Incremental reconstruction of a complex scene using omnidirectional camera

Vadim Litvinov

N° d'ordre : D.U : 2541
EDSPIC : 686

Université Blaise Pascal - Clermont II

Ecole Doctorale
Sciences Pour L'Ingénieur De Clermont-Ferrand

Thèse

présentée par :

Vadim Litvinov

pour obtenir le grade de

Docteur d'Université

Spécialité : Vision pour la robotique

Reconstruction incrémentale d'une scène complexe à l'aide d'une caméra omnidirectionnelle

Incremental reconstruction of a complex scene using omnidirectional camera

---

Soutenue publiquement le 13 janvier 2015 devant le jury :

M.	Thierry	Chateau	Prof. Univ. B. Pascal	Président
Mme.	Sylvie	Treuillet	MCF Univ. d'Orléans	Rapporteur
M.	George	Vogiatzis	MCF Aston University	Rapporteur
M.	El Mustapha	Mouaddib	Prof. Univ. de Picardie J. Verne	Examinateur
M.	Marc	Daniel	Prof. Univ. d'Aix - Marseille	Examinateur
M.	Maxime	Lhuillier	CR CNRS Institut Pascal	Directeur de thèse

Acknowledgements

First of all, I would like to thank Sylvie Treuillet and George Vogiatzis for having reviewed this work as well as all the other jury members, namely Thierry Chateau, El Mustapha Mouaddib and Marc Daniel. Particularly, my sincere thanks goes to my supervisor, Maxime Lhuillier, without his continuous support I would never been able to get so far.

I would like to acknowledge Serge Alizon and François Marmoiton of the technical staff for their help with the data sets acquisitions and our secretary, Eliane De Dea, for all the paper work related to my travels. Their work, although necessary, is often forgotten.

Of cause, my thanks also goes to the other PhD students, namely Clément Deymier, Datta Ramadasan, Alexis Wilhelm, Pierre Bouges, Claude Aynaud, François Chadebeq and all the others as well as all the staffs of the Institut Pascal for have endured my presence during these long three years.

Finally, I would like to thank FEDER and Région Auvergne for funding.

Abstract

The automatic reconstruction of a scene surface from images taken by a moving camera is still an active research topic. This problem is usually solved in two steps: first estimate the camera poses and a sparse cloud of 3D points using *Structure-from-Motion*, then apply dense stereo to obtain the surface by estimating the depth for all pixels.

Compared to the previous approaches, ours accumulates the following properties. The output surface is a 2-manifold, which is useful for applications and post-processing. It is computed directly from the sparse point cloud provided by the first step, so as to avoid the second and time consuming step and to obtain a compact model of a complex scene. The computation is incremental to allow access to intermediary results during the processing.

The principle is the following. At each iteration, new 3D points are estimated and added to a 3D Delaunay triangulation; the tetrahedra are labeled as free-space or matter thanks to the visibility information provided by the first step. We also update a second partition of outside and inside tetrahedra whose boundary is the target 2-manifold. Under some assumptions, the time complexity of one iteration is bounded (there is only one previous method with the same properties, but its complexity is greater than that).

Our method is experimented on synthetic and real sequences, including a 2.5 km. long urban sequence taken by an omnidirectional camera. The surface quality is similar to that of the batch method which inspired us. However, the computations are not yet real-time on a commodity PC. We also study the use of contours in the reconstruction process.

Key words: Surface reconstruction, Geometry estimation from a set of images, Sculpture, Delaunay 3D, 2-Manifold.

Résumé

Un problème toujours d'actualité est la reconstruction automatique de la surface d'une scène à partir du flot d'images prises par une caméra en mouvement. Il se résout en général en deux étapes : le calcul de la géométrie où les poses de la caméra et un nuage épars de points 3D de la scène sont simultanément estimés, et un calcul de stéréo dense qui permet d'obtenir une surface en estimant la profondeur de tous les pixels.

L' approche que nous proposons se distingue des précédentes en cumulant les caractéristiques suivantes. La surface est une 2-variété, ce qui est utile pour les traitements ou utilisations ultérieurs. Elle est calculée directement à partir du nuage épars donné par la première étape, afin d'éviter la seconde étape coûteuse et pour obtenir une modélisation compacte d'une scène complexe. Le calcul est incrémental afin d'avoir un résultat pendant la lecture de la vidéo.

Le principe est le suivant. A chaque itération, de nouveaux points 3D sont estimés et insérés dans une triangulation de Delaunay 3D. Celle-ci partitionne l'espace en tétraèdres vides et pleins grâce à l'information de visibilité également fournie par la première étape. On met aussi à jour une seconde partition en tétraèdres intérieurs et extérieurs dont le bord est la 2-variété recherchée. Sous certaines hypothèses, et contrairement à la seule méthode précédente ayant les même propriétés et hypothèses, la complexité d'une itération est bornée.

Notre méthode a été expérimentée sur des séquences synthétiques et réelles, dont une séquence longue de $2{,}5$ km prise en milieu urbain avec une caméra omnidirectionnelle. La qualité du résultat est proche de celle obtenue par la méthode globale (non incrémentale) qui a servi d'inspiration, mais le temps de calcul ne permet pas actuellement une utilisation en-ligne sur un PC standard. On a aussi étudié l'intérêt d'ajouter des contours dans le processus de reconstruction.

Mots clés : Reconstruction de surface, Estimation de géométrie à partir d'images, Sculpture, Delaunay 3D, 2-Variété.

Contents

List of Figures

List of Tables

CHAPTER I

Introduction

Being able to calculate a 3D representation of a surface from a video sequence is a classical problem in the domain of Computer Vision. The majority of the current automatic methods proceed in two distinct steps. First, a *Structure-from-Motion* (*SfM*) algorithm is used to compute the cameras *poses* associated to each input image. Second, a *dense stereo* step is performed to compute the 3D points (or depths) associated to each pixel of the input images and a surface is computed from this dense 3D point cloud. However, *SfM* already provides a sparse cloud of points as a byproduct of *poses* computation. So, it would be interesting, from the computational point of view, to calculate the surface directly from it. The main objective of this dissertation is to continue the previous works in this direction.

So, the objective of this work is to develop an algorithm capable of reconstructing a realistic and long scenes, i.e. a method that takes a sequence of images as its input and produces a 3D surface that closely approximates the observed scene. We want this method to be *sparse*, i.e. to extract a necessary minimum of information from the input images. We want it to be *incremental*, i.e. we want to be able to access to the intermediary surfaces during the processing of the input video sequence. Last, we want the output surface to be *2-manifold*, i.e. being a suitable input for the majority of the surface processing algorithms.

Such a method has a lot of potential applications. First of all, a *sparse* method would have a reduced memory consumption and an *incremental* one would have it bounded, this will allow to reconstruct very large scale scenes. Another potential application would be the dynamic obstacle detection (for a mobile robot for instance), thanks to the *incremental* method ability to dynamically update the scene. Finally, such a method could be useful for correct occlusion handling in augmented reality.

We will begin by reviewing the motivations behind each of the three algorithm key properties (*sparse*, *incremental*, *2-manifold*) in section I.1 and so why such an algorithm would be interesting. Then, we will review the set of hypothesis that restrain this work to a reasonable scope in section I.2. Next, section I.3 will review the major and secondary contributions. Last, section I.4 will discuss the general structure of this dissertation.

I.1 Motivations

We want to develop a surface reconstruction method having three key properties at the same time: a method that is *sparse*, *incremental* and enforcing the topological constraints of the output surface (output is a *2-manifold*). In this section we explain why each one of these properties is interesting in practice.

I.1.1 Why a *2-manifold*?

Although triangulated manifold surfaces are widely used, it is useful to remind their definition and explain their importance. The importance of the *manifold* property is acknowledged in Computer Graphic and Computational Geometry, but Computer Vision works similar to ours [Hilton05, Labatut07, Pan09, Lovi10] have a tendency to ignore this property.

A surface is said *2-manifold* if the neighborhood of each surface point is topologically a disk [Botsch10]. A triangulated *manifold* without boundary is, in practice, represented by a list of triangles, each triangle shares each of its edges with exactly one another triangle of the list. Such a surface divides the space into two distinct regions: *inside* and *outside* (*inside* can be non connected).

Such a property is needed by the surface denoising and also required by other post-processing or refinements on the surface like *dense stereo* and surface fairing. Assuming that the true scene surface is a smooth *2-manifold*, the continuous differential operators of normal and curvature are well defined. If the triangle list which approximates the true surface is also a *2-manifold*, these operators can be extended to the discrete case [Meyer03, Botsch10]. More generally, a lot of Computer Graphic algorithms do not apply if the triangle list is not a *2-manifold* [Botsch10]. We also use the *manifold* property as a constraint to search the surface interpolating the point cloud (and reject some bad points).

I.1.2 Why a *sparse* method?

The majority of the surface reconstruction methods found in the literature are *dense*, i.e. they use the totality of the pixels of the input images. This kind of methods have the advantage to provide good results, i.e. the reconstructions are both visually appealing and detailed. Nevertheless, a *sparse* approach, i.e. a method that only reconstructs in 3D the "interesting" parts of the image would have several advantages.

The first and the most obvious advantage is the run time improvement. The *dense* approaches are usually computationally expensive, because they need to perform a lot of photo-consistency calculations across a lot of views and for many depth hypotheses. *Sparse* approaches on the other hand only work with a limited number of entries and so perform less 3D computations. When compared on a standard small scale multi-view data set [Seitz06, Mid], *dense* CPU based approaches taking hours when a *sparse* one takes seconds [Yu13]. So, a *sparse* approach would be much more convenient for the reconstruction of large scale scenes.

The second advantage of a *sparse* algorithm over a *dense* one is the memory usage. In fact, *dense* methods usually perform a regular subdivision of space. This is OK if the observed scene is relatively small but quickly become cumbersome when the size of the scene grows. Moreover, a regular subdivision of space have another

troublesome consequence: even the flat portions of the output surface are subdivided in many triangles. So a *sparse* method would not only consume less memory during the reconstruction process, but also produces a lighter and more compact 3D model. Once again, this property made it more suitable for the large scale scenes reconstruction.

These two advantages makes a *sparse* method useful when the computation time and the memory usage efficiency is at least as important as the output surface quality. Particularly, we could cite large scale scenes reconstruction [Lhuillier13] and the reconstruction on the devices with limited resources [Pan11]. Moreover, such a method could be useful to initialize a more precise, but slower, *dense* reconstruction [Esteban04, Hiep09].

I.1.3 Why an *incremental* method?

A surface reconstruction method is said *incremental* if it reads the input images in order (from the oldest to the newest one) and provides the intermediary surfaces for each of these images. Such a method would have several advantages over a more classical *batch* approaches.

The first and the more obvious advantage is the ability to give access to the intermediary surfaces for each input image. This actually allows the algorithm to be used on-line [Lovi10] if the iterations are fast enough. This can be useful for applications such as mobile robot navigation and obstacle detection. Another potential application is the augmented reality, for example for the correct handling of the virtual object occlusions. Finally, the real time visualization of the partially reconstructed surface can allow the user to select the next image taking point in a way to better retrieve the lacking details [Pan09].

A less obvious advantage lies in the domain of memory consumption. In fact, for each input image, an *incremental* method would only modify a small portion of the surface. This locality of the modification would theoretically allow to discharge the biggest part of the data structures to a more slow but usually much bigger support (i.e. a hard drive). This low and constant memory consumption makes *incremental* methods highly useful when reconstructing very large scale scenes.

I.2 Hypotheses

Reconstructing a 3D surface from a video sequence in a general context is a too ambitious problem. To ensure that such a task is feasible, this work is restrained by a set of hypotheses. These hypotheses are:

1. The observed scene is rigid;
2. The *intrinsic* calibration of the camera used to record the input sequence is known;
3. The camera used to record the input sequence is omnidirectional.

The first hypothesis ensures the feasibility of the computations. Although the algorithms reconstructing deformable surfaces exists, they need a set of prior knowledge about the observed surface. Of course, an observed scene is never 100% rigid, but mobile objects are intrinsically filtered by the *Structure-from-Motion* algorithm.

The knowledge of the *intrinsic* parameters (the second hypothesis) is a classical assumption. The geometry estimation can be done without this knowledge [Hartley04], but the computations would be less accurate and suffer from additional uncertainties.

Finally, the third hypothesis is here because our main objective is to reconstruct a large scale complex scene including all its components (ground, buildings, rooftops, etc.). Although it is possible with a classical camera, it would require more efforts to capture the entire environment.

I.3 Contributions

During the work on this dissertation, two major and several secondary contributions were produced. These contributions were presented during three international and one national conferences.

The major contributions are:

- A *sparse incremental* surface reconstruction method that guarantees that output surface is *2-manifold*. Contrary to the previous works [Yu12], the time of one iteration is bounded in practice even when the input trajectory contains one or several loops. This work was published in [Litvinov13] and [Litvinov14a].

- A new algorithm for artifacts removal step. It is almost as efficient as the previous one [Lhuillier13], but, at the same time, faster and easier to use in an incremental context. It was published in [Litvinov14b].

The secondary contributions are:

- The *batch sparse* 3D surface reconstruction algorithm from [Lhuillier13] is adapted to use a rigid multi-camera system instead of a single catadioptric camera.

- *Key frame* selection algorithm from [Mouragnon09] is improved by the introduction of an additional criterion.

- The influence of the input images resolution on the output surface quality is studied.

- An additional acute tetrahedra removal step is added to the *batch* and *incremental* versions of the algorithm. It enhances the quality of the output surface.

- The performance of the *batch sparse* 3D surface reconstruction method from [Lhuillier13] is evaluated using the standard multi-view data sets [Seitz06, Strecha08]. It was published in [Litvinov12].

- The performance of this method is evaluated when *curves* are added in addition to interest points. Also published in [Litvinov12].

- The overall execution speed is improved with the help of the parallel processing.

I.4 Structure of this dissertation

This dissertation is structured in seven chapters. The second chapter is the state of the art of the image based surface reconstruction. It quickly reviews the different types of the surface reconstruction methods found in the literature. The works closest to ours are reviewed in more details.

The next two chapters discuss the core of the subject: the *sparse incremental* surface reconstruction method. The third chapter details the mathematical modeling of the rigid multi-camera system and the *Structure-from-Motion* algorithm used to compute a 3D cloud of points from a series of images. The fourth chapter discusses the surface reconstruction algorithm. The algorithm is explained in details and its complexity is analyzed. In the fifth chapter the algorithm is experimented on the real and synthetic data sets.

The sixth chapter focuses on a particular step of the algorithm: the artifacts removal step. It is the slowest part of the algorithm, so finding a faster way to perform this step is interesting to accelerate the overall algorithm. Two new possible algorithms are presented as well as their complexities. Then, they are experimented on a real data set and their performances are compared to the old algorithm.

Finally, the seventh chapter studies what happens when the curves are added to the surface reconstruction pipeline. Different ways to use curves in the domain of surface reconstruction found in the literature are reviewed. Additional steps needed to integrate the curves into the reconstruction process are detailed. Finally, the performance of the algorithm with and without curves is evaluated using some standard multi-view data sets.

CHAPTER II

State of the art of image based surface reconstruction

Multi-view scene reconstruction is a vast topic of the computer vision and it was discussed in a lot of previous works. This chapter is an attempt to provide an overview of the literature on the subject. Of course, because of the great number of publications this overview can't be, and is not meant to be, exhaustive. Only the works close enough to the subject of this dissertation are discussed. Specifically, all the works that considers deformable or mobile objects are willingly omitted. Moreover, we consider only the publications using passive sensors exclusively, i.e. any kind of camera, but not a radar, lidar or similar technology.

Our main classification criterion will be the density of the image features used to reconstruct the final surface. Section II.1 considers the methods that use the totality of the pixels of the input image: the *dense* modeling methods. Sections II.2 and II.3 considers the *sparse* methods, i.e. the methods that use only a relatively small number of pixels.

Moreover, we subdivide the previous works in function of the way they access the input sequence. The methods that use all the images of the sequence at the same time are called *batch* methods and are discussed in subsections II.1.1 and II.1.3 and section II.2. The methods that access the input images in a chronological order are called *incremental* and are discussed in subsection II.1.4 and section II.3.

The reason why the sparse methods have two dedicated sections and the dense methods only one is because this dissertation main contribution is a new sparse method. So it seems appropriate to discuss the other sparse methods in greater detail.

Finally, the classification of the multi-view surface reconstruction methods proposed in this document is, of course, not the only one possible. The curious reader can refer himself to [Seitz06] for a more detailed discussion on the possible classification criteria.

II.1 Dense modeling methods

We begin our overview by the *dense* family of 3D surface modeling methods. For memory, a multi-view surface reconstruction method is called *dense* if it uses the totality or almost the totality of the pixels of the input images.

Usually, this kind of algorithms requires that the input images are fully calibrated, i.e. not only the camera *intrinsic* parameters, but also the relative transformations (rotations and translations) between images taking points are known. Finding this calibration is a very classical and well studied problem of the computer vision. There is a lot of algorithms to choose from, depending on the context of the problem. The reader can refer himself to [Hartley04] for an extensive overview.

The *batch dense* variation of the surface reconstruction algorithms is the most commonly found in the literature. We can separate these methods into two categories: those which use the input images directly (subsection II.1.1) and those which first reconstruct all the pixels (subsection II.1.2) and then builds the surface using the dense 3D point cloud (subsection II.1.3).

II.1.1 Batch methods using the input images directly

In the first place, let us consider the methods that use the input images directly. These approaches can be classified by the data structures they use during the reconstruction process. We find the methods using voxels, level sets and triangular meshes.

The **voxel** based methods [Seitz99, Kutulakos00, Broadhurst01, Slabaugh04, Treuille04] use a cartesian grid to represent a volume to model. The exact dimension of the volume must be either provided by the algorithm user or computed by some other mean. Each individual cell of this volume is called a *voxel* and the entire volume is called a *voxel-occupancy function*. To decide if a voxel is occupied or free, the majority of the methods [Kutulakos00, Seitz99, Slabaugh04, Treuille04] compute its photo-consistency score by back-projecting it to the input images. The color deviation of the pixels must be inferior to some user defined threshold. The values of *voxel-occupancy function* are computed in an order to prevent the occlusion problems. Finally, the computed grid can be converted to a triangular mesh using an algorithm such as "marching cubes" [Lorensen87].

The other way to model the scene to reconstruct is to use a **level set** of a scalar function $f(\mathbf{x})$ as in [Faugeras98, Lhuillier03, Jin05, Pons07]. The surface to model is an *implicit* surface at $f(\mathbf{x}) = 0$. In practice, $f(\mathbf{x})$ is sampled on a cartesian grid. The basic idea is to initialize $f(\mathbf{x})$ to some initial guess and then to evolve it to maximize the photo-consistency of the surface with the input images, this is why these methods are called *variational*. For example, [Faugeras98] defines the speed of evolution of a point $\mathbf{x}$ along its normal to the surface based on the similarity between the back-projections of $\mathbf{x}$ to the input images. The final surface mesh can, again, be computed by an algorithm such as [Lorensen87]. The main interest of this kind of approaches is the fact that the surface topology can change during workflow without any side effects.

The same basic principles as in the *voxel* and *level set* based methods can also be directly applied to a triangular **mesh**. For memory, a *triangular mesh* is a set of vertices connected by edges and triangular faces and it is a most common and efficient way to store and process 3D models. The methods [Vogiatzis05, Hornung06, Tran06, Vogiatzis07] represents the cartesian grid of the volume to model as a

graph. The nodes of the graph are the voxels of the grid and the edges are the faces separating two voxels. The edges are weighted by a cost function. For example, in [Vogiatzis05] the cost of an edge is a mean value of the photo-consistency of the two connected voxels. The final mesh is directly computed as a minimal cut (minimal cost) of the graph. On the other hand, the methods [Fua95, Isidoro03, Esteban04, Franco09] are variational approaches that deforms the *mesh* directly by trying to minimize a cost function. For example, in [Fua95] the cost function is a weighted sum of three terms: a smoothing term, a term of intensity correlation over a series of input images and a smoothing based on albedo coefficient of adjacent triangles (ratio of reflected to incident light).

II.1.2 Dense point cloud generation

Instead of using the input images directly, another large family of *dense* surface reconstruction methods works with a dense cloud of 3D points. Therefore, these algorithms can be separated into two well distinct steps: dense point cloud generation and surface reconstruction. In this subsection we will overview the methods to generate the cloud of points.

To generate a cloud of 3D points from a series of images, a classic way to proceed is to compute a *disparity map* for each pair of images. A *disparity map* is a vector field that associates a 2D *disparity vector* to each pixel of the first image. A *disparity vector* represents a displacement between the considered pixel and the corresponding pixel in the second image.

So the problem is to associate the pixels between two images. For a vast majority of dense matching algorithms, the first step is to rectify the input images for the *disparity vectors* to become 1D. Notable exception to this rule are [Collins96, Lhuillier02, Yang03]. During the rectification step, the images are deformed in a way for *epipolar lines* to become horizontal. For memory, an *epipolar line* is an intersection of plane containing the two cameras centers and the image plane. This way, the pixel of the second image associated to the pixel of the first image lie somewhere on the corresponding line and not anywhere in the image. It can now be found either using the similarity (Zero Norm Cross-Correlation (*ZNCC*)) or dissimilarity (Sum of Squared Difference (*SSD*)) applied to a window of pixels.

In practice, performing a dense matching using only similarity comparison is insufficient because of a great number of false positive matches. So, another set of constraints should be added to the problem such as uniqueness, disparity continuity, matches ordering on the *epipolar line*, etc. ([Dhond89]) So the pixel matching becomes a cost function optimization problem. The cost function can be minimized locally for each pixel as in [Kanade94, Bobick99]. Another approach is to minimize the global energy function which combines the constraints of individual pixels as in [Roy98, Brown03].

Once the *disparity maps* have been computed, we could compute a *depth map* for each input image [Faugeras93]. A *depth map* (or Z-buffer) associates a depth information to each pixel. The depth of a pixel is the euclidian distance between the camera center and the corresponding 3D point. This value can be computed thanks to the cameras calibration information and matching.

Some 3D surface reconstruction methods can use the *depth maps* directly, but this data structure is not well suited for this problem, fortunately it can easy be converted to *organized* or *unorganized* point clouds. An *organized* point cloud is a 3D point cloud where the connections between the points are known. Because we

know the positions of the cameras centers and the distances between the centers and the observed points for each pixel, the 3D coordinates for each point are easily computed. Some surface reconstructions algorithms use the additional information provided by the *organized* point cloud to enhance the modeling process.

An *unorganized* point cloud is a cloud where each point is independent. It can be computed by *registering* the *depth maps* into the same coordinate system. This process is usually performed using the ICP algorithm [Besl92]. Let call P_i the point cloud computed from one *depth map* I_i. The transformation (rotation and translation) between two point clouds P_i and P_j is iteratively refined by minimizing a cost function. The cost function is a sum of squared distances between the same points in the different point clouds.

II.1.3 Batch methods using the dense point cloud

The generation of dense cloud of 3D points has been discussed in the previous subsection, so now we proceed to a brief overview of surface reconstruction methods that uses this cloud of points as their input. This kind of algorithms is quite frequently found in the literature. For the sake of clarity, the input dense cloud of points will be called P in the remaining of this subsection.

3D Delaunay triangulation based methods

The first type of methods are those which use the 3D Delaunay triangulations and Voronoi diagrams. The Delaunay triangulation and its properties will be discussed in details in the chapter IV, but for memory, a Delaunay triangulation T is a triangulation of a set P of 3D points such as for any tetrahedron $\Delta \in T$, its circumscribing sphere doesn't contain any points of P. The Voronoi diagram is the dual of the Delaunay triangulation. More precisely, the Voronoi diagram V of P is a cellular complex having several properties. The first property is for each cell $c \in V$ it exists one and only one point $\mathbf{p} \in P$ such as $\mathbf{p} \in c$. The second is for each point $\mathbf{x} \in c$ and for each point $\mathbf{p}' \in P \setminus \{\mathbf{p}\}$, $||\mathbf{p} - \mathbf{x}|| < ||\mathbf{p}' - \mathbf{x}||$, i.e. the distance between $\mathbf{x}$ and $\mathbf{p}$ is smaller than the distance between $\mathbf{x}$ and any other point of P. The Delaunay triangulation is frequently used in the field of surface reconstruction because it can be proven that it contains a "good" approximation of the surface ([Cazals04]).

The methods [Amenta99, Amenta00a] reconstruct the surface using the properties of **the Voronoi diagrams**. For a point $\mathbf{p} \in P$, we call the poles $\mathbf{p}^+$ and $\mathbf{p}^-$ of $\mathbf{p}$ the two furthest vertices of the Voronoi cell of $\mathbf{p}$. The poles of a Voronoi diagram are usually far from the surface to reconstruct, so the algorithms proceed by computing a Delaunay triangulation of P and the poles of the Voronoi diagram of P, then by eliminating the triangles which poles as vertices. Finally they eliminate the irregular triangles. A triangle is irregular if the angles between its normal and vectors from vertices to poles are large. Finally, a set of triangles such that each edge has at last two adjacent triangles is constructed and the final surface is grown inside this set. The method [Amenta00a] is a simplification of [Amenta99] which works directly with a Delaunay triangulation of P.

Another interesting property of Delaunay triangulation that can be used to reconstruct a surface is the property of Delaunay circumspheres or **polar balls** (because they are centered at the poles). The union of these balls can be used to approximate the surface to compute. This property is exploited by [Amenta01, Dey04, Kolluri04]. For example, the method [Amenta01] uses a *power diagram*.

This diagram partitions the space into polyhedral cells, one cell per polar ball. Each cell contains the points such as their distance to the corresponding polar ball is smaller than the distance to other polar balls. The cells are sorted into two categories: *inner* and *outer*, by using the heuristic from [Amenta00b]. The resulting surface is the frontier between these two categories.

Finally, a last big family of methods based on Delaunay triangulation are **sculpting methods** [Boissonnat84, Edelsbrunner94, Bajaj95, Veltkamp95, Floriani98]. The basic idea of this algorithms is to perform a Delaunay triangulation T of the point cloud P and then to remove the tetrahedra from it, thus sculpting the triangulation. The first paper to discuss this approach was [Boissonnat84]. In this article, the tetrahedra were removed from T one-by-one by an order of priority until all the points of P appear on the exterior surface (boundary) of the triangulation. The priority criterion was the maximum of distances between the faces of the tetrahedron and the circumscribing sphere of this tetrahedron. An interesting property of this method is the fact that the resulting surface is 2-manifold, but it's genus can only be zero. This problem was resolved by [Floriani98] by removing tetrahedra by packs instead of one-by-one. More recent *sculpting* methods can also use α-shapes [Bajaj95, Edelsbrunner94], geometric convection and the Gabriel property [Chaine03] or graph cut energy optimization [Labatut09, Jancosek11]. The methods [Boissonnat84] and [Amenta00a] can also be used together for better results as in [Gezahegne05].

Triangulated mesh based methods

The second type of point cloud based methods are those which form the **triangulated mesh** directly by connecting the neighboring points. They are very few and usually require that the input point cloud is sufficiently dense and uniform. In this family, we could find the BPA (Ball-Pivoting Algorithm) [Bernardini99] which uses a fixed size ball pivoting around the edges of the triangles. Each time a ball touches a point of P, a new triangle is formed by the point and the edge around which the ball was pivoting. Another work of interest is [Gopi00]. It begins by estimating a tangent plane for each point $\mathbf{p} \in P$ and by performing a 2D Delaunay triangulation of projections of its neighbors to this tangent plane. Then, the triangles in 3D are formed between each set of three points if they are neighbors in one of the 2D triangulations.

Implicit function based methods

The third category of point cloud based methods are those which model the surface to reconstruct by an **implicit function** $f : \mathbb{R}^3 \rightarrow \mathbb{R}$. Usually, the function f is constrained in such a way that $\forall \mathbf{p} \in P, f(\mathbf{p}) = 0$ or at last $f(\mathbf{p}) \simeq 0$. This kind of algorithms can further be separated into global and local approaches.

In the global approaches, the function f is a sum of global functions. This can be a signed distance functions as in [Hoppe92, Boissonnat00a, Alliez07]. For example, [Hoppe92] used the distance between the considered point and the tangent plane of the nearest point of P. Or the function f can be a sum of radial basis functions (*RBFs*) as in [Savchenko95, Carr01, Turk02]. A *RBF* is a function whose value depend only on a distance from some well defined point.

In the local approaches, the function f is defined locally in some restrained space and then the global function is a blend of these local functions. Notable

methods [Ohtake03, Morse05] are using compactly supported *RBFs*. A compactly supported *RBF* is a *RBF* defined in the neighborhood of its center and is zero elsewhere. Another notable methods are locally fitting methods [Tobor04, Ohtake05]. They partition the space into a set of subspaces and, in each subspace, they find a function that fits the points of P in this subspace.

Another category of methods using an implicit function to model the surface to reconstruct are Moving Least Squares [Alexa03, Levin04, Kolluri08, Oztireli09]. They are based on the iterative solution of a least squares problem. For example, [Levin04] optimizes the errors between the points of P and their projections on the polynomial locally approximating the surface to find.

Finally, instead of using implicit function such as $\forall \mathbf{p} \in P, f(\mathbf{p}) = 0$, some methods model the surface to reconstruct using an **indicator function** such as for any point $\mathbf{x}$ in space $f(\mathbf{x}) = 1$ if the point $\mathbf{x}$ is inside an object and $f(\mathbf{x}) = 0$ otherwise. Most notable methods are Poisson reconstruction methods [Kazhdan05, Kazhdan06]. They take an oriented point cloud as their input, i.e. for each $\mathbf{p} \in P$ its normal vector $\mathbf{n}_p$ is known. So the basic idea of this methods is to compute a vector field $\overrightarrow{V}$ by applying a Gaussian blur to the normals of the input point cloud. Then to compute a f unction f such as the gradient of f fits the vector field $\overrightarrow{V}$ by resolving a Poisson equation.

Organized point cloud based methods

Lastly, the fourth category of point cloud based surface reconstruction methods are the methods which work with an **organized point cloud** directly. These algorithms can be further subdivided into the "*fusion*" methods and the methods using implicit surfaces. The first kind of methods [Chen92, Turk94, Soucy95, Pito96, Mičušík09] begins by reconstructing the *mesh* for each individual *range image*, then they merge all the *meshes* to form the final surface, hence the name. The methods [Curless96, Hilton96, Hilton97, Zach07, Fuhrmann11] of the other category fit an implicit function to all of depth maps and then they extract an isosurface to obtain the final result.

II.1.4 Incremental methods

In this final subsection of the discussion about *dense* surface reconstruction methods, we will discuss about the reconstruction methods working *incrementally*. The definition and advantages of the *incremental* methods was discussed in details in the chapter I, but for memory the *incremental* algorithms have two main advantages over their *batch* counterparts: possibility of on-line reconstruction and the constant memory consumption. On-line reconstruction is possible because the incremental methods enhance a previously computed surface with the information from the newly acquired image and so, if the computation is fast enough, the reconstruction can be performed real-time. The constant memory consumption is allowed by the fact that incremental algorithm usually work with a very limited and well defined portion of the overall data structure. This property allows to work with the very large scale sequences.

Incremental surface reconstruction methods using the totality or almost the totality of the input image pixels (*dense* methods) are scarce a in the literature. The primary reason is the fact that *dense* methods are usually very computation expensive and so are difficult to implement under a real-time constraint without the us-

age of a special hardware such as the GPU. Nevertheless, there are some methods [Allègre07, Pollefeys08, Newcombe10] that were developed recently and so in this subsection we will proceed to their brief overview.

The method described in **[Allègre07]** is a 3D Delaunay triangulation based dense and incremental surface reconstruction method. It uses the same basic ideas and principles as the *geometry convection* algorithm described in [Chaine03]. The basic idea of the method can be separated into two steps. The first step consists in computing a 3D Delaunay triangulation of a dense input point cloud P. The second step is to initialize a surface S as the convex hull of the triangulation and then to iteratively remove the facets (or the tetrahedra) to shrink S until it closely fit the surface to find.

On the first glance, one may think that this method can easily be converted to an incremental scheme by simply adding the new 3D points issued from a newly added input image into the triangulation and then to perform the convection on the newly added tetrahedra. Unfortunately, addition of the new points into the triangulation can not only add the new tetrahedra but can also modify the existing ones. The solution to this problem proposed by [Allègre07] is to classify the input points into slices along an arbitrary selected axis. The points are classified in such a way that the addition of slice σ_t into the triangulation will not modify the tetrahedra issued from addition of points of slices $\sigma_i, i \leq t-3$. Therefore, it is easy to identify the part of triangulation in which the convection of the surface S should be performed anew.

This algorithm is fast and can be used with a very large input data sets using only a limited amount of memory. But the usage of axis aligned slices is difficult in practice. Firstly because it is difficult to sort the input points into slices without pre-calculating all of them. Secondly because, for the method to be effective, the surface to estimate must be significantly longer along one axis than the others.

Another incremental dense surface reconstruction algorithm was described in **[Pollefeys08]**. It is real-time and suitable for modeling of large scale scenes, but it needs a very costly hardware to function properly (*GPS/INS* is not strictly mandatory, but highly recommended) and performs a part of the computations on the *GPU* to be useable in real-time.

The method can be separated into three distinct stages: cameras poses estimation, preliminary estimation of the 3D planes in the observed scene using a sparse point cloud and finally the final dense reconstruction. The camera poses are estimated by performing a fusion of the information provided by *GPS/INS* and the estimation provided by the *Structure-from-Motion* (*SfM*) algorithm from the visual data by a Kalman filter. If the *GPS/INS* information is not available only the *SfM* is used. Secondly, the sparse features tracked by the cameras (and also used by the *SfM*) are reconstructed and used to estimate the position and normals of the 3D planes in the observed scene. Next, a *plane sweeping* dense reconstruction guided by the information from the previous step is used to compute a *depth map* for each input image. Finally the *depth maps* of the consecutive images are merged to estimate the final surface.

The advantages of this method is its ability to successfully model a very large scale scenes and the fact that the computation can be performed in real-time. It has, however, several drawbacks. First is the fact that the overall system is costly. The second is the fact that the output surface is not a manifold (i.e. the triangles are not always connected) and so it is difficult to use as input for the usual surface processing algorithms.

Finally, the most recent dense and incremental surface reconstruction method the author of this work is aware of is the method described in **[Newcombe10]**. It is somewhat similar in spirit to the algorithm in [Pollefeys08] in the sense that it uses a sparse point cloud produced by *SfM* to guide the dense reconstruction.

The algorithm consists in several steps. First of all the camera poses and a sparse cloud of points is estimated using an incremental *Structure-from-Motion*. Then the sparse point cloud is used to estimate a rough model of the observed surface by using a *RBFs* based implicit surface reconstruction. Then the algorithm automatically selects a reference camera C_{ref} and a bunch of cameras around it in such a way that the fields of view of these cameras overlaps. Next the algorithm uses a optical flow based method to compute a *depth map* for the camera C_{ref} by deforming the rough surface so it become photo-consistent with all the images of the bunch of cameras. Finally, the resulting partial surface is merged into the global model. Of course, in practice, all this steps are performed in parallel.

The method provides excellent results in term of surface quality, but it suffers from several problems. First of all, the method is really slow. Partial surface generation takes more than a second using several *GPUs* to perform the computation. The second drawback is although the resulting surface is not a *triangle soup*, it is not guaranteed to be a 2-manifold.

II.2 Batch sparse modeling methods

In this section, we will begin the discussion of another family of the 3D surface modeling methods: the *sparse* methods. More precisely, a *sparse* method is a method which reconstruct the surface using only a small fraction of the pixels of the input images (usually only around 0.5%). The advantages and inconveniences of the *sparse* methods compared to the *dense* ones was discussed in details in the chapter I, but for memory, the *sparse* methods are usually much faster and have a much lesser memory footprint that their *dense* counterparts because they works with a smaller amount of data. Moreover, the reconstructed features are usually more precise and the methods could be applied even for a poorly textured environments. Finally, the reconstructed model is relatively light (a small number of triangles). The disadvantage of the *sparse* modeling is the output surface quality that tends to be lower than that of the *dense* methods due to the lack of points.

The *sparse* methods are reconstructing the surface from a sparse cloud of 3D points (called P for convenience in the remaining of this section) that firstly need to be computed. For that, we begin by detecting some sparse features in the input images. The most popular detector is [Harris88]. Then the features are tracked and reconstructed by an algorithm that is usually called *Structure-from-Motion* such as [Lhuillier08, Mouragnon09]. This process will be described in more details in the chapter III.

An easy approach would be to attempt to reconstruct the surface from this *sparse* cloud of points by using the same algorithms as those used in the *dense* case. For example, the methods [Boissonnat84, Veltkamp95, Floriani98] could be used. Unfortunately, the quality of the output will be poor for any complex scene. The methods specifically developed to work with *sparse* clouds usually use the visibility information in addition to the 3D coordinates of each point. Specifically, we call C the sets of the camera poses that observed the scene and we call R the set of rays (line segments). If the camera $\mathbf{c} \in C$ have observed the point $\mathbf{p} \in P$, the ray (line

segment)$r \in R$ from **c** to **p** exists. So we could help the reconstruction process by using the heuristic that the space crossed be a ray r is empty.

As for the *dense* methods, the *sparse* methods can be separated into *batch* and *incremental*. In this section, we will discuss specifically the *batch* algorithms, i.e. the algorithms that need access to the totality of the input images during the reconstruction process. These methods can themselves be separated into the methods based on multiple 2D Delaunay triangulations in the input images (subsection II.2.1) and the methods using the 3D Delaunay triangulation of the *sparse* cloud of points (subsection II.2.2).

II.2.1 2D Delaunay triangulation based methods

We begin our discussion of the *batch sparse* surface reconstruction methods by the algorithms based on the 2D Delaunay triangulation of the features in the input images. The basic idea is to back-project the features in space and so to obtain an initial estimation of the output surface by a connected set of triangles taken from the triangulation in the image, i.e. two features connected in the image are also connected in 3D space.

[Morrisoo] use this idea. The main contribution of this publication is a surface optimization algorithm. The user is supposed to select and track the features in the input images by hand. Then a Structure-from-Motion algorithm is used to find the 3D position of tracked points as well as the position of the cameras.

Next, an initial estimation of the output surface is computed. For that purpose, a 2D Delaunay triangulation of the tracked points is computed in one of the input images (called triangulation image) and then back-projected in the 3D space to obtain the estimation. For a more complex objects, the back-projected triangulation from several images can be merged. Then an image different from the triangulation image is chosen and called the reference image. An estimated image is computed for the estimated surface with the help of the camera calibration and position of the reference image. The estimated image is compared with the real one and the result of this comparison defines a score of the triangulation.

The goal of the algorithm is to find the 2D triangulation of the triangulation image having the best possible score. To compute a new 2D triangulation from an initial one, the *edge swap* operator is defined. Two adjacent triangles in the triangulation have one common edge between two vertices and each triangle have a vertex not in common with the other triangle. The vertices not in common are called opposite vertices. The *edge swap* operator deletes the common edge and creates a new edge between the two opposite vertices. Using this operator, the method apply a greedy optimization algorithm to sweep the space of possible 2D triangulations and expect to find the one with the best score. The projection of the best triangulation is the final surface.

The advantage of this method is the fact that the output surface is 2-manifold, but the object to reconstruct is explicitly considered to be of zero genus (having a spherical topology). This limitation makes the method almost unusable in practice. Moreover, the methods was only tested in the very simple cases.

Another interesting surface reconstruction algorithm based on the 2D Delaunay triangulation inside the input image was published in **[Taylor03]**. The main contribution of this paper is the *"Freespace Theorem"*. Suppose there is three 3D points **P**, **Q** and **R**. Suppose they are projected in to the image taken by a camera at point **C** to the points **p**, **q** and **r**. Finally, suppose that the triangle $\triangle pqr$ is a

triangle of the 2D Delaunay triangulation in the said image. The theorem says: the tetrahedron $CPQR$ is entirely free space.

This theorem can be used to define a surface reconstruction method. As the first step, the features are detected and tracked in a series of input images. Then some algorithm is used to reconstruct the cameras positions and the 3D coordinates of the features. As the second step, a 2D Delaunay triangulation of the detected features is computed in each input image and the *"Freespace Theorem"* is used to compute a set of free space tetrahedra. Then, the free space tetrahedra computed from each individual image are merged together to define an implicit function $f : \mathbb{R}^3 \rightarrow \{0, 1\}$. $f(\mathbf{p}) = 0$ if the point $\mathbf{p}$ lies within one of the free space tetrahedra and $f(\mathbf{p}) = 1$ otherwise. To compute the final surface, the space is subdivided in a regular grid and an algorithm such as *"Marching cubes"* [Lorensen87] is applied.

The main drawback of this algorithm is the fact that a regular subdivision of space is used. This makes this method difficult to apply to large scale scenes. Nevertheless, the implicit function can directly be used in some applications, for example for obstacle avoidance.

A more recent surface reconstruction algorithm based on the 2D Delaunay triangulation is the method proposed in **[Salmano9]**. This method takes a sparse 3D cloud of points and the corresponding tracks as its input. A track of a 3D point is a set of cameras that observed the point and the 2D projections of this point to this cameras. This information can be computed, for example, by a Structure-from-Motion algorithm.

The method can be subdivided into three distinct steps: initial point cloud filtering, triangle soup computation and the computation of the final mesh. The initial filtering step is needed to circumvent the errors in the cloud of points produced by an automatic reconstruction process. It has three steps. First, the 3D points that are close are merged together. The corresponding tracks are also merged. Second, some of the 3D points are removed due to the fact that they are too imprecise. Specifically, the points that are too far away from they neighbors are removed as well as the points with a small observation angle. Finally, the cloud of points is smoothed.

The second step of the method is the reconstruction of a triangles soup. For this purpose, a 2D Delaunay triangulation of the tracked points is computed inside each input image. Moreover, for better final results, the Delaunay triangulation is constrained with the contours detected inside the image. Specifically, connections are forced between the tracked points near the same contour. Finally, each triangle is back-projected into 3D space using the known coordinates of each tracked point and so the triangle soup is formed.

The third and final step of the method is the generation of the output surface mesh. For this, first of all, the triangle soup is filtered. The triangles intersected by a ray are removed, as well as the triangles with a small angle between the triangle normal and the rays of the triangle vertices. "Big" triangles are also checked using photo-consistency. Then the Delaunay refinement surface mesh generation algorithm [Boissonnat05] is applied to the triangle soup to compute the final mesh.

The final surface quality is good, but the method is slow and the input cloud of points must be relatively dense.

II.2.2 3D Delaunay triangulation based methods

In the second part of the discussion about *batch sparse* surface reconstruction methods, we will overview the family of methods based on the 3D Delaunay triangulation. These methods work directly with a sparse cloud of 3D points and the associated visibility information. The basic idea of these algorithms is to compute a Delaunay triangulation of the point cloud and to assemble the final surface from the triangles included in this triangulation.

In the first place, we will discuss the *sparse* surface reconstruction method proposed by **[Faugeras90]**. This method is based on tetrahedra labeling and incremental 2-manifold growing based on [Boissonnat84] and has the advantage to be simple to implement, but it suffers from numerous drawbacks.

In the paper the method begin by detecting the edges in the input images and reconstructing them in 3D (the experiments were performed using a stereo pair). Then a 3D Delaunay triangulation T constrained by these edges is computed. The method can directly be used with the interest points instead of the edges.

The next step is to mark some tetrahedra of T as *free-space*. We call $F \subseteq T$ the set of *free-space* tetrahedra. A tetrahedra $\Delta \in F$, if and only if it exists a ray $r \in R$ that intersects Δ. Because the authors of the paper works with edges instead of points, $\Delta \in F$ if and only if it exists a triangle $\triangle CAB$ such as the segment AB have been observed by a camera C and $\triangle CAB \cap \Delta \neq \emptyset$. We also define a set of *unconsidered* tetrahedra $U = T \setminus F$.

The final step of the algorithm is to classify the unconsidered tetrahedra into *free-space* or *matter* (called *obstacle* in the paper, but we call them *matter* to remain coherent with the remaining of this document). To achieve this goal, all the tetrahedra of U are classified into a series of subsets such that the border of each subset is 2-manifold. A subset $S \subseteq T$ is created by adding some random *unconsidered* tetrahedra $\Delta \in U$. This tetrahedra is then removed from U by $U = U \setminus \{\Delta\}$. Then the subset S is incrementally grown by adding the tetrahedra in U which share exactly one, two or three faces with S and all the tetrahedra added to S are removed from U. Finally, when all the tetrahedra of U were considered, the subsets who share at last one connected set of triangles are merged. All the triangles in the same subset must have the same status (*free-space* or *matter*). To classify the subsets, the paper use a heuristic that a subset with a large number of vertices is an obstacle.

The resulting surface is the frontier between *free-space* and *matter* tetrahedra. This method has the advantage of been easy, but it was only experimented on very simple data sets. Moreover, the used heuristic can provide false results and the result is a set of disconnected zero genus 2-manifolds, so the method can not be applied to reconstruct the surfaces of non zero genus.

Another surface reconstruction method based on 3D Delaunay triangulation was published in **[Labatut07]**. As the previous one, it considers the resulting surface as a frontier between two kind of tetrahedra in the triangulation, but it relies on a graph-cut algorithm to perform the tetrahedra labeling.

The method begins by detecting the features in the input images using a SIFT detector [Lowe04]. The features are matched between images using the SIFT descriptor and epipolar geometry [Hartley04] and the resulting 3D cloud of points is reconstructed. The Delaunay triangulation of the point cloud is computed incrementally and the redundant points are eliminated in the process. When a new 3D point is about to be inserted in to the triangulation, the algorithm considers the nearest neighbor of the point and project it in the input image. If the reprojection

error between the new and the reference point is small, the points are considered to be the same and the position of the reference point is updated. Otherwise, the new point is inserted into the triangulation and new tetrahedra are formed.

Once all the 3D points are reconstructed and the Delaunay triangulation computed, the algorithm proceeds with the labeling step. To achieve this goal, the triangulation is considered as a graph with tetrahedra as nodes and facets between tetrahedra as edges. The method considers the output surface S as the frontier between two kinds of tetrahedra (*free-space* and *matter*). This frontier is found by applying a graph-cut algorithm minimizing an energy function $E(S)$. The energy function is defined as $E(S) = E_{vis}(S) + \lambda_{photo}E_{photo}(S) + \lambda_{area}E_{area}(S)$. The first term $E_{vis}(S)$ is the sum of visibility constraints, i.e. the sum of intersections of rays in R with the oriented surface S. The second term $E_{photo}(S)$ is the photo-consistency constraint, it is the sum of photo-consistency measure of each triangle of S. Finally, the last term $E_{area}(S)$ is the area of the surface S and is needed to enforce the smoothness of the final surface.

The results of the experiments showed in the paper are good, but the resulting surface is not guaranteed to be 2-manifold and the 3D point cloud is dense compared to other sparse methods.

An on-line surface reconstruction method allowing user feed-back was proposed by **[Pan09]**. The basic principle is to place the considered object in front of a fixed camera and to allow the user to turn the object around by visualizing the currently reconstructed surface. This allows the user to obtain an optimal number of view to accurately model the object, but the algorithm must be fast enough to allow on-line function.

The features tracking and the surface reconstruction are two separated tasks performed in parallel in two different threads. The feature tracking is performed in real time and allows the camera pose estimation and key frame selection. The 3D surface reconstruction is only performed once per key-frame.

The features tracking and camera poses estimation is performed by several trackers to allow at the same time the tracking of a maximal number of features and a drift-free pose estimation. First of all, in a new incoming image, the features are detected by a FAST detector [Rosten06] and matched with the previous frame by SSD matching.

Once a new key frame is selected, the 3D model is totally recomputed. At first, the 3D cloud of points and the associated visibility information is computed from the tracked features by using a global bundle adjustment. Next the 3D Delaunay triangulation of this point cloud is performed. Then the tetrahedra are removed from the triangulation using the visibility constraints, i.e. a tetrahedra is removed from the triangulation when it was intersected by a ray. To remove the noise from the final surface, the algorithm considers that each observation (and so each ray) exhibit Gaussian noise and so the method compute a probability of the intersections instead of counting them exactly. To accelerate the computations, the algorithm proceeds iteratively. At each iteration, the intersections probabilities are computed only for the tetrahedra which has at last one face belonging to the current surface. Once no more tetrahedra can be removed, the final surface is extracted as the border of the triangulation and the triangles are reprojected to the input images to recover the texture information.

This algorithm has the advantage to be an on-line method, but it is not incremental because the model is completely recomputed at each iteration and so it is unsuited for modeling of large scale scenes. Moreover, the resulting surface is

not guaranteed to be 2-manifold and the experiments was performed only on very simple objects.

Another *sparse batch* surface reconstruction method was made available recently in **[Lhuillier13]**. This method is based on a usage of an omnidirectional catadioptric camera and it is suitable for reconstruction of a large scale urban scenes and has the advantage to produce a 2-manifold output.

The algorithm begins by computing a 3D point cloud and the associated visibility information by using a *Structure-from-Motion* algorithm. More precisely, the features are detected in input images using an Harris detector and are matched between images using ZNCC correlation. Then, the point cloud is computed by a bundle adjustment.

Then, the method perform a 3D Delaunay triangulation T of the point cloud. The tetrahedra of T are classified into *free-space* or *matter* using visibility constraints. A tetrahedron $\Delta \in T$ is *free-space* ($\Delta \in F$) if it exist a ray $r \in R$ such as r intersects Δ. The frontier between *free-space* and *matter* can directly be used as an output surface, but it is not guaranteed to be manifold, so the algorithm performs another partition of T.

The algorithm performs the partition of T into *outside* (O) and *inside* ($T \setminus O$) components, such as *outside* is a subset of *free-space* ($O \subseteq F$). To achieve this goal, a method inspired by [Boissonnat84] is used. First, the tetrahedra of *free-space* are added to *outside* one by one in such a way that the border of O remains manifold. Then, to allow genus changes and not be limited to reconstruction of surfaces with a spherical topology, the tetrahedra are added by packs, assuring that the border of O is manifold. Once no more tetrahedra can be added to *outside*, some post-processing is executed and the final surface is extracted as the border between *outside* and *inside* regions.

The method is fast, can be applied to large scale outdoor scenes and produces a 2-manifold surface. But, the quality of the resulting surface can suffer from artifacts that need to be removed by an additional step. Because the work discussed in this dissertation is heavily inspired by this algorithm, it will be discussed in more details in the section IV.1.

To end the discussion about *batch sparse* surface reconstruction methods a recent publication **[Ohrhallinger13]** should be mentioned. The particularity of this method is the fact that it is fully geometrical, i.e. it compute a surface that fit a point cloud without any additional visibility information. This can be achived by a set of heuristics, but the drawback is that the input cloud of points should be a "good" sampling of the surface to model.

The algorithm begins by computing a Delaunay trinagulation of the input cloud of points P. Then it constructs a *boundary complex* BC_0 by a greedy algorithm. A *boundary complex* is a set of triangles included into Delaunay triangulation and for all points $\mathbf{p} \in P, \mathbf{p} \in BC_0$. The greedy algorithm uses the length of the longest edge of a triangle as a priority criterion when adding triangles to BC_0. The constructed boundary complex BC_0 can contain holes that did not exists in the original surface, so they are detected and covered.

The surface constructed by the previous steps can be non manifold. So an operation called *inflating* is performed to ensure the output surface is a *2-manifold*. A *hull* $H(BC_0)$ is a set of tetrahedra in the Delaunay triangulation which lie inside the boundary defined by the triangles of BC_0. *Inflating* consists in adding tetrahedra to the hull so all the "flat" triangles are eliminated from the boundary. Finally the remaining artifacts are eliminated and a sculpting method similar to [Boissonnat84]

is applied to compute the final surface.

The advantages of this method (particularly compared to [Boissonnat84]) are a good visual quality of the output surface, no limitations concerning the genus of the observed surface and the fact that no visual information is needed. But the drawbacks are the fact that the input point cloud must provide a regular sampling in difficult areas of the surface. The method can also have difficulties with certain kinds of holes and the surface to reconstruct can't have boundaries.

II.3 Incremental sparse modeling methods

In this new section, we will discuss about the surface reconstruction methods which combine two characteristics: being *sparse* and being *incremental*. For memory, a method is *sparse* if it uses only a small fraction of the totality of the pixels of the input images. And a method is called *incremental* if it uses only a small number of the most recently acquired images and if it updates the final model locally instead of totally recomputing it. These methods are similar to the method we describe in this dissertation, but at the same time they are rare in the bibliography. Nevertheless, we describe in this section the few ones which exist.

The first (to the knowledge of the author) *sparse* and *incremental* surface reconstruction algorithm is **[Hilton05]**. It is an extension of [Manessis00]. The basic idea of this algorithm is to incrementally construct a triangle soup by back-projecting the 2D Delaunay triangulations of input images and maintain the visibility constraints.

First of all, 3D reconstruction of the features detected in input images and the successive cameras positions must be incrementally computed. The authors of the paper used an incremental *Structure-from-Motion* in their experiments. The features can be either points or straight lines.

The algorithm begins by initializing the output surface by an empty set of triangles and then for each incoming image it proceed as follow. First of all, the features are detected and reconstructed. The existing triangles are updated if their vertices have moved due to the features update. Then the existing triangles are checked against the visibility constraints introduced by the features detected in the new image. Then a 2D Delaunay triangulation of the detected features is computed in the incoming image. Finally, the triangulation is back-projected into 3D space and the non-redundant triangles are integrated into the output triangle soup.

This method suffers from several important drawbacks. First, the algorithm output is a triangles soup and so is difficult to use. Second, the experiments are performed only on very short and simple sequences.

Another *sparse incremental* method was proposed by **[Lovi10]**. It is part of the family of space carving approaches and its basic idea is to continuously update a partition (*free-space*, *matter*) of tetrahedra in a 3D Delaunay triangulation. The method is coupled with an incremental *Structure-from-Motion*. The authors used an implementation of [Klein07].

At each time, the algorithm maintains a 3D Delaunay triangulation of the current 3D point cloud. Some tetrahedra of the triangulation are marked as *free-space* because they are intersected by one or several rays. The current output surface is the frontier between *free-space* and *matter*. When the *SfM* updates the points cloud, the algorithm reacts in a way dependent on the nature of changes.

First of all, some new points can be added to the cloud of points. The algorithm

updates the Delaunay triangulation, so some tetrahedra are deleted and some tetrahedra are created. The newly created tetrahedra are checked against the previous visibility information (using an heuristic to improve the speed of the computation). Then, the tetrahedra crossed by the rays induced by newly inserted point are set to *free-space*.

Second kind of event produced by the *SfM* algorithm are the addition or suppression of rays. If a new ray is added, the tetrahedra crossed by this ray are set to *free-space*. If a ray is removed, the tetrahedra crossed by this ray are reset to *matter* if they are not crossed by any other ray.

Finally, a point can be deleted from the cloud of points. In this case, the triangulation is updated, the newly created tetrahedra are rechecked against previous visibility constraints. And finally, all the rays induced by the deleted point are deleted as previously described.

If a position of some 3D point is changed because *SfM* refined it, the point is removed then reinserted into the triangulation and the steps described above are applied.

This method has the advantage to be fully incremental, but it uses several heuristics to accelerate the processing and performs a lot of useless computations. Actually, only the most currently observed points are updated by *SfM*. So instead of inserting and removing them several times, the points can only be inserted when the *SfM* doesn't change them.

Another space carving *incremental sparse* surface reconstruction method was proposed by **[Yu12]**. It is a tentative to create an *incremental* version of the method from [Lhuillier13]. The basic idea is to compute a partition of the 3D Delaunay triangulation by region growing so the frontier between the tetrahedra remain 2-manifold at all times. When the new points are inserted into the triangulation, the region growing is performed only in the necessary part of the triangulation.

The 3D cloud of points is computed by an *incremental Structure-from-Motion* algorithm from [Mouragnon09]. The new points are inserted into the triangulation only when the are fully stabilized, i.e. when their position is no longer modified by the *SfM*.

Two separate partitions of the 3D Delaunay triangulation of the cloud of points are maintained. The first is the *free-space* and *matter* partition (a tetrahedron is *free-space* if it was intersected by a ray). The second is the *outside* and *inside* partition. The *outside* tetrahedra are *free-space* and the border of *outside* is the current output surface and is 2-manifold.

When the new points are inserted into the Delaunay triangulation, some tetrahedra are removed and some new tetrahedra are created. Each tetrahedron has a date associated with him. The date of the newly created tetrahedra is set to the current date. Moreover, the *outside* set of tetrahedra O is subdivided into several layers. A layer O_t is a set of tetrahedra that was added to O at a time t. So we have $O_0 \subseteq O_1 \subseteq \cdots \subseteq O_t$ and the boundary of every O_t is *2-manifold*.

We also compute the earliest date of the removed *outside* tetrahedra called t'. Them we remove from *outside* region all the tetrahedra in the layers O_t such as $t' \leq t$. Then a region growing algorithm from [Lhuillier13] is applied starting from the tetrahedra in $O_{t'-1}$ to compute the new *outside* region. This way, the border between layers remain manifold and so the layers can safely be removed from O as long as they are removed in order.

This algorithm have the advantage to produce a 2-manifold surface and been applicable to large scale outdoors scenes, but if the trajectory of the camera taking

input images contains closed loops, the totality of the loop will be recomputed at its end. So the complexity of the single iteration is unbounded and the method is almost *batch* in the presence of large loops.

Finally, another family of *incremental Structure-from-Motion* based surface modeling methods based on graph-cuts was presented in [Sugiura13, Hoppe13]. The basic idea of **[Hoppe13]** is to use a dynamic graph-cut algorithm to incrementally compute a binary labeling of tetrahedra of a 3D Delaunay triangulation of the cloud of points. The output surface is the frontier between the two type of tetrahedra.

For each tetrahedra Δ of the 3D Delaunay triangulation T, the method defines an energy $E(\Delta) = E_u(\Delta, R_\Delta) + \sum_{\Delta' \in \mathcal{N}_\Delta} E_b(\Delta, \Delta', R_\Delta)$ where R_Δ is a set of rays starting from the vertices of Δ and $\mathcal{N}_\Delta$ is the set of four neighboring tetrahedra of Δ. So the energy function to optimize is $\sum_{\Delta \in T} E(\Delta)$. $E_u(\Delta, R_\Delta)$ is defined by the number of rays that intersect Δ and the number of rays in front of Δ. If Δ is intersected by numerous rays it has a high probability of being *free-space*. In the same manner, if a tetrahedron Δ have a high number of rays in front of it, it has a high probability of being *matter*. The term $E_b(\Delta, \Delta', R_\Delta)$ is enforcing the fact that two neighboring tetrahedra Δ and Δ' have a very high probability of bearing the same label, except if they have a common facet intersected by a ray of R_Δ.

A great property of this energy function is that if a new visibility information is added, only the energies of tetrahedra adjacent to the vertex that induced the new ray need to be updated. Moreover, when a new point is inserted into the Delaunay triangulation, only the values of the newly created tetrahedra need to be recomputed. To incrementally update the labeling, the method use a dynamic graph-cut algorithm [Kohli07]. This algorithm takes the previously computed labeling as its input and computes the new one in a time proportional to the number of changed tetrahedra.

This algorithm is fast and well suited to the reconstruction of a large scale scenes, but the resulting surface is not a 2-manifold.

II.4 Conclusion

In this chapter we have surveyed the 3D surface reconstruction methods available in the literature. We have seen that all of them can be separated into *dense* and *sparse* approaches. We have seen that the *sparse* approaches are quite rare compared to their *dense* counterparts. Moreover, as well *dense* as *sparse* can be further separated into *batch* and *incremental* methods and we have seen that *incremental* methods are more scarce that the *batch*.

The 3D surface reconstruction algorithm described in this dissertation is a *sparse incremental* one. The advantages of this properties were described in the chapter I, but we can see that this kind of methods are the less studied ones. Moreover, our algorithm produces a 2-manifold surface and the only one *sparse* and *incremental* reconstruction method producing manifold surfaces is [Yu12]. However, this method have a severe limitation on the trajectory of the camera that observed the scene and our method has not. So we could conclude that the algorithm studied in this document is an useful addition to the state of the art.

CHAPTER III

3D point cloud computation

The *sparse incremental* surface reconstruction method described in this dissertation can be separated into two well distinguishable steps: the computation of the sparse 3D cloud of points with the associated visibility information and the reconstruction of a polygonal surface (a set of connected triangles) from this cloud. In this chapter we describe the first step: computing a 3D cloud of points from a series of 2D images.

The input series of images is taken by a rigid synchronized system of cameras. Although all the experiments are conducted using the *Ladybug* [Lad] (see figure III.1), any system sharing similar characteristics can be used instead. In section III.1 we review the mathematical model describing a rigid multi-camera system used in this dissertation. In section III.2 we describe the algorithm used to compute the point cloud from the input images. This kind of algorithms is called *Structure-from-Motion* or *SfM*. The one used in this dissertation is the algorithm described in [Mouragnon09]. Finally, in section III.3 we describe some steps that could be added to improve the sparse cloud of points.

III.1 Multi camera system modeling

In this section the way of mathematically modeling a rigid multi-camera system is discussed. To simplify the modeling, we consider that all the pictures taken by the cameras of the multi-camera system are concatenated on the same plane to form the output image of the multi-camera system (see figure III.1b for an example). Given a set S, we call $\mathcal{P}_n(S)$ a set of all the subsets of S with no more than n elements. Then, given a 3D point $\mathbf{q} \in \mathbb{R}^3$, we want to define a function $p : \mathbb{R}^3 \to \mathcal{P}_n(\mathbb{R}^2)$ such as $p(\mathbf{q})$ is the set of pixel(s) of the multi-camera system image containing the image of $\mathbf{q}$. $p(\mathbf{q}) = \emptyset$ if the point $\mathbf{q}$ is invisible by the multi-camera system. Several pixels of the image can be associated to the same 3D point if the point is observed by several cameras of the system at the same time. The function p is called the *projection* function.

Moreover, we also want to describe a function $p^{-1} : \mathbb{R}^2 \to \mathcal{P}(\mathbb{R}^3)$ such as if $\mathbf{q}' \in \mathbb{R}^2$ and $\mathbf{q}' = p(\mathbf{q})$, then $p^{-1}(\mathbf{q}')$ is the half-line starting at an optical center of

(a) Pointgrey Ladybug 2 camera.

(b) Example of an image taken by the camera.

Figure III.1: Ladybug camera and an example image.

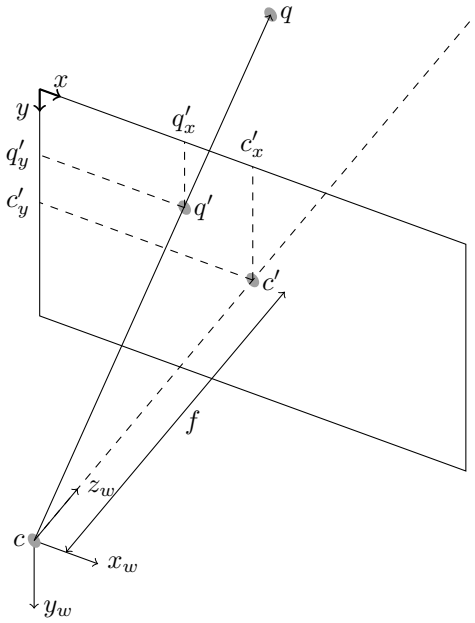


Figure III.2: Pinhole camera model. $\mathbf{c}$ is the camera optical center, $\mathbf{c}'$ is it projection to the image plane. $\mathbf{q}$ is an arbitrary point in 3D space, $\mathbf{q}'$ is its projection to the image plane. f is the camera focal length.

one of the cameras and directed to $\mathbf{q}$. The function p^{-1} is called *back-projection* and the this half-line is called a *ray* associated to $\mathbf{q}'$.

Because the function p for a complete multi-camera system is complex, it is build up incrementally. We begin by defining the *projection* for a simple *pinhole* camera placed at the center of the world in the subsection III.1.1, then we gradually add complexity to finally define the complete *projection* in the subsection III.1.4.

III.1.1 Pinhole camera located at the center of the world

At first, we will consider the model of the *pinhole* camera with the *optical center* located at the center of the world coordinate system (see figure III.2). So we define the world coordinate system by the triplet of direction vectors $\{\mathbf{x}_w, \mathbf{y}_w, \mathbf{z}_w\}$ centered at the point $\mathbf{c}$ called the *optical center*. The distance between the *optical center* and the *image plane* is called the *focal distance* and equals to f. The *image plane* is the plane inside which the final camera image is located. This plane has it own coordinate system defined by a pair of vectors $\{\mathbf{x}, \mathbf{y}\}$ centered at the top left corner of the image. The orthogonal projection of the *optical center* $\mathbf{c}$ to the *image plane* is called $\mathbf{c}'$ and is located at $(c'_x, c'_y)^T$ in the camera image.

Given a point $\mathbf{q} \in \mathbb{R}^3$ located at $(q_x, q_y, q_z)^T$ we want to compute the point $\mathbf{q}' \in \mathbb{R}^2$ located at $(q'_x, q'_y)^T$ such as $\mathbf{q}'$ is the intersection between the segment $\mathbf{cq}$ and the *image plane*. The point $\mathbf{q}'$ is defined by the following expression:

$$\begin{bmatrix} q'_x \\ q'_y \\ 1 \end{bmatrix} \equiv K \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} \text{ where } K = \begin{bmatrix} f_x & s & c'_x \\ 0 & f_y & c'_y \\ 0 & 0 & 1 \end{bmatrix} \tag{III.1}$$

where $\equiv$ means equals up to a non-zero scale factor, f_x and f_y are the *focal length* f expressed in width and height of the pixels and s is the *skew* factor to correct the non rectangular pixels geometry (almost always equals to 0).

The values of the *focal length* f and the *central point* $\mathbf{c}'$ are commonly called *intrinsic* parameters of the camera. These parameters are supposed to be known in the remaining of this work.

The *back-projection* function for an *pinhole* camera is easy to write after the equation (III.1). We only need to compute the inverse of the matrix K. Then we can write:

$$\begin{bmatrix} d'_x \\ d'_y \\ d'_z \end{bmatrix} \equiv K^{-1} \begin{bmatrix} q'_x \\ q'_y \\ 1 \end{bmatrix} \tag{III.2}$$

The direction of the ray (the direction of the segment between $\mathbf{c}$ and $\mathbf{q}$) is computed by normalizing the vector $\mathbf{d}' = (d'_x, d'_y, d'_z)^T$ ($||\cdot||$ is the *euclidean* norm):

$$\begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \frac{1}{||(d'_x, d'_y, d'_z)^T||} \begin{bmatrix} d'_x \\ d'_y \\ d'_z \end{bmatrix} \tag{III.3}$$

III.1.2 Distortion modeling

Unfortunately, the *pinhole* camera model explained in the previous subsection is usually insufficient to model a real camera, because it doesn't take into account the distortion introduced into the image by the camera optical lenses. Actually, if we compute the projection $\mathbf{q}'$ of a 3D point $\mathbf{q}$ using the equation (III.1), we will notice that $\mathbf{q}'$ is different of the point $\mathbf{q}^d \in \mathbb{R}^2$ where the image of $\mathbf{q}$ is actually lying. So we must introduce a function $d : \mathbb{R}^2 \to \mathbb{R}^2$ such as $\mathbf{q}' = d(\mathbf{q}^d)$. This function is called the *distortion* function [Lavest98].

The *distortion* function can be modeled in several ways. The first way is to use a *look-up table*. A *look-up table* is a two dimensional table that, for each pixel $\mathbf{q}^d$ of the image gives the coordinates of the corresponding undistorted point $\mathbf{q}$. So this 2D table allows us to remove the distortion from any pixel with integer coordinates. For the other pixels, the coordinates of the undistorted point are simply linearly interpolated from the four neighboring integer pixels.

Another way to model the distortion is to use a polynomial function [Lavest98]. The *distortion* is separated into two distinct components: the *tangential* distortion and the *radial* distortion. The *tangential* distortion is neglected. To define the *radial* distortion we begin by defining the function $r : \mathbb{R}^2 \to \mathbb{R}$ as the normal squared distance between $\mathbf{q}^d$ and $\mathbf{c}'$. Its expression is

$$r(\begin{bmatrix} q^d_x \\ q^d_y \end{bmatrix}) = ||\frac{\mathbf{q}^d - \mathbf{c}'}{f}||^2 = (\frac{q^d_x - c'_x}{f})^2 + (\frac{q^d_y - c'_y}{f})^2 \tag{III.4}$$

Then we write the function $d(\mathbf{q}^d)$ as follow:

$$\begin{aligned} d(\mathbf{q}^d) = \mathbf{q}^d + (a_5.r(\mathbf{q}^d)^5 + a_4.r(\mathbf{q}^d)^4 + a_3.r(\mathbf{q}^d)^3 \\ + a_2.r(\mathbf{q}^d)^2 + a_1.r(\mathbf{q}^d))(\mathbf{q}^d - \mathbf{c}') \end{aligned} \tag{III.5}$$

The polynomial coefficients a_5, a_4, a_3, a_2 and a_1 are called the *distortion* coefficients.

III.1.3 The camera at an arbitrary position in space

In the previous subsections, we have written the *projection* function for a camera placed at the center of the world coordinate system. This would be fine if we have to work with a single image. In practice we should be able to write a projection function for a moving camera.

As in the subsection III.1.1, the world coordinate system is defined by the triplet of vectors $\{\mathbf{x}_w, \mathbf{y}_w, \mathbf{z}_w\}$ and now is centered at some arbitrary point in space called $\mathbf{c}_w$. We define the camera coordinate system by the triplet of direction vectors $\{\mathbf{x}_c, \mathbf{y}_c, \mathbf{z}_c\}$ centered at the camera *optical center* $\mathbf{c}$. We define the translation between $\mathbf{c}$ and $\mathbf{c}_w$ by a vector $\mathbf{t}_c^w$ and the rotation between $\{\mathbf{x}_c, \mathbf{y}_c, \mathbf{z}_c\}$ and $\{\mathbf{x}_w, \mathbf{y}_w, \mathbf{z}_w\}$ by a rotation matrix R_c^w. The pair $\{R_c^w, \mathbf{t}_c^w\}$ is called the camera *pose*.

Given an arbitrary vector $\mathbf{v}_w \in \mathbb{R}^3$ expressed in the world coordinate system, we call $\mathbf{v}_c \in \mathbb{R}^3$ the same vector expressed in the camera coordinate system. It can be computed by $\mathbf{v}_c = R_c^{wT}(\mathbf{v}_w - \mathbf{t}_c^w)$. And inversely $\mathbf{v}_w = R_c^w \mathbf{v}_c + \mathbf{t}_c^w$. Using this expressions and the camera *projection* equation (III.1), we can write the *projection* equation for an arbitrary placed camera as follows:

$$\begin{bmatrix} q'_x \\ q'_y \\ 1 \end{bmatrix} \equiv K R_c^{wT} [I_3 | - \mathbf{t}_c^w] \begin{bmatrix} q_x \\ q_y \\ q_z \\ 1 \end{bmatrix} \tag{III.6}$$

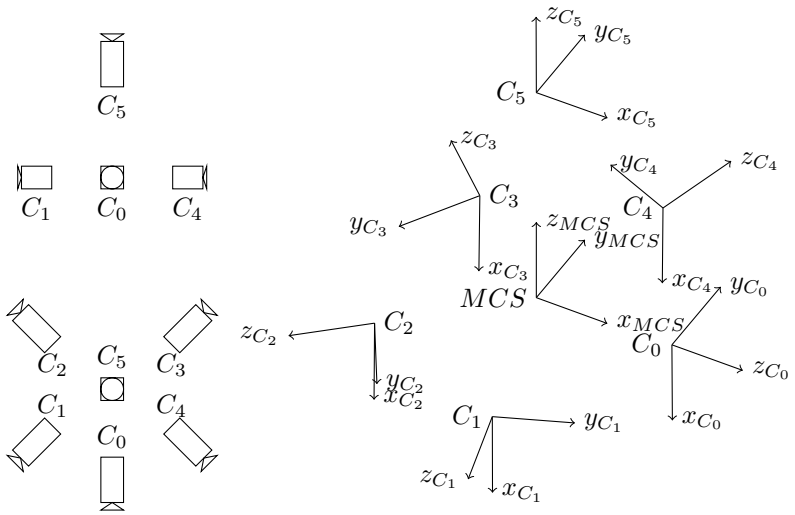
The *back-projection* is simply computed by applying the equation (III.2), then by multiplying the resulting direction by R_c^w. The camera rotation is enough because the *back-projection* returns the *direction* of the ray associated to the pixel (the ray origin is always the camera optical center $\mathbf{c}$).

III.1.4 Rigid multi-camera system modeling

Once we have written the *projection* and the *back-projection* functions for an arbitrary positioned camera, we can define these functions for a complete rigid multi-camera system. We begin by associating a coordinate system to each individual camera. To a camera C_n we associate a coordinate system centered at the camera *optical center* and defined by the triplet $\{\mathbf{x}_{C_n}, \mathbf{y}_{C_n}, \mathbf{z}_{C_n}\}$ (see figure III.3 for an example). We also define a *Multi-Camera System* (MCS) coordinate system centered at the barycenter of the cameras optical centers and defined by the triplet $\{\mathbf{x}_{MCS}, \mathbf{y}_{MCS}, \mathbf{z}_{MCS}\}$. The transformations between the cameras coordinate systems and the multi-camera system coordinate system $\{R_{C_n}^{MCS}, \mathbf{t}_{C_n}^{MCS}\}$ are supposed to be known.

Having all this information, we can now define the *back-projection* function. Knowing the pixel coordinates $(q_x^d, q_y^d)^T$, we can find the number n of the camera of the multi-camera system in which it was seen (we suppose that the coordinates of the top left corner of each cameras images are known). We compute then the *undistorted* pixel coordinates using $d(\mathbf{q}^d)$, then we compute the direction of the ray in the camera coordinate system using the equation (III.2). Finally, the direction of the ray in the world coordinate system is computed by successively multiplying the direction of the ray in the camera coordinate system by $R_{C_n}^{MCS}$ and R_{MCS}^w.

The *projection* function is more difficult to write because we need to be sure that the 3D point is actually visible by one of the cameras and to find in which of the cameras of the multi-camera system it is visible, but fortunately, we never need



(a) Cameras placement. (b) Cameras coordinate systems.

Figure III.3: Ladybug and its cameras coordinate systems. MCS is the coordinate system of the multi-camera system and C_i is the coordinate system of the sub-camera i.

it in practice for 3D point cloud computation. Moreover, the distances between the cameras *optical centers* and the center of the multi-camera system is, usually, only around some centimeters. And because our system is designed to reconstruct large scale scenes and so the observed points are usually further than 3 m. from the camera, we could consider that for all n, $\mathbf{t}_{C_n}^{MCS} = \mathbf{0}$. This is called the *central approximation* [Schmeing11] and it greatly simplifies the computations in the section III.2.

III.2 Incremental Structure-from-Motion

In this section we will review the algorithm called *Structure-from-Motion* or *SfM* that we use. This algorithm is a part of the family of methods commonly called *Simultaneous Localization and Mapping* or *SLAM*. Given a sequence of images and the parameters of the camera (or multi-camera system) model (see section III.1), the *SfM* estimates the *poses* of the camera for each of the images (or for a subset of them) and a 3D cloud of points observed throughout the sequence. Moreover, the method presented here is *incremental*, which means that it updates the list of *poses* and the point cloud when a new image arrives (instead of recomputing the sequence entirely) and so it could be performed at the same time as the acquisition.

The *SfM* can be split in four distinct parts: features detection, features tracking, camera poses estimation and features reconstruction. Each time a new image is acquired, the algorithm proceeds as follow:

- Detect the features (interest points in our case) in the newly acquired image.
- Match these features with the features detected in the previous image (subsection III.2.1).

- Update the set of features *tracks* accordingly. A *track* is a set of occurrences of a particular feature in the successive images, for example a series of 2D points in successive images corresponding to the same 3D point in space.
- Decide if the previous image is a *key frame* (see subsection III.2.2).
- If the previous image is a *key frame*, perform the following additional steps:
  - Compute an approximate pose of the *key frame* (see subsection III.2.3) using a robust estimation method (RANSAC).
  - Refine the poses of the N_{pose} last key frames and the 3D points visible from the N_{obs} last key frames by using a *bundle adjustment* (see subsection III.2.7).
- Acquire the next image and go to the beginning.

In the following subsections, we will review all these steps in more ample details.

III.2.1 Interest points detection and matching

The first task to perform, when a new frame is acquired, is to detect the interest points. By *frame* we mean the concatenation of the n images acquired by the cameras of the multi-camera system, see figure III.1b for an example

The points of interests are detected in each of the images of the frame using the *Harris* interest points detector [Harris88]. They are mainly located in heavily textured parts of the image and include the projections of the corners of the observed objects. This later property is particularly interesting for the surface reconstruction problem. The *Harris* detector is also invariant to illumination conditions. The actual implementation details was taken from [Nistér06]. Each detected point has a confidence score (numerical value) associated to it by the detector, we keep only N_{poi} points with the best confidence score.

After the features were detected, they are matched with the features from the previous frame. First of all, we consider that a feature seen by one of the cameras of the system can only be matched with the feature detected in the same camera. Second, we consider that the movement between the two images is small, so we search for a potential match in a X_{roi} by Y_{roi} zone centered at the current point position. A *Zero mean Normalized Cross-Correlation* (ZNCC) in a w by w neighborhood is computed between the point and all the potential candidates, then the best candidate is kept. For memory a ZNCC can be written:

$$ZNCC_{w\times w}(\mathbf{q_1}, \mathbf{q_2}) = \frac{\Sigma_{\mathbf{d}\in\nu_w}(I(\mathbf{q_1}+\mathbf{d}) - \bar{I}(\mathbf{q_1}))(I(\mathbf{q_2}+\mathbf{d}) - \bar{I}(\mathbf{q_2}))}{\sqrt{\Sigma_{\mathbf{d}\in\nu_w}(I(\mathbf{q_1}+\mathbf{d}) - \bar{I}(\mathbf{q_1}))^2}\sqrt{\Sigma_{\mathbf{d}\in\nu_w}(I(\mathbf{q_2}+\mathbf{d}) - \bar{I}(\mathbf{q_2}))^2}} \quad \text{(III.7)}$$

with

$$\bar{I}(\mathbf{q_i}) = \frac{1}{w^2}\sum_{\mathbf{d}\in\nu_w} I(\mathbf{q_i}+\mathbf{d}) \quad \text{(III.8)}$$

where $\mathbf{q_1}$ and $\mathbf{q_2}$ are the two points to be compared, $I(\mathbf{q})$ is the luminance value of the point $\mathbf{q}$ and ν_w is the $w \times w$ neighborhood.

If the cross-correlation value of the best candidate is less than threshold S_m, the match is rejected. If the match is kept, we check if the point we have selected has

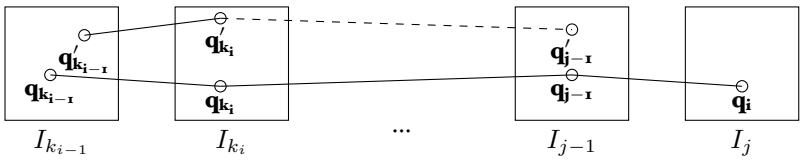


Figure III.4: Features tracking notations. I_j is an image of the sequence with j as index. k_i are the indexes of the key frames. $\mathbf{q}_{k_i}$ is an observation of the point $\mathbf{q}$ in the frame I_{k_i}. The regular lines are true matches and the dashed lines are considered a false matches.

already been matched with another point. If it was not been matched or the cross-correlation value of the previous match is inferior to the new one, the new match is kept and the previous match is removed. This technique have the advantage to avoid a great number of false matches.

III.2.2 Interest points tracking and key frame selection

We have seen how the points of interest are detected and matched with the previous image. In the present subsection we will explain how the points are tracked throughout the sequence. Moreover, we also introduce the concept of ***key frame***.

The problem with an usual video sequence is that the consecutive images are very close to each other. If we try to compute the camera positions associated to each image of the sequence, numerical stability problems occur. To avoid this problem, we choose to perform these computations only on a subset of images evenly distributed across the sequence in a way to be not too close and not to far to each other. These images are called ***key frames***. This technique also helps to detect the false matches as will be explained later.

First of all, we need to define the concept of 2D ***track***. Let n be the number of the current image, so I_j with $j \in \{0, \dots, n\}$ are the currently processed images of the video sequence. Moreover, let m be the number of currently selected key frames. Thus, their indexes (see figure III.4) are k_i with $i \in \{0, \dots, m\}$ and $0 \leq k_i \leq n-1$. Let $\mathbf{q} \in \mathbb{R}^3$ be a point in 3D space. If it was observed in the image I_j the corresponding pixel is named $\mathbf{q}_j$. A ***track*** T_q associated to the 3D point $\mathbf{q}$ is a set of pairs $T_q = \{\{k_i, \mathbf{q}_{k_i}\}, \dots, \{k_m, \mathbf{q}_{k_m}\}, \{n-1, \mathbf{q}_{n-1}\}, \{n, \mathbf{q}_n\}\}$, each pair correspond to an observation of $\mathbf{q}$.

We begin by considering the matches between the current image I_n and the previous image I_{n-1}. If $\mathbf{q}_n$ and $\mathbf{q}_{n-1}$ are two matched points and there is T_q such as $\{n-1, \mathbf{q}_{n-1}\} \in T_q$, then the new observation $\{n, \mathbf{q}_n\}$ is added to T_q, otherwise a new track is formed by $\{n-1, \mathbf{q}_{n-1}\}$ and $\{n, \mathbf{q}_n\}$.

Then we need to decide if the previous image I_{n-1} is a ***key frame***. First, if the previous image is the first image of the sequence then it is always a ***key frame***. Second, we compute two values. The value N_2 is the number of matches between I_{n-1} and I_n. The value N_3 is the number of tracks with the last observation at n and containing at last three observations. Now, if $N_2 \leq M_2$ or $N_3 \leq M_3$ then I_{n-1} is a ***key frame***. The values M_2 and M_3 are user defined thresholds.

If the previous image was **not** a key frame, then we suppress all the observations at $n-1$ from all the currently maintained tracks. This way, the points are contiguously tracked between two key frames $I_{k_{i-1}}$ and I_{k_i}. If there is enough images

between the two consecutive *key frames*, it is a good way to get rid of false matches.

For an ideal sequence, this *key frames* selection mechanism would be enough, but unfortunately in the real world the vehicle or human transporting the camera needs to sometimes stop. Because of the noise in the images, the less and less features will be tracked and if the pause was long enough, a new *key frame* will be taken at the same position as the previous one. If this happen, the entire process may fail because of the unstable computations.

To circumvent this problem, another *key frame* selection criterion was added. Let $d(\mathbf{q}_{n-1}, \mathbf{q}_n)$ be the euclidean distance between the 2D coordinates of the two points $\mathbf{q}_{n-1}$ and $\mathbf{q}_n$. If the value of $d(\mathbf{q}_{n-1}, \mathbf{q}_n) \leq D_s$ (a user defined threshold) for 70% of the matches between I_{n-1} and I_n then I_{n-1} is **not** a *key frame*. This is based on the following heuristic: if the overall displacement of points inside the image is short, then the camera displacement was short.

III.2.3 Epipolar geometry and pose estimation

When a new *key frame* is selected, our first task is to evaluate the camera *pose* corresponding to this image. For memory, a camera *pose* is a pair $\{R_c^w, \mathbf{t}_c^w\}$ defining the transformation between the world coordinate system and the camera coordinate system. Now let $\mathbf{q} \in \mathbb{R}^3$ be some arbitrary known point of 3D space observed by the camera at the undistorted pixel $(q'_x, q'_y)^T$. According to the equation (III.6) and because the matrix K is known, this 3D point provides two equations with the pair $\{R_c^w, \mathbf{t}_c^w\}$ as unknown parameters. So if we have enough known 3D points observed by the camera we can compute the *pose*. In this subsection, we suppose that all the matches between the detected features are true (we will see how to handle false matches later), so only three 3D points is necessary using the Grunert's method [Haralick94]. It is based on the trigonometry relationships inside the tetrahedron formed by the camera's *optical center* and the three observed points.

The problem is more complicated if no 3D point position is known. This case arises at the beginning of the algorithm when neither *pose* nor 3D point is being computed yet. To solve it, we will use a set of constraints that enable to define the relative transformation between two cameras coordinate systems when the two are observing the same scene. This set of constraints is called *epipolar geometry* and is illustrated on the figure III.5.

We consider two cameras with the *optical centers* at unknown points $\mathbf{c}_1$ and $\mathbf{c}_2$ in the world coordinate system. These two cameras are observing a 3D point $\mathbf{q}$ at unknown coordinates in the *undistorted* pixels $\mathbf{q}'_1$ and $\mathbf{q}'_2$ respectively. Using the equations (III.2) we can compute the direction $\mathbf{d}_1$ of the ray $\mathbf{c}_1\mathbf{q}$ in the first camera coordinate system and the direction $\mathbf{d}_2$ of the ray $\mathbf{c}_2\mathbf{q}$ in the second camera coordinate system. So if $\{R_{c_1}^w, \mathbf{t}_{c_1}^w\}$ is the unknown *pose* of the first camera and $\{R_{c_2}^w, \mathbf{t}_{c_2}^w\}$ is the unknown *pose* of the second camera, the vectors $\mathbf{t}_{c_1}^w$, $\mathbf{t}_{c_2}^w$, $\mathbf{t}_{c_1}^w + R_{c_1}^w\mathbf{d}_1$ and $\mathbf{t}_{c_2}^w + R_{c_2}^w\mathbf{d}_2$ are lying on the same plane. Hence[1]:

$$\mathbf{d}_1^T R_{c_1}^{wT} [\mathbf{t}_{c_2}^w - \mathbf{t}_{c_1}^w]_\times R_{c_2}^w \mathbf{d}_2 = \mathbf{o} \tag{III.9}$$

with $[\mathbf{a}]_\times$ is a skew-symmetric 3×3 matrix of the vector $\mathbf{a}$ such as $\mathbf{a} \wedge \mathbf{b} = [\mathbf{a}]_\times \mathbf{b}$. If we define a matrix E as $E = R_{c_1}^{wT} [\mathbf{t}_{c_2}^w - \mathbf{t}_{c_1}^w]_\times R_{c_2}^w$, the equation (III.9) is written

[1] The vectors $\mathbf{t}_{c_1}^w, \mathbf{t}_{c_2}^w, \mathbf{t}_{c_2}^w + R_{c_2}^w\mathbf{d}_2$ and $\mathbf{t}_{c_1}^w + R_{c_1}^w\mathbf{d}_1$ are coplanar. So the vectors $\mathbf{o}, \mathbf{t}_{c_2}^w - \mathbf{t}_{c_1}^w, R_{c_1}^w\mathbf{d}_1$ and $\mathbf{t}_{c_2}^w - \mathbf{t}_{c_1}^w + R_{c_2}^w\mathbf{d}_2$ are coplanar. The equation of the plane is $\mathbf{d} + \mathbf{n}^T\mathbf{x} = \mathbf{o}$. We have $\mathbf{d} = \mathbf{o}$ since the vector $\mathbf{o}$ is one of the coplanar points. $\mathbf{n} = (\mathbf{t}_{c_2}^w - \mathbf{t}_{c_1}^w) \wedge (\mathbf{t}_{c_2}^w - \mathbf{t}_{c_1}^w + R_{c_2}^w\mathbf{d}_2) = [\mathbf{t}_{c_2}^w - \mathbf{t}_{c_1}^w]_\times R_{c_2}^w\mathbf{d}_2$. So $\mathbf{d}_1^T R_{c_1}^{wT}\mathbf{n} = \mathbf{o}$, hence the equation.

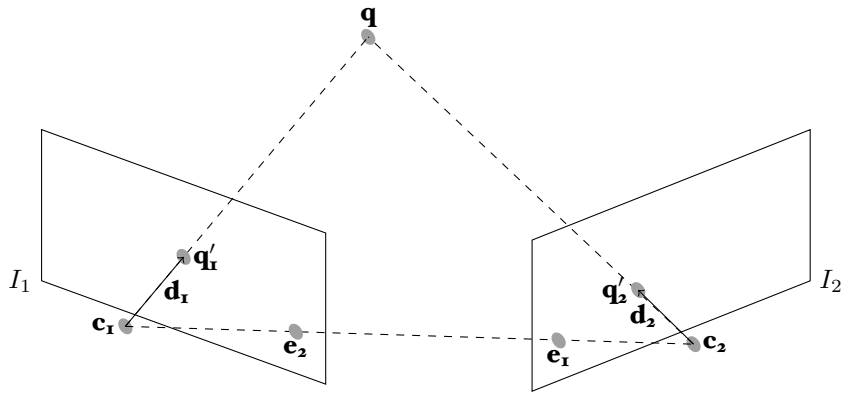


Figure III.5: Epipolar geometry. $\mathbf{c}_1$ and $\mathbf{c}_2$ are the cameras optical centers. I_1 and I_2 are the image planes of the cameras. $\mathbf{e}_1$ is the projection of $\mathbf{c}_1$ to I_2 and $\mathbf{e}_2$ is the projection of $\mathbf{c}_2$ to I_1. $\mathbf{q}$ is an arbitrary 3D point, $\mathbf{q}'_1$ is the projection of $\mathbf{q}$ to I_1 and $\mathbf{q}'_2$ to I_2. $\mathbf{d}_1$ is the direction of $\mathbf{c}_1\mathbf{q}$ and $\mathbf{d}_2$ is the direction of $\mathbf{c}_2\mathbf{q}$.

in a simplified form:

$$\mathbf{d}_1^T E \mathbf{d}_2 = \mathbf{o} \tag{III.10}$$

The matrix E is called the *essential* matrix [Faugeras93].

If we can compute the matrix E, we obtain $R^w_{c_1}$, $R^w_{c_2}$ and $\mathbf{t}^w_{c_2} - \mathbf{t}^w_{c_1}$ up to a scale. Thanks to the equation (III.10) each 3D point observed by the two cameras provides a constraint on E. If we consider the matches to be perfect, five commonly observed points is enough. A curious reader can read [Nistér04] for more details.

At the beginning of the sequence, we define the world coordinate system as being centered at the first camera *pose*, so $\mathbf{t}^w_{c_1} = \mathbf{o}$. This gives us the *poses* $\{R^w_{c_1}, \mathbf{t}^w_{c_1}\}$ and $\{R^w_{c_2}, \mathbf{t}^w_{c_2}\}$ of the two cameras.

III.2.4 3D point estimation

When the *pose* of the newly selected *key frame* is computed, another problem remains unresolved. We need a way to compute the initial estimation of newly observed 3D points.

At first glance, the problem appears to be simple. Let $\mathbf{q} \in \mathbb{R}^3$ be a point observed by two cameras with the *optical centers* $\mathbf{c}_1$ and $\mathbf{c}_2$ and known *poses*. We can easily compute the directions (in the world coordinate system) of the associated rays $\mathbf{d}_1$ and $\mathbf{d}_2$. Then the point $\mathbf{q}$ is the intersection of the two half-lines $[\mathbf{c}_1; \mathbf{d}_1[$ and $[\mathbf{c}_2; \mathbf{d}_2[$. Unfortunately these half-lines almost never intersects in 3D due to noise in the observations.

In reality we find ourselves in a case illustrated on the figure III.6. Let l be a line perpendicular to the half-lines $[\mathbf{c}_1; \mathbf{d}_1[$ and $[\mathbf{c}_2; \mathbf{d}_2[$ and intersecting these two half-lines. Let $\mathbf{q}_1$ be the intersection between l and $[\mathbf{c}_1; \mathbf{d}_1[$. Let $\mathbf{q}_2$ be the intersection between l and $[\mathbf{c}_2; \mathbf{d}_2[$. We define the point $\mathbf{q}$ as lying in the middle of the segment $\mathbf{q}_1\mathbf{q}_2$. This method is called the *middle point* algorithm [Faugeras93].

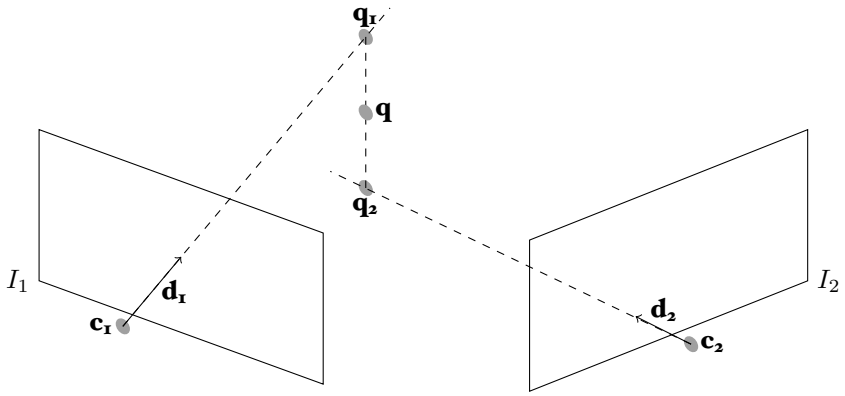


Figure III.6: Middle point triangulation method. $\mathbf{c}_1$ and $\mathbf{c}_2$ are the cameras optical centers. I_1 and I_2 are the image planes of the cameras. $\mathbf{q}$ is the observed 3D point. $\mathbf{d}_1$ is the direction of the segment between $\mathbf{c}_1$ and the detected image of $\mathbf{q}$ in I_1. $\mathbf{d}_2$ is the direction of the segment between $\mathbf{c}_2$ and the detected image of $\mathbf{q}$ in I_2.

III.2.5 Ray-based error function

The computations of the subsections III.2.3 and III.2.4 would be enough if we were sure that the tracking process of the 2D features was perfect, i.e. all the observations of the same track are of the same 3D point. Unfortunately, the process described in subsections III.2.1 and III.2.2 is not, so we need a way to detect that some observations of the track or even the entire track is erroneous. To achieve this goal we use the same angular error as in [Mouragnon09].

To be able to reconstruct a 3D point (see subsection III.2.4) in a robust manner, at last three 2D observations of the point are needed. The first two are used to estimate the 3D position of the point and all the others are needed to verify this estimation. Let see the situation on the figure III.7. We are in the camera coordinate system, $\mathbf{q}$ is the 3D position of the observed point and $\mathbf{c}$ is the camera *optical center*. We call $\mathbf{d}_q$ the direction of the half-line $\mathbf{cq}$ (the projection ray). If $\mathbf{q}'$ is the 2D observation of the point $\mathbf{q}$ in the camera image plane, we call $\mathbf{d}'_q$ the direction of the half-line $\mathbf{cq}'$ (the observation ray). We can define the angular error as the angle α_q between $\mathbf{d}_q$ and $\mathbf{d}'_q$.

In practice, instead of computing the angle α_q directly, we prefer to compute a 2D vector $\mathbf{e}_q$ such as:

$$||\mathbf{e}_q||^2 = tan^2(\alpha_q) \tag{III.11}$$

We use a function $\pi_2 : \mathbb{R}^3 \to \mathbb{R}^2$ as $\pi_2((x, y, z)^T) = (\frac{x}{z}, \frac{y}{z})^T$ and a rotation matrix $R_{q'}$ such as $R_{q'}\mathbf{d}'_q = (0, 0, 1)^T$.

Then:

$$\begin{aligned} tan^2(\alpha_q) &= tan^2(angle(\mathbf{d}'_q, \mathbf{d}_q)) \\ &= tan^2(angle(R_{q'}\mathbf{d}'_q, R_{q'}\mathbf{d}_q)) \\ &= tan^2(angle((0, 0, 1)^T, R_{q'}\mathbf{d}_q)) \end{aligned} \tag{III.12}$$

And because $||\pi_2((x, y, z)^T)||^2 = \frac{x^2+y^2}{z^2} = tan^2(angle((0, 0, 1)^T, (x, y, z)^T))$ the vector $\mathbf{e}_q$ can be written:

$$\mathbf{e}_q = \pi_2(R_{q'}\mathbf{d}_q) \tag{III.13}$$

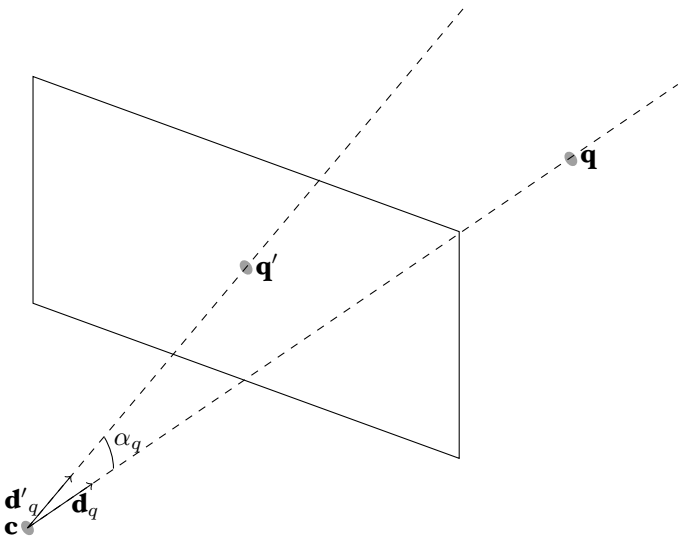


Figure III.7: Angular error. **c** is the camera optical center. **q** is an arbitrary 3D point. $\mathbf{q}'$ is the detected image of **q** in the camera image plane. $\mathbf{d}_q$ is the direction of the segment **cq**. $\mathbf{d}'_q$ is the direction of the segment $\mathbf{cq}'$. α_q is the angle between **cq** and $\mathbf{cq}'$.

The same equation can also be written using the world coordinate system. If the camera *pose* is $\{R_c^w, \mathbf{t}_c^w\}$ then the equation become:

$$\mathbf{e}_q = \pi_2(R_{q'} R_c^{wT}[I_3| - \mathbf{t}_c^w] \begin{bmatrix}\mathbf{q}\\ 1\end{bmatrix}) \tag{III.14}$$

For an observation to be considered true, the angular error associated to this observation must be less than a given threshold. The usual threshold is an angular error corresponding to a shift of two pixels in the image plane. It is defined by the camera *intrinsic* parameters. An observation with the error less than the threshold is called *inlier* and an observation with the error greater than the threshold is called *outlier*. A 3D point is considered during the remaining of the reconstruction process only if the associated *track* contain at last three *inliers*.

III.2.6 Robust estimation

The methods of estimation of the camera *pose* and the positions of 3D points explained in the subsection III.2.3 and III.2.4 are well suited for the case when all the matches between consecutive images are true, unfortunately, in the real world they are not. Moreover, the false matches are quite numerous. So we need some sort of a mechanism to eliminate the false observations (*outliers*) during our reconstruction process. The method we use to robustly estimate the 3D coordinates of a point or camera *pose* is called Random SAmple Consensus or *RANSAC* for short. This technique was firstly presented in [Fischler81].

We will begin by the computation of the 3D coordinates of a point **q** (see subsection III.2.4). Suppose that the *track* T_q associated to the point **q** contains N_{obs} observations. Only two observations are needed to estimate a point position. To

find these two observations we will proceed iteratively. At each iteration, we randomly select two observations out of N_{obs} available and estimate the 3D coordinates of the point $\mathbf{q}$ as in III.2.4. Then we compute the value of the angular error (see III.2.5) for each observation of T_q. The number of *inliers* in the *track* is the iteration score. The final 3D coordinates of the point are the coordinates computed during the iteration with the largest score.

Assume that we want to find a correct final solution with a probability of $p = 0.99$ knowing that approximately ϵ percent of the observations are *outliers*. If we need m observations to estimate one putative solution ($m = 2$ in our case), then the number of *RANSAC* iterations N can be estimated using the following equation [Fischler81]:

$$N \geq \frac{log(1-p)}{log(1-(1-\epsilon)^m)} \tag{III.15}$$

The camera *pose* (subsection III.2.3) can be robustly estimated using the same technique. Suppose that the camera image associated to a newly selected *key frame* contains the observations of N_{3D} 3D points. According to the subsection III.2.3, we only need the observations of 3 3D points to estimate the *camera* pose. So during one iteration of *RANSAC*, we randomly select three 3D points from N_{3D} and use them to estimate the pair $\{R_c^w, \mathbf{t}_c^w\}$ (see III.2.3). Then we compute the angular error for all the observations of the image (see subsection III.2.5) and count the number of *inliers*. This gives us a putative camera *pose* and the associated score (the number of *inliers*). The final *pose* is the *pose* computed during the iteration with the largest score.

For the initial estimation of the three first camera *poses* when no 3D points positions are known, we proceed in the similar manner [Nistér04]. The initial *poses* estimation is taking place when the first three *key frames* were selected, hence the name of the first triplet estimation. Let these three *key frames* be called I_{k_0}, I_{k_1} and I_{k_2}. During the iteration of *RANSAC*, we begin by selecting 5 tracks observed in these three *key frames*. The five points algorithm referenced in the subsection III.2.3 is applied to estimate the positions of the *key frames* I_{k_0} and I_{k_2}. Then the position of the *key frame* I_{k_1} is estimated using the Grunert's [Haralick94] algorithm. Finally the 3D coordinates of the points associated to all the *tracks* are computed using the observations from I_{k_0} and I_{k_2} as well as the associated errors. The final number of *inliers* is the *RANSAC* iteration score. The final *poses* are the *poses* computed during the iteration with the largest score.

III.2.7 Bundle adjustment

The camera *pose* and the positions of the 3D points estimated using *RANSAC* as explained in the previous subsection are not guaranteed to be perfect. Moreover, we need some way to refine the *poses* and 3D points coordinates when some new observations are added in the future. This is why, once a new *key frame* was selected, the initial *pose* estimated and new 3D points reconstructed, an optimization step called bundle adjustment (*BA*) is applied.

Let L_k be the set of the N_{pos} last *key frames*. Let L_q be the set of the 3D points observed in the last N_{obs} *key frames*. Finally, let L_{obs} be the set of the observations of the points of L_q in the last N_{obs} *key frames*. We define $\mathbf{e}_k^q$ as the angular error of the observation of the point $\mathbf{q}$ in the *key frame* I_k.

The bundle adjustement is a non linear optimization problem with the objective

function defined by the following equation:

$$E(L_k, L_q) = \sum_{\{k,\mathbf{q}\} \in L_{obs}} ||\mathbf{e}_k^q||^2 \tag{III.16}$$

The optimized parameters are the ***poses*** associated with the ***key frames*** of L_k and the coordinates of the 3D points of L_q. This problem is numerically solved using a sparse *Levenberg-Maquard* method. The description of this method is outside the scope of this dissertation, a curious reader can read [Mouragnon09] for more details.

If all the observations are considered directly, the optimization can actually degrade the results because it will try to fit to all the constraints, even the false ones. On the other hand, we need a way to update the ***inlier*** status of the observations. So the optimization is performed twice per ***key frame***. First the optimization is performed by considering only the angular errors associated to ***inliers*** as defined by the previous optimizations and the last ***pose*** estimation. This way, the optimization is not disturbed by the false matches. Then the ***inlier/outlier*** status of all the observations is updated, this usually leads to more ***inliers***. Then the optimization is performed anew with the help of new ***inliers***.

III.3 Sparse 3D point cloud improvement

The 3D cloud of points computed from a sequence of images by the *Structure-from-Motion* algorithm as explained in the previous sections can directly be used by the surface reconstruction algorithm. But this is not always the best solution. In fact, this algorithm was optimized for the resolution of the camera localization application. For surface estimation application instead, the 3D points themselves are more important than the camera positions.

In this brief section we will explain some optional steps of the *SfM* algorithm as described in the section III.2. They consists in computing the poses of all the frames in between the ***key frames*** (subsection III.3.1), then in recomputing the cloud of points using these ***poses*** (subsection III.3.2). This way, even the points observed between ***key frames*** will be reconstructed and so hopefully the quality of the final surface will be enhanced.

The overall algorithm (the base algorithm plus the additional steps) remains incremental. The additional steps described here are performed each time a new ***key frame*** was processed by the basic algorithm of the section III.2. The output of the overall algorithm is not the cloud of points maintained by the base algorithm, but the cloud of points maintained by the step described in the subsection III.3.2.

III.3.1 Intermediate poses estimation

The first of the additional steps is the evaluation of the camera ***poses*** in between the ***key frames*** of the input sequence. First of all, to achieve this goal, the base algorithm is performed exactly as described in the section III.2.

There we call I_{k_0} the lastly selected ***key frame***, $I_{k_{-1}}$ the previous one, etc. The first ***key frame*** that won't be modified by the base algorithm any further is $I_{k_{-N_{pos}}}$. So once a new key frame is selected and was processed by the base algorithm, we compute the camera ***poses*** associated to the frames between $I_{k_{-N_{pos}}}$ and $I_{k_{(-N_{pos}+1)}}$ (which will not be updated).

These frames are processed in order. First the points of interest are detected in the frame and matched with the previous frame (or $I_{k-N_{pos}}$ for the first frame of the set). This step is needed because the base algorithm hasn't kept the intermediate matches. Then, only the tracks corresponding to the 3D points reconstructed by the base algorithm are kept. Now we have the correspondences between 3D points and their 2D observations in the frames between the *key frames*. The initial estimation of the camera *pose* associated with the considered frame is performed by the Grunert's algorithm and *RANSAC* (see subsections III.2.3 and III.2.6). Finally, the camera *pose* is optimized using a simplified bundle adjustment. The only parameter to optimize is the *pose* and the objective function is computed only considering the observations of the current frame.

III.3.2 Additional 3D points estimation

When the camera *poses* associated to the frames in between the *key frames* are known, we can proceed to the additional 3D points estimation. In practice, the computations of this subsection are performed each time the *pose* of a new intermediary frame (named I_0 in this subsection) is computed by the processing of the subsection III.3.1.

First of all, the interest points are detected in the frame I_0 and matched with the previous frame I_{-1}. The number of the detected and tracked features is usually greater than the number used by the conventional *SfM*. This part of the algorithm maintains a set of *tracks* totally isolated from the set of *tracks* maintained by the base algorithm.

Once a *track* is stopped, i.e. it has no observation in the current frame I_0, the 3D coordinates of the corresponding point are computed. To achieve this goal, we begin by computing an initial estimation of the point coordinates by performing a *RANSAC*. Inside an iteration of the *RANSAC*, we begin by randomly selecting two observation in the *track*. Then we triangulate the 3D point position using these observations and the middle point algorithm (see subsection III.2.4). Finally, the angular error is computed for each observation of the *track* and the number of *inliers* is the iteration score of *RANSAC*.

When the initial estimation of the 3D point position was computed by the *RANSAC*, we refine this position using a simplified bundle adjustement. The only parameter of the optimization problem is the position of the 3D point and the objective function is computed only considering the observations of the *track*.

III.4 Conclusion

In this chapter we have reviewed how to compute a sparse 3D cloud of points with the associated visibility information from an input sequence of images. We have begun by defining the mathematical model of the rigid multi-camera system. Then we have described the steps which extract the points of interest from the input images, match them between the successive frames and finally compute the cameras and 3D points positions [Mouragnon09]. Moreover, we have talked about an optional extension to the *SfM* algorithm that allows to compute an improved sparse cloud of points by taking into account the observations between *key frames*.

The results of this algorithm on both the real and synthetic sequences will be presented at the same time as the results of the surface reconstruction algorithm

in section V.2 of the chapter V.

Of course, this algorithm is not the only option to compute a cloud of points from a set of images. But, this one is widely used at Institut Pascal and so its performance and limitations are well known to us. This makes it a good choice.

CHAPTER IV

Incremental surface reconstruction algorithm

In this chapter the proposed incremental surface reconstruction algorithm is explained and studied in details. The input of this algorithm is a 3D cloud of points and the associated set of rays (for memory, a *ray* is a segment between a camera position and the 3D point observed by this camera). This cloud of points is incrementally produced by the *Structure-from-Motion* algorithm described in the chapter III. Each time the point cloud is modified by the *SfM*, the surface reconstruction algorithm updates the output surface. The output surface is a set of triangles. This surface is guaranteed to be *2-manifold* at all times.

We begin by a detailed description of the 2-manifold sparse surface reconstruction method from [Lhuillier13] in the section IV.1. The algorithm proposed by this dissertation is heavily inspired by this work and so this description is useful to understand some basic concepts. Moreover, for the sake of clarity, it is easier to begin the description of the surface reconstruction process in a batch context.

Section IV.2 details our incremental surface reconstruction algorithm. Finally, its time complexity is analyzed in section IV.3.

IV.1 Detailed description of previous works

In this section we will describe the sparse 2-manifold surface reconstruction algorithm published in [Lhuillier13]. This is a *batch* method which takes as input the final cloud of 3D points $P = \{\mathbf{p}_i\}$, the cameras positions $C = \{\mathbf{c}_j\}$ and the set of rays $V = \{\mathbf{c}_j\mathbf{p}_i, \mathbf{c}_j \in C, \mathbf{p}_i \in P\}$ as produced by the *Structure-from-Motion*. The algorithm output is the 2-manifold surface S, a set of triangles.

We consider that the scene observed by the omnidirectional camera is rigid and opaque. The main idea of the algorithm is to separate the space into two distinct regions: the *free-space* region and the *matter* region. The *free-space* region is the space visible by the camera, i.e. intersected by one of the rays of V. The *matter* region is the remaining space. We suppose that the surface approximating the observed scene lies between these two regions. So the objective of the algorithm is to compute a 2-manifold approximation of this surface.

The batch algorithm proceed step by step:

1. A tetrahedral discretization of space is computed by performing a 3D Delaunay triangulation of the cloud P. The computed triangulation is called T (see subsection IV.1.1).

2. A free-space/matter binary labeling of tetrahedra of T is performed using the rays of V (see subsection IV.1.2).

3. The binary labeling is refined with the aid of acute tetrahedra removal (see subsection IV.1.3).

4. A greedy algorithm is used to perform a second binary labeling of T. This one is called outside/inside. The outside tetrahedra are free-space tetrahedra, but the boundary of outside is 2-manifold (see subsection IV.1.5).

5. The outside/inside binary labeling is refined by an artifacts removal algorithm (see subsection IV.1.6).

6. The outside/inside binary labeling is refined by a peak removal algorithm (see subsection IV.1.7).

7. The final surface S is the border of outside region. It is extracted and smoothed (see subsection IV.1.8).

In the following subsections, all these steps will be reviewed in details.

IV.1.1 Space discretization (3D Delaunay)

The first step of the batch surface reconstruction is to perform the discretization of space. Before we can explain how this can be achieved we need to mathematically define what the discretization is. We can use the notion of *simplical complex* [Moise97, Giblin10].

First, we define a *simplex*:

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

$$\sigma_P = \{\sum_{i=0}^{k} \lambda_i \mathbf{p}_i, \lambda_i \in \mathbb{R}^+, \sum_{i=0}^{k} \lambda_i = 1\}$$

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

In $\mathbb{R}^3$, we can find four kind of *simplices*: vertices (0-simplices), edges (1-simplices), faces (2-simplices) and tetrahedra (3-simplices). If P' is a subset of P and contains k' elements, the simplex $\sigma_{P'}$ is a k'-face of σ_P (we note $\sigma_{P'} < \sigma_P$).

Then, we can define the *simplical complex*:

Definition IV.2 (Simplical complex)
A simplical 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. A *simplical complex* K' is a subcomplex of K if $K' \subseteq K$. Two k-simplices σ and τ of K are incident if $\sigma \cap \tau \neq \emptyset$. They are adjacent, if σ and τ have the same dimension and $\sigma \cap \tau$ is a $(k-1)$-simplex of K.

So, we obtain a discretization of space by the computation of a *simplical complex* in $\mathbb{R}^3$ with the points of P as *vertices*. In practice, we use a 3D Delaunay triangulation which is a basic tool of surface reconstruction from cloud of points [Cazals04, Dey07]. It is defined as follows:

Definition IV.3 (3D Delaunay triangulation)
Let $P = \{\mathbf{p}_0, \cdots, \mathbf{p}_n\}$ *be a set of* $n \geq 4$ *non coplanar points of* $\mathbb{R}^3$*. A 3D Delaunay triangulation of* P *is a 3D* simplical complex T *such as*

- P *is the set of vertices of* T*;*
- *the union of the tetrahedra of* T *is the convex hull of* P*;*
- *the circumscribing sphere of every tetrahedra of* T *doesn't contain any vertex in its interior.*

We call $\mathcal{T}$ *the set of tetrahedra of* T*.*

The advantages of the 3D Delaunay triangulation are that it always exists, it is unique for a given points set P if P doesn't contain 4 coplanar or 5 cospherical points and it can be proven that it contains a subcomplex that is a "good" approximation of the surface that is sampled by P [Amenta99, Boissonnat05]. Moreover, when a new vertex is inserted into the triangulation, the only modified tetrahedra are those whose circumscribing sphere contains the vertex. This property will be used in our *incremental* algorithm (see section IV.2).

We also define the notion of boundary for a subcomplex of the triangulation T:

Definition IV.4 (Triangulation boundary)
Let T *be a 3D Delaunay triangulation and* $L \subseteq \mathcal{T}$*. We call the boundary of* L*:*

- *A list* ∂L *of triangles that are faces of tetrahedra of* L*;*
- *Each triangle of* ∂L *is included in one and only one tetrahedron of* L*.*

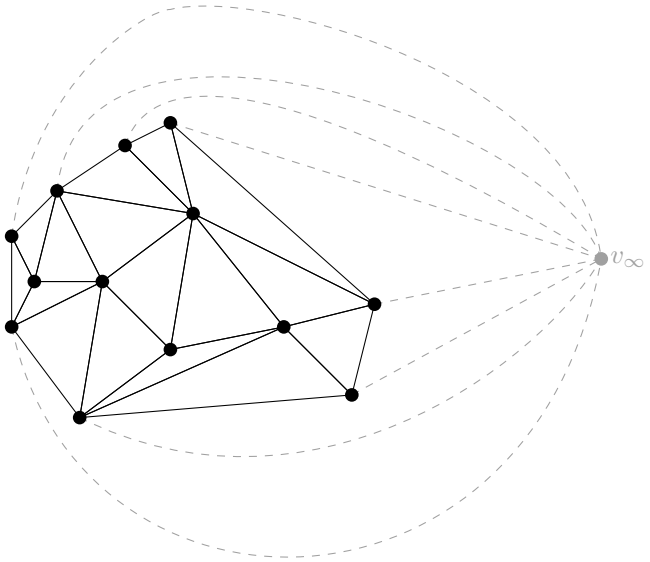


Figure IV.1: Infinite vertex role inside a Delaunay triangulation (2D case). The black dots are the vertices of the Delaunay triangulation and the gray dot is the infinite vertex. The black solid lines are the real edges of the triangulation and the gray dashed lines are the virtual edges.

An adjacency graph Γ_T can be associated to the 3D Delaunay triangulation T. It is defined as follows:

Definition IV.5 (Adjacency graph)
The adjacency graph Γ_T of the 3D Delaunay triangulation T is a graph such as:

- *The vertices of the graph are the tetrahedra of T;*
- *The edges of the graph are the triangles between two tetrahedra of T.*

As can easily be seen, the graph Γ_T isn't 4-regular: most of the tetrahedra of T have 4 neighbors, but some of them have only three or even less. To simplify the implementation of the algorithms, we use the *infinite vertex* v_∞ [Boissonnat00b]. This vertex didn't exists in the triangulation, but virtual tetrahedra are introduced to Γ_T that connects each triangle of ∂T to v_∞ (see figure IV.1). This way Γ_T becomes a 4-regular graph.

Before constructing the 3D Delaunay triangulation of the cloud of points P produced by the *SfM*, the set P is filtered to remove the points with bad accuracy. Consider a point $\mathbf{p} \in P$. We call $V_\mathbf{p} = \{\mathbf{c}_j\mathbf{p}, \mathbf{c}_j\mathbf{p} \in V\}$, the set of rays associated to the point $\mathbf{p}$. If all the rays of $V_\mathbf{p}$ is nearly parallel, the accuracy of the point $\mathbf{p}$ will be poor [Hartley04]. If there is at least one angle $\widehat{\mathbf{c}_i\mathbf{p}\mathbf{c}_j}$ such as $\mathbf{c}_i\mathbf{p} \in V$, $\mathbf{c}_j\mathbf{p} \in V$ and $\epsilon \leq \widehat{\mathbf{c}_i\mathbf{p}\mathbf{c}_j} \leq \epsilon - \pi$ then the point is kept, otherwise it is removed. The angle ϵ is a user defined threshold.

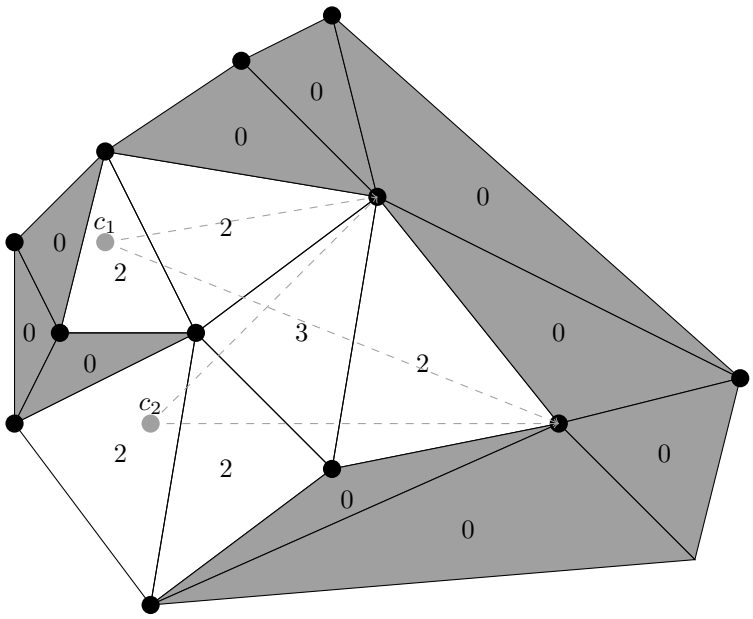


Figure IV.2: The principle of the *free-space/matter* binary labeling (2D case). White triangles are *freespace*, gray triangles are *matter*. The numbers inside the triangles are their number of intersections $I(\Delta)$. The gray dots are the cameras and the gray lines are the rays.

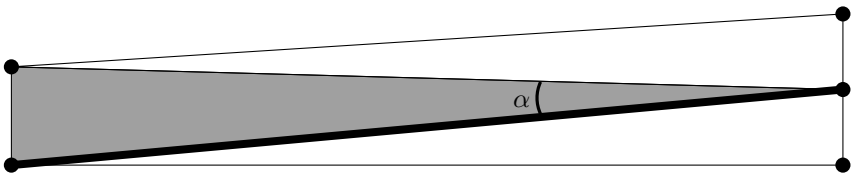
IV.1.2 Free-space/Matter binary labeling

After the space was discretized by performing a 3D Delaunay triangulation T of the input set of points P, we need to perform a binary labeling of the tetrahedra of $\mathcal{T}$. We say that a tetrahedron is *free-space* if we can see through, i.e. it is traversed by at least one ray of V. Otherwise, the tetrahedron is *matter*.

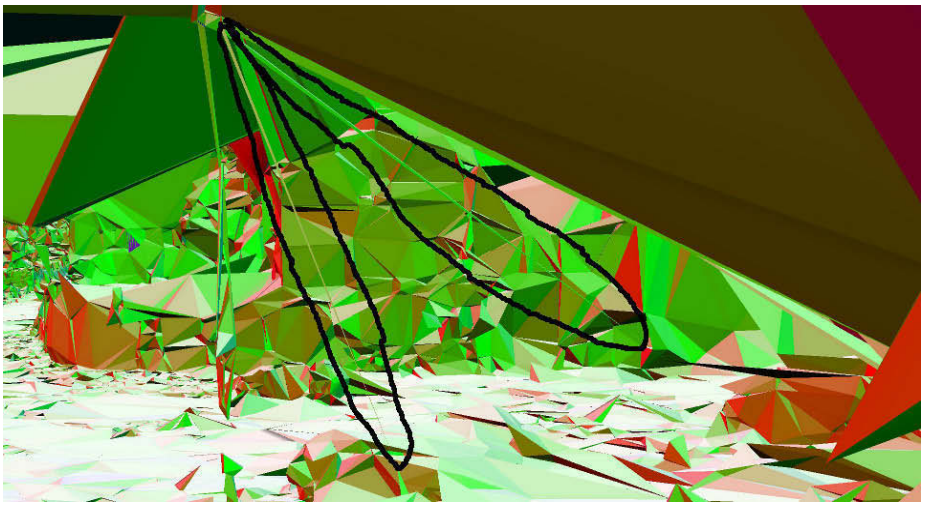
We define a function $I : \mathcal{T} \to \mathbb{R}$ that for each tetrahedron $\Delta \in \mathcal{T}$ associates the number of rays that intersect Δ. This way, if $I(\Delta) = 0$ then Δ is *matter*, otherwise it is *free-space*. We define the set of *free-space* tetrahedra $F = \{\Delta \in \mathcal{T}, I(\Delta) > 0\}$. So the problem of binary labeling becomes the problem of computing the values of $I(\Delta)$ (see figure IV.2 for an example).

In practice, we use the algorithm called *ray tracing*. In short, we follow each ray of V from the camera to the point and increment $I(\Delta)$ of each tetrahedron encountered in the way. Following a ray $\mathbf{c}_i\mathbf{p}_j \in V$ means traveling in the adjacency graph Γ_T. We begin in the tetrahedron containing $\mathbf{c}_i$, then we walk to the tetrahedron adjacent by the face intersected by the segment $\mathbf{c}_i\mathbf{p}_j$ and so on. We stop when the tetrahedron containing $\mathbf{p}_j$ as one of its vertices is reached.

As can be noted, the rays are followed from the camera to the 3D point. Of course, they can also be followed from the 3D point to the camera. We also suppose that the ray is always inside the convex hull of the triangulation. This is true in practice because our camera omnidirectional. More precisely, let $\mathbf{c} \in C$ and $P_{\mathbf{c}} = \{\mathbf{p}_i, \mathbf{cp}_i \in V\}$ is the set of points observed by the camera $\mathbf{c}$. Because the camera is omnidirectional (it observes the points in all directions around it), the point $\mathbf{c}$ is located somewhere in the convex hull of the point cloud $P_{\mathbf{c}}$. Then, it is included



(a) Example of an acute *matter* tetrahedron in the middle of the *free-space* (2D case). White triangles are *free-space* and gray triangles are *matter*. Thick black line is the longest edge e_l and α is the angle inferior to ϵ_{acute}.



(b) Real example of acute tetrahedra in the middle of the *free-space* region. The artifacts are encircled by black traits.The illustration shows the border of *free-space* region (∂F) computed by processing the *aubiere* sequence (see chapter V). The colors encode the triangles normals.

Figure IV.3: The illustration of the kinds of artifacts handled by the acute tetrahedra removal post-processing step.

in the convex hull of T.

This kind of algorithms can easily be converted to the parallel processing, a curious reader can refer himself to the appendix A for details.

IV.1.3 Acute tetrahedra removal

Before performing the 2-manifold generation step, the binary labeling is enhanced using the acute tetrahedra removal heuristic. This step is an enhancement compared to [Lhuillier13].

The case of the acute tetrahedron is illustrated on the figure IV.3. If one of the angles formed by the edges of a tetrahedron is very acute and at the same time one of the edges of the tetrahedron is long, the area of the corresponding tetrahedron facet is small. This means that the probability of the tetrahedron to be intersected by a ray is diminished. If such a tetrahedron is traversing a large *free-space* region, it has a high probability to remain *matter* and disturb the 2-manifold generation.

In practice, we proceed by checking each *matter* tetrahedron $\Delta \in \mathcal{T}$. We search the longest edge e_l of Δ and check the four angles adjacent to e_l. If one of the four angles is inferior to a user defined threshold ϵ_{acute}, the tetrahedron Δ is considered

acute. For example, on the figure IV.3a, the tetrahedron is considered acute if $\alpha < \epsilon_{acute}$. If Δ is acute, we check the status of the other tetrahedra adjacent to its longest edge. If one of these tetrahedra is *free-space*, the $I(\Delta)$ is forced to 1 and so Δ become *free-space*.

IV.1.4 2-Manifold definition

A *2-Manifold* is a particular case of a *topological space* [Moise97]. The *topological space* is defined as follow:

Definition IV.6 (Topological space)
Let X be a set and Y be a set of subsets of X. The pair (X, Y) is a topological space *if*

- $\emptyset \in Y$ *and* $X \in Y$*;*
- *Every union of elements of Y is also an element of Y;*
- *every finite intersection of elements of Y is also an element of Y.*

For each $k \in \mathbb{N}, \mathbb{R}^k$ is a topological space.

Y is the topology for X and the elements of Y are called *open sets*.

We call $d : \mathbb{R}^k \to \mathbb{R}$ the Euclidian distance in $\mathbb{R}^k$. We define an *open ball* with the center $\mathbf{c} \in \mathbb{R}^k$ and radius $r > 0$ as a set of points defined by

$$B_k(\mathbf{c}, r) = \{\mathbf{x} \in \mathbb{R}^k, d(\mathbf{x}, \mathbf{c}) < r\} \tag{IV.1}$$

The open balls of $\mathbb{R}^k$ generate a topology of $\mathbb{R}^k$, i.e. $\mathbb{R}^k$ is a *topological space*.

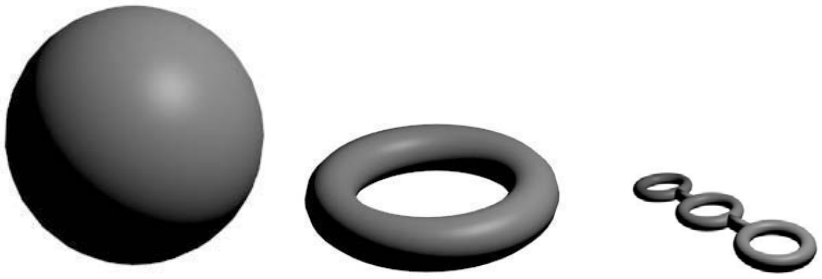
Let $M \subseteq X$ and $Y_M = \{V \cap M, V \in Y\}$. (M, Y_M) is a subspace of (X, Y) and Y_M is an induced topology of M by Y. If $X = \mathbb{R}^3$ and Y is the topology generated by d, the subspace (M, Y_M) is *2-manifold* if its dimension is 2. To mathematically formulate this property, we need the notion of *homeomorphism*:

Definition IV.7 (Homeomorphism)
Let (X_1, Y_1) and (X_2, Y_2) be two topological spaces. *The function $f : X_1 \to X_2$ is a homeomorphism between these two spaces if*

- *The function f is bijective;*
- *The function f is continuous, i.e. $\forall V \in Y_2, f^{-1}(V) \in Y_1$;*
- *The function f^{-1} is continuous, i.e. $\forall V \in Y_1, f(V) \in Y_2$.*

If it exist an homeomorphism between the spaces (X_1, Y_1) and (X_2, Y_2) they are called *homeomorphic*.

Then, the *k-manifold* is defined as follow:



(a) A sphere. Genus is 0. (b) A torus. Genus is 1. (c) A triple torus. Genus is 3.

Figure IV.4: Examples of *2-manifold* surfaces in $\mathbb{R}^3$.

Definition IV.8 (k-manifold)
Let $M \subseteq \mathbb{R}^n$ and $k \in \mathbb{N}$ such that $1 \leq k \leq n$. The topological space (M, Y_M) *is a* k-manifold *in $\mathbb{R}^n$, if every $\mathbf{x} \in M$ is contained in an open set $V \in Y_M$ such that V is homeomorphic to $B_k(\mathbf{o}, 1)$.*

In short, a *k-manifold* is a topological space that locally looks like $\mathbb{R}^k$. In our case, a *2-manifold* is a set of points in $\mathbb{R}^3$ that is locally homeomorphic to a disk. Some examples of a *2-manifold* surfaces in $\mathbb{R}^3$ are shown in the figure IV.4.

The manifolds of $\mathbb{R}^3$ are classified by their *genus*:

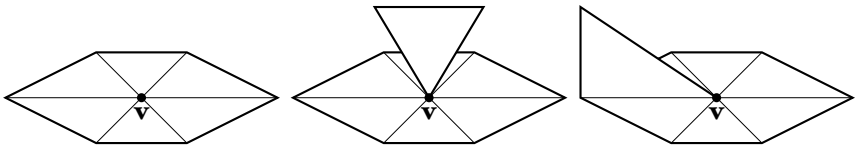
Definition IV.9 (Manifold genus)
Let M be a compact and connected 2-manifold *in $\mathbb{R}^3$. There is an unique $h \in \mathbb{N}$ such that M is homeomorphic to a 2-sphere with h handle(s). h is called the* genus *of M.*

More precisely, M is homeomorphic to

- *A sphere $\mathbb{S}^2 = \{(x, y, z)^T \in \mathbb{R}^3, x^2+y^2+z^2 = 1\}$ if $h = 0$ (see figure IV.4a);*
- *A torus $\mathbb{T}^2 = \{(x, y, z)^T \in \mathbb{R}^3, z^2 + (\sqrt{x^2 + y^2} - 1)^2 = \frac{1}{9}\}$ if $h = 1$ (see figure IV.4b);*
- *A* 2-manifold *defined by h tori $\mathbb{T}^2$ joined by $h - 1$ tubes if $h \geq 2$ (see figure IV.4c).*

IV.1.5 2-Manifold extraction

The binary labeling step previously described have computed the values of $I(\Delta)$ for each tetrahedron Δ of the 3D Delaunay triangulation T. If $I(\Delta) > 0$, the tetrahedron is called *free-space*. For memory, we define the set of *free-space* tetrahedra $F = \{\Delta \in \mathcal{T}, I(\Delta) > 0\}$. The boundary ∂F of F can be used as the first approximation of the final surface S.



(a) **v** is regular. (b) **v** is singular. (c) **v** is singular

Figure IV.5: Examples of regular and singular vertices.

Unfortunately, we have no guarantees that ∂F is *2-manifold*. Our goal is to compute another binary labeling of T (*outside*/*inside*) such as the boundary of this second labeling will be guaranteed to be *2-manifold*. We want this property because most of the surface processing algorithms have it as their applicability condition [Botsch10]. Furthermore, this property acts as a regularization constraint to search the surface.

Problem formulation

Our problem is to find a binary labeling of the tetrahedra of the 3D Delaunay triangulation T with the following characteristics:

- Tetrahedra are labeled *outside* or *inside*. We call O the set of *outside* tetrahedra.
- $O \subseteq F$. For memory, F is the set of *free-space* tetrahedra.
- The border ∂O of the *outside* region is *2-manifold*.

We want the set O to be as big as possible (ideally, we want $O = F$). In this sense, the ideal surface ∂O is the closest *2-manifold* approximation of the surface ∂F. We begin by extending the function "number of intersections" $I(\Delta)$ to a set of tetrahedra:

$$I(O) = \sum_{\Delta \in O} I(\Delta) \tag{IV.2}$$

So for the set O to be as big as possible, we want the value of $I(O)$ to be as high as possible. This way, if we can't put all the tetrahedra of *free-space* into O, we put in priority the tetrahedra with the higher number of intersections. Our goal can then be formulated as follow:

$$O = arg \left\{ \begin{array}{ll} O' \subseteq F, & max(I(O')) \\ \partial O' \text{ is 2-manifold.} \end{array} \right. \tag{IV.3}$$

From this equation, we can see that our problem is formulated as an optimization problem.

2-Manifold tests

Before discussing the optimization algorithm we need a way to check that the boundary of O is 2-manifold. We call *singular*, the vertices of T at which the manifold constraint of ∂O is violated. The other vertices are called *regular*. The figure IV.5 gives us examples of *regular* and *singular* vertices.

The detailed discussion about *2-manifold* tests is given in the appendix B. What we need to known here is that we have two manifold tests: the *fast* and the *slow* one. The fast test tells us if ∂O remains *2-manifold* after the addition of a **single** tetrahedron to O. The *slow* one is more general, it tells us if the addition of a **pack** of tetrahedra is possible without violating the manifold constraint, but, as its name tells us, it is slower.

Region growing

To compute the *outside* region as defined by the equation IV.3 we use a greedy optimization algorithm called "region growing". The basic idea is quite simple: we start from $O = \emptyset$, we choose the tetrahedron Δ of $F \setminus O$ with the higher value of $I(\Delta)$ and we check if it can be added to O without violating the *2-manifold* constraint of ∂O using the *fast* manifold test. If Δ can be added to O, it is done. Then, we proceed by trying to add another tetrahedron of F and so on.

In practice, the algorithm can start from $O = \emptyset$ or from any $O \subseteq F$ such that ∂O is *2-manifold*. The usefulness of the second starting will become clear shortly. To avoid getting stuck in local extrema, we only try to insert the tetrahedra of $F \setminus O$ directly adjacent to O (except the case of $O = \emptyset$), this way the set O grows continuously. The algorithm ends when no more tetrahedra can be added.

The details can be found in the algorithm IV.1 and an example of its application is given in the figure IV.6. The parameter Q_0 is used to give a set of tetrahedra from where the growing should begin in the case where $O \neq \emptyset$.

Limitations of the region growing

Unfortunately, the *region growing* algorithm is not always able to provide a good solution to the problem formulated by the equation IV.3. Let see the example on the figure IV.7. When the *region growing* algorithm have finished execution, there is no single tetrahedron that can be added to *outside* without breaking its *2-manifold* property. But, as clearly can be seen, there exists a pack of *free-space* tetrahedra that can be added to O and ∂O remains *manifold* nevertheless.

The problem of the *region growing* is that it can't change the *genus* of the surface ∂O and because the initial surface always have a spherical topology (we begin by a single tetrahedron), the algorithm is limited to the reconstruction of 0 genus surfaces. We need a way, once the *region growing* is stuck, to check if a *genus* change can unstuck it.

Topology extension

To unstuck the *region growing* we use another algorithm called *topology extension*. Its main idea is to try to add a pack of *free-space* tetrahedra to O when it is possible without breaking the *manifold* property of ∂O. The algorithm takes the *outside* set O as its input and returns the updated O and the seed Q_0. The seed is needed to bootstrap the new *region growing* process after the *topology extension*.

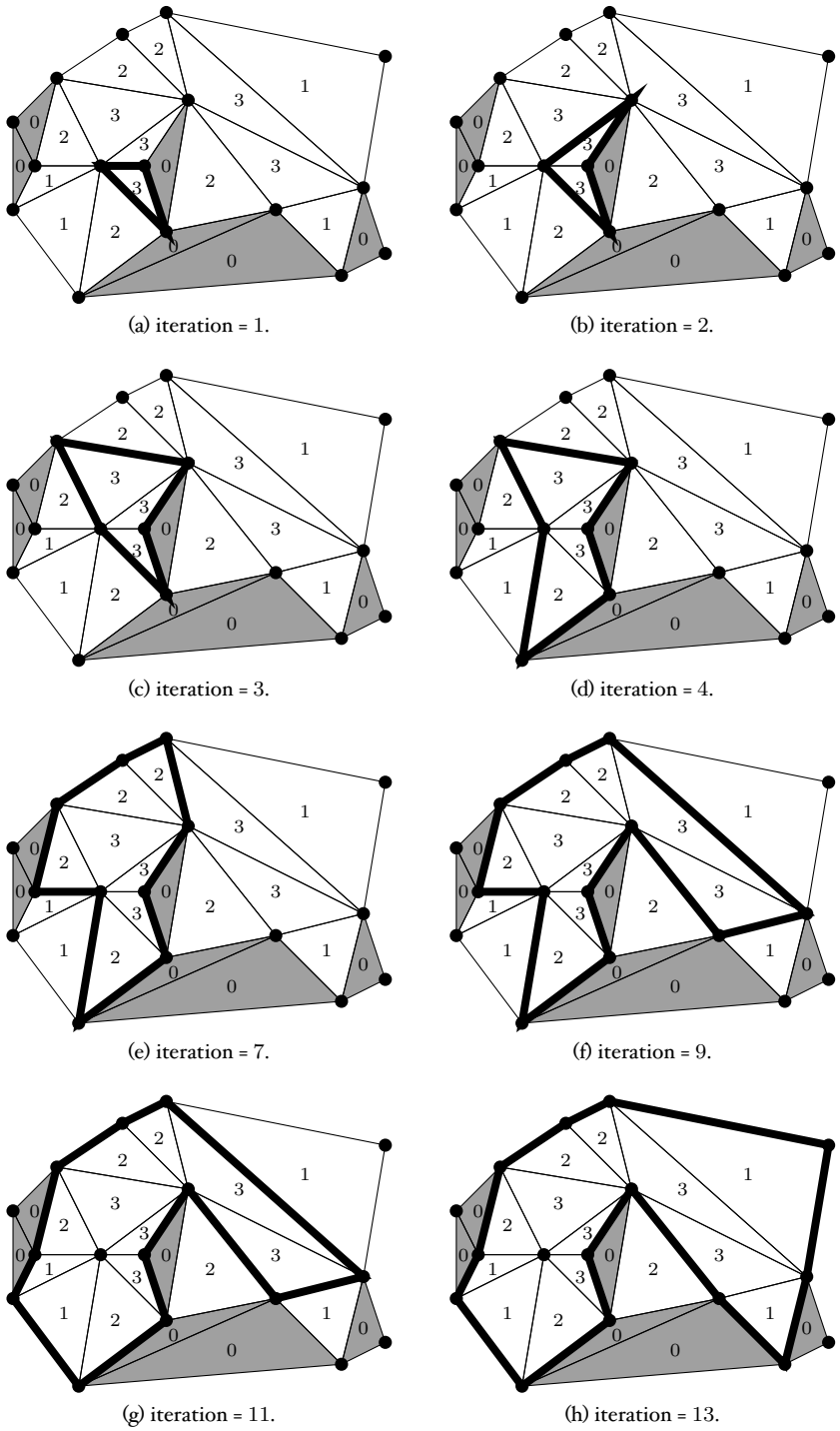
The details are given in the algorithm IV.2. The algorithm proceed by a traversal of all vertices of ∂O. For each vertex, we compute the list L_Δ of tetrahedra including this vertex, not in the *outside* region and *free-space*. The tetrahedra of L_Δ are forced to *outside*. If ∂O remains *2-manifold* (checked by the *slow* test), we have succeeded and the region growing is launched with Q_0 set to a list of neighbors of L_Δ. Otherwise, another vertex is tried.

Algorithm IV.1. The region growing

```
1:  procedure REGION_GROWING(F, O, Q_0)
2:      Let Q be a priority queue sorted by I(Δ)                ▷ Initialization
3:      Q = ∅
4:      if O = ∅ then
5:          Let Δ ∈ F such as ∀Δ' ∈ F, I(Δ') ≤ I(Δ)
6:          PUSH(Q, Δ)
7:      else
8:          for each Δ ∈ Q_0 ∩ F do
9:              if Δ ∉ O and Δ is adjacent to a tetrahedron in O then
10:                 PUSH(Q, Δ)
11:             end if
12:         end for
13:     end if

14:     while not EMPTY(Q) do                                   ▷ Region growing
15:         Δ ← POP(Q)

16:         if Δ ∉ O then
17:             O ← O ∪ {Δ}
18:             if O is 2-manifold then
19:                 for each Δ' adjacent to Δ do
20:                     if Δ' ∈ F \ O then
21:                         PUSH(Q, Δ')
22:                     end if
23:                 end for
24:             else
25:                 O ← O \ {Δ}
26:             end if
27:         end if
28:     end while
29: end procedure
```

Algorithm IV.2. The topology extension

```
1:  function TOPOLOGY_EXTENSION(F, O)
2:      for each vertex v of ∂O do
3:          L_Δ ← tetrahedra in F \ O having v as vertex

4:          O ← O ∪ L_Δ
5:          if ∂O is 2-manifold then
6:              Q_0 ← tetrahedra adjacent to the tetrahedra of L_Δ
7:              return Q_0
8:          else
9:              O ← O \ L_Δ
10:         end if
11:     end for
12: end function
```



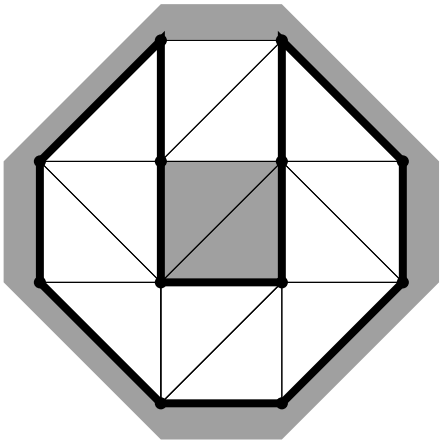
(a) iteration = 1. (b) iteration = 2.

(c) iteration = 3. (d) iteration = 4.

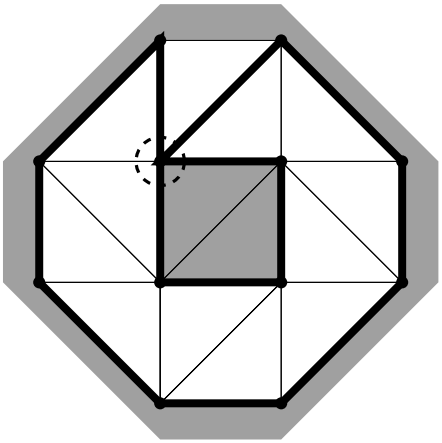
(e) iteration = 7. (f) iteration = 9.

(g) iteration = 11. (h) iteration = 13.

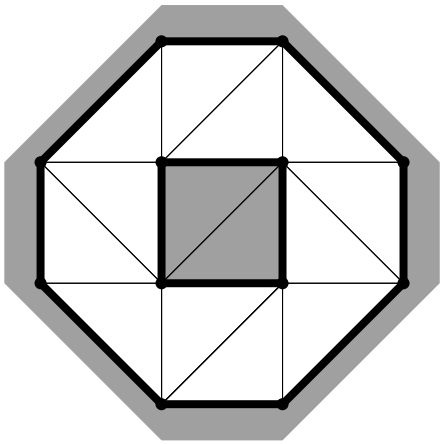
Figure IV.6: An example of the application of the region growing algorithm (2D case). Gray triangles are *matter*, the white ones are *free-space*. The numbers inside the triangles are the values of $I(\Delta)$. The thick line is the frontier of *outside* region ∂O.



(a) The triangulation after the region growing. The annulus isn't closed.



(b) If one of the two remaining *free-space* triangles is added to O, a *singular* vertex will appear (black dashed circle).



(c) If the two *free-space* triangles are added at the same time, the surface remain manifold.

Figure IV.7: An example of the region growing limitations: an annulus (2D case of a tore). Gray triangles are *matter*, the white ones are *free-space*. The thick line is the frontier of *outside* region ∂O.

In practice we alternate the *region growing* and the *topology extension* until no more *free-space* tetrahedra can be added to O. This way the reconstructed surface can have an arbitrary *genus*.

Another important remark is that the *topology extension* is slower than the *region growing*. But we was able to accelerate this step with the help of the parallel processing, see the appendix A for details.

IV.1.6 Artifacts removal

At this point, we can consider that the output surface is the boundary of the *outside* region O produced by the previous step. It is a *2-manifold* approximation of the border of the *free-space* region F and so it is a good solution to our problem. Unfortunately, if no additional steps are taken, this solution suffers from *visual artifacts*. An example of a *visual artifact* is shown on the figure IV.8. A bunch of triangles appear on the final surface, but didn't exist in reality.

These artifacts are due to the fact that sometimes the *2-manifold* growing algorithm get stuck in a local extremum. To remove them, an additional step called *artifacts removal* is performed. This algorithm can be separated into three distinct phases: detect, force and repair. We will now review each of them.

Detect

A *visual artifact* is a connected (in term of adjacency graph) set of tetrahedra A, such as $A \subseteq F \setminus O$, i.e. A is *free-space*, but included in the *inside* volume. Checking the entire triangulation T for the *visual artifacts* would be too computationally expensive. So it is preferable to detect only the *visual artifacts* easily visible by the human eye.

To achieve this goal, we define the notion of *visually critical edge*. Let e be an edge of T with end-vertices $\mathbf{e}_a$ and $\mathbf{e}_b$. e is a *visually critical edge* if the following conditions are met:

- All the tetrahedra including e are *free-space*;
- At last one of the tetrahedra including e is *inside*;
- There is a camera location $\mathbf{c}_i \in C$ such that angle $\widehat{\mathbf{e}_a\mathbf{c}_i\mathbf{e}_b} > \alpha$, where α is an user defined threshold.

All the visually critical edges of T are stored in the list L_α. The greater the threshold α, the smaller is the list L_α. The next two steps of the algorithm are applied to each edge of L_α.

Force

To enhance the probability that we would be able to exit from the local extremum of the region growing (the cause of the apparition of the visual artifact), we begin by modifying the local configuration of the tetrahedra. We need to define the notion of a *Steiner* point: this is a vertex added into the 3D Delaunay triangulation T that have no visibility information attached to it. We insert such a point into the middle of the considered critical edge e. The tetrahedra created by this insertion inherits their status (*free-space/matter* and *outside/inside*) from the initial tetrahedron

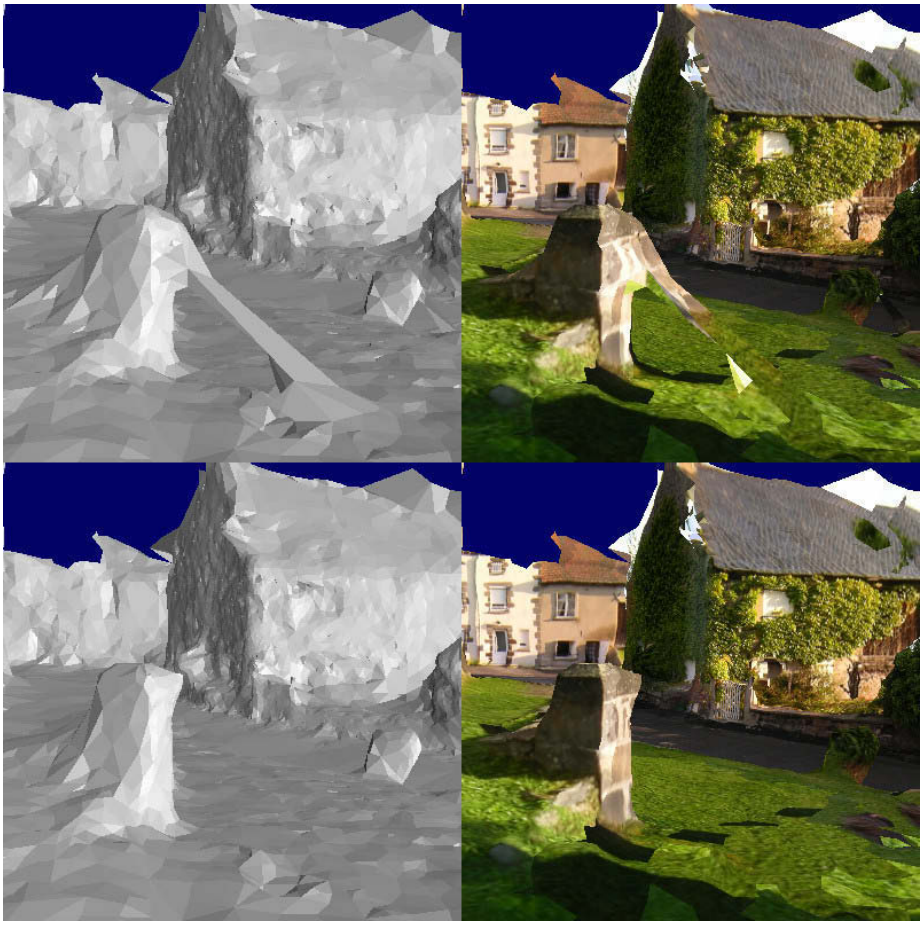
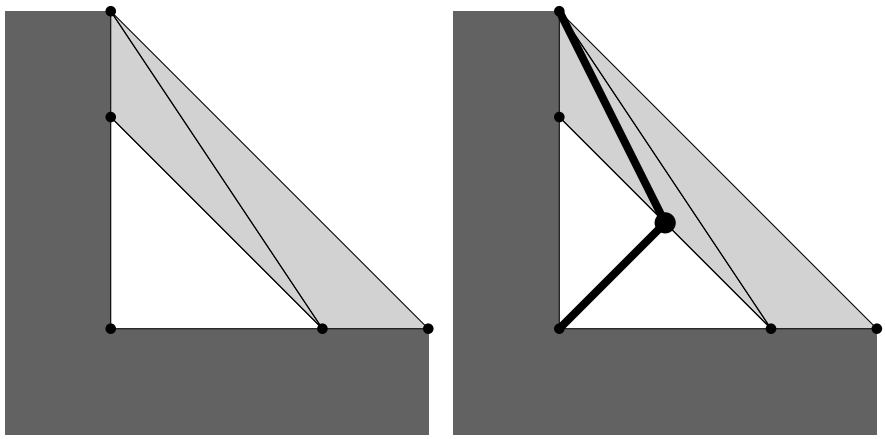


Figure IV.8: An example of a visual artifact. The first row shows the surface with an visual artifact and the second row shows the same surface with the artifact removed. The left column shows the surface with color encoded normals and the right column shows the same surface with the appropriate texture. The example is taken from [Lhuillier13].



(a) Before insertion. (b) After insertion.

Figure IV.9: An example of the edge splitting by *Steiner* vertex insertion (2D case). Dark gray triangles are *matter*. Light gray triangles are *inside free-space*. White triangles are *outside*. The big dot is the newly inserted *Steiner* point and the thick lines are the edges created by the insertion.

(see figure IV.9 for an example). Thus the resulting triangulation T is not guaranteed to be Delaunay, but the boundary of the *outside* region ∂O remains *2-manifold*. This operation is called the *edge splitting*.

We call $\mathbf{v}_s$, the *Steiner* vertex inserted by the *edge splitting* operation. So we create a list G of *free-space* and *inside* tetrahedra including the vertices $\mathbf{e}_a$, $\mathbf{e}_b$ and $\mathbf{v}_s$. Then we apply the algorithm IV.3. We force the tetrahedra of G to *outside*. This will create a certain amount of *singular* vertices n_s. So our task is to try to remove this singularities by inserting additional tetrahedra in the *outside* region. This is the *repair* step.

Repair

To remove the singularities created by the *force* step we use an algorithm similar to the *region growing* of the subsection IV.1.5, but with two key differences. First of all, we can't use the fast *2-manifold* test because the boundary ∂O is already non *manifold* at the beginning of the algorithm. Instead, we ensure that the number n_s of singular vertices decreases. The second difference is that we can't be sure that our goal ($n_s = 0$) is achievable and so the *repair* process can fail. In practice, we also limit the maximum number of iterations because of the computation time.

Otherwise, the process is mostly the same as the usual *region growing* (see algorithm IV.4 for details). We begin from the pack of *forced* tetrahedra (called G). We take a *free-space* tetrahedron Δ in the neighborhood of G and add it to the *outside* region. Then, we count a new number of singularities n. If $n > n_s$, Δ is removed from *outside*. Either way, we proceed to the next tetrahedron.

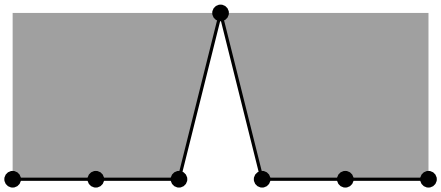
The process is stopped if either $n_s = 0$ or the maximum number of iterations g_{max} is reached.

Algorithm IV.3. The artifacts removal

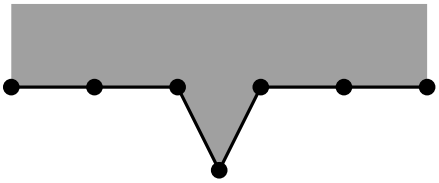
```
1:  procedure ARTIFACTS_REMOVAL(F, O, L_α)
2:      L_vert ← ∅
3:      for each e ∈ L_α do
4:          v_s ← EDGE_SPLIT(e)
5:          L_vert ← L_vert ∪ {v_s} ∪ { the two vertices of e}
6:      end for
7:      for each v ∈ L_vert do
8:          G ← the set of free-space inside tetrahedra adjacent to v
9:          O ← O ∪ G                                          ▷ Force
10:         if not REPAIR(F, O, G) then
11:             O ← O \ G
12:             for each Δ ∈ G do
13:                 O ← O ∪ {Δ}                    ▷ Force a single tetrahedron
14:                 if not REPAIR(F, O, {Δ}) then
15:                     O ← O \ {Δ}
16:                 end if
17:             end for
18:         end if
19:     end for
20: end procedure
```

Algorithm IV.4. The repair process

```
1:  function REPAIR(F, O, G)
2:      Let Q be a priority queue sorted by I(Δ)                ▷ Initialization
3:      Q ← ∅
4:      n_s ← the number of singular vertices of ∂O

5:      for each Δ ∈ G do
6:          for each Δ' adjacent to Δ do
7:              if Δ' ∈ F and Δ' ∉ O then
8:                  PUSH(Q, Δ')
9:              end if
10:         end for
11:     end for

12:     while not EMPTY(Q) do                                   ▷ Repair process
13:         Δ ← POP(Q)
14:         if Δ ∉ O then
15:             Let b_i^0 = 1, if the i-th vertex of Δ is singular, b_i^0 = 0 otherwise
16:             n_0 ← Σ_{i=1}^{4} b_i^0

17:             O ← O ∪ {Δ}
18:             Let b_i^1 = 1, if the i-th vertex of Δ is singular, b_i^1 = 0 otherwise
19:             n_1 ← Σ_{i=1}^{4} b_i^1

20:             if n_0 ≥ n_1 and ∀i ∈ [1;4], b_i^0 ≥ b_i^1 then
21:                 G ← G ∪ {Δ}
22:                 n_s ← n_s + n_1 − n_0

23:                 if the size of G is g_max then
24:                     break                                   ▷ Too many iterations
25:                 end if

26:                 for each Δ' adjacent to Δ do
27:                     if Δ' ∈ F and Δ' ∉ O then
28:                         PUSH(Q, Δ')
29:                     end if
30:                 end for
31:             else
32:                 O ← O \ {Δ}
33:             end if
34:         end if
35:     end while

36:     if n_s ≠ 0 then
37:         O ← O \ G
38:         return false
39:     end if
40:     return true
41: end function
```



(a) Example of a *outside* peak.



(b) Example of a *inside* peak.

(c) Example of a *inside* peak. Left image shows the surface computed with the peak removal enabled. The central image shows the surface without peak removal, the artifact is encircled by a black trait. The right image shows the textured surface for reference. The illustrations shows the final surface computed by processing the *aubiere* sequence (see chapter V). The colors encode the triangles normals.

Figure IV.10: The illustration of the kinds of artifacts removed by the peaks removal heuristic. White triangles are *free-space* and gray triangles are *matter*.

IV.1.7 Peak removal

The peaks removal problem is illustrated on the figure IV.10. The problem arises from the fact that the 3D points generated by the *SfM* algorithm are not guaranteed to be correct. If a false 3D point is located behind the observed surface, it will generate a spurious concavity as on the figure IV.10a. If the density of the true points in enough, the solid angle at the apex of the spurious concavity is small and so this tetrahedron can easily be detected and forced to *inside*.

In practice, we check all the vertices of ∂O. For all vertex $\mathbf{v}$ of ∂O, we compute the list $L_\mathbf{v}$ of *outside* tetrahedra having $\mathbf{v}$ as vertex. For each tetrahedron $\Delta \in L_\mathbf{v}$, we compute the solid angle α_Δ at $\mathbf{v}$. Then we compute the sum of these angles: $\alpha = \sum_{\Delta \in L_v \cap O} \alpha_\Delta$. If $\alpha < \epsilon_{peak}$ where ϵ_{peak} is a user defined threshold, the set $L_\mathbf{v}$ is a peak.

We force the tetrahedra of $L_\mathbf{v}$ to be *inside* and check that ∂O remains *2-manifold*. If it is the case, we have removed a peak. Otherwise, Δ is added back to *outside* and we try another vertex of ∂O.

The case of the *inside* peak as on the figure IV.10b is handled in a similar manner.

IV.1.8 Surface post-processing

The previous steps have established the *inside*/*outside* binary labeling. The final surface is the boundary ∂O of the *outside* region. This surface is a *2-manifold* approximation of the observed surface and so can directly be considered as the output surface of our algorithm. Nevertheless, we could perform some additional steps to

enhance the visual quality and facilitate the visualization.

The post-processing steps are the following:

- The smoothing of the final surface;
- Removing the sky triangles to facilitate the visualization;
- Computation of the triangles textures.

The smoothing step is useful because, as with any other type of acquisition hardware or reconstruction methods, the final surface taken directly is noisy. But, thanks to the *2-manifold* property, there is a lot of options to perform the smoothing. In practice, we use the uniform Laplacian flow operator [Taubin95] because of its simplicity.

Let $\mathbf{v}$ be a vertex of ∂O be the original position of the vertex and $\mathbf{v}'$ is its smoothed position. We define $\mathcal{N}(\mathbf{v}) = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\}$ the set of neighboring vertices of $\mathbf{v}$ on ∂O (i.e. there is an edge of ∂O between $\mathbf{v}$ and $\mathbf{v}_i$). We define the displacement operator Δ as follow:

$$\Delta\mathbf{v} = \frac{1}{n} \sum_{\mathbf{v}_i \in \mathcal{N}(\mathbf{v})} (\mathbf{v}_i - \mathbf{v}) \tag{IV.4}$$

Then the Laplacian flow operator is written as:

$$\mathbf{v}' = \mathbf{v} + \lambda\Delta\mathbf{v} \tag{IV.5}$$

To smooth the surface ∂O, the Laplacian operator is applied to each vertex $\mathbf{v}$ of ∂O where λ is a user defined threshold.

The sky triangles removal is useful to allow a bird view visualization of the final surface. It is also performed easily thanks to the camera positions and rotation provided by the *SfM*. As can be seen from the figure III.3 of the chapter III, the vector $\mathbf{z}_{MCS}$ defines the direction of the sky. So we begin by removing the triangles intersected by each $\mathbf{z}_{MCS}$ of each camera pose $\mathbf{c}_i \in C$. Then we remove the neighbors of the removed triangles and so on, n_{sky} times. n_{sky} is a user defined threshold. The obtained surface is a *2-manifold* with border (holes in the sky).

Finally, we could also find a texture corresponding to each triangle of ∂O. Let $t \in \partial O$ be a triangle. For each camera position $\mathbf{c}_i \in C$, we project t to the camera image. If the 2D triangle $t_{\mathbf{c}_i}$ corresponding to t is entirely contained in the image, the camera position $\mathbf{c}_i$ is added to the list of candidate cameras $L_{cam}(t)$. Then, we search the camera $\mathbf{c}_t \in L_{cam}(t)$ which maximizes the area of $t_{\mathbf{c}_i}$. The texture of $t_{\mathbf{c}_t}$ is the texture of the 3D triangle t.

IV.2 New incremental 2-manifold surface reconstruction

In the previous section, a sparse surface reconstruction algorithm issued from previous works was explained. In this section, we will details the main contribution of our work: an incremental sparse surface reconstruction algorithm. It is an incremental extension of [Lhuillier13] and was published in [Litvinov13].

The basic idea of the incremental algorithm is to locally update the output surface each time the *SfM* algorithm of the chapter III processes a new *key frame*. So, when the *key frame* I_{k+1} was processed by the *SfM*, the surface reconstruction algorithm receives as it input a new camera pose $\mathbf{c}_{k+1}$, the set of new 3D points P_{k+1}

and the associated set of rays V_{k+1}. Moreover, at this point in time, the following data structures were created by the previous iterations:

- The 3D Delaunay triangulation T_k;
- The *free-space/matter* binary labeling of T_k: F_k is the set of *free-space* tetrahedra;
- The *outside/inside* binary labeling of T_k: O_k is the set of *outside* tetrahedra; We have $O_k \subseteq F_k$ and the boundary ∂O_k of the *outside* region is *2-manifold*.

In practice, it is difficult to modify the position of a vertex inside 3D Delaunay triangulation because it can modify a lot of tetrahedra around it (for example, see [Lovi10]). So the 3D points are considered by the surface reconstruction algorithm only once they are not modified by the *SfM* any more. So, actually, the surface reconstruction process have N_{pos} iterations lag compared to the cloud of points (see chapter III), i.e. at *key frame* I_{k+1} we process the points of $P_{k+1-N_{pos}}$. But, to simplify the notations, we will consider that we process the points of P_{k+1} in the remaining of this document.

One iteration of the surface reconstruction algorithm can be separated into several steps as illustrated on the figure IV.11. This section will begin by formulating the problem of inserting the new points inside an existing 3D Delaunay triangulation (subsection IV.2.1). Then, we detail each step of the surface reconstruction iteration. Finally, we will finish this section by an explanation of how the algorithm is initialized (subsection IV.2.9).

IV.2.1 Problem formulation

Let consider a 3D point $\mathbf{q} \in P_{k+1}$. Our problem is to update the surface ∂O_k using this new point. If we insert $\mathbf{q}$ into the 3D Delaunay triangulation T_k directly, it destroys some tetrahedra and creates new ones. Let call $D_{\mathbf{q}}$ the set of tetrahedra destroyed by the insertion of the point $\mathbf{q}$ into the triangulation T_k.

Any newly created tetrahedron Δ is initialized with $I(\Delta) = 0$. So we have $F_{k+1} = F_k \setminus D_{\mathbf{q}}$ and $O_{k+1} = O_k \setminus D_{\mathbf{q}}$. More generally, when a set of 3D points P_{k+1} is inserted into T_k, we call D the list of destroyed tetrahedra. So we have $F_{k+1} = F_k \setminus D$ and $O_{k+1} = O_k \setminus D$.

There is no problem for F, but the boundary ∂O of the *outside* region must be *2-manifold* at all times. However, we have no such guarantees for the boundary ∂O_{k+1} because arbitrary suppression of some of the tetrahedra can lead to singularities (see sub-figure IV.11f for an example).

So, our problem is to find a way to insert new points inside Delaunay triangulation without disturbing the *outside* region O_k.

IV.2.2 Enclosing destroyed tetrahedra

The easiest way to solve this problem is to modify the *outside* region O_k in a controlled manner (ensuring that ∂O_k remains *manifold*) in such a way that $O_k \cap D = \emptyset$ without inserting the points of P_{k+1} into the triangulation.

In practice, we prefer to work with a set E such as $D \subseteq E$. We want E to be as small as possible, but at the same time we want the boundary of ∂E to be as "smooth" as possible (i.e. like a ball, not a star). The later property is needed to ensure that it would be easy to shrink the *outside*.

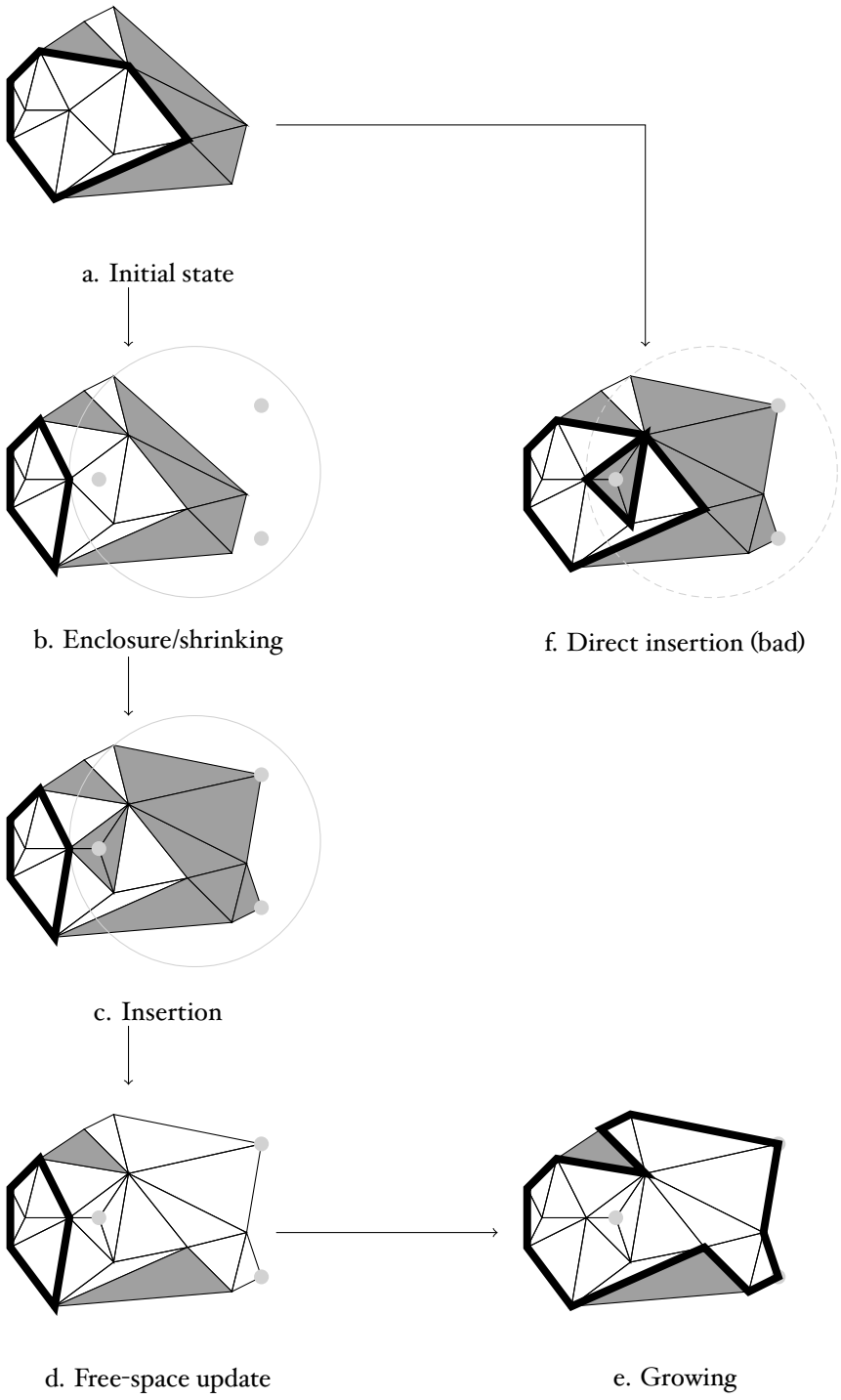
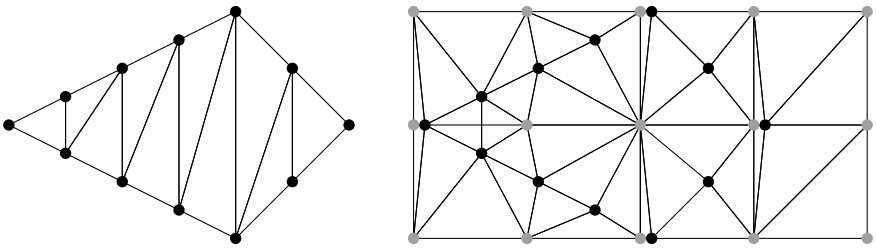


Figure IV.11: An overview of the incremental surface reconstruction method (2D case). White triangles are *free-space*, gray triangles are *matter*. Thick black line is the frontier of the *outside* region. Light gray dots are the new points to insert and the light gray circle is the enclosing sphere E.



(a) Triangulation without Steiner vertices.

(b) The same triangulation with a regular grid of Steiner vertices.

Figure IV.12: Bounding the size of tetrahedra using a grid of Steiner points (2D case). Black dots are reconstructed vertices and the gray dots are Steiner vertices.

First of all, because we want E to be as small as possible we need a way to limit the maximum size of a tetrahedron. Inside a usual Delaunay triangulation, the tetrahedra can be extremely large. If such a tetrahedron comes to be destroyed by an insertion of a new point, the set E will be big. To avoid this kind of problems, we introduce a regular grid of *Steiner* vertices.

As was defined in the subsection IV.1.6, a *Steiner* vertex is a 3D point introduced inside a Delaunay triangulation that has no visibility information attached to it. We introduce a regular grid of such vertices inside our Delaunay triangulation T_k in such a way that all of the points of $P = P_0 \cup \cdots \cup P_k$ are lying inside this grid. Every time a new set of points is inserted, the grid is updated and the new *Steiner* vertices are inserted as needed. The step g of the regular grid is a user defined parameter of the algorithm.

Thanks to the grid, the diameter of the circumscribing sphere of a tetrahedron is limited to $\sqrt{3}g$ (see figure IV.12 and [Lhuillier13] for proof). After the definition of the Delaunay triangulation, when a point $\mathbf{q}$ is inserted, the tetrahedra that can be modified by this insertion are those whose circumscribing sphere contains $\mathbf{q}$. So they are lying inside a ball centered at $\mathbf{q}$ with radii equal to $\sqrt{3}g$.

According to this, we can define the union of the balls centered at each point of P_{k+1} with $\sqrt{3}g$ radii: $B = \bigcup_{\mathbf{q}\in P_{k+1}} B(\mathbf{q}, \sqrt{3}g)$. A great advantage of a set defined this way is that it is easy to enclose in practice. We simply compute a tight bounding sphere of the 3D points of P_{k+1} (easy thanks to [CGA]) and add $\sqrt{3}g$ to it radii. We call $B_g(P_{k+1})$ the ball contained within this sphere. Then finally, we define E as a set of tetrahedra with at last one vertex contained in $B_g(P_{k+1})$. It is easy to see that $D \subseteq E$.

IV.2.3 Shrinking of the outside region

Now that we know the set of tetrahedra potentially destroyed by the insertion of the new points, we want to remove them from the *outside* region. More precisely, we want to compute O_{k+1} such that $O_{k+1} \subseteq O_k \setminus E$. This way we will find ourselves in the configuration of the figure IV.11b and the points of P_{k+1} could safely be inserted into the triangulation.

We begin by initializing $O_{k+1} = O_k$ and progressively remove tetrahedra from O_{k+1} in such a way that ∂O_{k+1} remains manifold until $O_{k+1} \cap E = \emptyset$. The O_{k+1} shrinking is an inverse of growing from the section IV.1.5. Let Q be the list (prior-

ity queue prioritized by $I(\Delta)$) of tetrahedra in $O_{k+1} \cap E$ which have a triangle in ∂O_{k+1}.

First we apply an ***one by one*** shrinking. We remove from Q the tetrahedron Δ which has the smallest intersection counter $I(\Delta)$. If $\Delta \notin O_{k+1}$ or Δ does not have a triangle in ∂O_{k+1}, we take another Δ in Q. Then we try to remove Δ from O_{k+1} such that ∂O_{k+1} remains ***2-manifold*** using the ***fast subtraction test*** (it is the inverse and have the same performance and limitations as the ***fast*** test. See appendix B for further details). In case of success, we add to Q the tetrahedra of $O_{k+1} \cap E$ which are adjacent to Δ. We continue until $Q = \emptyset$. This shrinking is fast thanks to the ***fast subtraction test***, but we can obtain $E \cap O_{k+1} \neq \emptyset$ (e.g. if $E \cap O_{k+1} = \emptyset$ implies that ∂O_{k+1} genus changes).

At a second time, we apply a ***shrinking by pack*** to allow genus changes. We find a vertex $\mathbf{v}$ which is both in a triangle of ∂O_{k+1} and in a tetrahedron of E, define L as the list of tetrahedra in O_{k+1} having $\mathbf{v}$ as vertex, apply $O_{k+1} \leftarrow O_{k+1} \setminus L$, and apply the ***slow*** test for ∂O_{k+1} at every vertex of L. In case of success, we redefine a list Q with the adjacent tetrahedra of L, and redo the ***one by one*** shrinking above. In case of failure, we apply $O_{k+1} \leftarrow O_{k+1} \cup L$ and try another $\mathbf{v}$. The overall process stops when we can not find a successful $\mathbf{v}$.

We have chosen to use $I(\Delta)$ as the priority criterion when choosing which tetrahedron to remove from O_{k+1}. This choice appears appropriate because this way the shrinking process is close to the inverse of the growing and so we can hope it will be efficient. Nevertheless, other choices of the priority criterion are possible. They will be compared in the experimental study in the section V.7.

Another important remark is that we can't theoretically guarantee that this algorithm will lead to $O_{k+1} \cap E = \emptyset$. It is possible that some tetrahedra of E will remain in O_{k+1}. We hope that this case rarely occurs in practice and we will explain in the next subsection how to process these border cases.

IV.2.4 New points insertion

Now, we find ourselves in the configuration of the figure IV.11b and so we can proceed to the insertion of the points of P_{k+1} into the Delaunay triangulation. We initialize $T_{k+1} = T_k$ and $F_{k+1} = F_k$. We filter the set P_{k+1} to remove the points with bad accuracy in the same manner as in the subsection IV.1.1. Then for each point $\mathbf{q} \in P_{k+1}$ we apply the steps that follow.

Because we are unsure that $O_{k+1} \cap E = \emptyset$, we begin by computing the list $D_{\mathbf{q}}$ of tetrahedra destroyed by the insertion of the point $\mathbf{q}$ without actually inserting it. It is easy in practice thanks to CGAL [CGA].

If $D_{\mathbf{q}} \cap O_{k+1} = \emptyset$ we add $\mathbf{q}$ to T_{k+1} (see figure IV.11c). This does not modify O_{k+1} and so ∂O_{k+1} remains ***2-manifold***. This insertion also implicitly perform $F_{k+1} \leftarrow F_{k+1} \setminus D_{\mathbf{q}}$ because for each freshly created tetrahedron Δ, $I(\Delta) = 0$ and so it is ***matter***. We still have $O_{k+1} \subseteq F_{k+1}$. We associate the creation date $k+1$ to each newly created tetrahedron, this information will be used during the next steps.

If $D_{\mathbf{q}} \cap O_{k+1} \neq \emptyset$ it would be difficult to update O_{k+1} is such a way that ∂O_{k+1} remains ***manifold*** and T_{k+1} is still a Delaunay triangulation. Since this case is rare in practice, we decide not to insert $\mathbf{q}$ in this case. When this happens, we also perform $P_{k+1} \leftarrow P_{k+1} \setminus \{\mathbf{q}\}$.

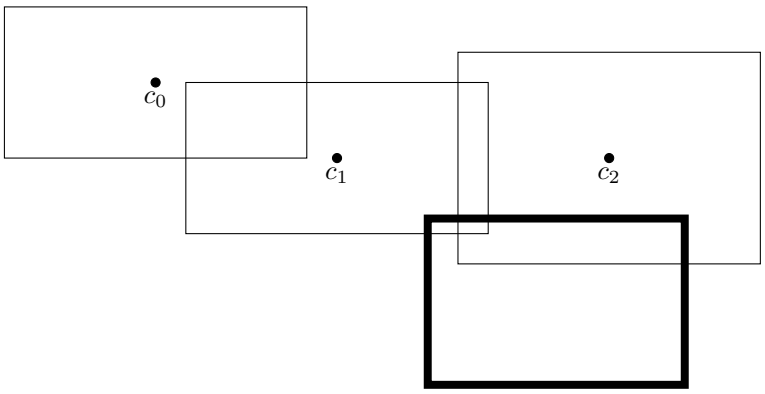


Figure IV.13: Fast rays elimination using the cameras bounding boxes illustration (2D case). Thick bounding box is B_N and the black bounding boxes are B_{c_i}. We can see that the rays originating at c_0 can be eliminated without the need to follow them.

IV.2.5 Update of free-space/matter binary labeling

After the insertion of new points we need to update the numbers of intersections of tetrahedra of T_{k+1}. First of all, the points of P_{k+1} have the associated set of rays V_{k+1}. These rays have not yet been taken into account and so they are needed to be processed. We follow each of them using the algorithm described in the subsection IV.1.2 and update the numbers of intersections accordingly.

Now, let N be a set of newly created tetrahedra (i.e. the set of tetrahedra with $k+1$ as their creation time). We need to check the intersection of each of these tetrahedra with the rays generated by the previously inserted points (the rays of V_i with $i \in \{0, \cdots, k\}$). The easy solution would be to trace all the rays and update the numbers of intersections for each intersected tetrahedron of N. Unfortunately, in practice this would be too long and, moreover, the computation time would grow with the number of processed key frames. This will be contradictory with incremental property of our algorithm.

So we need a method to rapidly eliminate the rays of V_i with $i \in \{0, \cdots, k\}$ that can't intersect the tetrahedra of N. We begin by computing the bounding box B_N of tetrahedra of N. Then, for each camera location $\mathbf{c}_i$ with $i \in \{0, \cdots, k\}$ we maintain a bounding box B_{c_i} of 3D points visible by this camera. This way all the rays originating at $\mathbf{c}_i$ are entirely included in B_{c_i}.

Now, for each $i \in \{0, \cdots, k\}$ we check the intersection of B_{c_i} with B_N. This computation is very fast. If the two bounding boxes have no intersection, all the rays originating at $\mathbf{c}_i$ are rejected (see figure IV.13). On the other hand, if the two bounding boxes have an intersection, the rays originating at $\mathbf{c}_i$ can eventually intersect the tetrahedra of N. We call V_{c_i} the set of rays originating at $\mathbf{c}_i$.

For each ray $r \in V_{c_i}$, we compute the intersection of r and B_N. Again, this computation is very fast. If the ray r didn't intersect B_N, it is rejected. Otherwise, this ray is followed using the algorithm of the subsection IV.1.2. Only the intersections of the tetrahedra of N are updated.

When this process is finished, the free-space/matter binary labeling is updated and we find ourselves in the configuration of the figure IV.11d. We also apply the

acute tetrahedra removal as describe in the subsection IV.1.3 to the tetrahedra of N.

IV.2.6 Working zone definition

Before we proceed to the next steps we need to define the notion of ***working zone*** W. We want our algorithm to be incremental, so we need to avoid working with the entire data structure. We need a way to compute the area (or zone) inside the Delaunay triangulation where the changes are taking place.

We begin from the sphere defined by the tight bounding box of the points of P_{k+1} with radii increased by $\sqrt{3}g$: $B_g(P_{k+1})$ (see subsection IV.2.2). We define W as a set of tetrahedra with at last one vertex contained within $B_g(P_{k+1})$. The set W is called the ***working zone*** in the remaining of this document. It is noteworthy that, although similar, $W \neq E$. In fact, a part of the tetrahedra of E didn't exist anymore because they was destroyed by the insertion of the new points, so we need to recompute a new set.

This definition of the working zone isn't enough because, after the subsection IV.2.3, $W \cap O_{k+1} = \emptyset$ or almost $\emptyset$. However, the ***region growing*** must begin from the neighbors of the ***outside*** tetrahedra to update the ***outside*** efficiently. So we will enlarge the ***working zone*** a little bit.

Let A be a set of tetrahedra. We define $\mathcal{N}(A)$ as the set of tetrahedra with at last one adjacent tetrahedron in A. So we add to W all the ***free-space*** tetrahedra of $\mathcal{N}(W)$. This way the ***working zone*** will contain a small part of the ***outside*** region to bootstrap the ***region growing***.

As can easily be seen, the size of the ***working zone*** is essentially determined by the step g of the regular grid of Steiner vertices. So we want it to be as small as possible. But, at the other hand, if the step is to small, the Steiner vertices will perturb the resulting surface [Lhuillier13]. The impact of the various values of g will be studied in the chapter V.

IV.2.7 Update of inside/outside binary labeling

After the ***free-space/matter*** binary labeling was updated, we update the ***inside/outside*** binary labeling by growing O_{k+1} in ***free-space*** F_{k+1} by adding tetrahedra ***one by one*** (using the ***fast manifold*** test) and ***by pack*** (using the ***general manifold*** test). As required $O_{k+1} \subseteq F_{k+1}$ and ∂O_{k+1} is ***2-manifold*** (see figure IV.11e).

This step is similar to that of the batch method in the subsection IV.1.5. The only notable differences are that we begin the ***region growing*** from the tetrahedra adjacent to the tetrahedra of $O_{k+1} \cap W$ instead of beginning from ***ex nihilo***. The other difference is that we only add the tetrahedra from $F_{k+1} \cap W$, the other ***free-space*** tetrahedra are forbidden to set an upper limit to the single iteration computation time. For memory, we re-give an overview of the algorithm here.

First, we apply an ***one by one*** growing. A priority queue Q stores the tetrahedra in $F_{k+1} \setminus O_{k+1}$ which have a triangle in ∂O_{k+1} (we initialize Q with a tetrahedron in $F_{k+1} \cap W$). At each step, Q provides tetrahedron Δ with the largest ray intersection counter $I(\Delta)$. We try to add Δ to O_{k+1} using the ***fast manifold*** test. If this is successful, the tetrahedra in $F_{k+1} \setminus O_{k+1}$ which are adjacent to Δ are added to Q. The process stops when Q is empty. This growing is fast thanks to the ***fast*** test, but it can not change the ∂O genus.

Second, we apply a ***growing by pack*** to allow genus changes. We find a vertex $\mathbf{v}$ of a tetrahedron of W and in ∂O_{k+1} such that all $\mathbf{v}$-incident tetrahedra are in F_{k+1}, and try to add to O_{k+1} those tetrahedra which are in $F_{k+1} \setminus O_{k+1}$ using the ***general*** test. If this is successful, we try to start ***one by one*** growing from these tetrahedra. The overall process stops when we can not find a successful $\mathbf{v}$.

IV.2.8 Post-processing steps

When the update of the ***inside/outside*** binary labeling is complete, the ***outside*** region O_{k+1} and the resulting surface can further be refined by some number of the post-processing steps.

First of all, the ***artifacts removal*** as described in the subsection IV.1.6 is performed. The difference with the batch case is that we can't add Steiner vertices in the middle of the ***critical edges***. The insertion of a vertex in the middle of an edge can break the Delaunay property of the triangulation. This wasn't a problem in the batch case because the triangulation wasn't modified afterward. However, in the incremental case, new points will be added to the triangulation during the next iteration and so the triangulation must remain Delaunay at all times (see subsection IV.1.1). The other difference is that the ***visually critical edges*** are only detected in the ***working zone*** W.

Secondly, the ***peaks removal*** step is performed on the tetrahedra of W. It is performed exactly as in the batch case (see subsection IV.1.7).

Finally, we can also perform some final surface ∂O_{k+1} refinement steps as in the subsection IV.1.8, namely smoothing, sky triangles removal and triangles texture computation. The only difference with the ***batch*** case is that they are applied to W instead of the entire triangulation.

IV.2.9 Algorithm initialization

The previous subsections described the different steps of an iteration of the incremental surface reconstruction algorithm. As can be seen, one of the user defined parameters of the algorithm is the regular Steiner vertices grid step g. So, one problem remain: how the user can define this value?

Setting this value directly is not an option because of the scale factor problem. The 3D cloud of points and the associated camera ***poses*** are reconstructed by the ***Structure-from-Motion*** algorithm up to a scale. So even for the exactly same scene reconstructed with different parameters of the ***SfM***, the optimal value of g can be different.

On the other hand, the distance between the camera ***poses*** of the ***key frames*** is quite constant (at last for the same type of scenes) thanks to the ***key frames*** selection algorithm. So, our solution is to define g as a multiple of the mean distance between successive camera ***poses***.

But a new problem arise: how to compute the mean distance between camera ***poses*** at the very beginning of the algorithm? The solution that we adapted is to instead of beginning the surface computation at the ***key frame*** K_0, begin it at $K_{N_{init}}$ where N_{init} is a user defined value.

Thus we have ***key frame pose*** of $K_0, K_1, \ldots, K_{N_{init}-1}$, so we can compute the mean distance between them and initialize the grid. Then, we apply the iteration of the surface reconstruction algorithm with all the 3D points reconstructed so far,

so with $P_{N_{init}} = P_0 \cup \cdots \cup P_{N_{init}-1}$ (i.e. we use the batch method with $P_{N_{init}}$ as input to initialize the algorithm). Then the algorithm can process normally.

IV.3 Time complexity analysis

In the two previous sections we have explained in details the incremental surface reconstruction algorithm proposed by this dissertation. In this section, we perform the theoretical time complexity analysis of the different steps of this algorithm. The complete time complexity analysis of the batch sparse surface reconstruction algorithm was already performed in [Lhuillier13]. We use some of these results in the complexity analysis of our own method.

The complexity analysis be performed in the worst case, but using a two sets of assumptions. The loose set is a bare minimum of assumptions which are the direct consequences of the properties of the *Structure-from-Motion* algorithm. The tight set contains more assumptions, but the results computed using this set are closer to reality.

We begin our analysis by establishing a list of properties and assumptions that we need in the subsection IV.3.1. Then we perform the complexity analysis of each step of an iteration of our surface reconstruction algorithm. The steps are not necessary treated in order but instead from easiest to the more complicated.

IV.3.1 Assumptions

We begin by establishing a list of assumptions that are used to establish the loose and tight time complexities of different steps of our algorithm. Those of these assumptions that are not trivial were proved in the sections 5 and 6 of [Lhuillier13].

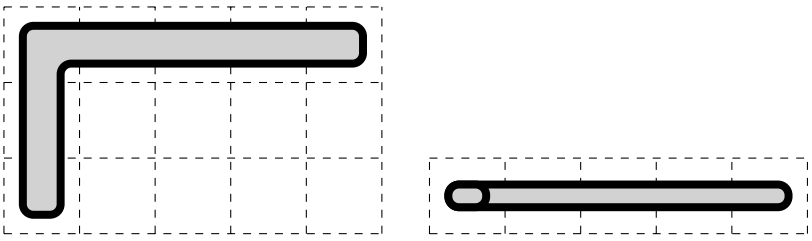
For a list L, we use $|L|$ to note the number of elements contained in this list. For the iteration corresponding to the *key frame* $k+1$ (see section IV.2 for notations), we note $|P_{k+1}| = n_{k+1}$ and d the maximum vertex degree (the maximum number of tetrahedra adjacent to a single *non infinite* vertex) of the 3D Delaunay triangulation T_{k+1}. We also use notation $N_{k+1} = \sum_{i=0}^{k} n_i$ for the total number of 3D points added to Delaunay triangulation from the beginning of the processing up to the current *key frame*.

In the loose case scenario, we use the following assumptions:

- **L1**: Let $\mathbf{q} \in P_{k+1}$. We call $V_{\mathbf{q}}$ the list of rays terminating at $\mathbf{q}$. We suppose that for all $\mathbf{q} \in P_{k+1}$, $|V_{\mathbf{q}}| = \mathcal{O}(1)$.
- **L2**: We suppose that $|P_{k+1}| = \mathcal{O}(1)$.

In the tight case scenario, the following assumptions are added to the previous ones:

- **T1**: The density of the currently reconstructed 3D cloud of points $P = P_0 \cup \cdots \cup P_{k+1}$ is bounded. More precisely, there are $m > 0$ and $n > 0$ such that every m-ball (a ball with m as radius) contains at most n 3D points.
- **T2**: Let $\mathbf{c}_k$ be the camera *pose* associated to the *key frame* k. We call V_k the list of rays originating at $\mathbf{c}_k$. We suppose that $|V_k| = \mathcal{O}(1)$.
- **T3**: For all $r \in V_{k+1}$, let $|r|$ be the length of the ray r, i.e. the number of tetrahedra intersected by r. We suppose that for all $r \in V_{k+1}, |r| = \mathcal{O}(1)$.



(a) XY projection of the triangulation. (b) YZ projection of the triangulation.

Figure IV.14: Illustration of a regular Steiner grid encompassing a horizontal trajectory. The Steiner points are located at the intersections of black dashed lines. The gray region is the *outside* and the thick black line is its border.

Moreover, we also suppose that the euclidean length of all the $r \in V_{k+1}$ is also bounded.

- **T4**: We suppose that the camera trajectory is relatively horizontal and so the regular Steiner grid have a constant number of vertical layers (see figure IV.14), i.e. the maximum height difference between the camera *poses* is inferior to the grid step g.
- **T5**: The density of tetrahedra in T_{k+1} is bounded, i.e. there are $m > 0$ and $n > 0$ such that every m-ball intersects at most n tetrahedra.
- **T6**: The maximum vertex degree meets $d = \mathcal{O}(1)$.
- **T7**: Adding a 3D point $\mathbf{q}$ to T_{k+1} has $\mathcal{O}(1)$ complexity [Faugeras90, Yu13].

The assumption T1 comes from the fact the the 3D cloud of points is reconstructed from 2D *interest points* and the textures in the observed scene are such that the density of these *interest points* is bounded. This would be false if the observed texture were fractal like, but we assume that this doesn't happens in practice.

The assumptions T5 and T6 are the consequences of the assumption T1. The reader can refer himself to the appendix D of [Lhuillier13] for proof.

The assumption L1 come from the fact that each 2D interest point is tracked in a limited number of input images. The assumptions T2 and L2 are reasonable because the number of interest points detected in a single image is constant. And finally, the assumption T3 comes from the false points filtering of the input cloud of points (see subsection IV.1.1) and T5 (proof in the appendix E.1 of [Lhuillier13]).

IV.3.2 New points insertion

First of all, the new points need to be filtered. The check of one single 3D point using the aperture angle is $\mathcal{O}(1)$ because of the assumption L1, so the complexity of the filtering is $\mathcal{O}(1)$ in the loose and tight cases (thanks to assumption L2).

First, we suppose that the regular grid of Steiner vertices doesn't need to be updated. In the loose case, the 3D Delaunay triangulation T_k contains $\mathcal{O}(k)$ vertices because, according to assumption L2, $\mathcal{O}(1)$ vertices are added to the triangulation at each iteration of the algorithm. So the triangulation contains $\mathcal{O}(k^2)$ tetrahedra. Then, the complexity of a single vertex insertion is $\mathcal{O}(k^2)$ [Boissonnat09]. So, in

the loose case, the complexity of the new points insertion is $\mathcal{O}(k^2)$. In the tight case, this complexity becomes $\mathcal{O}(1)$, thanks to the assumption T7.

If the Steiner grid needs to be updated, we usually need to add one or several rows or columns of Steiner vertices. So if we call m_{k+1} the number of Steiner vertices that need to be added to the triangulation at *key frame* $k+1$, the complexity of grid update is $\mathcal{O}(m_{k+1}^2)$ in the loose case and $\mathcal{O}(m_{k+1})$ in tight one. According to the assumption T4, we can safely suppose that $m_{k+1} = k$ because we only add one or several rows or columns and not an entire horizontal layer of vertices. So the Steiner grid update complexity is $\mathcal{O}(k^2)$ and $\mathcal{O}(k)$ in the loose and tight cases.

IV.3.3 Update of free-space/matter binary labeling

The loose case

We begin by estimating the complexity of the free-space/matter binary labeling update in the loose case. For memory, updating the free-space/matter labeling is to update the number of intersections value $I(\Delta)$ for each tetrahedron Δ of the 3D Delaunay triangulation. To achive this goal we follow each ray that need to be updated and increment the number of intersections of the encountered tetrahedra.

In the loose case, a ray intersects the totality of the tetrahedra of the triangulation [Shewchuk04]. This number can vary from $\mathcal{O}(n)$ to $\mathcal{O}(n^2)$ where n is the number of vertices [Bern94]. Thanks to the assumption L2, the Delaunay triangulation T_{k+1} contains $\mathcal{O}((k+1)^2) = \mathcal{O}(k^2)$ tetrahedra in our case. So the loose case complexity of following a single ray is $\mathcal{O}(k^2)$.

Because we are in the loose case, the totality of the rays are needed to be updated. Each 3D point has $\mathcal{O}(1)$ associated rays thanks to the assumption L1. So we need to follow $\mathcal{O}(k)$ rays. So, in the loose case scenario, the complexity of free-space/matter binary labeling is $\mathcal{O}(k^3)$.

The tight case

The tight case is more interesting. Thanks to the assumption T3, a ray intersects $\mathcal{O}(1)$ tetrahedra, so the complexity of following a single ray is $\mathcal{O}(1)$ in the tight case. We always need to follow the rays associated to the currently added 3D points. Thanks to the assumption L2, $|P_{k+1}| = \mathcal{O}(1)$, so the complexity of following the newly added rays is $\mathcal{O}(1)$.

Now we study the complexity of following the old rays. For memory, each camera *pose* has an associated bounding box B_i. This is a bounding box of the 3D points observed by this camera. It contains all the rays departing from this particular camera. We only follow these rays if the camera bounding box intersects the bounding box of the newly created tetrahedra (see subsection IV.2.5).

We want to prove that the density of the camera bounding boxes is bounded, i.e. the following lemma:

Lemma IV.1 *There are $m > 0$ and $n > 0$, such that every m-ball intersects at most n bounding boxes B_i.*

Proof. Thanks to the assumption T3, all the cameras bounding boxes B_i have bounded size. Let $m > 0$ and $\mathbf{x} \in \mathbb{R}^3$. The set L of bounding boxes B_i intersecting the m radii ball centered at $\mathbf{x}$ is included in a bounded sphere S. Thanks to the assumption T1, the number of 3D points contained in the sphere S is bounded. Or, this

number of points is larger that the number of bounding boxes contained in L, so m is bounded. □

According to the assumption T3 and the bounded size of tetrahedra (thanks to the Steiner grid), the bounding box of the newly created tetrahedra is bounded. Combined with the lemma IV.1, we see that the newly created tetrahedra bounding box intersect $\mathcal{O}(1)$ cameras bounding boxes.

According to the assumption T2, the number of rays inside a single camera bounding box is $\mathcal{O}(1)$. So we need to follow $\mathcal{O}(1)$ old rays to update the numbers of intersections. So, the complexity of the free-space/matter binary labeling update in the tight case is $\mathcal{O}(1)$.

IV.3.4 The working zone size

Before we estimate the complexities of the remaining steps of our incremental algorithm iteration, we need to estimate the size of the ***working zone*** (see subsection IV.2.6) in the loose and tight cases. For memory, the ***working zone*** is a set of tetrahedra intersecting the tight bounding sphere of the newly inserted points with radius augmented by a constant. The constant in question is the Steiner grid step.

In the loose case scenario we have no choice but to suppose that the bounding sphere of the newly inserted points contains the totality of the Delaunay triangulation T_{k+1}. So the ***working zone*** has $\mathcal{O}(k^2)$ tetrahedra.

In the tight case the bounding sphere of the points of P_{k+1} has $\mathcal{O}(1)$ radius thanks to the assumption T3. And thanks to the assumption T1, the density of the currently reconstructed 3D points is bounded. So the number of vertices in this sphere is bounded. Moreover, after the assumption T6, the maximum vertex degree is also bounded. So the number of tetrahedra in the ***working zone*** is $\mathcal{O}(1)$.

IV.3.5 Acute tetrahedra removal

We begin by estimating the complexity of the acute tetrahedra removal in the loose case scenario. This step checks each tetraheron of the working zone. The working zone contains $\mathcal{O}(k^2)$ tetrahedra in the loose case. Finding the longest edge of a tetrahedron and checking the four angles has $\mathcal{O}(1)$ complexity. Moreover, each tetrahedron have exactly four neighbors, so the complexity of checking the neighbors of a tetrahedron is $\mathcal{O}(1)$. So the loose case complexity of the acute tetrahedra removal is $\mathcal{O}(k^2)$.

In the tight case, the working zone contain $\mathcal{O}(1)$ tetrahedra. So, the complexity of the acute tetrahedra removal is $\mathcal{O}(1)$.

IV.3.6 Region growing

The region growing algorithm is the algorithm IV.1. According to the section 5.3 of [Lhuillier13] it has the following complexity:

$$\mathcal{O}(X + g(log(g) + d)) \tag{IV.6}$$

where g is the number of grown tetrahedra, i.e. the difference between $|O_{k+1}|$ before and after the algorithm. X is the size $|Q_0|$ of the set of tetrahedra where the initial tetrahedron is selected. Finally, d is the maximum vertex degree.

We begin by computing the complexity of this algorithm in the loose case. The region growing can only add the tetrahedra in the ***working zone***, so $g = \mathcal{O}(k^2)$. The

list Q_0 is the list of tetrahedra in the ***working zone*** and their immediate neighbors. So because a tetrahedron have exactly four neighbors, $X = \mathcal{O}(k^2)$. Finally, $d = \mathcal{O}(k)$ according to the appendix C. So, in the loose case, the complexity of the region growing is

$$\begin{aligned} & \mathcal{O}(k + k^2(log(k^2) + k)) \\ = \quad & \mathcal{O}(k^3) \end{aligned} \tag{IV.7}$$

In the tight case, the size of the ***working zone*** is $\mathcal{O}(1)$. So $g = \mathcal{O}(1)$ and because each tetrahedron have exactly four neighbors $X = \mathcal{O}(1)$. Moreover, thanks to the assumption T6, $d = \mathcal{O}(1)$. So, in the tight case, the complexity of the region growing is $\mathcal{O}(1)$.

IV.3.7 Topology extension

The loose case

For memory, the ***topology extension*** consists in trying for each vertex of ∂O_{k+1} in the ***working zone*** to add the neighboring tetrahedra to O_{k+1}. If we succeed, we perform a region growing from these tetrahedra, otherwise we return to the previous state and try another vertex.

The border ∂O_{k+1} contains $\mathcal{O}(k)$ vertices in the ***working zone***. Adding a pack of tetrahedra to O_{k+1} is to perform a series of ***slow*** manifold tests. The complexity of these tests is $\mathcal{O}(nd)$ in the loose case (see appendix B). We add the tetrahedra adjacent to a vertex, so we add $d = \mathcal{O}(k)$ tetrahedra. So the complexity of a single topology extension in the loose case is $\mathcal{O}(k^2)$.

If we combine the region growing and the topology extension, we perform a region growing each time a successful topology extension is performed. So, in the loose case scenario, we perform a region growing for each vertex of ∂O_{k+1} in the ***working zone***. For the i-th vertex, the complexity of the region growing is $\mathcal{O}((g_i + q_i)(log(g_i + q_i) + d))$ (see appendix E.2 of [Lhuillier13]). $\sum_i g_i = \mathcal{O}(k^2)$ because the working zone contains $\mathcal{O}(k^2)$ tetrahedra. Moreover, $\sum_i \mathcal{O}(q_i) = \sum_i \mathcal{O}(d) = \mathcal{O}(kd) = \mathcal{O}(k^2)$.

So, if we combine the computations at all vertices:

$$\begin{aligned} \sum_i (g_i + q_i)(log(g_i + q_i) + d) &\leq (\sum_i (g_i + q_i))(log(\sum_i g_i + \sum_i q_i) + d) \\ &= (\mathcal{O}(k^2) + \mathcal{O}(k^2))(log(\mathcal{O}(k^2) + \mathcal{O}(k^2)) + \mathcal{O}(k)) \\ &= \mathcal{O}(k^2)(log(\mathcal{O}(k^2)) + \mathcal{O}(k))) \\ &= \mathcal{O}(k^3) + \mathcal{O}(k^3) \\ &= \mathcal{O}(k^3) \end{aligned} \tag{IV.8}$$

To conclude, the complexity of the total growing step of an iteration of our incremental algorithm is $\mathcal{O}(k^3)$ in the loose case.

The tight case

In the tight case, the number of vertices tried by the topology extension is $\mathcal{O}(1)$ because the ***working zone*** is bounded. Moreover, the maximum vertex degree $d = \mathcal{O}(1)$ thanks to the assumption T6. So the complexity of a single topology extension is $\mathcal{O}(1)$ in the tight case.

The region growing has $\mathcal{O}(1)$ complexity in the tight case and it is performed for $\mathcal{O}(1)$ vertices, so the total complexity of the growing step of an iteration in the tight case is $\mathcal{O}(1)$.

IV.3.8 Artifacts removal

According to the subsection 6.2.2.6 of [Yu13], the complexity of the artifacts removal in the loose case is $\mathcal{O}(n^4)$ where n is the total number of vertices. Because we apply this algorithm to the *working zone*, its complexity is $\mathcal{O}(k^4)$ in the loose case.

To compute this complexity in the tight case, we will need the following lemma:

Lemma IV.2 *The number of camera* poses *that can observe a critical edge of the* working zone *is* $\mathcal{O}(1)$.

Proof. The diameter of a single tetrahedron is bounded thanks to the regular grid. This implies that the length of a critical edge **ab** is bounded.

We define a zone $Z = \{\mathbf{x} \in \mathbb{R}^3, \widehat{\mathbf{axb}} > \epsilon \text{ where } \epsilon > 0\}$. Because the length of **ab** is bounded, the size of Z is bounded. Thus, zone Z is covered by a finite number of m-balls. According to lemma IV.1, Z is intersected by a finite number of bounding boxes B_i. So, Z contains a finite number of cameras *poses* $\mathbf{c}_i$. So the critical edges of the *working zone* are observed by a finite number of cameras.□

In the tight case, the complexity of the artifacts removal is $\mathcal{O}(nm)$ (subsection 6.3.2.5) where m is the number of camera poses. In our case, $n = \mathcal{O}(1)$. Moreover, m is also $\mathcal{O}(1)$ since we only consider the cameras that can observe the edges of the working zone and this number is limited thanks to the lemma IV.2. So the complexity in the tight case is $\mathcal{O}(1)$.

IV.3.9 Remaining steps

To compute the tetrahedra that need to be removed from *outside* region before inserting the new points, we use an algorithm that is almost the same as the actual insertion. So the enclosure time complexities are the same as for the insertion step, i.e. $\mathcal{O}(k^2)$ and $\mathcal{O}(1)$ in the loose and tight cases.

The shrinking algorithm is the inverse of the growing algorithm and so the shrinking step has the same complexities as growing. So the shrinking step has $\mathcal{O}(k^3)$ and $\mathcal{O}(1)$ complexities in the loose and tight cases.

The peaks removal has the complexities $\mathcal{O}(n^3)$ and $\mathcal{O}(n)$ in the loose and tight cases according to [Yu13]. Since we apply the peaks removal on the tetrahedra of the *working zone*, its complexities are $\mathcal{O}(k^3)$ and $\mathcal{O}(1)$ in the loose and tight cases.

IV.3.10 Conclusion

To conclude this section, the complexities of all the steps of an iteration of our incremental surface reconstruction algorithm in both the loose and tight cases are collected in the table IV.1. As can be seen, in the tight case, an iteration of our algorithm has the complexity of $\mathcal{O}(1)$ for all the steps except the update of the regular grid of Steiner vertices.

Fortunately, as will be seen in the next chapter, the computation time of the grid update step is small compared to the other steps of the algorithm. Moreover,

Algorithm step	Loose case	Tight case
Enclosure	$\mathcal{O}(k^2)$	$\mathcal{O}(1)$
Shrinking	$\mathcal{O}(k^3)$	$\mathcal{O}(1)$
New points insertion	$\mathcal{O}(k^3)$	$\mathcal{O}(1)$
Steiner grid update	$\mathcal{O}(k^2)$	$\mathcal{O}(k)$
Free-space/*matter* update	$\mathcal{O}(k^3)$	$\mathcal{O}(1)$
Acute tetrahedra removal	$\mathcal{O}(k^2)$	$\mathcal{O}(1)$
Growing	$\mathcal{O}(k^3)$	$\mathcal{O}(1)$
Artifacts removal	$\mathcal{O}(k^4)$	$\mathcal{O}(1)$
Picks removal	$\mathcal{O}(k^3)$	$\mathcal{O}(1)$

Table IV.1: Overview of the loose and tight cases complexities of the different steps of one iteration of the incremental surface reconstruction algorithm. k is the number of currently processed key frames.

in practice the number of Steiner vertices is very small compared to the number of *SfM* points. Thus, the complexity of this step can be considered as bounded in practice.

IV.4 Conclusion

In this chapter, we have begun by reviewing in depth the *sparse batch* surface reconstruction algorithm from [Lhuillier13]. This algorithm is an inspiration for our own work and it allowed us to more clearly explain the basic concepts that we used. Then, we have explained our own *sparse incremental* surface reconstruction algorithm producing a *2-manifold* surface. After that, we have established its complexity under loose and tight assumptions.

The execution time of an iteration of our algorithm is independent of the geometry of the camera trajectory. This is a huge advantage over our main concurrent work [Yu12] which recomputes the surface of the entire loop if the trajectory crosses itself.

In the next chapter, we will evaluate the quality and the real execution time of our algorithm.

CHAPTER V

Experimental study

The previous chapter details our incremental surface reconstruction algorithm and its complexity analysis. In this chapter we review the results of different experimentation performed using a synthetic and a real world sequences.

All the experiments in this document are performed on a machine with an Intel Core i7 processor at 3.33 GHz with 6 cores and 24 GB of RAM.

We begin by describing the video sequences that are used during this chapter in section V.1. Section V.2 reviews the results of the *Structure-from-Motion* of the chapter III. Then we study the influence of the density of the input cloud of points in section V.3, the influence of the working zone size in section V.5 and the utility of the acute tetrahedra removal in the section V.6. Finally, we experiment different tetrahedra ordering during the shrinking phase in section V.7.

V.1 Description of experimental data sets

We begin by describing the data sets or experimental video sequences that are used in our experiments. Of course, we experiment with a real world video sequence taken in a classical urban environment, but unfortunately the lack of the ground truth restricts us to a qualitative analysis. So we also experiment with a synthetic video sequence, a sequence that was generated from an existing 3D model. This way, a quantitative analysis can also be performed.

V.1.1 Synthetic data set: *synth*

To generate the synthetic video sequence used in our experiments we use a 3D model provided by the CRISTAL[1] project. This is a textured 3D model of an urban environment. It represents a part of the city center of the city of Clermont-Ferrand. The virtual buildings have the same dimensions as the real ones and the textures of the buildings are generated using the real world pictures.

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

The experimental video sequence pictures are generated using a virtual *Ladybug 2* omnidirectional camera. The *poses* of this camera forms a 621 m. closed loop. To generate each pixel of the output image, we compute the associated ray using the same calibration information as our real *Ladybug* camera. Then we find the first (relative to the camera) intersection of this ray with the 3D surface. The color of the pixel is the color of the 3D model at the intersection point.

The final video sequence contains 1553 distinct images. The 3D model used to generate the sequence as well as some representative final images can be see on the figure V.1.

V.1.2 How to compare the results with the ground truth?

To obtain quantitative results we need a way to compare the reconstructed surface with the 3D model used to generate the synthetic sequence. To achieve this result, we need to solve two problems. First, because the *SfM* algorithm of the chapter III reconstructs the cloud of points up to a scale, we need a way to map the reconstructed surface to the ground truth coordinate system. Second, we need a way to compare the two surfaces.

Let $C_g = \{\mathbf{c}_0^g, \mathbf{c}_1^g, \ldots, \mathbf{c}_N^g\}$ be the list of virtual camera *poses* used to generate the images of the synthetic sequence (N is the length of this sequence). Let $C_e = \{\mathbf{c}_0^e, \mathbf{c}_1^e, \ldots, \mathbf{c}_N^e\}$ is the list of *poses* reconstructed by the *SfM*. We estimate the similarity transformation matrix Z (7 DOF) between the reconstructed model and the ground truth using 3 points RANSAC [Fischler81]. Then the transformation is refined using the Levenberg-Maquard algorithm minimizing $E(Z) = \sum_{i=0}^{N} ||Z(\mathbf{c}_i^e) - \mathbf{c}_i^g||$.

After the reconstructed surface and the ground truth surface are in the same coordinate system, we need a way to compare them. For this, we select a random uniform distribution of points P^e on the reconstructed surface. For each point $\mathbf{p}^e \in P^e$, we search the closest (in terms of euclidean distance) point $\mathbf{p}^g$ of the ground truth surface. Then an error $e(\mathbf{p}^e) = ||\mathbf{p}^g - \mathbf{p}^e||$ is computed. If $e(\mathbf{p}^e) > \mu_0$, the matching between $\mathbf{p}^e$ and $\mathbf{p}^g$ is considered outlier, otherwise it is kept (in practice, we use $\mu_0 = 6$ m.). The distribution of e for all the points of P^e is our quantitative comparison between two surfaces.

V.1.3 Real data set: *aubiere*

To perform a realistic qualitative experiments we record a video sequence in a real world complex suburban environment: the city of Aubière where the Institut Pascal headquarters are located.

To perform the recording we use a *PointGrey Ladybug 2* omnidirectional camera. This camera provides six 1024×768 images at 15 frames/second. The camera is installed high above the experimental vehicle using a rigid pole. The installation can be seen on the figure V.2. Each of the six cameras composing the *Ladybug* has an independent auto adapting shutter, so the luminosity of the sub-images of a single *Ladybug* image can be different.

The recorded trajectory is 2.5 km. long and contains a 2.3 km. loop. The final video sequence contains 7735 images. The recorded environment contains all the classical objects found in an average suburban area: buildings, vegetation, parked cars, stairs, etc...Some representative images can be seen on the figure V.2. The recording takes place under challenging illumination conditions (bright sunshine

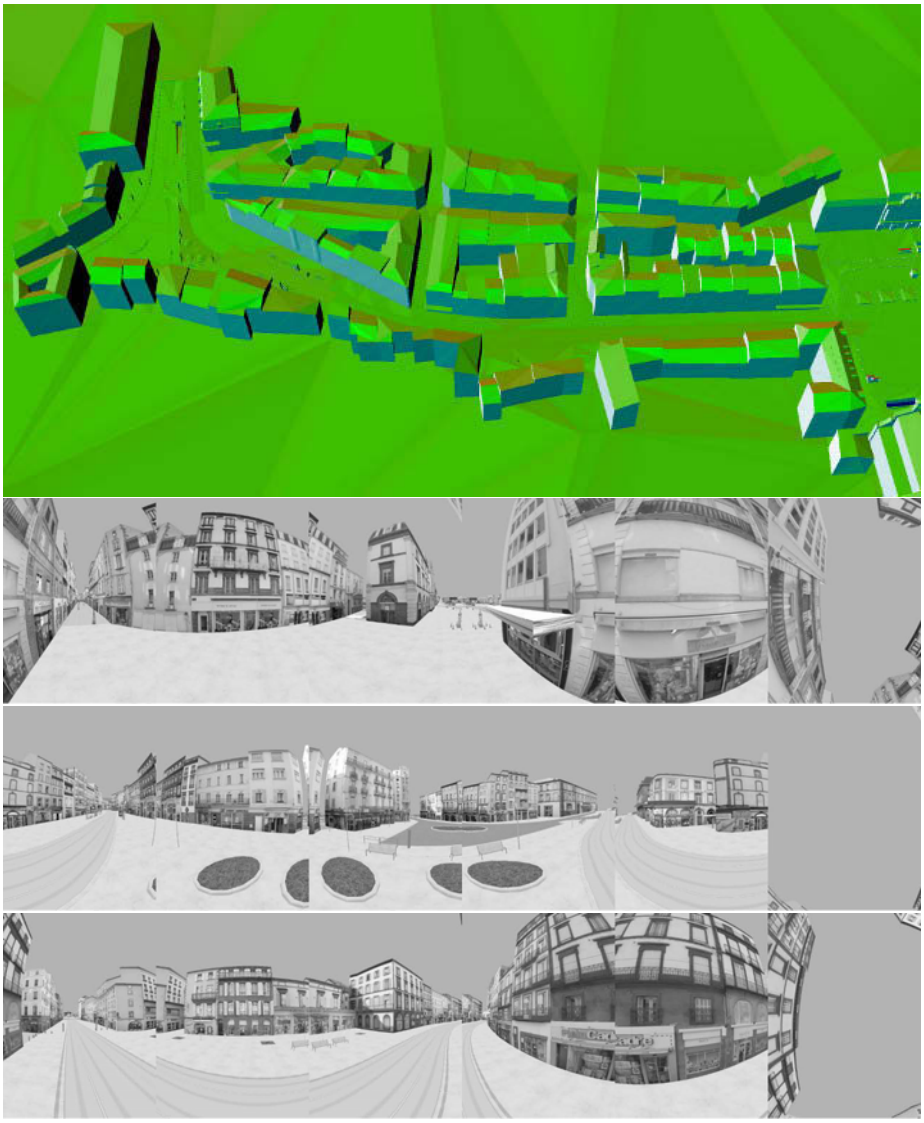


Figure V.1: The top image shows a bird view of the 3D model used to generate the synthetic video sequence. The three other images shows some representative pictures from the generated video sequence.

Figure V.2: The top image shows the experimental vehicle equipped with the *Ladybug* omnidirectional camera used to acquire the *aubiere* real world sequence. The three following images shows some representative pictures.

The set	N_{poi}	X_{roi}	Y_{roi}	w	S_m	M_2	M_3	N_{pos}	N_{obs}
div2	6000	80	120	11	0.8	1500	1000	3	10
normal	24000	160	240	11	0.8	6000	4000	3	10
poses	1250	80	120	11	0.8				

Table V.1: Summary of the sets of *SfM* parameters. See the chapter III for the notations.

and shade) and in an average traffic, so there are moving cars and pedestrians. The experimental vehicle speed isn't constant (less then 30 km./h.).

V.2 Structure-from-Motion results

The first step of our surface reconstruction algorithm is to reconstruct a 3D cloud of points with the associated visibility information from the input sequence of images. This is performed by the *Structure-from-Motion* algorithm described in the chapter III.

To study the influence of the density of the 3D point cloud and at the same time of the resolution of the input images the *SfM* algorithm was executed using three different experimental configurations: *div2*, *normal* and *extended*. The numerical values can be seen in the table V.1. The *normal* set of parameters is used when the computations are performed using the original input images directly. The *div2* set of parameters is used when the input images resolution was divided by two. Finally, we call the *extended* experimental configuration the case when the additional steps of the section III.3 are performed (the base *SfM* uses the parameters from *div2* in this case). *poses* parameters are used during the computation of the intermediary poses, the additional 3D points are computed using the parameters from *normal* set. The value of N_{poi} parameter in the case of *poses* is small compared to other sets because quite a few points is enough in practice to compute a *pose*.

During these computations, an additional step was added to the algorithm as described in this dissertation. Because of the scale factor, the positions at the end of the trajectories are not coincident with the same positions at the beginning. So when the experimental trajectories contains a loop (and this is the case) it is not properly closed. This is a known limitation of the incremental *Structure-from-Motion* and can add a significant bias to the experimental results. So we performed a single global (and so non incremental) *bundle adjustment* at the end of the *SfM* computations to close the loop. This way the experimental data are easier to interpret.

V.2.1 Synthetic data set: *synth*

First of all, we have applied the *Structure-from-Motion* algorithm to the synthetic sequence in the three experimental configurations described earlier. This way we are not only able to compare the number of produced 3D points, but also the quality of the computed camera poses.

The different numerical results can be observed in the table V.2 and the top view of the resulting clouds of points can be seen on the figure V.3. The main conclusion is that the *div2* experimental configuration is the faster one, but it produces the sparser point cloud. On the other hand, the *extended* and *normal* configuration have

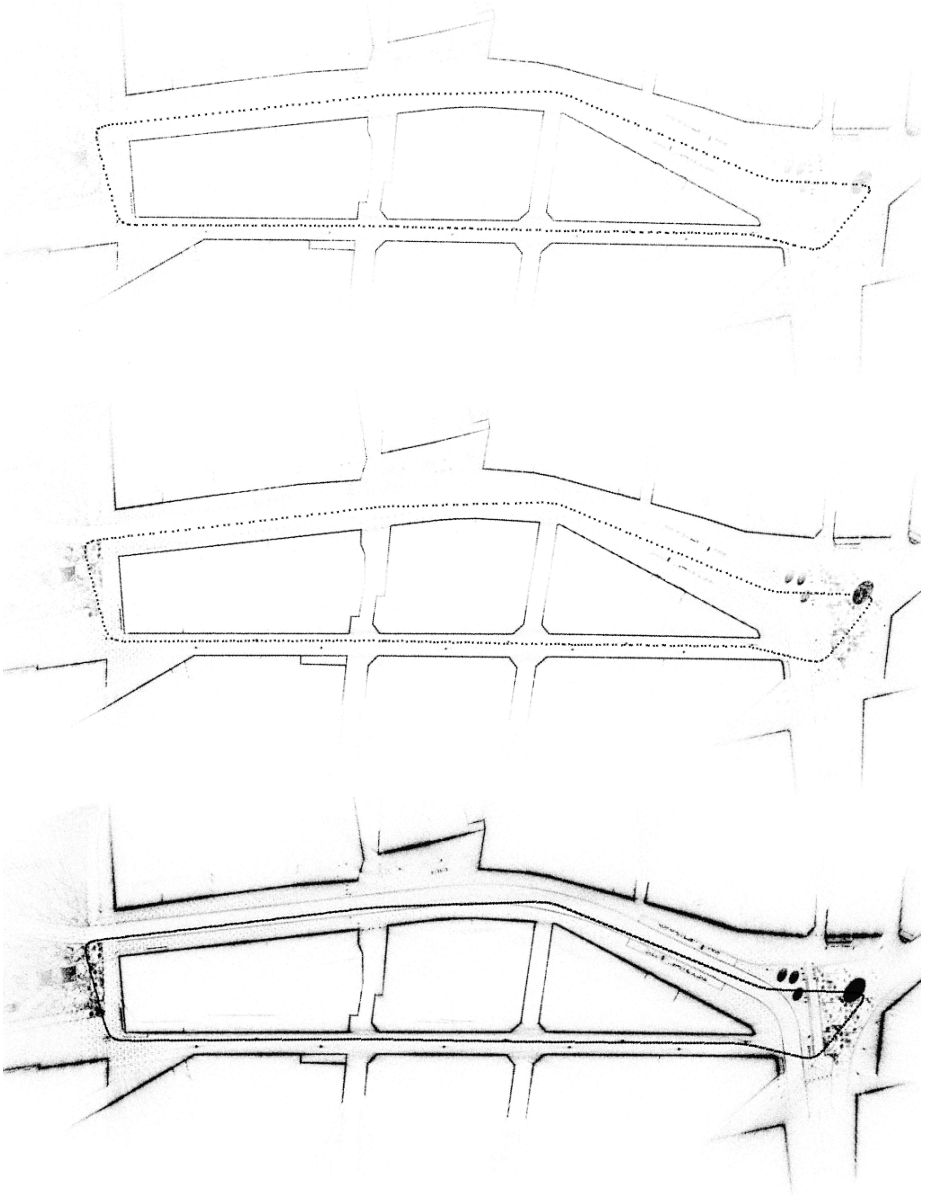


Figure V.3: The top view of the cloud of points and the camera poses produced by *SfM* algorithm from synthetic data set. The experimental configurations used to produce the clouds of points are from top to bottom: *div2*, *normal* and *extended*.

The set	Num. of key frames	Num. of 3D points	Mean time	Max time	Total time	Mean error
div2	345	143960	0.14 s.	0.96 s.	3 min. 39 s.	0.11 m.
normal	383	596801	1.09 s.	4.6 s.	28 min. 46 s.	0.19 m.
extended:	1553	2845044			28 min. 42 s.	0.12 m.
poses			0.05 s.	0.07 s.	1 min. 31 s.	
points			0.91 s.	1.49 s.	23 min. 32 s.	

Table V.2: Numerical results of the *SfM* algorithm applied to the synthetic sequence in different experimental configurations. The mean and max times are per input image. The total time is the total computation time.

The set	Num. of key frames	Num. of 3D points	Mean time	Max time	Total time
div2	1308	457368	0.12 s.	0.69 s.	16 min.
normal	1934	2243444	1.13 s.	5.46 s.	2 h. 26 min.
extended:	7508	13173793			2 h. 37 min.
poses			0.07 s.	0.51 s.	9 min. 12 s.
points			1.06 s.	1.76 s.	2 h. 12 min.

Table V.3: Numerical results of the *SfM* algorithm applied to the real world sequence in different experimental configurations. The mean and max times are per input image. The total time is the total computation time.

almost the same execution time, but *extended* produces the higher amount of 3D points.

When we compare the error between the computed and the real camera *poses* we could see that the *div2* configuration produces the best results. This could be explained by the fact that in this configuration we detect less points of interest and so the matching is easier and the final matches are of better quality (compared to *normal*). The *extended* configuration is almost as good as the *div2* because it uses the *poses* from it to localize the intermediary cameras.

V.2.2 Real data set: *aubiere*

In a second time, we have applied the *Structure-from-Motion* algorithm to our real world sequence. The numerical results produced by these experiments are summarized in the table V.3 and a top view of the 3D cloud of points produced using the *normal* experimental configuration can be seen on the figure V.4.

We come to the same conclusion as for the synthetic data set. The *div2* experimental configuration produces the less 3D points, but is the fastest. On the other side, the *extended* experimental configuration produces a lot of 3D points, but is the slowest.

Unfortunately, we have no means to evaluate the quality of the *poses* because of the lack of ground truth for this sequence.

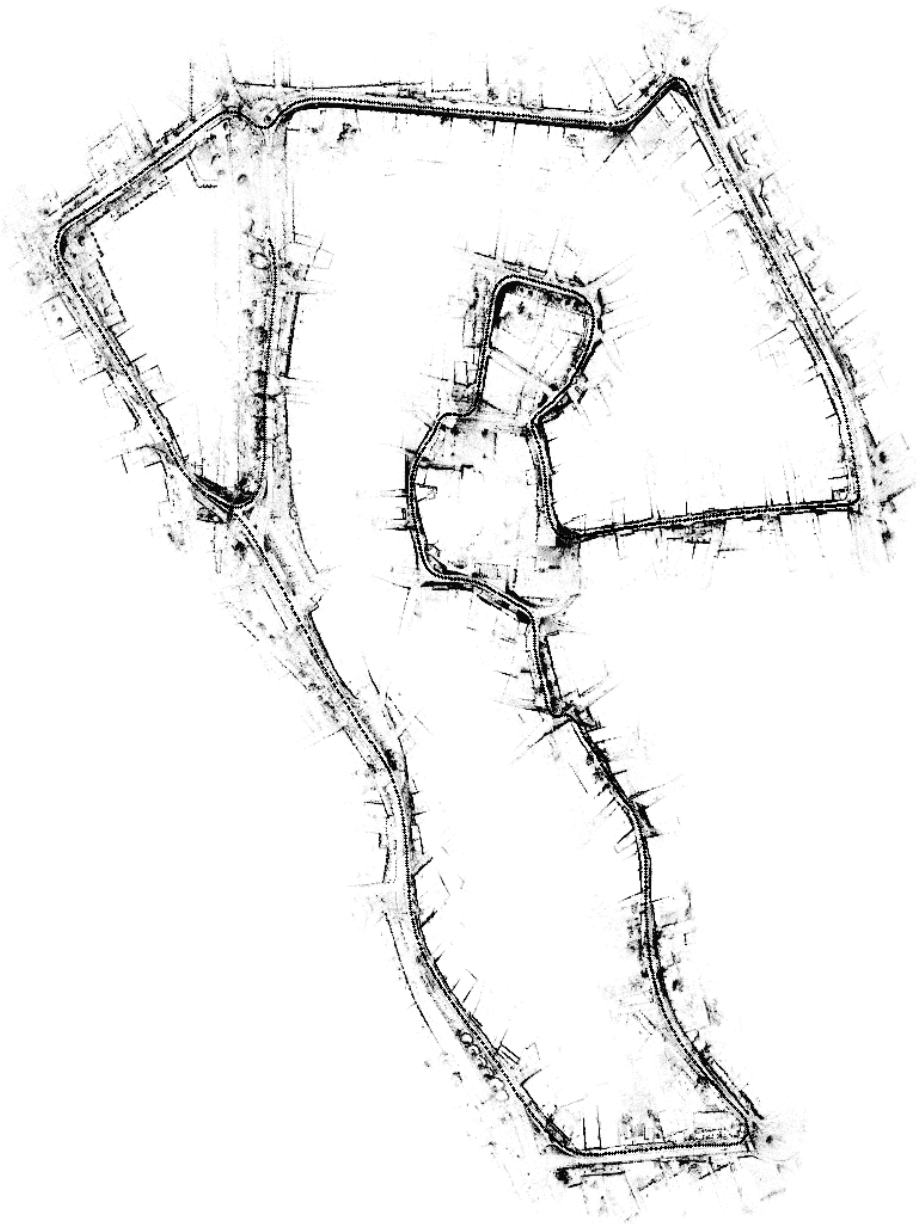


Figure V.4: The top view of the cloud of points and the camera poses produced by *SfM* algorithm from real world data set in *normal* experimental configuration.

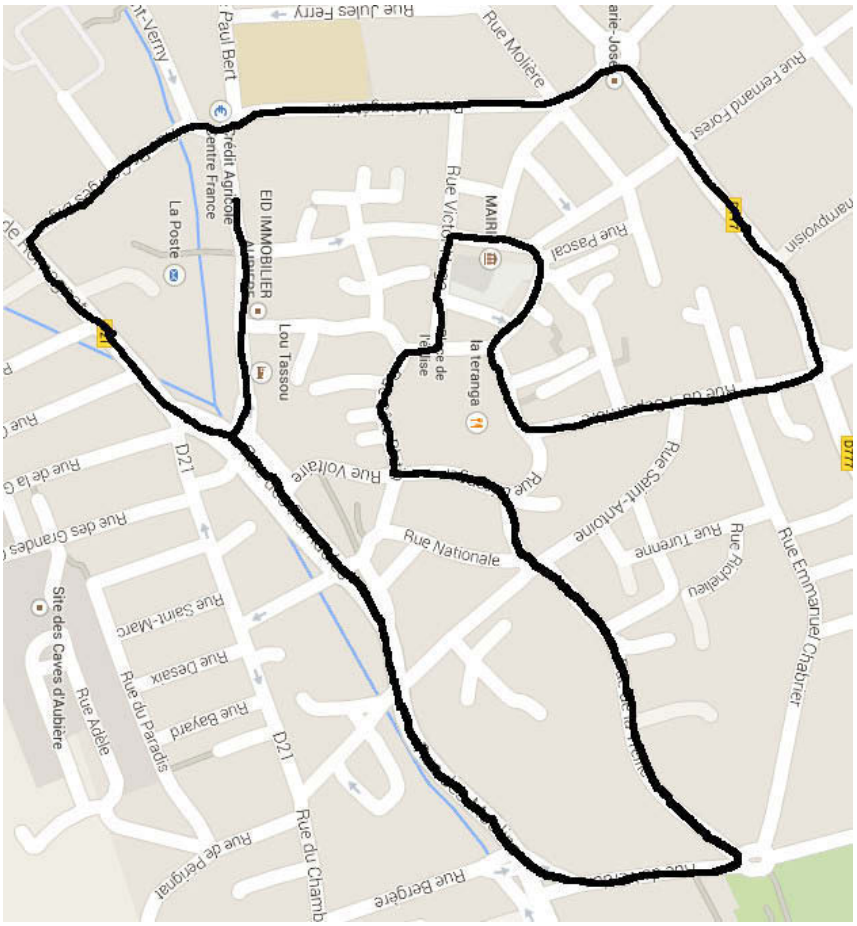


Figure V.5: The map showing the trajectory followed by the experimental vehicle during the acquisition of the *aubiere* data set.

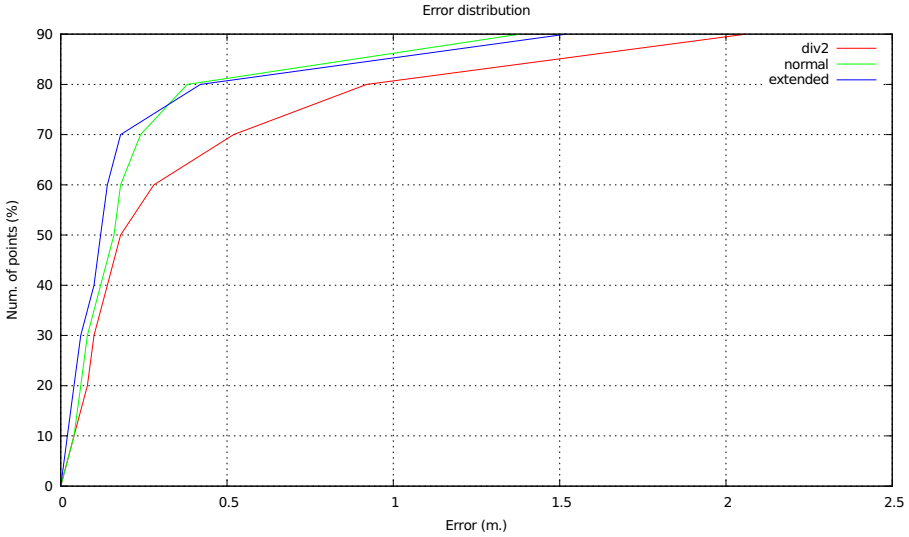


Figure V.6: Error distribution for the *synth* data set reconstructed with *batch* method from the cloud of points produced using different experimental configurations. 100% of the rays have an error inferior to 6 m.

Sequence	The set	Num. of triangles	Computation time
synth	*div2*	200750	45.51 s.
	normal	841055	2 min. 35 s.
	extended	1784896	6 min. 45 s.
aubiere	*div2*	608915	4 min. 36 s.
	normal	2674060	15 min. 24 s.
	extended	7640046	48 min. 54 s.

Table V.4: A summary of some numerical results of the batch surface reconstruction from the point clouds produced using different experimental sets.

V.3 Study of the influence of points density of the input cloud

To evaluate how the density of the input cloud of 3D points influences the quality of the output surface we have applied the batch version of the surface reconstruction algorithm to the cloud of points of the previous subsection. We performed this computation using the batch version because it is faster and the comparison results are the same in the incremental case. The number of triangles in the final surface and the computation times can be found in the table V.4.

First, we have evaluated the quality of the output surface reconstructed from the synthetic sequence using the method from subsubsection V.1.2. The distribution of errors for the three experimental set can be seen on the figure V.6. The *normal* and *extended* experimental sets performs clearly better then the *div2*. On the other hand, the difference between *normal* and *extended* is quite small. This is probably caused by two factors. First, the ground truth 3D model is simple and have few relief, the difference of results between different experimental sets is dif-

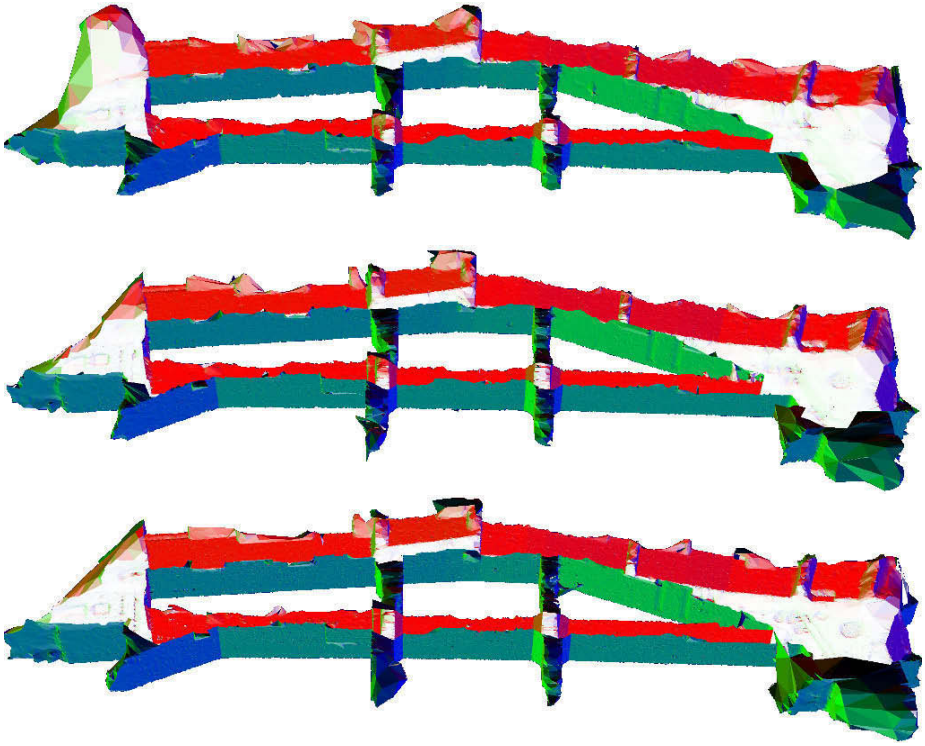


Figure V.7: The overviews of the surfaces produced by the batch algorithm from the *synth* data set. The experimental configurations used to produce the input clouds of points are from top to bottom: *div2*, *normal* and *extended*. Triangle colors encode the normals. Best viewed in color.

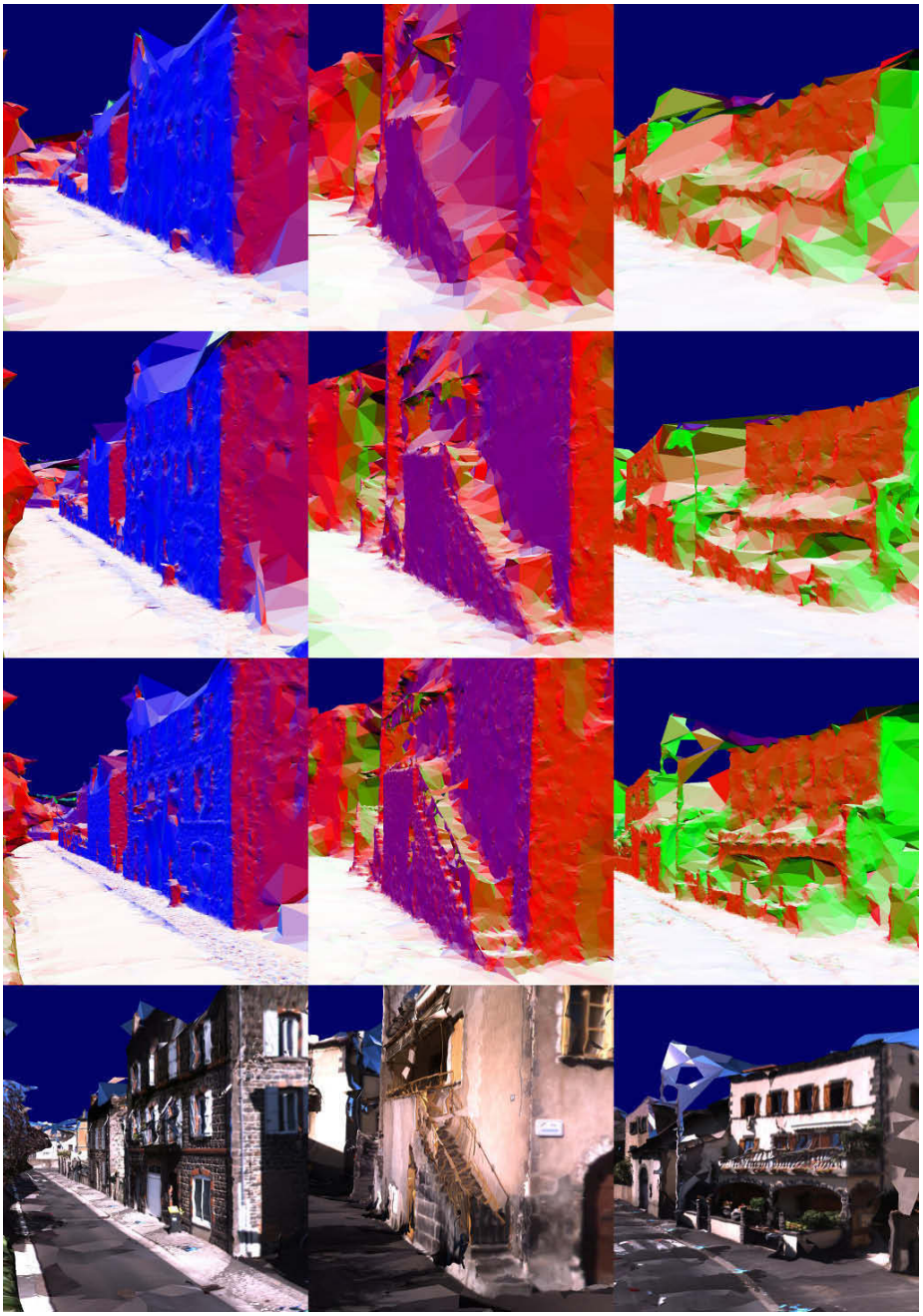


Figure V.8: Some close views of the surfaces produced by the batch algorithm from the *aubiere* data set. The experimental configurations used to produce the input clouds of points are from top to bottom: *div2*, *normal* and *extended*. The bottom view is the textured version of the *extended* surface. Triangle colors encode the normals. Best viewed in color.

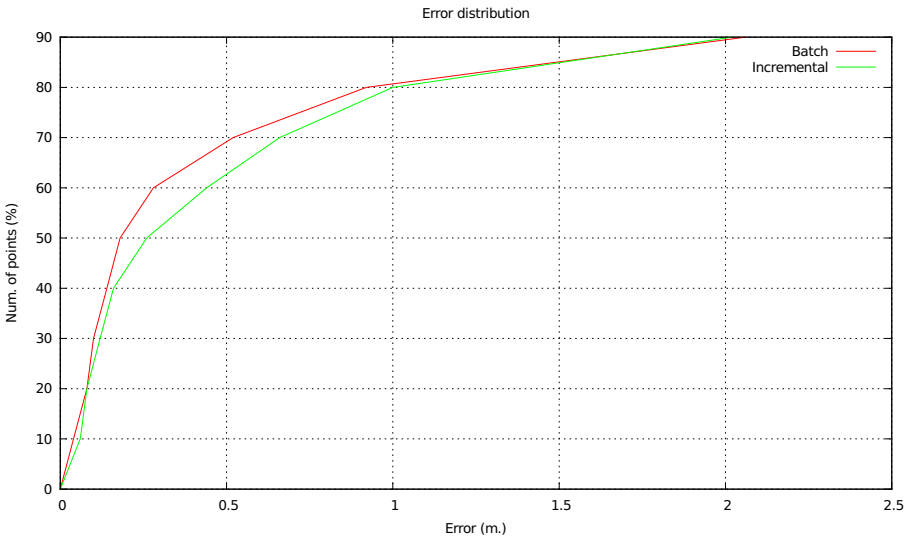


Figure V.9: Error distribution for the *synth* data set reconstructed with *batch* and *incremental* versions of the surface reconstruction algorithm. 100% of the rays have an error inferior to 6 m.

ficult to perceive as can be seen on the figure V.7. Second, the majority of errors in *normal* and *extended* cases have the same magnitude as the *poses* registration errors (see table V.2).

In a second time, we have qualitatively evaluated the results of the reconstruction algorithm applied to the *aubiere* data set. Some representative views of the final surface can be seen on the figure V.8. Here, the differences between different experimental sets are clear. *Extended* is the best one and able to reconstruct even small details of the observed surface. On the other hand, *div2* only captures a simplified shape of the objects.

The *extended* experimental set clearly provides the best results, but unfortunately it is slow. Moreover, our objective is to develop a sparse surface reconstruction algorithm, so it is interesting to perform the experiments using the fewest density of points. For these two reasons, the remaining experiments will be performed using the point clouds produced by the *div2* experimental set (as in our publications [Litvinov13, Litvinov14a]).

V.4 Comparison between batch and incremental algorithms

To evaluate the performance of the incremental surface reconstruction algorithm proposed in this dissertation, we have applied it to *div2* cloud of points from *synth* and *aubiere* sequences. Then we compared the resulting surfaces with those produced by the *batch* version of the algorithm.

First, we have compared the errors distributions for the *synth* surfaces between batch and incremental versions. The results can be seen on the figure V.9. It is easy to note that the performance of the incremental version is slightly lower that that of the batch algorithm. This can be explained by two factors. First, the artifacts

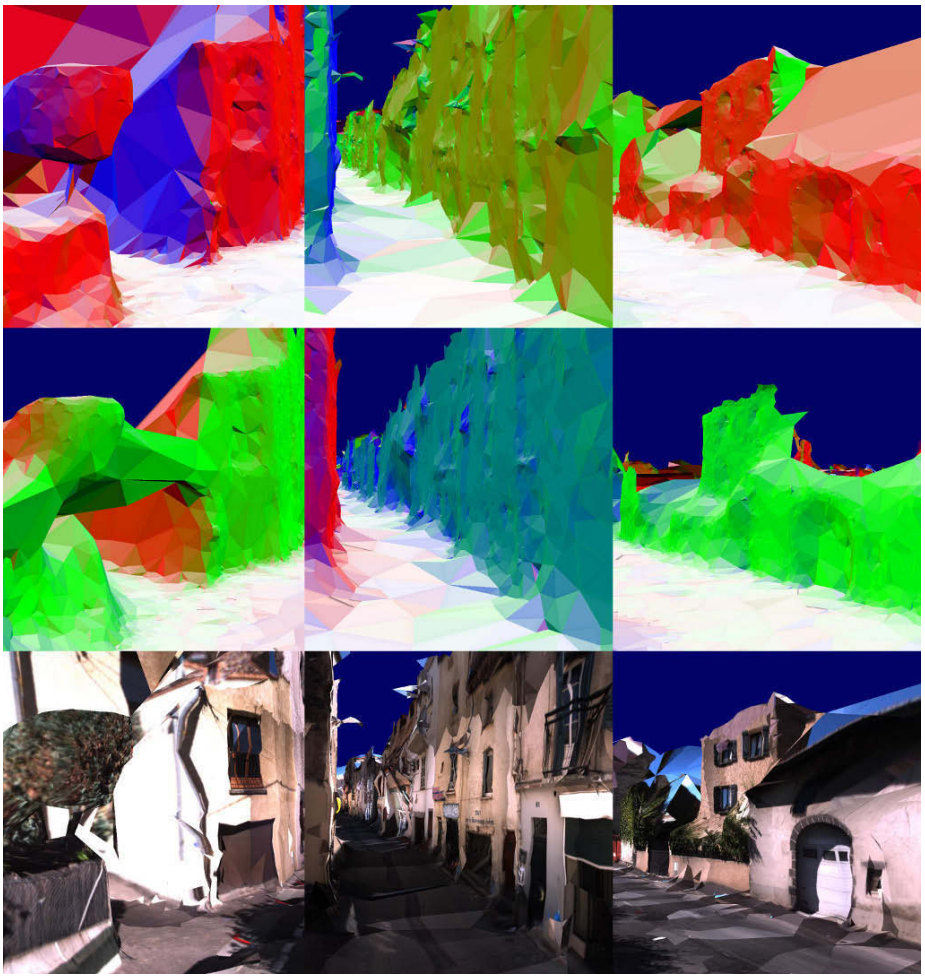


Figure V.10: Some close views of the surfaces produced by the batch and the incremental versions of the algorithm from the *aubiere* data set. The top is the batch version and the middle is the incremental version. The bottom view is the textured version of the batch reconstructed surface. Triangle colors encode the normals. Best viewed in color.

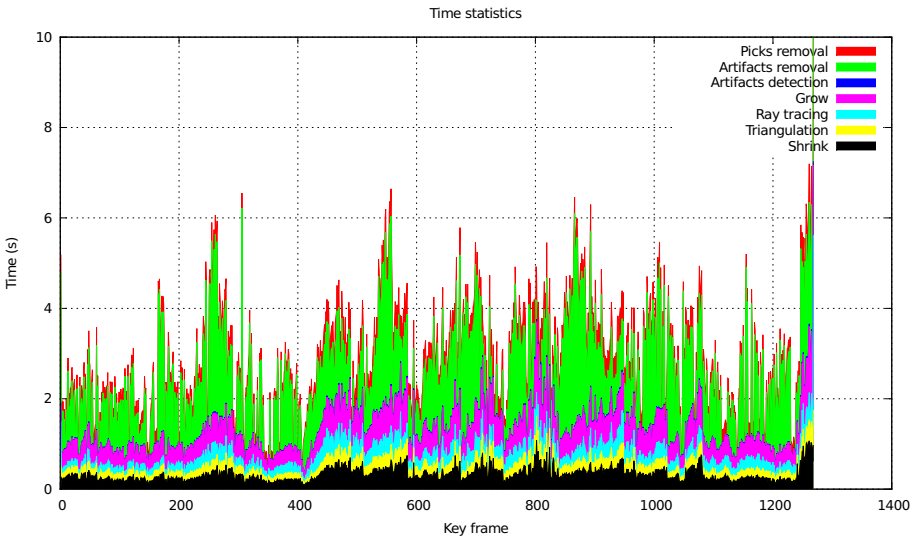


Figure V.11: Incremental algorithm execution times for each iteration when the algorithm was applied to the *aubiere* video sequence. The colored area height corresponds to the iteration execution time. The area of each color corresponds to the execution time of each step. Best viewed with colors.

removal step of the incremental version doesn't add Steiner points in the middle of the critical edges, this can significantly reduce its performance. Second, the *region growing* is artificially limited to the *working zone* in the incremental case to ensure a bounded complexity. This can however diminish the final surface quality. Nevertheless, the difference between the *batch* and *incremental* results is small.

In a second time, we have qualitatively compared the output surfaces for the *aubiere* data set. Some representative scenes can be seen on the figure V.10. As was expected, we can find some typical visual artifacts that was removed in the batch, but not in the incremental case (as in the first column of the figure). Otherwise, there are no significant differences between the two surfaces: they looks slightly different but we are unable to say which one is the best.

Third, we have compared the computation times between the batch and the incremental case. The incremental algorithm took 15 min. 13 s. to reconstruct the synthetic sequence and 58 min. 14 s. to reconstruct the *aubiere* sequence. Compared to the batch algorithm (see table V.4), the incremental version is slower by almost a factor of 10. This is explained by the fact that the distance between the key frames is small, so the incremental algorithm reconstructs the same part of the surface again and again.

We have also studied the computation times of individual iterations. They can be seen on the figure V.11. The peak at the end of the trajectory can be explained by the fact that we pass in an area where we have passed before, so the density of the *SfM* points is three times the normal. Another interesting observation is that slowest step of the processing is the artifacts removal. If we can accelerate it we could gain a significant amount of computation time. This problem is studied in the chapter VI.

Finally, we have compared the computation time of the update of the regular

grid of Steiner points. For memory, it is the only step of an iteration which is not bounded under the tight assumptions. The mean computation time of this step is 0.004 s. and the maximum time is 0.08 s. As can be seen it is negligible compared to the total computation time of an iteration. Moreover, the mean computation time of the insertion of normal points is 0.186 s. and maximal time is 0.77 s. So the update of the grid is even negligible compared to the normal points insertion.

V.5 Influence of the working zone size

The main user defined parameter of our *incremental* surface reconstruction algorithm is the step of the regular grid of Steiner vertices (g). For memory, it is expressed in the number of mean euclidean distances between successive cameras *poses*. This value is important because it indirectly defines the maximum size of a tetrahedron edge inside the triangulation and so the size of the *working zone* (see subsection IV.2.6). So we have tried different values of g to find the optimal one.

To perform the comparison we have applied the *incremental* algorithm to the cloud of points generated from the *synth* data set with *div2* set of parameters. We have applied the algorithm using different values of g, namely $2, 5, 10, 15, 20$. Then we have compared the resulting surfaces with the *ground truth* as explained in the subsection V.1.2. The results can be seen on the figure V.12. As expected, the greater the value of g, the smallest are the errors. This is the expected result since a bigger g means bigger *working zone* and the *region growing* algorithm is limited to the *working zone* to ensure a bounded iteration time. So, the bigger is the *working zone*, the more opportunities to grow has the *region growing* step.

Then we have compared the computation times for each value of g. The results can be seen on the figure V.13. In average, a bigger g means slower computations. One notable exception is the value of $g = 2$. It is often slower than $g = 5$ and even the slowest of all for the central part of the trajectory. This can be explained by the fact that, because the *working zone* is small, the *region growing* is unable to correctly fill the empty space with *outside* tetrahedra (the central part of the trajectory corresponds to a large place, see figure V.3) and this lead to a final surface having a complicated topology. This is why the value of g shouldn't be too small.

In conclusion, considering the results in terms of output surface quality and of the computation times, the value of $g = 10$ provides a good balance between the two. Higher values are significantly slower for only a slight quality enhancement.

V.6 Influence of acute tetrahedra removal

One of the secondary contributions of this work is the acute tetrahedra removal step as described in the subsection IV.1.3. This step is based on the heuristic that if the area of the tetrahedron face is small it has a few chances been intersected by a ray. If a *matter* tetrahedron with such a facet is located in the middle of the *free-space* region it is probably a *free-space* tetrahedron. So we force it to *free-space*.

To study what influence such a heuristic would have on the quality of the final surface we have applied our *incremental* surface reconstruction algorithm with and without acute tetrahedra removal step on the *aubiere* data set. The comparison between some close views of the resulting surface can be observed on the figure V.14. As could easily be noted, this new step allows to remove some big and very easily visible artifacts. This comes from the fact that the removal of the false *matter*

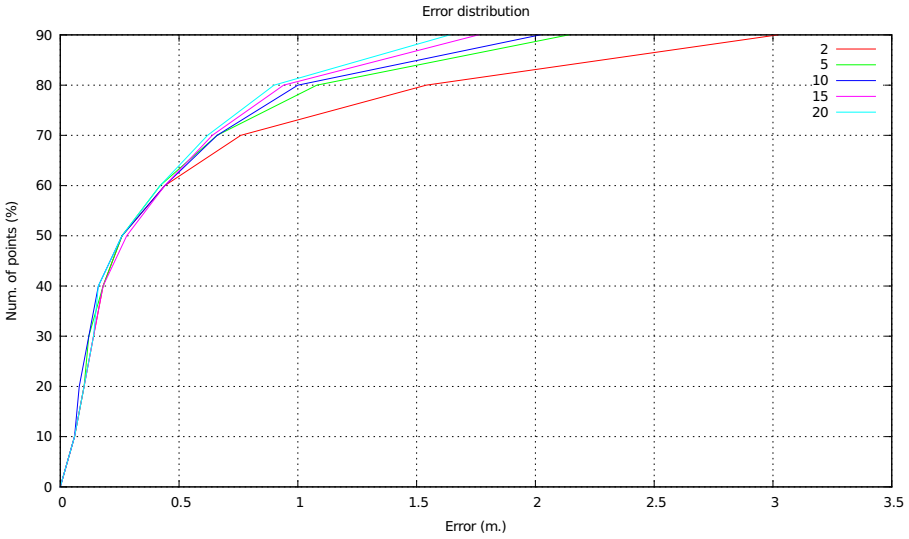


Figure V.12: Error distribution for the *synth* data set reconstructed using different values of the grid step size g (and so the working zone size). 100% of the rays have an error inferior to 6 m.

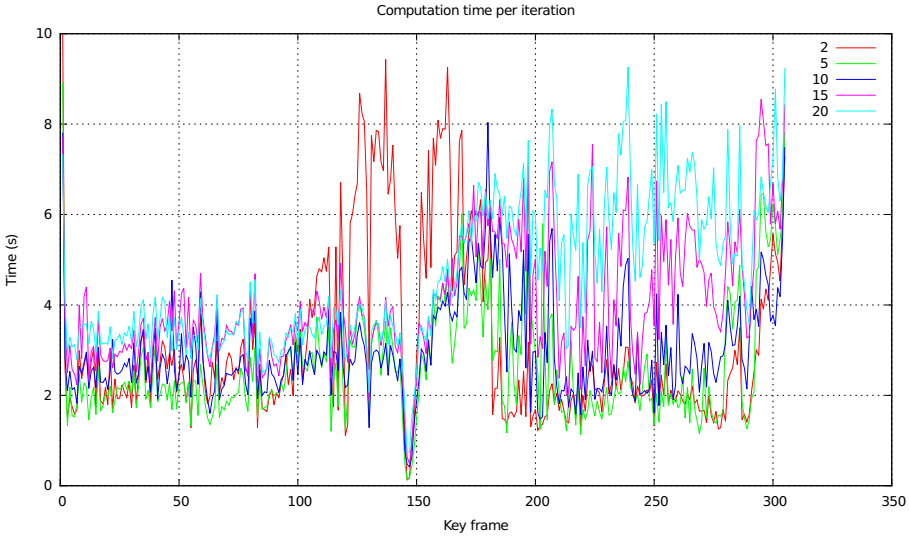


Figure V.13: One reconstruction iteration times for the *synth* data set reconstructed using different values of the grid step size g (and so the working zone size).

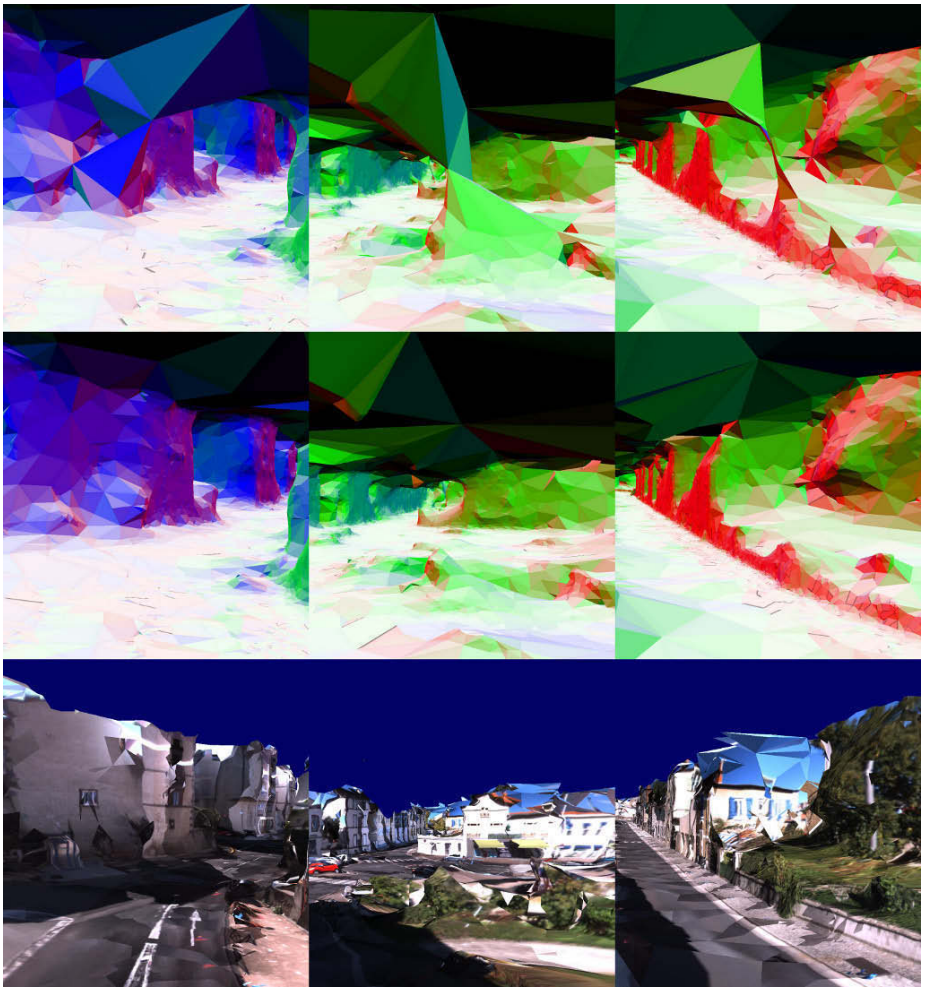


Figure V.14: Some close views of the surfaces produced by the *incremental* algorithm with and without acute tetrahedra removal from the *aubiere* data set. The top is the surface produced without acute tetrahedra removal. The middle is the same surface produced with the acute tetrahedra removal. Triangle colors encode the normals. The bottom images is the textured version of the surface for reference. Best viewed with colors.

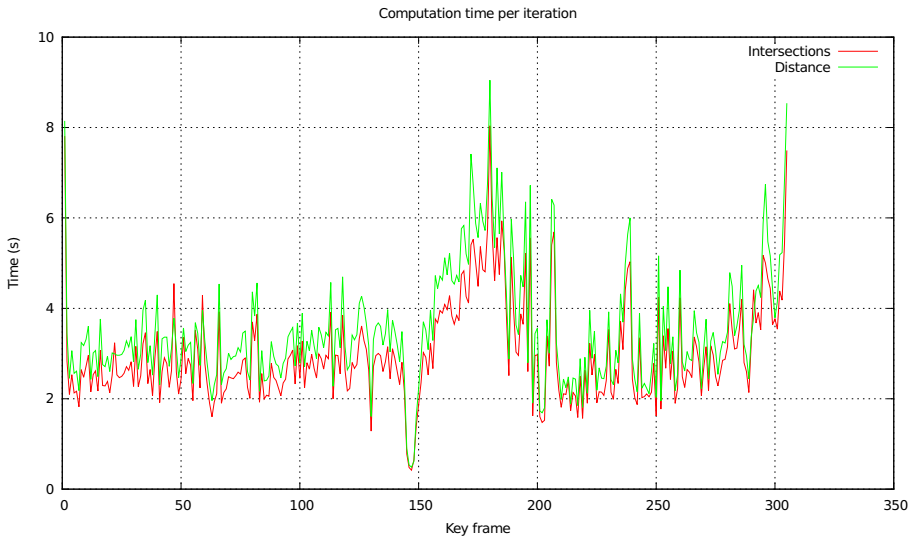


Figure V.15: One reconstruction iteration times for the *synth* data set reconstructed using different shrinking priorities.

tetrahedra allowed to the *region growing* algorithm to fill the *free-space* region more easily.

The mean execution time of the acute tetrahedra removal step is 0.004 s. and the maximum time is 0.07 s. The mean iteration time of the *incremental* algorithm is around 2 s., so we can safely conclude that the addition of this step has no influence on the overall execution time.

In conclusion, the acute tetrahedra removal step significantly enhances the output surface quality and at the same time doesn't increase the algorithm computation time, so it is an useful addition that can be used in the *batch* and *incremental* version.

V.7 Influence of the shrinking order

As was explained in the subsection IV.2.3, the *shrinking* step of an iteration of the *incremental* surface reconstruction algorithm consist in removing from the *outside* region the tetrahedra that can potentially be modified by the insertion of new 3D points inside the Delaunay triangulation. It proceed in a way similar to the *outside/inside* binary labeling. First, we try to remove the tetrahedra *one by one* using the *fast* manifold test. Then, to allow genus changes, the tetrahedra are removed *by packs*. The two steps are alternated until all the designated tetrahedra are removed or the removal of the remaining tetrahedra is impossible.

The question is: In what order the tetrahedra should be removed from the *outside* during the *one by one* phase? The obvious response to this question is in the inverse order of the number of intersections $I(\Delta)$. Not only this order is easy to implement, but it would be an inverse of the addition of the tetrahedra during the *growing* step.

Nevertheless, we could think of another *shrinking* order. For memory, the tetrahedra scheduled for removal are contained in a spherical zone (see subsection IV.2.2).

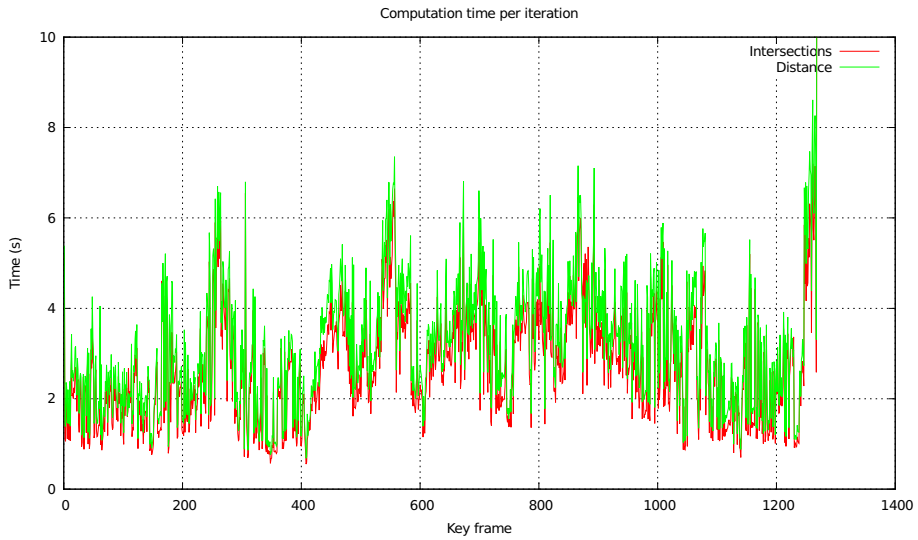


Figure V.16: One reconstruction iteration times for the *aubiere* data set reconstructed using different shrinking priorities.

We can begin by removing the tetrahedra further away from the center of the zone, i.e. the removal priority is the distance between the tetrahedra barycenter and the center of the new points bounding sphere. This way, the process is similar to onion peeling.

To compare the two possible priority and decide which one to use, we apply our *incremental* algorithm to the synthetic sequence. Then, we compare the produced surfaces with the ground truth 3D model. The quality of the output surface was identical in the two cases. The computation times per iteration can be seen on the figure V.15. We remark that the algorithm is usually slightly faster when the number of intersection is used as priority.

To confirm these results, we perform the same experiments, but this time using the *aubiere* data set. The results are the sames as in the previous case. The output surfaces are visually identical and using the number of intersections as priority is slightly faster (see figure V.16).

To further compare the tetrahedra removal efficiency of the two priorities, we have compared the number of *SfM* that was rejected because it insertion would modify an *outside* tetrahedron (see subsection IV.2.4). During the processing of the *synth* sequence only 1 point was rejected from 143960. This result was the same in the two cases. For the *aubiere* sequence, the results was 1 point out of 457368 in the two cases.

Considering these results, we conclude that the number of intersections is a good choice as priority criterion during the *shrinking* step.

V.8 Conclusion

In this chapter we have studied the performance of our incremental surface reconstruction algorithm using synthetic and real world data sets. First of all, we have

studied the influence of the input images resolution on the quality of the output surface. A bigger resolution leads to better results, but the computation time of the algorithm grows. Then, we have compared the results of the *incremental* algorithm with the results of the *batch*: the output surface quality is almost the same. We have also proved the utility of the acute tetrahedra removal and found a good value of g (step of the Steiner grid) and the *outside* shrinking order to use.

We have seen in the previous chapter that the complexity of a single iteration of our algorithm is $\mathcal{O}(k)$ under the tight assumptions. This complexity is due to the Steiner grid update, the complexity of all the other steps are bounded. However, when we observe the computation times of our algorithm applied to a real data setin the section V.4, the duration of this step is negligeable.

Unfortunately, the computation time is too slow for a real time application. This limits the practical utility of this algorithm. We reviewed the computation times for every step and see that the slowest part of the algorithm is the *artifacts removal*. If we enhance the processing time of this step, the overall algorithm will be accelerated. This is the problem studied in the next chapter.

CHAPTER VI

Study of artifacts removal

In the previous chapter, we have experimented the new incremental sparse surface reconstruction method proposed by this dissertation. Unfortunately, it has a significant drawback: as can be seen in section V.4, the method is slow. This reduces its practical interest. When we refer ourselves to the details of the computation times (see figure V.11), we can see that the slowest step in an iteration is the *artifacts removal*. So if we can reduce the time of this step, we could significantly improve the global computation time.

The *artifacts removal* step is used to remove the artifacts (hence the name) found on the final surface. These artifacts are caused by the local maxima of the objective function optimized by a greedy optimization algorithm to create the *outside* region (see subsection IV.1.5). In this chapter, we briefly review the previously proposed methods to overcome this problem and we present two new methods. Their performances are compared with that of the previous one using a real world sequence. The method presented here is published in [Litvinov14b].

We begin by defining what we seek to remove in section VI.1. Then we quickly review the previously proposed methods in section VI.2. We detail the newly proposed ones in sections VI.3 and VI.4. And finally we compare them with the previous ones in section VI.5.

VI.1 Visual artifact definition

The visual artifacts problem that we try to solve is illustrated on the figure VI.1. The artifact that connects the wall of the building to the ground on the left part of the figure does not exist in reality and should be removed as shown on the central part. This is only one example of situations when the problem arise. It can also occur in other contexts [Chaine03, Wood04, Zhou07].

Before we discuss the different *artifacts removal* methods, we should precisely define what we seek to remove. In this chapter, we use the notations of the chapter IV, more precisely of the subsection IV.1.1. For memory, the 3D Delaunay triangulation T is encoded by an adjacency graph Γ_T in which the nodes are the tetrahedra of T and the edges are the triangles between two tetrahedra. In the

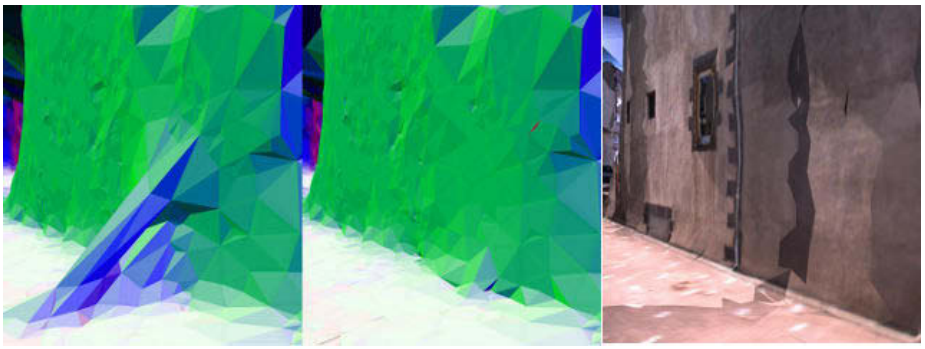
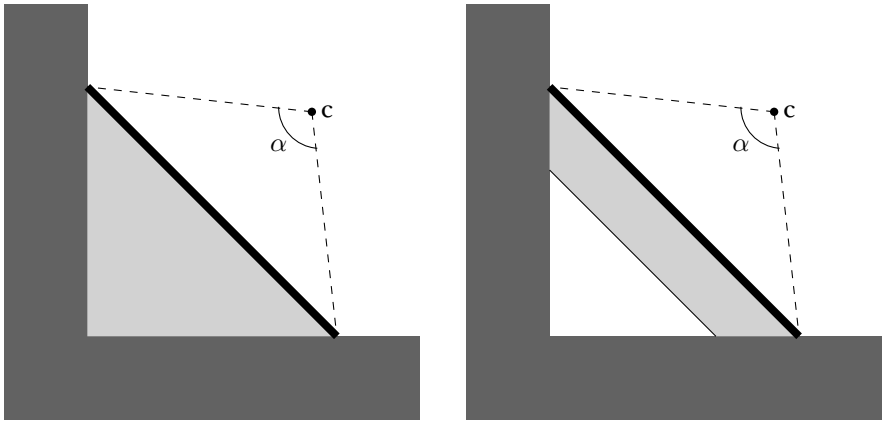


Figure VI.1: A visual artifact example. On the left image the surface contains an easily visible visual artifact. On the central image the visual artifacts was removed from the surface. Colors encode the triangles normals. On the right image the visual artifact is removed and the surface is textured.



(a) A general artifact. (b) A handle.

Figure VI.2: A schematic illustration of what is a visually critical artifact (2D case). White triangles are *free-space outside*. Light gray triangles are *free-space inside*. Gray triangles are *matter*. Thick black line is a *visually critical edge*.

same manner, for any set $S \subseteq \mathcal{T}$ of tetrahedra, $\Gamma_S \subseteq \Gamma_T$ is the corresponding adjacency graph.

We define a *visual artifact* A by $A \subseteq F \setminus O$ and Γ_A is connected, i.e. a set of tetrahedra that is included in the inside volume ($A \cap O = \emptyset$), but not in the scene matter ($A \subseteq F$). Unfortunately, detecting and removing all the artifacts in the resulting model using this definition alone would be too slow to be useful in practice. So we define a *visually critical edge* [Lhuillier13] as an edge **ab** such as $\mathbf{a} \in P, \mathbf{b} \in P$ and $\exists \mathbf{c} \in C$ such that **ab** is an edge of a tetrahedron in $F \setminus O$ and $\widehat{\mathbf{acb}} > \alpha$ where α is a user defined threshold (we use $\alpha = 5^\circ$ in practice). Then, we define a *visually critical artifact* as a *visual artifact* which has (at least) a tetrahedron containing a *visually critical edge* (see figure VI.2 for an illustration).

The artifacts removal methods discussed in this chapter seek to remove as many *visually critical artifacts* as they can. We do not use the term *spurious handle* as in [Lhuillier13], because it is too restrictive. Our algorithms deal with more than just ``*handles*''.

VI.2 Previous methods

As can be seen in the chapter II, there are multiple surface reconstruction methods based on the 3D Delaunay triangulation and *region growing*. And we are not the first who have acknowledged the problem. Several previous tentative exists to solve it.

First, there are computational geometry only approaches [Wood04, Zhou07] that use scanner data and suppress only a particular, but very usual kind of visual artifact, namely a spurious handle. They don't have our visibility information, so they need to introduce an heuristic: they consider that spurious handles are usually small by contrast to the real world handles that would be large. This works with the scanner data because the 3D cloud of points is dense. Unfortunately, this assumption is false in our case because, with the sparse input cloud of points, spurious handles, and more generally visual artifacts, can be as large as any other feature of the surface.

Another method that is very simple and more adapted to our case was first proposed in [Yu11]. This isn't exactly a *visual artifacts* removal method, but instead a way to reduce their risk of apparition. The idea is to split large tetrahedra that can be involved in a *visual artifacts* by adding some Steiner (artificial) vertices in the Delaunay triangulation T before the surface is reconstructed.

These artificial vertices are added in the region that is highly visible by the human observing the final surface: in the close neighborhood of the camera locations. For each camera *pose*, a small number n_s of Steiner points is added at random locations inside a ball centered at this pose. The ball radius is a user defined value r_s. The risk with this method is the apparition of invalid *matter* tetrahedra because we increase the number of tetrahedra without increasing the number of rays. But, since the Steiner points are inserted in regions (the camera locations) with a high density of rays, this risk is low.

The advantage of this method is its simplicity and its speed. But the problem is that it isn't very efficient in terms of the final surface quality.

Finally, another method was proposed in [Lhuillier13] and detailed in the subsection IV.1.6 of this document. Here we remember the principle.

The input of this algorithm is the list E_α of the *visually critical edges*. Each edge of E_α is split in its middle by adding a Steiner point. Each tetrahedron including

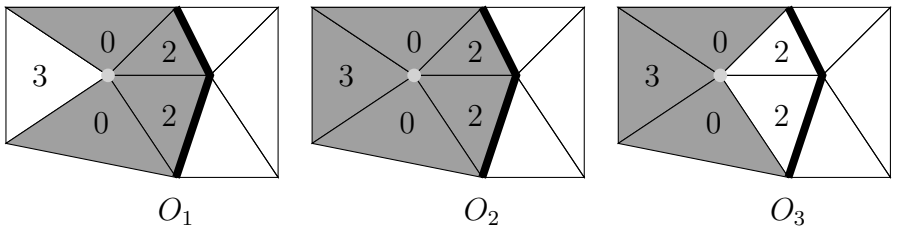


Figure VI.3: Example of an escape from local extremum thanks to our new artifacts removal (2D case). White tetrahedra are *outside*, gray tetrahedra are *inside*. The number in tetrahedron Δ is $I(\Delta)$. The light gray dot is the vertex considered by the algorithm and thick lines are visually critical edges. Left: O before removal. Middle: force neighboring tetrahedra out of O. Right: local growing of O.

the edge is also split in two, and the two resulting tetrahedra are assigned the same number of intersections and status as the initial one. This way, we reduce the size of tetrahedra and hope to locally unlock region growing in the neighborhood of this edge. Furthermore, ∂O is still manifold. The drawback is that the triangulation is not guaranteed to be Delaunay anymore.

Then we try to remove each *visual artifact* by forcing its tetrahedra to *outside*. This creates some singularities (i.e. vertices where the surface isn't *2-manifold*). Then, we try to remove these singularities by a local *region growing* (*repair* step of subsection IV.1.6). More precisely, we try to add *free-space inside* tetrahedra to *outside* in a way to lower the number of singularities. If we succeed (i.e. the number of singularities becomes zero), a *visual artifact* was removed, otherwise all the tetrahedra added to O are removed from it.

The advantage of this method is that it is efficient in terms of output surface quality. But it is slow. This comes from the fact that it tends to perform a lot of operations before concluding that the number of singularities can't become zero, so all these operations turn to be useless, but consume the computation time.

Moreover, the edges should not be split when the algorithm is applied in the incremental case because of the triangulation becoming non Delaunay and we need some properties of the Delaunay triangulation to ensure the correct functioning of our algorithm (see subsection IV.1.1). So, its efficiency is reduced.

VI.3 New artifacts removal method

In this section, we propose a faster *visual artifacts removal* method that doesn't require Steiner points insertion. We begin by describing the algorithm, then we study its theoretical complexity under the loose and tight assumptions.

VI.3.1 The algorithm

The region growing described in the subsection IV.1.5 is a greedy optimization algorithm that maximizes

$$f(O) = \sum_{\Delta \in O} I(\Delta) \tag{VI.1}$$

Algorithm VI.1. New visual artifacts removal algorithm

function REGION_GROWING(Δ_0)
▷ Local region growing in G_α
  Let Q be a priority queue of tetrahedra based on $I(\Delta)$
  PUSH(Q, Δ_0)
  $L_{add} \leftarrow \emptyset$
  while $Q \neq \emptyset$ **do**
    $\Delta \leftarrow$ POP(Q)
    if $\Delta \notin O$ and $\partial(O \cup \{\Delta\})$ is manifold **then**
      $O \leftarrow O \cup \{\Delta\}$
      $L_{add} \leftarrow L_{add} \cup \{\Delta\}$
      for all tetrahedra Δ' adjacent to Δ **do**
        if $\Delta' \in G_\alpha$ and $\Delta' \notin O$ **then**
          PUSH(Q, Δ')
        end if
      end for
    end if
  end while
  return L_{add}
end function

procedure ARTIFACTS_REMOVAL
▷ The main artifacts suppression algorithm
  $s_{old}, s, it \leftarrow 0$
  repeat
    $s_{old} \leftarrow s$
    for all vertex **v** of both ∂O and G_α **do**
      $N_{\mathbf{v}} \leftarrow$ all the tetrahedra incident to **v**
      $L_{sub} \leftarrow O \cap N_{\mathbf{v}}$
      $L_{seed} \leftarrow (G_\alpha \cap N_{\mathbf{v}}) \setminus O$
      $L_{add} \leftarrow \emptyset$
      $O \leftarrow O \setminus L_{sub}$ ▷ Local shrinking
      if ∂O is manifold **then**
        for all $\Delta \in L_{seed}$ **do** ▷ Local growing
          $L_{add} \leftarrow L_{add} \cup$ REGION_GROWING(Δ)
        end for
        $R_{sub} \leftarrow \sum_{\Delta \in L_{sub}} I(\Delta)$
        $R_{add} \leftarrow \sum_{\Delta \in L_{add}} I(\Delta)$
        if $R_{sub} > R_{add}$ **then**
          $O \leftarrow O \setminus L_{add}$
          $O \leftarrow O \cup L_{sub}$
        else
          $s \leftarrow s + R_{add} - R_{sub}$
        end if
      else
        $O \leftarrow O \cup L_{sub}$
      end if
    end for

    $it \leftarrow it + 1$
  until $s_{old} \neq s$ **or** $it < it_{max}$
end procedure

in the discrete search space of tetrahedra lists O such that $O \subseteq F$ and ∂O is a 2-manifold. In [Lhuillier13], we have a steepest descent heuristic for function $-f(O)$ (by region growing of O) which can get stuck to a local maximizer of $f(O)$. A ***visual artifact*** (e.g. spurious handle) can be seen in this situation. The basic idea of our new method is to remove some tetrahedra from O (and so to decrease $f(O)$) to kick the algorithm out of its local extrema.

As in the previous section, we call E_α the list of ***visually critical edges***. Let $G_\alpha$be the set of tetrahedra $\Delta \in F$ which have an edge in E_α. Then, for every vertex $\mathbf{v}$ of both ∂O and G_α, we force neighboring tetrahedra out of O and we try local region growing which begins from neighboring tetrahedra included in G_α. If the final value of $f(O)$ is greater than the initial one, we are able to escape from a local maximum, otherwise we revert everything to the initial state and try another vertex $\mathbf{v}$. See the figure VI.3 for an example and the algorithm VI.1. Once we tried all ∂O vertices, we complete the result using ***region growing*** and ***topology extension*** (restricted to the tetrahedra included in the ***working zone*** W in the incremental case (see section IV.2)).

The details are in the algorithm VI.1.

VI.3.2 Complexity analysis

Loose assumptions

We begin the complexity study of the algorithm VI.1 by an analysis in the loose case. The notations and assumptions are those of the section IV.3 and we suppose that the algorithm is executed during an iteration of the incremental pipeline.

We consider that all the edges of the working zone are ***visually critical*** for a worst case complexity analysis. So $|G_\alpha| = \mathcal{O}(k^2)$ (subsection IV.3.4) and the ***for*** loop at the line 23 has $\mathcal{O}(k)$ iterations (***working zone*** contains $\mathcal{O}(k)$ vertices). We suppose that for each ***local shrinking*** ∂O remains ***2-manifold***. $|L_{seed}| = \mathcal{O}(k)$ because $|N_v| = \mathcal{O}(k)$ (see appendix C). So the ***local region growing*** at line 31 is performed $\mathcal{O}(k)$ times.

The ***region growing*** complexity is given by the equation IV.6: $\mathcal{O}((g+q_0)(log(g+q_0)+d))$. The number of added tetrahedra g is $\mathcal{O}(k^2)$ because the ***growing*** is limited to the ***working zone***. The initial size of the queue q_0 is 1 and the maximum vertex degree d is $\mathcal{O}(k)$. So one ***local growing*** has the complexity of $\mathcal{O}(k^2(log(k^2)+k)) = \mathcal{O}(k^3)$. So the ***for loop*** of the line 23 has the complexity of $\mathcal{O}(k^5)$.

Finally, the complexity of the main loop at line 21 is bounded thanks to the maximum number of iterations it_{max}. So the complexity of the new artifacts removal algorithm in the loose case is $\mathcal{O}(k^5)$.

Tight assumptions

Now we will compute the complexity of the algorithm VI.1 using tight assumptions. In this scenario, the size of the ***working zone*** is $\mathcal{O}(1)$. We still suppose that all the edges of the ***working zone*** are ***visually critical***. This gives us $|G_\alpha| = \mathcal{O}(1)$. Thus the loop at line 23 has $\mathcal{O}(1)$ iterations.

The maximum vertex degree is $\mathcal{O}(1)$ thanks to T6, so $|L_{seed}| = \mathcal{O}(1)$. So the ***local region growing*** is performed $\mathcal{O}(1)$ times. And the complexity of one ***growing*** is $\mathcal{O}(1)$ because we are in the tight case (see subsection IV.3.6). So one iteration of the main loop at line 21 has the complexity of $\mathcal{O}(1)$.

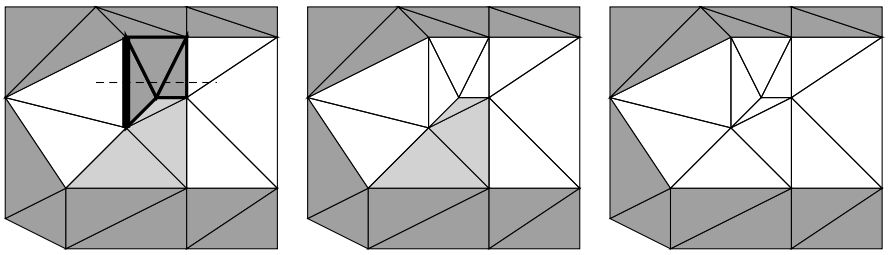


Figure VI.4: An example of application of spurious handle removal (2D case). White tetrahedra are *outside*, gray tetrahedra are *inside*, light gray tetrahedra are *free-space inside*. The thick line is a *visually critical edge*, the dashed line is plane π and the triangles with thick borders are selected tetrahedra.

Finally, the number of iterations of the main loop is bounded thanks to it_{max}. This gives us the complexity of the new artifacts removal algorithm in the tight case: $\mathcal{O}(1)$.

VI.4 Handles removal method

In addition to the new artifacts removal method described in the previous section, we propose here a more specialized approach: an algorithm to remove the handles. This kind of visual artifacts are more easily visible by the human eye. As in the previous method, we begin by describing the algorithm, then we study its complexity.

VI.4.1 The algorithm

This algorithm is a specialization of the subsection IV.1.6 method. The latter is slow mainly because it consists in many attempts to remove a small pack of tetrahedra and the majority of these attempts are unsuccessful. So, the idea is to remove bigger packs of tetrahedra. This is possible because we are in a more restrained context.

Instead of trying to remove the *visually critical artifacts*, we seek to remove a particular kind of artifact: the *handle*. The second column of figure VI.6 shows an example. This is the most visible kind of artifact for the human eye.

The basic idea of the algorithm is to remove the *handle* (by contrast to the subsection IV.1.6 where we remove all the tetrahedra including a vertex) and then try to restore the manifold property of O by a local *region growing*. To do that, we begin as usual, by computing the list E_α of *visually critical edges*. For each edge $\mathbf{ab} \in E_\alpha$, we check if it is contained in a handle.

We define a plane π perpendicular to **ab** and intersecting segment **ab** at some points. In practice we try several planes intersecting **ab** in $\frac{2\mathbf{a}+\mathbf{b}}{3}$, $\frac{\mathbf{a}+\mathbf{b}}{2}$ and $\frac{\mathbf{a}+2\mathbf{b}}{3}$. Let L_π be the list of the tetrahedra intersected by π. Let $N_{\mathbf{ab}}$ be the list of the tetrahedra including edge **ab**. A *handle* H is a set of tetrahedra forming a *visual artifact*, so $H \subseteq F \setminus O$. With this definition, we begin to form our *handle* by $H \leftarrow (N_{\mathbf{ab}} \cap L_\pi) \cap (F \setminus O)$.

Let N_H be the list of the tetrahedra directly adjacent to the set H (i.e. $\Delta \in N_H$, if and only if Δ has a 4-neighbor in H). We iteratively grow H by performing $H \leftarrow H \cup (N_H \cap L_\pi \cap (F \setminus O))$ until no more tetrahedra can be added or the

number of iterations becomes greater then an user defined value. Then we check that the final H is surrounded by O in plane π (see the the figure VI.4), i.e. $\forall \Delta \in (N_H \cap L_\pi) \setminus H, \Delta \in O$.

Once we have detected a *handle* H, we try to remove it as in subsection IV.1.6. First we force H in O (i.e $O \leftarrow O \cup H$), then we try to restore the *2-manifold* property of ∂O using the *repair* step initialized by the tetrahedra in $N_H \cap (F \setminus O)$.

The *repair* step is the same as in the algorithm of the subsection IV.1.6. We seek to add a bunch of *free-space* tetrahedra to O such that ∂O become *manifold* once again. To achieve this goal, we apply a *local region growing* (the *repair* step of the subsection IV.1.6) algorithm to O in F starting in the neighborhood of $N_H \cap (F \setminus O)$ and which decreases the number n of singular vertices. The algorithm stops if a number of iterations g_0 (fixed by the user) has been reached or no more tetrahedra can be added to O.

If the repair step succeeds (i.e. $n = 0$: ∂O is *2-manifold*), we are able to remove a *visual artifact* and proceed to the next vertex. Otherwise, we restore O to the previous state and try another edge in E_α.

VI.4.2 Complexity analysis

Loose assumptions

We begin the complexity analysis of the handle removal algorithm by an analysis in the loose case. As for the new artifacts removal algorithm we use the same notations and assumptions as used in the section IV.3.

We suppose that all the edges are *visually critical* for a worst case complexity analysis. Because all our computations are taking place in the *working zone*, there are $\mathcal{O}(k^2)$ *visually critical* edges (consequence of the assumption L2).

For each *visually critical* edge, we use an iterative growing to construct a *handle*. Because the number of iterations of this growing is bounded, its complexity is $\mathcal{O}(1)$. To check that constructed *handle* is enclosed by the *outside* region we need to check the neighbors of this pack of tetrahedra. Because each tetrahedron has exactly 4 neighbors, the complexity of this step is $\mathcal{O}(1)$. So the total complexity of the handle detection step is $\mathcal{O}(1)$ in the loose case.

Once a *handle* is detected it is forced to *outside* and a greedy *region growing* algorithm is used to repair the singularities. The complexity of the *region growing* is bounded by a maximum number of iterations g_0, so the complexity of the *repair* step is $\mathcal{O}(1)$ in the loose case.

In conclusion, the complexity of one handle detection and removal is $\mathcal{O}(1)$. In a worst case complexity analysis, we suppose that each *visually critical edge* is adjacent to a *handle*. So the complexity of the handles removal algorithm is $\mathcal{O}(k^2)$ in the loose case.

Tight assumptions

Now we will estimate the complexity of the handle removal algorithm in the tight case. As in the loose case, we suppose that all the edges of the *working zone* are *visually critical*. Because the size of the *working zone* is bounded, there are $\mathcal{O}(1)$ *visually critical edges*.

To check if a *visually critical edge* is contained inside a *handle* we perform an iterative growing and we check the neighborhood of the computed pack of tetrahedra.

Removal method	Artifacts removal computation time	Num. artif.
None	0 s.	52
A without Steiner vertices	55.22 s.	28
A with Steiner vertices	2 min. 46.27 s.	23

Table VI.1: Numerical results of the old artifact removal method (called **A**) with and without Steiner vertices. The experiments are performed using the *batch* method.

Because the iterative growing is bounded by a user defined threshold, the complexity of one *handle* detection is $\mathcal{O}(1)$.

Once a *handle* is detected, it is forced to *outside* region and the singular vertices are repaired using a *local region growing*. The complexity of the *region growing* is $\mathcal{O}(1)$ in the tight case because it is bounded by a maximum number of iterations. So the complexity of the handles removal algorithm is $\mathcal{O}(1)$ in the tight case.

VI.5 Experimental study

In this section, we perform a comparison between the two new artifacts removal methods detailed in the previous sections and the previous artifacts removal method proposed by [Lhuillier13]. To perform the comparison, we use the *Aubiere* real video sequence described in the subsection V.1.

We begin by a discussion on the way to compare the different artifacts removal methods in the subsection VI.5.1, then we study the influence of Steiner points insertion on the quality of the final surface when the old artifacts removal method is used in the subsection VI.5.2. The comparison between the different *artifacts removal* methods is performed in the subsection VI.5.3. And finally the influence of the value of the *visually critical edges* detection angle is studied in the subsection VI.5.4.

VI.5.1 How to compare the artifact removal methods?

Every removal method is integrated in the surface post-processing step of the incremental surface reconstruction method (see subsection IV.2) (before peaks removal). To compare the methods in terms of output surface quality, we manually count the *visually critical artifacts* (see section VI.1) remaining on the final surface. Because we seek to remove in priority the *artifacts* that are visually critical (and so are easily noticed by a human eye), we consider this number as a good quantitative metric. The figure VI.6 shows examples of artifacts that are manually counted.

Moreover, we also estimate the final number of *free-space inside* tetrahedra (i.e. the union of the *visual artifacts*) and the final value of the objective function f (equation VI.1). The latter quantifies the ability of every method to unlock the *region growing* and *topology extension* steps (see subsection IV.1.5), or in other words, the ability to escape from local extremum of f.

Removal method	Mean time	Max. time	Num. artif.	Size of $F \setminus O$	f= $\sum_{\Delta \in O} I(\Delta)$ ($M = 10^6$)
None	0	0	18	156288	32.506M
A	1.19 s.	4.74 s	11	154152	32.522M
B	0.32 s.	1.07 s	12	147281	32.552M
C	0.21 s.	0.70 s	14	155413	32.511M
B & **C**	0.46 s.	1.40 s	9	147181	32.555M

Table VI.2: Numerical results of different artifacts removal methods. **A** is the old artifacts removal method, **B** is the new artifacts removal method from the section VI.3 and **C** is the handle removal method from the section VI.4.

VI.5.2 Comparison of the old method with and without Steiner vertices

One of the advantages of the new artifacts removal methods discussed here is the fact that they don't need the insertion of Steiner vertices. So, before comparing the old method with the new ones, we evaluated the influence of these points on the quality of the previous algorithm (this is not done in [Yu13, Lhuillier13]).

To achieve this goal, we have applied the *batch* algorithm without *artifacts removal* step, with the old *artifacts removal* algorithm, and without Steiner points insertion and finally with the old *artifacts removal* performed normally. Then, we compare their computation times and the number of manually counted tetrahedra (see table VI.1). We don't compare the numbers of *free-space inside* tetrahedra and the values of the *objective function* because the insertion of Steiner vertices changes the number of tetrahedra. So the comparison between these values is meaningless.

When we observe the values of the table VI.1, we conclude that the insertion of Steiner vertices increases the efficiency of the old artifacts removal algorithm, but at the same time the computation time becomes almost 3 times slower. So the Steiner vertices insertion could be omitted, even in the *batch* case, if the computation time is very important.

VI.5.3 Comparison of the three methods

We evaluate five *visual artifacts* removal methods: None (no removal method), **A** (the old method from [Lhuillier13] detailed in the subsection IV.1.6), **B** (escape from local extremum using the method from the section VI.3), **C** (handle removal using the method from the section VI.4), **B** & **C** (use **C** after **B**).

The results are summarized in the table VI.2. The removal methods **A**, **B**, **C** and **B** & **C** provide improvements (increases) of the objective function f. The differences between them are small, but we see that **B** is slightly better than **A**. Furthermore, we see that the combination of **B** and **C** is even better, and this is confirmed by both the number of tetrahedra in $F \setminus O$ (union of all *visual artifacts*) and the number of artifacts that are manually detected. The figure VI.6 compares the results of all methods at four locations in the reconstruction and is consistent with these comparisons. We note that **B** has important visual artifacts (as the one in the second column of this figure) although it has a good (small) $F \setminus O$, and **C** can correct them in spite of its greater $F \setminus O$. Then we think that the combination **B** & **C** is a good choice.

A difference between the four methods is the computation time. Indeed, **B**,

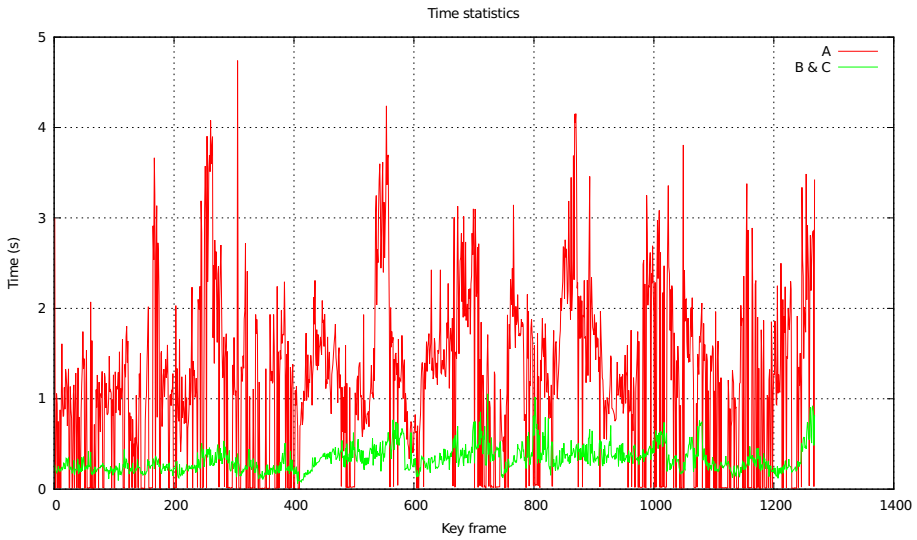


Figure VI.5: Computation times at every *key frame* for both the previous (**A**, red) and the new (**B** & **C**, green) visual artifact removal methods.

Value of detection angle	Mean time	Max. time	Num. artif.	Size of $F \setminus O$	f= $\sum_{\Delta \in O} I(\Delta)$ ($M = 10^6$)
None	0	0	18	156288	32.506M
1°	0.58 s	1.73 s	9	145558	32.558M
5°	0.46 s	1.40 s	9	147181	32.555M
10°	0.34 s	1.06 s	9	150364	32.535M
15°	0.28 s	0.95 s	15	152126	32.530M
20°	0.25 s	0.81 s	17	152570	32.532M

Table VI.3: Numerical results for different values of the *visually critical edges* detection angle α.

C and **B** & **C** are significantly faster than **A** since their mean time per keyframe is about 2.5-4 times smaller (table VI.2 and the figure VI.5).

VI.5.4 Study of the influence of the value of the detection angle

The *artifacts removal* algorithms discussed here remove *visually critical artifacts*. A *visually critical artifact* is an artifact including a *visually critical edge* (see section VI.1). And the *visually critical edges* are detected using an user defined threshold α. In the experiments of this dissertation we used $\alpha = 5^\circ$ to use the same value as [Lhuillier13]. But, we also perform a series of experiments to find the optimal value of this parameter.

To achieve this goal, we apply the *incremental* algorithm to *aubiere* data set using the combination of **B** and **C** *visual artifacts* removal algorithms. We vary the value of α and compared the results as explained in the subsection VI.5.1. The different numerical values are in table VI.3.

As expected, the smaller is the value of α, the slower are the computations.

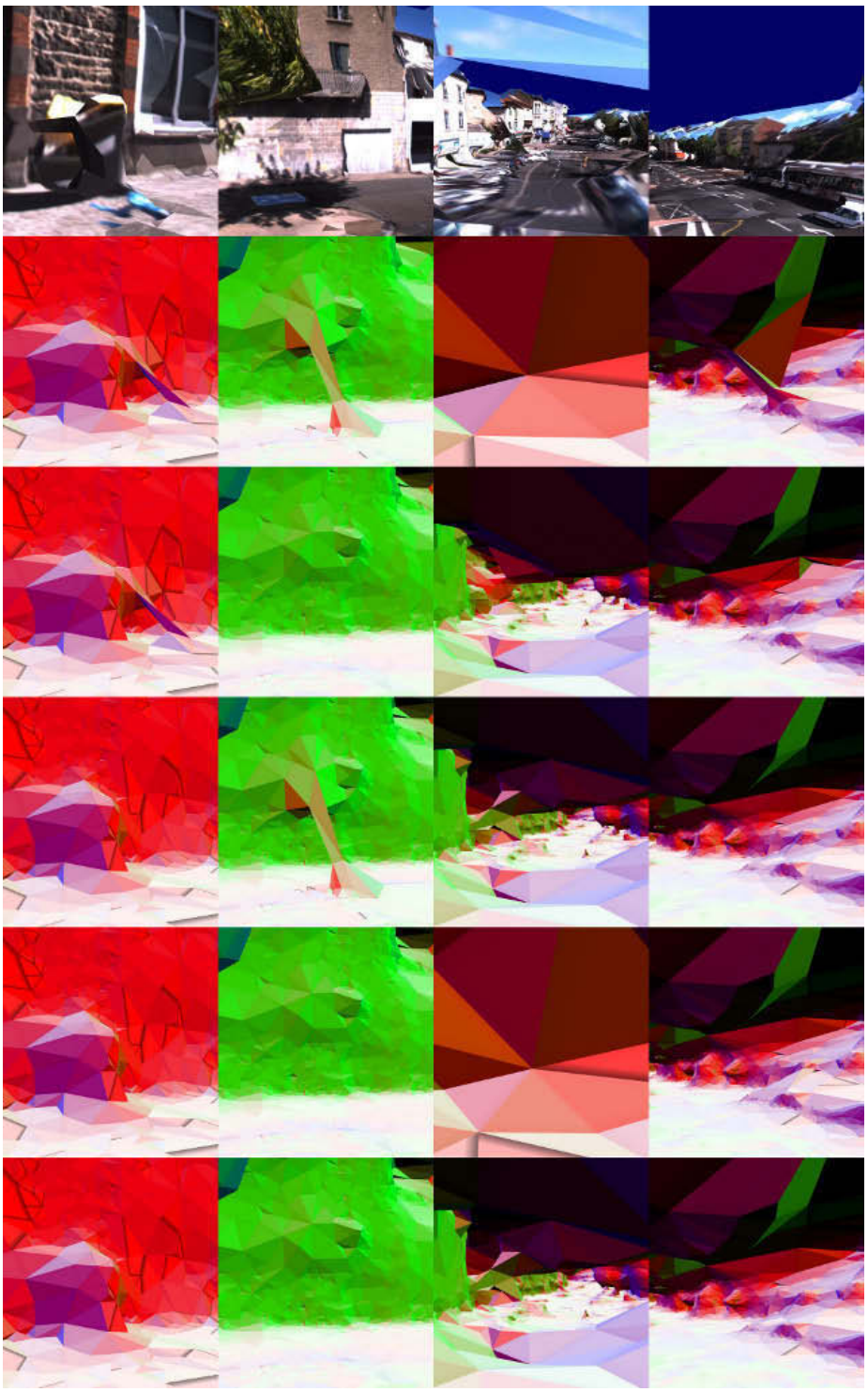


Figure VI.6: Results of artifacts removal methods at four locations (one location per column). Lines from top to bottom: textured scene, no artifacts removal, method **A**, method **B**, method **C**, method **B** & **C**.

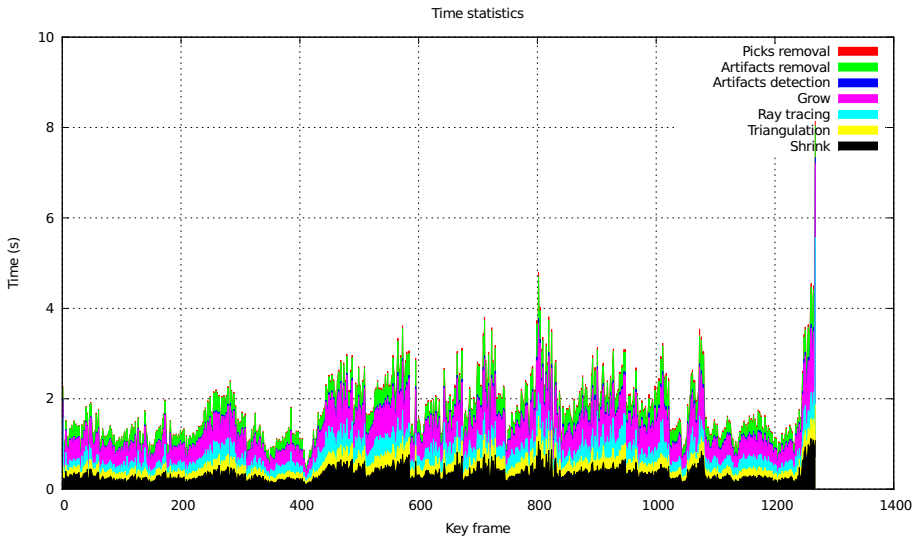


Figure VI.7: Computation times at every keyframe for the steps of our incremental surface reconstruction method including **B** & **C** as *visual artifact* removal method. Note that these times are accumulated for steps shrink, ..., artifact removal. Times are smaller than in figure V.11.

Moreover, if we compare the number of *free-space inside* tetrahedra and the final value of the *objective function*, we see that a smaller value of α means better results. But, if we compare the number of manually counted artifacts, we see that setting the threshold α to a value smaller than 10 provides no visually perceptible enhancement to the final surface. So we can conclude that a good value of α is 10.

VI.6 Conclusion

This chapter has studied the artifacts removal step of the sparse incremental surface reconstruction algorithm proposed by this dissertation. We have defined the notion of an *visual artifact* and have briefly reviewed the previous attempts to solve this problem.

The artifacts removal method that was previously proposed is good in terms of surface quality. But it is slow and needs the insertion of Steiner points for maximum efficiency. Unfortunately, the later is difficult in our incremental context.

We replace this step by the same method restricted on particular cases of *visual artifacts* (*handles*) and preceded by another method which escapes from local extrema of the objective function. The methods are compared using *aubiere* experimental data set. The conclusion is that the processing time of the *artifacts removal* step was greatly reduced without loss of surface quality. The new computation times (see figure VI.7) are smaller compared to the old method (see figure V.11).

CHAPTER VII

Adding contours to the reconstruction process

When we think about the primitives to use for *sparse* surface reconstruction, we usually think about interest points. The question is: Is it the only type of primitive useful to solve this problem? The response is, of course, no. The other types of possible primitives are line segments, curves, planes, etc...

In this chapter we explore the usefulness of adding curves (in supplement to the 3D points) to our surface reconstruction pipeline. The curves are less explored in the literature than the points, but they have some potential advantages. The environments created by humans are often low textured and so it can be difficult to find a sufficient number of stable interest points. On the other hand, such environments usually provide a high amount of curves.

The results of this study were published in [Litvinov12]. Although, the evaluation of the batch sparse surface reconstruction using the standard *Middlebury* [Seitz06] data set are in [Yu13], it is our contribution.

We begin by a review of the previous works using the curves for surface reconstruction in section VII.1. It is followed by a general outline of the surface reconstruction algorithm when the curves are used in section VII.2. Then, some detailed explanations are provided for the two parts that are added to the original algorithm: curves detection and matching (section VII.3) and 3D curve reconstruction (section VII.4). Finally, the results of the algorithm with and without curves are compared in the section VII.5.

VII.1 Previous works

The usage of the curves for surface reconstruction is a relatively less explored topic. Our objective is to evaluate the added value provided by the curves compared to the interest points alone when integrated in our method. So we begin by performing an overview of the different approaches to contours based surface reconstruction to see which one can easily be integrated in our pipeline.

VII.1.1 Contours based surface reconstruction methods

The first type of algorithms that make use of curves for (at least partial) surface reconstruction are the **occluding contours** based approaches. An **occluding contour** is a depth discontinuity in the image. More precisely it is a region of the observed surface where the dot product between camera view direction and the surface normal is zero. In this category we find [Koenderink84, Cipolla92, Vaillant92, Zheng94] and others. The basic idea of this kind of methods is to deduce the geometrical properties of the observed surface by observing the deformation of contours with the camera motion. The majority of these methods only allow the reconstruction of small surface patches around the curves with the notable exception of [Zheng94] who reconstruct the complete 3D model.

In [Zheng94], the input sequence of images is acquired from a turntable motion. The object to reconstruct is placed in front of a well distinct background, so the silhouette contours can easily be detected. These contours are then parametrized following the direction parallel to the rotation axis (y axis, in practice). The study of the evolution of these points allows to reconstruct a 3D point cloud and ultimately a 3D model of the observed object. An interesting property of this algorithm is that it detects the concavities, parts of the surface that can't be reconstructed by this method, and so some other approach can be used to complete the surface.

The main problem of this kind of algorithms are that they don't work well with concave surfaces, the need for a highly controlled acquisition processes and the fact that usually only a set of patches are reconstructed. The advantage is that they work well with smooth objects.

Another way to use **occluding contours** is to construct an approximate surface for initializing a *dense* reconstruction. Examples of this kind of methods are [Laurentini94, Kutulakos00, Bottino04, Esteban04].

Their principle is simple. Given an input image, it is segmented into two distinct zones: the object and the background. The rays corresponding to the object pixels are back-projected and forms a cone like shaped zone in 3D space. We know that the the observed object is inside this zone. Once this computation is performed for each image of the input sequence, these visibility zones are intersected. The computed intersection is an approximate surface (visual hull) of the object.

The drawbacks of these methods is that the resulting surface is only a loose approximation of the really observed object, the observed scene must contain only a single object and generally the acquisition must be highly controlled (green/blue background). Also the concave parts of the object can't be reconstructed.

Finally, a different use of the curves for surface reconstruction is to reconstruct the image curves in 3D thus forming a **3D sketch** (or wire-frame model). We could cite [Faugeras90, Kahl03, Wu05, Liu06, Martinsson07, Hofer13]. Once the sketch is reconstructed, it is possible to integrate it into the 3D Delaunay triangulation and perform the surface reconstruction.

For example, [Faugeras90] shows that it is possible to subdivide a segment into a set of points in such a way that when they are inserted into a Delaunay triangulation, the triangulation will contain the said segment as the union of some its edges. To achieve this goal, the segment should be subdivided into $D/2d-1$ points where D is the length of the segment and d is the minimal distance between the segment and the nearest other primitive (segment or point) of the triangulation.

[Faugeras90] also extends the classical visibility constraint to the case of line segments. If a camera **c** sees a segment **ab**, all the tetrahedra of the Delaunay trian-

gulation intersected by the triangle **acb** are *free-space*. The method is experimented on a synthetic and real examples. In the case of real sequences, the line segments are matched using the method from [Ayache87]. The results are interesting, but the experimented sequences are limited to three view.

Another example of such an approach is [Wu05]. This method is not automatic. The end user indicates by hand the interest points and curves in the input images. Moreover, the user also indicates the matches between points and curves in different input images. Then, the algorithm computes the camera *poses* associated with the input images using a *Structure-from-Motion* and the points correspondences. The 2D points of each pair of matched curves are matched using the *epipolar* constraint. These matches are used to reconstruct the 3D sketch. Finally, this sketch is used to define a set of surface patches and each patch is refined using photo-consistency to obtain the final 3D surface. The quality of the resulting surface is very good, but the inconvenient of this method is, of course, the fact that it is non automatic.

For a more recent example, we could cite [Hofer13]. It is an incremental and real-time **3D sketch** reconstruction method. There are two steps. First, the camera *pose* is evaluated for each incoming image using interest points and a *SfM* algorithm. Second, it detects straight line segments inside the upcoming images, matches them with the previous views and enhances the sketch with the new information.

The matching between segments is performing using only geometrical considerations, namely epipolar constraints and reprojections. For each putative match, the end points of one of the segment are matched with the points of the second segment using *epipolar* constraints and reconstructed in 3D to obtain the corresponding putative 3D segment. Then, the putative 3D segment is reprojected to other images to see if it has enough supporting 2D segments. This way even dense wiry structures (such as electrical pylons) can successfully be reconstructed.

The **3D sketches** produced by this algorithm of good quality, but it only reconstructs straight line segments which is a bit restrictive.

The **occluding contours** approaches are too different to point based reconstructions and can't be easily integrated into our pipeline because, in our case, it is very difficult to distinguish foreground from the background. On the other hand, our 3D point cloud can be enhanced by a **3D sketch** and the results of [Wu05] indicates that this could lead to a significant surface quality increase. However, we seek to develop an automatic method and so we need a way to match the curves of different input images in a fully automatic way.

VII.1.2 Curves matching methods

In this subsection we overview the solutions of the problem of 3D sketch computation from the curves detected in a series of images. More precisely, focus on the problem of finding the correspondences of curves between images observing the same scene.

The first publications concerning the problem of curve matching are for two views with known intrinsic and extrinsic parameters. Among these publications we could cite [Arnold80, Brint90, Nasrabadi92]. To resolve the problem of disambiguation of correspondences, they try to use some additional geometric constraints. For example, the method in [Arnold80] use the curve tangents. It uses a heuristic that the tangents directions in images tend to be similar at the corresponding points of two curves. Unfortunately, it is only true for a very short

baseline and this is a general problem for the two view matching methods. They need to make use of heuristics that severely limits they range of applicability.

If we have three views with known intrinsic and extrinsic parameters the problem of biased heuristics can elegantly be avoided. The idea proposed by [Ayache87] is to uses the points and tangents at these points from two potentially corresponding curves to compute the corresponding point and tangent in the third view, then to compare it with observations. The works [Robert91, Schmid00] use the same technique but with the curvature instead of tangent. Finally, these techniques were generalized to N-view and a complete curve reconstruction pipeline was proposed by [Fabbri10]. Another advantage of this pipeline is that the cameras calibration can be approximate.

It is noteworthy that the 3D sketch can also be computed without performing the curve matching, for example by voxel like technique [Teney12].The advantage of this approach is that no correspondences between curves need to be performed. Drawback of this method is the fact that it is voxel based. This makes it difficult to use for large scale scenes and so it is not really interesting for us.

VII.2 Algorithm outline

The final objective of this study is to evaluate how the quality of the final surface changes if the curves detected in the input images are added to the reconstruction process. So we modify the global bath surface reconstruction process in order to integrate this new information. To achieve this goal, additional steps are added after *Structure-from-Motion* step and before the surface reconstruction itself.

The modified algorithm has the following steps:

1. The *Structure-from-Motion* algorithm is applied to the input video sequence. This algorithm is described in the chapter III. After this step, a set of *key frames* is selected in the entire sequence. The cameras *poses* associated to each *key frames* are computed, as well as the 3D cloud of points and the corresponding visibility information.

2. The curves are detected and matched in the consecutive *key frames* in order to create a set of curves tracks. This is discussed in the section VII.3.

3. The curves are reconstructed in 3D, i.e. for each curve track, we compute the corresponding curve in the 3D space. Each 3D curve is represented by a set of 3D points by sampling. This way they are straightforward to integrate to the existing 3D cloud of points provided by the *SfM*. This step is discussed in the section VII.4.

4. Finally, the 3D cloud of points computed by the initial *SfM* and augmented by the sampled 3D curves of the previous step is processed by the batch surface reconstruction algorithm of the section IV.1 in order to produce the final output surface.

As can be seen, the tests of this chapter was performed in a batch context for practical reasons. But, as can easily be seen, this algorithm is straightforward to convert to an incremental scheme. The *Structure-from-Motion* algorithm is already incremental. Each time a new *key frame* is selected and the corresponding camera *pose* is computed, we can detect the curves in the corresponding input image and

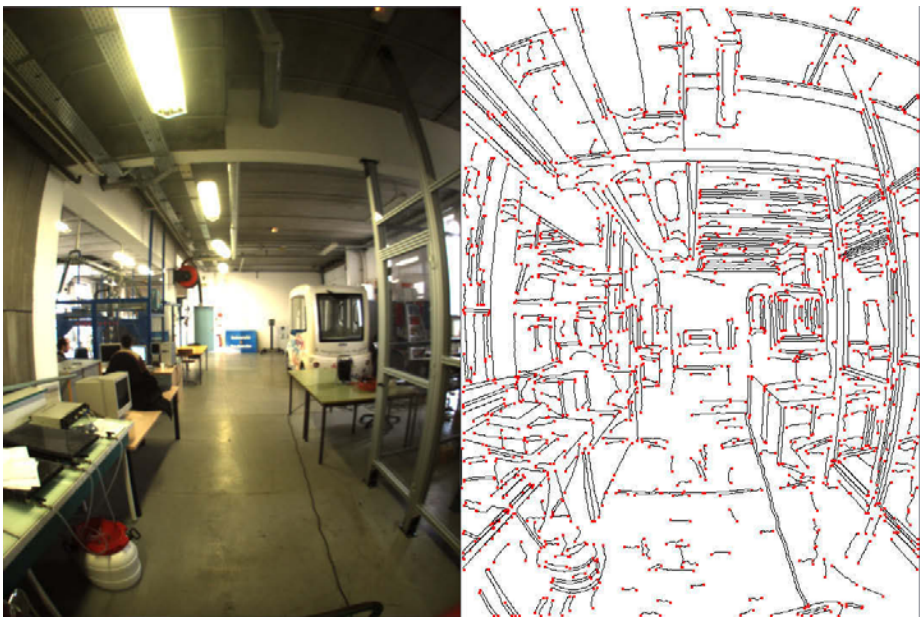


Figure VII.1: An example of curves detection. On the left side you can seen the original image. On the right side you can see the curves detected in it. The red dots are the ending points of individual curves.

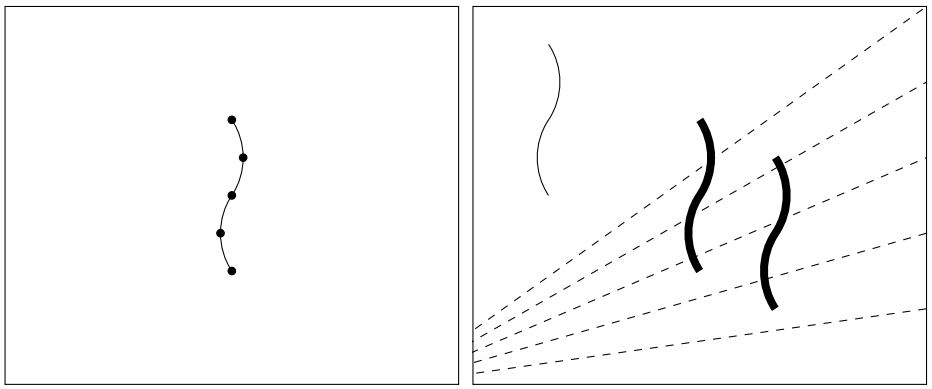
match them with the previous *key frame*. Then, we can reconstruct the curves that are no longer tracked and integrate them into the cloud of points. Finally, we can apply the incremental surface reconstruction algorithm of the chapter IV as usual. This is why the results of this chapter can directly be applied to the incremental case.

VII.3 Curves detection and matching

After the *Structure-from-Motion* algorithm has selected the *key frames* and computed the corresponding camera *poses*, the extraction of the curves from the input images begins. First, the curves are detected and parametrized. This step is explained in the subsection VII.3.1 Then, the detected curves are matched with the ones from the previous *key frame*. This is detailed in the subsection VII.3.2.

VII.3.1 Curves detection

First, we perform an edge detection in the incoming image. It is performed by a slightly simplified version of the well known Canny algorithm [Canny86]. The simplification consists in dropping the hysteresis edge tracing step because it was found almost useless (the enhancement provided by this step was unnoticeable). The algorithm output is a binary image. The pixel value of *false* means that the pixel is not a part of a curve, *true* means it is. The secondary outputs of the algorithm are the gradient value and its orientation for each pixel of the input image. The main parameter of *Canny* is the detection threshold, it is calculated by the formula: $t \times max(G)$, where t is the value supplied by the user and G is the set of gradient



(a) Image $t-1$ (b) Image t

Figure VII.2: An example of application of the curves matching algorithm. Black dots are the points sampled on the curve to match, dashed lines are the *epipolar lines* corresponding to these points in the next image. Thick curves are the potential match candidates. Thin curve is not.

values of each pixel of the input image.

The next step after the detection of edges in the incoming image is to convert the resulting bitmap into a set of parametric curves: the *edge vectorization*. In fact this is a two step process: first the image is processed from top to bottom to trace the curves with horizontal parametrization, then the image is processed again but, this time, from left to right and vertically parametrized curves are traced. The two algorithms are almost identical, so we only review the horizontal edge tracing.

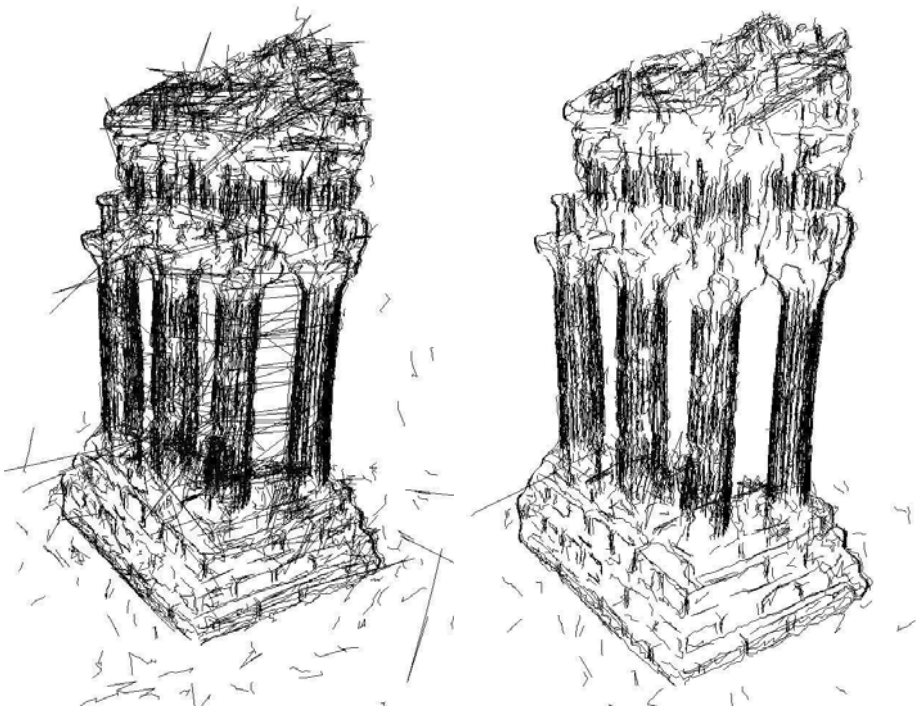
The algorithm itself is easy to understand. It scans each line in search of detected points that appears to be a beginning of an almost horizontal curve (all point neighbors are on its right side). When one is found, the algorithm follows the edge from pixel to pixel and stores the result. The only user supplied parameter is L_{min}: the minimal length a curve should have in order to be processed in the next stages. It allows to filter the unneeded noise (and also speed up the processing) and ensures that all the curves are long enough so their reconstruction quality can be verified.

An example of the curves detected and extracted from an image can be seen on the figure VII.1.

VII.3.2 Curves matching

The next step is to math the curves with those in the previous *key frame*. Unfortunately, curve geometry based matching method as described in [Fabbri10] exhibited bad performance when the images have a lot of densely located similar curves (for example, as in the Ladybug datasets, see the section VII.5), it just can't make a matching decision based on curve geometry alone. So, we use a correlation based method heavily inspired by a classical points matching.

To perform a curve matching between images I_{k-1} and I_k, we proceed as follow: for each curve C_i in I_{k-1} we construct an *epipolar lines strip* in the image I_k (see figure VII.2 for an example). An *epipolar lines strip* is a set of *epipolar lines* (see subsection III.2.3) corresponding to each point of the curve C_i. This is possible because the *SfM* already computed the *poses* corresponding to each *key frame*. It is



(a) Classical RANSAC (b) Unified RANSAC

Figure VII.3: A comparison between the sets of 3D curves reconstructed with different curve initialization methods in use.

not useful to compute the lines for each pixel of the curve, an uniform sampling of points is enough.

Then we reject all the curves not intersected by *epipolar* lines of this strip. The remaining curves of I_k form a list of candidates L_c. For each curve C_j in L_c we compute a zero-normalized cross-correlations (*ZNCC*) between intersections of C_j with the epipolar lines (in I_{k+1}) and the points of C_i that produced them (in I_k). Finally, the mean value of all these correlations define the score of the curve C_j. C_i is matched with the curve of L_c having the best score if this curve wasn't already matched or the score of the previous match is inferior.

The output of this step is a list of curve tracks. A curve track is a list of matched curves $C_0, \ldots, C_{k-1}, C_k$ such that C_i is in the i-th ***key frame***.

VII.4 3D curves reconstruction

The next step is to transform each completed curve track (curve track that wasn't updated during the processing of the last frame) to the corresponding 3D curve. This step can be divided into 3 successive stages: inter-curve points matching, points initialization and points reconstruction.

The curve tracks generated by the previous step are converted to lists of image points tracks thanks to the *epipolar* constraints. A point track is a list of matched points $\mathbf{p}_0, \ldots, \mathbf{p}_{k-1}, \mathbf{p}_k$ where $\mathbf{p}_i$ is in the i-th ***key frame***: $\mathbf{p}_k$ is obtained from $\mathbf{p}_{k-1}$

	Temple	**Fountain-P11**	**Herzjesu-P8**
Image resolution	1x640x480	1x3072x2048	1x3072x2048
Key-frames (total frames)	312 (312)	11 (11)	8 (8)
Harris points per frame	$1.5k$	$60k$	$50k$
curve sampling size s	4	8	8
SfM 3D Points (curves)	$46k$ ($71k$)	$67k$ ($71k$)	$29k$ ($33k$)
Final triangles (curves)	$63k$ ($96k$)	$120k$ ($127k$)	$53k$ ($59k$)
Time: 3D points (+curves)	12 s. (+28 s.)	100 s. (+71 s.)	64 s. (+47 s.)
Time: Surface (curves)	19 s. (30 s.)	19 s. (24 s.)	19 s. (21 s.)
		hall	**aubière-2**
Image resolution		6x1024x768	6x1024x768
Key-frames (total frames)		117 (1211)	493 (2443)
Harris points per frame		$24k$	$24k$
curve sampling size s		8	8
SfM 3D Points (curves)		$86k$ ($145k$)	$295k$ ($374k$)
Final triangles (curves)		$83k$ ($305k$)	$399k$ ($520k$)
Time: 3D points (+curves)		17 min. (+102 s.)	55 min. (+8 min.)
Time: Surface		47 s.	81 s.

Table VII.1: Numerical results for our experiments with curves for different data sets (1k= 1000).

by the intersection of C_k and the epipolar line of $\mathbf{p}_{k-1}$ in the k-ith *key frame*. If the angle between the *epipolar line* and the C_k tangent at $\mathbf{p}_k$ is small, the intersection is inaccurate and the point track is truncated (a point track can be shorter than the curve track).

Then, for every point track, an initial 3D point position is initialized by *RANSAC* (see subsection III.2.6) for robustness. Unfortunately, using this direct approach leads to some strange artifacts (see the figure VII.3a for an example using the *Temple* dataset from the section VII.5). In fact, all the points of the same curve don't necessarily use the same views to initialize themselves by *RANSAC*. This difference in initial position leads to zigzag like 3D curves. To avoid this problem, we perform a two-step initialization: first, all the points are initialized independently; then, at the second stage, we select the two views that were used the most often and all the points are initialized using these two views. As can be seen on the figure VII.3b, this approach greatly improves the quality of the reconstructed curves.

Once the point position is initialized, it is refined using a simplified *bundle adjustment* in a way identical to subsection III.3.2. For memory it is a Levenberg-Marquardt optimization: The estimated parameter is the position of the 3D point and the objective function is computed only considering the observations of the point *track*.

Finally, we only retain for surface estimation one reconstructed point over s points for every curve track. The step size s is large enough to preserve speed and sparsity of the global algorithm and at the same time small enough to ensure that curves are correctly integrated into the Delaunay triangulation [Faugeras90].

(a) Points only. (b) Points and curves. (c) Ground truth.

Figure VII.4: *Temple* data set reconstruction results.

VII.5 Experimental study

To compare the batch surface reconstruction method results with and without the usage of curves we have performed the evaluation on five different data sets: *Temple* dataset [Seitz06], *Fountain-P11* and *Herzjesu-P8* data sets [Strecha08], and finally our own *hall* and *aubière-2* dataset. Please note that the *aubière-2* data set is different than the *aubiere* data set from subsection V.1. The first three are provided with the ground truth so this allows us to quantitatively evaluate the performance of the algorithm and to compare it with the others. The *hall* and *aubière-2* allow us to qualitatively evaluate the surface reconstruction method on a cluttered interior and a residential scene. All the processing times are evaluated on a 4xIntel Xeon W3530 at 2.8 GHz. A summary of important values about every dataset presented in this section can be found in table VII.1.

Note that the batch surface reconstruction method was initially designed for the reconstruction of complete environments using omnidirectional cameras as in our own (*hall* and *aubière-2*) data sets. In this case, the camera positions are assumed to be inside the convex hull of the reconstructed scene points. In the other cases (first, second and third data sets) a workaround is needed: we add *Steiner* vertices (extra points) at the border of a large bounding box of the object (*Temple*) or behind the camera (*Fountain-Herzjesu*) to the Delaunay triangulation, then we apply our method and remove the triangles which are incident to these *Steiner* vertices.

VII.5.1 *Temple* data set

The *Temple* provided by [Seitz06] is a standard multi-view reconstruction data set (312 separate views). In this evaluation, we use the camera positions provided with the data set to reconstruct the 3D points (we do as the other methods evaluated using this data set). We also used the *visually critical handles* detection threshold of 1°, because the usual value of 5° leaved some easily visible artifacts.

Two models were sent for evaluation to the data set creators. The first (see figure VII.4a) is reconstructed using the points only and the second one (see figure VII.4b) is reconstructed using points and curves. The ground truth image is

provided for reference on the figure VII.4c The point-only model is reconstructed in 31 s. and provides an accuracy of 0.59 mm. (for 90% of reconstructed points) and completeness of 97% (for 1.25 mm. error). The points and curves reconstruction takes 59 s. and provides an accuracy of 0.53 mm. and completeness of 97.6%. (These values can be compared to those of other algorithms at [Mid]).

This lead us to two conclusions. Firstly, the use of the curves provides indeed an enhancement to the resulting model. Secondly, compared to other methods evaluated using the same data sets, the batch sparse surface reconstruction algorithm is, from the best of our knowledge, the fastest CPU based reconstruction method, and the resulting precision is not so bad compared to majority of the dense stereo algorithms (accuracy of 0.34 mm. for 25 min. of computation for the best method at this time, the majority have an accuracy of $0.40 - 0.60$ mm.). This makes this method a good candidate for initialization of some dense stereo methods.

VII.5.2 *Fountain-P11* and *Herzjesu-P8* data sets

Fountain-P11 and *Herzjesu-P8* are another standard set of multi-view stereo data sets provided with their respective ground truths (courtesy of [Strecha08]). They contain, respectively, 11 and 8 high-resolution images. In this evaluation, we use the camera positions provided with the data set to reconstruct the 3D points.

To evaluate the error distribution against the ground truth we subdivide our output mesh with a very thin step and, for each points of the output mesh, we calculate the distance to the nearest point of the ground truth mesh. (see subsection V.1.2).

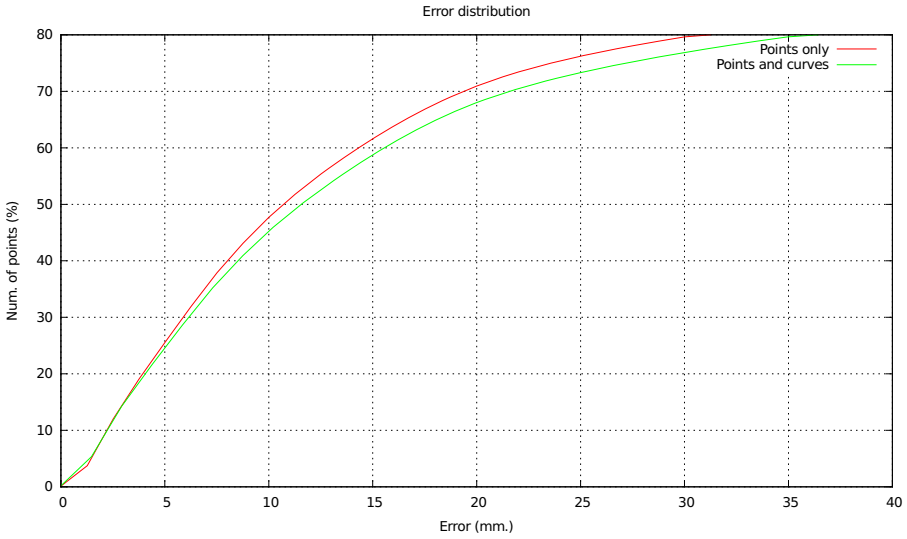
The results for the *fountain-p11* data set can be seen on the figure VII.6. There is no noteworthy difference between the surface computed with the 3D points alone and the surface computed using the mix of points and curves. The distribution of errors obtained by comparing the output with the ground truth can be seen on the figure VII.5a. The surface computed with the points alone is slightly better than the surface produced by points and curves, but the difference between the two is small (about $2 - 5$ mm.).

The results for the *herzjesu-p8* can be seen on the figure VII.7. Once again, there is almost no visual difference between the two surfaces. The results of the comparison with the ground truth are visible on the figure VII.5b. This time the difference between the surface with and without curves is even smaller.

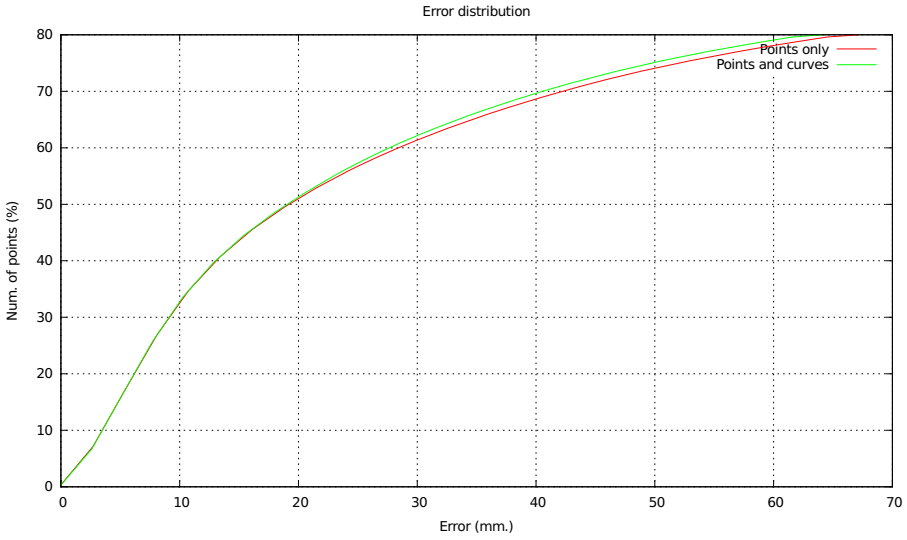
In conclusion, the *batch sparse* surface reconstruction method provides honorable results for these data sets, so it can be a good choice when the speed of computation is at least as important as precision. On the other hand, adding curves provides no significant improvement for these examples.

VII.5.3 *Hall* and *Aubière-2* data sets

Hall and *aubière-2* data sets are two large data sets taken in and in the proximity of Institut Pascal with a *PointGrey Ladybug 2* omnidirectional camera. For memory, *Ladybug* is a rigid multi-camera system consisting of six synchronized cameras each of which takes 1024x768 images at 15 frames/second. Unfortunately, ground truth is not available for these acquisitions, so we provide only qualitative results. This allows us to evaluate the influence of the curve related additional steps on the final surface quality in a realistic setting.

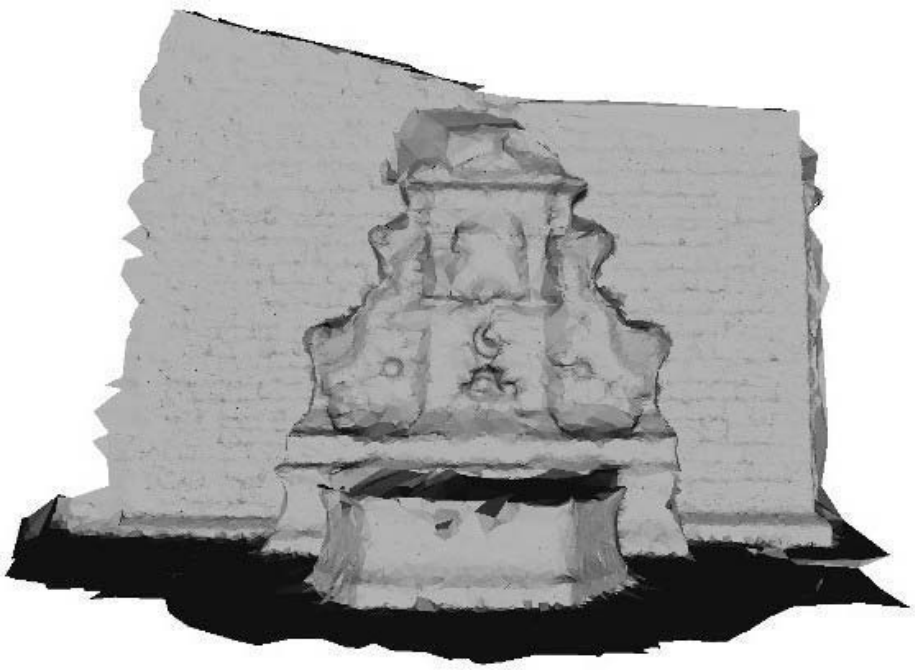


(a) *Fountain-P11* data set.

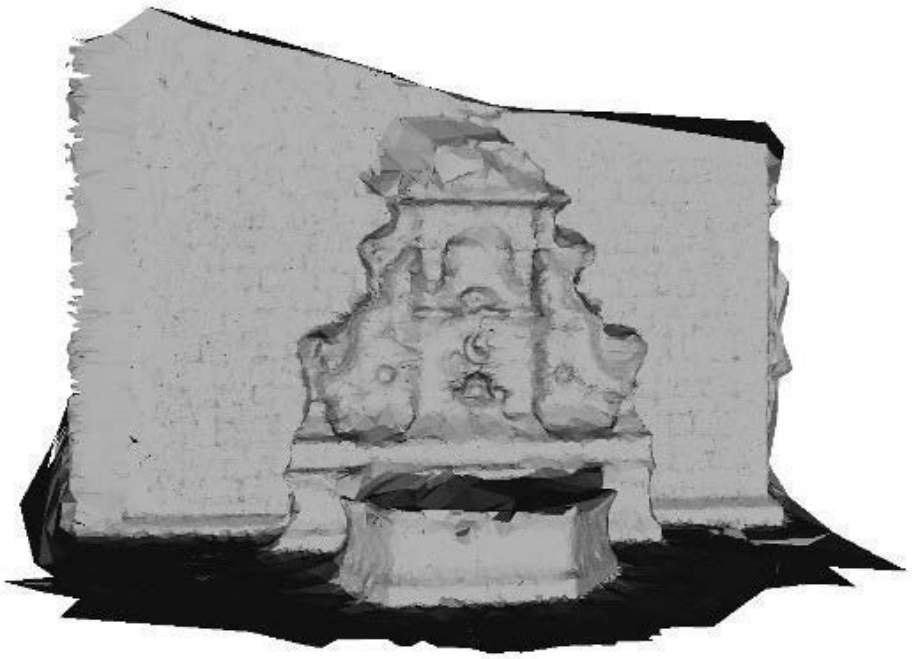


(b) *Herzjesu-P8* data set.

Figure VII.5: Error distributions for *Fountain-P11* and *Herzjesu-P8*

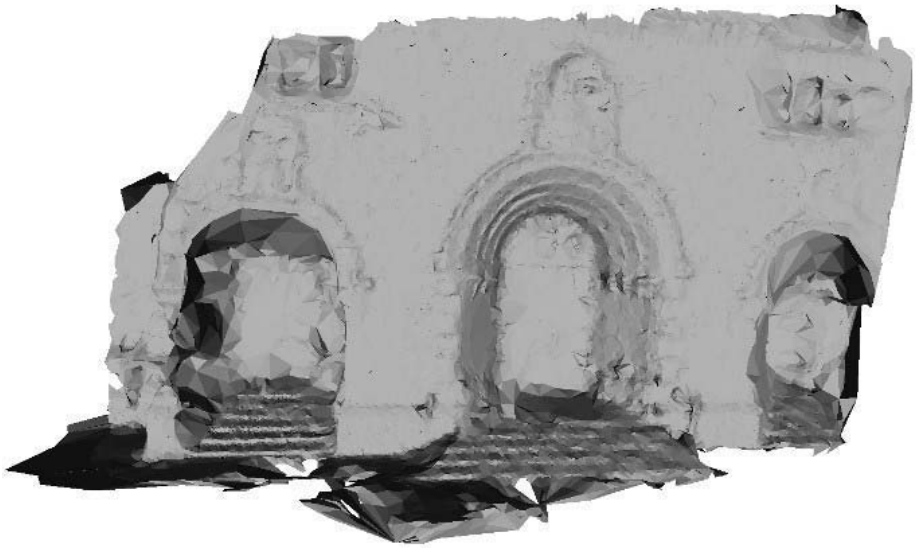


(a) Surface reconstructed using the 3D points only.



(b) Surface reconstrcuted using the 3D points and curves.

Figure VII.6: *Fountain-P11* data set reconstruction results.



(a) Surface reconstructed using the 3D points only.

(b) Surface reconstrcuted using the 3D points and curves.

Figure VII.7: *Herzjesu-P8* dataset reconstruction results.

Hall sequence is roughly 20 m. long and contains 1211 frames. It was taken in one of the rooms of the Institut Pascal building and is a typical highly cluttered robotic laboratory hall. There are 117 selected *key frames*; the total processing time is about 19 min. including curves (the main computation part is due to *SfM*). Reconstruction of the representative scene can be seen on the figure VII.8: the subfigure VII.8a contains a real image of the scene for reference. We see that mixing points and curves provides a better result than the usage of the points alone.

Aubière-2 sequence is around 700 m. long and contains 3140 frames. It is a typical urban environment similar to the *aubière* sequence of the subsection V.1, but is shorter. There are 655 selected *key frames* and the total processing time is about 56 min. (once again, the main computation part is due to *SfM*). The results are in the figure VII.9. Contrary to the previous sequence, the curves don't provide any significant improvement over the points alone.

VII.6 Conclusion

This chapter have reviewed the results of our attempt to integrate curves to the surface reconstruction pipeline. The contributions that arise from this PhD work and published in [Litvinov12] are:

1. The evaluation of the *2-manifold* surface sparse reconstruction method described in [Lhuillier13] against *ground truth* in some common multi-view stereo data sets;

2. The evaluation of the enhancement provided by mixing the input point cloud of this algorithm with reconstructed curves.

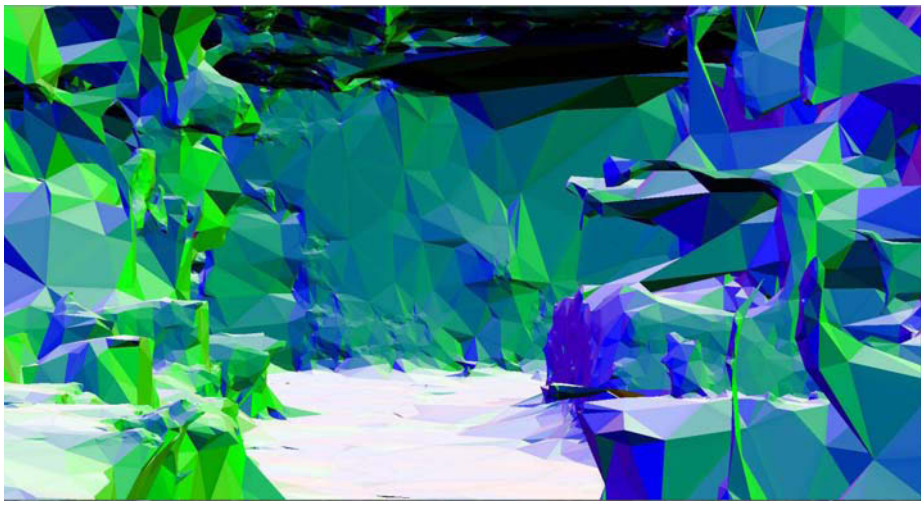
According to the comparison of the output surface with the *ground truth*, the precision achieved by the sparse surface reconstruction method cannot yet compete with the dense stereo methods, but we think that it is sufficient to be used as initialization of dense stereo. Furthermore, given the achieved 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.

On the other hand, mixing the curves with the points of interest provides some significant enhancement if input images are slightly textured as in *Temple* and *hall*. But it is of little interest in the textured case, as for *Fountain-P11* and *Herzjesu-P8*. So there are currently no general conclusion and the real benefits must be evaluated for each sequence individually. Nevertheless, it will be interesting to perform more investigations on this topic as better results can certainly be achieved.

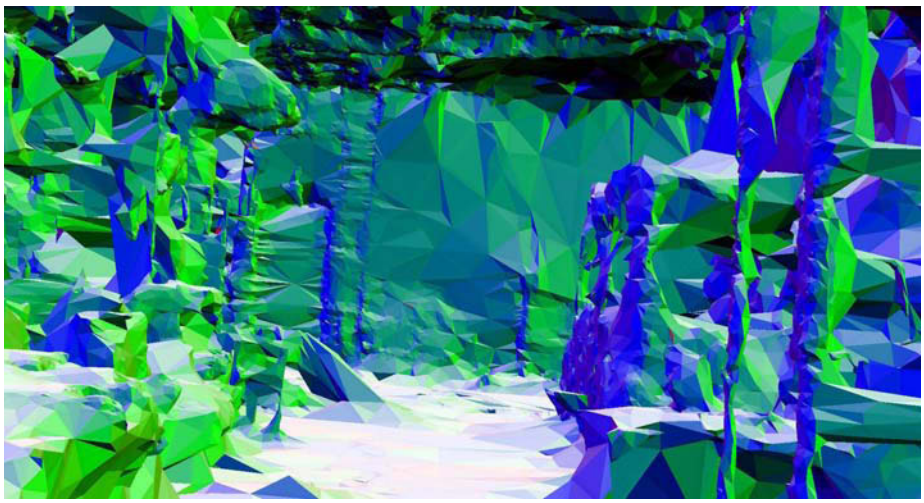
All the experiments of this chapter were performed using the *batch* algorithm. There are two reasons: the curves reconstruction is easier to implement in this context and the computation are faster.

We haven't performed the *incremental* experiments because the impact of the curves on the final surface quality wasn't deemed enough.

(a) Viewed part of the *hall*.

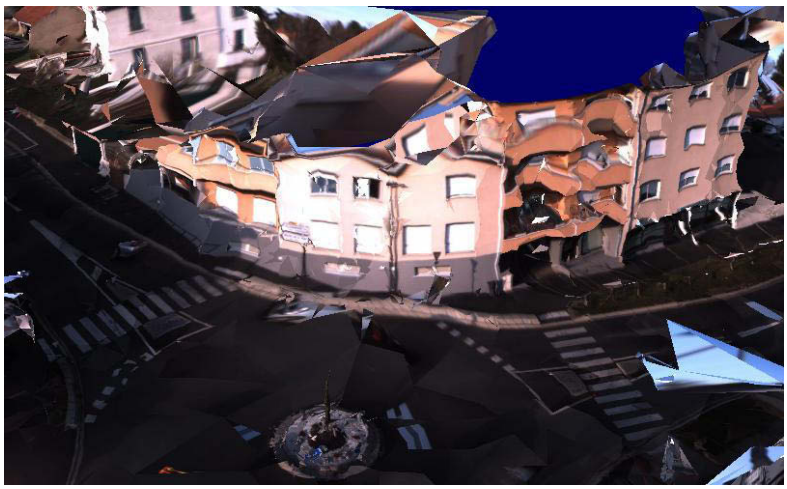


(b) Scene reconstructed using points only.

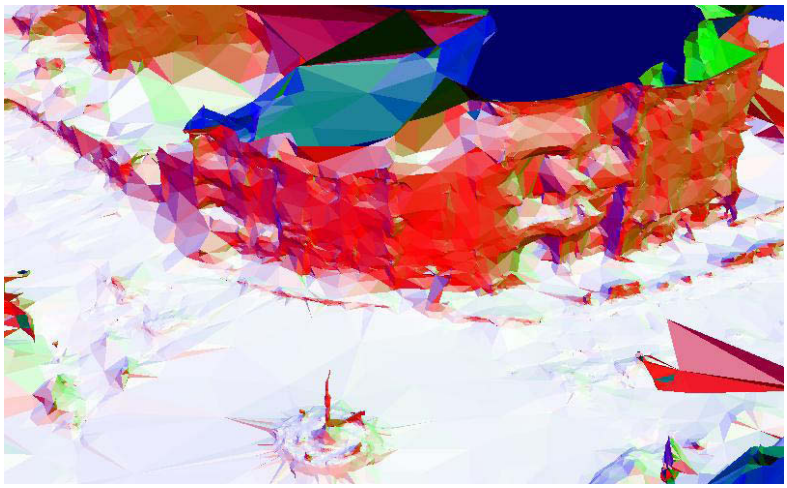


(c) Scene reconstructed using points and curves.

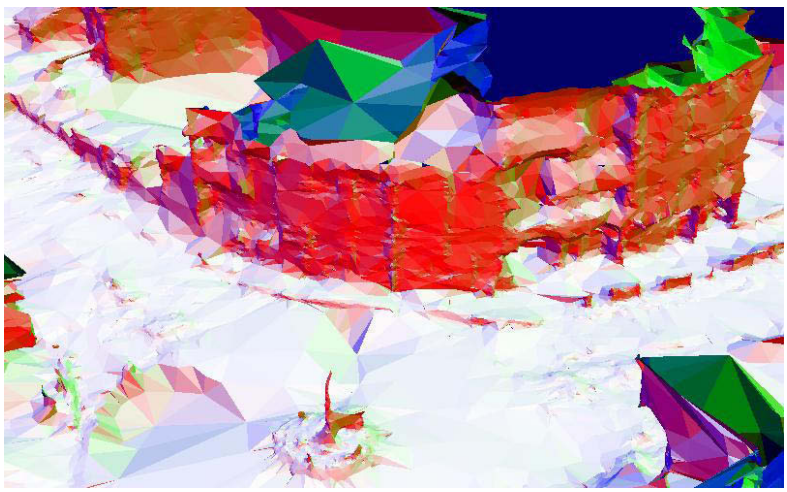
Figure VII.8: *Hall* data set reconstruction results.



(a) Textured version of the resulting mesh using points only.



(b) Resulting mesh using points only.



(c) Resulting mesh using points and curves.

Figure VII.9: *Aubière-2* data set reconstruction results.

CHAPTER VIII

Conclusion

Now we summarize the results obtained during the work on this dissertation, discuss its potential applications and overview the possibilities of further research.

VIII.1 Summary

The objective of this work was to develop a surface reconstruction method having three key characteristics: being *sparse*, being *incremental* and producing a *2-manifold* output surface. We have begun by reviewing the other works about surface reconstruction in chapter II. We have seen that the number of the *sparse* methods is relatively low compared to the number of *dense* ones. In the same manner, there are relatively few *incremental* methods. We have found only one other method having the same characteristics [Yu12], but it has a severe limitation: if the camera trajectory crosses itself, the entire loop must be reconstructed. So, we have concluded that a *sparse*, *incremental* and *2-manifold* enforcing method would be a useful addition to the state of the art.

Therefore, we have developed such a method inspired by a *batch* method from [Lhuillier13]. The overall processing has two steps: cloud of 3D points computation from the input video sequence and surface computation from the point cloud. The *Structure-from-Motion* algorithm used to compute the cloud of points and the associated visibility information is described in chapter III. Moreover, we have also described optional additional steps useful to enhance the output cloud. Then, in the first section of chapter IV, we have reviewed in details the base-line *batch* surface reconstruction algorithm. Finally, our own *incremental* surface reconstruction algorithm is described in the section IV.2.

Once the algorithm was detailed, we have performed a study of its complexity (section IV.3). We have concluded that under the tight assumptions, the complexity of all the steps of a single iteration (except the Steiner grid update) are bounded. This is interesting since the complexity of the algorithm is independent of the camera trajectory properties (compared to [Yu12]). We have also run experiments with the algorithm using real and synthetic data sets. We have seen, that the *incremental* version output surface quality is very similar to the quality of the base-line *batch*

algorithm. Unfortunately, the computation speed is, albeit being bounded, too long for reconstruction to be performed on-line. This limits the usefulness of the algorithm. We have also seen that optional additional steps of the *Structure-from-Motion* can significantly improve the surface quality, but the processing becomes even slower.

Hence, we have made a tentative to solve this problem. When looking at the execution times of each step of a single iteration, it was seen that the slowest part of the algorithm is the *artifacts removal* post-processing step. Therefore, the most obvious way to boost the algorithm computation speed is to improve the speed of the *artifacts removal*. This is the goal of chapter VI. Here we propose two other algorithms to solve this problem, they are detailed and their complexities are studied. Their tight complexity is bounded. Then, they are experimented on a real video sequence. The new methods are much faster than the previous algorithm, but taken separately, they provide lower quality. When combined they are still faster and the output quality is almost the same as previously measured. So, we have concluded that the combination of the two new methods is a good drop-in replacement of the previous algorithm even for the *batch* base-line algorithm [Lhuillier13].

Finally, we add curves in addition to the interest points to our surface reconstruction pipeline in the hope that this improve the output surface quality. This work is detailed in chapter VII. We have reviewed the previous works on the surface reconstruction methods that make use of curves and we developed our own steps to add them to the reconstruction process. Then we have performed the experimentations. First, we have tested our algorithm on the classical multi-view data sets. We have shown that the curves can improve the output surface quality if the input surface has low texture. The same experiments were performed on interior and exterior data sets and the conclusion was the same.

VIII.2 Results and potential applications

The work on this thesis was published in three international [Litvinov12, Litvinov13, Litvinov14b] and one french [Litvinov14a] publications. The two major contributions are an *incremental sparse 2-manifold* surface reconstruction method and new visual artifacts/handle removal algorithms useful in the *incremental* and *batch* cases. The secondary contributions are adaptation of the *batch* surface reconstruction method from [Lhuillier13] to the rigid multi-camera system, study of the influence of the input resolution on the output surface quality, additional acute tetrahedra removal step, evaluation of the *batch* method on the standard data sets, and study of the curves integration on the output surface quality.

An obvious practical application for an *incremental* method would be an *on-line* surface reconstruction, but unfortunately, even after coding optimization, the computation time of a single iteration is too large for real time processing. Nevertheless, our method has an interesting application due to the bounded nature of its iterations.Since all the processing is confined to the *working zone* (except the Steiner grid update), only the data structures containing these tetrahedra must be in memory at every given time. This means that not only the time, but the space complexity is also bounded.

In practice, if the algorithm is correctly implemented, its memory consumption is constant, so it is naturally well suited for the reconstruction of the very large scale scenes. Moreover, the memory consumption is further reduced by the fact that our

method is *sparse* and so its data structures are more compact.

VIII.3 Future works

The work performed during the preparation of this dissertation have lead to some interesting results, but of course, there are still a lot of open topics waiting to be explored. Now, we present a non exhaustive list of interesting directions for further works.

VIII.3.1 *Structure-from-Motion*

There is a problem inherent to the *bundle adjustment* based *Structure-from-Motion* algorithms. The numerical errors tend to accumulate over time, and so the computed *camera* positions drifts slightly from the real ones. In practice, this means that when the camera passes two or more times at the same spot, the points won't be reconstructed at the same place and so the output surface quality will be degraded and even lead to a totally erroneous reconstruction.

A solution to this problem is to detect when the camera passes by an already reconstructed place, perform an image matching between the images of the same point of the scene and perform a global *bundle adjustment* with the additional constraints provided by this matching. This process is called loop detection and closure. So it would be interesting to solve this problem in an *incremental* manner.

It would also be interesting to experiment with the other interest points detection and matching algorithms for our *Structure-from-Motion* implementation. Currently, we use *Harris* [Harris88] points detector and *ZNCC* based matching. But, other detectors (such as *SIFT* [Lowe04], *SURF* [Bay06] or *FAST* [Rosten06]) and matching algorithms could be used and eventually enhance the output surface quality or the reconstruction speed.

VIII.3.2 Surface reconstruction

There are also several ways to improve the surface reconstruction algorithm. The first direction would be the theoretical foundations of the method. In fact, currently we are unable to prove that the *region growing* algorithm (subsection IV.1.5) is able to reach every possible surface in the 3D Delaunay triangulation. This is also true in the simple case when the surface to reconstruct is homeomorphic to a sphere. There is the same problem for the *shrinking* step (subsection IV.2.3): we are unable to prove that any surface can correctly be shrunk.

Another direction for the further works would be the algorithm computational time. The computation time is currently too long for on-line applications. We could, for example, try to replace the *ray tracing* based binary labeling step by an energy based algorithm such as [Hoppe13] that could be faster.

Finally, the output surface quality of the method could also be improved. For example, we can use a more sophisticated denoising method (Laplacian-Beltrami [Botsch10], for instance). We can also try to make use of the uncertainty information that can be provided by the *Structure-from-Motion* in our reconstruction pipeline.

VIII.3.3 Curves

Finally, there are still a lot of work to do concerning the integration of curves in the surface reconstruction pipeline. Our initial experimentation have confirmed that the curves can significantly improve the reconstructed surface quality in some particular cases, namely when the observed scene is slightly textured. However, results of papers such as [Wu05] suggests that much better results could be achieved.

For this, the curves matching algorithm performance should be improved or a better a way to reconstruct the curves without matching should be found. For example, we could try to develop a more large scale scenes friendly version of [Teney12]: i.e. a method that wouldn't require a regular subdivision of space.

Finally, a better way to sub-sample the curves into a set of points when inserting them into the Delaunay triangulation can be found. Currently, one point over s in a curve is inserted where s is a user defined threshold. It would be interesting to find a way to dynamically adjust s so that we have a guarantee that the edge exists in the Delaunay.

APPENDIX A

Notes on the parallel processing

In the recent years, a general tendency in the computer hardware was to multiply the number of CPU cores, i.e. the number of the processing units available to the user program. So today, for an algorithm to use the modern hardware capacity in an optimal manner, it is important to be easily adaptable to the parallel processing, i.e. to execute several parts of the algorithm simultaneously.

In the present appendix we provide a brief overview of the parallel processing opportunities available in our incremental surface reconstruction algorithm. They are located in the *free-space/matter* binary labeling part (see section A.1), in the *topology extension* (see section A.2) and in the *Structure-from-Motion* (see section A.3). Finally, a small example of how to implement a parallel code in practice is given in section A.4. All the experiments found in this dissertation were performed on a six cores computer taking advantage of the parallel processing techniques presented here.

A.1 *Free-space/matter* binary labeling

The first opportunity to take advantage of the multiple CPU cores during the execution of our surface reconstruction algorithm is located in the *free-space/matter* binary labeling part. For memory, the *free-space/matter* binary labeling step (see subsection IV.1.2) consists in updating the number of intersections for each tetrahedron of the Delaunay triangulation. For this, we select the rays that eventually intersect new tetrahedra and we follow them. Each time a new tetrahedron is intersected, its number of intersections value is incremented.

It is easy to see that following a ray is totally independent from following another ray, this process didn't modify any data structure during it execution. So several rays can be followed in parallel without any risk. Moreover, checking that a tetrahedron is new is also without side effects and so can be performed in parallel without any problems.

The only potentially dangerous operation during this process is the update of the number of intersections. Fortunately, this is not a problem in practice because most of the multi-processing frameworks out there (such as *OpenMP* [Ope] that

we used) allows to perform an increment of an integer as an *atomic* operation. The operation is said to be *atomic* if it is guaranteed that the current process won't be interrupted during it execution.

In conclusion, the *free-space/matter* binary labeling is an excellent opportunity for parallel processing because it is easy to achieve in practice and following of each ray is a perfectly independent operation.

A.2 Topology extension

There are other interesting opportunities for parallelization: the *topology extension* (which is used in both the *outside* region growing and shrinking steps).

In fact, *one by one* region growing is a bad choice for parallel processing for two reasons. First, adding a new tetrahedron operation depends heavily on the results of the previous one. Adding a new tetrahedron to *outside* enables eventually the addition of another. Second reason is that this sub-step is fast compared to *topology extension* and so, even if we parallelize it, the computation time gain will be small.

On the other hand, *topology extension* presents an interesting opportunity. For memory, during the *topology extension* (see subsection IV.1.5) we check each vertex located on the boundary of the *outside* region. For each vertex, we try to add the *free-space inside* tetrahedra incident to it to the *outside*. Then we check if the boundary ∂O of the *outside* is still *2-manifold*. If it is true, we have succeeded and the algorithm stops, otherwise we check another vertex.

The majority of the computation time of this step is the unsuccessful trials of different vertices. So we save a lot of time if we perform several trials in parallel. Unfortunately, this is not trivial because these operations are interdependent. In fact, when we add a pack of tetrahedra to the *outside*, it modifies our capacity to add another pack without breaking the *manifold* property.

Fortunately, there is a way to get rid of this problem. In fact, the only interdependency is during a trial of a particular vertex. So the problem can be solved by using a particular version of the *manifold* test. Instead of adding the tetrahedra to the *outside* region (and so modify the global data structures) and then performing a test, we can perform a test without modifying the triangulation.

To achieve this goal, we pass to the test routine the list of tetrahedra we want to add. During its internal processing, the routine will consider these tetrahedra as *outside* even if they are labeled *inside* in the triangulation. This way, each check could be performed independently. Once a good vertex was found, the process stops and only the status of tetrahedra around this vertex is changed to *outside*.

In conclusion, the *topology extension* provides a good opportunity for taking advantage of the several CPU cores provided by the modern processors because it is relatively easy to implement. The only problem is that the algorithm becomes non deterministic: there are in general multiple packs of tetrahedra that can be added to *outside* without breaking its *manifold* property. However, when multiple trials are performed in parallel, the pack which is actually chosen becomes random. During one particular execution, the algorithm will choose one particular pack of tetrahedra at a given time. When the algorithm is executed a second time, the pack chosen at this given time is different.

A.3 Structure-from-Motion

Another set of opportunities to take advantage of the multiple CPU cores during the execution of our surface reconstruction algorithm are located in the *Structure-from-Motion* part. Particularly, we can cite the interest points matching and the *bundle adjustment*.

During the interest points matching step (see subsection III.2.1), for each point $\mathbf{q}_i$ of the current image I_i we find the corresponding point $\mathbf{q}_{i-1}$ of the previous image I_{i-1} if it exists. To achieve this goal, we compute the correlation value (*ZNCC*) between $\mathbf{q}_i$ and the list of potential candidates. Then, the candidate with the highest correlation value is chosen. As can easily be seen, the search of match for a given point has no side effects. So the matching of several points can be performed in parallel.

The same reasoning holds for the robust new *pose* initial estimation (see subsection III.2.6). The *pose* is estimated many times using different selection of points, then the *pose* having the best score is kept. Each estimation is completely independent and have no side effects. So we can safely perform many estimation in parallel.

Finally, we parallelize the estimation of the Hessian matrix during the *bundle adjustment* step (see subsection III.2.7). Computations of the cells of this matrix are independent and so we can perform many of them in parallel.

A.4 Parallel processing in practice

Multiple CPU possessing machines have appeared a long time ago. But, writing a parallel application was a difficult task. Fortunately, nowadays, there are a lot of *frameworks* facilitating it as much as possible.

During the work on this dissertation, the algorithms were written in *C++*. To actually implement the parallel processing considerations of this appendix, we have used the *OpenMP* [Ope] *framework*. This is a language extension and a library implemented today by all the major compilers. It has the advantage to be very easy to use in simple cases.

We illustrate it on the matrix multiplication example. Consider two matrices: $A_{m,n}$ and $B_{n,m}$. We want to compute a matrix $C_{m,m}$ such as $C = A \times B$. For memory, each cell $C(i,j)$ with $i \in [1;m]$ and $j \in [1;m]$ of the resulting matrix is computed using the following equation:

$$C(i,j) = \sum_{k=1}^{n} A(i,k) \times B(k,j) \tag{A.1}$$

In plain *C++*, a function computing this matrix will look like this:

```
void matrix_multiply(int m, int n, float* A, float* B, float* C) {
   for(int j = 0; j < m; ++j)
      for(int i = 0; i < m; ++i) {
         C[i + m*j] = 0;
         for(int k = 0; k < n; ++k)
            C[i + m*j] += A[i + m*k]*B[k + n*j];
      }
}
```

The main loop of this function is composed of two *for* statements iterating over the lines and columns of the output matrix. The body of the loop computes the value of a given cell. It is easy to see that this computation is independent of the results of other cells. So each iteration of the main loop can be performed in parallel.

Thanks to *OpenMP*, we only need to add a single line of code to say it to the compiler:

```
void matrix_multiply(int m, int n, float* A, float* B, float* C) {
   #pragma omp parallel for
   for(int j = 0; j < m; ++j)
      for(int i = 0; i < m; ++i) {
         C[i + m*j] = 0;
         for(int k = 0; k < n; ++k)
            C[i + m*j] += A[i + m*k]*B[k + n*j];
      }
}
```

The *pragma* statement tells to the compiler that the iterations of the following *for* statement can be performed in parallel. So when the code is executed, the program will use all the available CPU cores to lunch in parallel as much iteration as possible.

This code must be compiled with *-fopenmp* option if *GCC* compiler is used. For other compilers, refer yourself to their documentation.

APPENDIX B

Details of the 2-manifold tests

In the incremental surface reconstruction algorithm (the chapter IV) we need a way to check if the addition of a single tetrahedron or a bunch of tetrahedra won't modify the *2-manifold* property of the boundary ∂O of the *outside* region. For this, we use several implementations of the *manifold tests*.

At first glance, only one implementation of the manifold test is enough: the general one that can check the addition or removal of the arbitrary number of tetrahedra. But in practice, we find that the *manifold tests* consume the majority of the computation time of the *region growing* process. So we implemented a faster test that is only applicable when a single tetrahedron is added to or removed from the *outside* region.

In this appendix, we review the implementation details and the complexity of the *2-manifold* tests. We begin by the *slow* general test (section B.1), then the *fast* single tetrahedron addition and subtraction tests (section B.2).

B.1 General manifold test

B.1.1 General test implementation

In theory, a vertex $\mathbf{v} \in \partial O$ is regular (i.e. the surface is *2-manifold* around this vertex) if and only if the triangles in ∂O including $\mathbf{v}$ can be ordered as $t_0, \cdots, t_{k-1}$ such as $t_i \cap t_{(i+1) \bmod k}$ is an edge, and such an edge is included in exactly two triangles t_i and t_j [Goodman04].

Unfortunately, a direct implementation of such a test would be slow. Instead, we prefer to use the algorithm proposed in [Lhuillier13]. The idea is to use the adjacency graph Γ_T of the Delaunay triangulation T. For a vertex $\mathbf{v}$ of ∂O, we note $\gamma_\mathbf{v} \subseteq \Gamma_T$ the graph of tetrahedra having $\mathbf{v}$ as vertex. In [Lhuillier13], it was proven that the vertex $\mathbf{v}$ is regular if and only if all the *inside* tetrahedra of $\gamma_\mathbf{v}$ are connected and all the *outside* tetrahedra of $\gamma_\mathbf{v}$ are connected.

So, in practice, to check that a vertex $\mathbf{v}$ is regular, it is sufficient to remove the edges between *inside* and *outside* tetrahedra in $\gamma_\mathbf{v}$ and check that it contains exactly 2 connected components. Because *CGAL* already encodes the Delaunay triangula-

tion as a graph, the implementation of this algorithm is fast and straightforward.

B.1.2 Complexity analysis

Checking a single *non infinite* vertex $\mathbf{v}$ is traversing the graph $\gamma_{\mathbf{v}}$. So the complexity of checking a single vertex is $\mathcal{O}(d)$ under the loose assumptions. Under the tight assumptions, $d = \mathcal{O}(1)$ thanks to assumption T6, so the complexity of testing a single *non infinite* vertex is $\mathcal{O}(1)$.

We use a general test to check the *manifold* property when adding or substracting a pack of tetrahedra. Let n be the number of tetrahedra in this pack. Each tetrahedron have exactly 4 vertices, so the number of vertices in the pack is $\mathcal{O}(n)$. Thus the complexity of a general manifold test is $\mathcal{O}(nd)$ under the loose and $\mathcal{O}(n)$ under the tight assumptions.

B.2 Addition/subtraction tests for one tetrahedron

If the boundary ∂O of the *outside* region is already *2-manifold* and we want to add a single tetrahedron to it, we can use a faster implementation of the *2-manifold* test. This *fast* test is based on the *2-manifold* test proposed in [Boissonnat84]. The basic idea is to check that the tetrahedron can be added using a condition based on the neighborhood configuration of the tetrahedron.

When we want to add a tetrahedron, we use the following test:

Property B.1 (Addition test)
Assume that ∂O is 2-manifold. *Let Δ be a finite tetrahedron in $\mathcal{T} \setminus O$ and f be the number of Δ-triangles included in ∂O. Then, the boundary $\partial(O \cup \{\Delta\})$ of $O \cup \{\Delta\}$ is* 2-manifold *if and only if one of the following conditions is meet (we note $O_{\mathbf{v}}$ the set of* outside *tetrahedra having $\mathbf{v}$ as vertex):*

- *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 ∂O meets $O_{\mathbf{v}} = \emptyset$;*
- *if $f = 2$ and the Δ-edge which is not in ∂O has end-vertices $\mathbf{v}$ and $\mathbf{w}$ such that $O_{\mathbf{v}} \cap O_{\mathbf{w}} = \emptyset$;*
- *if $f = 3$ or 4.*

This addition test is efficient because we only need to access the list of the *outside/inside* status of the tetrahedra incident to each of the vertices of the tetrahedron to test. This avoids us the incidence graph traversal. Under the loose assumptions, the complexity of this test is $\mathcal{O}(1)$ if $f = 3$ or 4 and $\mathcal{O}(d)$ otherwise. So the complexity of the test is $\mathcal{O}(d)$. It is $\mathcal{O}(1)$ under the tight assumptions thanks to the assumption T6.

When we want to subtract a tetrahedron, we can use a similar test:

Property B.2 (Subtraction test)
Assume that ∂O is 2-manifold. *Let Δ be a finite tetrahedron in O and f be the number of Δ-triangles included in ∂O. Then, the boundary $\partial(O \setminus \{\Delta\})$ of $O \setminus \{\Delta\}$ is* 2-manifold *if and only if one of the following conditions is meet (we note $\bar{O}_{\mathbf{v}}$ the set of* inside *tetrahedra having* $\mathbf{v}$ *as vertex):*

- *if $f = 0$ and every Δ-vertex $\mathbf{v}$ meets $\bar{O}_{\mathbf{v}} = \emptyset$;*
- *if $f = 1$ and the Δ-vertex $\mathbf{v}$ which is not in ∂O meets $\bar{O}_{\mathbf{v}} = \emptyset$;*
- *if $f = 2$ and the Δ-edge which is not in ∂O has end-vertices $\mathbf{v}$ and $\mathbf{w}$ such that $\bar{O}_{\mathbf{v}} \cap \bar{O}_{\mathbf{w}} = \emptyset$;*
- *if $f = 3$ or* 4.

This test has the same complexity as the addition test, namely $\mathcal{O}(d)$ and $\mathcal{O}(1)$ under the loose and tight assumptions.

APPENDIX C

Proof of the maximum vertex degree

In this brief appendix, we provide a proof of the maximum vertex degree d (the maximum number of tetrahedra incident to a single ***non infinite*** vertex) under the loose assumptions. We call n the number of vertices in the 3D Delaunay triangulation. Trivially, $d = \mathcal{O}(n^2)$ because the number of tetrahedra in the Delaunay triangulation is $\mathcal{O}(n^2)$. Here, we show that $d = \mathcal{O}(n)$.

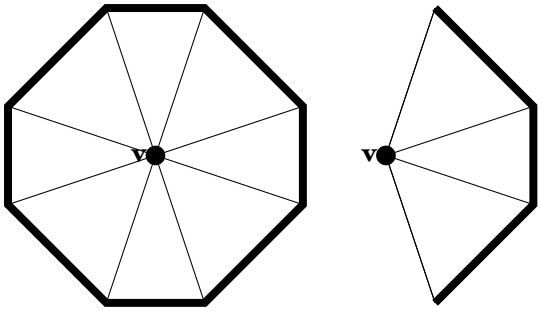
Let $\mathbf{v}$ be a vertex of the 3D Delaunay triangulation T, so $\mathbf{v} \neq \mathbf{v}_\infty$.

Case 1: If $\mathbf{v} \notin \partial T$ (figure C.1a), $\mathbf{v}$ is an internal vertex of T and the set of tetrahedra $\mathbf{v}t_i$ (where t_i is a triangle) are such that the set of t_i forms a *2-sphere* [Boissonnat98].

For a ***manifold*** surface (and a *2-sphere* is ***manifold***), we call n_v the number of vertices, n_e the number of edges, n_t the number of triangles and g it genus. We use the *Euler* equation [Botsch10]:

$$n_v - n_e + n_t = 2(1 - g) \tag{C.1}$$

In our case $g = 0$. $n_t = d$ because each triangle of the *2-sphere* corresponds to a single tetrahedron adjacent to $\mathbf{v}$. Moreover, because each edge is adjacent to



(a) $\mathbf{v} \notin \partial T$ (b) $\mathbf{v} \in \partial T$

Figure C.1: Illustration of why the set of tetrahedra having $\mathbf{v}$ as vertex forms a *2-ball* (2D case). Thick black line is the border of the triangulation.

exactly two triangles $2n_e = 3n_t$. So the equation become:

$$\begin{aligned} n_v - \frac{3}{2}n_t + n_t &= 2 \\ n_t &= 2n_v - 4 \end{aligned} \tag{C.2}$$

As was said before $n_t = d$ and $n_v = \mathcal{O}(n)$. Thus $d = \mathcal{O}(n)$.

Case 2: If $\mathbf{v} \in \partial T$ (figure C.1b), the set of t_i form a *2-ball* whose boundary edges $e_1, \ldots, e_k$ meets $\mathbf{v}e_i \in \partial T$. After the corollary 11.2.3 of [Boissonnat98], the number of triangles in planar triangulation (of a *2-ball*, for instance) is linear to the number of vertices. Thus, $d = \mathcal{O}(n)$.

ANNEXE D

Résumé étendu en français

D.1 Introduction

Le calcul de la surface d'une scène à partir d'un ensemble d'images est un problème classique en vision par ordinateur. La majorité des algorithmes existants peuvent être décomposés en deux étapes. Premièrement, les poses de la caméra associées aux prises de vue sont estimées par un algorithme dit de *Structure-from-Motion* (ou *SfM*). Dans un deuxième temps, la profondeur de chaque pixel de chaque image est calculée, cela définit un nuage de points 3D dense, et la surface finale est estimée à partir de ce nuage de points. Cependant, le *SfM* fournit déjà un nuage de points épars calculé simultanément avec les poses, qu'il serait intéressant d'utiliser directement dans l'étape d'estimation de surface. Ceci est un des objectifs principaux de ce travail de thèse.

On a développé un algorithme de reconstruction des surfaces qui combine trois caractéristiques principales : il est *épars*, *incrémental*, et produit une surface ayant la propriété de *2-variété*. Par *épars*, on comprend que les calculs 3D s'effectuent uniquement sur une minorité de pixels bien choisis dans les images (là où il y a le plus d'information), plutôt que d'appliquer uniformément le même traitement à l'ensemble des pixels. Ceci permet à la fois de réduire les temps de calculs, la place mémoire et de stockage de la surface, mais aussi d'obtenir des modèles 3D compacts (car simplifiés) pour des scènes complexes. La méthode est dite *incrémentale* parce qu'elle traite les images au fur et à mesure de leurs arrivée. Plus précisément, la surface de la scène est complétée à chaque nouvelle image lue dans la vidéo. Finalement, la surface obtenue à tout instant est une *2-variété*, c'est à dire que tout point de la surface a un voisinage dans la surface topologiquement équivalent à un disque. Cette propriété permet de nombreux traitements et utilisations ultérieures.

D.2 Contributions

Dans ce travail de thèse, il y a deux contributions majeures et un certain nombre de contributions secondaires. Elles ont été présentées dans trois conférences internationales et une conférence nationale. Les contributions majeures sont :

- Une méthode *incrémentale* et *éparse* de reconstruction de surfaces avec la garantie que la surface produite est une *2-variété*. Contrairement au travail précédent [Yu12], le temps d'une itération (pour chaque nouvelle image) est borné en pratique, même si la trajectoire comprend une ou plusieurs boucles. Ce travail a été publié dans [Litvinov13] et [Litvinov14a].
- Un nouvel algorithme de suppression d'artefacts visuels. Ses performances sont similaires à l'existant [Lhuillier13], mais il est beaucoup plus rapide et plus facile à utiliser dans notre contexte incrémental. Il a été publié dans [Litvinov14b].

Les contributions secondaires sont :

- L'algorithme de reconstruction de surfaces *épars* de [Lhuillier13] a été adapté pour l'utilisation d'un système rigide de plusieurs caméras au lieu d'une caméra catadioptrique.
- L'algorithme de sélection d'images *clés* de [Mouragnon09] a été amélioré grâce à l'introduction d'un critère supplémentaire.
- L'influence de la résolution des images d'entrée a été étudiée.
- Une étape supplémentaire (suppression des tétraèdres aigus) améliorant la qualité du résultat final a été mise au point.
- Les performances de la méthode *non incrémentale* [Lhuillier13] ont été évaluées en utilisant les jeux de données standards [Seitz06, Strecha08]. Ceci permet la comparaison avec les autres méthodes. Les résultats ont été publiés dans [Litvinov12].
- L'intérêt d'introduire des courbes (en supplément des points d'intérêt) dans le processus de reconstruction a été étudié. Les résultats ont été publiés dans [Litvinov12].
- Certains des calculs sont effectués en parallèle sur plusieurs processeurs (disponibles sur les PC standards) ce qui accélère notre méthode.

D.3 État de l'art

La reconstruction de surfaces à partir d'images est un vaste sujet qui a donné lieu à un très grand nombre de publications. Globalement, les méthodes existantes peuvent être séparées en deux grandes catégories : les méthodes *denses* et les méthodes *éparses*. Les méthodes *denses* utilisent l'ensemble des pixels des images d'entrée, alors que les méthodes *éparses* essayent de trouver des parties << intéressantes >>. Chacune de ces deux catégories peut être subdivisée en méthodes *globales* et *incrémentales*. Les méthodes *globales* ont besoin de l'ensemble de la séquence d'image pour produire une surface, tandis que les méthodes *incrémentales* mettent à jour une surface au fur et à mesure que les images sont lues dans la vidéo.

Les méthodes *denses* sont les plus nombreuses, particulièrement *globales*. On peut les subdiviser en méthodes travaillant avec les images directement et les méthodes interpolant/approximant un nuage de points *denses*. Parmi les méthodes travaillant directement avec les images, on peut citer, par exemple, les méthodes basées sur les

voxels [Slabaugh04, Treuille04] ou basées sur les surfaces de niveau [Jin05, Pons07] (ou Level Sets). Parmi les méthodes travaillant à partir d'un nuage des points, on trouve celles basées sur la triangulation de Delaunay [Boissonnat84, Amenta01, Labatut09] et celles basées sur les fonctions implicites [Alliez07] ou indicatrices [Kazhdan06]. Les méthodes *denses incrémentales* sont assez peu nombreuses, on peut citer [Pollefeys08, Newcombe10].

Le nombre de méthodes *éparses* est bien inférieur au nombre de méthodes *denses*. Les méthodes *globales* entrant dans cette catégorie peuvent être séparées en méthodes utilisant la triangulation de Delaunay 2D [Morris00, Salman09] ou 3D [Pan09, Lhuillier13]. Les méthodes *éparses incrémentales* sont peu nombreuses : [Hilton05, Lovi10, Yu12, Hoppe13]. Parmi ces méthodes, uniquement [Yu12] produit une surface *2-variété*.

L'étude de l'état de l'art sur le sujet nous à donc amené à la conclusion qu'il existe une seule méthode [Yu12] réunissant toutes les caractéristiques de la méthode qu'on a développé durant cette thèse. Cependant, cette méthode possède une limitation importante : lorsque la trajectoire de la caméra filmant la scène passe par un endroit déjà vu, la surface de la boucle ainsi formée doit être complètement recalculée. Notre méthode n'a pas ce genre de limitations. Elle est donc une contribution intéressante à l'état de l'art.

D.4 Calcul du nuage de points 3D

Notre méthode de reconstruction de surfaces à partir d'une vidéo d'images peut être séparée en deux parties : le calcul du nuage de points 3D à partir des images et le calcul de la surface à partir du nuage des points. Pour calculer le nuage des points on utilise un algorithme de *Structure-from-Motion* [Mouragnon09].

L'algorithme est *incrémental* : il met à jour le nuage de points à chaque fois qu'une nouvelle image est acquise par la caméra. Une nouvelle image est traitée en utilisant les étapes suivantes :

- Les points d'intérêt sont détectés dans la nouvelle image.
- Ces points sont mis en correspondance avec les points d'intérêt de l'image précédente.
- Les résultats de la mise en correspondance sont utilisés pour mettre à jour l'ensemble des *pistes*. Une *piste* est l'ensemble des points 2D (ou observations) dans les images successives correspondant à un même point 3D de l'espace.
- On décide si l'image précédente est une *image clé*. Uniquement les observations des images *clés* sont conservées et traitées par les étapes suivantes. Ceci permet de diminuer le temps de traitement et de stabiliser numériquement les calculs.
- Si l'image précédente est *clé*, on effectue deux étapes supplémentaires :
  - On calcule la pose approximative de la caméra correspondant à la nouvelle image *clé* et les positions approximatives de nouveaux points 3D que l'on vient de détecter et de mettre en correspondance.
  - On raffine les poses et les points 3D les plus récents en appliquant un algorithme d'optimisation (appelé *ajustement de faisceaux* local).

Cet algorithme n'est pas le seul algorithme de *Structure-from-Motion* existant, mais l'avantage de celui-ci est qu'il a été mis au point et est très largement utilisé à l'Institut Pascal. On connait donc bien ses avantages et limitations.

Par défaut, on calcule et utilise uniquement les points visibles dans les images *clés*. En option, on peut ajouter des étapes supplémentaires pour calculer les poses des images intermédiaires et enrichir le nuage de points (qui reste épars). Mais cela a un prix : le temps de calcul augmente.

D.5 Reconstruction de la surface

L'algorithme de *Structure-from-Motion* décrit dans la section précédente nous fourni un certain nombre d'informations : la pose de la caméra correspondant à chaque image *clé*, un ensemble de points 3D et un ensemble de rayons. Un rayon est un segment reliant un point 3D à une position de la caméra qui l'a observé. On utilise toutes ces informations pour estimer une surface correspondant à la scène observée.

À chaque nouvelle image *clé*, le *SfM* calcule la pose de la caméra correspondante, un ensemble de nouveaux points 3D et un ensemble de nouveaux rayons. Notre but est de mettre à jour la surface à l'aide de ces informations. On procède de la façon présentée sur la Fig. D.1. On détaille maintenant chaque étape de ce processus.

D.5.1 État initial

Vu que notre méthode est *incrémentale*, son objectif principal est de mettre à jour la surface compte tenu de nouvelles informations, plutôt que de construire une nouvelle surface *ex nihilo*. On suppose donc qu'une partie de la surface a déjà été reconstruite à partir un nuage de points 3D correspondant au début de la séquence d'images.

À l'état initial (Fig. D.1.a), l'espace est subdivisé de façon non uniforme en un ensemble de tétraèdres grâce à une triangulation de Delaunay 3D du nuage de points. Chaque tétraèdre est soit *vide* soit *matière*. Si un tétraèdre est intersecté par au moins un rayon, on peut voir à travers, donc il est *vide*. Autrement, il est *matière*.

En première approximation, on peut considérer que la frontière entre les tétraèdres *vides* et *matière* est la surface recherchée. Mais la surface ainsi obtenue n'est pas une 2-variété. On introduit donc une deuxième classification des tétraèdres : les tétraèdres *extérieurs* et *intérieurs*. Les tétraèdres *extérieurs* sont *vides* et la frontière entre *extérieur* et *intérieur* est une *2-variété*. Cette frontière est la surface que l'on cherche.

D.5.2 Insertion des nouveau points 3D

On doit maintenant insérer un certain nombre de nouveaux points 3D. Quand on insère ces points dans la triangulation de Delaunay, des tétraèdres sont détruits et d'autres sont créés. Les tétraèdres nouvellement crées sont initialisés *matière* (on n'a pas encore testé leurs intersections avec des rayons), donc le fait d'insérer des nouveaux points élimine des tétraèdres *vides* et donc aussi élimine des tétrahèdres *extérieurs*. Le bord de l'*extérieur* peut donc perdre sa propriété de *2-variété* (Fig. D.1.f).

Pour éviter ce cas, on utilise une décroissance contrôlée. On calcule une sphère contenant l'ensemble des tétraèdres qui peuvent potentiellement être détruits par

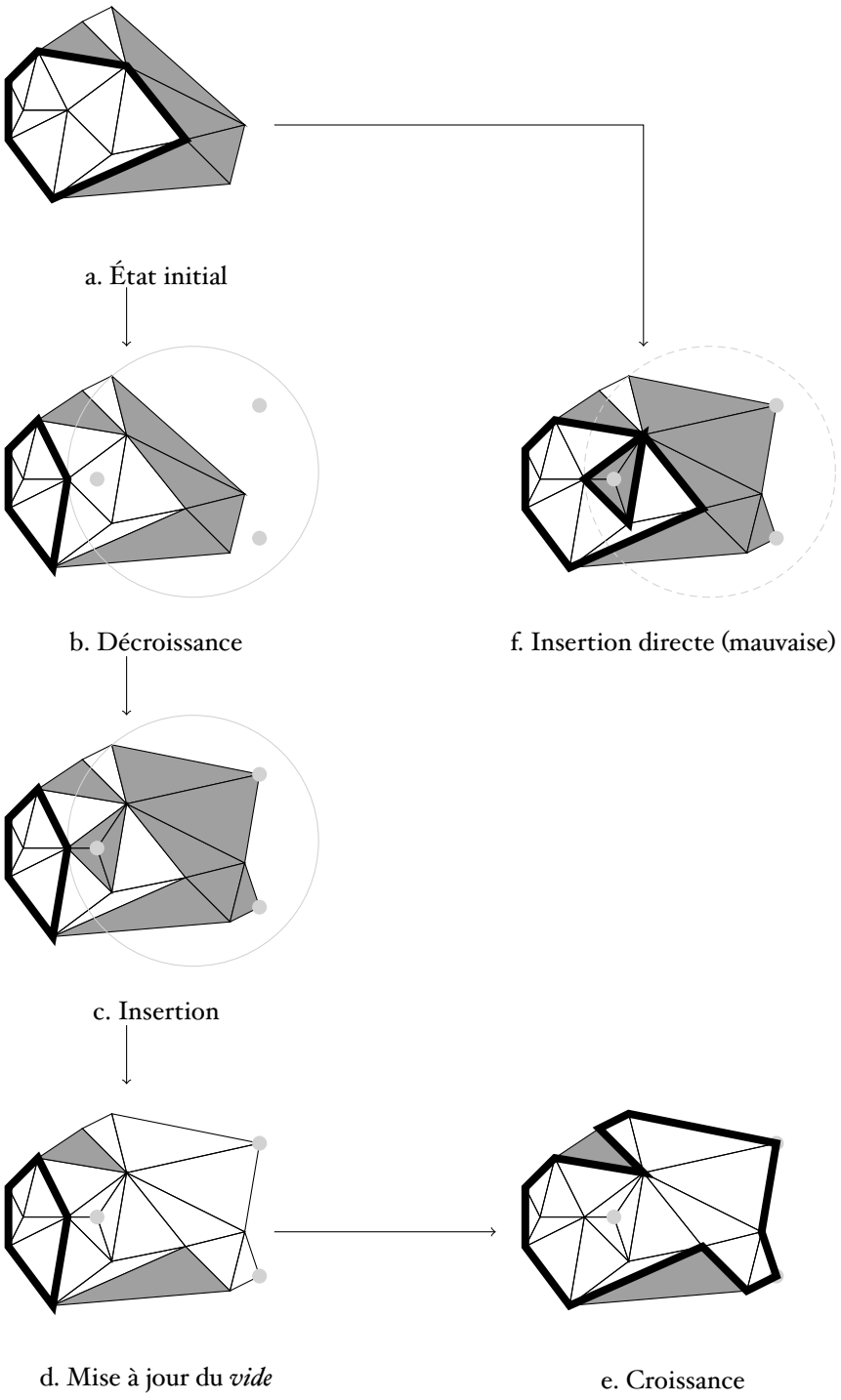


FIGURE D.1 – Vue d'ensemble de notre algorithme de reconstruction de surface à partir d'un nuage de points *épars* (ici en 2D). Les triangles blancs sont *vides*, les triangles gris sont *matière*. La ligne noire en gras est la frontière de la région *extérieur*. Les points gris clairs sont les nouveaux points 3D à insérer et le cercle gris clair est la sphère contenant les tétraèdres potentiellement modifiés.

FIGURE D.2 – Vue globale du nuage des points 3D calculée à partir du jeu des données réelles, ainsi qu'un exemple d'une image produite par la caméra utilisée.

l'insertion des nouveaux points. Ceci est possible parce que la triangulation est Delaunay. Puis, on enlève progressivement les tétraèdres contenus dans cette zone de l'ensemble des *extérieurs* en s'assurant que la frontière reste une *2-variété* (Fig. D.1.b).

Une fois que ceci est fait, on peut insérer les nouveaux points dans la triangulation (Fig. D.1.c), ceci ne modifie pas le bord de l'*extérieur*.

D.5.3 Mise à jour de la surface

Quand les nouveaux points ont été insérés, on met à jour la classification *vide/matière* des tétraèdres. Les nouveaux points 3D sont associés à de nouveaux rayons. On met à *vide* tous les tétraèdres intersectés par ces rayons. On calcule aussi les intersections entre les nouveaux tétraèdres et les anciens rayons et on met à jour cette classification en conséquence (Fig. D.1.d).

Ensuite, on met à jour la classification *extérieur/intérieur*. Pour ceci, on essaye d'ajouter un maximum des tétraèdres *vides* dans l'ensemble des tétrahèdres *extérieurs*, tout en s'assurant que le bord de l'*extérieur* reste une *2-variété*. Finalement, on applique un certain nombre de post-traitements (en particulier, la suppression d'artefacts visuels) pour raffiner ce bord. On obtient finalement la mise à jour de la surface (Fig. D.1.e).

D.6 Étude expérimentale

On a effectué un certain nombre d'expériences pour évaluer la performance de notre méthode. On a utilisé deux jeux de données : un synthétique et un réel. Le jeu de données synthétique est un ensemble d'images généré à partir d'un modèle 3D existant de milieu urbain. On peut donc comparer la surface calculée avec ce

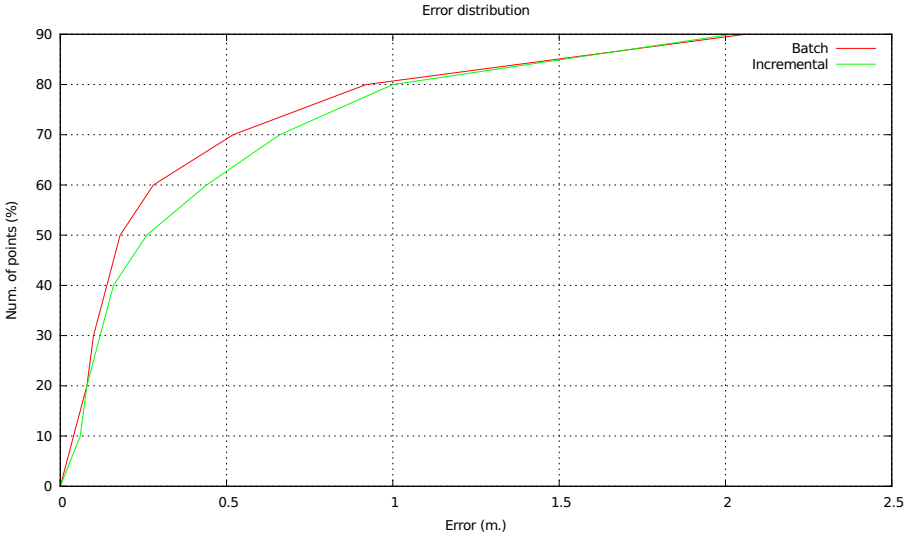


FIGURE D.3 – Comparaison de la distribution des erreurs pour la surface produite par la méthode *globale* et la notre, pour le jeu de données synthétique.

modèle et obtenir l'erreur numérique. Le jeu de données réel (Fig. D.2) permet d'évaluer qualitativement la performance de la méthode en conditions réelles.

En particulier, pour le jeu de données synthétique, on a comparé la qualité de la surface reconstruite par notre méthode avec celle reconstruite par la méthode *globale* [Lhuillier13] (qui nous a servi de base d'inspiration pour notre méthode *incrémentale*). Comme on peut le constater sur la Fig. D.3, la qualité de la nôtre est légèrement inférieur à la qualité de la méthode globale, l'écart est néanmoins faible (la longueur de la trajectoire de la caméra est de 621 m).

On a aussi appliqué notre méthode au jeu des données réelles, en faisant varier la résolution des images d'entrée (la résolution divisée par 2 ou la résolution originale) et en activant ou pas les étapes supplémentaires de *SfM* (voir Sec. D.4). On peut observer quelques vues de la surface obtenue sur la Fig. D.4.

Globalement, les résultats sont visuellement corrects, surtout quand on utilise la pleine résolution. Cependant les temps de calcul sont assez long. Pour les images d'entrée divisées par deux, il est de 20 min sur un PC standard (la longueur de la trajectoire de la caméra est 2,5 km, il y a 7700 6-uplets d'images 1024×768). Si on utilise les images de taille originale et si on active les étapes supplémentaires, le temps de calcul total devient 3 h. 10 min.

En conclusion, notre méthode produit des résultats intéressants et permet de reconstruire de très grandes scènes. Malheureusement, les temps de calculs ne permettent pas l'utilisation en-ligne de l'algorithme. Ceci limite le champ d'application de la méthode. On a étudié en détails les temps de traitement de chaque sous-étape d'une itération, et avons constaté que la majorité du temps est consommé par la suppression d'artefacts visuels. On a donc cherché à améliorer cette sous-étape.

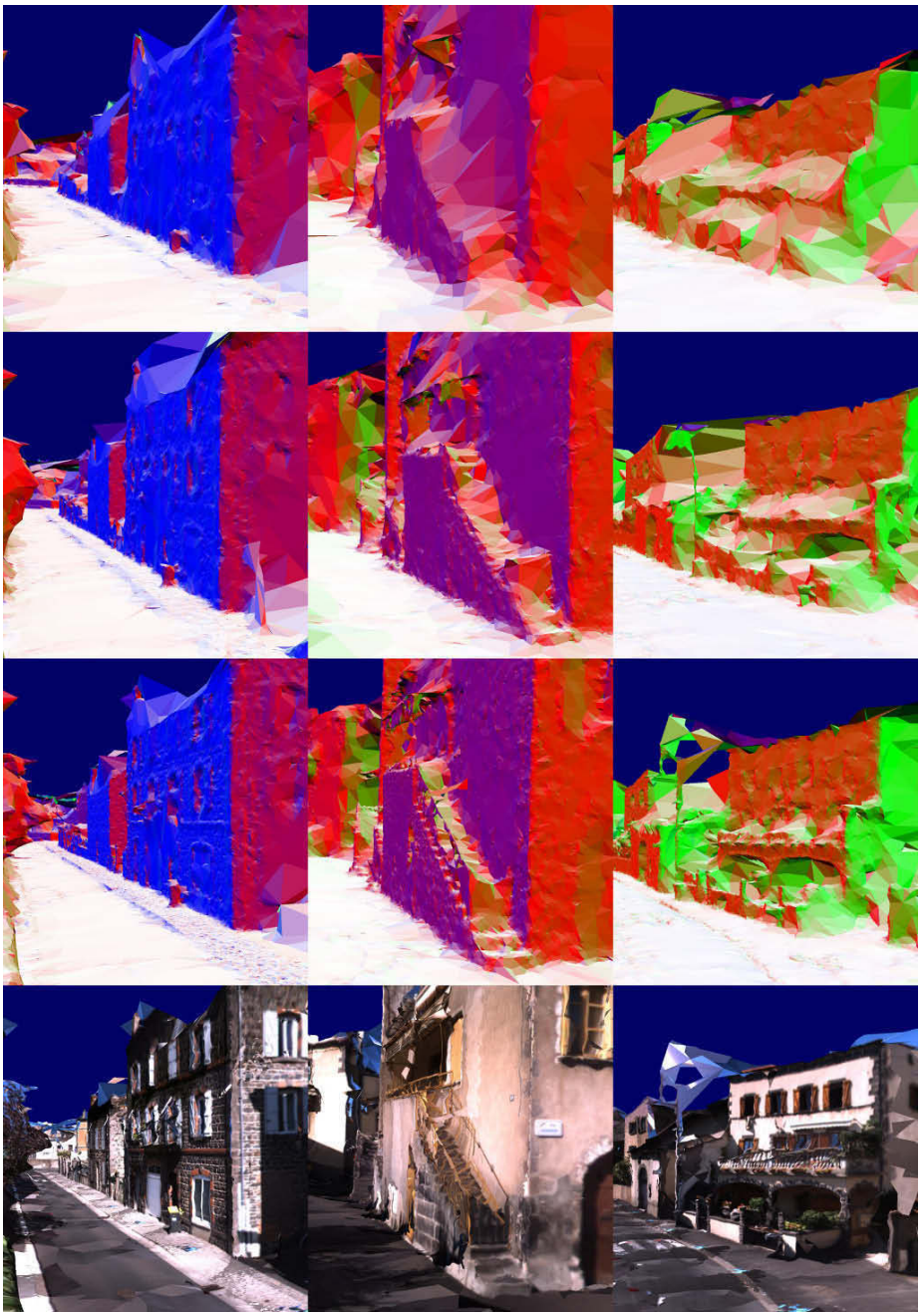


FIGURE D.4 – Quelques vues de la surface finale reconstruite avec différentes résolutions d'images en entrée. De haut en bas : images clés de dimensions divisées par 2, images clés de dimension originale, toutes les images (clés ou pas) de dimension originale, la surface avec sa texture. Les normales sont encodées avec des couleurs.

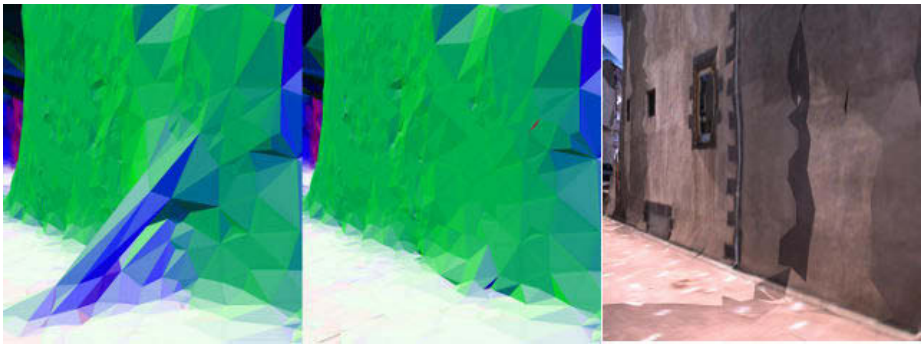


FIGURE D.5 – L'exemple d'un artefact visuel. De gauche à droite : artefact à eliminer (les normales de la surface sont encodées par de la couleur), artefact éliminé, texture (sol en bas, mur à droite).

Méthode	Temps moyen	Temps maximal	Nombre d'artefacts
Aucun	0	0	18
A	1, 19 s.	4, 74 s	11
B	0, 32 s.	1, 07 s	12
C	0, 21 s.	0, 70 s	14
B & **C**	0, 46 s.	1, 40 s	9

TABLE D.1 – Résultat de comparaison entre différentes méthodes de suppression d'artefacts visuels. **A** est la méthode précédante, **B** et **C** sont les nouvelles.

D.7 Suppression d'artefacts visuels

Un artefact visuel est un ensemble de tétraèdres *vides* qui n'ont pas pu être insérés dans l'*extérieur* par l'algorithme. Cela peut conduire à une perturbation très visible sur la surface, comme dans la Fig. D.5. On a donc besoin d'un post-traitement pour l'éliminer.

Une méthode précédante a déjà été proposée pour cela dans [Lhuillier13]. Mais malheureusement elle est très lente. On a donc mis au point deux autres méthodes et on les a comparées avec la précédante en utilisant le jeu des données réel.

On a donc exécuté notre algorithme de reconstruction incrémentale de surface avec plusieurs méthodes de suppressions d'artefacts : la précédante, chacune des deux nouvelles méthodes, ainsi que la combinaison des deux nouvelles. Puis on a compté manuellement les artefacts visuels restants. Même si cette méthode de comparaison peut paraître subjective, elle nous a paru judicieuse parce que l'on cherche à éliminer les artefacts visuels facilement remarqués par l'oeil humain. Les résultats peuvent être observés sur la table D.1.

On peut constater que les deux nouvelles méthodes prises séparément sont bien plus rapides que la précédante, mais elles sont moins performantes. Par contre, la combinaison des deux est meilleure que la méthode précédante et reste néanmoins trois fois plus rapide. On remplace donc la méthode précédante par cette combinaison.

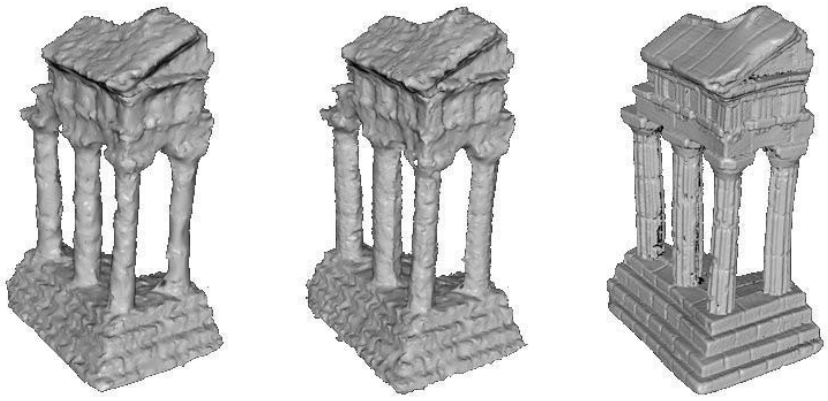


FIGURE D.6 – Surfaces obtenues à partir du jeu de données standard *Temple*. De gauche à droite : avec les points d'intérêt seulement, avec les points et les courbes, et la vérité terrain (modèle 3D de référence).

D.8 Les courbes

On a aussi voulu savoir si le fait d'utiliser les courbes dans les images, pour compléter le nuage de points épars, pouvait améliorer la qualité de la surface reconstruite. Les courbes (ou contours) forment l'ensemble de points dans les images où le gradient de luminance est maximal dans le sens du gradient.

On a donc ajouté des étapes supplémentaires dans notre algorithme. Quand une nouvelle image *clé* est détectée, le *SfM* calcule la pose de la caméra correspondante. À ce moment, on détecte les courbes dans la nouvelle image et les mettons en correspondance avec les courbes détectées dans l'image *clé* précédente. Une fois que ceci est fait, on met en correspondance les pixels entre les deux courbes correspondantes. Ceci nous permet de reconstruire l'ensemble des points 3D correspondants aux courbes. Ces points 3D sont sous-échantillonnés puis insérés dans le nuage de points 3D utilisé pour reconstruire la surface.

Pour évaluer quantitativement l'apports des courbes à la qualité de la surface, on a appliqué notre algorithme à un certain nombre d'exemples standards dans la communauté de vision par ordinateur. Par exemple, la Fig. D.6 montre les surfaces produites par l'algorithme appliqué au jeu de données *Temple* [Seitz06] (312 images de taille 640×480). La reconstruction avec les points d'intérêt prend 31 s. et la reconstruction avec les courbes dure 59 s. L'erreur maximale est de 0,59 mm. pour les points seulement et de 0,53 mm. pour les points et courbes (la boite englobante de l'objet fait de l'ordre de 10 cm de coté). L'ajout des courbes au processus de reconstruction peut donc améliorer la qualité de la surface, même si on a pu observer qu'il y a des cas assez fréquents de scènes d'extérieurs texturées pour lesquelles les courbes n'améliorent pas de façon significative la qualité de la surface.

D.9 Conclusion

Ce travail de thèse présente une méthode de reconstruction de surface qui est à la fois *éparse*, *incrémentale*, et dont le résultat possède la propriété de *2-variété*. De plus, une étape de suppression d'artefacts visuels a été améliorée, et on a évalué l'apport

des courbes à la qualité de la surface finale. Finalement, un certain nombre d'améliorations plus techniques ont été apportées par rapport aux travaux précédants ayant eu lieu à l'institut Pascal (détails dans la Sec. D.2). La vitesse d'exécution de notre méthode ne permet malheureusement pas son utilisation en-ligne. Des travaux futurs sont possibles à toutes les étapes, notamment le calcul automatique et incrémental des boucles dans le *SfM*, l'expérimentation de différent types de détecteurs de points et courbes, l'accélération du calcul de surface, l'amélioration des méthodes de mises en correspondance pour les points et les courbes.

Author's publications

[Litvinov12] Vadim Litvinov, Shuda Yu and Maxime Lhuillier. 2-Manifold reconstruction from sparse visual features. In *Proc. International Conference on 3D Imaging (IC3D)*, pp. 1--8. IEEE, december 2012. doi: 10.1109/IC3D.2012.6615134.

[Litvinov13] Vadim Litvinov and Maxime Lhuillier. Incremental solid modeling from sparse and omnidirectional Structure-from-Motion data. In *Proc. British Machine Vision Conference (BMVC)*, pp. 61.1--61.11. BMVA Press, september 2013. doi:10.5244/C.27.61.

[Litvinov14a] Vadim Litvinov and Maxime Lhuillier. Estimation incrémentale de surface à partir d'un nuage de points épars reconstruit à partir d'images omnidirectionnelles. In *Proc. Congrés National sur la Reconnaissance de Formes et l'Intelligence Artificielle (RFIA)*, june 2014.

[Litvinov14b] Vadim Litvinov and Maxime Lhuillier. Incremental solid modeling from sparse Structure-from-Motion data with improved visual artifacts removal. In *Proc. International Conference on Pattern Recognition (ICPR)*. IAPR, august 2014.

Bibliography

[Alexa03] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin and Claudio T. Silva. Computing and rendering point set surfaces. In *IEEE Transactions on Visualization and Computer Graphics*, *vol. 9*(1), pp. 3--15, 2003. doi:10.1109/TVCG.2003.1175093.

[Alliez07] Pierre Alliez, David Cohen-Steiner, Yiying Tong and Mathieu Desbrun. Voronoi-based variational reconstruction of unoriented point sets. In *Proc. Symposium on Geometry Processing (SGP)*, pp. 39--48. ACM, 2007.

[Allègre07] Rémi Allègre, Raphaëlle Chaine and Samir Akkouche. A streaming algorithm for surface reconstruction. In *Proc. Symposium on Geometry processing (SGP)*, pp. 79--88. ACM, 2007.

[Amenta99] Nina Amenta and Marshall Bern. Surface reconstruction by Voronoi filtering. In *Discrete & Computational Geometry*, *vol. 22*(4), pp. 481--504, december 1999. doi:10.1007/PL00009475.

[Amenta00a] Nina Amenta, Sunghee Choi, Tamal Krishna Dey and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proc. Symposium on Computational Geometry (SCG)*, pp. 213--222. ACM, 2000. doi:10.1145/336154.336207.

[Amenta00b] Nina Amenta and Ravi Krishna Kolluri. Accurate and efficient unions of balls. In *Proc. Symposium on Computational Geometry (SCG)*, pp. 119--128. ACM, 2000. doi:10.1145/336154.336193.

[Amenta01] Nina Amenta, Sunghee Choi and Ravi Krishna Kolluri. The power crust. In *Proc. Symposium on Solid Modeling and Applications (SMA)*, pp. 249--266. ACM, 2001. doi:10.1145/376957.376986.

[Arnold80] R. Douglas Arnold and Thomas O. Binford. Geometric constraints in stereo vision. In *Proc. Image Processing for Missile Guidance*, vol. 0238, p. 281. SPIE, december 1980. doi:10.1117/12.959157.

[Ayache87] Nicholas Ayache and Francis Lustman. Fast and reliable passive trinocular stereovision. In *Proc. International Conference on Computer Vision (ICCV)*, june 1987.

[Bajaj95] Chandrajit L. Bajaj, Fausto Bernardini and Guoliang Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *Proc. Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pp. 109--118. ACM, 1995. doi:10.1145/218380.218424.

[Bay06] Herbert Bay, Tinne Tuytelaars and Luc Van Gool. SURF: Speeded up robust features. In *Proc. European Conference on Computer Vision (ECCV)*, pp. 404--417. IEEE, may 2006. doi:10.1007/11744023_32.

[Bern94] Marshall Bern, David Eppstein and John Gilbert. Provably good mesh generation. In *Journal of Computer and System Sciences*, *vol. 48*, pp. 384--409, june 1994. doi:10.1016/S0022-0000(05)80059-5.

[Bernardini99] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. In *IEEE Transactions on Visualization and Computer Graphics*, *vol. 5*(4), pp. 349--359, october--december 1999. doi:10.1109/2945.817351.

[Besl92] Paul J. Besl and Neil D. McKay. A method for registration of 3-D shapes. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 14*(2), pp. 239--256, february 1992. doi:10.1109/34.121791.

[Bobick99] Aaron F. Bobick and Stephen S. Intille. Large occlusion stereo. In *International Journal of Computer Vision (IJCV)*, *vol. 33*(3), pp. 181--200, september 1999. doi:10.1023/A:1008150329890.

[Boissonnat84] Jean-Daniel Boissonnat. Geometric structures for three-dimensional shape representation. In *ACM Transactions on Graphics (TOG)*, *vol. 3*(4), pp. 266--286, 1984. doi:10.1145/357346.357349.

[Boissonnat98] Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic geometry*, vol. 5. Cambridge University Press, 1998.

[Boissonnat00a] Jean-Daniel Boissonnat and Frédéric Cazals. Smooth surface reconstruction via natural neighbour interpolation of distance functions. In *Proc. Symposium on Computational Geometry (SCG)*. ACM, 2000. doi:10.1145/336154.336208.

[Boissonnat00b] Jean-Daniel Boissonnat, Olivier Devillers, Monique Teillaud and Mariette Yvinec. Triangulations in CGAL. In *Proc. Symposium on Computational Geometry (SCG)*, pp. 11--18. ACM, 2000. doi:10.1145/336154.336165.

[Boissonnat05] Jean-Daniel Boissonnat and Steve Oudot. Provably good sampling and meshing of surfaces. In *Graphical Models*, *vol. 67*(5), pp. 405--451, september 2005. doi:10.1016/j.gmod.2005.01.004.

[Boissonnat09] Jean-Daniel Boissonnat, Olivier Devillers and Samuel Hornus. Incremental construction of the Delaunay triangulation and the Delaunay graph in medium dimension. In *Proc. Annual Symposium on Computational Geometry (SCG)*, pp. 208--216. ACM, 2009. doi:10.1145/1542362.1542403.

[Botsch10] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez and Bruno Lévy. *Polygon mesh processing*. AK Peters, 2010.

[Bottino04] Andrea Bottino and Aldo Laurentini. The visual hull of smooth curved objects. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 26*(12), pp. 1622--1632, december 2004. doi:10.1109/TPAMI.2004.130.

[Brint90] Andrew T. Brint and Michael Brady. Stereo matching of curves. In *Image and Vision Computing*, *vol. 8*(1), pp. 50--56, february 1990. doi:10.1016/0262-8856(90)90056-B.

[Broadhurst01] Adrian Broadhurst, Tom W. Drummond and Roberto Cipolla. A probabilistic framework for space carving. In *Proc. International Conference on Computer Vision (ICCV)*, vol. 1, pp. 388--393. IEEE, july 2001. doi:10.1109/ICCV.2001.937544.

[Brown03] Myron Z. Brown, Darius Burschka and Gregory D. Hager. Advances in computational stereo. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 25*(8), pp. 993--1008, august 2003. doi:10.1109/TPAMI.2003.1217603.

[Canny86] John Canny. A computational approach to edge detection. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 8*(6), pp. 679--698, november 1986. doi:10.1109/TPAMI.1986.4767851.

[Carr01] Jonathan C. Carr, Rick K. Beatson, J. B. Cherrie, T. J. Mitchell, W. Richard Fright, Bruce C. McCallum and T. R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *Proc. Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pp. 67--76. ACM, 2001. doi:10.1145/383259.383266.

[Cazals04] Frédéric Cazals and Joachim Giesen. Delaunay triangulation based surface reconstruction: Ideas and algorithms. Tech. Rep. 5393, Institut National de Recherche en Informatique et en Automatique (INRIA), november 2004.

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

[Chaine03] Raphaëlle Chaine. A geometric convection approach of 3-D reconstruction. In *Proc. Symposium on Geometry Processing (SGP)*, pp. 218--229. ACM, 2003.

[Chen92] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. In *Image and Vision Computing (IVC)*, *vol. 10*(3), pp. 145--155, april 1992. doi:10.1016/0262-8856(92)90066-C.

[Cipolla92] Roberto Cipolla and Andrew Blake. Surface shape from the deformation of apparent contours. In *International Journal of Computer Vision (IJCV)*, *vol. 9*(2), pp. 83--112, november 1992. doi:10.1007/BF00129682.

[Collins96] Robert T. Collins. A space-sweep approach to true multi-image matching. In *Proc. Computer Vision and Pattern Recognition*, pp. 358--363. IEEE, june 1996. doi:10.1109/CVPR.1996.517097.

[Curless96] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proc. Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pp. 303--312. ACM, 1996. doi:10.1145/237170.237269.

[Dey04] Tamal K. Dey and Samrat Goswami. Provable surface reconstruction from noisy samples. In *Proc. Symposium on Computational Geometry (SCG)*, pp. 330--339. ACM, 2004. doi:10.1145/997817.997867.

[Dey07] Tamal K. Dey. *Curve and surface reconstruction: algorithms with mathematical analysis*. Cambridge University Press, february 2007.

[Dhond89] Umesh R. Dhond and J. K. Aggarwal. Structure from stereo - a review. In *IEEE Transactions on Systems, Man and Cybernetics*, *vol. 19*(6), pp. 1489--1510, november/december 1989. doi:10.1109/21.44067.

[Edelsbrunner94] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. In *ACM Transactions on Graphics (TOG)*, *vol. 13*(1), pp. 43--72, january 1994. doi:10.1145/174462.156635.

[Esteban04] Carlos Hernández Esteban and Francis Schmitt. Silhouette and stereo fusion for 3D object modeling. In *Computer Vision and Image Understanding (CVIU)*, *vol. 96*(3), pp. 367--392, december 2004. doi:10.1016/j.cviu.2004.03.016.

[Fabbri10] Ricardo Fabbri and Benjamin Kimia. 3D curve sketch: Flexible curve-based stereo reconstruction and calibration. In *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1538--1545. IEEE, june 2010. doi:10.1109/CVPR.2010.5539787.

[Faugeras90] Olivier Faugeras, Elizabeth Le Bras-Mehlman and Jean-Daniel Boissonnat. Representing stereo data with the Delaunay triangulation. In *Artificial Intelligence*, *vol. 44*(1--2), pp. 41--87, july 1990. doi:10.1016/0004-3702(90)90098-K.

[Faugeras93] Olivier Faugeras. *Three-dimensional computer vision: A geometric viewpoint*. The MIT Press, 1993.

[Faugeras98] Olivier Faugeras and Renaud Keriven. Variational principles, surface evolution, PDE's, level set methods and the stereo problem. In *IEEE Transactions on Image Processing*, *vol. 7*(3), pp. 336--344, mars 1998. doi:10.1109/83.661183.

[Fischler81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Communications of the ACM*, *vol. 24*(6), pp. 381--395, june 1981. doi:10.1145/358669.358692.

[Floriani98] Leila De Floriani, Paola Magillo and Enrico Puppo. Managing the level of detail in 3D shape reconstruction and representation. In *Proc. International Conference on Pattern Recognition (ICPR)*, vol. 1, p. 389. IEEE, 1998. doi:10.1109/ICPR.1998.711162.

[Franco09] Jean-Sébastien Franco and Edmond Boyer. Efficient polyhedral modeling from silhouettes. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 31*(3), pp. 414--427, march 2009. doi:10.1109/TPAMI.2008.104.

[Fua95] Pascal Fua and Yvan G. Leclerc. Object-centered surface reconstruction: Combining multi-image stereo and shading. In *International Journal of Computer Vision (IJCV)*, *vol. 16*(1), pp. 35--56, september 1995. doi:10.1007/BF01428192.

[Fuhrmann11] Simon Fuhrmann and Michael Goesele. Fusion of depth maps with multiple scales. In *ACM Transactions on Graphics (TOG)*, *vol. 30*(6), p. 148, december 2011. doi:10.1145/2070781.2024182.

[Gezahegne05] Abel Gezahegne. *Surface reconstruction with constrained sculpting*. Master's thesis, University of California, 2005.

[Giblin10] P. Giblin. *Graphs, surfaces and homology*. Cambridge University Press, 2010.

[Goodman04] Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. Chapman & Hall/CRC, 2 edn., 2004.

[Gopi00] M. Gopi, S. Krishnan and C. T. Silva. Surface reconstruction based on lower dimensional localized Delaunay triangulation. In *Computer Graphics Forum*, *vol. 19*(3), pp. 467--478, september 2000. doi:10.1111/1467-8659.00439.

[Haralick94] Robert M. Haralick, Chung-nan Lee, Karsten Ottenberg and Michael Nölle. Review and analysis of solutions of the three point perspective pose estimation problem. In *International Journal of Computer Vision (IJCV)*, *vol. 13*(3), pp. 331--356, december 1994. doi:10.1007/BF02028352.

[Harris88] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Proc. Alvey Vision Conference (AVC)*, pp. 23.1--23.6. BMVA Press, 1988. doi:10.5244/C.2.23.

[Hartley04] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, 2 edn., april 2004.

[Hiep09] Vu Hoang Hiep, Renaud Keriven, Patrick Labatut and Jean-Philippe Pons. Towards high-resolution large-scale multi-view stereo. In *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1430--1437. IEEE, june 2009. doi:10.1109/CVPR.2009.5206617.

[Hilton96] Adrian Hilton, A. J. Stoddart, John Illingworth and Terry Windeatt. Reliable surface reconstruction from multiple range images. In *Proc. European Conference on Computer Vision (ECCV)*, vol. I, pp. 117--126. Springer, april 1996. doi:10.1007/BFb0015528.

[Hilton97] Adrian Hilton and John Illingworth. Multi-resolution geometric fusion. In *Proc. International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, pp. 181--188. IEEE, may 1997. doi:10.1109/IM.1997.603864.

[Hilton05] Adrian Hilton. Scene modelling from sparse 3D data. In *Image and Vision Computing (IVC)*, *vol. 23*(10), pp. 900--920, september 2005. doi:10.1016/j.imavis.2005.05.018.

[Hofer13] Manuel Hofer, Andreas Wendel and Horst Bischof. Incremental line-based 3D reconstruction using geometric constraints. In *Proc. British Machine Vision Conference (BMVC)*, pp. 92.1--92.11. BMVA Press, 2013. doi:10.5244/C.27.92.

[Hoppe92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald and Werner Stuetzle. Surface reconstruction from unorganized points. In *Proc. Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, vol. 26. ACM, july 1992. doi:10.1145/142920.134011.

[Hoppe13] Christof Hoppe, Manfred Klopschitz, Michael Donoser and Horst Bischof. Incremental surface extraction from sparse Structure-from-Motion point clouds. In *Proc. British Machine Vision Conference (BMVC)*, pp. 94.1--94.11. BMVA Press, 2013. doi:10.5244/C.27.94.

[Hornung06] Alexander Hornung and Leif Kobbelt. Hierarchical volumetric multi-view stereo reconstruction of manifold surfaces based on dual graph embedding. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. 503--510. IEEE, june 2006. doi:10.1109/CVPR.2006.135.

[Isidoro03] John Isidoro and Stan Sclaroff. Stochastic refinement of the visual hull to satisfy photometric and silhouette consistency constraints. In *Proc. International Conference on Computer Vision (ICCV)*, vol. 2, pp. 1335--1342. IEEE, october 2003. doi:10.1109/ICCV.2003.1238645.

[Jancosek11] Michal Jancosek and Tomas Pajdla. Multi-view reconstruction preserving weakly-supported surfaces. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, pp. 3121--3128. IEEE, june 2011. doi:10.1109/CVPR.2011.5995693.

[Jin05] Hailin Jin, Stefano Soatto and Anthony J. Yezzi. Multi-view stereo reconstruction of dense shape and complex appearance. In *International Journal of Computer Vision (IJCV)*, *vol. 63*(3), pp. 175--189, july 2005. doi:10.1007/s11263-005-6876-7.

[Kahl03] Fredrik Kahl and Jonas August. Multiview reconstruction of space curves. In *Proc. International Conference on Computer Vision (ICCV)*, vol. 2, pp. 1017--1024. IEEE, october 2003. doi:10.1109/ICCV.2003.1238461.

[Kanade94] Takeo Kanade and Masatoshi Okutomi. A stereo matching algorithm with an adaptive window: theory and experiment. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 16*(9), pp. 920--932, september 1994. doi:10.1109/34.310690.

[Kazhdan05] Michael Kazhdan. Reconstruction of solid models from oriented point sets. In *Proc. Symposium on Geometry Processing (SGP)*, p. 73. ACM, 2005.

[Kazhdan06] Michael Kazhdan, Matthew Bolitho and Hugues Hoppe. Poisson surface reconstruction. In *Proc. Symposium on Geometry Processing (SGP)*, pp. 61--70. ACM, 2006.

[Klein07] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 225--234. IEEE, november 2007. doi:10.1109/ISMAR.2007.4538852.

[Koenderink84] Jan J. Koenderink. What does the occluding contour tell us about solid shape? In *Perception*, *vol. 13*(3), pp. 321--330, 1984.

[Kohli07] Pushmeet Kohli and Philip H. S. Torr. Dynamic graph cuts for efficient inference in Markov random fields. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 29*(12), pp. 2079--2088, december 2007. doi:10.1109/TPAMI.2007.1128.

[Kolluri04] Ravi Krishna Kolluri, Jonathan Richard Shewchuk and James F. O'Brien. Spectral surface reconstruction from noisy point clouds. In *Proc. Symposium on Geometry Processing (SGP)*, pp. 11--21. ACM, 2004. doi:10.1145/1057432.1057434.

[Kolluri08] Ravi Krishna Kolluri. Provably good moving least squares. In *ACM Transactions on Algorithms (TALG)*, *vol. 4*(2), p. 18, may 2008. doi:10.1145/1361192.1361195.

[Kutulakos00] Kiriakos N. Kutulakos and Steven M. Seitz. A theory of shape by space carving. In *International Journal of Computer Vision (IJCV)*, *vol. 38*(3), pp. 199--218, july 2000. doi:10.1023/A:1008191222954.

[Labatut07] Patrick Labatut, Jean-Philippe Pons and Renaud Keriven. Efficient Multi-View Reconstruction of Large-Scale Scenes using Interest Points, Delaunay Triangulation and Graph Cuts. In *Proc. International Conference on Computer Vision (ICCV)*, pp. 1--8. IEEE, october 2007. doi:10.1109/ICCV.2007.4408892.

[Labatut09] Patrick Labatut, Jean-Philippe Pons and Renaud Keriven. Robust and efficient surface reconstruction from range data. In *Computer Graphics Forum*, *vol. 28*(8), pp. 2275--2290, december 2009. doi:10.1111/j.1467-8659.2009.01530.x.

[Lad] Ladybug 2 360° Video Camera. http:// www.ptgrey.com/ products/ ladybug2/ ladybug2_360_video_camera.asp.

[Laurentini94] Aldo Laurentini. The visual hull concept for silhouette-based image understanding. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 16*(2), pp. 150--162, february 1994. doi:10.1109/34.273735.

[Lavest98] Jean-Marc Lavest, Marc Viala and Michel Dhome. Do we really need an accurate calibration pattern to achieve a reliable camera calibration? In *Proc. European Conference on Computer Vision (ECCV)*, vol. I, pp. 158--174. Springer, june 1998. doi:10.1007/BFb0055665.

[Levin04] David Levin. Mesh-independent surface interpolation. In *Geometric modeling for scientific visualization*, pp. 37--49. Springer, 2004. doi:10.1007/978-3-662-07443-5_3.

[Lhuillier02] Maxime Lhuillier and Long Quan. Match propagation for image-based modeling and rendering. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 24*(8), pp. 1140--1146, august 2002. doi:10.1109/TPAMI.2002.1023810.

[Lhuillier03] Maxime Lhuillier and Long Quan. Surface reconstruction by integrating 3D and 2D data of multiple views. In *Proc. International Conference on Computer Vision (ICCV)*, vol. 2, pp. 1313--1320. IEEE, october 2003. doi:10.1109/ICCV.2003.1238642.

[Lhuillier08] Maxime Lhuillier. Automatic scene structure and camera motion using a catadioptric system. In *Computer Vision and Image Understanding (CVIU)*, *vol. 109*(2), pp. 186--203, february 2008. doi:10.1016/j.cviu.2007.05.004.

[Lhuillier13] Maxime Lhuillier and Shuda Yu. Manifold surface reconstruction of an environment from sparse Structure-from-Motion data. In *Computer Vision and Image Understanding (CVIU)*, *vol. 117*(11), pp. 1628--1644, november 2013. doi:10.1016/j.cviu.2013.08.002.

[Liu06] Jun Liu and Roger Hubbold. Mesh optimisation using edge information in feature-based surface reconstruction. In *Proc. International Symposium on Visual Computing (ISVC)*, pp. 434--444. Springer, november 2006. doi:10.1007/11919476_44.

[Lorensen87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proc. Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, vol. 21, pp. 163--169. ACM, july 1987. doi:10.1145/37402.37422.

[Lovi10] David Lovi, Neil Birkbeck, Dana Cobzaş and Martin Jägersand. Incremental free-space carving for real-time 3D reconstruction. In *Proc. 3D Data Processing, Visualization and Transmission (3DPVT)*, 2010.

[Lowe04] David D. Lowe. Distinctive image features from scale-invariant keypoints. In *International Journal of Computer Vision (IJCV)*, *vol. 60*(2), pp. 91--110, november 2004. doi:10.1023/B:VISI.0000029664.99615.94.

[Manessis00] Anastasios Manessis, Adrian Hilton, Phil Palmer, Phil McLauchlan and Xinquan Shen. Reconstruction of scene models from sparse 3D structure. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 666--671. IEEE, june 2000. doi:10.1109/CVPR.2000.854938.

[Martinsson07] Hanna Martinsson, François Gaspard, Adrien Bartoli and Jean-Marc Lavest. Reconstruction of 3D curves for quality control. In *Proc. Scandinavian Conference on Image Analysis (SCIA)*, pp. 760--769. Springer, june 2007. doi:10.1007/978-3-540-73040-8_77.

[Meyer03] Mark Meyer, Mathieu Desbrun, Peter Schröder and Alan H. Barr. *Visualization and Mathematics III*, chap. Discrete differential-geometry operators for triangulated 2-manifolds, pp. 35--57. Springer Berlin Heidelberg, 2003. doi:10.1007/978-3-662-05105-4_2.

[Mid] Middlebury multi-view stereo evaluation. http://vision.middlebury.edu/mview/eval/.

[Mičušík09] Branislav Mičušík and Jana Košecká. Piecewise planar city 3D modeling from street view panoramic sequences. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, pp. 2906--2912. IEEE, june 2009. doi:10.1109/CVPR.2009.5206535.

[Moise97] E.E. Moise. *Geometric topology in dimensions 2 and 3*. Springer Verlag, 1997.

[Morris00] Daniel D. Morris and Takeo Kanade. Image-consistent surface triangulation. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. 332--338. IEEE, june 2000. doi:10.1109/CVPR.2000.855837.

[Morse05] Bryan S. Morse, Terry S. Yoo, Penny Rheingans, David T. Chen and K. R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Proc. Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2005. doi:10.1145/1198555.1198645.

[Mouragnon09] Etienne Mouragnon, Maxime Lhuillier, Michel Dhome, Fabien Dekeyser and P. Sayd. Generic and real-time structure from motion using local bundle adjustment. In *Image and Vision Computing (IVC)*, *vol. 27*(8), pp. 1178--1193, july 2009. doi:10.1016/j.imavis.2008.11.006.

[Nasrabadi92] Nasser M. Nasrabadi. A stereo vision technique using curve-segments and relaxation matching. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 14*(5), pp. 566--572, may 1992. doi:10.1109/34.134060.

[Newcombe10] Richard A. Newcombe and Andrew J. Davison. Live dense reconstruction with a single moving camera. In *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1063--6919. IEEE, june 2010. doi:10.1109/CVPR.2010.5539794.

[Nistér04] David Nistér. An efficient solution to the five-point relative pose problem. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 26*(6), pp. 756--770, june 2004. doi:10.1109/TPAMI.2004.17.

[Nistér06] David Nistér, Oleg Naroditsky and James Bergen. Visual odometry for ground vehicle applications. In *Journal of Field Robotics*, *vol. 23*(1), pp. 3--20, january 2006. doi:10.1002/rob.20103.

[Ohrhallinger13] Stefan Ohrhallinger, Sudhir Mudur and Michael Wimmer. Minimizing edge length to connect sparsely sampled unstructured point sets. In *Computer & Graphics*, *vol. 37*(6), pp. 645--658, october 2013. doi:10.1016/j.cag.2013.05.016.

[Ohtake03] Yutaka Ohtake, Alexander Belyaev and Hans-Peter Seidel. A multi-scale approach to 3D scattered data interpolation with compactly supported basis functions. In *Proc. Shape Modeling International (SMI)*, pp. 153--161. IEEE, may 2003. doi:10.1109/SMI.2003.1199611.

[Ohtake05] Yutaka Ohtake, Alexander Belyaev, Greg Turk and Hans-Peter Seidel. Multi-level partition of unity implicits. In *Proc. Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 173. ACM, 2005. doi:10.1145/1198555.1198649.

[Ope] The OpenMP API specification for parallel programming. http://www.openmp.org.

[Oztireli09] A. Cengiz Oztireli, Gael Guennebaud and M. Gross. Feature preserving point set surfaces based on non-linear kernel regression. In *Computer Graphics Forum*, *vol. 28*(2), pp. 493--501, april 2009. doi:10.1111/j.1467-8659.2009.01388.x.

[Pan09] Qi Pan, Gerhard Reitmayr and Tom Drummond. ProFORMA: Probabilistic feature-based on-line rapid model acquisition. In *Proc. British Machine Vision Conference (BMVC)*, pp. 112.1--112.11. BMVA Press, 2009. doi:10.5244/C.23.112.

[Pan11] Qi Pan, Clemens Arth, Gerhard Reitmayr and Edward Rosten. Rapid scene reconstruction on mobile phones from panoramic images. In *Proc. International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 55--64. IEEE, october 2011. doi:10.1109/ISMAR.2011.6092370.

[Pito96] Richard Pito. Mesh integration based on co-measurements. In *Proc. International Conference on Image Processing (ICIP)*, vol. 2, pp. 397--400. IEEE, september 1996. doi:10.1109/ICIP.1996.560846.

[Pollefeys08] Marc Pollefeys, David Nistér, Jan-Michael Frahm, Amir Akbarzadeh, Philippos Mordohai, Brian Clipp, Cris Engels, David Gallup, S.-J. Kim, Paul Merrell, C. Salmi, S. Sinha, B. Talton, Liang Wang, Qiang Yang, Henrik Stewénius, Ruigang Yang, Greg Welch and Herman Towles. Detailed real-time urban 3D reconstruction from video. In *International Journal of Computer Vision (IJCV)*, *vol. 78*(2--3), pp. 143--167, july 2008. doi:10.1007/s11263-007-0086-4.

[Pons07] Jean-Philippe Pons, Renaud Keriven and Olivier Faugeras. Multi-view stereo reconstruction and scene flow estimation with a global image-based matching score. In *International Journal of Computer Vision (IJCV)*, *vol. 72*(2), pp. 179--193, april 2007. doi:10.1007/s11263-006-8671-5.

[Robert91] Luc Robert and Olivier Faugeras. Curve-based stereo: figural continuity and curvature. In *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 57--62. IEEE, june 1991. doi:10.1109/CVPR.1991.139661.

[Rosten06] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Lecture Notes in Computer Science*, *vol. 3951*, pp. 430--443, 2006. doi:10.1007/11744023_34.

[Roy98] Sébastien Roy and Ingemar J. Cox. A maximum-flow formulation of the N-camera stereo correspondence problem. In *Proc. International Conference on Computer Vision (ICCV)*, pp. 492--499. IEEE, january 1998. doi:10.1109/ICCV.1998.710763.

[Salman09] Nader Salman and Mariette Yvinec. Surface reconstruction from multi-view stereo. In *Lecture Notes in Computer Science*, 2009.

[Savchenko95] Vladimir V. Savchenko, Alexander A. Pasko, Oleg G. Okunev and Tosiyasu L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. In *Computer Graphics Forum*, *vol. 14*(4), pp. 181--188, october 1995. doi:10.1111/1467-8659.1440181.

[Schmeing11] Benno Schmeing, Tomas Läbe and Wolfgang Förstner. Trajectory reconstruction using long sequences of digital images from an omnidirectional camera. In *Proc. Deutsche Gesellschaft für Photogrammetrie (DGPF) Conference*, 2011.

[Schmid00] Cordelia Schmid and Andrew Zisserman. The geometry and matching of lines and curves over multiple views. In *International Journal of Computer Vision (IJCV)*, *vol. 40*(3), pp. 199--233, december 2000. doi:10.1023/A:1008135310502.

[Seitz99] Steven M. Seitz and Charles R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *International Journal of Computer Vision (IJCV)*, *vol. 35*(2), pp. 151--173, november 1999. doi:10.1023/A:1008176507526.

[Seitz06] Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. 519--528. IEEE, june 2006. doi:10.1109/CVPR.2006.19.

[Shewchuk04] Jonathan Richard Shewchuk. Stabbing Delaunay tetrahedralizations. In *Discrete & Computational Geometry*, *vol. 32*, pp. 339--343, september 2004. doi:10.1007/s00454-004-1095-5.

[Slabaugh04] Gregory G. Slabaugh, W. Bruce Culbertson, Thomas Malzbender, Mark R. Stevens and Ronald W. Schafer. Methods for volumetric reconstruction of visual scenes. In *International Journal of Computer Vision (IJCV)*, *vol. 57*(3), pp. 179--199, may 2004. doi:10.1023/B:VISI.0000013093.45070.3b.

[Soucy95] Marc Soucy and Denis Laurendeau. A general surface approach to the integration of a set of range views. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 17*(4), pp. 344--358, april 1995. doi:10.1109/34.385982.

[Strecha08] Christoph Strecha, Wolfgang von Hansen, Luc Van Gool, Pascal Fua and Ulrich Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1--8. IEEE, june 2008. doi:10.1109/CVPR.2008.4587706.

[Sugiura13] Takayuki Sugiura, Akihiko Torii and Masatoshi Okutomi. 3D surface extraction using incremental tetrahedra carving. In *Proc. ICCV 2013 Workshop: Big Data in 3D Computer Vision*, pp. 692--699, 2013.

[Taubin95] Gabriel Taubin. A signal processing approach to fair surface design. In *Proc. Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pp. 351--358. ACM, 1995. doi:10.1145/218380.218473.

[Taylor03] Camillo J. Taylor. Surface reconstruction from feature based stereo. In *Proc. International Conference on Computer Vision (ICCV)*, vol. 1, pp. 184--190. IEEE, october 2003. doi:10.1109/ICCV.2003.1238338.

[Teney12] Damien Teney and Justus Piater. Sampling-based multiview reconstruction without correspondences for 3D edges. In *Proc. International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*, pp. 160--167. IEEE, october 2012. doi:10.1109/3DIMPVT.2012.28.

[Tobor04] Ireneusz Tobor, Patrick Reuter and Christopher Schlick. Efficient reconstruction of large scattered geometric datasets using the partition of unity and radial basis functions. In *Journal of WSCG, vol. 12*(1--3), pp. 467--474, 2004.

[Tran06] Son Tran and Larry Davis. 3D surface reconstruction using graph cuts with surface constraints. In *Proc. European Conference on Computer Vision (ECCV)*, vol. 3952, pp. 219--231. Springer, may 2006. doi:10.1007/11744047_17.

[Treuille04] Adrien Treuille, Aaron Hertzmann and Steven M. Seitz. Example-based stereo with general BRDFs. In *Proc. European Conference on Computer Vision (ECCV)*, pp. 457--469. Springer, may 2004. doi:10.1007/978-3-540-24671-8_36.

[Turk94] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proc. Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pp. 311--318. ACM, 1994. doi:10.1145/192161.192241.

[Turk02] Greg Turk and James F. O'Brien. Modelling with implicit surfaces that interpolate. In *ACM Transactions on Graphics (TOG), vol. 21*(4), pp. 855--873, 2002. doi:10.1145/571647.571650.

[Vaillant92] Régis Vaillant and Olivier D. Faugeras. Using extremal boundaries for 3-D object modeling. In *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14*(2), pp. 157--173, february 1992. doi:10.1109/34.121787.

[Veltkamp95] Remco C. Veltkamp. Boundaries through scattered points of unknown density. In *Graphical Models and Image Processing, vol. 57*(6), pp. 441--452, november 1995. doi:10.1006/gmip.1995.1038.

[Vogiatzis05] George Vogiatzis, Philip H.S. Torr and Roberto Cipolla. Multi-view stereo via volumetric graph-cuts. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 391--398. IEEE, june 2005. doi:10.1109/CVPR.2005.238.

[Vogiatzis07] George Vogiatzis, Carlos Hernández Esteban, Philip H. S. Torr and Roberto Cipolla. Multiview stereo via volumetric graph-cuts and occlusion robust photo-consistency. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 29*(12), pp. 2241--2246, december 2007. doi:10.1109/TPAMI.2007.70712.

[Wood04] Zoë Wood, Hugues Hoppe, Mathieu Desbrun and Peter Schröder. Removing excess topology from isosurfaces. In *ACM Transactions on Graphics (TOG)*, *vol. 23*(2), pp. 190--208, april 2004. doi:10.1145/990002.990007.

[Wu05] Hong Wu and Yizhou Yu. Photogrammetric reconstruction of free-form objects with curvilinear structures. In *The Visual Computer*, *vol. 21*(4), pp. 203--216, may 2005. doi:10.1007/s00371-005-0281-7.

[Yang03] Ruigang Yang and Marc Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *Proc. Computer Vision and Pattern Recognition*, vol. 1, pp. I--211--I--217. IEEE, june 2003. doi:10.1109/CVPR.2003.1211356.

[Yu11] Shuda Yu and Maxime Lhuillier. Surface reconstruction of scenes using a catadioptric camera. In *Lecture Notes in Computer Science*, *vol. 6930*, pp. 145--156, 2011. doi:10.1007/978-3-642-24136-9_13.

[Yu12] Shuda Yu and Maxime Lhuillier. Incremental reconstruction of manifold surface from sparse visual mapping. In *Proc. International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*, pp. 293--300. IEEE, october 2012. doi:10.1109/3DIMPVT.2012.11.

[Yu13] Shuda Yu. *Automatic 3D modeling of environments: a sparse approach from images taken by a catadioptric camera*. Ph.D. thesis, École Doctorale Sciences pour l'Ingénieur de Clermont-Ferrand, june 2013.

[Zach07] Christopher Zach, Thomas Pock and Horst Bischof. A globally optimal algorithm for robust TV-L1 range image integration. In *Proc. International Conference on Computer Vision (ICCV)*, pp. 1--8. IEEE, october 2007. doi:10.1109/ICCV.2007.4408983.

[Zheng94] Jiang Yu Zheng. Acquiring 3-D models from sequences of contours. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *vol. 16*(2), pp. 163--178, february 1994. doi:10.1109/34.273734.

[Zhou07] Qian-Yi Zhou, Tao Ju and Shi-Min Hu. Topology repair of solid models using skeletons. In *IEEE Transactions on Visualization and Computer Graphics*, *vol. 13*(4), pp. 675--685, july 2007. doi:10.1109/TVCG.2007.1015.