{\mathbf{v}} \setminus O_{\mathbf{v}}$ is connected, there is a path P_1 defined by $\Delta_i \in T_{\mathbf{v}} \setminus O_{\mathbf{v}}, i \in \{0, 1 \cdots k\}$ between Δ_a and Δ_b. If there is i_0 such that $\Delta_{i_0} = \Delta$, $\Delta_{i_0-1} \cap \Delta \cap \Delta_{i_0+1}$ is an edge e of Δ. Since $\Delta_{i_0-1} \notin O$ and $\Delta_{i_0+1} \notin O$, $e = \mathbf{vw}$. Thus $T_e \cap O \subseteq O_{\mathbf{v}} \cap O_{\mathbf{w}} = \emptyset$ implies $T_e \subseteq T_{\mathbf{v}} \setminus O_{\mathbf{v}}$. Furthermore g_e is a cycle, this implies that there is a path P_2 in $T_e \setminus \{\Delta\}$ between Δ_{i_0-1} and Δ_{i_0+1}. In P_1, we replace $(\Delta_{i_0-1}, \Delta, \Delta_{i_0+1})$ by P_2 and obtain a path in $T_{\mathbf{v}} \setminus \{O_{\mathbf{v}} \cup \{\Delta\}\}$ between Δ_a and Δ_b. Thus $T_{\mathbf{v}} \setminus \{O_{\mathbf{v}} \cup \{\Delta\}\}$ is connected.

Conversely, we assume that $O_{\mathbf{v}} \cap O_{\mathbf{w}} \neq \emptyset$ and show that $\delta(O \cup \{\Delta\})$ is not 2-manifold. Since $f = 2$, cycle g_e of edge $e = \mathbf{vw}$ contains three successive and different tetrahedra $\Delta_a, \Delta, \Delta_b$ which are not in O. Since $O_{\mathbf{v}} \cap O_{\mathbf{w}} \neq \emptyset$, there exists $\Delta_c \in O_{\mathbf{v}} \cap O_{\mathbf{w}} = T_e \cap O$. We see that g_e contains (at least) four pairs of consecutive tetrahedra Δ_i, Δ_{i+1} (indices modulo the cycle length) such that $\{\ \Delta_i \in O \cup \{\Delta\}$ and $\Delta_{i+1} \notin O \cup \{\Delta\}\ \}$ or $\{\ \Delta_i \notin O \cup \{\Delta\}$ and $\Delta_{i+1} \in O \cup \{\Delta\}\ \}$. This implies that e is included in (at least) four triangles of $\delta(O \cup \Delta)$. Assuming that $\delta(O \cup \Delta)$ is 2-manifold (reductio ad absurdum), e is included in only two triangles of $\delta(O \cup \Delta)$ (read p. 723 of [58]). Then $\delta(O \cup \Delta)$ is not 2-manifold.

C.5 Δ is adjacent to four tetrahedra in O, i.e. $f = 4$

Since $f \geq 3$, every Δ-vertex $\mathbf{v}$ is in δO. Then $T_{\mathbf{v}} \setminus O_{\mathbf{v}}$ is connected since δO is 2-manifold. Furthermore $T_{\mathbf{v}} \setminus O_{\mathbf{v}}$ contains Δ, which is not adjacent to other tetrahedra in $T_{\mathbf{v}} \setminus O_{\mathbf{v}}$ (Δ is adjacent to 3 $\mathbf{v}$-incident tetrahedra, which are in $O_{\mathbf{v}}$). We see that $T_{\mathbf{v}} \setminus O_{\mathbf{v}} = \{\Delta\}$. Thus $\mathbf{v}$ is not in $\delta(O \cup \{\Delta\})$.

C.6 Δ is adjacent to three tetrahedra in O, i.e. $f = 3$

As in Section C.5, $O_{\mathbf{v}}$ is connected and $T_{\mathbf{v}} \setminus O_{\mathbf{v}}$ is connected for every Δ-vertex $\mathbf{v}$ (see the right of Fig. C.1 for an example). Assume that $\mathbf{v}$ is the vertex incident to the three triangles of $\delta\Delta$ which are in δO. As in Section C.5, $\mathbf{v}$ is not in $\delta(O \cup \{\Delta\})$.

Assume that $\mathbf{w}$ is one of the three other vertices of Δ. Since $O_{\mathbf{w}}$ is connected and Δ is adjacent to two tetrahedra of $O_{\mathbf{w}}$, $O_{\mathbf{w}} \cup \{\Delta\}$ is connected. Let Δ_a and Δ_b be two tetrahedra of $T_{\mathbf{w}} \setminus \{O_{\mathbf{w}} \cup \{\Delta\}\}$. Since $T_{\mathbf{w}} \setminus O_{\mathbf{w}}$ is connected, there is a path $\Delta_i \in T_{\mathbf{w}} \setminus O_{\mathbf{w}}, i \in \{0, 1 \cdots k\}$ between Δ_a and Δ_b. If there is i_0 such that $\Delta_{i_0} = \Delta$, $\Delta_{i_0-1} = \Delta_{i_0+1}$ (since Δ is connected to a single tetrahedron not in O and $\Delta_i \notin O$). We remove (Δ, Δ_{i_0+1}) from the path and

obtain a path in $T_{\mathbf{w}} \setminus \{O_{\mathbf{w}} \cup \{\Delta\}\}$ between Δ_a and Δ_b. Then $\mathbf{w}$ is regular in $\delta(O \cup \{\Delta\})$ since $\{O_{\mathbf{w}} \cup \{\Delta\}\}$ and $T_{\mathbf{w}} \setminus \{O_{\mathbf{w}} \cup \{\Delta\}\}$ are connected.

C. PROOF OF THE SINGLE TETRAHEDRON 2-MANIFOLD TEST

Appendix D

Solid Angle Calculation

Here are details about the solid angle of a tetrahedron **vabc** at vertex **v** and how to calculate it. As showed by Fig. D.1, the solid angle w on vertex **v** is the area in a unit sphere covered by the projected triangle $\triangle \mathbf{a'b'c'}$ of $\triangle \mathbf{abc}$ onto the sphere. The solid angle of a sphere is 4π. Besides, the threshold w_0 is set to be the solid angle of one eighth of a unit sphere, which is $\pi/2$.

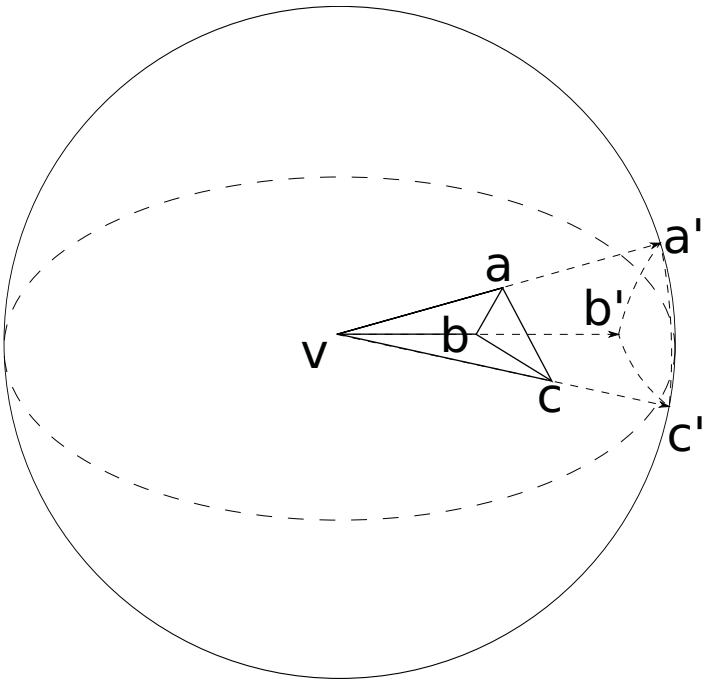


Figure D.1: Example of a solid angle. The solid angle of a tetrahedron **vabc** on **v** is the area in a unit sphere covered by the projected triangle $\triangle \mathbf{a'b'c'}$ of $\triangle \mathbf{abc}$ onto the sphere

To calculate w, an efficient method proposed by Oosterom and Strackee [144] is used.

D. SOLID ANGLE CALCULATION

Let $\vec{a}, \vec{b}, \vec{c}$ be vectors $\mathbf{a} - \mathbf{v}, \mathbf{b} - \mathbf{v}, \mathbf{c} - \mathbf{v}$ respectively. We have,

$$\tan\left(\frac{1}{2}w\right) = \frac{|\vec{a}\ \vec{b}\ \vec{c}|}{\|\vec{a}\|\|\vec{b}\|\|\vec{c}\| + (\vec{a}\cdot\vec{b})\|\vec{c}\| + (\vec{a}\cdot\vec{c})\|\vec{b}\| + (\vec{b}\cdot\vec{c})\|\vec{a}\|} \tag{D.1}$$

where $\left|\vec{a}\ \vec{b}\ \vec{c}\right|$ is the determinant of the matrix $[\vec{a}\ \ \vec{b}\ \ \vec{c}]$, and $\|\vec{a}\|$ is the norm of $\vec{a}$. $\frac{1}{2}w \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ thus $w \in [-\pi, \pi]$. As a solid angle is always positive, we apply $w \leftarrow w + 2\pi$ if $w < 0$, and finally $w \in [0, 2\pi]$.

Appendix E

Proofs of Complexities

E.1 Bounded Radius of Tetrahedron Circumsphere

Let $A = \{-1, 0, 1, 2\}$. Without loss of generality, the grid contains vertices A^3, a point $\mathbf{q}$ is in the cube $[0, 1]^3$ and the length of the cube diagonal is $\sqrt{3}$. Let $\mathbf{c}$ be the center of a circumsphere $\mathcal{S}$ of a tetrahedron which has vertex $\mathbf{q}$. Now we show that $||\mathbf{c} - \mathbf{q}|| \leq \sqrt{3}/2$.

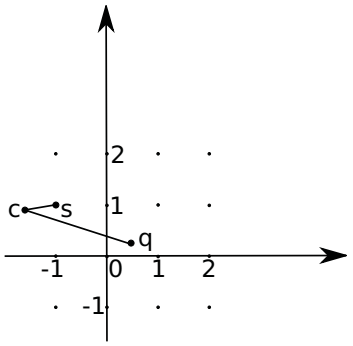


Figure E.1: Relation between $\mathbf{c}$ and $\mathbf{q}$ (2d case). We see grid vertices (small round points) of A^2 and another point $\mathbf{q}$ is in square $[0, 1]^3$. If point $\mathbf{c} \notin [-1, 2]^2$, then we have $\min_{s \in A^3} \|\mathbf{c} - \mathbf{s}\| < \|\mathbf{c} - \mathbf{q}\|$.

Assume (reductio ad absurdum) that $\mathbf{c} \notin [-1, 2]^3$. According to Fig. E.1, we show that

$$\min_{\mathbf{s} \in A^3} ||\mathbf{c} - \mathbf{s}|| < ||\mathbf{c} - \mathbf{q}||. \tag{E.1}$$

Using coordinates of vectors $\mathbf{c}$, $\mathbf{s}$ and $\mathbf{q}$, Eq. E.1 means that

$$\min_{s_x \in A}(c_x - s_x)^2 + \min_{s_y \in A}(c_y - s_y)^2 + \min_{s_z \in A}(c_z - s_z)^2 \tag{E.2}$$

is less than

$$(c_x - q_x)^2 + (c_y - q_y)^2 + (c_z - q_z)^2. \tag{E.3}$$

Using $u \in \{x, y, z\}$, we introduce

$$f_u = (c_u - q_u)^2 - \min_{s_u \in A} (c_u - s_u)^2. \tag{E.4}$$

Thanks to $0 \leq q_u \leq 1$ and $s_u \in \{-1, 0, 1, 2\}$, it can be shown that

$$\begin{aligned}
c_u \in [-1, 2] \quad &\Rightarrow \quad f_u \geq - \min_{s_u \in A} (c_u - s_u)^2 f_u \geq -\frac{1}{4} \\
c_u \leq -1 \quad &\Rightarrow \quad f_u = (c_u - q_u)^2 - (c_u + 1)^2 \\
&\qquad\quad f_u = q_u^2 - 1 - 2c_u(q_u + 1) \\
&\qquad\quad f_u(c_u = -1) = q_u^2 - 1 + 2(q_u + 1) = (q_u + 1)^2 \geq 1 \\
&\qquad\quad \text{Besides, } \frac{\partial f_u}{\partial c_u} = -2(q_u + 1) \leq 0 \\
&\qquad\quad \text{Thus, } f_u(c_u \leq -1) \geq 1 \\
2 \leq c_u \quad &\Rightarrow \quad f_u = (c_u - q_u)^2 - (c_u - 2)^2 \\
&\qquad\quad f_u = q_u^2 - 4 + 2c_u(2 - q_u) \\
&\qquad\quad f_u(c_u = 2) = q_u^2 - 4 + 4(2 - q_u) = (q_u - 2)^2 \geq 1 \\
&\qquad\quad \text{Besides, } \frac{\partial f_u}{\partial c_u} = 2 - q_u \geq 0 \\
&\qquad\quad \text{Thus, } f_u(c_u \geq 2) \geq 1.
\end{aligned} \tag{E.5}$$

Since $\mathbf{c} \notin [-1, 2]^3$, there is $u \in \{x, y, z\}$ such that $c_u \notin [-1, 2]$. Thus $f_x + f_y + f_z \geq 1 - 1/4 - 1/4 > 0$ and Eq. E.1 is meet.

However, $||\mathbf{c} - \mathbf{q}||$ is the $\mathcal{S}$ radius and Eq. E.1 means that the interior of $\mathcal{S}$ contains a vertex $\mathbf{s}$ of the grid. This is not possible in a 3d Delaunay triangulation. Thus assumption $\mathbf{c} \notin [-1, 2]^3$ is wrong.

Since $\mathbf{c} \in [-1, 2]^3$, $\mathbf{c}$ is in a cube of A^3 grid. So there is a vertex $\mathbf{s} \in A^3$ such that $||\mathbf{s} - \mathbf{c}||^2 \leq (1/2)^2 + (1/2)^2 + (1/2)^2$, i.e. $||\mathbf{s} - \mathbf{c}|| \leq \sqrt{3}/2$. Since the interior of $\mathcal{S}$ does not contains $\mathbf{s}$, we see that $||\mathbf{c} - \mathbf{q}|| \leq ||\mathbf{s} - \mathbf{c}|| \leq \sqrt{3}/2$.

E.2 Complexity of Tight Incremental 2-manifold Generation

The incremental 2-manifold generation step is composed by several successive region-growings, which grow in

$$F_t = \mathcal{L}_{(i_0-1)l} \cup \cdots \cup \mathcal{L}_t : \tag{E.6}$$

from $O_{i_0 l}$ to $O_{(i_0+1)l}$, from $O_{(i_0+1)l}$ to $O_{(i_0+2)l}$, ... from $O_{i_t l}$ to O_t.

Let $I = \{i_0, i_0+1, \cdots, i_t\}$. The i-th region-growing has a complexity of $\mathcal{O}((g_i+q_i)\log(g_i+q_i))$ (already shown in Sec. 6.2.2.4). And the complexity of incremental 2-manifold generation is $\mathcal{O}(\sum_{i \in I}(g_i + q_i)\log(g_i + q_i))$.

Now in the following paragraphs we want to show, for large values of $t - d_t$, that

$$\sum_{i \in I} g_i = \mathcal{O}(t - d_t) \tag{E.7}$$

and

$$\sum_{i \in I} q_i = \mathcal{O}(t - d_t) \tag{E.8}$$

We note $|L|$ the number of tetrahedra in a list L of tetrahedra. The successive region-growings in F_t imply that

$$\sum_{i \in I} g_i \leq |F_t| \tag{E.9}$$

$\mathcal{L}_t$ is a list of *free-space* tetrahedra whose creation date is t. When we add one point at time t, we create at most d tetrahedra which have creation date t. As the number of new points added in the Delaunay triangulation at t is $\mathcal{O}(1)$ (H4 Sec. 6.3.1.3) and the vertex degree is bounded (H7 Sec. 6.3.1.3), we have,

$$|\mathcal{L}_t| = \mathcal{O}(1) \tag{E.10}$$

Remember that i_0 is the largest integer such that $i_0 l < d_t$. Thus $(i_0 + 1)l \geq d_t$ and $-i_0 l \leq -d_t + l$. For large $t - d_t$ and $l = \mathcal{O}(1)$, we obtain

$$\sum_{i'=(i_0-1)l}^{t} 1 = t - i_0 l + l + 1 \leq t - d_t + 2l + 1 = \mathcal{O}(t - d_t). \tag{E.11}$$

Thanks to Eqs. E.6, E.10 and E.11, we obtain

$$|F_t| = \sum_{i'=(i_0-1)l}^{t} |\mathcal{L}_{i'}| = \mathcal{O}(\sum_{i'=(i_0-1)l}^{t} 1) = \mathcal{O}(t - d_t). \tag{E.12}$$

We see that Eqs. E.9 and E.12 imply Eq. E.7.

Furthermore, $l \geq 1$, $i_t l \leq t$ and Eq. E.11 imply

$$\sum_{i=i_0}^{i_t} 1 = i_t - i_0 + 1 \leq l(i_t - i_0 + 1) \leq t - (i_0 - 1)l = \mathcal{O}(t - d_t). \tag{E.13}$$

Since the initial priority queue collects $b_0 + l + 1 = \mathcal{O}(1)$ layers and $|\mathcal{L}_t| = \mathcal{O}(1)$, we have $q_i = \mathcal{O}(1)$. Then $q_i = \mathcal{O}(1)$ and Eq. E.13 imply Eq. E.8.

E.3 Conjecture on the Time Complexity of 3D Delaunay Triangulation under our Assumptions

Here we explain our conjecture in Sec. 6.3.1.3: the worst case time complexity of 3d Delaunay is linear thanks to the assumptions in Sec. 6.3.1.3. We do not investigate the robustness effects of points in non general position.

At the beginning, we initialize the 3d Delaunay triangulation D by n_s Steiner vertices, which are the corners of the cartesian grid. Since the eight vertices of a grid cube are on the

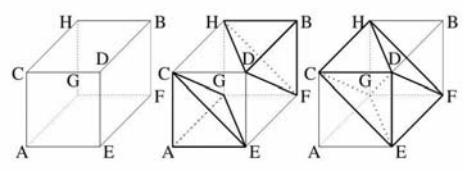


Figure E.2: 3d Delaunay triangulation for every cube. Left: eight vertices. Middle: opposite tetrahedra ACEG and BDFH in the cube. Right: tetrahedra DGCE, DGEF, DGFH and DGHC around diagonal DG. The triangulation is consistent with those of the adjacent cubes thanks to parallel edges: CE//HF, CG//DF, HD//GE.

same sphere, we can choose the Delaunay tetrahedra of every cube as in Fig. E.2. Thus, the time complexity of this initialization is $\mathcal{O}(n_s)$.

Then, $\mathcal{O}(n)$ reconstructed points are added in D, one-by-one. We show that adding one reconstructed point $\mathbf{q}$ to D has time complexity $\mathcal{O}(1)$. The addition of $\mathbf{q}$ to D has three steps [49]: localization, deletion of a list L_d of old tetrahedra, creation of a list L_c of new tetrahedra.

Firstly, we study the localization step. We find the closest Steiner vertex $\mathbf{v}$ from $\mathbf{q}$. Thanks to the cartesian grid filling the rectangular bounding box, this is $\mathcal{O}(1)$ and the distance between $\mathbf{v}$ and $\mathbf{q}$ is less than voxel diagonal l. Thus we can go from $\mathbf{v}$ to the tetrahedron $\Delta_{\mathbf{q}}$ which contains $\mathbf{q}$ by $\mathbf{vq}$-tracing in $\mathcal{O}(1)$ (according to Sec. 6.3.2.2). This is the localization step, and it is $\mathcal{O}(1)$.

Secondly, we study the deletion step. Since D is Delaunay, L_d is the list of tetrahedra whose circumsphere contains $\mathbf{q}$. The diameters of these spheres are less than l (Sec. E.1). This implies that L_d volume is included in the l-ball centered at $\mathbf{q}$. This l-ball can be covered by a bounded number of p'-balls such that every p'-ball intersects at most q' tetrahedra (H1). Thus L_d has a bounded number of tetrahedra. It is computed as the list of tetrahedra by a traversal in graph D started from $\Delta_{\mathbf{q}}$, while the current tetrahedron has his circumsphere which contains $\mathbf{q}$. We see that the calculation of L_d is $\mathcal{O}(1)$. The deletion step is $\mathcal{O}(1)$.

Lastly, there is the creation step. The tetrahedra of L_c connect vertex $\mathbf{q}$ to triangles, which are the border of L_d. Since list L_d is bounded, list L_c is also bounded. Now we see that the creation step is also $\mathcal{O}(1)$.

Publications of the Author

Submission in:

- International Journal

[1] M. Lhuillier and S. Yu. Manifold surface reconstruction of an environment from sparse structure-from-motion data. *Submitted to Computer Vision and Image Understanding (CVIU) in December 2012*.

In Proceedings:

- International Conferences

[2] V. Litvinov, S. Yu, and M. Lhuillier. 2-manifold reconstruction from sparse visual features. In *International Conference on 3D Imaging (IC3D)*, 2012.

[3] S. Yu and M. Lhuillier. Surface reconstruction of scenes using a catadiopric camera. In *Conference on Computer Vision/Computer Graphics Collaboration Techniques and Applications (MIRAGE) (also in LNCS, volume 6930)*, 2011.

[4] S. Yu and M. Lhuillier. Incremental reconstruction of manifold surface from sparse visual mapping. In *International Conference on 3D Imaging, Modeling, Processing, Visualization, and Transmission (3DimPVT)*, 2012.

[5] S. Yu and M. Lhuillier. Genus refinement of manifold surface reconstructed by sculpting the 3d-delaunay triangulation of structure-from-motion points. In *International Conference on Pattern Recognition (ICPR)*, 2012.

- National Conferences

[6] S. Yu and M. Lhuillier. Modèles 3d à partir d'un nuage épars de points et une caméra catadioptrique. In *Actes de la conférence de Reconnaissance des Formes et d'Intelligence Artificielle (RFIA)*, 2012.

E. PROOFS OF COMPLEXITIES

Bibliography

[7] CGAL, The Computational Geometry Algorithm Library, www.cgal.org.

[8] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C.T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003.

[9] R. Allègre, R. Chaine, and S. Akkouche. A streaming algorithm for surface reconstruction. In *Eurographics Symposium on Geometry Processing*, pages 79–88, 2007.

[10] P. Alliez, D. Cohen-Steiner, Y. Tong, and M. Desbrun. Voronoi-based variational reconstruction of unoriented point sets. In *ACM International Conference Proceeding Series*, volume 257, pages 39–48, 2007.

[11] L. Alvarez, R. Deriche, J. Sanchez, and J. Weickert. Dense disparity map estimation respecting image discontinuities: A PDE and scale-space based approach. *Journal of Visual Communication and Image Representation*, 13(1):3–21, 2002.

[12] N. Amenta and M. Bern. Surface reconstruction by voronoi filtering. *Discrete Computational Geometry*, 22(4):481–504, 1999.

[13] N. Amenta and R.K. Kolluri. Accurate and efficient unions of balls. In *Proceedings of the sixteenth annual Symposium on Computational Geometry*, pages 119–128, 2000.

[14] N. Amenta, S. Choi, T.K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *The sixteenth annual symposium on Computational Geometry*, pages 213–222, 2000.

[15] N. Amenta, S. Choi, and R.K. Kolluri. The power crust. In *ACM symposium on Solid modeling and applications*, pages 249–266, 2001.

[16] C.L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3d scans. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 109–118, 1995.

[17] J. Bergen, P. Anandan, K. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *European Conference on Computer Vision*, pages 237–252, 1992.

[18] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *Computer and System Sciences*, 48(3):384–409, 1994.

[19] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[20] P.J. Besl and N.D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[21] S. Birchfield and C. Tomasi. Multiway cut for stereo and motion with slanted surfaces. In *International Conference on Computer Vision*, volume 1, pages 489–495, 1999.

[22] A.F. Bobick and S.S. Intille. Large occlusion stereo. *International Journal of Computer Vision*, 33(3):181–200, 1999.

[23] J.D. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, 1984.

[24] J.D. Boissonnat and F. Cazals. Smooth surface reconstruction via natural neighbour interpolation of distance functions. In *Proceedings of the sixteenth annual symposium on Computational geometry*, pages 223–232, 2000.

[25] J.D. Boissonnat and S. Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67(5):405–451, 2005.

[26] J.D. Boissonnat, O. Devillers, S. Hornus, et al. Incremental construction of the delaunay graph in medium dimension. In *Annual Symposium on Computational Geometry*, pages 208–216, 2009.

[27] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Levy. *Polygon Mesh Processing*. AK Peters, 2010.

[28] A. Broadhurst, T.W. Drummond, and R. Cipolla. A probabilistic framework for space carving. In *International Conference on Computer Vision*, volume 1, pages 388–393, 2001.

[29] M.Z. Brown, D. Burschka, and G.D. Hager. Advances in computational stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):993–1008, 2003.

[30] J.C. Carr, R.K. Beatson, J.B. Cherrie, T.J. Mitchell, W.R. Fright, B.C. McCallum, and T.R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76, 2001.

[31] F. Cazals and J. Giesen. Delaunay triangulation based surface reconstruction: ideas and algorithms, 2004. INRIA technical report 5394.

[32] R. Chaine. A geometric convection approach of 3-d reconstruction. In *Eurographics Symposium on Geometry Processing*, pages 218–229, 2003.

[33] Y. Chen and G. Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, 1992.

[34] R.T. Collins. A space-sweep approach to true multi-image matching. In *Computer Vision and Pattern Recognition*, pages 358–363. IEEE, 1996.

[35] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.

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

[37] L. De Floriani, P. Magillo, and E. Puppo. Managing the level of detail in 3d shape reconstruction and representation. In *International Conference on Pattern Recognition*, volume 1, pages 389–391, 1998.

[38] O. Devillers. The delaunay hierarchy. *International Journal of Foundations of Computer Science*, 13(02):163–180, 2002.

[39] T.K. Dey. *Curve and surface reconstruction: algorithms with mathematical analysis*, volume 23. Cambridge University Press, 2006.

[40] T.K. Dey and S. Goswami. Provable surface reconstruction from noisy samples. *Annual Symposium on Computational Geometry*, 8(11):330–339, 2004.

[41] U.R. Dhond and J.K. Aggarwal. Structure from stereo-a review. *IEEE Transactions on Systems, Man and Cybernetics*, 19(6):1489–1510, 1989.

[42] P. Doubek and T. Svoboda. Reliable 3d reconstruction from a few catadioptric images. In *Third Workshop on Omnidirectional Vision*, pages 71–78, 2002.

[43] P. Doubek, I. Geys, and L. Van Gool. Cinematographic rules applied to a camera network. In *The fifth Workshop on Omnidirectional Vision, Camera Networks and Non-Classical Cameras*, 2004.

[44] H. Edelsbrunner. *Geometry and topology for mesh generation.* Cambridge University Press, 2001.

[45] H. Edelsbrunner and E.P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.

[46] D.W. Eggert, A. Lorusso, and R.B. Fisher. Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*, 9(5):272–290, 1997.

[47] O. Faugeras. *Three-dimensional computer vision*. MIT press, 1993.

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

[49] O. Faugeras, E. Le Bras-Mehlman, and J.D. Boissonnat. Representing stereo data with the delaunay triangulation. In *Artificial Intelligence*, pages 41–47, 1990. also in INRIA technical report 788.

[50] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[51] J.S. Franco and E. Boyer. Efficient polyhedral modeling from silhouettes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(3):414–427, 2009.

[52] P. Fua and Y.G. Leclerc. Object-centered surface reconstruction: Combining multi-image stereo and shading. *International Journal of Computer Vision*, 16(1):35–56, 1995.

[53] S. Fuhrmann and M. Goesele. Fusion of depth maps with multiple scales. *ACM Transactions on Graphics (TOG)*, 30(6):148, 2011.

[54] C. Geyer and K. Daniilidis. A unifying theory for central panoramic systems and practical implications. In *European Conference on Computer Vision*, pages 445–461, 2000.

[55] P. Giblin. *Graphs, surfaces and homology*. Cambridge University Press, 2010.

[56] D. Glickenstein. Chapter 1: Smooth manifolds, 2010. URL `http://math.arizona.edu/~glickenstein/math534_1011/Chap1.pdf`. Lecture Notes.

[57] N. Gonçalves and H. Araújo. Estimating parameters of noncentral catadioptric systems using bundle adjustment. *Computer Vision and Image Understanding*, 113(1):11–28, 2009.

[58] J. Goodman and J. O'Rourke. *Handbook of Discrete and Computational Geometry, Second Edition*. CRC Press, 2004.

[59] M. Gopi, S. Krishnan, and C.T. Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. *Computer Graphics Forum*, 19(3):467–478, 2000.

[60] A. Guéziec, G. Taubin, F. Lazarus, and B. Horn. Cutting and Stitching: Converting Sets of Polygons to Manifold Surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):136–151, 2001.

[61] B.M. Haralick, C.N. Lee, K. Ottenberg, and M. Nölle. Review and analysis of solutions of the three point perspective pose estimation problem. *International Journal of Computer Vision*, 13(3):331–356, 1994.

[62] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50, 1988.

[63] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[64] C. Hernández Esteban and F. Schmitt. Silhouette and stereo fusion for 3d object modeling. *Computer Vision and Image Understanding*, 96(3):367–392, 2004.

[65] A. Hilton. Scene modelling from sparse 3d data. In *Image and Vision Computing*, volume 23, pages 900–920, 2005.

[66] A. Hilton and J. Illingworth. Multi-resolution geometric fusion. In *International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, pages 181–188, 1997.

[67] A. Hilton, A. Stoddart, J. Illingworth, and T. Windeatt. On reliable surface reconstruction from multiple range images. In *European Conference on Computer Vision*, pages 117–126, 1996.

[68] C.M. Hoffmann. *Geometric and solid modeling: an introduction*. Morgan Kaufmann Publishers Inc., 1989.

[69] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Computer Graphics*, pages 71–78, 1992.

[70] A. Hornung and L. Kobbelt. Hierarchical volumetric multi-view stereo reconstruction of manifold surfaces based on dual graph embedding. In *Computer Vision and Pattern Recognition*, volume 1, pages 503–510, 2006.

[71] J. Isidro and S. Sclaroff. Stochastic refinement of the visual hull to satisfy photometric and silhouette consistency constraints. In *International Conference on Computer Vision*, pages 1335–1342, 2003.

[72] H. Jin, S. Soatto, and A.J. Yezzi. Multi-view stereo reconstruction of dense shape and complex appearance. *International Journal of Computer Vision*, 63(3):175–189, 2005.

[73] T. Kanade and M. Okutomi. A stereo matching algorithm with an adaptive window: Theory and experiment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):920–932, 1994.

[74] S.B. Kang, R. Szeliski, and J. Chai. Handling occlusions in dense multi-view stereo. In *Computer Vision and Pattern Recognition*, volume 1, pages I–103, 2001.

[75] M. Kazhdan. Reconstruction of solid models from oriented point sets. In *Proceedings of the third Eurographics symposium on Geometry processing*, page 73, 2005.

[76] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Eurographics Symposium on Geometry Processing*, pages 61–70, 2006.

[77] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 225–234, 2007.

[78] L. Kobbelt, S. Campagna, J. Vorsatz, and H.P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 105–114, 1998.

[79] R. Kolluri. Provably good moving least squares. *ACM Transactions on Algorithms (TALG)*, 4(2):18, 2008.

[80] R. Kolluri, J.R. Shewchuk, and J.F. O'Brien. Spectral surface reconstruction from noisy point clouds. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 11–21, 2004.

[81] K.N. Kutulakos and S.M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):199–218, 2000.

[82] 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 *International Conference on Computer Vision*, pages 1–8, 2007.

[83] V. Lempitsky and D. Ivanov. Seamless mosaicing of image-based texture maps. In *Computer Vision and Pattern Recognition 2007*, pages 1–6, 2007.

[84] V. Lempitsky, Y. Boykov, and D. Ivanov. Oriented visibility for multiview reconstruction. In *European Conference on Computer Vision*, pages 226–238, 2006.

[85] D. Levin. Mesh-independent surface interpolation. In *Geometric modeling for scientific visualization*, volume 3, pages 37–49, 2003.

[86] M. Lhuillier. Effective and generic structure from motion using angular error. In *International Conference on Pattern Recognition*, volume 1, pages 67–70. IEEE, 2006.

[87] M. Lhuillier. Automatic scene structure and camera motion using a catadioptric system. *Computer Vision and Image Understanding*, 109(2):186–203, 2008.

[88] M. Lhuillier. Environment modeling from images taken by a low cost camera. In *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, (also in proceedings of PCV'10)*, volume XXXVIII part 3A, 2010.

[89] M. Lhuillier. A generic error model and its application to automatic 3d modeling of scenes using a catadioptric camera. In *International Journal of Computer Vision*, volume 91, pages 175–199, 2011.

[90] M. Lhuillier and L. Quan. Match propagation for image-based modeling and rendering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(8):1140–1146, 2002.

[91] M. Lhuillier and L. Quan. Surface reconstruction by integrating 3d and 2d data of multiple views. In *International Conference on Computer Vision*, pages 1313–1320, 2003.

[92] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM Siggraph Computer Graphics*, 21(4):163–169, 1987.

[93] D. Lovi, N. Birkbeck, D. Cobzas, and M. Jagersand. Incremental free-space carving for real-time 3d reconstruction. In *3D Data Processing, Visualization and Transmission*, 2010.

[94] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[95] B.M. Maggs. A maximum likelihood stereo algorithm. *Computer Vision and Image Understanding*, 63(3):542–567, 1996.

[96] A. Manessis, A. Hilton, P. Palmer, P. McLauchlan, and X. Shen. Reconstruction of scene models from sparse 3d structure. In *Computer Vision and Pattern Recognition*, volume 2, page 2666, 2000.

[97] M. Meyer, M. Desbrun, P. Schroder, and A.H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics*, pages 35–57, 2003.

[98] B. Micusik and T. Pajdla. Structure from motion with wide circular field of view cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(7): 1135–1149, 2006.

[99] E.E. Moise. *Geometric Topology in Dimensions 2 and 3*. Springer Verlag, New York, 1997.

[100] D.D. Morris and T. Kanade. Image-consistent surface triangulation. In *Computer Vision and Pattern Recognition*, volume 1, pages 332–338, 2000.

[101] B.S. Morse, T.S. Yoo, P. Rheingans, D.T. Chen, and K.R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Proceedings of the 32nd annual conference on Computer graphics and interactive techniques*, page 78, 2005.

[102] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Real time localization and 3d reconstruction. In *Computer Vision and Pattern Recognition*, volume 1, pages 363–370, 2006.

[103] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Generic and real-time structure from motion. In *British Machine Vision Conference*, 2007.

[104] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Generic and real-time structure from motion using local bundle adjustment. *Image and Vision Computing*, 27(8):1178–1193, 2009.

[105] R.A. Newcombe and A.J. Davison. Live dense reconstruction with a single moving camera. In *Computer Vision and Pattern Recognition*, pages 1498–1505, 2010.

[106] R.A. Newcombe, S.J. Lovegrove, and A.J. Davison. Dtam: Dense tracking and mapping in real-time. In *International Conference on Computer Vision*, pages 2320–2327, 2011.

[107] D. Nistér. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770, 2004.

[108] Y. Ohtake, A. Belyaev, and H.P. Seidel. A multi-scale approach to 3d scattered data interpolation with compactly supported basis functions. In *Shape Modeling International*, pages 153–161, 2003.

[109] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.P. Seidel. Multi-level partition of unity implicits. In *Proceedings of the 32nd annual conference on Computer graphics and interactive techniques*, page 173, 2005.

[110] M. Okutomi and T. Kanade. A locally adaptive window for signal matching. *International Journal of Computer Vision*, 7(2):143–162, 1992.

[111] A.C. Öztireli, G. Guennebaud, and M. Gross. Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum*, 28(2):493–501, 2009.

[112] Q. Pan, G. Reitmayr, and T. Drummond. Proforma: Probabilistic feature-based on-line rapid model acquisition. In *British Machine Vision Conference*, 2009.

[113] R. Pito. Mesh integration based on co-measurements. In *International Conference on Image Processing*, volume 1, pages 397–400, 1996.

[114] 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. Sinha, B. Talton, L. Wang, Q. Yang, G. Welch, and H. Towles. Detailed real-time urban 3d reconstruction from video. *International Journal of Computer Vision*, 78(2):143–167, 2008.

[115] J.P. Pons, R. Keriven, and O. Faugeras. Multi-view stereo reconstruction and scene flow estimation with a global image-based matching score. *International Journal of Computer Vision*, 72(2):179–193, 2007.

[116] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge University Press, 2007.

[117] L. H. Quam. Hierarchical warp stereo. In *Image Understanding Workshop*, pages 149–155, 1984.

[118] S. Roy and I.J. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *International Conference on Computer Vision*, pages 492–499, 1998.

[119] N. Salman and M. Yvinec. Surface reconstruction from multi-view stereo. In *Asian Conference on Computer Vision*, 2009.

[120] V.V. Savchenko, A.A. Pasko, O.G. Okunev, and T.L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14(4):181–188, 1995.

[121] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1):7–42, 2002.

[122] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. In *International Journal of Computer Vision*, 2000.

[123] S.M. Seitz and C.R. Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35(2):151–173, 1999.

[124] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Computer Vision and Pattern Recognition*, volume 1, pages 519–528, 2006.

[125] J.S. Sereni. *Lecture 1 on Graph and Counting*. Dept. of applied math. of Charles University, Prague, 2009.

[126] C. Shen, J.F. O'Brien, and J.R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. *ACM Transactions on Graphics (TOG)*, 23(3):896–904, 2004.

[127] J.R. Shewchuk. Stabbing delaunay tetrahedralizations. *Discrete & Computational Geometry*, 32(3):339–343, 2004.

[128] Hayong Shin, Joon C. Park, Byoung K. Choi, Yun C. Chung, and Siyoul Rhee. Efficient Topology Construction from Triangle Soup. In *Geometric Modeling and Processing*, volume 0, page 359, 2004.

[129] G.G. Slabaugh, W.B. Culbertson, T. Malzbender, M.R. Stevens, and R.W. Schafer. Methods for volumetric reconstruction of visual scenes. *International Journal of Computer Vision*, 57(3):179–199, 2004.

[130] N. Snavely, S.M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. *ACM Transactions on Graphics (TOG)*, 25(3):835–846, 2006.

[131] M. Soucy and D. Laurendeau. A general surface approach to the integration of a set of range views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17 (4):344–358, 1995.

[132] C. Strecha, R. Fransens, and L. Van Gool. Wide-baseline stereo from multiple views: a probabilistic account. In *Computer Vision and Pattern Recognition*, volume 1, pages I–552, 2004.

[133] C. Strecha, W. Von Hansen, L. Van Gool, P. Fua, and U. Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[134] J. Stühmer, S. Gumhold, and D. Cremers. Real-time dense geometry from a handheld camera. In *Pattern Recognition*, pages 11–20. Springer, 2010.

[135] G. Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358, 1995.

[136] C.J. Taylor. Surface reconstruction from feature based stereo. In *International Conference on Computer Vision*, volume 1, page 184, 2003.

[137] I. Tobor, P. Reuter, C. Schlick, et al. Efficient reconstruction of large scattered geometric datasets using the partition of unity and radial basis functions. *Journal of WSCG*, 12(3):467–474, 2004.

[138] S. Tran and L. Davis. 3d surface reconstruction using graph cuts with surface constraints. In *European Conference on Computer Vision*, pages 219–231, 2006.

[139] A. Treuille, A. Hertzmann, and S. Seitz. Example-based stereo with general brdfs. In *European Conference on Computer Vision*, pages 457–469, 2004.

[140] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustmenta modern synthesis. In *Vision algorithms: theory and practice*, pages 153–177. Springer, 2000.

[141] E. Trucco and A. Verri. *Introductory techniques for 3-D computer vision*, volume 93. Prentice Hall Upper Saddle River NJ, 1998.

[142] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318, 1994.

[143] G. Turk and J.F. O'brien. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics (TOG)*, 21(4):855–873, 2002.

[144] A. Van Oosterom and J. Strackee. The solid angle of a plane triangle. In *IEEE Transactions on Biomedical Engineering*, number 2, pages 125–126, 1983.

[145] R.C. Veltkamp. Boundaries through scattered points of unknown density. *Graphical Models and Image Processing*, 57(6):441–452, 1995.

[146] G. Vogiatzis, P.H.S. Torr, and R. Cipolla. Multi-view stereo via volumetric graph-cuts. In *Computer Vision and Pattern Recognition*, volume 2, pages 391–398, 2005.

[147] G. Vogiatzis, C. Hernández, P.H.S. Torr, and R. Cipolla. Multiview stereo via volumetric graph-cuts and occlusion robust photo-consistency. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2241–2246, 2007.

[148] R. Yang and M. Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *Computer Vision and Pattern Recognition*, volume 1, pages I–211, 2003.

[149] C. Zach, T. Pock, and H. Bischof. A globally optimal algorithm for robust TV-L1 range image integration. In *International Conference on Computer Vision*, pages 1–8, 2007.

[150] H.K. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. In *IEEE Workshop on Variational and Level Set Methods in Computer Vision*, pages 194–201, 2001.

[151] G.M. Ziegler. Shelling polyhedral 3-balls and 4-polytopes. *Discrete & Computational Geometry*, 19(2):159–174, 1998.

[152] A.J. Zomorodian. *Topology for computing*, volume 16. Cambridge University Press, 2005.

Résumé

La modélisation 3d automatique d'un environnement à partir d'images est un sujet toujours d'actualité en vision par ordinateur. Ce problème se résout en général en trois temps: déplacer une caméra dans la scène pour prendre la séquence d'images, reconstruire la géométrie, et utiliser une méthode de stéréo dense pour obtenir une surface de la scène. La seconde étape met en correspondances des points d'intérêts dans les images puis estime simultanément les poses de la caméra et un nuage épars de points 3d de la scène correspondant aux points d'intérêts. La troisième étape utilise l'information sur l'ensemble des pixels pour reconstruire une surface de la scène, par exemple en estimant un nuage de points dense.

Ici nous proposons de traiter le problème en calculant directement une surface à partir du nuage épars de points et de son information de visibilité fournis par l'estimation de la géométrie. Les avantages sont des faibles complexités en temps et en espace, ce qui est utile par exemple pour obtenir des modèles compacts de grands environnements comme une ville.

Pour cela, nous présentons une méthode de reconstruction de surface du type sculpture dans une triangulation de Delaunay 3d des points reconstruits. L'information de visibilité est utilisée pour classer les tétraèdres en espace vide ou matière. Puis une surface est extraite de sorte à séparer au mieux ces tétraèdres à l'aide d'une méthode gloutonne et d'une minorité de points de Steiner. On impose sur la surface la contrainte de 2-variété pour permettre des traitements ultérieurs classiques tels que lissage, raffinement par optimisation de photo-consistance ... Cette méthode a ensuite été étendue au cas incrémental: à chaque nouvelle image clef sélectionnée dans une vidéo, de nouveaux points 3d et une nouvelle pose sont estimés, puis la surface est mise à jour. La complexité en temps est étudiée dans les deux cas (incrémental ou non).

Dans les expériences, nous utilisons une caméra catadioptrique bas coût et obtenons des modèles 3d texturés pour des environnements complets incluant bâtiments, sol, végétation... Un inconvénient de nos méthodes est que la reconstruction des éléments fins de la scène n'est pas correcte, par exemple les branches des arbres et les pylônes électriques.

Mots-clefs: Reconstruction de 2-variété, triangulation de Delaunay 3d, sommets de Steiner, analyse de complexité, nuage de points épars, Structure-from-Motion.

Abstract

The automatic 3d modeling of an environment using images is still an active topic in Computer Vision. Standard methods have three steps: moving a camera in the environment to take an image sequence, reconstructing the geometry of the environment, and applying a dense stereo method to obtain a surface model of the environment. In the second step, interest points are detected and matched in images, then camera poses and a sparse cloud of 3d points corresponding to the interest points are simultaneously estimated. In the third step, all pixels of images are used to reconstruct a surface of the environment, e.g. by estimating a dense cloud of 3d points.

Here we propose to generate a surface directly from the sparse point cloud and its visibility information provided by the geometry reconstruction step. The advantages are low time and space complexities; this is useful e.g. for obtaining compact models of large and complete environments like a city.

To do so, a surface reconstruction method by sculpting 3d Delaunay triangulation of the reconstructed points is proposed. The visibility information is used to classify the tetrahedra in free-space and matter. Then a surface is extracted thanks to a greedy method and a minority of Steiner points. The 2-manifold constraint is enforced on the surface to allow standard surface post-processing such as denoising, refinement by photo-consistency optimization ... This method is also extended to the incremental case: each time a new key-frame is selected in the input video, new 3d points and camera pose are estimated, then the reconstructed surface is updated. We study the time complexity in both cases (incremental or not).

In experiments, a low-cost catadioptric camera is used to generate textured 3d models for complete environments including buildings, ground, vegetation ... A drawback of our methods is that thin scene components cannot be correctly reconstructed, e.g. tree branches and electric posts.

Key-words: 2-Manifold Reconstruction, 3d Delaunay Triangulation, Steiner Vertices, Complexity Analysis, Sparse Point Cloud, Structure-from-Motion.