3D topography by image segmentation approach : application to scanning electron microscopy

Sébastien Drouyer

THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres
PSL Research University

Préparée à MINES ParisTech

3D topography by image segmentation approach: application to scanning electron microscopy

Topographie 3D par approche segmentation : application au microscope électronique à balayage

École doctorale n°432

SCIENCES ET MÉTIERS DE L'INGÉNIEUR

Spécialité MORPHOLOGIE MATHÉMATIQUE

COMPOSITION DU JURY :

M. Joachim OHSER
Hochschule Darmstadt, Rapporteur

M. Jean-Marc CHAIX
Grenoble INP - Phelma, Rapporteur

M. Thierry FOURNEL
Université de Saint Etienne,
Président du jury

Mme Katja SCHLADITZ
Fraunhofer ITWM, Examinateur

M. Serge BEUCHER
MINES Paristech, Examinateur

M. Maxime MOREAUD
IFP Energies Nouvelles, Examinateur

Soutenue par **Sébastien DROUYER**
le 1er décembre 2017

Dirigée par **Serge BEUCHER**
& Michel BILODEAU

MINES ParisTech | PSL RESEARCH UNIVERSITY PARIS

Remerciements

Bien qu'elle soit le fruit d'une collaboration avec IFP Energies Nouvelles, cette thèse a été effectuée dans les locaux de Mines Paristech à Fontainebleau. Ces trois années sont décidément passées trop vite, et je garderai un excellent souvenir de cette thèse, tant du côté travail que du côté social.

Tout ceci n'aurait pas été possible sans mes encadrants qui m'ont suivi ces années-là: Serge Beucher (CMM), Michel Bilodeau (CMM), Maxime Moreaud (IFPEN) et Loïc Sorbier (IFPEN). Tout d'abord, ayant travaillé avant dans un domaine différent, je n'ai pas exactement le profil type des doctorants qu'ils ont l'habitude de recruter; ils ont osé prendre un risque, et je leur suis reconnaissant pour cela. Ensuite, ils ont su, pendant la thèse, faire preuve d'ouverture, de support et de confiance. Ils m'ont donné une autonomie suffisante me permettant d'explorer de nouvelles idées, parfois des bonnes, souvent des mauvaises, et ont su, aux bons moments, me proposer de nouvelles pistes. Ils ont aussi fourni d'excellentes remarques pour améliorer ce manuscrit et les différentes présentations et articles produits durant ces trois années.

Je souhaiterais aussi remercier les autres membres du jury - Joachim Ohser (rapporteur), Jean Marc Chaix (rapporteur), Thierry Fournel (examinateur), Katja Schladitz (examinateur) - pour leur intérêt et leur implication dans l'évaluation de cette thèse et lors de la soutenance. Merci encore pour leur nombreuses questions et remarques.

Mais cette thèse n'aurait pas été si épanouissante, tant d'un point de vue scientifique que personnel, sans aussi l'environnement dans lequel je me suis trouvé.

Tout d'abord, le Centre de Morphologie Mathématique, dans lequel j'ai travaillé ces trois dernières années.

Merci d'abord à Catherine et Anne-Marie, les deux secrétaires du CMM, qui accueillent toujours les doctorants avec bienveillance.

Merci aussi aux membres permanents du centre - Dominique Jeulin, Fernand Meyer, Jesus Angulo, Beatriz Marcotegui, Petr Dokladal, François Willot, Matthieu Faessel, Serge Koudoro, Bruno Figliuzzi, Jose-Marcio Martins Da Cruz, Santiago Velasco-Forero - qui, malgré souvent une grande expérience dans leur domaine, font preuve d'une grande ouverture et tissent toujours des liens avec les doctorants.

Et il y a bien sûr tous les doctorants et post-doctorants que j'ai pu rencontrer durant cette thèse.

Tout d'abord, ceux qui ont eu la chance (ou la malchance) de partager avec moi leur bureau : je pense à Vaïa, Haisheng, Kaïwen et Albanne (le lecteur averti remarquera la composition presque exclusivement féminine de cette liste - fait rare en science et en traitement d'image). Vaïa et Haisheng étaient déjà là depuis un an quand je suis arrivé, et ils m'ont superbement accueilli et intégré au groupe: merci encore pour votre gentillesse et votre humour, j'ai grandement apprécié ces deux années avec vous. Kaïwen et Albanne sont quant à elles arrivées quand j'étais en fin de thèse, donc cela a été mon tour de passer le flambeau. J'espère qu'elles sauront passer outre ce traumatisme et vivre le même épanouissement que j'ai vécu durant cette thèse.

Ensuite, je souhaite remercier les autres doctorants ainsi que ceux qui sont passé par le CMM: Emmanuel, Bassam, Joris, Enguerrand, Gianni, Jean-Charles, Jean-Baptiste, Théodore Flavien, Amin, Robin, Eric, Elodie, Luis, Sara et Luc. Des remerciements spéciaux à Jean-Charles, qui, ayant un sujet similaire au mien, m'a toujours aidé quand je rencontrais des points de blocage.

La délégation de Fontainebleau n'est pas uniquement composée du CMM mais aussi d'autres centres: malgré des intérêts scientifiques différents, les doctorants et post-doctorants se mélangent et il y a un excellent esprit de groupe. Merci d'abord à Marine et Aurélien pour avoir brillamment repris l'association Dopamine, organiser les nombreuses soirées et contribuer à cette excellente ambiance. Merci aussi à Léo, Rémi, Ricardo, Manon, Pierre, Tianyou, Nicolas, Angélique, Jean, Arezki, Laure, Julien, Martin, Paule, Samy, Jihane, Emilie. J'ai grandement apprécié votre compagnie, et j'espère qu'on gardera contact.

Enfin, merci à ma famille et surtout mes parents pour m'avoir toujours apporté du support dans mes projets personnels ou professionnels. Rien de tout cela n'aurait été possible sans vous.

Contents

Algorithm 1: Algorithm's name

```
// This is a single line comment.
/* This is a multiline comment.                                    */
a ← 2 ;                    // The variable a is assigned the number 2.
while condition do
    // This code will be looped while condition is True.
    // This code also.
for i ∈ [1, 2, ..., λ] do
    /* This code will be executed with i = 1, then i = 2, and so on
       until i = λ.                                                 */
foreach i in [1, 2, ..., λ] do
    /* This code will be executed with i = 1, then i = 2, and so on
       until i = λ.                                                 */
Function functionName(α, β)
    /* This is the inside of the function functionName.  The function
       has 2 parameters, α and β.  Algorithms will generally be
       presented as functions in order to make their inputs and
       outputs explicit.                                            */
    sum ← α + β ;
    return sum ;      /* The function ends and returns the value of the
     parameter sum.  */
functionName(α, β) ;        /* This is how the function functionName is
 called.  */
```

Figure 1: Pseudocode convention. *In this work, we will often use pseudocode to describe existing and new algorithms. This figure shows the convention that will be used.*

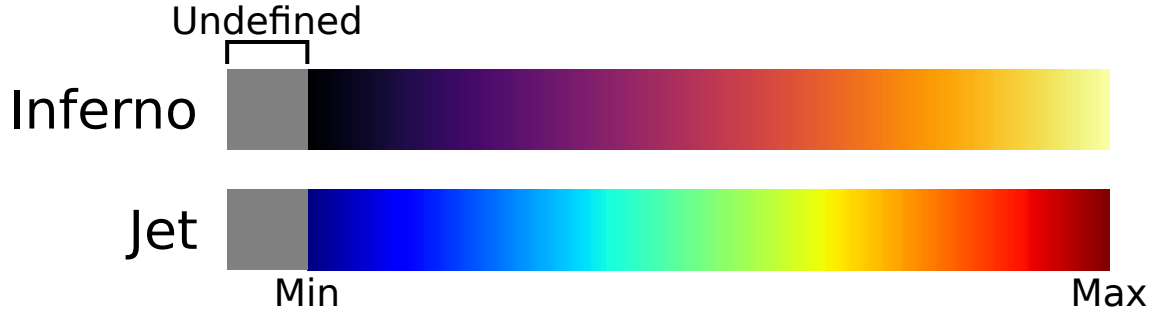


Figure 2: *Palettes used for displaying maps (disparity maps and height maps). The most used palette is Inferno as it is perceptually uniform. Undefined values are represented using the gray color.*

Chapter 1

Introduction

French summary / Résumé en français

L'objectif de ces travaux est de reconstruire la surface 3D de catalyseurs à partir d'images de Microscopes Electroniques à Balayage (ou MEB). Ce chapitre présente le sujet de thèse, le contexte, ainsi que le plan du manuscrit.

IFP Energies Nouvelles (IFPEN) is a major research and training player in the fields of energy, transport and the environment. IFPEN develops technologies to address the demand for major chemical intermediates and oil products meeting the strictest standards in terms of vehicle pollutant emissions, at the same time reducing their environmental impact. In this context, the analysis of the catalysts and their supports during their development stage is an essential step. The topography of these chemical prototypes provides important information about their properties. As these basic materials need to be evaluated on a micrometer scale, Scanning Electron Microscopes (SEM) are the tools of choice for obtaining correct quality images.

1.1 Context and application

This work aims to create a robust and generalist solution to establish the topography of materials from images of Scanning Electron Microscope. Initially, IFPEN will use this solution to analyze the surface of catalysts, catalyst supports and adsorbents (see Figure 1.1), but it is planned to extend its use to other materials.

The idea to use SEM images to evaluate the surface topography of materials is not new. However, conventional techniques do not perform well enough for the analysis of microscopic crystals because of their flat and smooth surfaces. This is the case of many catalysts, and this will be one of the major issues that this work will try to solve.

Why study the topography of catalysts?

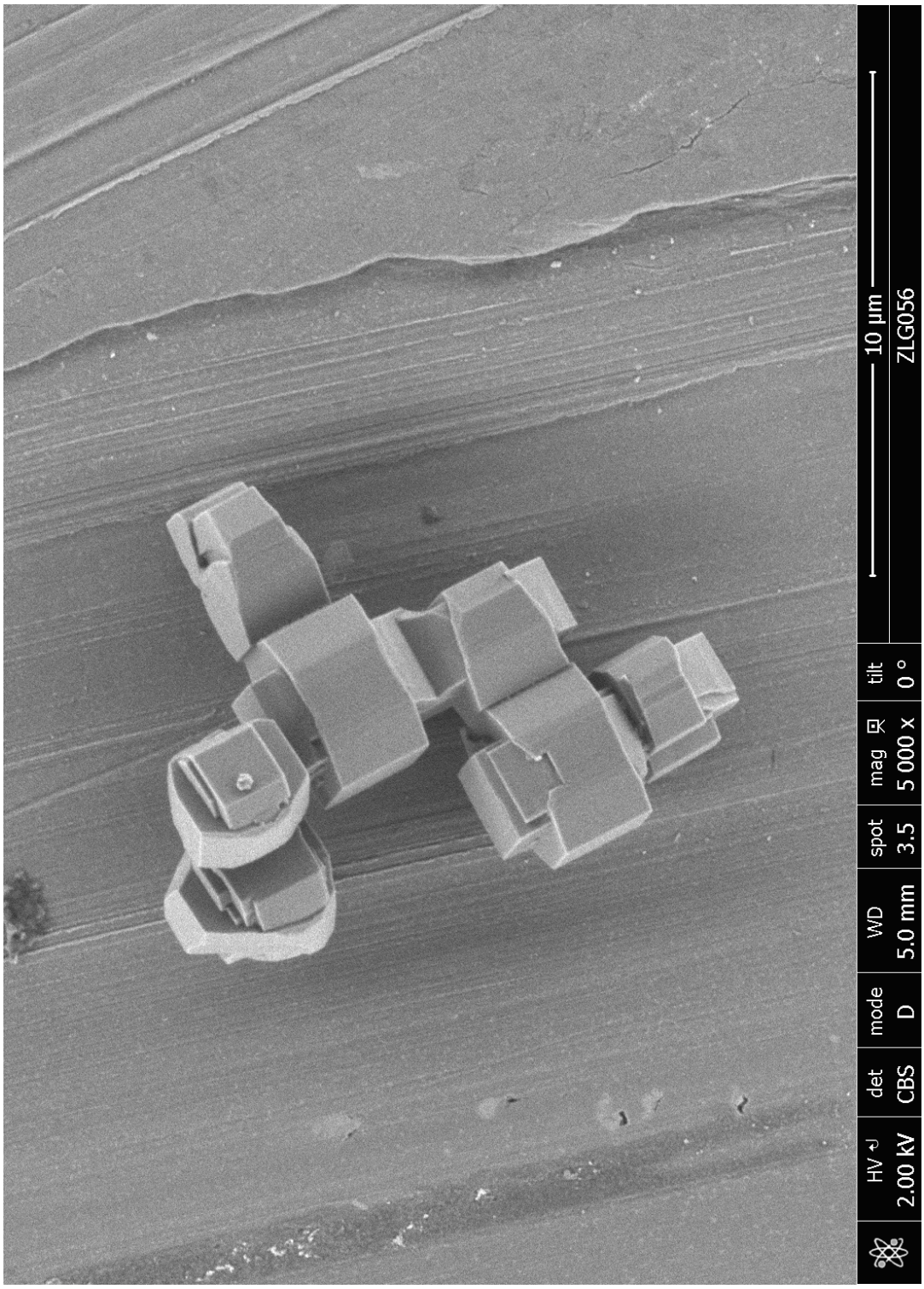


Figure 1.1: A SEM image of a catalyst support (Zeolite) for which a topographic characterization is sought.

The role of catalysis is to improve the efficiency of a chemical reaction and / or promote a chemical reaction rather than another. Combined with appropriate temperature and pressure conditions, it increases the speed and efficiency of a reaction.

The effectiveness of the catalysts is closely related to the morphology, size and localization of the active sites. Those are the main zones which will improve the efficiency of the reaction, however their size are too small (at the atomic scale, up to several nm) to be detected by the SEM.

The efficiency of a catalyst can, however, be related to the geometry of the support (including when it has active sites on its surface). The shape of the support may also have an influence on its mechanical strength.

Knowing the surface topography can also inform about the mechanisms of synthesis of this support, its crystallinity (amorphous phase, monocrystalline, polycrystalline). It can also serve as a tool for segmenting sub-parts of this support (agglomerates) probably more efficiently than from a single image.

1.2 Thesis objective

The topography of a microscopic element can be obtained using several ways: tomographic reconstruction, shape from shading and stereo reconstruction are the most widely used methods allowing to attain this objective. However, as it will be shown in chapter 4, only stereo reconstruction is adapted to the study of catalysts and their support.

Broadly speaking, stereo reconstruction is the task of obtaining a topography of an object or scene from several images. This field is very active and there is an array of applications. Since the 1980s, there have been attempts to reconstruct the topography of urban areas from multiple images taken from planes and satellites. The reconstruction of human-scale scenes are used in the field of robotics for object handling in industry or autonomous driving. In an even smaller scale, stereo reconstruction has been used on microscope images for checking the topography of integrated circuits.

Once several properties of the imaging process have been determined, such as the relative positions and orientations of the different shots or the distortions occurring in the images, the problem can be simplified to a matching problem: we need to determine, for each pixel of a reference image, its corresponding position in another image.

This isn't however a simple problem. Though most algorithms work very fine on textured images, pixels located in homogeneous areas are often misevaluated. This is due to the inherent locality of most methods: they try to match each pixel between images by looking at a fixed neighborhood, without adapting their functioning to the images at hand.

We can then understand why segmentation tools - which partition the images into a set of homogeneous regions - can bring a lot to stereo reconstruction techniques. But, once again, its use is not straightforward.

First, the segmentation's sensitivity must be carefully chosen. On the one hand, if it is too sensitive, the images might be separated into too much non-significant regions. On the other hand, if it is too insensitive, important borders in the images might be missed in the segmentation, merging objects that should be separated.

Secondly, the method must be able to handle occlusions - areas that are visible in an image but occluded in another one. Such phenomenon could lead to regions that disappear between two images, challenging the matching process.

The purpose of this thesis is to provide answers to these issues. The approaches that have been developed are not specific: they can be applied on their direct application - surface reconstruction of catalysts and their supports - but also on all stereo reconstruction problems in general. They have also been applied to satellite images and the well-known Middlebury 2014 database. These applications will notably be used to quantitatively compare the performance of our method compared to other state-of-the-art methods.

1.3 An overview of the thesis contents

The images we will be working on are not generated the same way than common digital photos are. Indeed, Scanning Electron Microscopes create images from the electrons they collect, whereas common cameras create images from photons. Lightning properties therefore tend to be different; we will explain how and why these differences exist in Chapter 2.

The existing stereo methods as well as the methods we created use several and common tools of image processing and mathematical morphology. We will introduce them in Chapter 3.

The topography of a microscopic sample can be obtained using several families of methods. We will first introduce them, but since stereo methods are best suited to our objective, we will describe the existing methods more thoroughly in Chapter 4.

We will then tackle the second part of the thesis and describe several methods that we developed and tested.

Since local and semi-global methods were unable to produce satisfying results despite our efforts to adapt them to our specific images, we created a method relying on segmentation tools. As expected, although results were not perfect, they appeared more in line with the ground truth. We will describe this new method - Top-Down Segmented Matching (TDSM) - in Chapter 5.

The TDSM method has however a major limitation: it supposes that all regions are fronto-parallel, that is to say parallel to the camera's plan. To take into account the potential slope of regions, we created a new, improved method - Top-Down Segmented Regression (TDSR) - that we will describe in Chapter 6.

In Chapter 7, we will then qualitatively and quantitatively compare the different methods we developed with the state of the art.

Finally, in the concluding Chapter 8, we will present different areas of improvements as well as new potential applications.

Part I

State of the art and existing tools

Chapter 2

SEM acquisition properties and tools

French summary / Résumé en français

Les travaux de cette thèse seront principalement appliqués aux images acquises avec des microscopes électroniques à balayage. Il est donc important de d'abord comprendre comment ces images sont générées ainsi que les effets de bords potentiels. Après décrire brièvement le fonctionnement général des microscopes électroniques à balayage, nous présentons les capteurs utilisés durant ces travaux ainsi que les principaux facteurs influençant l'intensité de gris des pixels. Nous décrivons aussi les principales difficultés survenant lors de l'acquisition des images.

As obtaining 3D topographies from Scanning Electron Microscope (SEM) images is our main objective, it is important to understand how our images were created as they constitute our input data.

SEMs do not generate images from light, as it is the case with optical microscopes, but generate images by capturing electrons. The resulting images are therefore very different from those of optical microscopes.

There is a high variety of SEMs and sensors available in the market. We will here discuss the general functioning of a Scanning Electron Microscope and the main types of sensors. The objective of this chapter is to explain the different factors that will influence the intensity values of pixels in SEM images.

2.1 SEM general functioning

Images of a Scanning Electron Microscope is shown in Figure 2.1 and a general diagram is shown in Figure 2.2. We will briefly explain the SEM image generation process by relying on these figures.

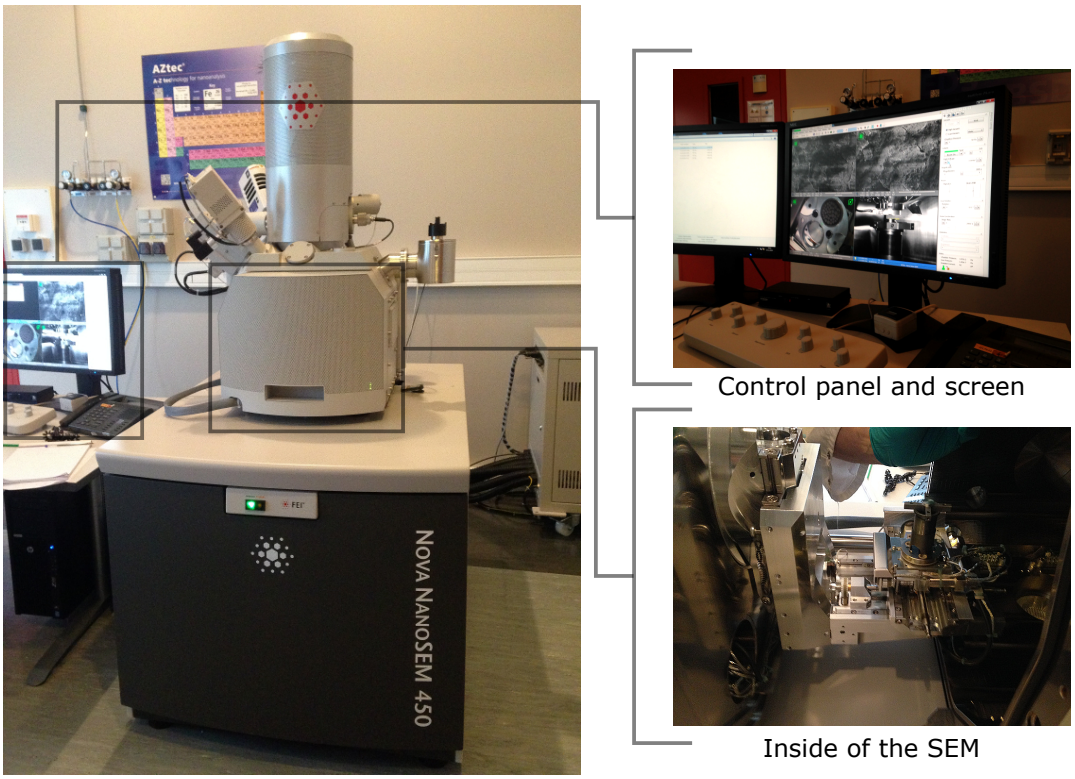


Figure 2.1: Images of a SEM.

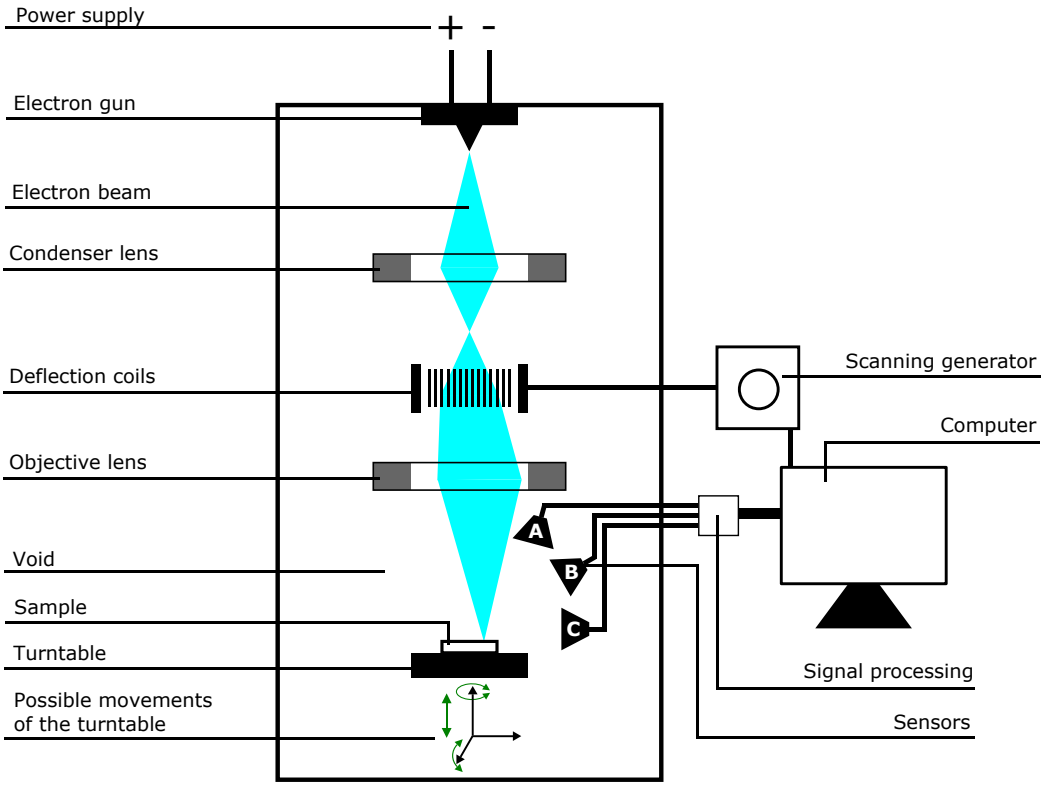


Figure 2.2: Diagram of a SEM.

First, an electron gun produces an electron beam in the upper part of the SEM. The beam passes through several condenser lens to make the beam converge. It then passes through deflection coils, controlled by a scanning generator. The objective of this group of components is to control the position on which the electron beam will converge. The scanning generator, as its name indicates, successively modifies the position of the point of convergence of the beam in order to scan the sample to be analyzed.

After passing through a last condenser lens (often named *objective lens*), the electron beam then collides with the sample. Some of the electrons bounce back, and it is those electrons that will be later collected and measured by the surrounding sensors.

The sample is placed on a turntable, whose position and orientation can be controlled by the computer. The turntable can move along the vertical axis, rotate around the vertical axis and around one or two horizontal axes, depending on the microscope. On the horizontal axes, the amplitude of rotation is generally limited (from -30 degrees to 30 degrees for example). For the FEI Nova NanoSEM 450 model (used for this work), the plate can rotate around two axes: the vertical axis and an horizontal axis. We will note **tilt** the angle of the turntable around the horizontal axis (see Figure 2.3).

Figures 2.4 and 2.5 illustrate how a SEM image is generated. In our example, the computer sent the instructions to the SEM to scan a specified area of the sample. The desired image is 5×5 pixels wide. The area to be scanned can therefore be divided into a 5 by 5 grid, each pixel of the image corresponding to a cell of the grid.

The scan generator is first set to target the cell corresponding to the pixel at the top left of the image. An electron beam is then activated for a determined time, during which the sensors surrounding count the number of electrons they receive. Once the measurement is completed, the number of electrons is converted to a gray value (according to defined parameters) that is assigned to the associated pixel. The more electrons have been received by the sensor, the lighter the pixel will be. The SEM then targets the cell corresponding to the next pixel, and the cycle begins again until the scan is complete.

It is possible to choose the trade-off between the quality of the image and the acquisition time. For our samples, getting usable images can take between 15 and 60 seconds.

2.2 Backscattered electrons, secondary electrons, and sensors

Depending on the type and position of the sensor, different images can be generated.

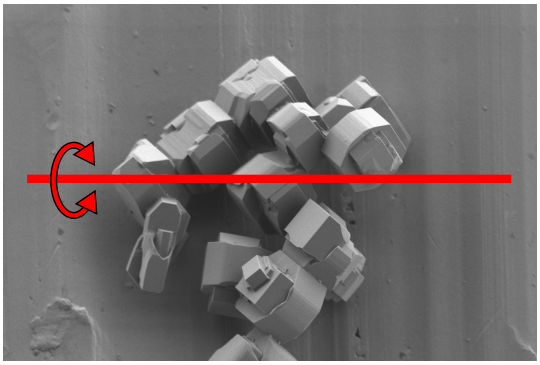


Figure 2.3: Tilt axis in SEM images.

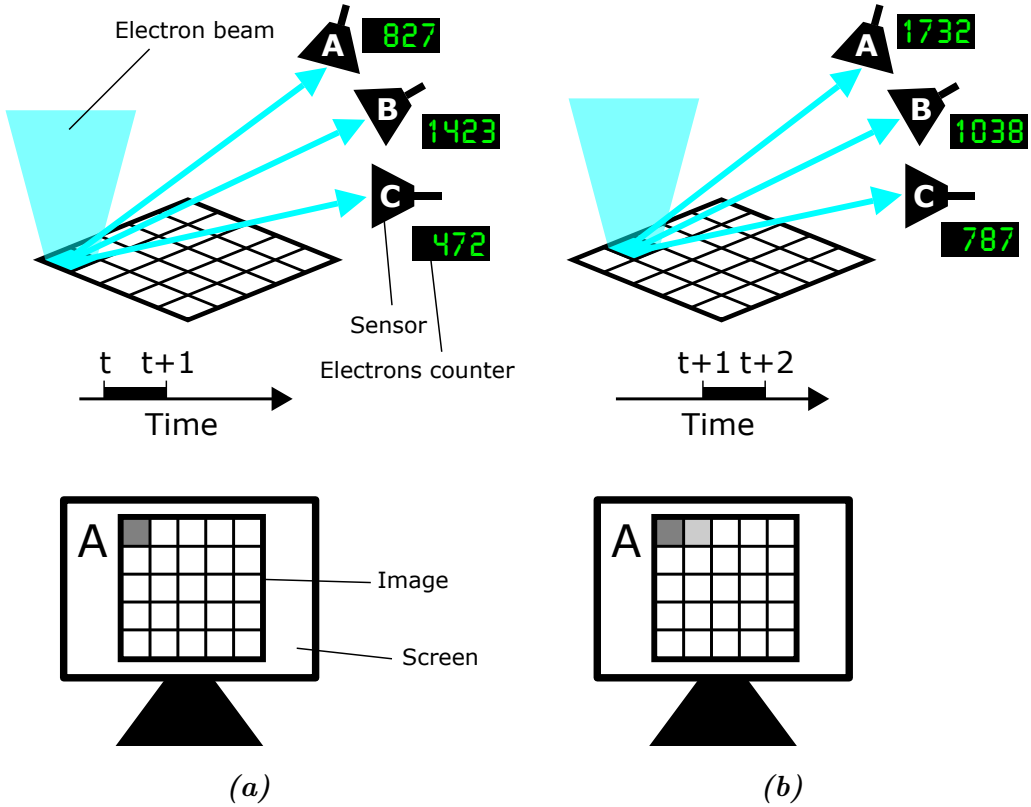
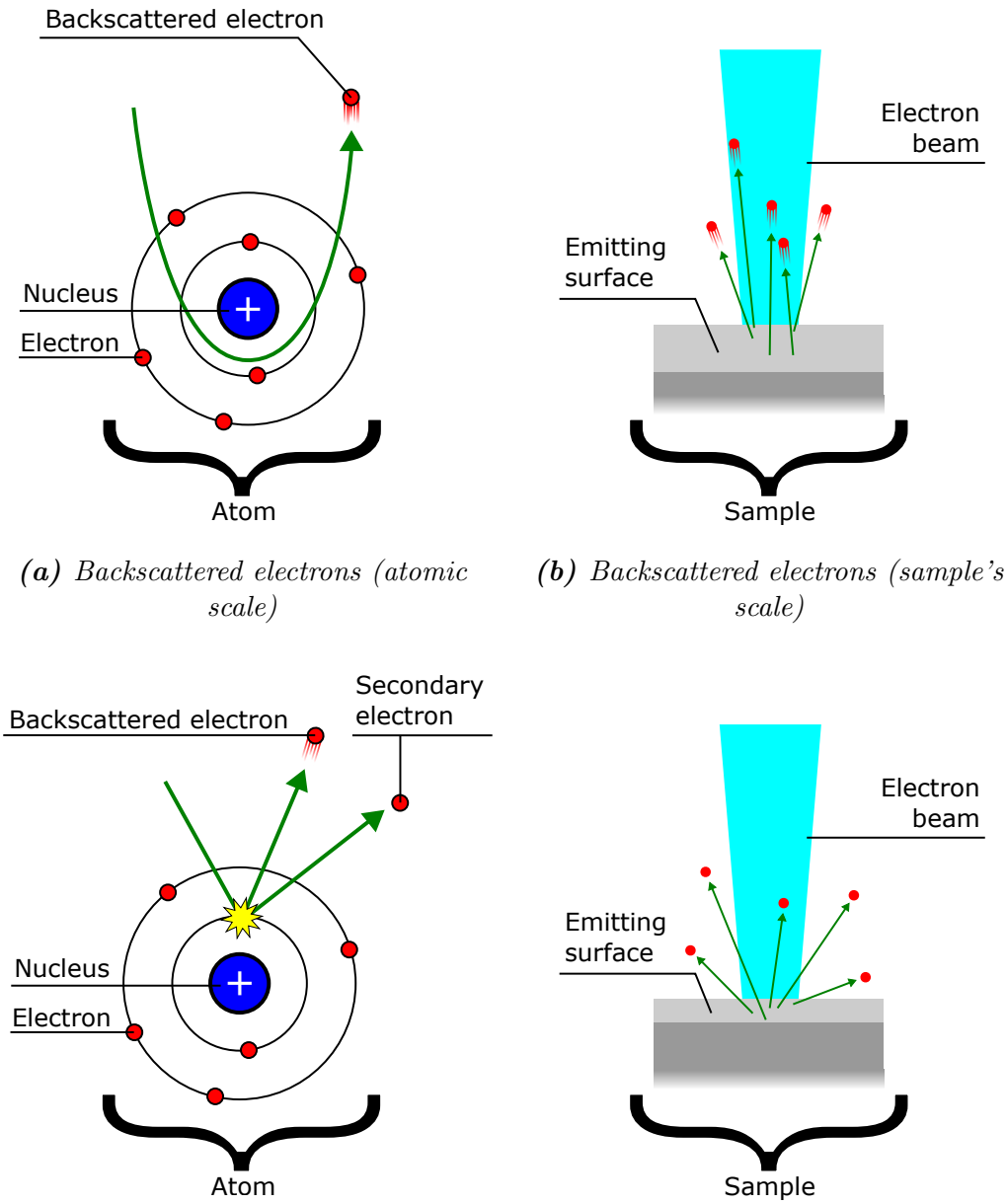


Figure 2.4: Illustration of the scanning process in SEMs. *Subfigure (a) represents the scanning process for getting the gray value of the top left pixel of the image. Subfigure (b) represents the process for the next pixel. The upper parts illustrates what is happening near the sample inside the microscope. The lower parts show the new pixels values that are added after each step, in the image generated by the detector A.*

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Figure 2.5: Order at which pixels of an image are scanned in most SEMs.

When the electrons of the beam come into contact with the sample, they will interact in several ways. The sensors will be able to detect 4 different signal types: backscattered electrons, secondary electrons, Auger electrons and X-rays. Some of the electrons from the beam will also simply disperse in the sample and will therefore not be detected by the sensors. We will focus on backscattered and secondary electrons, as we will only use images generated from these electrons.



(a) *Backscattered electrons (atomic scale)*

(b) *Backscattered electrons (sample's scale)*

(c) *Secondary electrons (atomic scale)*

(d) *Secondary electrons (sample's scale)*

Figure 2.6: Backscattered and secondary electrons.

During the scanning process, when an electron enters the sample, it collides many of the sample's atoms: electron-matter interactions are said to be very strong. These collisions deviate the electron's trajectory due to a physical process that is called **scattering**.

When the electron collides with an atom's nucleus, its initial energy is (almost) conserved: the scatterring is said to be **elastic**. On the other hand, when the electron collides with an atom's electrons, it loses a portion of its initial energy: the scattering is then said to be **inelastic**. It can also eject the atom's electron in the process which can eventually leave the sample and be detected by SEM's sensors. These electrons are called **secondary electrons**.

Even if the scattering process generally only slightly deviates the electrons, the very high number of collisions can reverse their direction, leading them to bounce back from the sample by its upper surface. These electrons are called **backscattered electrons**.

However, sensors are unable to precisely distinguish backscattered and secondary electrons. By convention, high energy electrons ($E_k > 50$ eV) are classified as backscattered electrons whereas low energy ones ($E_k < 50$ eV) are classified as secondary electrons.

The fraction of backscattered electrons emitted by the sample is called the **retrodiffusion coefficient**. It depends on the local mean atomic number (local composition), on the topography (slope of the surface) and for well crystalized samples, on crystal orientation due to electron channeling. At normal incidence, the angular dependence of the retrodiffusion coefficient follow a Lambert's cosine law (i.e. proportional to $\cos(X_{si})$, X_{si} being the angle between the surface normal and the direction of emission) [1] (p. 146). The emission of backscattered electrons is then maximal when the electron beam is aligned with the surface's normal. See figures 2.6a and 2.6b.

As secondary electron have a low kinetic energy, one consequence is that only secondary electrons emitted at a very superficial layer of the surface come out of the sample and are detected. A second consequence is that the portion of secondary electrons emitted largely depends on the slope of the surface (and therefore the topography). This relation makes the secondary electrons ideal for studying the topography of a material. See figures 2.6c and 2.6d.

In order to illustrate this dependence, we will study two simple cases: one where the electron beam collides with a perfectly plane surface, and one where the electron beam collides with an edge, an intersection of two perfectly plane surfaces but with different orientations.

When an electron from the beam collides with an atom's electron, a secondary electron will be emitted. Since the energy of this secondary electron is low, the more distance it has to travel, the higher is the probability that it will lose too much energy in the process to get out.

Let's suppose that the secondary electron has been emitted from a distance d under a plane surface. If the surface is perpendicular to the electron beam, the shortest path to the surface is still d. If the incidence angle between the surface and the electron beam is θ, then the shortest path will be $d \times \sin(\theta) < d$ (see Figure 2.7). Therefore, the lower the incidence angle, the higher is the probability that the electron will reach the surface and get out of the sample.

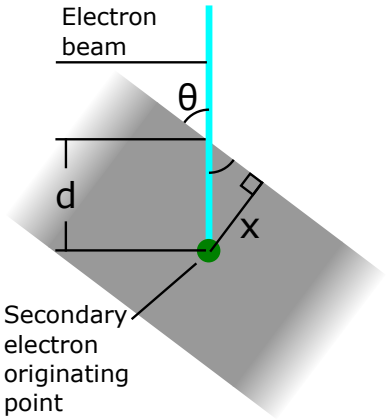


Figure 2.7: Shortest path to the surface. *The length of the shortest path x from the originating point to the surface can be computed from d and θ using trigonometry.*

As a result, surfaces that are perpendicular to the electron beam will appear darker in the image, and the lower the incidence angle, the lighter it will appear in the image. This relationship has been experimentally tested on flat samples [1], and the conclusion is that the number of emitted secondary electrons depends on $1/\cos(\theta')$ (for $|\theta'| < 80°$), θ' being the tilt angle of the sample. Therefore, the number of emitted secondary electrons depends on $1/\sin(\theta)$ if θ is the incidence angle between the surface and the electron beam. See Figure 2.8.

When the electron beam collides with a spike, it will appear lighter in the image (see Figure 2.8.C) as there will be more short paths to the surface. On the contrary, if it collides with a dip, there will be less short paths to the surface, and it will therefore appear darker in the image.

Another phenomenon that can be observed is that shadows seem to appear near objects in the foreground. In images created from secondary electrons, the cause of this shadow effect isn't the same as those created with natural light. Indeed, when the secondary electron leave the sample, since their energy is low, they can sometimes be attracted back to the sample due to electromagnetic forces. See figures 2.8.D and 2.9.

2.3 Sensors specificities

A same sample can be represented differently depending on the used sensor. Whether the sensor is set to detect backscattered electrons or secondary electrons is the main differentiating factor. This can be attained in multiple ways.

As the emission of backscattered electrons is maximal near surface normal at normal incidence, backscattered electrons detectors are placed above the sample, close to the objective lens hole: either above the objective lens (in-lens detector) or just below it (ring detectors).

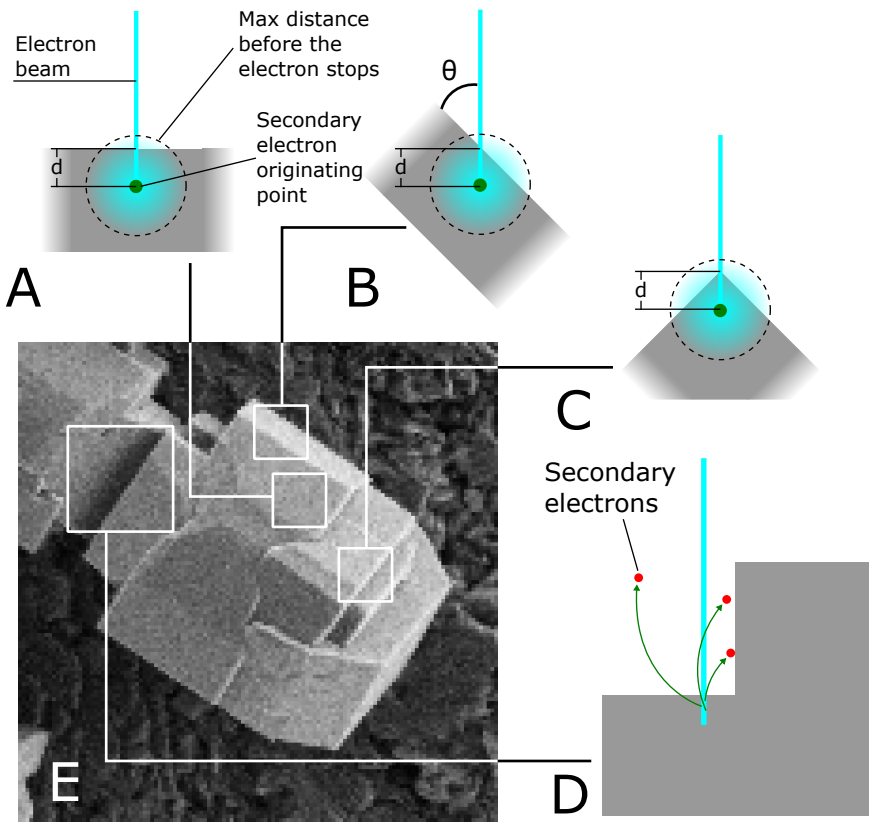
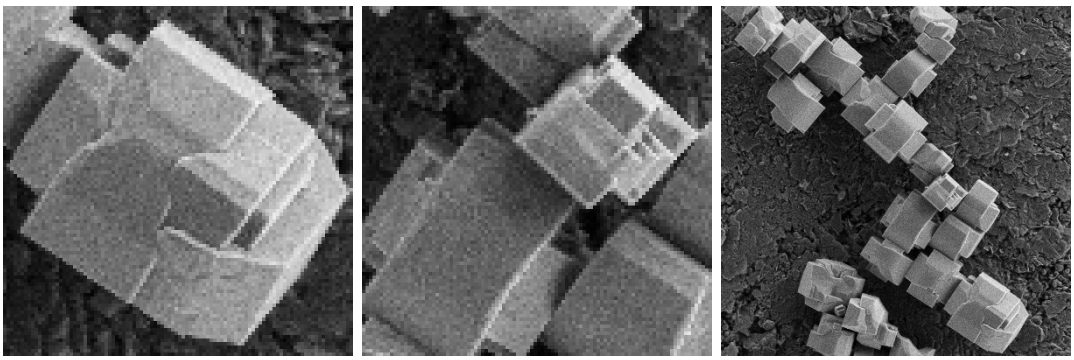


Figure 2.8: Influence of topography on images generated from secondary electrons. *A, B and C represent how the secondary electrons disperse within the sample in different cases. We have in each case schematized the penetration of an electron beam onto a surface. We also illustrated the dispersion of the electrons: the originating point is obtained after the electron beam has travelled a constant distance d inside the sample. Secondary electrons disperse in all directions, and the more they travel in the sample, the higher the probability will be that they will stop, captured by an atom (hence the gradient). A represents an horizontal surface, B represents an oblique surface and shows why more electrons are emitted in this case, C represents a spike and explains why spikes appear lighter in the images, D represents the shading phenomena.*



(a) ***(b)*** ***(c)***

Figure 2.9: Examples of SEM images generated from secondary electrons sensors. *In subfigure (a), we can observe the brightness of the different surfaces depending on their slope. In subfigure (b), in the center, we can observe the shadow effect as secondary electrons are attracted by an adjacent face. Subfigure (c) shows a shadow effect occurring around a crystal.*

Secondary electrons are more influenced by electromagnetic fields because of their low energy. The detector may then be placed laterally in the SEM chamber. Activating a field near a sensor can therefore attract or repel secondary electrons while having a negligible influence on backscattered electrons.

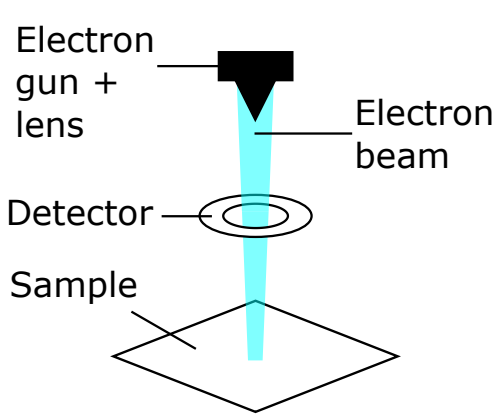
Even when sensors collect the same type of electrons, the produced image could be different due to a different sensor position or the way they collect electrons.

The sensors that are above the sample, such as the in-lens sensors, generate images with less shadow effects. Sensors that are placed at one side of the microscope, such as the Everhart-Thornley detector, generate images where the sample is lighter towards the sensor's side. See Figure 2.11.

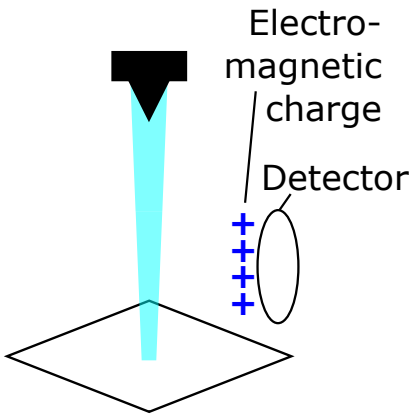
The Directional Back-Scatter (DBS) detector is positioned around the electron beam and is divided into 4 rings on which it will separately measure the amount of electrons it receives. The consequence is that each acquisition generate 4 complementary images. Since backscattered electrons tend to bounce back at the opposite direction than the beam, a higher portion will be collected by the smaller rings, closer to the beam. On the other hand, each ring will receive a similar amount of secondary electrons, since their direction is more random. The consequence is that the image produced by the smaller ring will be more dependent upon the chemical properties of the sample, and the one produced by the larger ring will depend more on the sample's topography. An example of DBS acquisition is shown in Figure 2.12.

The Angular Selective Back-scatter (AsB) sensor also generates 4 images. Instead of collecting the electrons in 4 rings like the DBS detector, it collects them in 4 different quadrants. The consequence is that each image highlight a different side of the sample, as if it were enlighten by light sources at different positions. An example of AsB acquisition is shown in Figure 2.13.

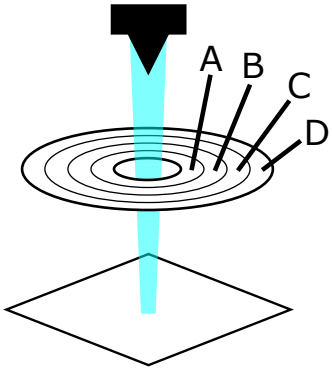
Section 2.5 presents a case study showing the relationship between the topography of a sample and observed gray levels in an image obtained using the Everhart-Thornley detector.



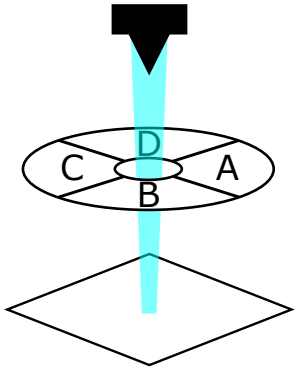
(a) In-lens detector



(b) Everhart-Thornley detector

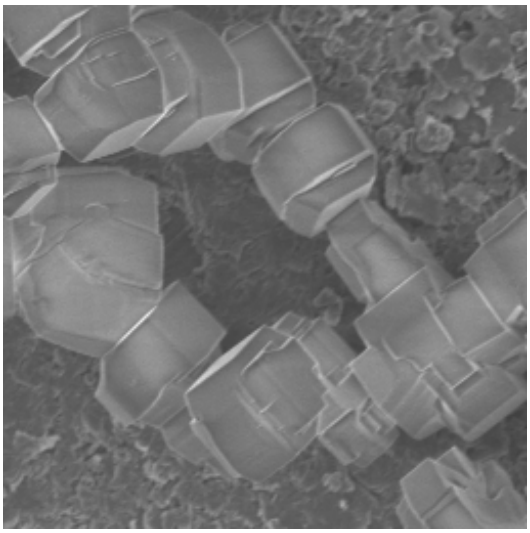


(c) DBS detector and its four rings.

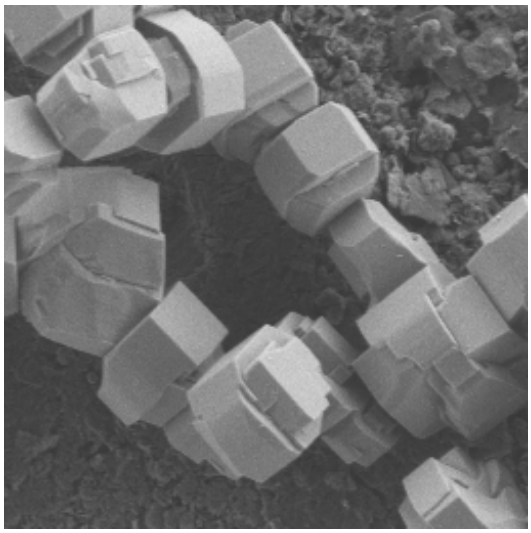


(d) AsB detector and its four quadrants.

Figure 2.10: Diagram of different detectors.

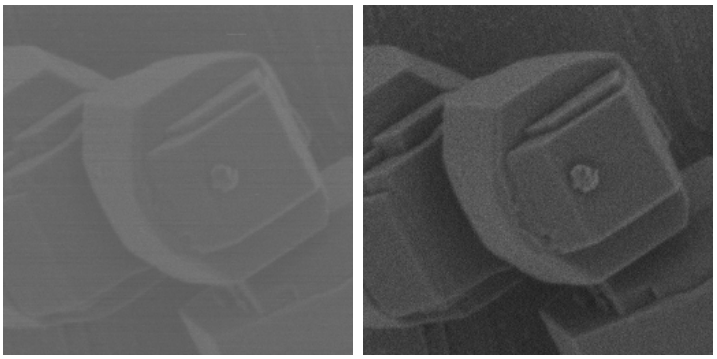


(a) In-lens sensor

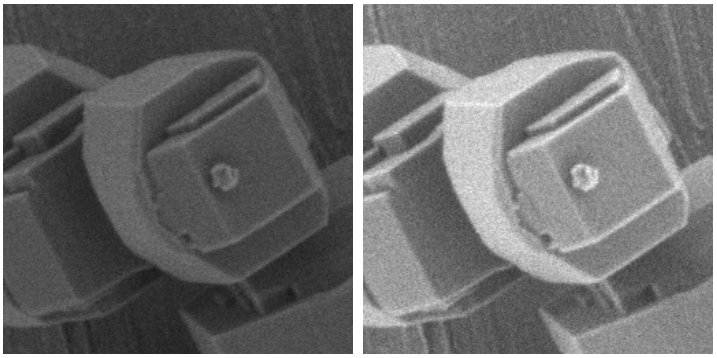


(b) Everhart-Thornley sensor

Figure 2.11: Images produced by secondary electrons sensors.

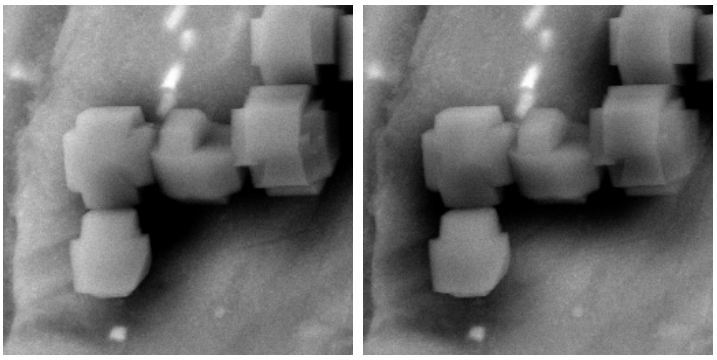


(a) *Ring A* **(b)** *Ring B*

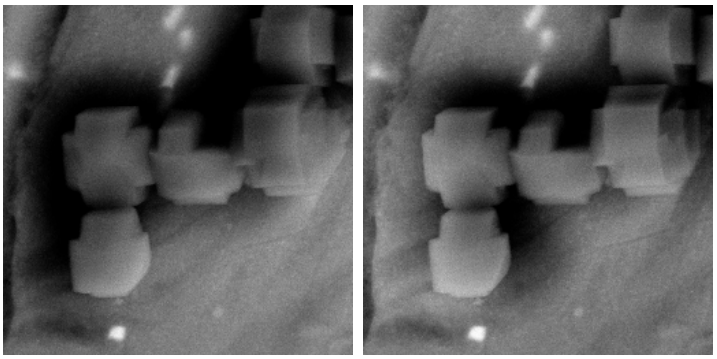


(c) *Ring C* **(d)** *Ring D*

Figure 2.12: Images of the different rings of the DBS detector.



(a) *Quadrant 1* **(b)** *Quadrant 2*



(c) *Quadrant 3* **(d)** *Quadrant 4*

Figure 2.13: Images of the different quadrants of the AsB detector.

2.4 Distorsions and acquisition issues

The specificities of the SEM images are not limited to the gray values of pixels but also concern distortions and issues that can happen during the acquisition.

First, due to the very nature of the acquisition process, produced images can contain a lot of Poisson noise [1] (see Figure 2.14). Indeed, gray levels depend on the number of electrons received by the sensors during a defined amount of time t. If t is too short, sensors will receive an insignificant and random amount of electrons. But if t is too long, other acquisition issues, later explained in this section, might be magnified. There is therefore a trade-off between the amount of noise in the image and acquisition time combined with potential other acquisition issues.

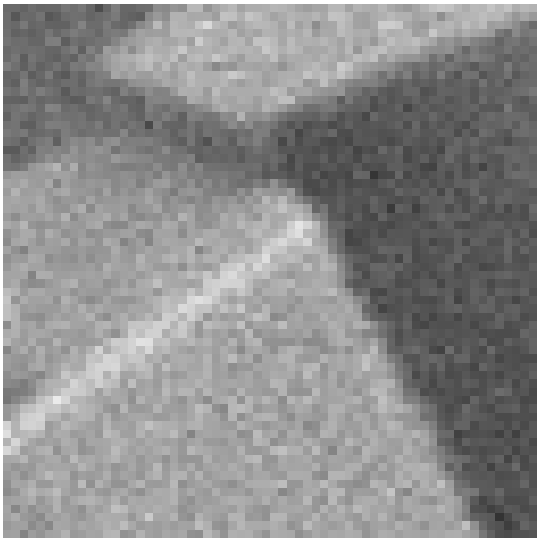


Figure 2.14: Poisson noise in SEM images.

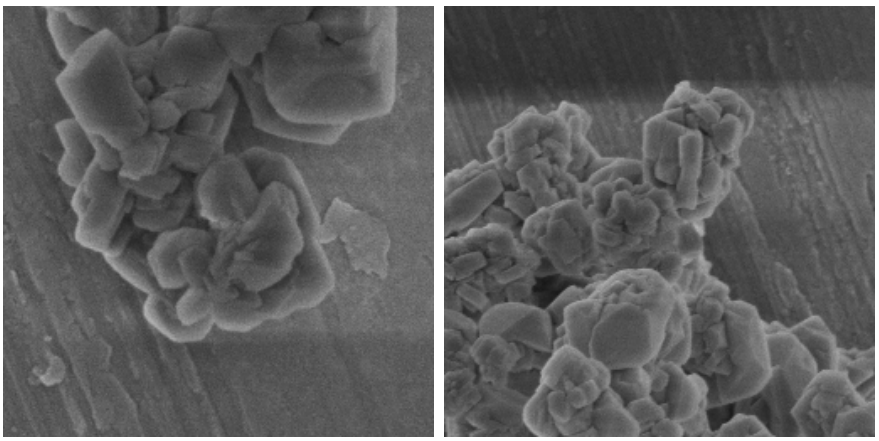
There exist various perspectives concerning the distortions in SEM images. For some, at least for acquisitions at magnifications greater than 500 $\times$, the effect of distortions is negligible[2]. For others, the consideration of distortions is essential for the correct estimation of the topography [3, 4]. For small magnifications, a radial distortion (magnifying effect) and / or a spiral distortion is often observed. Some distortions may also occur as a function of time, caused for instance by the accumulation of electrons on the surface of the sample.

Although SEMs are very precisely calibrated, there always exist small inaccuracies in the measured position and orientation of the turntable. According to [5], when comparing several views of the same sample with close tilts, these inaccuracies may have an impact on the quality of the measurements and must be taken into account.

In some cases where the samples are fixed with a gel, they can move over time. This shift is negligible when images are acquired within seconds of each other, but this should be taken into account when more than a minute elapses between two acquisitions.

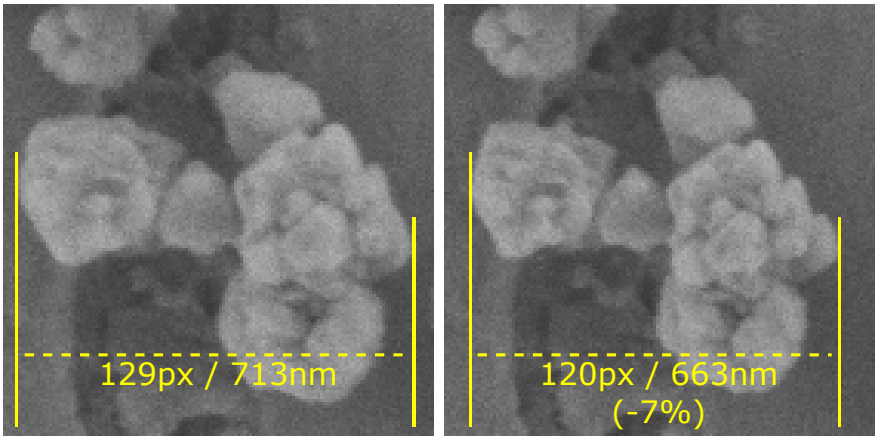
For insulating samples, electrons accumulate at the sample's surface and create an electrostatic potential disturbing further secondary electrons emission. It is often observed at high incidence electron energy with lateral detector (such as the Everhart-Thornley detector). This results in lighter levels in the image around the sample. The disturbance mainly appears at the right side or the left side, depending on the scanning direction. Figure 2.15 illustrates this phenomenon.

Finally, the electron beam can alter the molecular structure of some small samples. Such interactions result in the thinning of the sample or its contamination by neighboring residues. Examples of these phenomena are illustrated in Figure 2.16. Since the nature of this deformation varies greatly according to the sample and can't be easily predicted, care must be taken to ensure that this effect is as attenuated as possible during acquisitions.



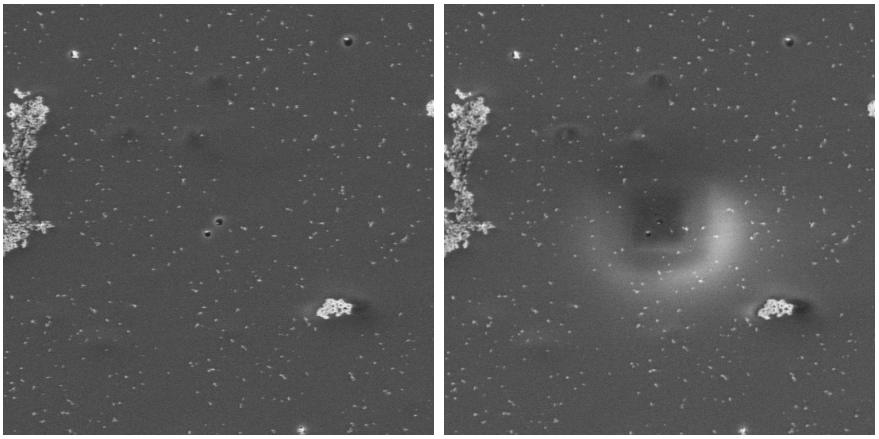
(a) *(b)*

Figure 2.15: Illustrations of the charge effect.



(a) *Thinning example 1st sample's acquisition*

(b) *Thinning example 4th sample's acquisition*



(c) *Contamination example 1st sample's acquisition*

(d) *Contamination example 4th sample's acquisition*

Figure 2.16: Examples of thinning and contamination effects due to the electron beam after several acquisitions. *The contamination effect has been obtained by focusing the scanning area to the center of the image and then zooming out: hence the square shape of the deformation.*

2.5 Case study: Topography and gray levels

We have described in this chapter the main factors that influence gray levels in an image. We will check them in this case study using a sample where the ground truth topography is known.

The market provides a set of calibration samples where the topography is known and guaranteed. They can be used in order to detect potential distortions in acquisitions and to calibrate the microscope. Here, we acquired images of the TGF11 silicon calibration grating [6]. The crystallography of silicon ensure that the sample's shape is an array of trapezoidal steps with a constant pitch, height, and geometry. Figure 2.17 shows an image of the sample taken using the Everhart-Thornley detector.

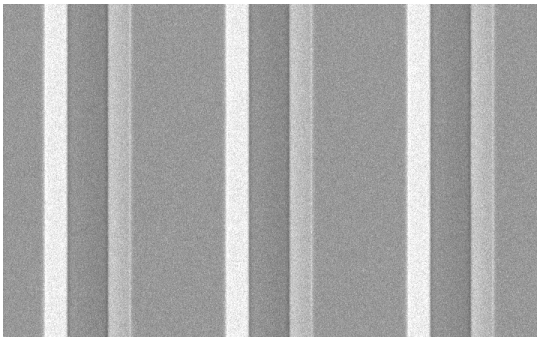


Figure 2.17: Image of TGF11 obtained using the Everhart-Thornley detector.

We extracted a single pattern and compared the evolution of gray levels with the known ground truth topography. Since the pattern is vertical, we computed for each column of the extracted image its average gray level to reduce the impact of noise. This comparison is shown in Figure 2.18.

We will now make the connection between the observed gray levels and what we discussed in this chapter, particularly in sections 2.2 and 2.3. We will read 2.18 from left to right.

In the ⓐ area, the surface of the topography is a horizontal plane, perpendicular to the electron beam. The gray levels are constant in this area.

ⓑ is the intersection of two planes at different angles. Since it is a spike, we observed a local maximum in gray levels.

There is then an oblique plane surface in ⓒ. This surface is not perpendicular to the electron beam. As a result, it appears lighter than ⓐ, with higher gray levels.

This plane surface meet another one in ⓓ. This time, this intersection is a dip, so gray levels experience a local minimum.

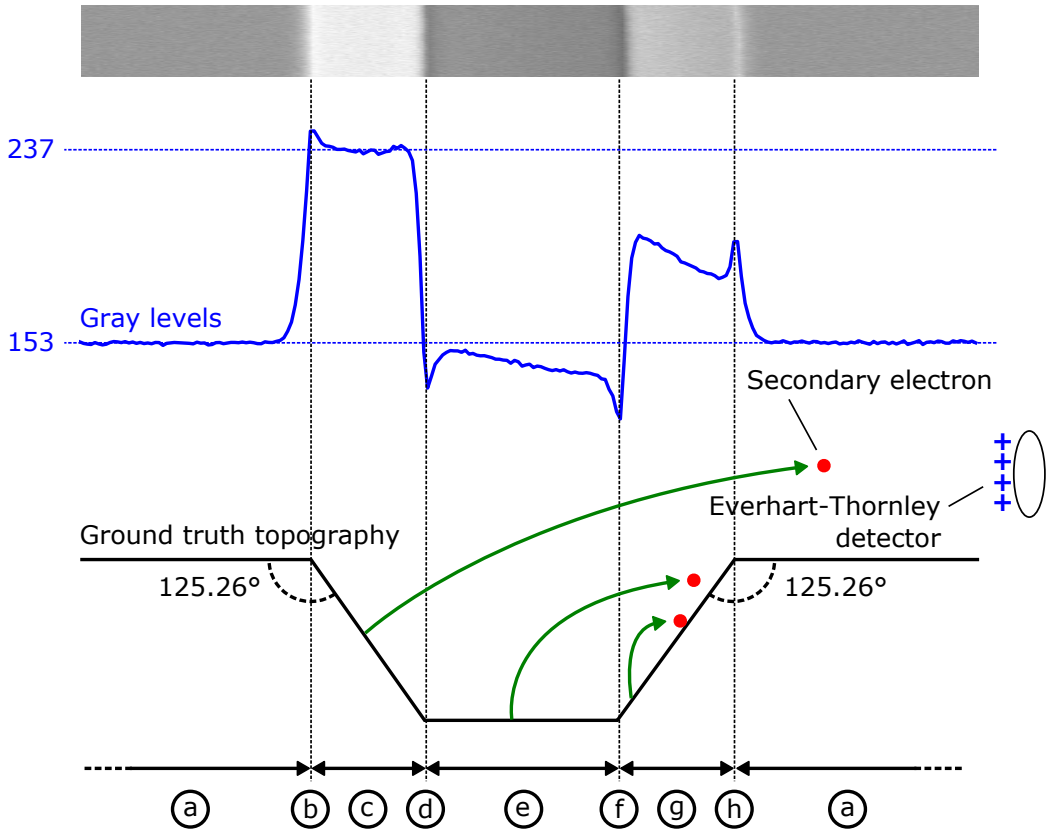


Figure 2.18: Comparison between the sample's topography and the observed gray levels.

ⓔ is, like ⓐ, a plane surface perpendicular to the electron beam. Gray levels recover first those observed in ⓐ, but then decrease. This is due to the position of the Everhart-Thornley detector and the position of the next surface. Indeed, a portion of secondary electrons are attracted by the next surface since it obstructs some otherwise possible paths to the detector: a shadow effect is occurring.

Gray levels in ⓕ, ⓖ and ⓗ can be explained by the same factors than ⓓ, ⓒ and ⓑ, except that a shadow effect is occurring, causing lower gray levels.

The reason behind the slight progressive decrease in gray levels in ⓖ near the spike is beyond the scope of this work.

2.6 Summary

In this chapter, we have presented the general functioning of a Scanning Electron Microscope and explained the image acquisition process. We have also shown the main factors that influence gray levels in the image: this information will help us understand some issues raised in Chapter 4.

Image-based topography reconstruction methods extensively use image processing tools. In order to present these methods, we will first introduce the tools they use in the next chapter.

Chapter 3

Introduction to image processing and mathematical morphology

French summary / Résumé en français

Les méthodes développées lors de nos travaux utilisent de nombreux outils de traitement d'images. Nous présentons ces outils dans ce chapitre en trois parties. Tout d'abord, les outils de traitement d'images tels que les convolutions ou le filtre médian. Ensuite, les outils de morphologie mathématique tels que la dilatation et l'érosion. Et enfin, les différentes méthodes de segmentation et segmentation hiérarchique permettant de diviser une image en régions et sous régions.

In order to explain existing and new methods of topography reconstruction, we will first define different tools of image processing and mathematical morphology that are used by these methods.

3.1 Image representations

As we saw in the previous Chapter (2), SEMs generate one or several grayscale images of the sample they acquire. In Computer Vision, a grayscale image is represented by a 2D matrix (see Figure 3.1b). Each element of the matrix, called **pixel**, generally contains an integer value between 0 and 255 (encoded in 8 bytes). This value represents the lightness of the pixel: 0 is black, 255 is white.

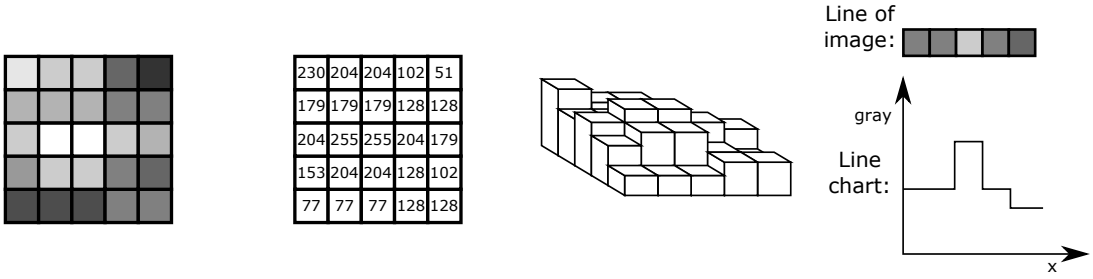
For illustration purposes, we can also represent these images as an elevation map: in this case, the lighter a pixel is, the higher its point will be on the elevation map (see Figure 3.1c). For a clearer view, operations are often illustrated on a slice of an image (a specific row or column of the matrix), that can be represented by a line chart (see Figure 3.1d).

It is common to consider images as functions. An image represented by a matrix M can be seen as a discrete function f such as $f(x, y) = M(x, y)$.

Binary images are similar to grayscale images but the value of each pixel can only take a boolean value: `True` (or 1) or `False` (or 0). Generally, pixels set to `True` are depicted in white and pixels set to `False` in black. Binary matrices can also be represented by binary images.

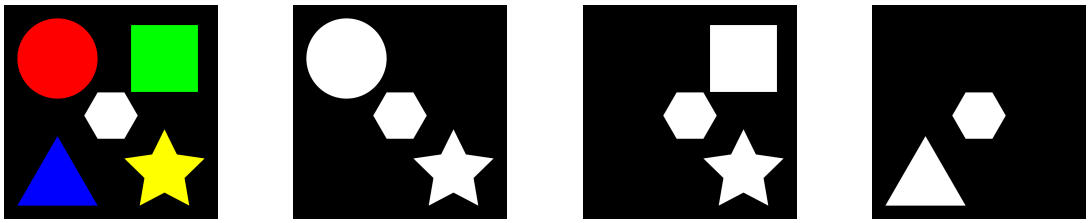
Color images are generally represented by the fusion of 3 grayscale images representing the Red, Green and Blue channels (see Figure 3.2). It is therefore a 3D matrix with dimensions $(width, height, 3)$. See Figure 3.2.

Images can also be represented by an hexagonal grid (see Figure 3.3). Among all the benefits of using hexagonal lattices[7], the most commonly reported is that pixels have a more circular shape than regular pixels. Thus, they have 6 direct neighbors at equal distance instead of 4 direct neighbors, resulting in less aliasing problems [8]. Moreover, it is more hardware-friendly since pixels' sensors usually have circular shapes: an hexagonal representation of the image could therefore allow a higher resolution.



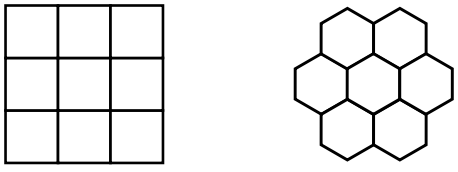
(a) *Grayscale image* ***(b)*** *Matrix* ***(c)*** *Elevation map* ***(d)*** *Line chart*

Figure 3.1: Different representations of an image. *(a), (b) and (c) represent the same underlying image.*



(a) *Color image* ***(b)*** *Red ch. of (a)* ***(c)*** *Green ch. of (a)* ***(d)*** *Blue ch. of (a)*

Figure 3.2: RGB channel of a color image. *Ch.: channel.*



(a) *Square grid* ***(b)*** *Hexagonal grid*

Figure 3.3: Square and hexagonal grids.

Notation	Meaning
$g = f + a$ (a is constant)	$g(x, y) = f(x, y) + a$
$g = f_1 + f_2$	$g(x, y) = f_1(x, y) + f_2(x, y)$
$g = \max(f_1, f_2, ..., f_n)$	$g(x, y) = \max(f_1(x, y), f_2(x, y), ...f_n(x, y))$
$g = f \ \mathcal{R} \ a$ (a is constant $\mathcal{R}$ is a relational operator e.g. $<$)	$g(x, y) = \begin{cases} \texttt{True}, & \text{if } f(x, y) \ \mathcal{R} \ a \\ \texttt{False}, & \text{otherwise} \end{cases}$
$g = f_1 \ \mathcal{R} \ f_2$ ($\mathcal{R}$ is a relational operator e.g. $<$)	$g(x, y) = \begin{cases} \texttt{True}, & \text{if } f_1(x, y) \ \mathcal{R} \ f_2(x, y) \\ \texttt{False}, & \text{otherwise} \end{cases}$
We suppose that $f_1 \ \mathcal{R} \ f_2$ ($\mathcal{R}$ is a relational operator e.g. $<$)	$\forall (x, y),\ f_1(x, y) \ \mathcal{R} \ f_2(x, y)$

Table 3.1: Element-wise operations

3.2 Image operations

An operator o takes one or several functions (or images) as input and outputs one or several functions. A basic operation can be $o(f) = f + 5$: it creates another image similar to f where the value of each pixel has increased by 5.

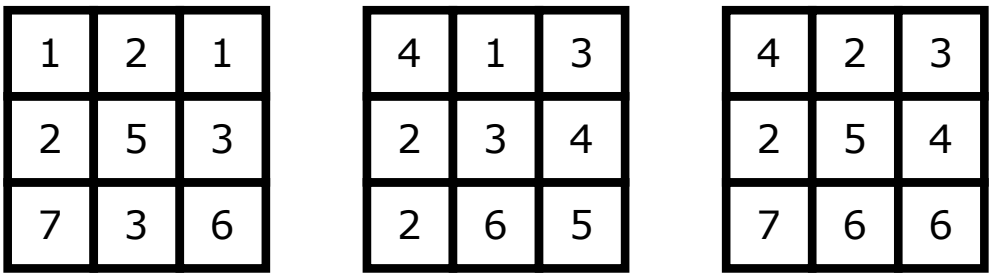
There are many operations that can be applied on images. We will mainly present the ones we will use during this thesis. We will also restrict their descriptions to 2-dimensional images, but most of them can be generalized to 3D images [9].

3.2.1 Classical operators

Element-wise operations

Let's suppose we have a list of n grayscale images $f_1, f_2, f_3...f_n$ of size (w, h). Element-wise operations are simple operations performed on each pixel of these images. For the sake of brevity, the notation of these operations is often simplified as shown in Table 3.1.

For instance $\max(f_1, f_2)$ computes the maximum of the two images f_1 and f_2. See Figure 3.4.



(a) f_1 ***(b)*** f_2 ***(c)*** f_g

Figure 3.4: Illustration of the maximum of images operator. $g = \max(f_1, f_2)$.

Convolution

In functional analysis, the convolution of two functions f and g is noted $f * g$, and is defined as follows:

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(y)g(x-y)dy = \int_{-\infty}^{+\infty} f(x-y)g(y)dy$$

However, in image processing, we use a discrete and two dimensional version of this convolution.

Let's suppose we have an image f of size (w, h) and a matrix g of size (w_g, h_g) with smaller size than f. g is generally called **kernel**. When g is a square matrix and its size is odd (this is the common case and the only one we will be using in this work), the **convolution** of f by g is:

$$(f * g)(x, y) = \sum_{i=-d}^{d} \sum_{j=-d}^{d} g(i, j) \cdot f(x-i, y-j)$$

With $d = \frac{w_g - 1}{2}$ and assuming the origin of the kernel is centered ($g(0, 0)$ is located at the center of g).

For each pixel of f we compute a weighted sum of neighboring pixels. The weights attributed to each neighbor are defined by the kernel g.

Depending on the kernel used, convolutions can have several applications, such as image blurring, sharpening or edge detection. See Figure 3.5.

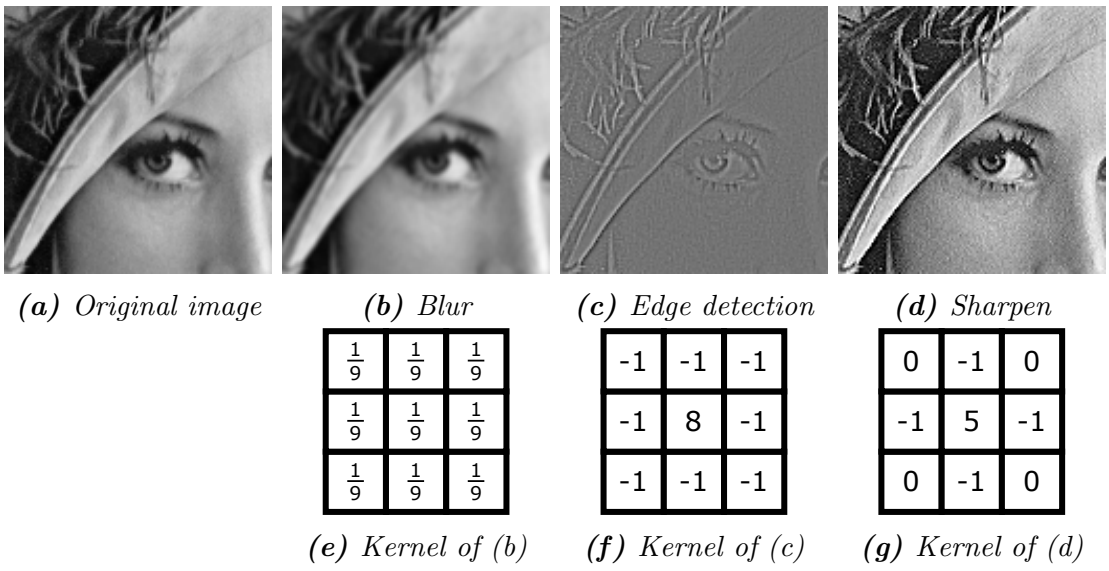


Figure 3.5: Some applications of convolutions. *The kernel used in (b) produces a blurred version of the image: it can be used to remove noise. The kernel used in (c) highlights edges in the image. The kernel used in (d) sharpens the image.*

Median filter

Let's suppose we have an image f of size (w, h) and a binary kernel g of size (w_g, h_g), with smaller size than f. The **median filter** [10] is defined as:

$$median(f, g)(x, y) = median\{f(x - i, y - j) \text{ where } g(i, j) = 1\}$$

Assuming the origin of the kernel is centered ($g(0, 0)$ is located at the center of g).

For each pixel of f we compute the median value of the neighboring pixels. The kernel g defines which pixels are considered as neighbors.

Contrary to the convolution filter presented in Section 3.2.1, the median filter is not linear. It is widely used in computer vision [11] because it can remove noise while better preserving edges than the blur convolution filter presented in Figure 3.5b. It is effective on small to moderate levels of gaussian noise, speckle noise and salt and pepper noise. See Figure 3.6.

Bilateral filter

Let's suppose we have an image f of size (w, h). The **bilateral filter** [12] is defined as:

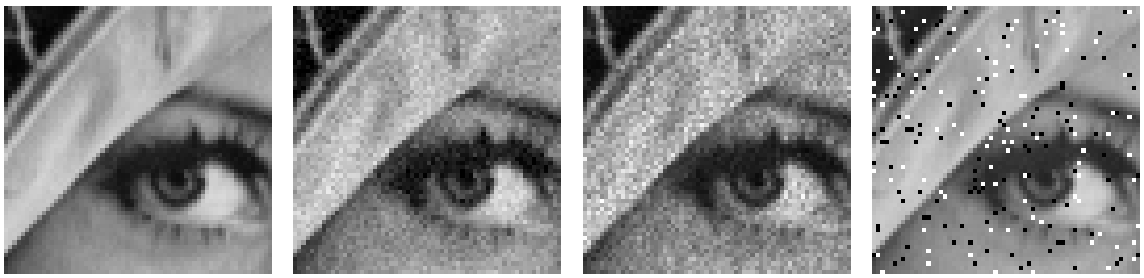
$$BF(f, d, g_r, g_s)(x, y) = \frac{1}{k} \sum_{i=-d}^{d} \sum_{j=-d}^{d} f(x, y) g_r(\|f(x-i, y-j) - f(x, y)\|) g_s(\|(x-i, y-j) - (x, y)\|)$$

With:

$$k = \sum_{i=-d}^{d} \sum_{j=-d}^{d} g_r(\|f(x - i, y - j) - f(x, y)\|) g_s(\|(x - i, y - j) - (x, y)\|)$$

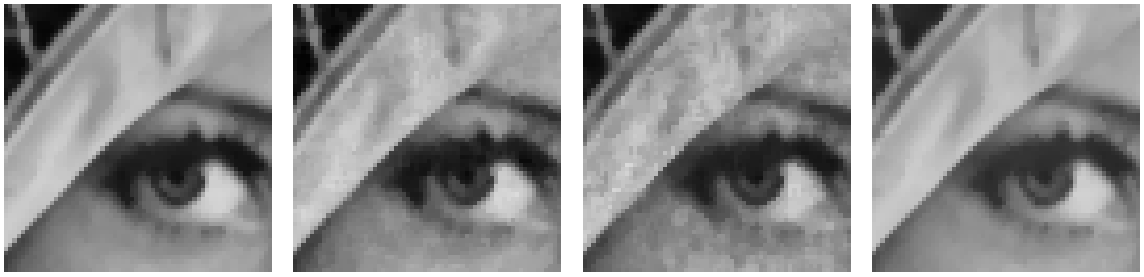
Where d, g_r and g_s are parameters and $\frac{1}{k}$ is a normalization term. d is the diameter of each pixel's neighborhood. g_r is the range kernel; it is generally a Gaussian function whose spread is defined by σ_r (specified as parameter). g_s is the spatial kernel; it is generally a gaussian function whose spread is defined by σ_s (specified as parameter).

As the blur convolution, the bilateral filter computes for each pixel of f a weighted average of neighboring pixels. However, the kernel defining these weights is not constant.

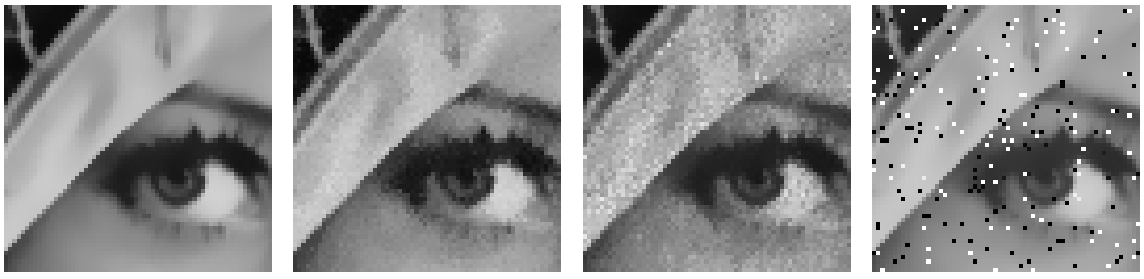


(a) *Image* ***(b)*** *(a)+Gauss. noise* ***(c)*** *(a)+speckle noise* ***(d)*** *(a)+S/P noise*

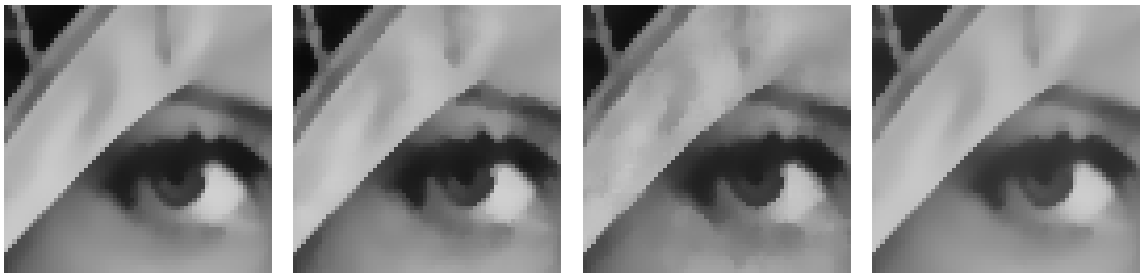
(e) *Blur filter on (a)* ***(f)*** *Blur filter on (b)* ***(g)*** *Blur filter on (c)* ***(h)*** *Blur filter on (d)*



(i) *Median filter on (a)* ***(j)*** *Median filter on (b)* ***(k)*** *Median filter on (c)* ***(l)*** *Median filter on (d)*



(m) *B filter on (a)* ***(n)*** *B filter on (b)* ***(o)*** *B filter on (c)* ***(p)*** *B filter on (d)*



(q) *MB filter on (a)* ***(r)*** *MB filter on (b)* ***(s)*** *MB filter on (c)* ***(t)*** *MB filter on (d)*

Figure 3.6: Comparison of the blur and median filter. *S/P: Salt and Pepper. B: Bilateral filter. MB: Median + Bilateral. The kernel size for the blur and median filter is 3x3. For the bilateral filter, we used* $\sigma_r = 20$ *and* $\sigma_s = 5$*. Gauss.: Gaussian.*

When computing the filtered value of the pixel P, the weight of a neighbor pixel N depends on two criteria. The farther away N is to P, the lower will be its associated weight. The more different the values of N and P are, the lower will be the weight of N. In other words, the kernel overweighs pixels that are near P and have a similar value.

This filter is generally used for removing noise in the image while preserving edges. Though the median filter is also used for this purpose, the strengths of the two filters are different. The median filter handles the speckle and salt and pepper noises better, whereas the bilateral filter handles gaussian noises better, see Figure 3.6. Therefore, they are often used in a complementary manner as shown in Figure 3.6

The parameter d is generally automatically computed from σ_s using the following formula:

$$d = \alpha \times \sigma_s$$

α being a constant. In the OpenCV [13] implementation, $\alpha = 3$.

3.2.2 Morphological operators

Morphological operators are conceived using a different approach than classical ones. They generally view the image as sets. A binary image is viewed as the set of pixels whose value is `True`. A grayscale image is often seen as a collection of binary images: if f is a grayscale image, then it is the collection of binary images produced by $f \leq \alpha$, α being all the possible values that a pixel of f can take. Operators on binary images can therefore be easily extended to grayscale images, provided that they are increasing.

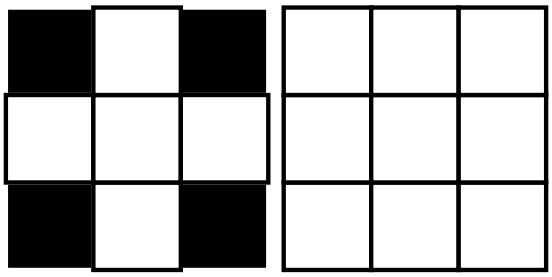
Due to this original approach, notations can be a bit different. We will therefore first present these operators using their original notation. Then, for improving the reader's insight, a simpler but less general notation will often be presented.

Although mathematical morphology tools can process images using an hexagonal grid, we will describe them using the square grid for simplicity's sake. It is important to note that some observations will therefore be only true for the square grid case. However, the general idea of the described operators remain true in the hexagonal grid.

We note S the transformation of the binary image into the set notation in mathematical morphology. We will often use the notion of **structuring element**. A structuring element B is defined as a set of points $S(B) = \{(d_x, d_y)\}$ characterizing an arbitrary shape centered at the origin. On square grids, it can be equivalent to a binary kernel matrix. In that case, $(d_x, d_y) \in S(B)$ is equivalent to $(d_x, d_y)/B(d_x, d_y) = 1$. In the following sections, we will suppose that the structuring elements are symmetric - $B(x, y) = B(-x, -y)$ - as we won't use more complex cases in our thesis.

In the square grid, two structuring elements are mainly used:

- Diamond, also known as N4 since it contains 4 neighboring pixels. See Figure 3.7a.
- Square, also known as N8 since it contains 8 neighboring pixels. See Figure 3.7b.



(a) *Diamond SE* ***(b)*** *Square SE*

Figure 3.7: Square and diamond structuring elements. *SE: Structuring Element.*

Dilation

Let's suppose we have a binary image f of size (w, h). A **dilation** [14] of an image is defined as:

$$S(\delta_B(f)) = \bigcup_{(d_x,d_y)\in S(B)} \{(x + d_x, y + d_y)|(x, y) \in S(f)\}$$

Where B is a structuring element.

An equivalent notation for binary and grayscale images is:

$$\delta_B(f)(x, y) = \max\{f(x - d_x, y - d_y), (d_x, d_y) \in S(B)\}$$

For each pixel of f we compute the maximum value of the neighboring pixels. The structuring element B defines which pixels are considered as neighbors.

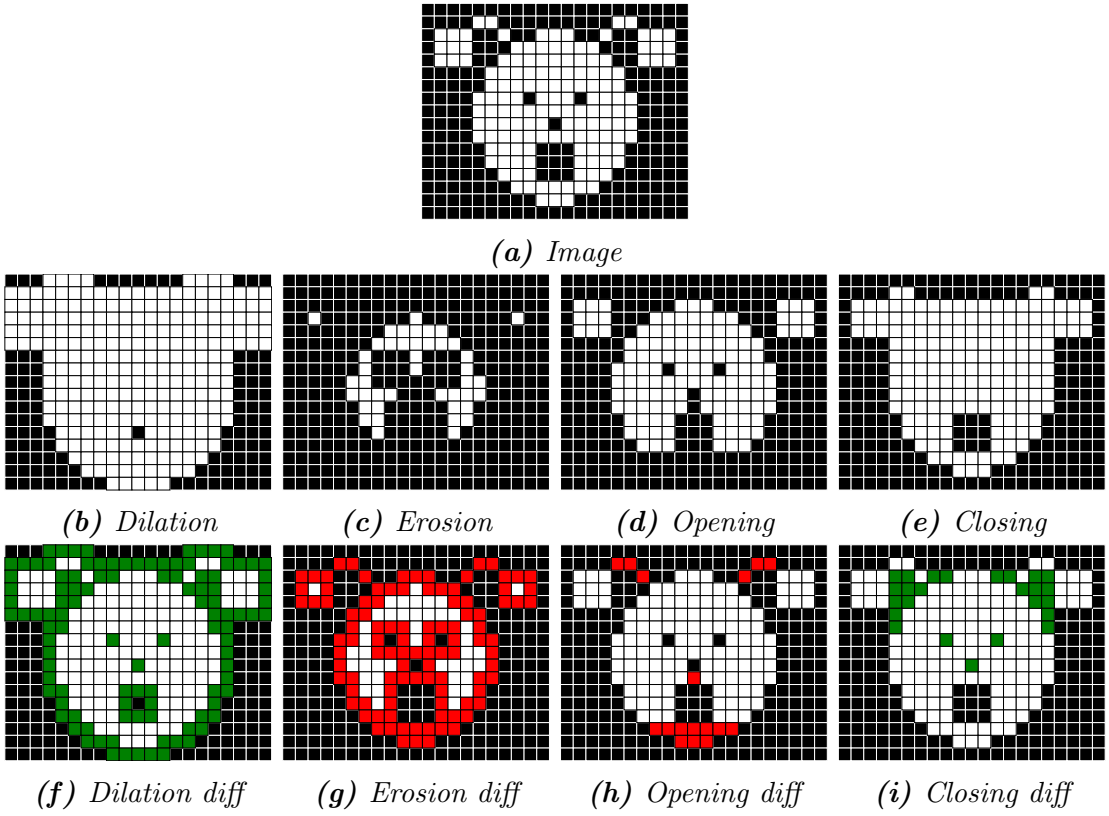
If $S(B)$ contains the center of the structuring element B, the dilation is an extensive operator: $\delta_B(f)(x, y) \geq f(x, y)$.

In this work, a dilation of size α will refer to a sequence of α dilations applied successively:

$$\delta_B^\alpha(f) = \underbrace{\delta_B \circ \delta_B \circ ... \circ \delta_B(f)}_{\alpha \text{ times}}$$

It is important to note that the result of such a sequence is different from a dilation obtained with a bigger structuring element.

The effects of this operator are illustrated in figures 3.8b, 3.8f and 3.9b.



***(a)** Image*

***(b)** Dilation* ***(c)** Erosion* ***(d)** Opening* ***(e)** Closing*

***(f)** Dilation diff* ***(g)** Erosion diff* ***(h)** Opening diff* ***(i)** Closing diff*

Figure 3.8: Comparison of different morphological operations on a binary image. *(b), (c), (d), (e) represent the result of an operation on (a). The structuring element being used is a square matrix of size 3×3 with all values set to 1. (f), (g), (h), (i) represent the difference between (a) and respectively (b), (c), (d), and (e). The green color represent pixels that went from black to white, and the red color pixels that went from white to black.*

Erosion

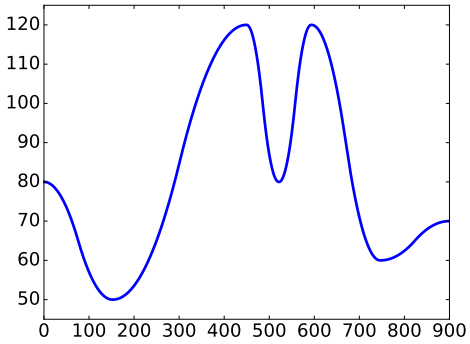
Let's suppose we have a binary image f of size (w, h). An **erosion** [14] of an image is defined as:

$$S(\varepsilon_B(f)) = \bigcap_{(d_x,d_y)\in S(B)} \{(x + d_x, y + d_y) | (x, y) \in S(f)\}$$

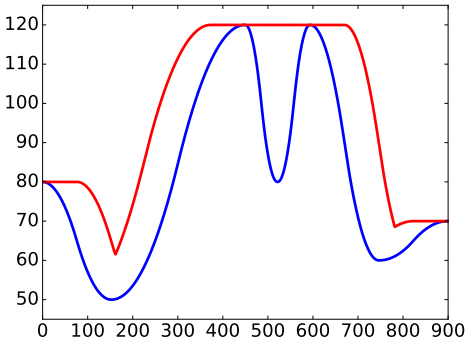
An equivalent notation for binary and grayscale images is:

$$\varepsilon_B(f)(x, y) = \min\{f(x - d_x, y - d_y), (d_x, d_y) \in S(B)\}$$

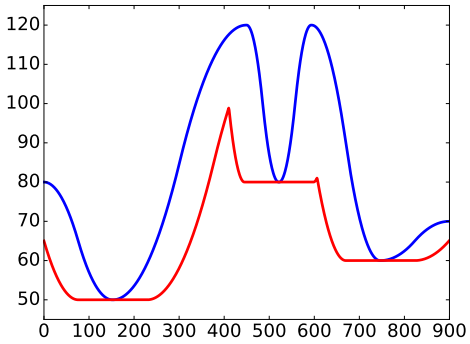
For each pixel of f we compute the minimum value of the neighboring pixels. The structuring element B defines which pixels are considered as neighbors.



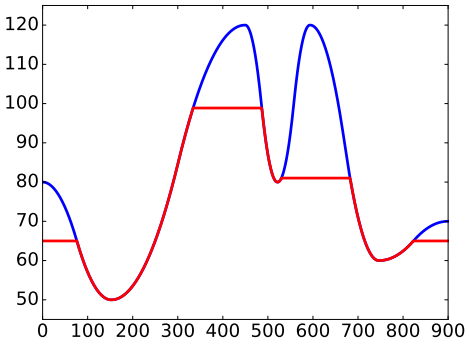
(a) *Slice of image*



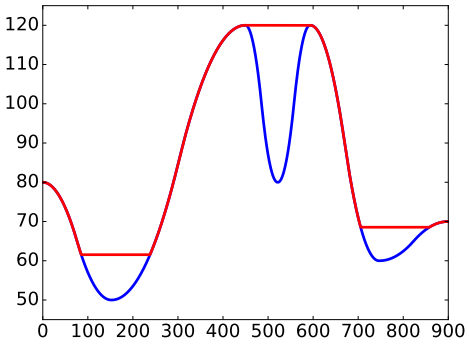
(b) *Dilation*



(c) *Erosion*



(d) *Opening*



(e) *Closing*

Figure 3.9: Comparison of different morphological operations on a slice of a grayscale image. *From (b) to (e), a structuring element of size* 153×153 *has been used.*

If $S(B)$ contains the center of the structuring element B, the erosion is an anti-extensive operator: $\varepsilon_B(f)(x, y) \leq f(x, y)$.

In this work, an erosion of size α will refer to a sequence of α erosions applied successively:

$$\varepsilon_B^\alpha(f) = \underbrace{\varepsilon_B \circ \varepsilon_B \circ ... \circ \varepsilon_B(f)}_{\alpha \text{ times}}$$

As for the dilation, the result of such a sequence is different from an erosion obtained with a bigger structuring element.

The effects of this operation are illustrated in figures 3.8c, 3.8g and 3.9c.

Opening

The opening operator [14] is the composition of the erosion operator and then the dilation operator. The structuring element B defines which pixels are considered as neighbors.

$$\gamma_B(f) = \delta_B \circ \varepsilon_B(f)$$

This is a anti-extensive operator: $\gamma_B(f)(x, y) \leq f(x, y)$.

It is also idempotent: $\gamma_B \circ \gamma_B(f)(x, y) = \gamma_B(f)(x, y)$.

In this work, an opening of size α will refer to:

$$\gamma_B^\alpha(f) = \delta_B^\alpha \circ \varepsilon_B^\alpha(f)$$

The effects of this operation are illustrated in figures 3.8d, 3.8h and 3.9d.

Closing

The closing operator [14] is the composition of the dilation operator and then the erosion operator. The structuring element B defines which pixels are considered as neighbors.

$$\phi_B(f) = \varepsilon_B \circ \delta_B(f)$$

This is an extensive operator: $\phi_B(f)(x, y) \geq f(x, y)$.

It is also idempotent: $\phi_B \circ \phi_B(f)(x, y) = \phi_B(f)(x, y)$.

In this work, a closing of size α will refer to:

$$\phi_B^\alpha(f) = \varepsilon_B^\alpha \circ \delta_B^\alpha(f)$$

The effects of this operation are illustrated in figures 3.8e, 3.8i and 3.9e.

Distance transform

Let's suppose we have a binary image f. The **distance transform** [15] creates a new image with the same size of f but containing integer values. The value of each pixel will equal the distance to the nearest pixel with value set to `False` in f.

A way to obtain such a result using morphological operators is described in Algorithm 2. However, more popular and efficient algorithms exist: for instance, by using directional scans of the image [16] or a priority pixel queue [17].

The effects of the distance transform are shown in Figure 3.10.

Towards geodesic operators

If we want to measure the distance a person will have to walk to go through a twisted corridor, we can't use standard distance metrics. Indeed, the walked distance will be higher than the distance between the start point and the end point of the corridor because of the numerous turns inside it. We call geodesic distance the evaluation of the walked distance. See Figure 3.11.

The geodesic operators we will now describe rely on this notion of geodesic distance.

Geodesic dilation

A geodesic dilation [14] can be viewed as a constrained dilation. Here, we have an image f of size (w, h) and an image $m \geq f$ of same size (w, h), generally called **mask**. A geodesic dilation is defined as:

$$D_g(f) = \delta_B(f) \cap m$$

An equivalent notation is:

$$D_g(f)(x, y) = \min(\delta_B(f)(x, y), m(x, y))$$

Where B is a structuring element. The used structuring element is generally not stated as it is often an isotropic elementary structuring element (Square or Diamond in the square grid).

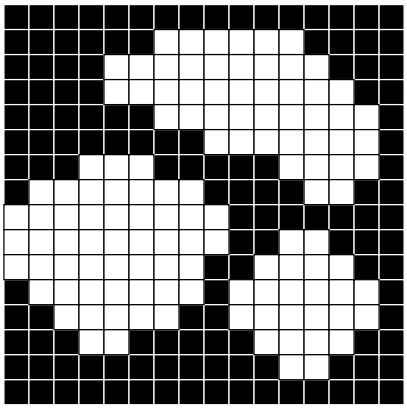
A geodesic dilation of size α is obtained by applying successively elementary geodesic dilation α times.

$$D_g^\alpha(f) = \underbrace{D_g(f) \circ D_g(f) \circ ... \circ D_g(f)}_{\alpha \text{ times}}$$

Algorithm 2: Distance transform

Function *distanceTransform(f, B)*

```
/* f is the binary image, B is an elementary structuring
   element.                                                    */
size ← size of f ;
d ← new image with size size ;          // Result of distance transform
temp ← f ;
i ← 1;
while temp has at least one pixel set to True do
    Set d to i where temp is True;
    temp ← ε_B(temp);
    i ← i + 1;
return d ;
```



(a) *Image*

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	2	2	2	2	1	1	0	0	0
0	0	0	0	1	1	1	2	2	2	2	2	1	1	0	0
0	0	0	0	0	0	1	1	1	2	2	2	2	1	1	0
0	0	0	0	0	0	0	0	0	1	1	1	1	2	2	1	0
0	0	0	1	1	1	0	0	0	0	0	1	1	1	1	0
0	1	1	1	2	1	1	1	0	0	0	0	1	1	0	0
1	1	2	2	2	2	2	1	1	0	0	0	0	0	0	0
2	2	2	3	3	3	2	1	1	0	0	1	1	0	0	0
1	1	2	2	3	2	2	1	0	0	1	1	1	1	0	0
0	1	1	2	2	2	1	1	0	1	1	2	2	1	1	0
0	0	1	1	1	1	1	0	0	1	1	2	2	1	1	0
0	0	0	1	1	0	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b) *Distance transform (Square SE)*

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	2	2	2	2	2	1	0	0	0	0
0	0	0	0	1	1	2	2	3	3	3	2	1	0	0	0
0	0	0	0	0	0	1	1	2	2	2	3	3	2	1	0
0	0	0	0	0	0	0	0	1	1	1	2	3	2	1	0
0	0	0	1	1	1	0	0	0	0	0	1	2	2	1	0
0	1	1	2	2	2	1	1	0	0	0	0	1	1	0	0
1	2	2	3	3	3	2	2	1	0	0	0	0	0	0	0
2	3	3	4	4	4	3	2	1	0	0	1	1	0	0	0
1	2	3	4	4	3	2	1	0	0	1	2	2	1	0	0
0	1	2	3	3	2	2	1	0	1	2	3	3	2	1	0
0	0	1	2	2	1	1	0	0	1	2	3	3	2	1	0
0	0	0	1	1	0	0	0	0	0	1	2	2	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(c) *Distance transform (Diamond SE)*

Figure 3.10: Distance transforms using the Square and Diamond structuring elements. *SE: Structuring Element.*

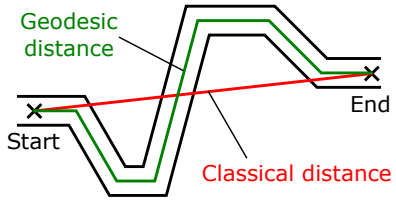


Figure 3.11: Difference between the classical distance and the geodesic distance.

The effects of geodesic dilation are shown in figures 3.12d and 3.13.

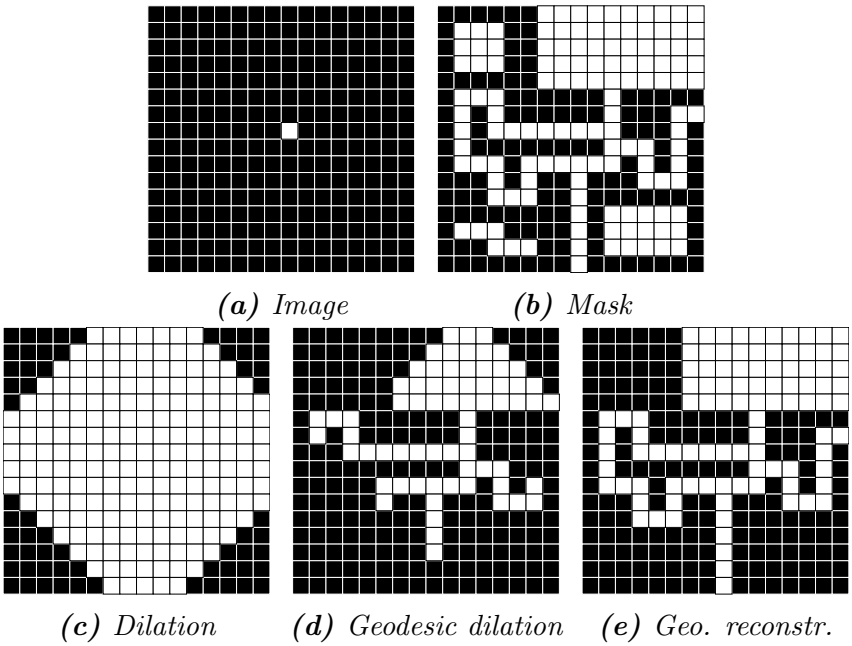


Figure 3.12: Comparison of dilation, geodesic dilation and geodesic reconstruction. *In (c) and (d), the size of the dilation and the geodesic dilation is 10. Geo. reconstr.: geodesic reconstruction.*

Geodesic reconstruction

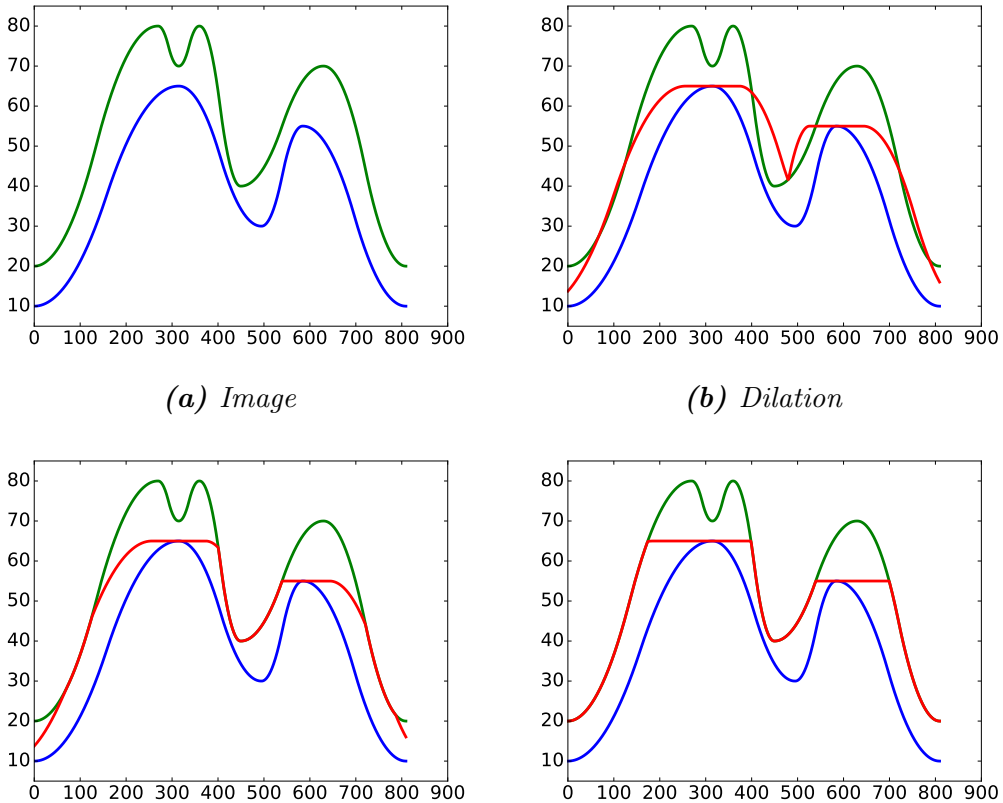
Geodesic reconstruction [14] is the result of geodesic dilation when its size tends to infinity:

$$R_g(f) = D_g^{+\infty}(f)$$

Concretely, it is the geodesic dilation of size α^* such as additional geodesic dilations wouldn't change results (idempotence).

$$R_g(f) = D^{\alpha^*}(f)/D^{\alpha^*}(f) = D^{\alpha^*+1}(f)$$

The effects of geodesic dilation are shown in figures 3.12e and 3.13.



(a) *Image* ***(b)*** *Dilation*

(c) *Geodesic dilation* ***(d)*** *Geodesic reconstruction*

Figure 3.13: Comparison of dilation, geodesic dilation and geodesic reconstruction on a slice of a grayscale images. *Blue: original image f. Green: mask g. Red: results.*

Geodesic erosion and dual geodesic reconstruction

The same principle than the geodesic dilation is applied to the geodesic erosion [14]. We have an image f of size (w, h) and a mask $m \leq f$ of size (w, h). A geodesic erosion is defined as:

$$E_g(f) = \varepsilon_B(f) \cup m$$

An equivalent notation is:

$$E_g(f)(x, y) = \max(\varepsilon_B(f)(x, y), m(x, y))$$

A geodesic erosion of size α is obtained by applying successively elementary geodesic erosion α times.

$$E_g^{\alpha}(f) = \underbrace{E_g(f) \circ E_g(f) \circ ... \circ E_g(f)}_{\alpha \text{ times}}$$

The dual geodesic reconstruction [14] is the result of geodesic erosion when its size tends to infinity:

$$R_g^*(f) = E_g^{+\infty}(f)$$

There exists a duality relationship with geodesic reconstruction:

$$R_g^*(f) = -R_{-g}(-f)$$

Strong leveling (sequential alternate leveling)

The geodesic reconstruction and dual reconstruction can be used to remove noise and simplify an image. If we consider an image f, the sequential alternate leveling, or **strong leveling** [18, 19], is defined in Algorithm 3.

Algorithm 3: Strong leveling

Function *strongLeveling(*f*,* λ*)*

```
/* f is a grayscale image, λ is the strength of the filter. For
   convenience, R_f(g) is written as R(f,g) and R*_f(g) is written
   as R*(f,g). δ_i is a dilation of size i and ε_i is an erosion of
   size i.                                                        */
res ← f ;
for i ∈ [1, 2, ..., λ] do
    dilated ← δ_i(f) ;
    eroded ← ε_i(f) ;
    reconstructed ← R*(res, dilated) ;
    res ← R(reconstructed, eroded) ;
return res ;
```

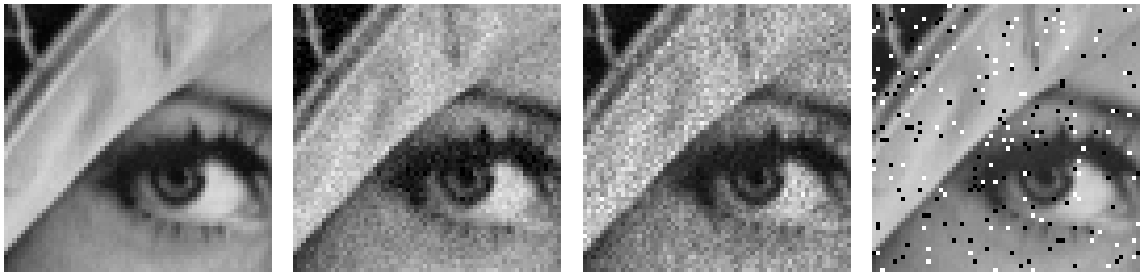
The effects of this filter are illustrated in Figure 3.14.

3.2.3 Edge handling

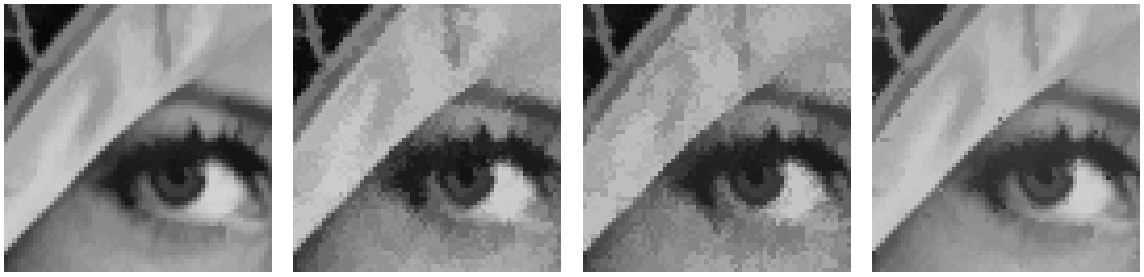
For producing a new image, most operators rely on the neighbors of each pixel of the original image. However, the needed neighbors are not always available for pixels near the image's borders. There are several ways to handle this issue.

A first way is to not process those pixels. It is not always convenient, as the size of the resulting image would then be smaller than the input image.

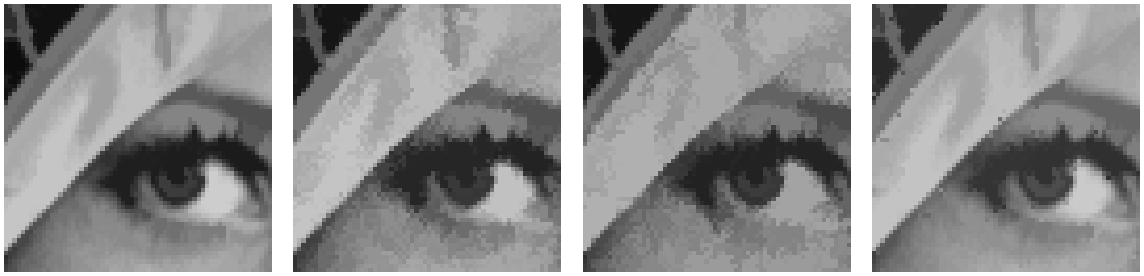
A second way consists of simulating values for pixels outside the image. We will denote $O = x < 1 \cup x > w \cup y < 1 \cup y > h$, the set of pixels outside the image.



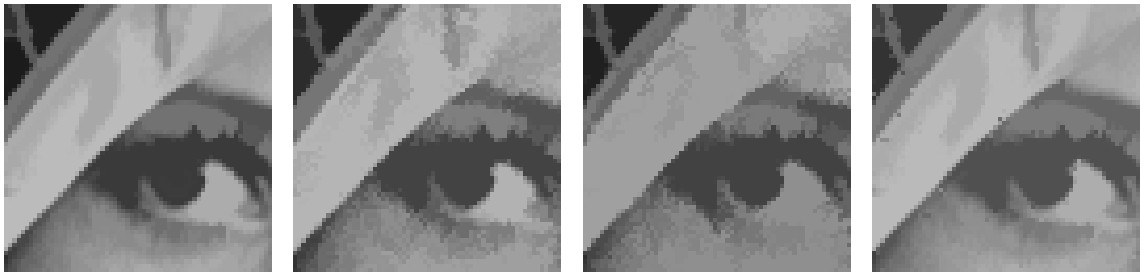
(a) *Image* ***(b)*** *(a)+Gauss. noise* ***(c)*** *(a)+speckle noise* ***(d)*** *(a)+S/P noise*



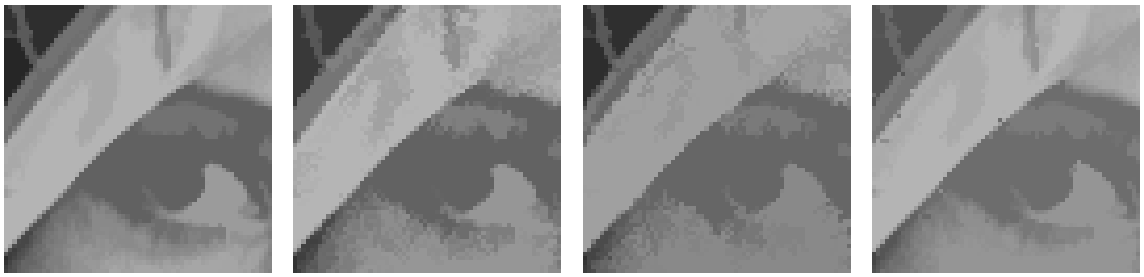
(e) *SL size 1 on (a)* ***(f)*** *SL size 1 on (b)* ***(g)*** *SL size 1 on (c)* ***(h)*** *SL size 1 on (d)*



(i) *SL size 2 on (a)* ***(j)*** *SL size 2 on (b)* ***(k)*** *SL size 2 on (c)* ***(l)*** *SL size 2 on (d)*



(m) *SL size 4 on (a)* ***(n)*** *SL size 4 on (b)* ***(o)*** *SL size 4 on (c)* ***(p)*** *SL size 4 on (d)*



(q) *SL size 8 on (a)* ***(r)*** *SL size 8 on (b)* ***(s)*** *SL size 8 on (c)* ***(t)*** *SL size 8 on (d)*

Figure 3.14: Results of the Strong Leveling filter using different sizes. *SL: Strong Leveling. Gauss: Gaussian.*

The simplest solution is to consider that $f(x, y) = 0$ for $(x, y) \in O$. This solution has however a major drawback: near borders, values of the resulting image are lower than they should be.

Common alternatives are to consider that $f(x, y) = f(N_d(x, y))$ or $f(x, y) = f(R_d(x, y))$ for $(x, y) \in O$. N_d is the nearest defined pixel to (x, y). R_d will consider that the image reflects itself: see Algorithm 4. Because of its low complexity, this last alternative is the one we will use by default for classical operators.

Algorithm 4: If necessary, returns the reflected position of (x, y).

Function *computeReflectedPosition(x, y, w, h)*

```
/* (x,y) is the position of the pixel, w and h are the width and
   height of the image.  We suppose that −w < x < 2w and
   −h < y < 2h.                                                  */
x′ ← x ;
y′ ← y ;
if x < 1 then
    x′ ← 1 − x ;
if y < 1 then
    y′ ← 1 − y ;
if x > w then
    x′ ← 2w − x ;
if y > h then
    y′ ← 2h − y ;
return x′, y′ ;
```

For morphological operators, the solution is to consider them as geodesic operators, with the mask being the whole image. Instead of guessing the neighbors' values, the operators will adapt their structuring element so that it doesn't overflow the original image.

3.3 Segmentation tools

The methods we developed for attaining our thesis' objective need to have a way to separate an image into regions, sub-regions, sub-sub-regions and so on. This process is called **hierarchical segmentation**. In this section, we will describe different algorithms allowing to obtain a hierarchical segmentation of an image.

In mathematical morphology, the most common pipeline used for obtaining a hierarchical segmentation of an image is shown in Figure 3.15. There are multiple ways for evaluating each step. The core component is the **watershed** algorithm [14] [20], where the image is split into regions, and the other components are conceived by taking into account the inner workings of this algorithm. Therefore, though it is not the first step of the pipeline, we will first describe the watershed algorithm in order to give the reader a better understanding of the other components.

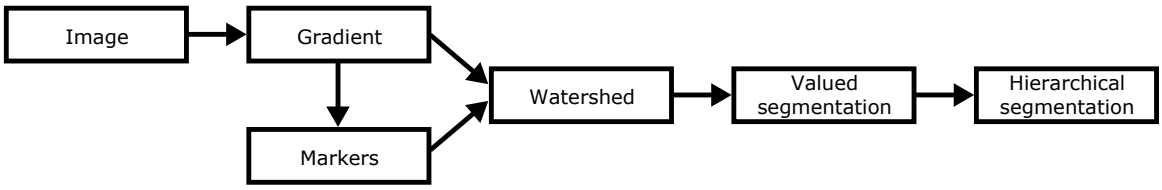


Figure 3.15: *General pipeline of the hierarchical segmentation*

3.3.1 The watershed algorithm

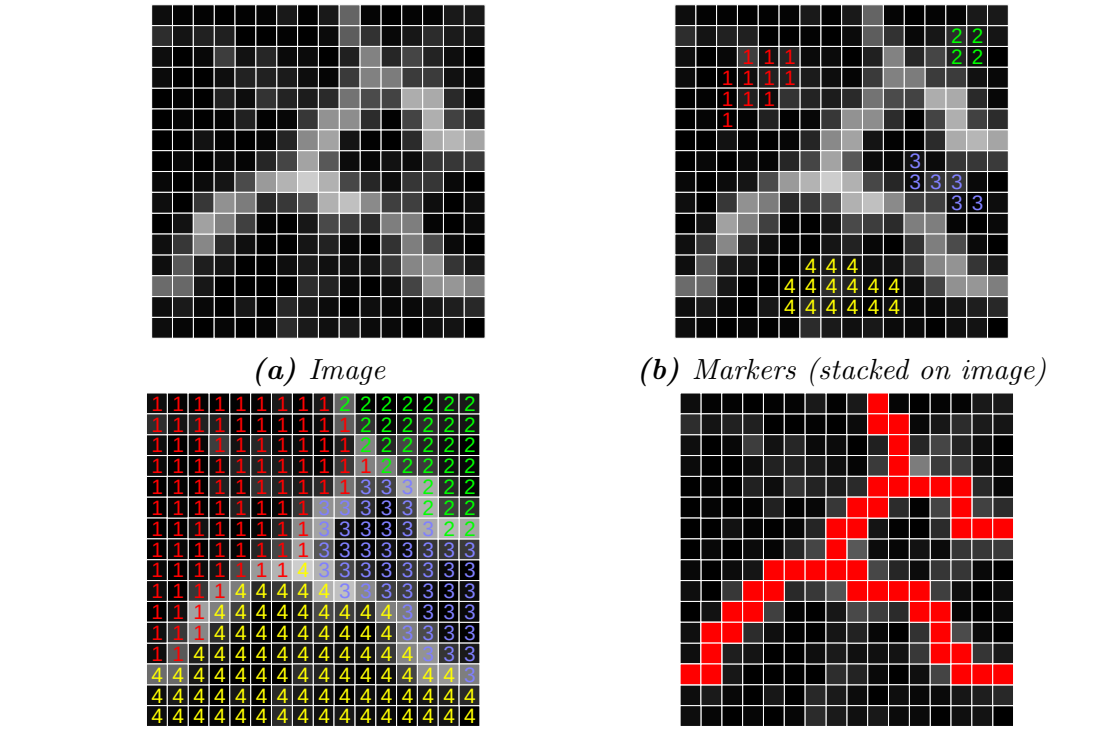
The watershed algorithm divides a grayscale image into regions. We will suppose that, through a previous operation, we have obtained so-called **markers** (see Figure 3.16b). Each marker is labeled by a unique identifier and is a set of points indicating where a region most probably lies. Markers however are not defined on the whole image, and we want to allocate a marker to each pixel of the image. Therefore, we need a way to propagate these markers by taking into account the information on the image (see Figure 3.16c).

The general idea of the watershed algorithm is shown in Figure 3.17. The watershed algorithm sees the grayscale image as a topographic map (Figure 3.17a), and markers (Figure 3.17b) are seen as water sources. The relief is progressively flooded, creating larger and larger lakes.

At each step of the flooding, for each lake, there are three possible choices. If the lake contains only one marker, the marker will be assigned to all the points in the lake (Figure 3.17d). If the lake contains two or more markers, the points not yet assigned are allocated to their nearest marker (Figure 3.17e). If the lake contains no marker at all, then the points in the lake stay unassigned (Figure 3.17f); they will be assigned at a later step of the flooding (Figure 3.17g). The segmentation is the resulting markers when all the relief is flooded (Figure 3.17i). The algorithm can also represent the segmentation contours by displaying intersections between markers (Figure 3.16d).

The general idea is concretized in the simple but inefficient Algorithm 5. Various implementations and variant of this algorithm exist. Among others, [21] uses graphs and [22] uses priority queues to improve the algorithm's efficiency.

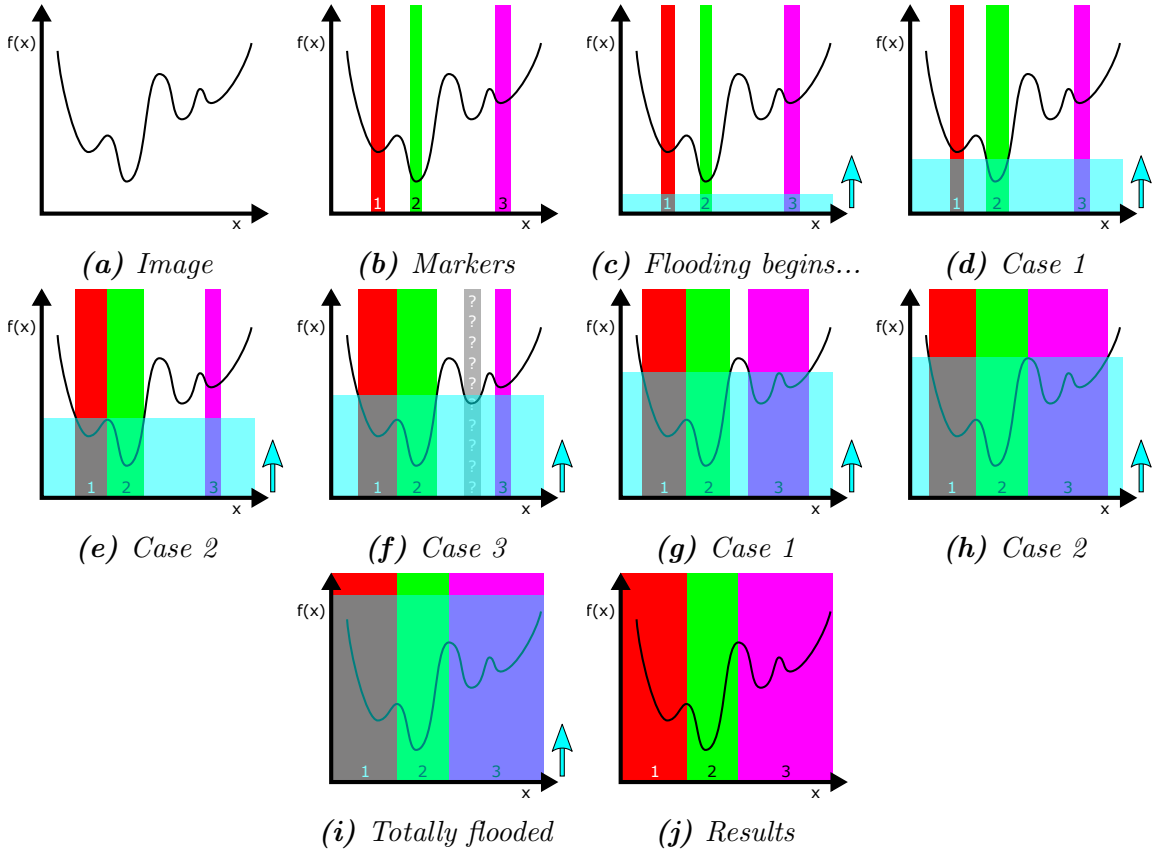
As explained in this section, the watershed algorithm needs two inputs: an image and a marker matrix. However, it is not appropriate to directly provide the original image to the watershed algorithm. Indeed, as it operates, the algorithm supposes that the brighter a point is, the higher is the likelihood that this point separates two or more regions. This isn't true for a regular image as there might be bright and dark objects. We need therefore an operator that transforms an original image into an appropriate image for the watershed algorithm. This is the objective of the gradient operators, and we will describe some methods in the following section.



(a) Image *(b) Markers (stacked on image)*

(c) Objective: markers after watershed *(d) Additional result: watershed lines*

Figure 3.16: Markers propagation using the watershed algorithm.



(a) Image *(b) Markers* *(c) Flooding begins...* *(d) Case 1*

(e) Case 2 *(f) Case 3* *(g) Case 1* *(h) Case 2*

(i) Totally flooded *(j) Results*

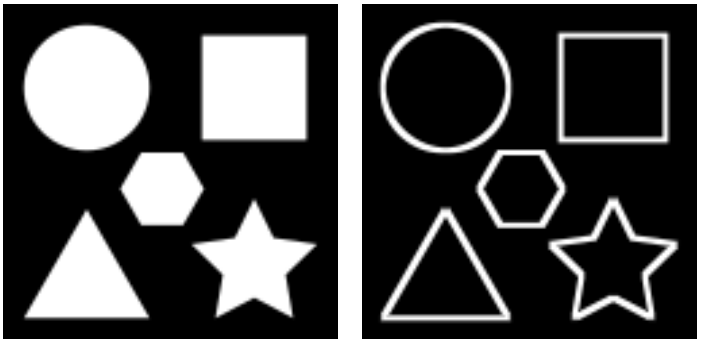
Figure 3.17: Watershed's general idea. *Case 1: the lake contains only one marker → all the points in the lake are allocated to this marker. Case 2: the lake contains two markers or more → unassigned points are affected to the nearest marker. Case 3: the lake contains no marker → the lake stays unassigned.*

Algorithm 5: The watershed algorithm

```
Function watershed(f, markers)
    /* f is a grayscale image, markers is a matrix containing the
       markers. There is no marker when markers = 0.                */
    imMax ← max(f) ;
    propagatedMarkers ← markers ;
    for i ∈ [1, 2, ..., imMax] do
        /* Flooding process, we compute higher and higher threshold of
           the image and we process the resulting connected
           components: our lakes.                                    */
        threshold ← f ≤ i ;
        lakes ← get the list of connected components in threshold ;
        foreach lake ∈ lakes do
            lakeMarkers ← get list of propagatedMarkers under lake ;
            if length(lakeMarkers) = 0 then
                /* If there is no marker under lake, then leave the lake
                   unassigned. It will be assigned in a later stage.
                   */
            if length(lakeMarkers) = 1 then
                /* If there is only one marker under lake, then we
                   associate the marker to the whole lake.           */
                Set propagatedMarkers to lakeMarkers[0] where lake ;
            if length(lakeMarkers) > 1 then
                /* If there are two markers or more under lake, we need
                   to determine the intersection of the two lakes.   */
                points ← Get all the points in lake where propagatedMarkers = 0
                 ;
                foreach point ∈ points do
                    nearestMarker ← the closest marker in lakeMarkers to point
                     ;
                    Set propagatedMarkers[point] to nearestMarker ;
    return propagatedMarkers ;
```

3.3.2 Gradient

The aim of the gradient operators is to detect edges in the images: the brighter a pixel is, the more likely it is an edge. An illustration of what is expected of a gradient is shown in Figure 3.18.



(a) *Image* ***(b)*** *Gradient of image (a)*

Figure 3.18: Illustration of gradient's result.

There are many ways to compute such a gradient. As it is only a single step of our hierarchical segmentation pipeline, we needed it to be computationally efficient. For instance, gradients such as described in [23] provide interesting results, but take too much time to be computed for our purpose.

Most gradient algorithms handle grayscale images. For color images, the standard approach is to separate it into different channels (red, green, blue), compute the gradient on each channel, and then compute the maximum of all the gradients. See Figure 3.19. There has been some research on how to improve this approach: [24] combines the gradient of alternative channels (hue, luminance, saturation), [25] computes a gradient based on perceptual color differences. However, as our thesis' main objective is to process SEM images, potential channels (if we use multiple sensors) don't have the same properties as classical color images. We therefore kept the standard approach.

Morphological gradient

For computing the gradient of a grayscale image, the morphological gradient operator [14] is a standard approach. It consists of the difference between the dilation operator and the erosion operator. The structuring element B define which pixels are considered as neighbors.

$$G_B(f) = \delta_B(f) - \varepsilon_B(f)$$

The effects of this operation are illustrated in figures 3.20b and 3.21.

(a) *Image*

(b) *Red channel*

(c) *Green channel*

(d) *Blue channel*

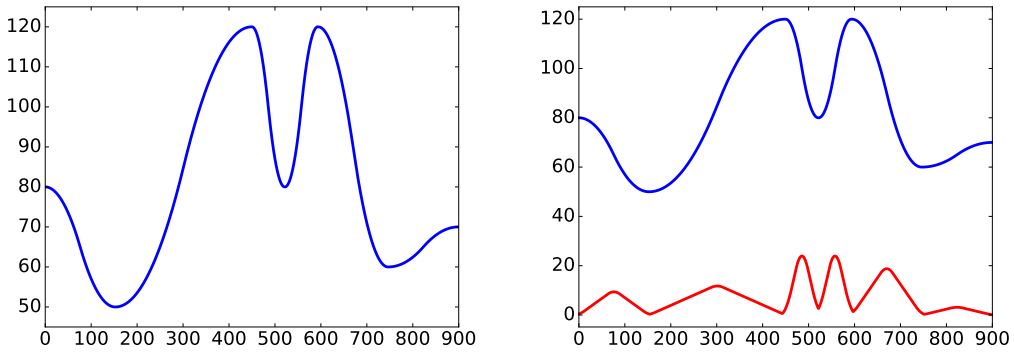
(e) *Red channel's gradient*

(f) *Green channel's gradient*

(g) *Blue channel's gradient*

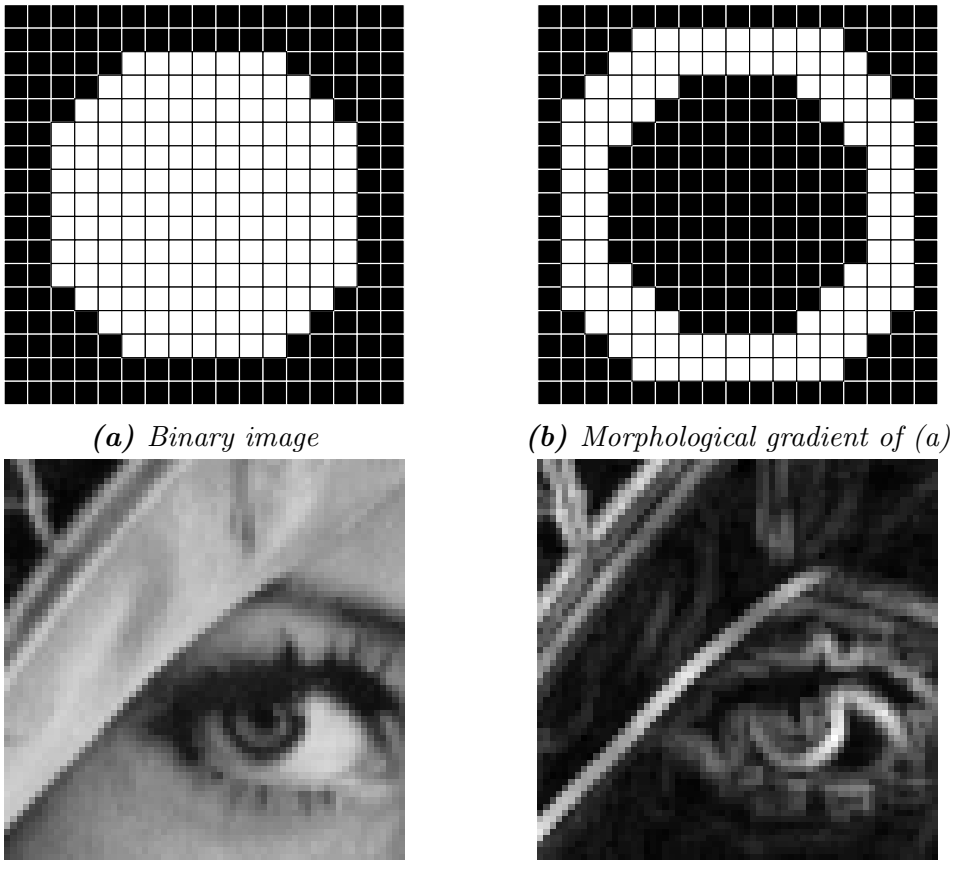
(h) *Maximum of (e) (f) (g)*

Figure 3.19: Gradient on color images.



(a) *Slice of image* ***(b)*** *Gradient*

Figure 3.20: The morphological gradient on a slice of a grayscale image. *A structuring element of size* 27×27 *has been used.*



(a) *Binary image* ***(b)*** *Morphological gradient of (a)*

(c) *Lenna image* ***(d)*** *Morphological gradient of (c)*

Figure 3.21: Illustrations of morphological gradient. *The structuring element being used is a square matrix of size* 3×3 *with all values set to 1.*

Multiscale morphological gradient

The morphological gradient detects most edges in the image, but it can miss progressive and / or faint contours. See figures 3.22a and 3.22b.

Those failures can be detrimental for the segmentation process, as it is likely that the watershed algorithm will merge regions that are not separated by a strong enough gradient. A first solution is to compute a morphological gradient of a larger size. Though progressive contours are more likely to be detected, near edges might be merged because of the **thickness of the gradient**.

The multiscale morphological gradient [26] attempts to solve this problem. It combines multiple gradients computed from the scale i to the scale j by computing their maximum:

$$\tilde{G}_i^j = \max(\tilde{G}_i, \tilde{G}_{i+1}, ..., \tilde{G}_j)$$

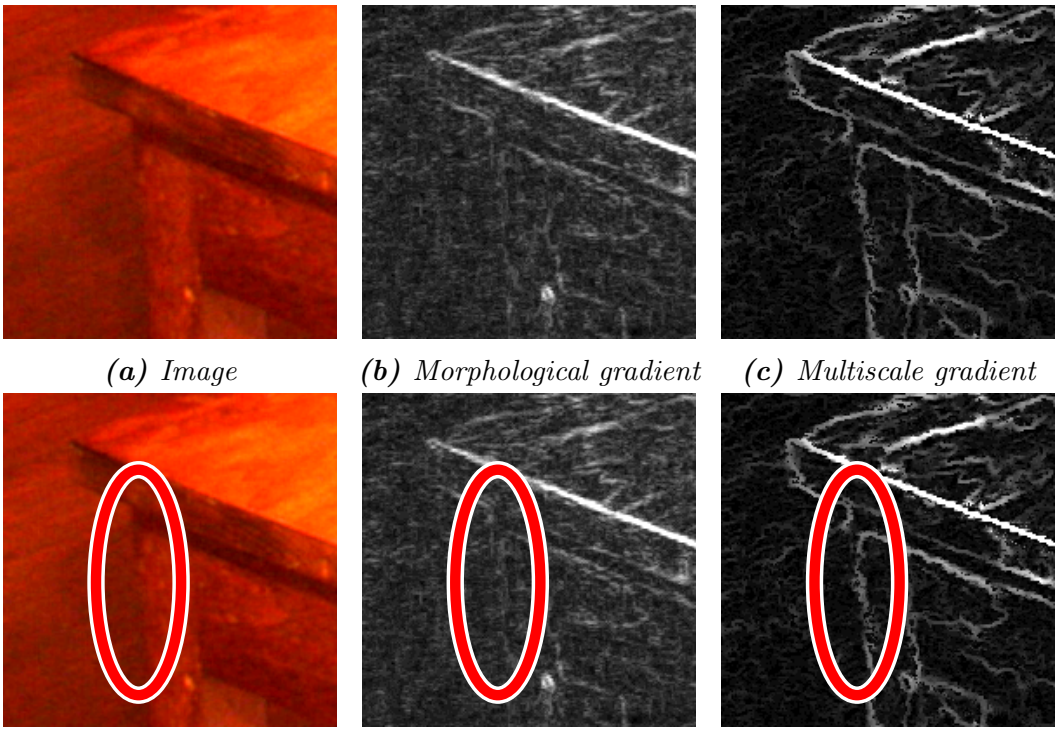
For each scale λ, the image is first filtered using the strong leveling operator of size λ. A morphological gradient of size λ is then computed on this filtered image. Borders with a large thickness are then discarded by using the opening operator and the gradient is thinned using a mask that detects transition points. The process is more thoroughly described in Algorithm 6.

The effects of the multiscale morphological gradient are illustrated in Figure 3.22c.

Algorithm 6: Multiscale gradient

Function *MultiscaleGradientAtScale(f, λ)*
- `/* f is a grayscale image, λ is the size of the gradient. */`
- $f' \leftarrow$ strong leveling of f of size λ ;
- $\tilde{G}_\lambda \leftarrow$ morphological gradient of f' of size λ ;
- $\tilde{G}_\lambda \leftarrow \tilde{G}_\lambda - \gamma_{2\lambda-1}(\tilde{G}_\lambda)$; `/* Remove contours of thickness ≥ 2λ − 1 with the opening */`
- $M_\lambda \leftarrow$ GetTransitionMaskAtScale(f', λ) ;
- Set $\tilde{G}_\lambda$ to 0 when M_λ is `False` ;
- **return** $\tilde{G}_\lambda$;

Function *GetTransitionMaskAtScale(f, λ)*
- **return** $\varepsilon(f) < \frac{1}{2}(\delta_\lambda(f) + \varepsilon_\lambda(f)) < \delta(f)$;



(a) *Image* ***(b)*** *Morphological gradient* ***(c)*** *Multiscale gradient*

(d) *Image* ***(e)*** *Morphological gradient* ***(f)*** *Multiscale gradient*

Figure 3.22: Comparison between the morphological and the multiscale gradient. *The ellipse in (d), (e) and (f) shows borders that are not detected using the morphological gradient and detected using the multiscale gradient.*

3.3.3 Markers

However, obtaining an adapted gradient is not sufficient for estimating a correct segmentation using the watershed algorithm. Markers also play a major role.

H-minima

A common way to obtain markers is to use the **h-minima** operator [14] on the image's gradient. It retrieves the regional minima of an image. Its only parameter h allows to remove small minima whose depth is less than h: it is especially useful when the image is subject to noise causing small minima to be artificially added.

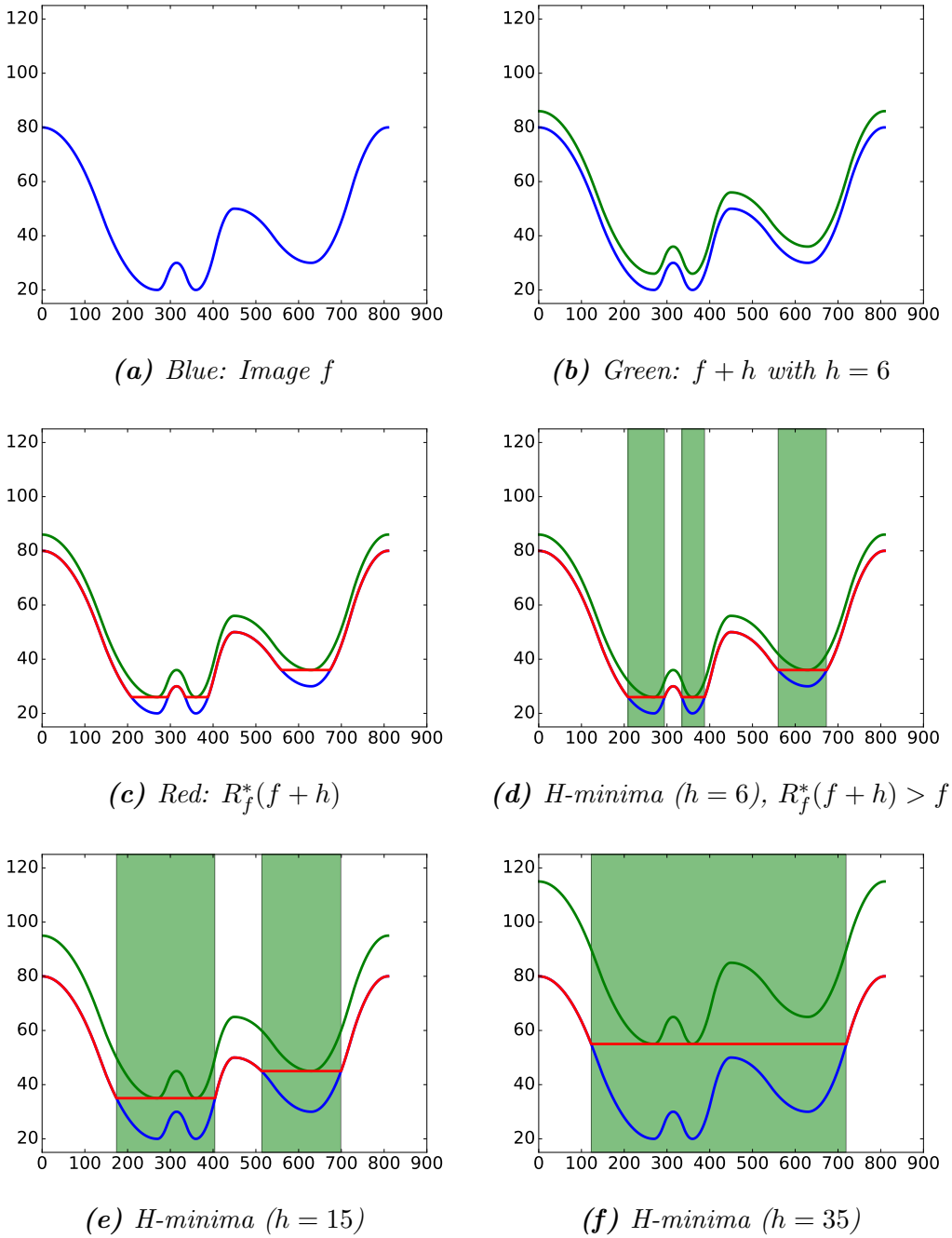
If we have an image f of size (w, h), the h-minima operator is defined as:

$$M_h(f) = R^*_f(f + h) > f$$

$R^*_f(f + h)$ being the dual geodesic reconstruction of $f + h$ by f: as a reminder, it means that we apply successive geodesic erosions of $f + h$ over f until idempotence.

The effects of this operator are shown in Figure 3.23.

Each connected components is then labeled by a unique identifier, producing our markers. See Figure 3.24.



(a) *Blue: Image f*

(b) *Green: $f + h$ with $h = 6$*

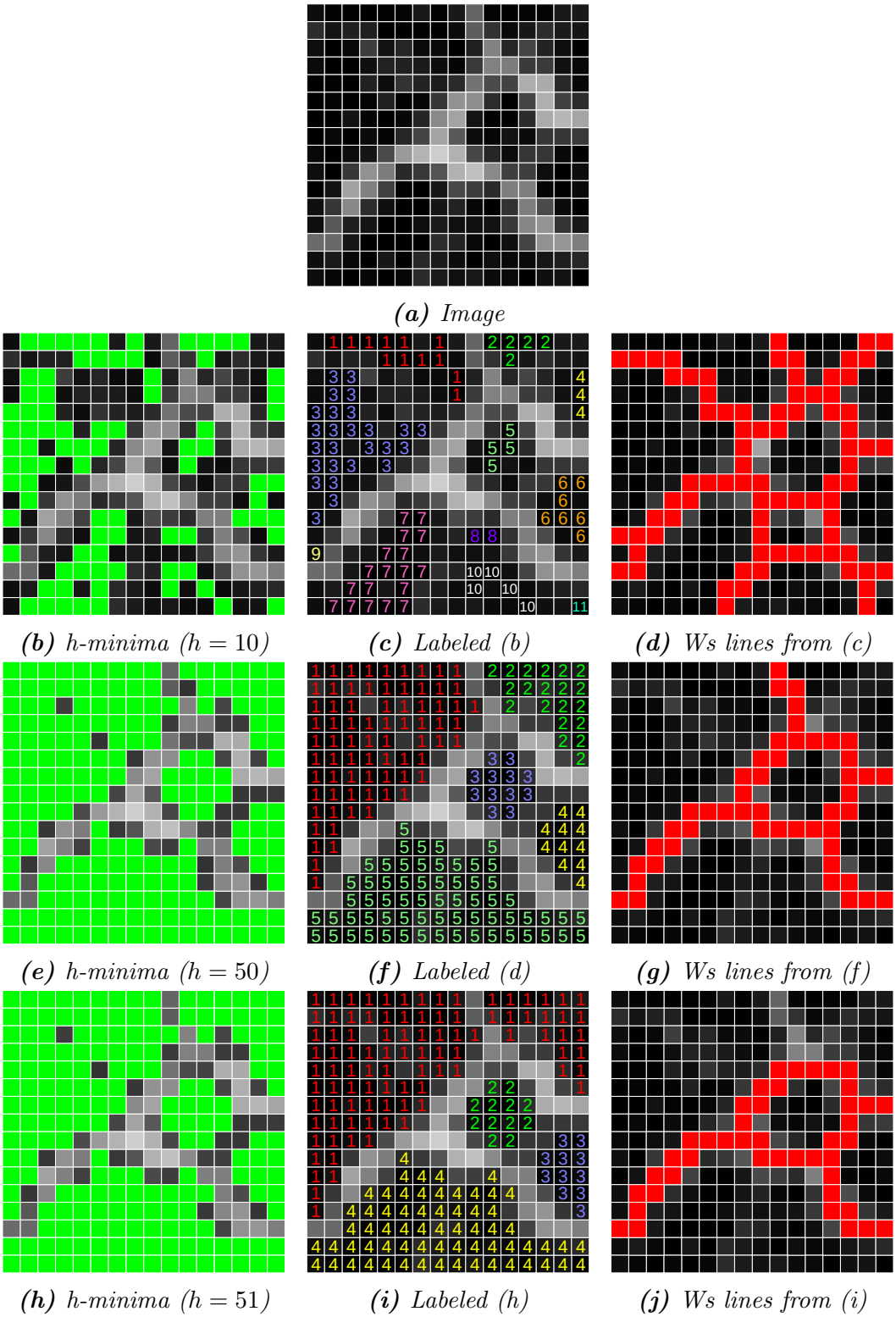
(c) *Red: $R^*_f(f + h)$*

(d) *H-minima ($h = 6$), $R^*_f(f + h) > f$*

(e) *H-minima ($h = 15$)*

(f) *H-minima ($h = 35$)*

Figure 3.23: The h-minima operator.



(a) *Image*

(b) *h-minima* ($h = 10$)

(c) *Labeled (b)*

(d) *Ws lines from (c)*

(e) *h-minima* ($h = 50$)

(f) *Labeled (d)*

(g) *Ws lines from (f)*

(h) *h-minima* ($h = 51$)

(i) *Labeled (h)*

(j) *Ws lines from (i)*

Figure 3.24: Markers produced by the h-minima operator. *If h is too low ($h = 10$ for instance), the watershed over segments the image. However, if h is too high ($h = 51$ for instance), some regions are merged. Ws: Watershed.*

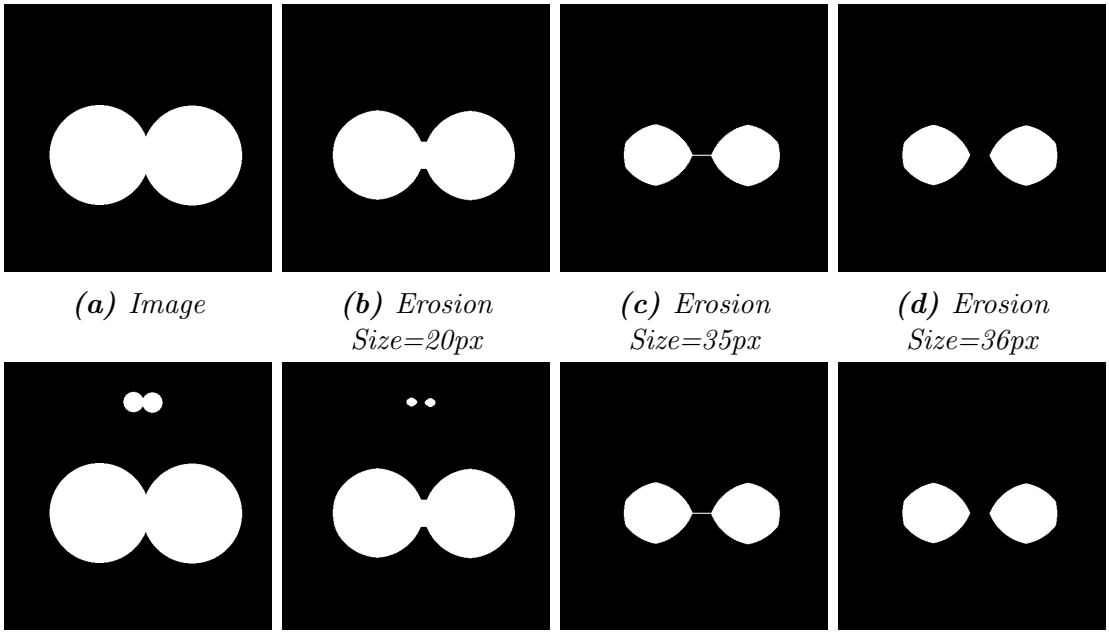
Gradient leakages and the adaptive erosion

One of the drawbacks of directly using h-minima as markers is that, even if there is only one pixel with a low gradient in the border between two regions, they are merged. This is known as a gradient leakage and it is illustrated in Figure 3.24h where the region 1 merges two regions separated by a strong gradient except for one pixel.

If this behavior is expected based on how the h-minima operates, we want to prevent this effect as this low gradient might be the consequence of noise or shadow effects. A common property of these deficient markers is that they get thinner around these leakages. We can take this feature to our advantage.

For illustration purposes, we will rely on simpler examples. In Figure 3.25a, two circles overlap each other. If we want to separate these objects, a first solution is to apply a sufficiently large erosion allowing to separate these objects without erasing them. For our example, the smallest erosion allowing that has a size of 36. See figures 3.25b, 3.25c and 3.25d.

However, this approach doesn't work when the sizes of the objects vary too much. In Figure 3.25e, we have added a similar but much smaller structure than in Figure 3.25a. The erosion that allows to separate the large structure completely erases the small structure. We will have similar problems with our markers as there are big objects (likely to produce large h-minima) and small objects (likely to produce small h-minima). Ideally, we would like to apply a small erosion for the small structure and a large erosion for the large one. See figures 3.25f, 3.25g and 3.25h.



(a) *Image* ***(b)*** *Erosion Size=20px* ***(c)*** *Erosion Size=35px* ***(d)*** *Erosion Size=36px*

(e) *Image* ***(f)*** *Erosion Size=20px* ***(g)*** *Erosion Size=35px* ***(h)*** *Erosion Size=36px*

Figure 3.25: Use of erosion for separating overlapping objects.

Hence the concept of adaptive erosion as described in [27][28][29]. It uses the distance transform of a binary image to erode each object according to its size:

$$\tilde{M}_\alpha(f) = D(f) > R_{D(f)}(\alpha D(f))$$

Where $D(f)$ is the distance transform of f, and α is a parameter between 0 (no erosion) and 1 (full erosion). See Figure 3.26.

The operator is used after computing the h-minima and before labeling the connected components. The effect of this operator is shown in Figure 3.27.

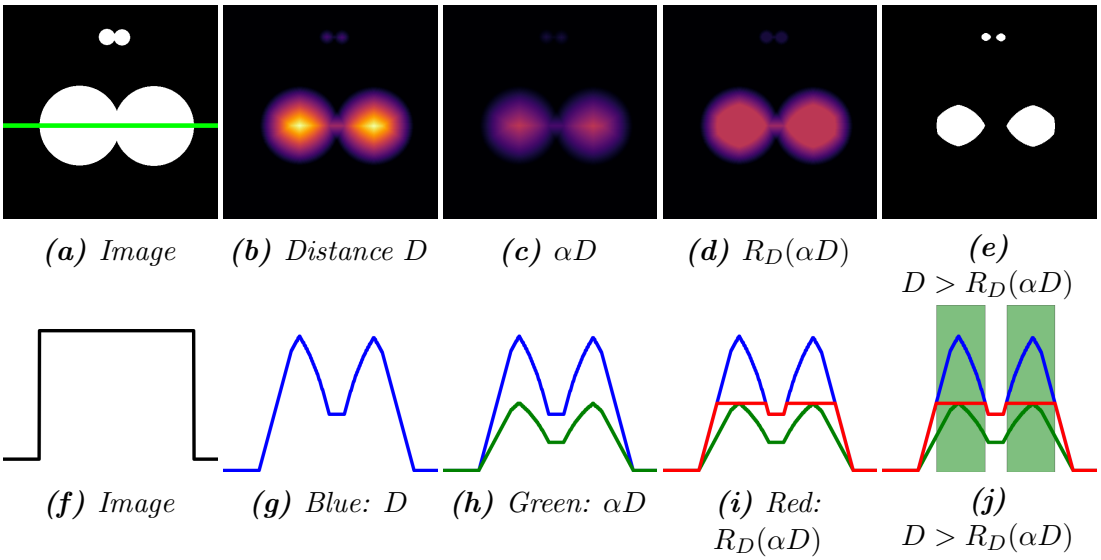
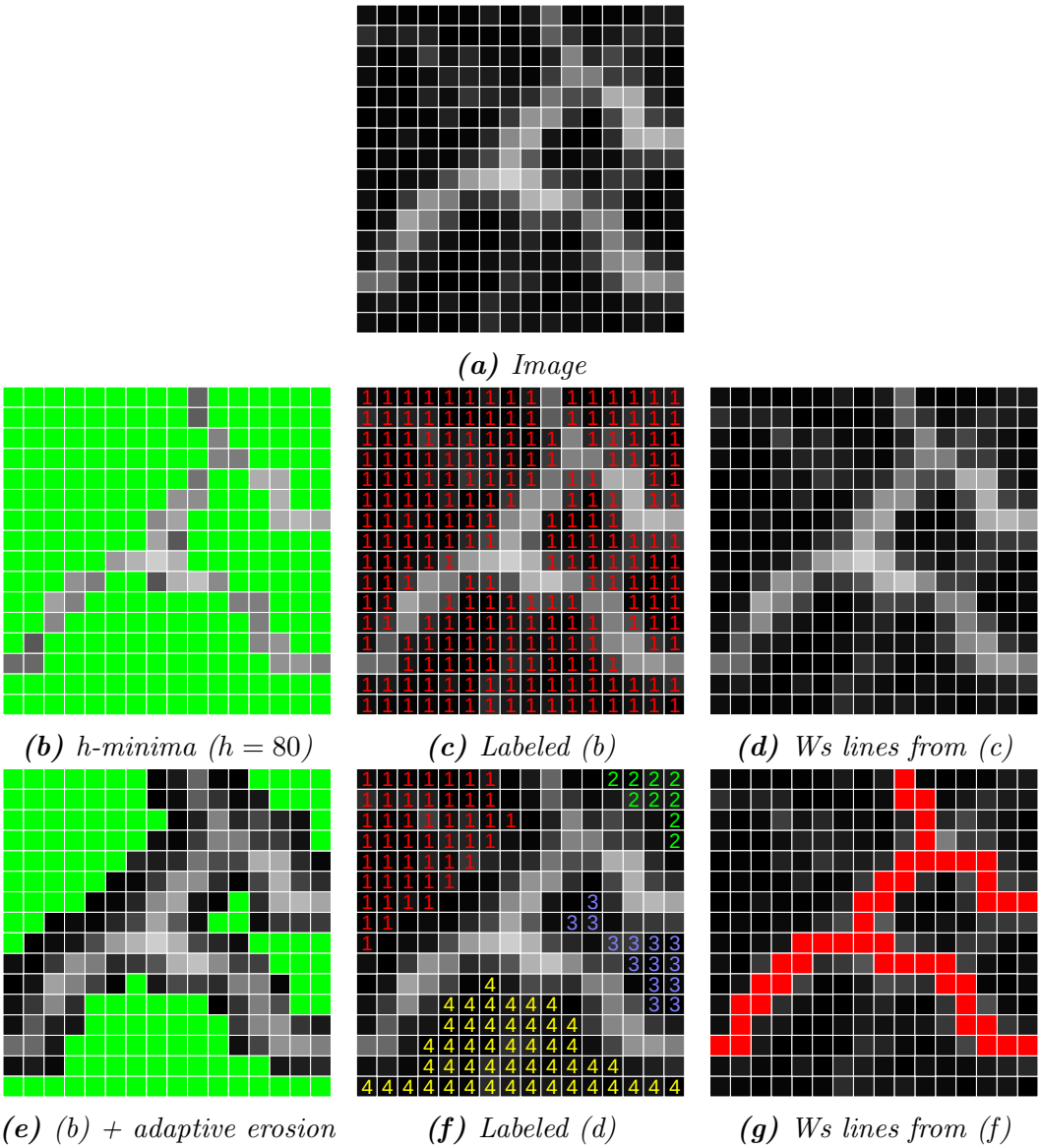


Figure 3.26: The adaptive erosion. *(f), (g), (h), (i) and (j) depict each step on the green line in (a).*



(a) *Image*

(b) *h-minima* ($h = 80$)

(c) *Labeled (b)*

(d) *Ws lines from (c)*

(e) *(b) + adaptive erosion*

(f) *Labeled (d)*

(g) *Ws lines from (f)*

Figure 3.27: Effect of the adaptive erosion. $\alpha = 0.1$. *Segmenting the image using a h-minima with* $h = 80$ *creates a single region because of the gradient leakage, though a separation is visible between markers. Using the adaptive erosion solves the problem. Ws: Watershed.*

3.3.4 Valued segmentation

Even by using advanced tools such as the multiscale gradient, obtaining a perfect segmentation of an image is very challenging. Indeed, there is often no way of obtaining markers that simultaneously avoid over segmentation and merging important regions. That is why the methods we developed use instead hierarchical segmentations where the most important regions are shown first, then sub-regions and so on.

The watershed as it was described in Section 3.3.1 doesn't differentiate most important contours from less important ones. It is important to add this information first to the watershed lines so that hierarchical segmentation algorithms can function properly.

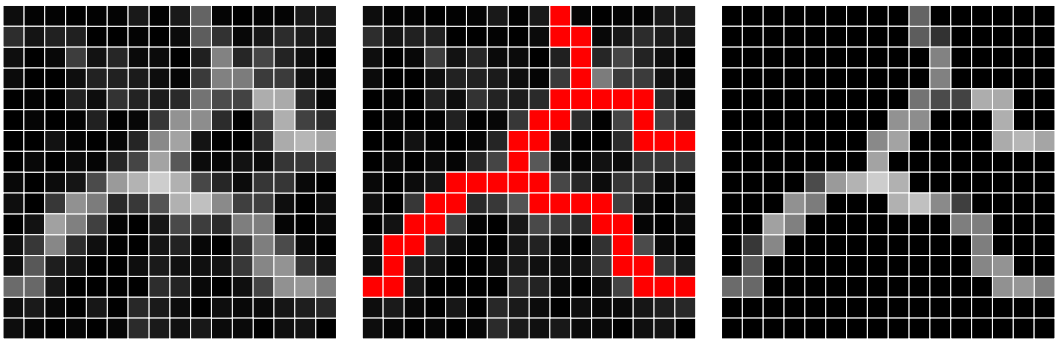
Classical valued watershed

A common approach is to replace the watershed lines by the gradient previously computed. Here is how the valued watershed is computed:

$$W_{valued}(x,y) = \begin{cases} G(x,y), & \text{if W(x,y) = } \texttt{True} \\ 0, & \text{otherwise} \end{cases}$$

With W being the watershed lines image, G being the gradient previously computed.

An illustration is shown in Figure 3.28.

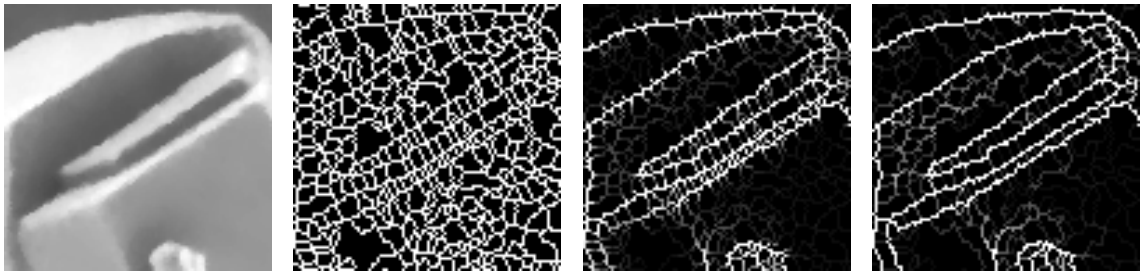


(a) *Image* ***(b)*** *Watershed lines* ***(c)*** *Valued watershed*

Figure 3.28: Classical valued watershed.

Mosaic image and gradient

One of the drawbacks of the classical valued watershed is that thin regions are not well processed. Indeed, they are likely to be divided into different regions, separated by highly valued borders. See Figure 3.29c. This is due to the thickness of the gradient: even with gradients using small structuring elements, values around strong edges are also likely to be high even two or three pixels away from the actual borders. For thin regions, between 1 to 8 pixels wide, it can create problems as the gradient will be high even inside the region. Using instead the mosaic gradient [14] can help attenuate the effect.



(a) *Image* ***(b)*** *Watershed* ***(c)*** *Valued watershed* ***(d)*** *Mosaic gradient*

Figure 3.29: Valued watershed (size 1) vs mosaic gradient.

The mosaic gradient is computed from the mosaic image [14] so we will describe it first. The mosaic image consists of segmenting an original image and then producing a simplified image where each region is displayed by a single color or gray intensity.

The classical way of computing this mosaic image is shown in Algorithm 7.

Algorithm 7: Mosaic image

Function *mosaicImage(f)*

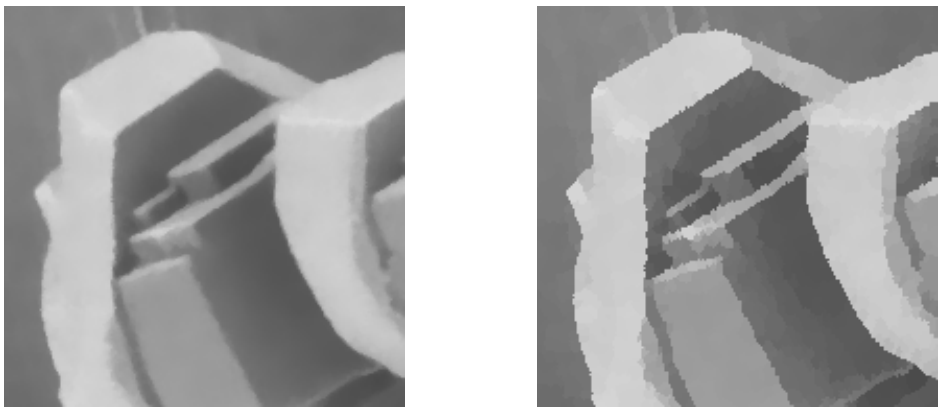
```
/* f is a grayscale image.  Please note there exist a much more
   efficient version.                                          */
```

mosaicImage ← new image of same size of f ;
gradient ← morphological gradient of f of size 1 ;
markers ← h-minima of *gradient* with $h = 1$;
regions ← watershed(*gradient*, *markers*) ;
foreach *region* ∈ *regions* **do**
    marker ← get *region*'s assocated marker ;
    grayValue ← get maximum gray value in f on *marker* ;

```
/* We select here the gray value to be affected to the region.  Since the gradient is probably low on the markers (computed using h-minima...), it isn't likely that there are a lot of variations.  For efficiency reasons, we choose the maximum. */
```

    Set *mosaicImage* to *grayValue* on *region* ;
return *mosaicImage* ;

An illustration of mosaic image is shown in Figure 3.30.



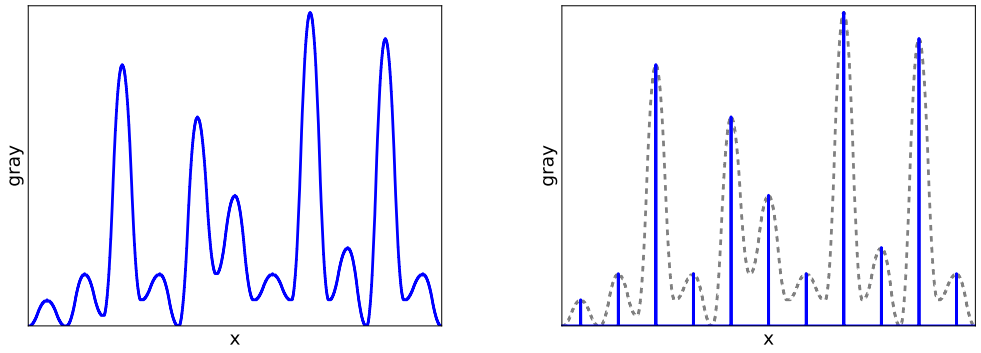
(a) *Image* ***(b)*** *Mosaic image*

Figure 3.30: Illustration of a mosaic image.

The mosaic gradient gets the watershed computed for the mosaic image. The border between each pair of regions A and B is set to $|V(A) - V(B)|$ where $V(x)$ is the gray value of a region x. See example in Figure 3.29d.

3.3.5 Hierarchical segmentation

Once the valued segmentation has been computed, the gradient can be transformed into a simpler image where only segmentation lines appear. See Figure 3.31. The image will often be over segmented, meaning that it will be divided into a lot of non-significant regions. A common property of these regions is that their borders have generally low values because of the low variation in color or gray intensity.



(a) *Gradient* ***(b)*** *Valued segmentation from (a)*

Figure 3.31: Gradient and valued segmentation.

A first solution to handle the problem is to remove low-valued borders. The problem is that this approach is parametric and obtaining a good segmentation that way is generally difficult: there is no threshold that simultaneously removes all non-significant regions and keep all significant ones.

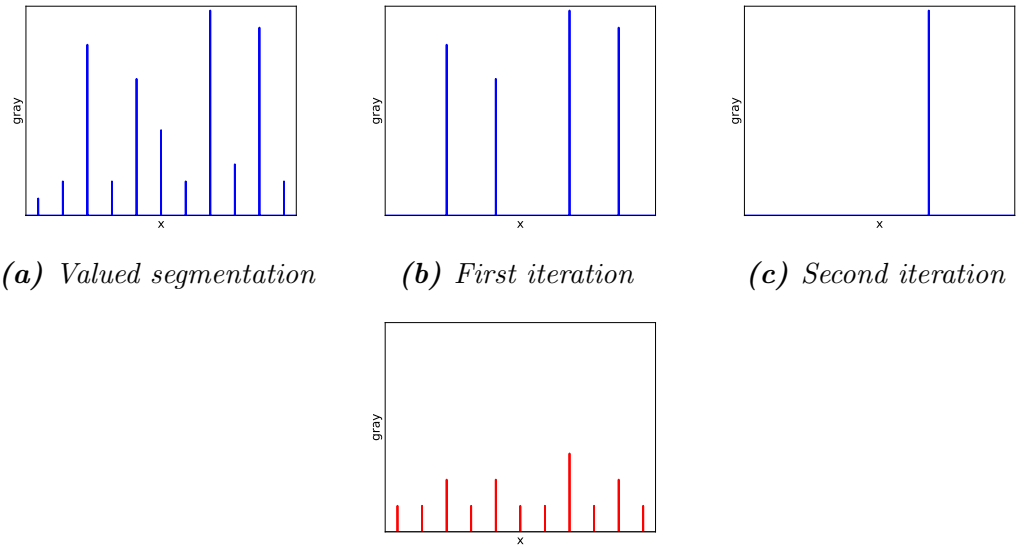
On the other hand, hierarchical segmentation algorithms rank borders from the most probable ones to the least probable ones, not according to their absolute values, but according to their relative values compared to neighboring borders. Furthermore, those algorithms are generally non parametric, enabling their users to choose the level of detail they want to work at. It is these interesting properties that led us to use hierarchical segmentation algorithms in our approaches.

Waterfalls

To obtain a hierarchical segmentation, a commonly used solution is the waterfalls' algorithm [14].

It supposes that a valued segmentation of the image has been obtained. The algorithm applies on this valued segmentation an operation (defined later) that produces a new valued segmentation where the least probable borders have been removed. The operation is repeated until all borders have been removed.

For each border, we register the number of times the operation was applied until it disappeared: this number is called **level**. The higher the level, the more important the border is. For the final output, the algorithm returns a new valued segmentation where the value of each border is its **level**. See Figure 3.32.



(a) *Valued segmentation* ***(b)*** *First iteration* ***(c)*** *Second iteration*

(d) *Final output*

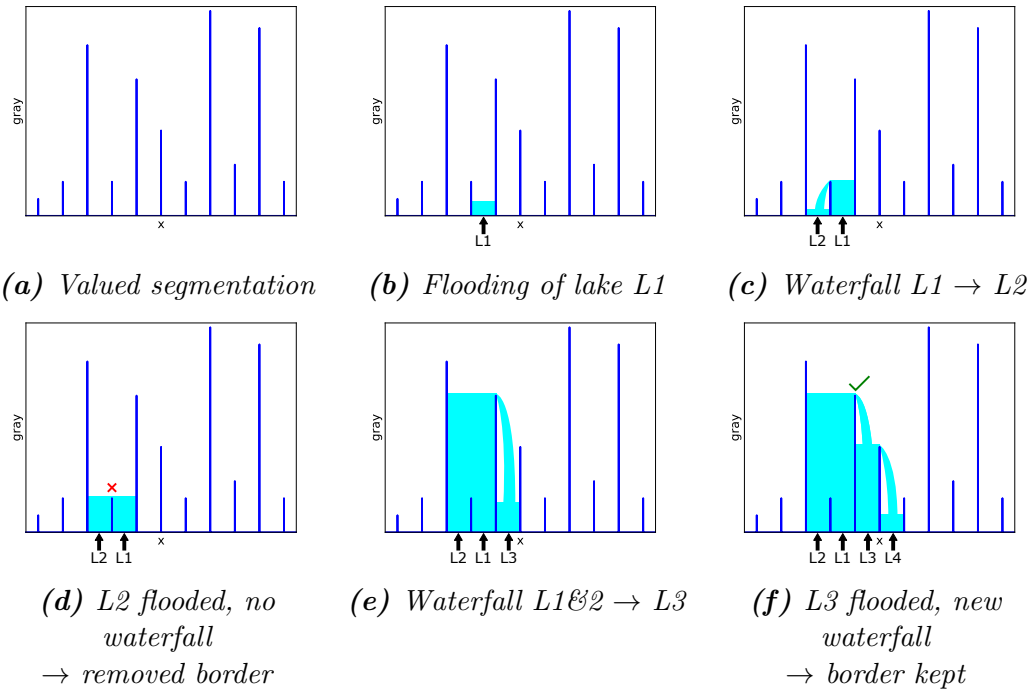
Figure 3.32: Waterfalls' illustration on a profile.

We need therefore to define this operation. Figure 3.33 has been created to illustrate its main idea and the origin of its name. In a nutshell, it retains the borders' local maxima: if we consider a profile, a border is retained if its value is greater than or equal to both the border on its left and the border on its right (and if it only has one neighbor, it is removed). The algorithm is detailed in Algorithm 8 and Figure 3.35. Figure 3.34 illustrates some results.

Enhanced waterfalls

Though the waterfalls algorithm is simple, it presents some defects that motivated additional research. Consider the gradient presented in Figure 3.37a. When computing the first level of waterfalls, we can observe that the second highest border disappears even though its value is very near from the highest one. Though the waterfalls' algorithm is correctly applied - this border was removed because it was not a borders' local maxima - this effect is not desired because it means that some important borders are removed from the lowest levels in the hierarchical segmentation.

Furthermore, if we add a small variation between the first and the second highest border (Figure 3.37b), the second border is then kept. As this small variation could be created by noise, it demonstrates the instability of the waterfalls' results.



(a) *Valued segmentation* ***(b)*** *Flooding of lake L1* ***(c)*** *Waterfall L1 → L2*

(d) *L2 flooded, no waterfall → removed border* ***(e)*** *Waterfall L1&2 → L3* ***(f)*** *L3 flooded, new waterfall → border kept*

Figure 3.33: Waterfalls' main idea. *We suppose that we have obtained the valued segmentation of an image (Figure (a)). We select a region* ***L1*** *and we begin to flood it (Figure (b)). It creates a lake that becomes higher and higher until it reaches a border: a waterfall then occurs from* ***L1*** *to the neighboring new region and lake* ***L2*** *(Figure (c)). Once* ***L2*** *is filled, we check if a new waterfall is created: here it is not the case, so the border between both lakes will be removed in the resulting image (Figure (d)). Both lakes are merged and continue to be flooded. Once a border is reached, there is a new waterfall to* ***L3*** *(Figure(e)). But this time, when* ***L3*** *is filled, it creates a new waterfall to* ***L4*** *so the border between* ***L1*** *&* ***L2*** *and* ***L3*** *is kept.*

A common solution to alleviate these problems is the enhanced waterfalls algorithm [30] [31]. Only one step is added when computing the next hierarchical segmentation level. For each removed border, the add-on anticipates the following hierarchical segmentation level by checking if that border would changes the next level's hierarchical image (defined in Algorithm 8); in other words, if the border is superior to the hierarchical image. If that is the case, it means that the border was removed too soon so it is added back. An alternative described in [31] is to also add back borders that are nearer to the hierarchical image than to 0.

The algorithm is described in Algorithm 9 and illustrated in Figure 3.38. Its results are shown in Figure 3.34.

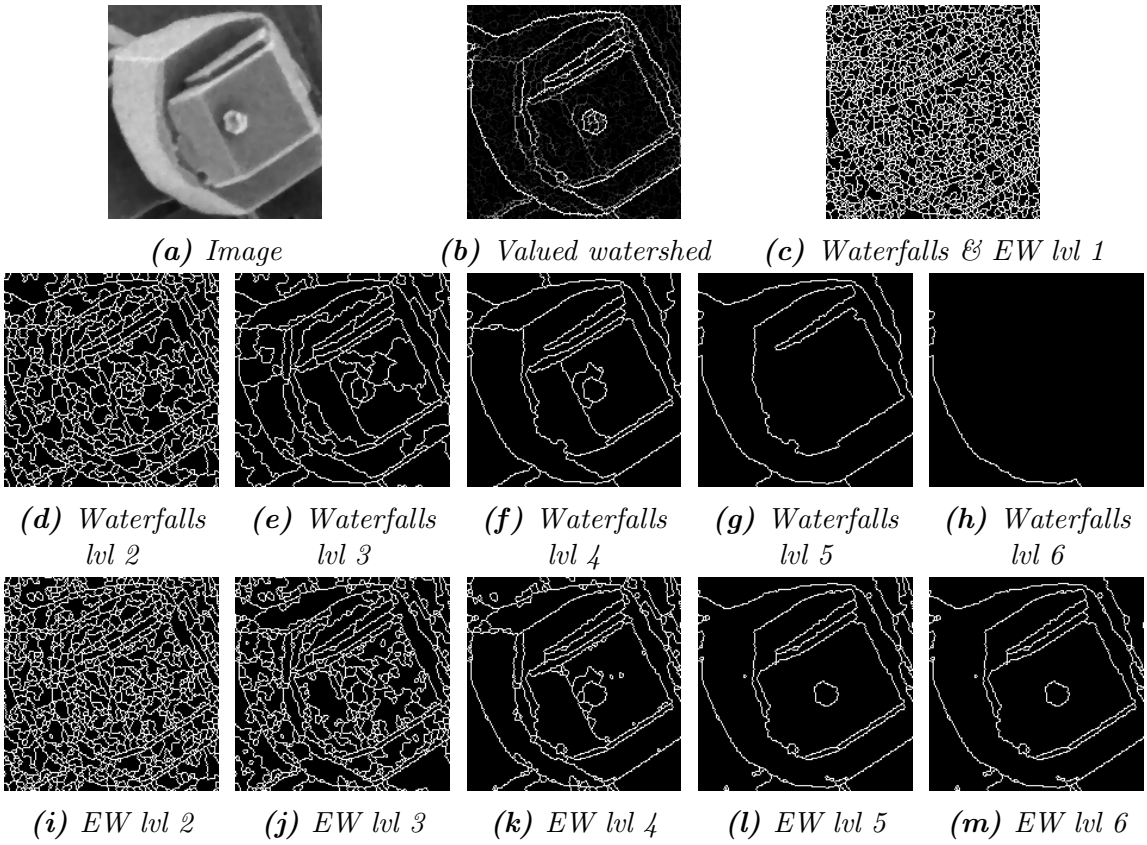


Figure 3.34: Illustration of the results of the waterfalls' and enhanced waterfalls' algorithms. *EW: Enhanced Waterfalls. Lower levels over segment the image. As we select higher level, fewer and more pertinent regions are kept. In the highest level, too much pertinent regions are removed.*

Other hierarchical segmentation algorithms

There exist other hierarchical segmentation algorithms, such as the P-algorithm [32], energy based methods [33], or more complex algorithms using tools such as directional gradient and histograms [23] [34]. Although they could produce better hierarchies on some criteria, they also require more computational power and are often parametric. As our solution had to produce results in a relatively low processing time, we didn't use those algorithms in the solutions we created. For applications where time isn't an issue, using these methods could constitute an axis of improvement.

Algorithm 8: The waterfalls algorithm

Function *waterfalls(f)*

```
/* f is a valued segmentation image.                              */
wf ← new image of same size as f ;
Set wf to 1 where f > 0 ;
nextF ← f ;
while there are still pixels > 0 in nextF do
    nextF ← getNextWaterfallsLevel(nextF) ;
    Add 1 to wf where nextF > 0 ;
return wf ;
```

Function *getNextWaterfallsLevel(f)*

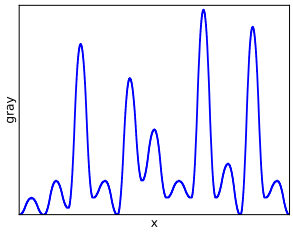
```
/* f is a valued segmentation image. This method keeps the most
   probable borders of f.                                         */
hierarchicalImage ← getHierarchicalImage(f) ;      // See Figure 3.35c
minima ← hminima of hierarchicalImage, h = 1 ;     // See Figure 3.35d
newF ← valued watershed of hierarchicalImage using minima as markers ;
newF ← handleEdgeCases(f, newF) ;                  // See Figure 3.36
return newF ;
;                                                  // See Figure 3.35e
```

Function *getHierarchicalImage(f)*

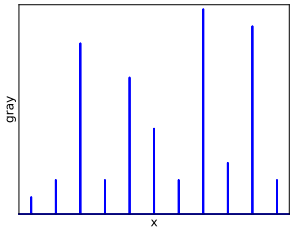
```
/* f is a valued segmentation image. This method simulate the
   flooding and waterfalls process.                               */
threshIm ← f ;
Set threshIm to 255 where threshIm = 0 ;
return dual geodesic reconstruction of threshIm by f ;
```

Function *handleEdgeCases(f, nextF)*

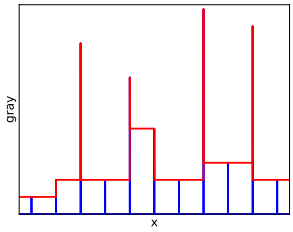
```
/* f is a valued segmentation image. nextF is the valued
   segmentation image of the next level of the hierarchical
   segmentation. On some edge cases, nextF might contain new
   borders compared to f, which is inconsistent as the next
   level of hierarchical segmentation should contain only a
   subset of the borders in f. See Figure 3.36. This method
   takes care of that.                                            */
floor ← new binary image of same size as f ;
Set floor to 1 where f is 0 ;
addedBorders ← new binary image of same size as f ;
Set addedBorders to 1 where on new borders in nextF compared to f ;
reachNeighboringBorders ← geodesic reconstruction of addedBorders in
  floor ;
reachNeighboringBorders ← dilation of reachNeighboringBorders ;
neighboringBorders ← new binary image of same size as f ;
Set neighboringBorders to f where reachNeighboringBorders > 0 and
  f > 0 ;
Remove addedBorders in nextF;
Add neighboringBorders to nextF ;
return nextF;
```



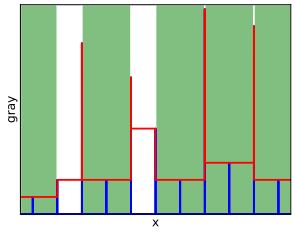
(a) *Gradient*



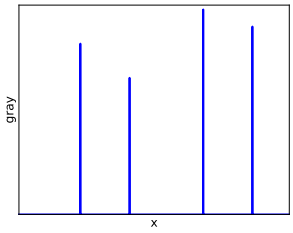
(b) *Valued seg. (lvl 1)*



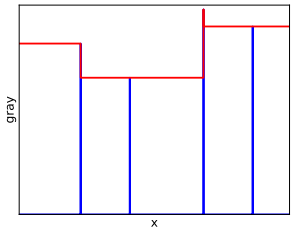
(c) *Hierarchical im. of (b)*



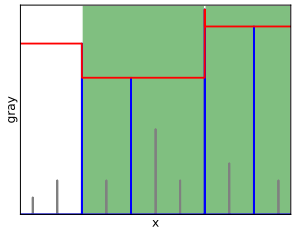
(d) *Minima of (c)*



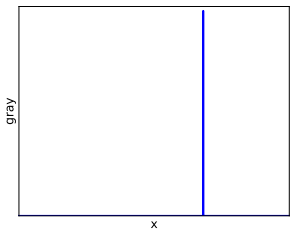
(e) *Valued seg. (lvl 2)*



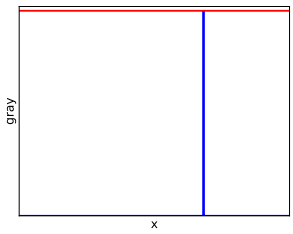
(f) *Hierarchical im. of (e)*



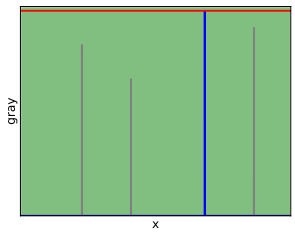
(g) *Minima of (f)*



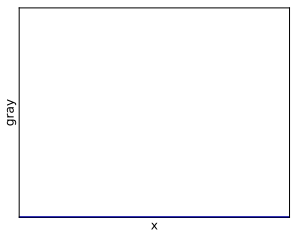
(h) *Valued seg. (lvl 3)*



(i) *Hierarchical im. of (h)*

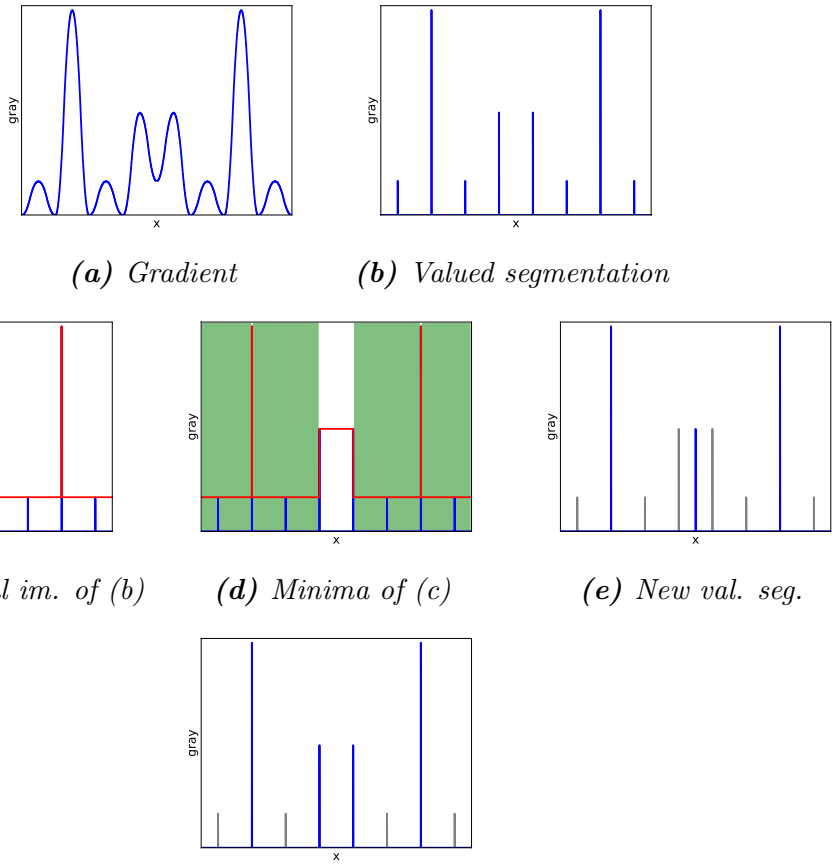


(j) *Minima of (i)*



(k) *Valued seg. (lvl 4)*

Figure 3.35: The different steps of the waterfall's algorithm. *Hierarchical im.: Hierarchical image.*



(a) *Gradient*

(b) *Valued segmentation*

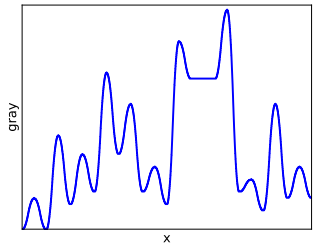
(c) *Hierarchical im. of (b)*

(d) *Minima of (c)*

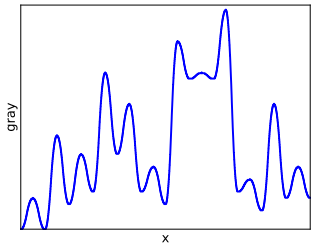
(e) *New val. seg.*

(f) *New valued segmentation with post-processing*

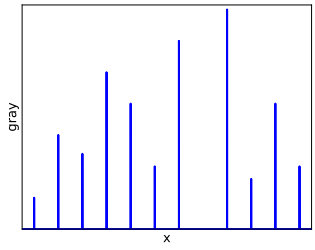
Figure 3.36: Edge case: waterfalls with and without post processing. *The middle border is added on the second level, which is inconsistent as no border should be added. The post-processing method removes this border and keep the two neighboring ones instead. Hierarchical im.: Hierarchical image. Val. seg. Valued segmentation.*



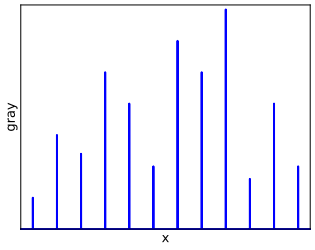
(a) *Gradient 1*



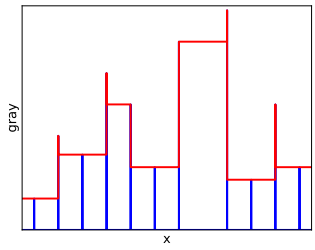
(b) *Gradient 2*



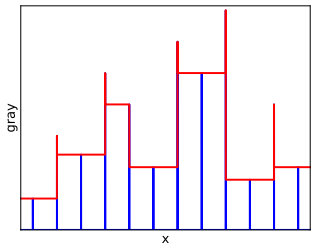
(c) *Valued segmentation of (a)*



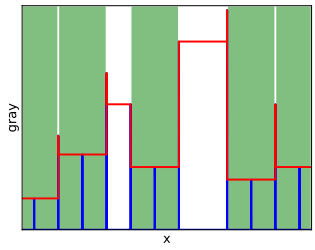
(d) *Valued segmentation of (b)*



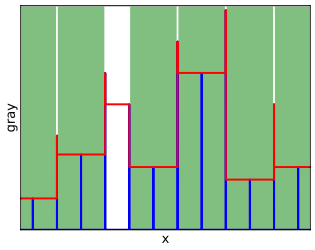
(e) *Hierarchical image of (c)*



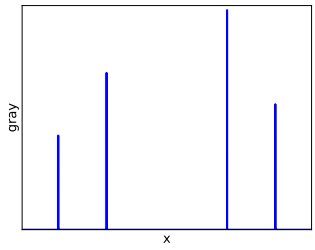
(f) *Hierarchical image of (d)*



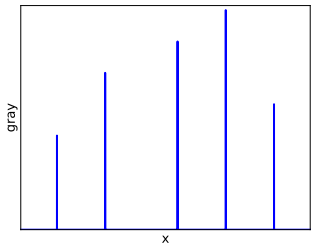
(g) *Minima of (e)*



(h) *Minima of (f)*



(i) *Resulting valued segmentation*



(j) *Resulting valued segmentation*

Figure 3.37: Waterfalls' instability. *Between (a) and (b), only a small change, that could be created by noise, has been added. Yet, applying the waterfalls operation results in a different hierarchy, with different strong edges. Val. seg.: Valued segmentation.*

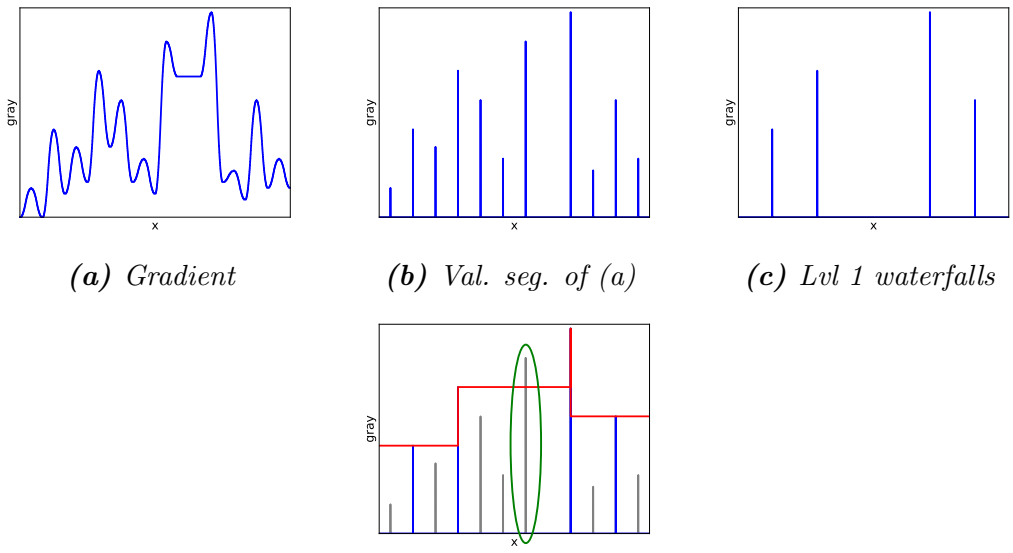
Algorithm 9: The enhanced waterfalls algorithm

Function *enhancedWaterfalls(f)*

```
/* f is a valued segmentation image.                                  */
wf ← new image of same size as f ;
Set wf to 1 where f > 0 ;
nextF ← f ;
previousF ← new image of same size as f ;
while previousF ≠ nextF do
    /* The enhanced waterfalls' algorithm stops when idempotence
       has been attained.                                             */
    previousF ← nextF ;
    nextF ← getNextEnhancedWaterfallsLevel(nextF) ;
    Add 1 to wf where nextF > 0 ;
return wf ;
```

Function *getNextEnhancedWaterfallsLevel(f)*

```
/* f is a valued segmentation image.  The methods adds a
   post-processing step to the waterfalls transformation.             */
newF ← getNextWaterfallsLevel(f) ;      // See Algorithm 8 and Figure
 3.38c
hier ← getHierarchicalImage(newf) ;     // See Algorithm 8 and Figure
 3.38d
Set newF to f where f is greater than hier ;          // see Figure 3.38d
return newF ;
```



(a) *Gradient* ***(b)*** *Val. seg. of (a)* ***(c)*** *Lvl 1 waterfalls*

(d) *Hierarchical im. of (c)*
Green ellipse: added border

Figure 3.38: Enhanced waterfalls' algorithm. *Borders that are added back into the hierarchical segmentation. Hierarchical im.: Hierarchical image. Val. seg.: Valued segmentation.*

3.4 Conclusion

In this chapter, we have presented state of the art image processing tools that will be used by our methods. After introducing the basic operators, we presented the hierarchical segmentation chain and multiple alternatives for each component.

We will now introduce multiple methods that estimate the topography of microscopic objects.

Chapter 4

Stereo reconstruction: common issues and solutions

French summary / Résumé en français

Les méthodes de reconstruction 3D que nous avons développés s'appuient sur le principe de reconstruction stéréoscopique. Après justifier notre choix par rapport aux autres méthodes de reconstruction 3D (comme la photoclinométrie), nous décrivons l'idée générale de la reconstruction stéréoscopique en deux parties principales. Tout d'abord, le processus dit de triangulation permettant de retrouver la position 3D d'un point à partir de deux photographies prises à des points de vue différents, en supposant que la position de ce point est connue sur chaque photographie. Ensuite, l'estimation de la carte de disparité permettant de suivre le déplacement de chaque point entre une paire d'image : sans ce suivi, la triangulation est impossible. Nous décrivons ensuite plusieurs méthodes permettant de raffiner cette carte de disparité, ainsi que plusieurs extensions possibles permettant d'utiliser plus de deux images.

There are multiple ways of obtaining the 3D topography of a microscopic element.

The Atomic-Force Microscope (AFM) [35] scans the sample's surface using a mechanical probe with a sharp tip. Though mechanical, this method achieves a very high resolution - up to 0.1 nm. However it is not adapted to our use case because it can only process samples with a height up to 20 µm and a typical acquisition can require several minutes to be completed.

Electron Tomography [36] scans a sample in a similar way than an X-ray but using electrons. A 3D topography of a sample can be obtained by acquiring multiple scans in different angles. The sample is tilted on a single axis or two axes, through a classical 140° range. The method is however not adapted to our use case as it can only process objects measuring a few hundreds of nanometers

X-ray Tomography can reconstruct the full volume of micrometer sized samples. The resolution can range from a few hundreds of nanometers to tens of micrometers [37]. This is not however an appropriate solution because of the large amount of time necessary to prepare and adjust the samples.

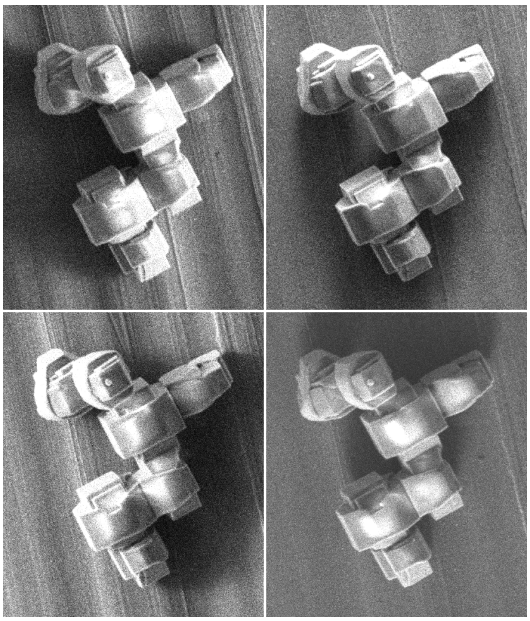
Photoclinometry, or Shape from Shading [38], is a general method that can also be applied on SEM images. Multiple images of an object are taken with different light directions, and the method estimates a 3D topography through detected changes in shadows and illumination levels. The different light directions can be simulated in the SEM by using an Everhart-Thornley (ET) or AsB detector as described in Section 2.3. This method has been often used to establish the 3D topography of microscopic objects, notably for testing the quality of integrated circuits during the 1990s [39, 40, 41, 42]. Though this method can be applied on our samples, the quality of the results is not satisfying for our purposes (see Figure 4.1). Nowadays, stereo reconstruction is generally preferred because of its higher precision [5, 43, 3, 4].

Stereo reconstruction is the process of estimating a 3D topography of a scene from several images taken at different viewpoints. Standard stereo methods work on pairs of images whereas multiview stereo methods can work on tens, hundreds or thousands images. In our case, we can only take a few images of the same sample as we want to get a topography estimation within about 30 to 60 seconds. This is insufficient for most multiview stereo methods, so we will mainly focus on the standard stereo methods.

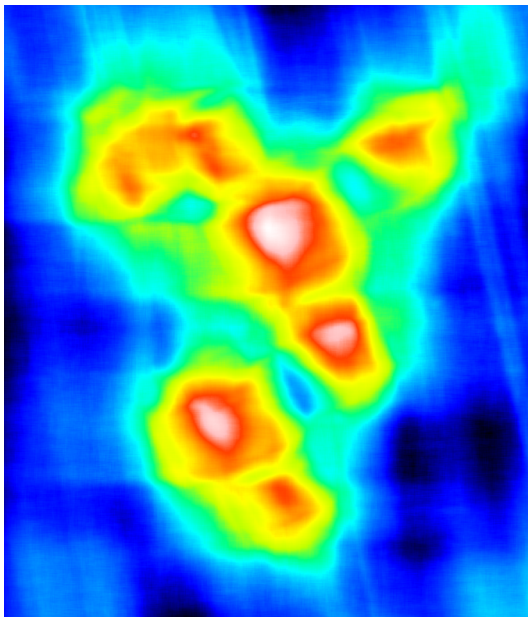
We will first suppose that we have a pair of images of the same scene taken at different viewpoints: a reference image and a secondary image (see Figure 4.2a). For a given point in the reference image, we can determine where this point is placed on the secondary image (see Figure 4.2b). Thanks to this match and various information such as the position and orientation of the viewpoints, we can evaluate the elevation of the point using the **triangulation** process described in Section 4.1.

However, matching all the points of the images would be too much time consuming if this is done manually. Therefore, stereo methods try to automatically match these points. The result of this matching process is the **disparity map**: it evaluates the displacement of each point between the reference image and the secondary image. The disparity map evaluation is the main problem that stereo methods try to solve. In Section 4.2, we describe several methods that attempt to tackle this problem, with varying degrees of success.

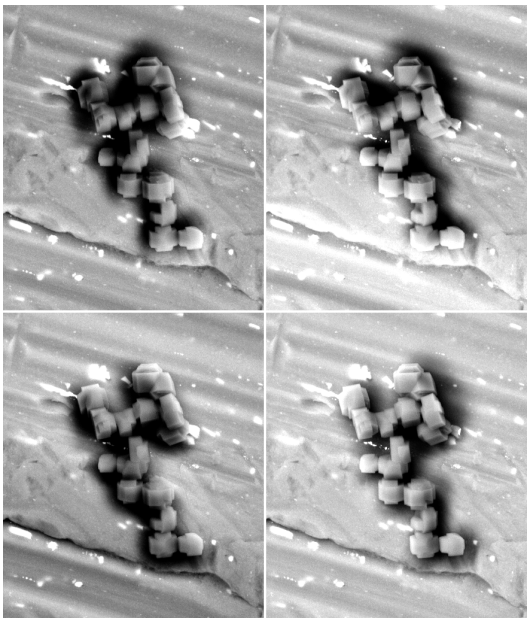
In a nutshell, stereo methods first evaluate automatically a disparity map of the pair of images. From this disparity map and various information such as the position and orientation of the viewpoints, a 3D map can then be deduced. See Figure 4.3.



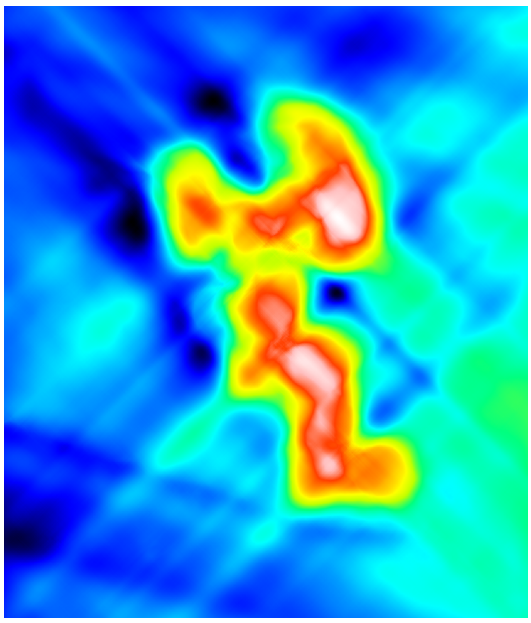
(a) *Images acquired using the ET detector*



(b) *Shape from Shading reconstruction of (a)*



(c) *Images acquired using the AsB detector*



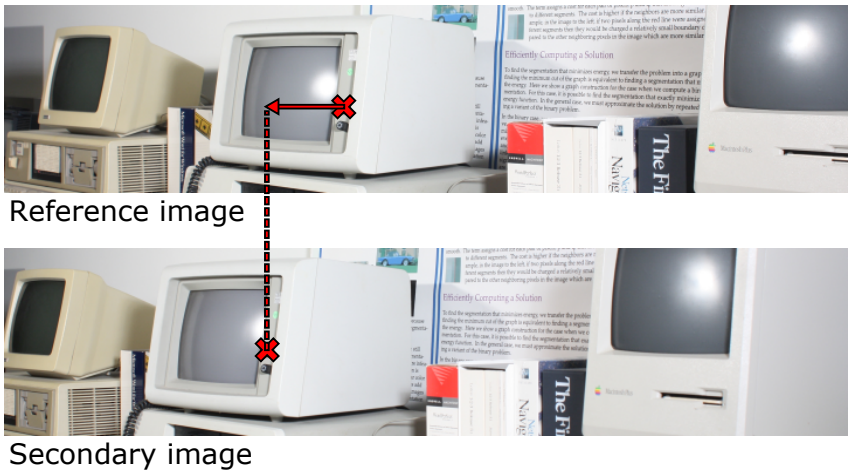
(d) *Shape from Shading reconstruction of (c)*

Figure 4.1: Some results obtained using Shape from shading. *ET: Everhart-Thornley. The different simulated light directions are* 0°, 90°, 180°, 270°. *Results have been obtained using the MountainsMap software [44].*

4.1 Geometry and the triangulation process

Before describing the triangulation process - which evaluates the distance of a point from its position on two images - we describe how a 3D point is projected on an image in Section 4.1.1. In Section 4.1.2, we then use the developed formulas of the previous section to define the inverse process that interests us: triangulation.

(a) Pair of stereo images



(b) Matching of a point

Figure 4.2: Point matching between a pair of images.

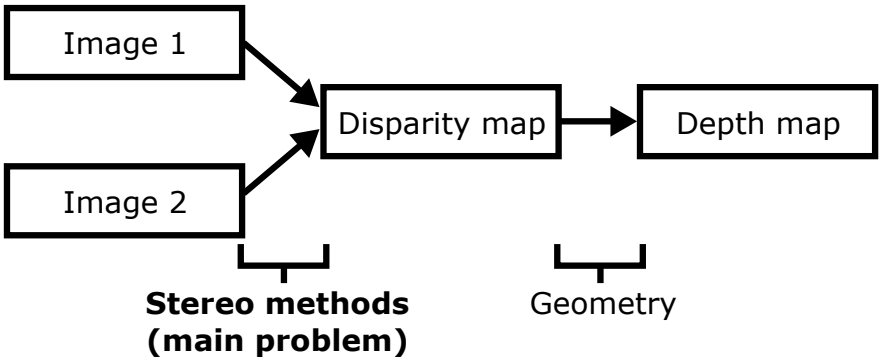


Figure 4.3: *Common depth map estimation process between two images.*

4.1.1 From the 3D coordinates of a point to its 2D projection

A picture, whether acquired by classical cameras or by SEMs, is the graphical projection of a 3D scene into a 2D plan, **the image plan**. There is therefore a relationship between the 3D position of a point in the scene and its 2D position in the image.

Several parameters will influence this relationship. These parameters can be grouped into two categories. Intrinsic parameters depend on the acquisition system. Extrinsic parameters depend on the relative position and orientation of the acquisition system to the scene.

We can determine the projection of a 3D point into the image in two steps.

First, the 3D coordinates of the point is relativized to the center and orientation of the acquisition system. From the initial homogeneous coordinates of the point $P = (X, Y, Z, 1)$, we compute the relative coordinates $P' = (X', Y', Z', 1)$ such as the position and orientation of the camera center is the origin.

$$P' = \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = EP = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

With:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, T = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

R^{-1} being the rotation matrix of the camera relative to the scene, and $-T$ being the position of the center of the camera relative to the origin point of the scene.

Once we obtained the relative coordinates P', we can compute the 2D projection of P' in the image plane. To do so, several models exist. The most commonly used is the pinhole camera model.

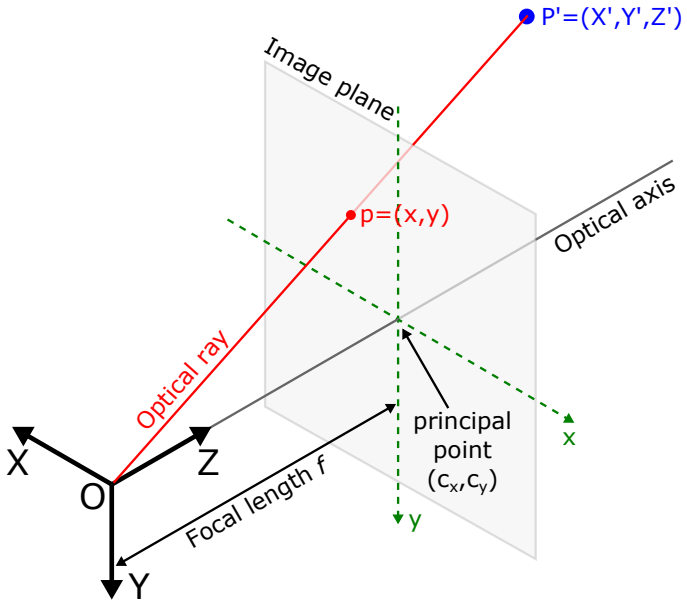
The pinhole camera model assumes that the point P' has an optical ray that passes through the origin point O (see Figure 4.4). The ray intersects with the image plane, located at a distance f of O, f being called the **focal length**. The projection of P' on the image plane is located at the intersection, named p. We want to know the 2D coordinates of p in the image plane.

Using trigonometry, we can obtain the homogenous coordinates of p, called $\tilde{p} = (\tilde{x}, \tilde{y}, \tilde{z})$:

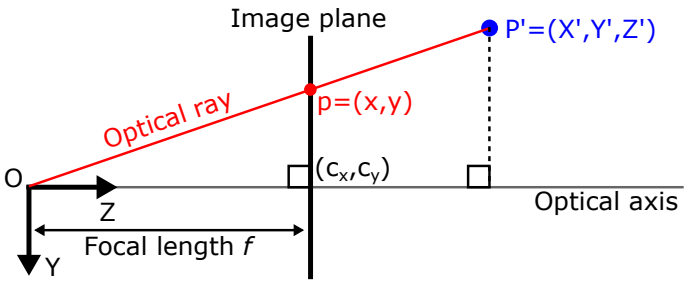
$$\tilde{p} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = KP' = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix}$$

(c_x, c_y) being the principal point, intersection between the optical axis ($X = 0$ and $Y = 0$) and the image plane.

The image coordinates $p = (x, y)$ are retrieved from the homogeneous coordinates as follows: $x = \tilde{x}/\tilde{z}$ and $y = \tilde{y}/\tilde{z}$.



(a) 3D view



(b) 2D view

Figure 4.4: Projection model of a pinhole camera.

To sum it up, here is how a point $P = (X, Y, Z, 1)$ will be projected in the image at a point $\tilde{p} = (\tilde{x}, \tilde{y}, \tilde{z})$:

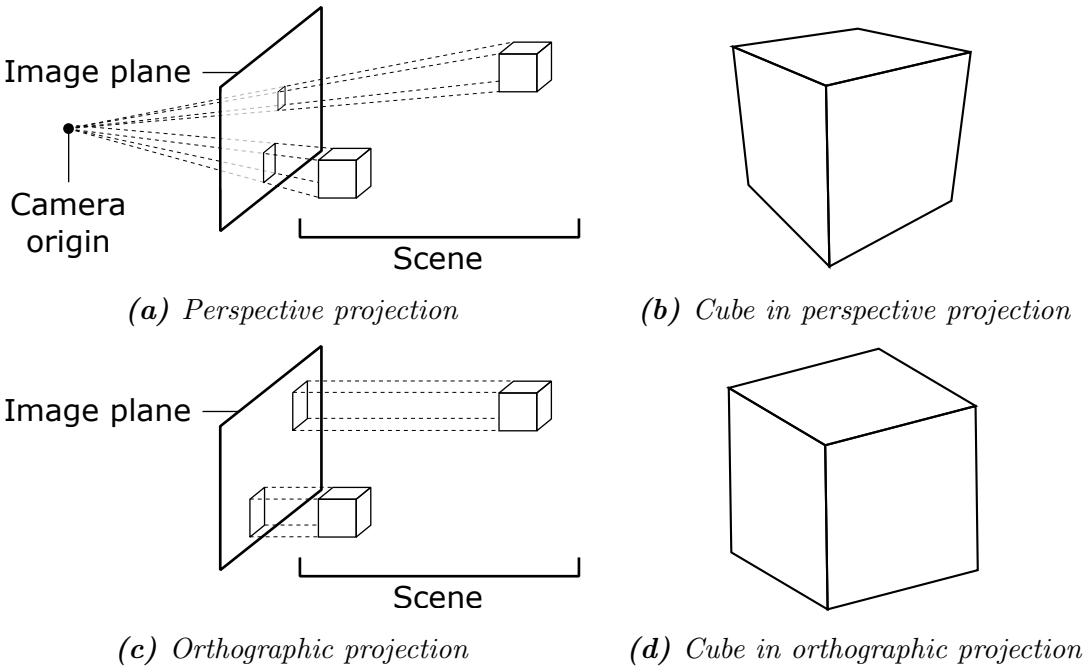
$$\tilde{p} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = KP' = KEP = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Special case: SEM images and orthographic projection

When acquiring SEM images of our catalysts, the magnification is generally around 5000× and the focal length f (given by the working distance) is around 5mm. Compared to the focal length, the height ΔZ of our catalysts are very low (typically 2 µm). One of the consequences is that the projection model is closer to the orthographic model [45, 4, 3]: the height of a point doesn't significantly affect its position on the acquired image. See Figure 4.5.

In that case, as $(f/\Delta Z) \to \infty$, the intrinsic matrix can be simplified:

$$K = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(a) *Perspective projection*

(b) *Cube in perspective projection*

(c) *Orthographic projection*

(d) *Cube in orthographic projection*

Figure 4.5: Perspective and orthographic projection.

However, if that approximation is sufficient for pixel-wise measurements, it might be too inexact for subpixel-wise measurements [5]. We will check if that hypothesis is accurate enough in Section A.

Distortions

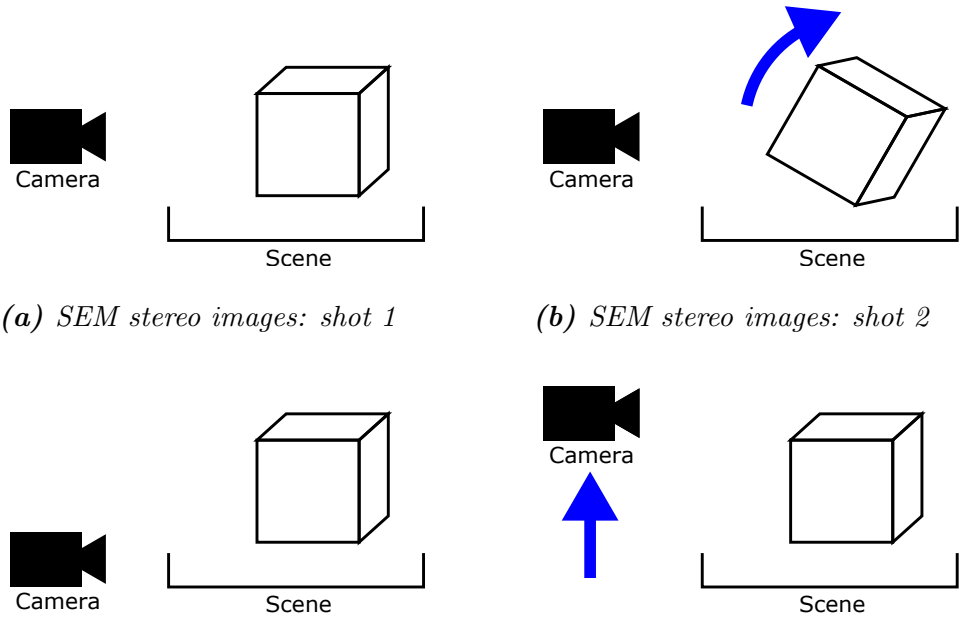
Pictures might contain some distortions due to the acquisition system. In particular, SEM images, especially on low magnifications, can present radial or spiral distortions [3] [4]. For magnifications higher than 500×, these distortions are considered negligible [2]. However, this approximation might be too inexact for subpixel-wise measurements, so we will check if that hypothesis is accurate enough in Section A.

4.1.2 Determining the height of a point from two images

We have previously explained the projection process: how a 3D point in the scene is projected into a 2D coordinates in an image. Our objective is to solve the inverse problem: we don't know the 3D topography of a scene, and we want to estimate it from several 2D images. This process is called **triangulation**.

In this section, we will suppose that we know the 2D projection of a 3D point in two images. From these two 2D coordinates, $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, we know that they correspond to the same point in the scene $P = (X, Y, Z)$, and we want to estimate Z. In order to solve this problem, we will suppose that the intrinsic and extrinsic parameters have been estimated by a process called calibration that we will describe in Section A.

In this work, we will study two categories of stereo images. SEM stereo images rely on the rotation of the sample in a single axis. Middlebury stereo images are taken at the same orientation but slightly different positions. See Figure 4.6.



(a) *SEM stereo images: shot 1*

(b) *SEM stereo images: shot 2*

(c) *Middlebury stereo images: shot 1*

(d) *Middlebury stereo images: shot 2*

Figure 4.6: How stereo images are acquired: SEMs and Middlebury.

SEM stereo images

In SEM stereo images, the acquisition system doesn't move, only the sample is rotated around the x axis by $\theta°$. The rotation matrix is therefore:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_i) & -\sin(\theta_i) \\ 0 & \sin(\theta_i) & \cos(\theta_i) \end{bmatrix}$$

We will assume that the focal length f, the principal point (c_x, c_y), and the translation (t_x, t_y, t_z) are known and remain constant between the two acquisitions. From Section 4.1.1:

$$\tilde{p_i} = \begin{bmatrix} \tilde{x_i} \\ \tilde{y_i} \\ \tilde{z_i} \end{bmatrix} = \tilde{z_i} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & \cos(\theta_i) & -\sin(\theta_i) & t_y \\ 0 & \sin(\theta_i) & \cos(\theta_i) & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X + t_x \\ Y\cos(\theta_i) - Z\sin(\theta_i) + t_y \\ Y\sin(\theta_i) + Z\cos(\theta_i) + t_z \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} f(X + t_x) + c_x(Y\sin(\theta_i) + Z\cos(\theta_i) + t_z) \\ f(Y\cos(\theta_i) - Z\sin(\theta_i) + t_y) + c_y(Y\sin(\theta_i) + Z\cos(\theta_i) + t_z) \\ Y\sin(\theta_i) + Z\cos(\theta_i) + t_z \end{bmatrix}$$

Therefore:

$$\tilde{z_i} = Y\sin(\theta_i) + Z\cos(\theta_i) + t_z$$

$$\tilde{y_i} = y_i\tilde{z_i}$$

$$y_i \times (Y\sin(\theta_i) + Z\cos(\theta_i) + t_z) = fY\cos(\theta_i) - fZ\sin(\theta_i) + ft_y + c_y \times (Y\sin(\theta_i) + Z\cos(\theta_i) + t_z)$$

$$(y_i\cos(\theta_i) + f\sin(\theta_i) - c_y\cos(\theta_i))Z + (y_i\sin(\theta_i) - f\cos(\theta_i) - c_y\sin(\theta_i))Y = ft_y + c_yt_z - y_it_z$$

$$((y_i - c_y)\cos(\theta_i) + f\sin(\theta_i))Z + ((y_i - c_y)\sin(\theta_i) - f\cos(\theta_i))Y = ft_y + c_yt_z - y_it_z$$

We must solve the following system:

$$\begin{cases} ((y_1 - c_y)\cos(\theta_1) + f\sin(\theta_1))Z + ((y_1 - c_y)\sin(\theta_1) - f\cos(\theta_1))Y = aZ + bY = ft_y + c_yt_z - y_1t_z \\ ((y_2 - c_y)\cos(\theta_2) + f\sin(\theta_2))Z + ((y_2 - c_y)\sin(\theta_2) - f\cos(\theta_2))Y = cZ + dY = ft_y + c_yt_z - y_2t_z \end{cases}$$

$$Y = \frac{ft_y + c_yt_z - y_2t_z - cZ}{d}$$

$$aZ + bY = aZ + b\frac{ft_y + c_yt_z - y_2t_z - cZ}{d} = Z(a - \frac{bc}{d}) + \frac{ft_y + c_yt_z - y_2t_z}{d} = ft_y + c_yt_z - y_1t_z$$

$$Z = \frac{(d - b)(ft_y + c_yt_z) - t_z(dy_1 - by_2)}{ad - bc}$$

Assuming $d \neq 0$.

If an orthographic projection is assumed, the problem is much simpler:

$$\tilde{p_i} = \begin{bmatrix} \tilde{x_i} \\ \tilde{y_i} \\ \tilde{z_i} \end{bmatrix} = \tilde{z_i} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_i) & -\sin(\theta_i) & 0 \\ 0 & \sin(\theta_i) & \cos(\theta_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y\cos(\theta_i) - Z\sin(\theta_i) \\ Y\sin(\theta_i) + Z\cos(\theta_i) \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} X \\ Y\cos(\theta_i) - Z\sin(\theta_i) \\ 1 \end{bmatrix}$$

Therefore:

$$\tilde{z}_i = 1$$
$$x_i = X$$
$$y_i = Y\cos(\theta_i) - Z\sin(\theta_i)$$

We must solve the following system:

$$\begin{cases} Y\cos(\theta_1) - Z\sin(\theta_1) = y_1 \\ Y\cos(\theta_2) - Z\sin(\theta_2) = y_2 \end{cases}$$

$$Y = \frac{y_2 + Z\sin(\theta_2)}{\cos(\theta_2)}$$

$$\frac{y_2 + Z\sin(\theta_2)}{\cos(\theta_2)}\cos(\theta_1) - Z\sin(\theta_1) = y_1$$

$$\frac{y_2\cos(\theta_1) + Z(\sin(\theta_2)\cos(\theta_1) - \cos(\theta_2)\sin(\theta_1))}{\cos(\theta_2)} = y_1$$

$$\frac{y_2\cos(\theta_1) + Z\sin(\theta_2 - \theta_1)}{\cos(\theta_2)} = y_1$$

$$Z = \frac{y_1\cos(\theta_2) - y_2\cos(\theta_1)}{\sin(\theta_2 - \theta_1)}$$

The orthographic projection model requires less parameters, limiting potential problems in the calibration phase.

Middlebury stereo images

For the Middlebury stereo pairs, the scene or the camera is not rotated but the camera is translated on the x axis. Between two images, the focal length l and rotation matrix R remain constant, however the x components of the translation vector T and the principal point change. For sake of simplicity, we consider that the camera is placed on the X axis and aligned to the Z axis.

$$\tilde{p_i} = \begin{bmatrix} \tilde{x_i} \\ \tilde{y_i} \\ \tilde{z_i} \end{bmatrix} = \tilde{z_i} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_{i,x} & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_{i,x} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} f & 0 & c_{i,x} & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X + t_{i,x} \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} f(X + t_{i,x}) + c_{i,x}Z \\ fY + c_yZ \\ Z \end{bmatrix}$$

Therefore:

$$\tilde{z_i} = Z$$
$$\tilde{x_i} = x_i\tilde{z_i}$$
$$x_iZ = f(X + t_{i,x}) + c_{i,x}Z$$
$$(x_i - c_{i,x})Z - fX = ft_{i,x}$$

We must solve the following system:

$$\begin{cases} (x_1 - c_{1,x})Z - fX = ft_{1,x} \\ (x_2 - c_{2,x})Z - fX = ft_{2,x} \end{cases}$$

$$-fX = ft_{2,x} - (x_2 - c_{2,x})Z$$
$$(x_1 - c_{1,x})Z + ft_{2,x} - (x_2 - c_{2,x})Z = ft_{1,x}$$
$$(x_1 - x_2 - (c_{1,x} - c_{2,x}))Z = f(t_{1,x} - t_{2,x})$$
$$Z = \frac{t_{1,x} - t_{2,x}}{x_1 - x_2 - (c_{1,x} - c_{2,x})} f$$

A characteristic of this acquisition system is that the closer an object is to the camera, the larger will be $x_1 - x_2$, the displacement of the point between both images.

Image rectification

For improving the performance of the matching process, images are often rectified [46] [47] : a projective transformation H is computed so that, in the new images, each projected 3D point share the same y-component. It heavily simplifies the matching process, because the search is limited to 1-dimension (the x-axis) instead of 2 dimensions.

As we don't need image rectification for our SEM images (shown in Appendix A), and Middlebury images are already rectified, we won't enter into details on how this transformation is computed. This is the subject of considerable research [47, 48, 49, 11]. We will assume from now on that our images share the properties of rectified images.

It is important to underline that, although we don't rectify our SEM acquisitions, we apply on them a 90° rotation to make them compatible with most stereo matching algorithms. Indeed, the samples are tilted around the x-axis, and, therefore, before rotating images, the points do not move on the x-axis as on standard stereo images but on the y-axis.

4.2 Disparity map estimation

In the previous section, we have described how we can deduce the 3D position of a point in the scene - notably its distance to the camera - from its 2D positions in a pair of images.

If we can match each pixel of the reference image with a pixel in the secondary image, we are able to determine for each pixel its distance to the camera, allowing us to produce a **depth map** or **3D map**.

We need therefore to estimate an intermediary map called **disparity map**. A disparity map estimates for each pixel of a reference image its displacement in the secondary image. For evaluating these displacements, stereo methods rely on the assumption that the shape and the color of the objects in the scene will remain similar between both images.

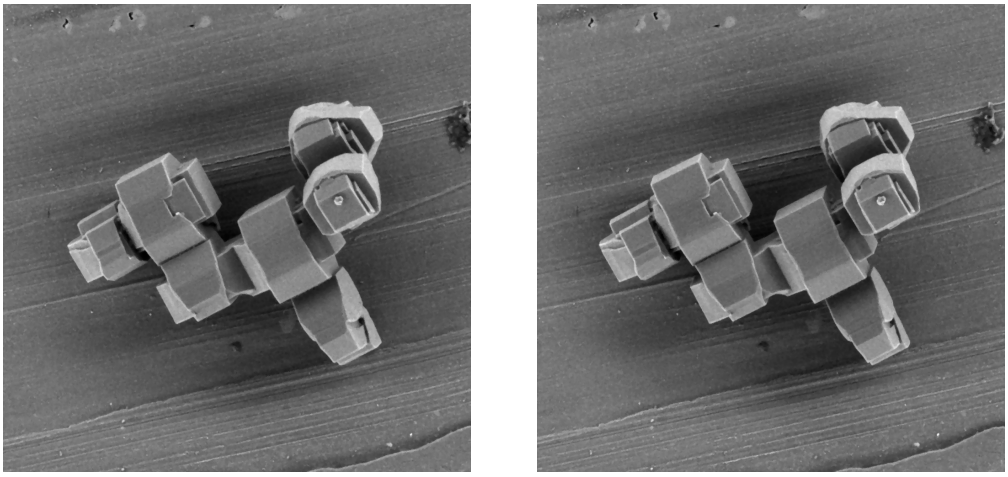
This assumption allows us to transform the problem to a correlation problem: we need to know, for each point p of a reference image, which point p' in the secondary image looks the most similar. The disparity of the point p is then: $d = p' - p$.

To find the point p', we need a similarity measure $s(p, q)$ that compares a point p in the reference image to a point q in the secondary image; the lower the score, the lower the differences and the higher the similarity. We will first assume that $p' = \arg\min_q(s(p, q))$, but we will see in Section 4.3 that there exist aggregation methods that take into account the similarity scores of neighboring points to choose a more robust match.

As images are supposed rectified, we measure this similarity score only on points on the same y-axis. Furthermore, in order to improve speed and accuracy, we define a range $[d_{\min}, d_{\max}]$ where this score will be tested. A point $p = (x, y)$ in the reference image will have its similarity compared to all the points $[(x + d_{\min}, y), (x + d_{\min} + 1, y), ..., (x + d_{\max}, y)]$ in the secondary image.

Moreover, algorithms will be described for grayscale images, but they can be easily extended to color images by combining the scoring metric of each channel for instance.

In this section, we will illustrate the results of the different matching algorithms on the pair of images shown in Figure 4.7. We suppose that the images have been aligned (following the process described in Section 7.3.1), and that they share the properties of rectified images.



(a) *Reference image* ***(b)*** *Secondary image*

Figure 4.7: Aligned and rectified pair of SEM images.

4.2.1 Pixel matching

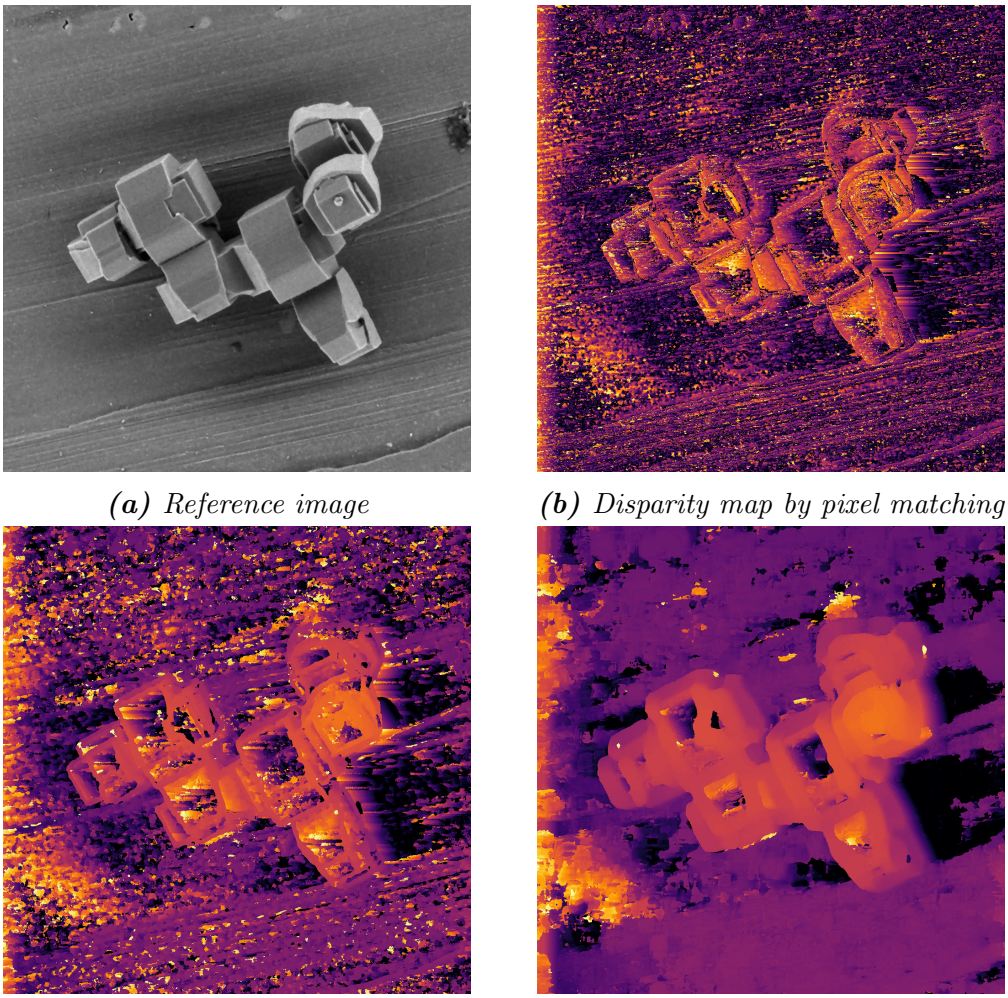
The simplest similarity measure is to compare the intensity of pixels:

$$s(p, q) = ||f_r(p) - f_s(q)||$$

f_r being the reference image and f_s being the secondary image. $s(p, q) = 0$ if p and q have the same intensity, and the more different their intensity are, the lower s will be.

However, if we compute the disparity map using this matching score on our baseline pair showed in Figure 4.7, results are disappointing. See Figure 4.8b.

Indeed, although some edges are recognizable, the disparity map is very noisy. This result can be explained by the short-sightedness of the comparison metric. It is indeed very probable that multiple pixels in the secondary images will have the same intensity, especially in homogeneous areas. Noise in the images, especially acute in SEM images, can make the problem even worse. In these cases, no correct matching can occur without additional information. The block matching algorithm improves the results by taking into account neighboring pixels in the similarity score.



(a) *Reference image*

(b) *Disparity map by pixel matching*

(c) *Disparity map by SAD (size: 9px)*

(d) *Disparity map by SAD (size: 31px)*

Figure 4.8: Results of pixel matching and block matching. *SAD: Sum of Absolute Difference (block matching).*

4.2.2 The block matching algorithm

Instead of matching pixels, the block matching algorithm compares blocks of pixels. The scoring metric between two pixels depends therefore also on their neighborhood. This neighborhood consists of a square centered on the pixels.

The size of these squares is an important parameter as it can have an important effect on the estimated disparity maps. Indeed, increasing this size can reduce the noise in the disparity map, but larger size can also lead to undesirable effects such as the fattening effect described in Section 4.2.4 and showed in Figure 4.8.

There are multiple ways of comparing these blocks of pixels. We describe a few ones in Section 4.2.3.

4.2.3 Block comparison metrics

Numerous block comparison metrics exist [50], with their relative advantages and disadvantages. We will discuss here the ones we will use in our solutions.

Sum of Absolute Differences (SAD) and Sum of Squared Differences (SSD)

The Sum of Absolute Differences (SAD) metric is an adaptation of the pixel matching metric to block matching. If B_p is the block around p and B_q is the block around q, and in both case the block size is (b_s, b_s):

$$s(p,q) = \sum_{x=1}^{b_s} \sum_{y=1}^{b_s} |B_p(x,y) - B_q(x,y)|$$

The Sum of Squared Differences (SSD) metric is a variation of SAD. It penalizes higher changes of intensity. It is obtained using this formula:

$$s(p,q) = \sum_{x=1}^{b_s} \sum_{y=1}^{b_s} (B_p(x,y) - B_q(x,y))^2$$

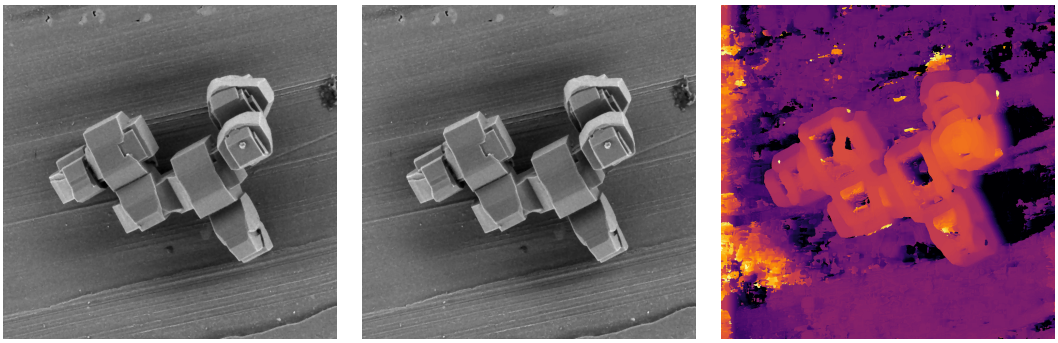
Some examples of disparity maps obtained using the SAD metric are shown in Figure 4.8. Results are quite similar for the SSD metric. Compared to pixel matching there is a clear improvement, but the quality of the estimated disparity map is still very dependent on the amount of texture in the image and the size of the block.

Image preprocessing + SAD

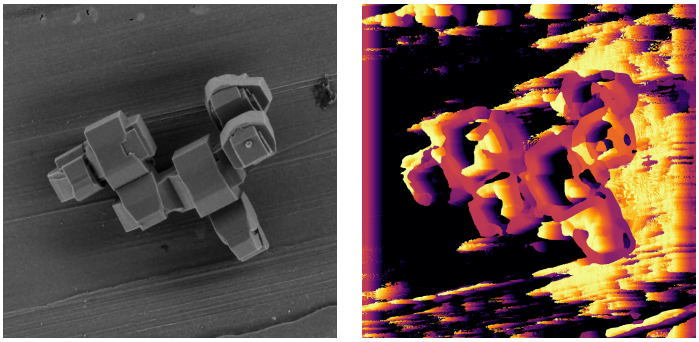
One problem of both the SAD and SSD metrics is their dependence on the illumination of the scene: if the reference and secondary images have globally different brightness levels, or if the illumination of some parts of the scene is reduced because of some shadowing effect, the accuracy of the matching processing will drop as shown in Figure 4.9.

A common approach, notably taken by the block matching implementation from the OpenCV library [13], is to apply a pre-processing step that filters the reference and secondary images so that the results are more robust to change of illumination.

We present in the following two alternative pre-processing, used in the Open-CV library.



(a) *Reference image* ***(b)*** *Secondary image* ***(c)*** *Disparity map by SAD*



(d) *Secondary image, darker* ***(e)*** *Disparity map by SAD*

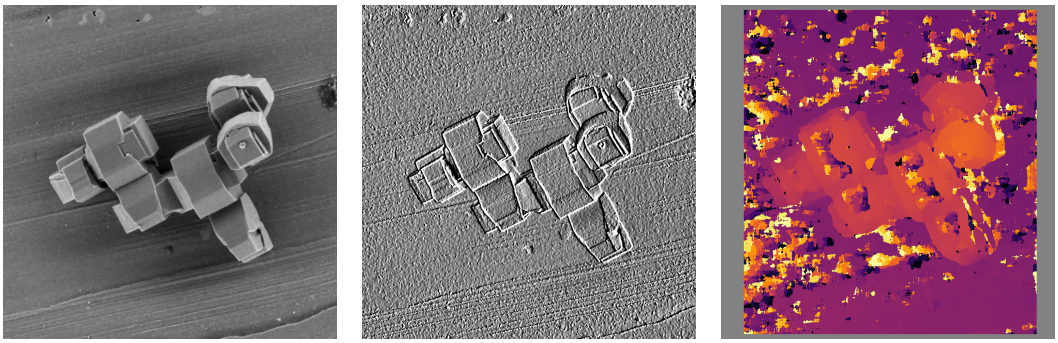
Figure 4.9: Disparity maps produced by the SAD metric with images of different brightness. *Size of SAD blocks: 31px.*

A first method consists of applying a convolution on the images with a kernel inspired from the x component of the Sobel [51] operator. The convolution kernel is shown in Figure 4.10. This kernel detects changes in gray values in the horizontal direction, independently of each pixel's illumination. Extreme values are capped so that all the values in the processed image are between $-cap$ and $+cap$, cap being a parameter; it enhances the illumination robustness as the match on important edges are then done according to their shape and not the relative difference in gray value. See processed image in 4.11b and SAD results in Figure 4.11c.

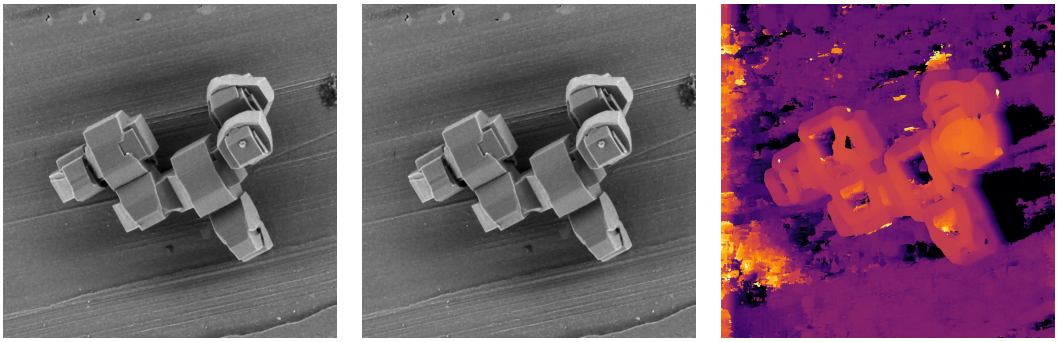
-1	0	1
-2	0	2
-1	0	1

Figure 4.10: X-Sobel convolution kernel.

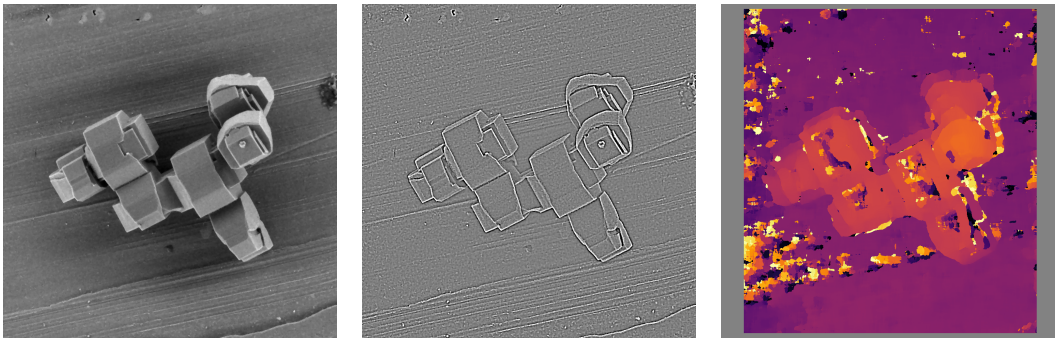
A second method is to normalize each pixel of the image with its neighborhood. Each pixel is subtracted to the mean of a block centered on it, the size of the block being a parameter. Extreme values are also capped so that all the values in the processed image are between $-cap$ and $+cap$, for the same reason as the previous method. See processed image in 4.11h and SAD results in Figure 4.11i.



(a) *Reference image* ***(b)*** *(a) + X-Sobel* ***(c)*** *SAD on (b)*



(d) *Reference image* ***(e)*** *No processing on (d)* ***(f)*** *SAD on (e)*



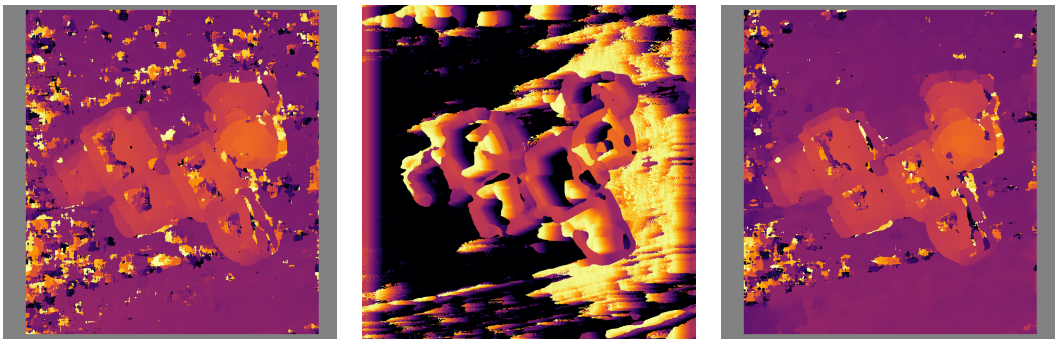
(g) *Reference image* ***(h)*** *(g) + normalization* ***(i)*** *SAD on (h)*

Figure 4.11: Effects of different preprocessings on the disparity maps produced by SAD. *Size of SAD blocks: 31px. Gray areas are undefined disparities; they are removed because they are too close from the border of the image and the matching search exceed the image's dimensions.*

Even though, as illustrated in Figure 4.12, the presented preprocessings make the SAD algorithm more robust to change in illumination, they are some negative consequences. Indeed, the scoring system doesn't take into account the gray level of each pixel, so if variations are similar, the algorithm can match two pixels that have a totally different intensity.

Census

Census is a different metric that SAD or SSD which is natively robust to illumination changes.



(a) X-Sobel *(b) No preprocessing* *(c) Normalization*

Figure 4.12: Results with and without preprocessing using a darker secondary image (See Figure 4.9). *Size of SAD blocks: 31px.*

The explanation that follows is illustrated in Figure 4.13. Compared blocks are first converted into bit strings using the census transform [52]. For achieving that, a block B_p centered on the point p with a gray intensity of g is transformed into a binary matrix B_p^*:

$$B_p^*(x,y) = \begin{cases} 0, \text{if } B_p(x,y) \leq g \\ 1, \text{otherwise} \end{cases}$$

All the elements inside B_p^*, except the center, are then concatenated inside a bit string S_p.

Two blocks are then compared using the Hamming distance [53] ; the resulting score is the number of differences between both strings.

$$s(p,q) = \text{number of differences between } S_p \text{ and } S_q$$

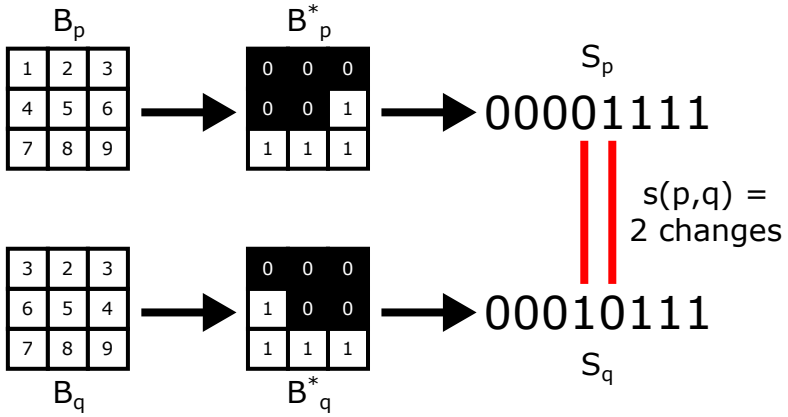
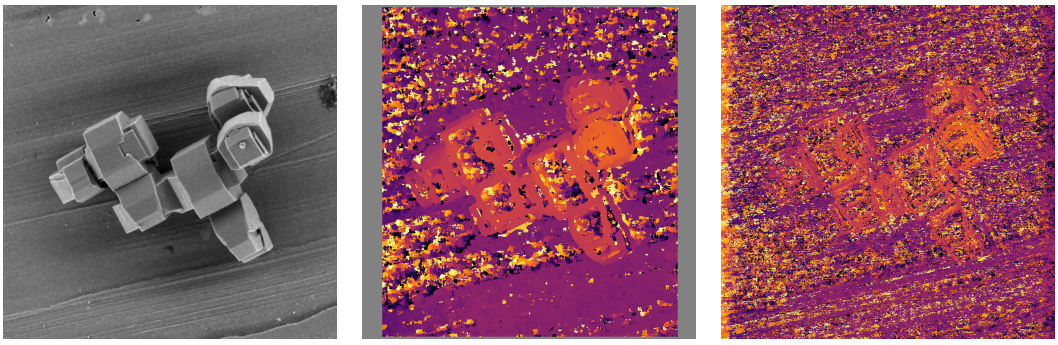


Figure 4.13: Illustration of the Census metric.

Results are shown in Figure 4.14. On our example, compared to the SAD metric with normalization as a pre-processing step, the results are more noisy. However, this metric is often used with semi-global methods (see Section 4.3).



(a) Reference image *(b) SAD + normalized* *(c) Census*

Figure 4.14: Comparison between the Census metric and the SAD metric with normalization pre-processing. *Size of blocks: 15px.*

MC-CNN

The MC-CNN method [54] has developed two pipelines that can be trained from a disparity map ground truth. The pipelines compare two 11 × 11 blocks and return a similarity score between 0 and 1 (for similarity scores, the higher the score the better the match). The first pipeline is faster and the second one is more accurate.

In a nutshell, both are computed in two steps (see Figure 4.15). The first step converts each block into a set of features. The second step estimates a similarity score from the two sets of features. The way each step is computed depends on the pipeline so we will describe both of them.

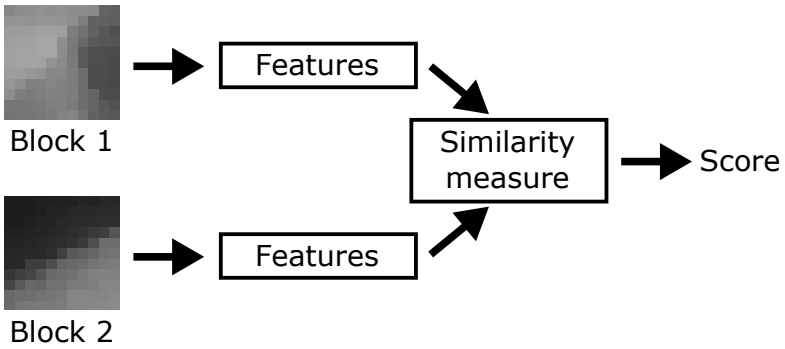


Figure 4.15: General pipeline of MC-CNN pipelines.

First, we will describe the **Fast** pipeline as the **Accurate** one is a variation of it.

The features of the two blocks are computed using the same convolutional neural networks [55]. Concretely, at the first step, for each block, 64 convolutions of size 3 × 3 are computed; the weights of each convolution are part of the parameters that are optimized in the training phase (explained later in this section). A bias is also added on each of the 64 resulting images, and they are then filtered using the ReLU function [56]:

$$f(x) = \max(0, x)$$

The 64 resulting images are the **first layer** of the convolutional neural network (see Figure 4.16).

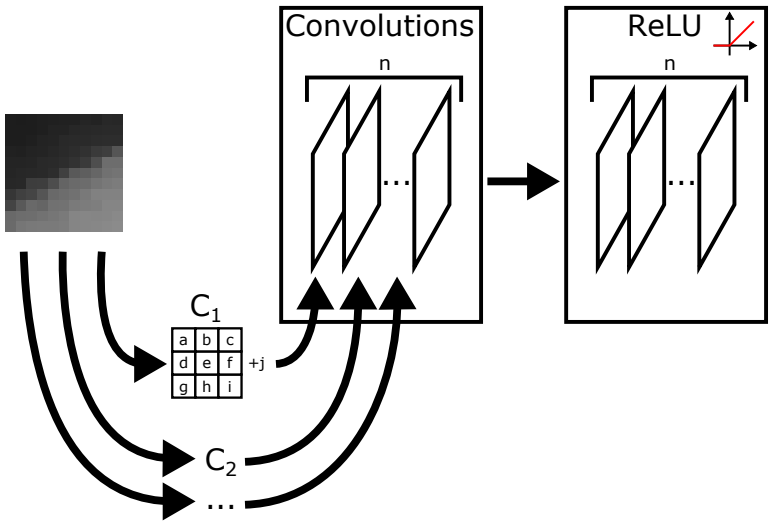


Figure 4.16: First layer of the convolutional neural network.

A second layer is computed from the first layer in a similar way, except this time the first layer is seen as an image with 64 channels (the results of the 64 convolutions). Therefore, the size of the applied convolutions is $3 \times 3 \times 64$. A similar process is used for computing a third, fourth and fifth layer. No ReLU function is applied at the end of the fifth layer (see Figure 4.17a).

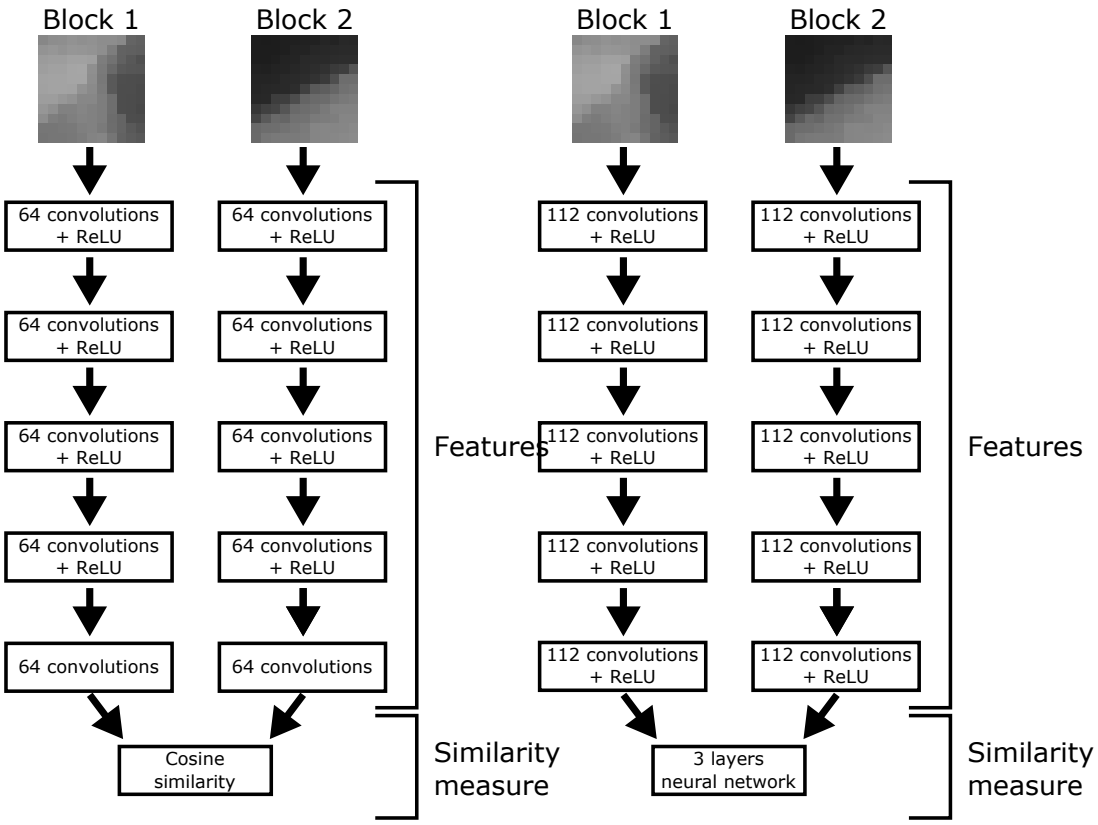
Each block produces therefore a $11 \times 11 \times 64$ vector. The similarity score is obtained by comparing the two vectors using the cosine similarity measure:

$$similarity = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

In this pipeline, we need to define 64 convolutions of size 3×3 (first layer) and 4×64 convolutions of size $3 \times 3 \times 64$ (second to fifth layer). There are therefore $64 \times 3 \times 3 + 4 \times 64 \times 3 \times 3 \times 64 = 148032$ parameters to define. There are also $5 \times 64 = 320$ parameters for the bias. We need to find the optimal values of all these parameters to make accurate matches.

For determining these parameters, a training set of pairs of blocks is generated using ground truth disparity maps.

Half of the training set is composed of pairs that should not match: if the block of the reference image is centered on (x, y), the block of the secondary image should be centered on $(x' + \alpha, y)$, x' being the position the block should have in the secondary image according to the ground truth disparity map, α being a random parameter between -6 and -1.5 or 1.5 and 6 for the **Fast** pipeline, and between -18 and -1.5 or 1.5 and 18 for the **Accurate** pipeline.



***(a)** Fast* ***(b)** Accurate*

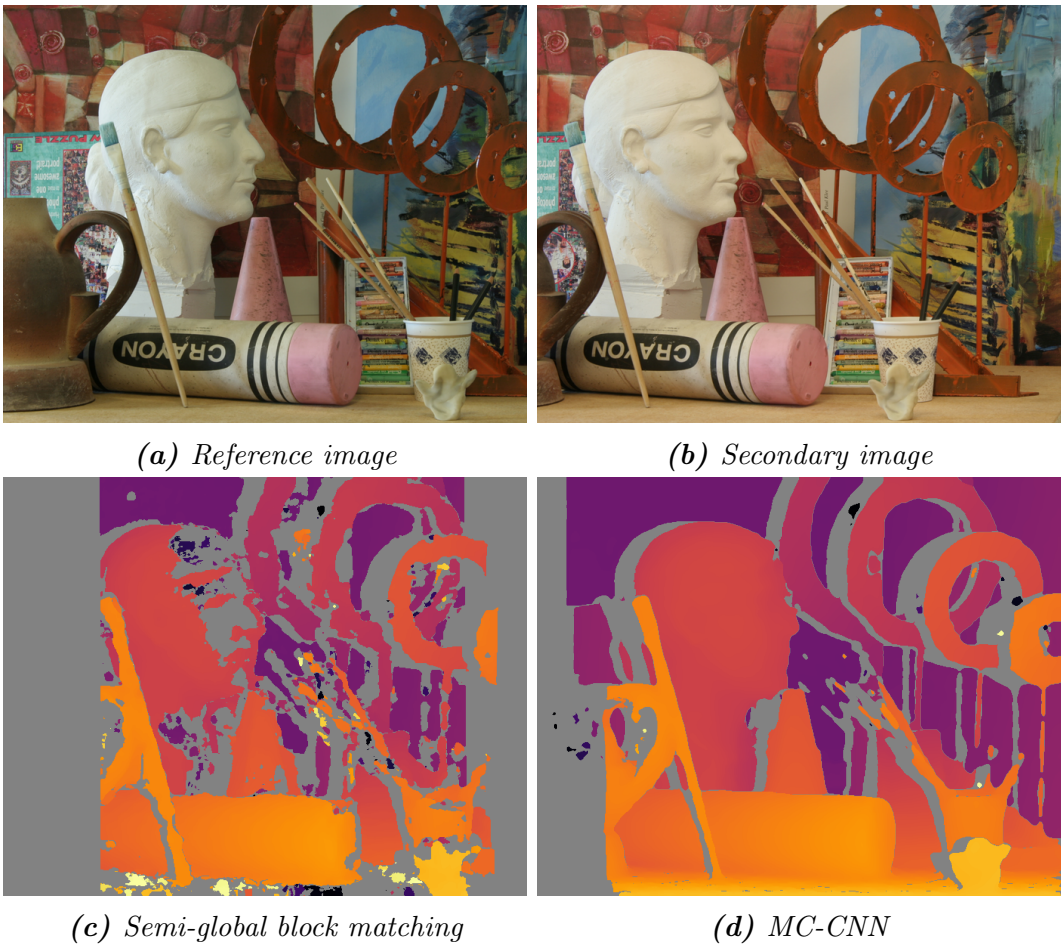
Figure 4.17: Pipeline of the Fast and Accurate MC-CNN metric.

Half of the training set is composed of pairs that should match: α here is a random parameter between -1 and 1. α is not set to 0 as we want to smooth the scores around good matches so that the cost aggregation methods used in post-processing perform better.

Using gradient descent, the parameters are optimized to make the pipeline correctly classify this set.

The **Accurate** pipeline is a variation of the **Fast** one. It computes its features similarly, but using more convolutions (112). However, instead of measuring the similarity between the features using a fixed metric (cosine), the **Accurate** pipeline also trains this measurement process using a 3 layers neural network [57].

There are also post-processing steps (cost aggregation, median filter, bilateral filter), and the available pipelines have been trained on the Middlebury dataset and not SEM images, so a direct comparison with other metrics is not possible. We show in Figure 4.18 a comparison between classical block matching with a similar post-processing and MC-CNN. As MC-CNN produces one of the best results, we used it to compare our solution with state of the art method on the Middlebury dataset. However, in order to adapt (or train) the neural network to our SEM images, a complete database of images associated to disparity map ground truths is required. Furthermore, it is computationally too expensive and requires an adapted GPU.



(a) Reference image *(b) Secondary image*

(c) Semi-global block matching *(d) MC-CNN*

Figure 4.18: Disparity produced by semi-global block matching and MC-CNN.

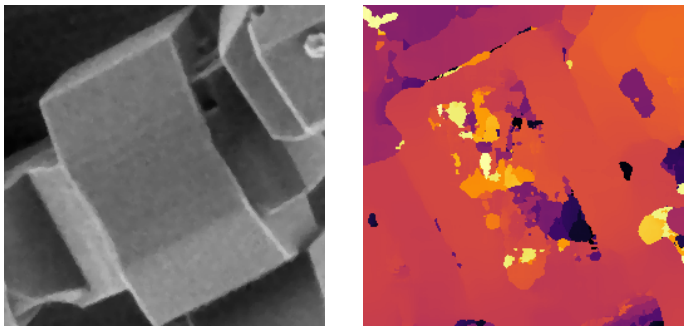
4.2.4 Limitations

Regardless of the used metric, the block matching algorithm has some shortcomings when evaluating disparity maps.

Low / repetitively textured regions

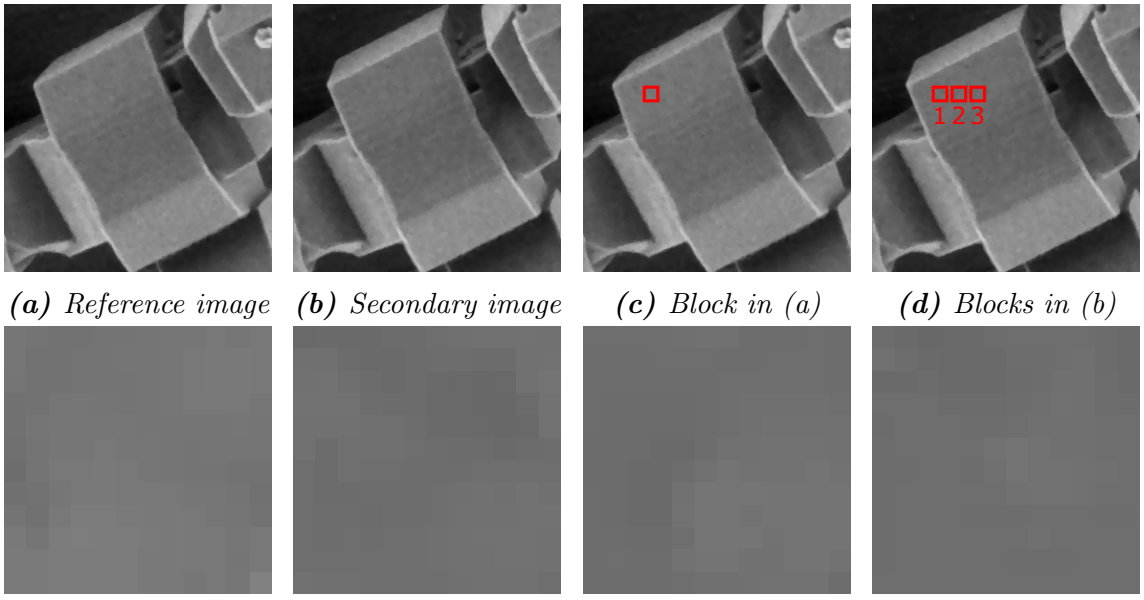
The most important issue is that low textured regions are not well handled. This issue is illustrated in Figure 4.19. The problem occurs because, on low texture regions, it can be impossible to accurately match blocks as there are a lot of possible similar matches (see Figure 4.20). Since there aren't any real detail in the block, the matching will occur on the noise in the blocks, leading to errors in the disparity estimation.

The problem is similar for repetitively textured regions: even if there is enough texture, if it is repetitive, the matching process is likely to get multiple plausible block candidates and could be unreliable. Though possible, this case is not very likely, especially in our SEM images.



(a) *Reference image* **(b)** *Disparity map*

Figure 4.19: Zoom-in on low textured regions.



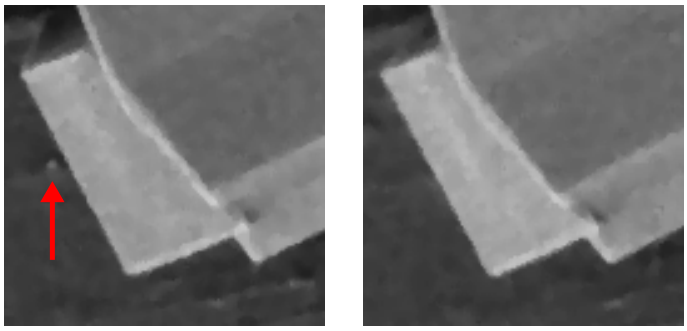
(a) *Reference image* **(b)** *Secondary image* **(c)** *Block in (a)* **(d)** *Blocks in (b)*

(e) *Ref. block* **(f)** *Sec. block 1* **(g)** *Sec. block 2* **(h)** *Sec. block 3*

Figure 4.20: Illustration of matching difficulties in low textured regions.
Ref.: Reference image. Sec.: Secondary image. We selected one block in the reference image and three neighboring blocks in the secondary image. Finding the correct block is very difficult as all blocks look similar. The match will probably fail because the matching process will occur on noise and not real image details.

Occlusions

If an object partially occludes another one in the reference image, it is probable that the occluded region differs in the secondary image as it is taken from a different view point (see figures 4.21 and 4.22). This challenges the matching process. For instance, there are no correct match on regions that are visible in the reference image and occluded in the secondary image.



(a) *Reference image* ***(b)*** *Secondary image*

Figure 4.21: Occlusion illustration. *The dot shown in the reference image goes behind the object in the front in the secondary image.*

Fattening effect

Another problem of the block matching algorithm is that the larger the block size are, the more correlated the disparity of two neighboring pixels will be. Indeed, let's suppose that we want to estimate the disparity of two pixels (x, y) and $(x + 1, y)$. If we estimate their disparity using $s \times s$ blocks, the two blocks overlap each other on $(s-1) \times s$ pixels. For instance, if we evaluate the disparity map using 9×9 blocks, the blocks of two neighboring pixels will overlap on 89% of their area. If we use 31×31 blocks, they will overlap on 97% of their area.

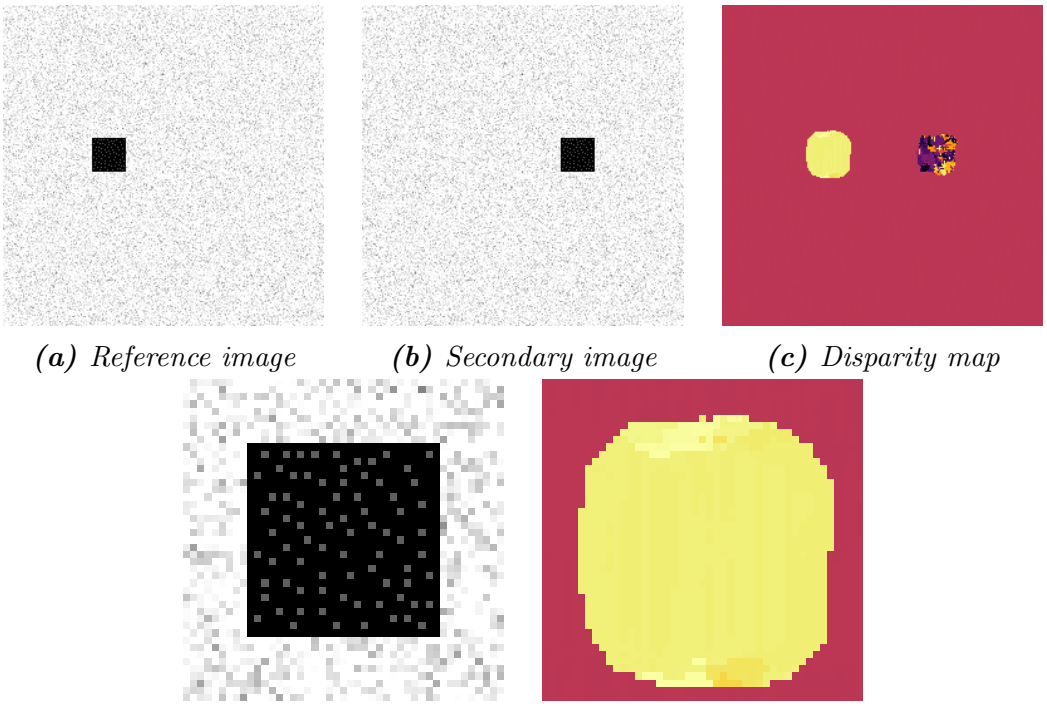
This triggers several negative effects. First, if the disparity of a pixel is not well estimated, neighboring pixels are likely to be erroneous also. Secondly, around strong edges, neighboring pixels whose blocks overlap with the edge are likely to inherit the disparity of the edge. This effect is called the fattening effect, because the most visible consequence is that objects in the foreground tend to be fattened as shown in Figure 4.22.

4.2.5 Confidence and consistency measures

We have seen that the disparity estimation on some pixels can be unreliable. It is therefore important to determine which pixels are likely to be misevaluated in order to correct them or filter them out. These confidence measures are complementary, and we will combine them in Chapter 6.

Texture threshold

As most problem occur on low-textured areas, a straightforward solution is to use texture as a confidence value. An example of confidence measure would be the morphological gradient of the reference image. Pixels with a high gradient are likely to be part of an important edge making the match highly reliable (see Figure 4.23c).



(a) Reference image *(b) Secondary image* *(c) Disparity map*

(d) Zoom on (a) *(e) Zoom on (c)*

Figure 4.22: Fattening and occlusion illustration. *In this artificial example, only the black square moved between the reference image and the secondary image. Pixels that are visible in the reference image but not in the secondary image are misevaluated. The black square is also fattened (see (d) and (e)). The block size chosen in* 5×5.

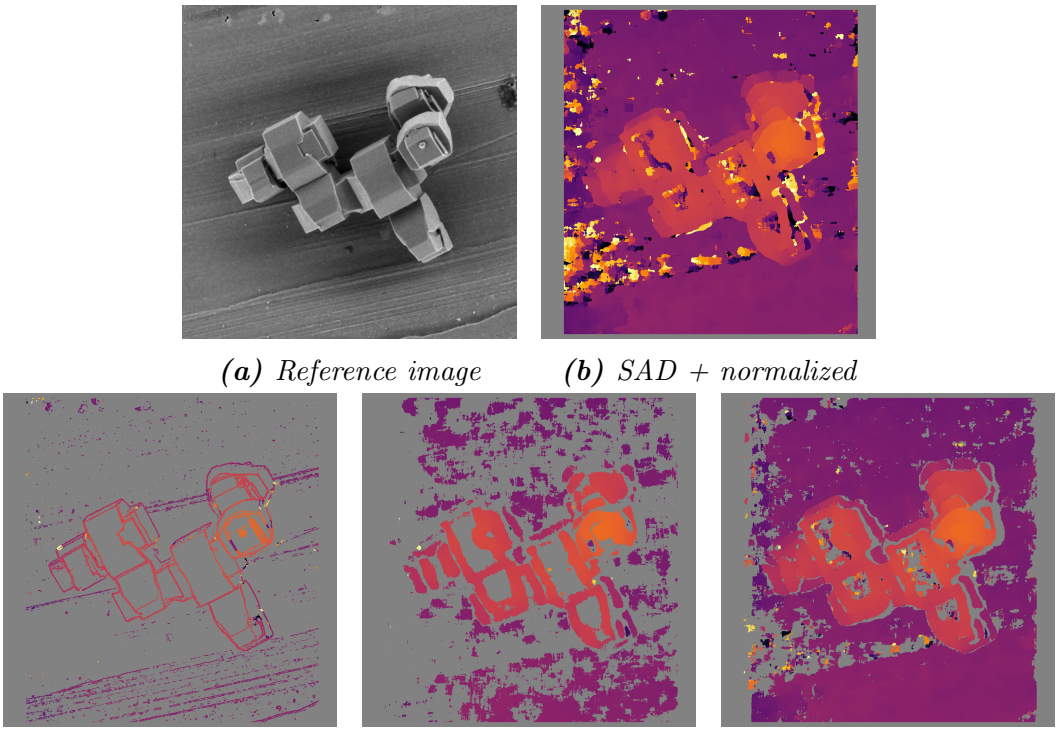
This simple gradient measure would however fail on repetitive textures, as in this case there is a high gradient but multiple plausible matches. A possible solution would then be to use autocorrelation or a variation of the Harris detector [58], but limited to the x-axis. However, as these cases are likely to be handled by the next measure, we will rely on the gradient which is less computationally expensive.

Uniqueness ratio

The uniqueness ratio takes advantage of the matching scores computed during the disparity map estimation. Indeed, for each pixel, multiple disparity values are tested and each test produce a matching score. If the retained disparity value is the one with the best score, we can also look at the second best score and compute its difference to the best score. If this difference is too low, it means that there is at least another very plausible candidate, making the match unreliable.

In the OpenCV library [13], the uniqueness ratio is computed as follow (we consider that the better the match, the lower the score):

$$\text{uniqueness} = \left(\frac{\text{second best score}}{\text{best score}} - 1\right) \times 100$$



(a) *Reference image* **(b)** *SAD + normalized*

(c) *(b) + gradient > 20* **(d)** *(b) + UR > 10* **(e)** *(b) + LRC < 1.5*

Figure 4.23: Confidence metrics. *UR: Uniqueness ratio. LRC: Left-Right Consistency check. We retain only pixels whose confidence is over a threshold. Size of blocks: 31px.*

As for the texture threshold, we can remove pixels with low uniqueness ratio (see Figure 4.23d). For a similar accuracy, thresholding the uniqueness ratio retains much more pixels than thresholding the gradient. However, thresholding the gradient better handles the fattening effect as pixels around edges are removed.

Left-right consistency check

The left-right consistency check [59] requires the **RS** disparity map from the reference (or left) image to the secondary (or right) image as well as the **SR** disparity map from the secondary image to the reference image.

The rationale behind this check is that, if the **RS** disparity map matches a point (x, y) in the reference image to (x', y) in the secondary image, then the **SR** should match (x', y) in the secondary image at or near (x, y) in the reference image. If that is not the case, the estimated disparity is inconsistent and should be removed.

A LRC map is computed as follow. Let's suppose that the RS disparity map matches (x, y) with (x', y), and that the SR disparity map matches (x', y) with (x'', y). The LRC value of (x, y) will be $|x - x''|$.

Pixels with LRC values greater than 1.5 are generally deemed inconsistent and removed (see results in Figure 4.23e). Though the check achieves to remove a lot of noise, it is best known for removing problems caused by occlusions.

4.3 Semi-global and global methods

Up until now, we have described a local approach to estimate disparity maps. Each pixel is assigned the disparity corresponding to the best block match, independently of the neighboring pixels outside the block. We have seen that this approach doesn't work well on low textured areas, as there are multiple possible matches with very similar scores. Increasing the size of the blocks can alleviate these problems, but it comes with its own set of challenges as the fattening effect.

Several solutions take a more global approach. Instead of simply choosing the disparity with the best score for each pixel, these methods try to choose the disparity that optimizes a trade-off between the matching score and disparity stability between neighboring pixels. Concretely, these methods try to minimize the following energy equation:

$$e = \sum_{\forall (x,y) \in f} (\text{matchingCost}(x, y) + \text{smoothingCost}(x, y))$$

f being the reference image, matchingCost being the block matching score between the reference and secondary images according to the chosen disparity (we suppose here that a lower score means a better match), smoothingCost being the term that ensures disparity stability between neighboring pixels. The general form of smoothingCost is:

$$\text{smoothingCost}(x, y) = \sum_{\forall (x',y') \in \mathcal{N}_{(x,y)}} V(d(x, y), d(x', y'))$$

$\mathcal{N}_{(x,y)}$ being a set of neighboring pixels to (x, y), $d(x, y)$ being the disparity value of (x, y), and $V(d_1, d_2)$ being the smoothness term penalizing differences between d_1 and d_2. V can also depend on the images: for instance, near edges, this penalization term can be attenuated for preventing unnecessary smoothing near two objects' boundaries.

Though minimizing this energy function is a NP-hard problem, several methods manage to approximate the minimum. Graph cuts [60] [61], belief propagation [62] and tree-reweighted message passing [63] [64] are among the most used algorithms for achieving this goal [65]. However, they are too slow and computationally expensive for our use case.

Alternative and faster algorithms produce reasonable results [66] [67] [68] [69] [70] [71]. We will mainly use, through the OpenCV library [13], a variant of the semiglobal matching (SGM) algorithm [67], one of the most popular algorithms [72] [73].

In the SGM algorithm, the standard way to compute V is:

$$V(d_1, d_2) = \begin{cases} 0, \text{if } d_1 = d_2 \\ P1, \text{if } |d_1 - d_2| = 1 \\ P2, \text{otherwise} \end{cases}$$

$P1$ and $P2$ being constants. $P2$ can also vary so that disparity discontinuities are aligned with the discontinuities in the image [67].

For minimizing the energy function, the problem is transformed into a set of 1-D problems. Indeed, the pixels in the image can be seen as a set of lines called **scan lines** in different directions: vertical lines, horizontal lines, and oblique lines. We can run the algorithm on 4 directions or 8 directions. See Figure 4.24.

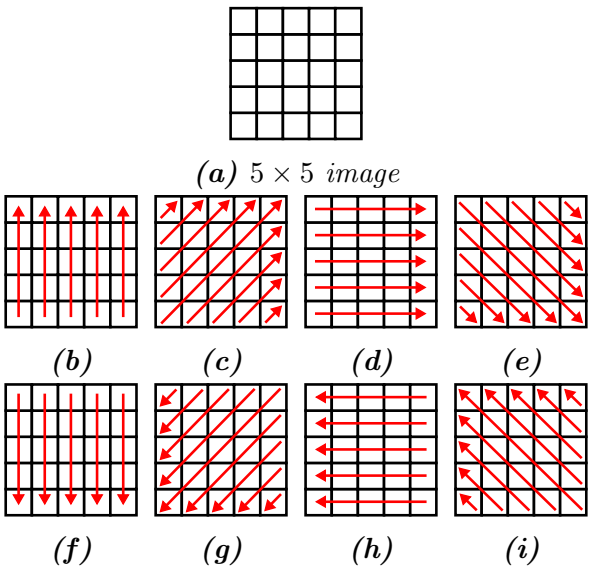


Figure 4.24: SGM scan-lines. *(f), (g), (h), (i) are used if we run the algorithm with 8 directions.*

For all (r, p, d), a cost matrix $L_r(p, d)$ is computed, r being a direction, p being a point in the image, d being a possible disparity value. This cost matrix is computed from the edges of the image using the following recursive equation:

$$L_r(p, d) = s_p(d) + \min_{\forall d' \in D} (L_r(p - v_r, d') + V(d, d'))$$

$s_p(d)$ being the matching cost of the point p with the disparity d, D being all possible disparity values, v_r being the unit vector of the direction r (one pixel displacement).

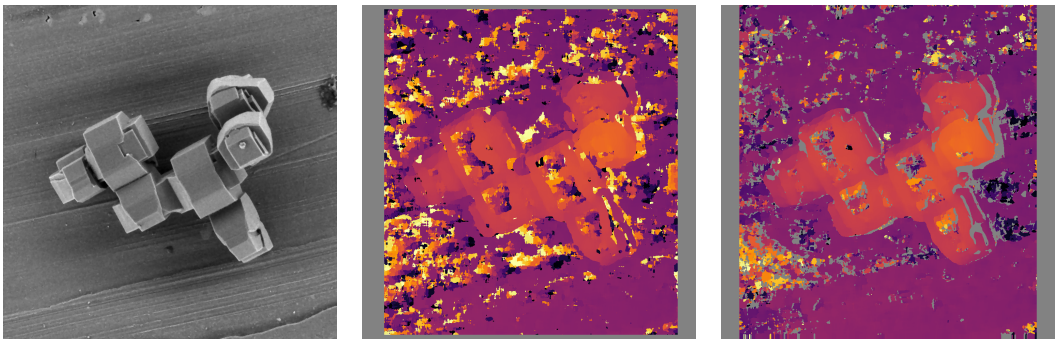
The costs of all the directions are then aggregated in the following cost matrix:

$$S(p, d) = \sum_r L_r(p, d)$$

Finally, the disparity chosen for each pixel p is the one that minimize $S(p, d)$.

The variant implemented by the OpenCV library, Semi-Global Block Matching (SGBM), also adds a post-filtering step - quadratic interpolation - to get subpixel values. The $s_p(d)$ cost are obtained using SAD with the image prefiltered using X-sobel.

A comparison between the Block Matching and SGBM algorithms is shown in Figure 4.25. The results of SGBM are much less noisy, especially on low textured areas.



(a) Reference image *(b) Block Matching* *(c) SGBM*

Figure 4.25: Comparison between the disparity map estimated using Block Matching (X-sobel) and the one estimated using SGBM. *Size of SAD blocks: 27px.*

4.4 Disparity map post-processing

If global or semi-global methods change the way the disparity map is estimated by checking the matching scores of neighboring pixels, other methods take a produced disparity map and try to refine it [74] [75] [76] [77] [78]. The common methodology is to first remove low confidence disparity values, and then propagate the high confidence values. This methodology is generalist and can be applied after any disparity map estimation algorithm - including global and semi-global methods.

In this section, we will apply the post-processing methods on the example shown in Figure 4.26d.

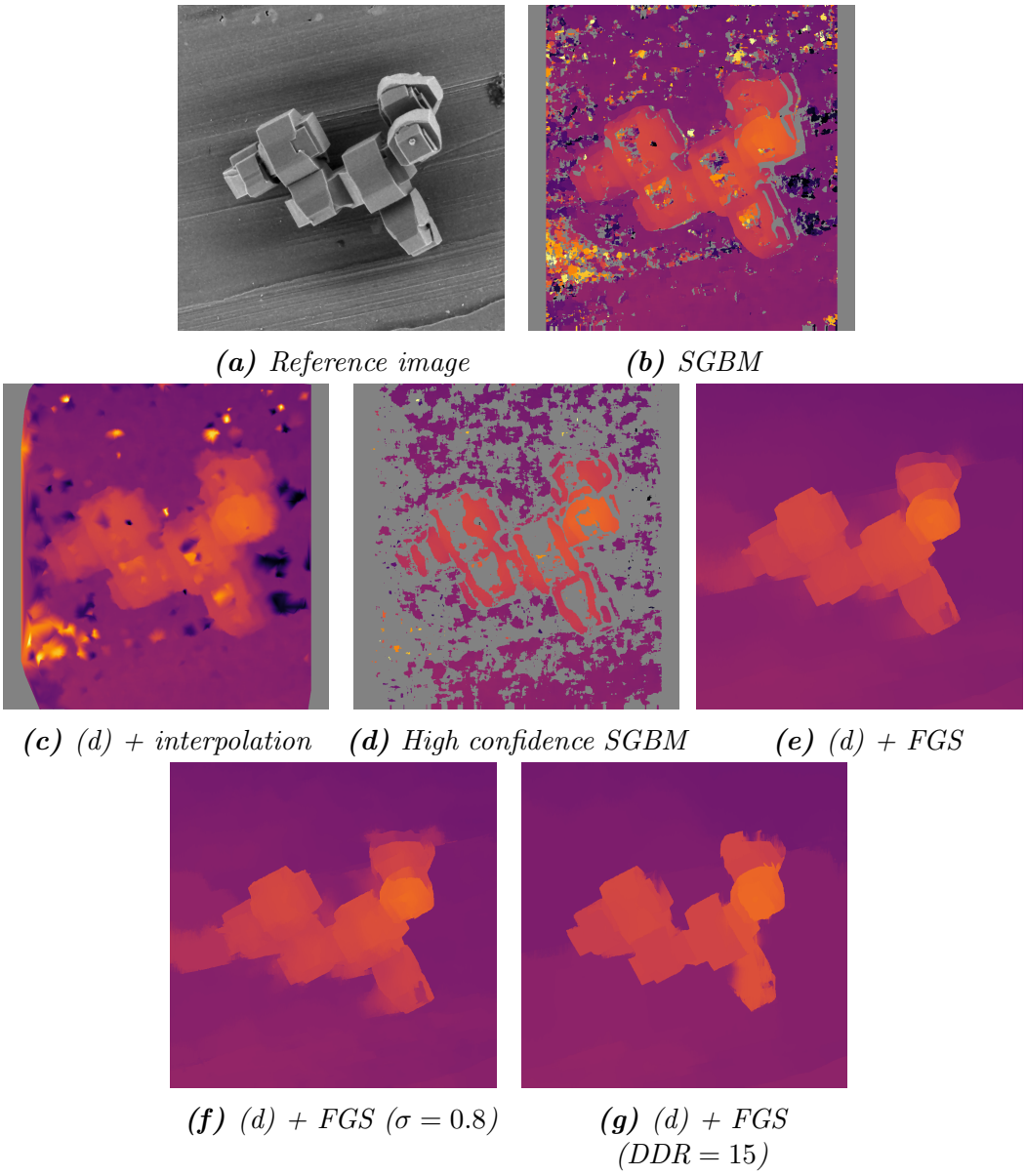
Simple techniques such as interpolation [79] or diffusion [80] don't produce satisfying results. First, remaining errors are likely to be propagated and exacerbated. Secondly, removed values are often located on occlusions, that is to say between two objects with different disparity values: on these areas, the disparity values should be filled by the occluded object and not a mix of both objects. See Figure 4.26c.

To counter these problems, common approaches propagate high confidence disparity values while respecting the edges in the reference image [77] [78]. They try to minimize the following energy function:

$$e = \sum_{\forall (x,y) \in f} (\text{offsetCost}(x,y) + \text{smoothingCost}(x,y))$$

f being the reference image, offsetCost being a term that measure the difference between the original and the resulting disparity maps on the high confidence points, smoothingCost being a term that ensures disparity stability on low texture areas.

Specifically, for the Fast Global Smoothing Filter [77] (FGS):



(a) *Reference image* ***(b)*** *SGBM*

(c) *(d) + interpolation* ***(d)*** *High confidence SGBM* ***(e)*** *(d) + FGS*

(f) *(d) + FGS* ($\sigma = 0.8$) ***(g)*** *(d) + FGS* ($DDR = 15$)

Figure 4.26: High confidence SGBM and disparity refinement methods. *Size of SAD blocks: 27px. The high confidence SGBM is obtained from the same parameters than (b) but additionally with uniqueness ratio set to 10 and a LRC check of 1.5. For FGS, unless specified otherwise,* $\lambda = 8000$, $\sigma = 1.5$ *and* $DDR = 7$.

$$\text{offsetCost}(x,y) = \begin{cases} (d(x,y) - d_f(x,y))^2, \text{if } (x,y) \text{ high confidence point} \\ 0, \text{otherwise} \end{cases}$$

$$\text{smoothingCost}(x,y) = \lambda \sum_{(x',y') \in N_4(x,y)} \text{colorDiff}((x,y),(x',y')) \times (d_f(x,y) - d_f(x',y'))^2$$

$d(x,y)$ being the inital disparity map, $d_f(x,y)$ being the resulting disparity map, λ being a parameter to be defined, $N_4(x,y)$ being the set of the four direct neighbors to (x,y). Finally $\text{colorDiff}((x,y),(x',y'))$ is a function that decreases as the color difference between (x,y) and (x',y') increases:

$$\text{colorDiff}((x,y),(x',y')) = \exp(\frac{-||f(x,y) - f(x',y')||}{\sigma})$$

σ being a parameter to be defined.

Results of the FGS algorithm are shown in Figure 4.26e. The disparity map is visually better as most noise have been removed. However, there is an overflow effect near edges.

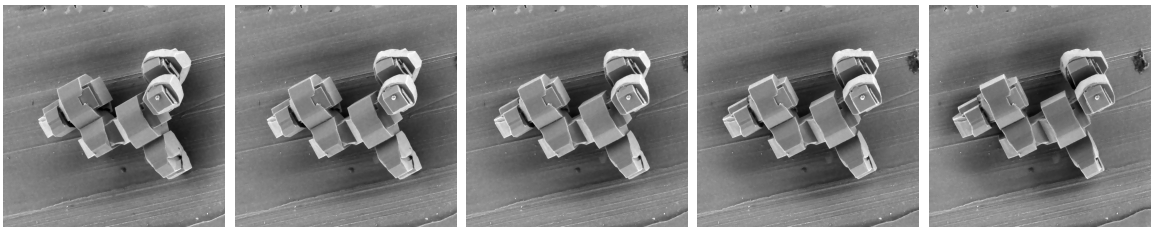
This effect is due, in part, to the low gradient of some edges: the disparity values of the two objects separated by the edge are smoothed though this is inappropriate. A solution is too choose a lower σ value, but in that case there is more noise. See Figure 4.26f.

Another cause is the fattening effect: some pixels on background objects inherit the disparity values of nearby foreground objects. In the background objects, correct and incorrect values are generally separated by low confidence values because those points are occluded in the secondary image. Thus, to counter the fattening effect, values at a distance near low confidence points and disparity discontinuities are removed. The DDR parameter (DDR stands for Depth Discontinuity Radius) defines the distance, in pixels, at which values are removed.

Choosing an appropriate DDR radius is difficult since the results' accuracy will drop if too many values are removed (see Figure 4.26g).

4.5 Multiview

Methods we have presented until now estimate a height map from a pair of images. However, we are able to take multiple images of the sample from different angles. We have estimated that we are able to capture up to 5 images of a sample in a reasonable timeframe (see Figure 4.27). MultiView Stereo reconstruction (MVS) methods allow us to use this additional information to improve the accuracy of our estimations.



(a) $-10°$ ***(b)*** $-5°$ ***(c)*** $0°$ ***(d)*** $+5°$ ***(e)*** $+10°$

Figure 4.27: Sequence of images of a sample taken at different angles.

Though there is a wide range of multiview stereo reconstruction methods, all of them are not adapted to our use. For instance, methods such as [81], [82] or [83] rely on the detection of feature points using methods such as SIFT [84]: on our images, as most areas are textureless, there isn't enough points that can be matched for an accurate reconstruction (see Figure 4.28).

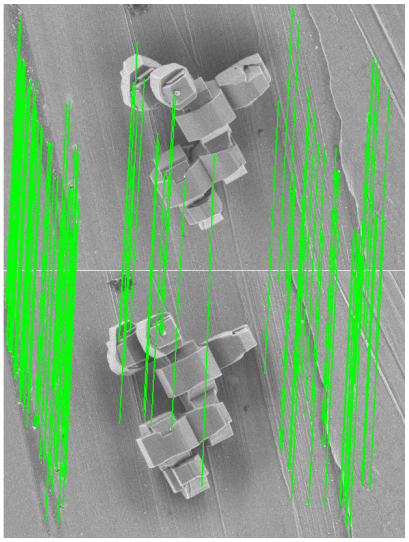


Figure 4.28: Feature points detected by Visual SFM [83] and matched between two images of the same sample.

We want a MVS method that satisfies the following criteria:

- It should be able to process 2 to 5 images. This is a relatively low number of images compared to standard datasets such as the Middlebury multi-view dataset [85] where the sparsest samples contain 16 images.
- The acquisitions won't fully cover the sample, unlike the Middlebury multi-view dataset. Acquisition tilt will vary from $-10°$ to $+10°$.
- Results should not be too sparse even on low textured areas.
- As we use fast stereo methods we don't have access to the matching scores.
- It should be reasonably fast (less than 30 seconds)

[86] reviews recent MVS methods. The review categorizes 4 families of representation: depth maps (equivalent to height maps), point-clouds, volume scalar-fields and meshes. Our acquisition tilts are restrained, so the most adapted representation is the height map.

As we want to be able to process a variable number of images in a limited amount of time, we need a simple and extendable approach.

If we have n images, we can choose one image as a reference image and $n-1$ images as secondary images. We have therefore $n-1$ pairs of images. For each pixel of the reference image and for each pair, we are able to plot a graph depicting the matching score vs disparity. Using geometry, we can transform this graph into matching score vs height. We will call $s_i(h)$ the matching score of the pair i at the height h. The objective is therefore to estimate a final height h_f from all $s_i(h)$.

For doing that, **Winner-Takes-All** is one of the simplest approaches, as used in [87]. For each point, the retained height is the one that have the best score:

$$h_f = \arg\min_h(s_i(h))$$

There are other methods that combine the different $s_i(h)$ function in a more robust way such as [88] or [89].

An even simpler approach is to only consider the height maps produced between the reference and the $n-1$ secondary images (see Figure 4.29). For each pixel (x, y), we have therefore multiple height evaluations $(h_1, h_2, ..., h_m)$, m being the number of defined evaluations. From there, the final height h_f can be computed using standard statistical approaches. As there might be errors or outliers, more robust methods than the average are needed. For instance :

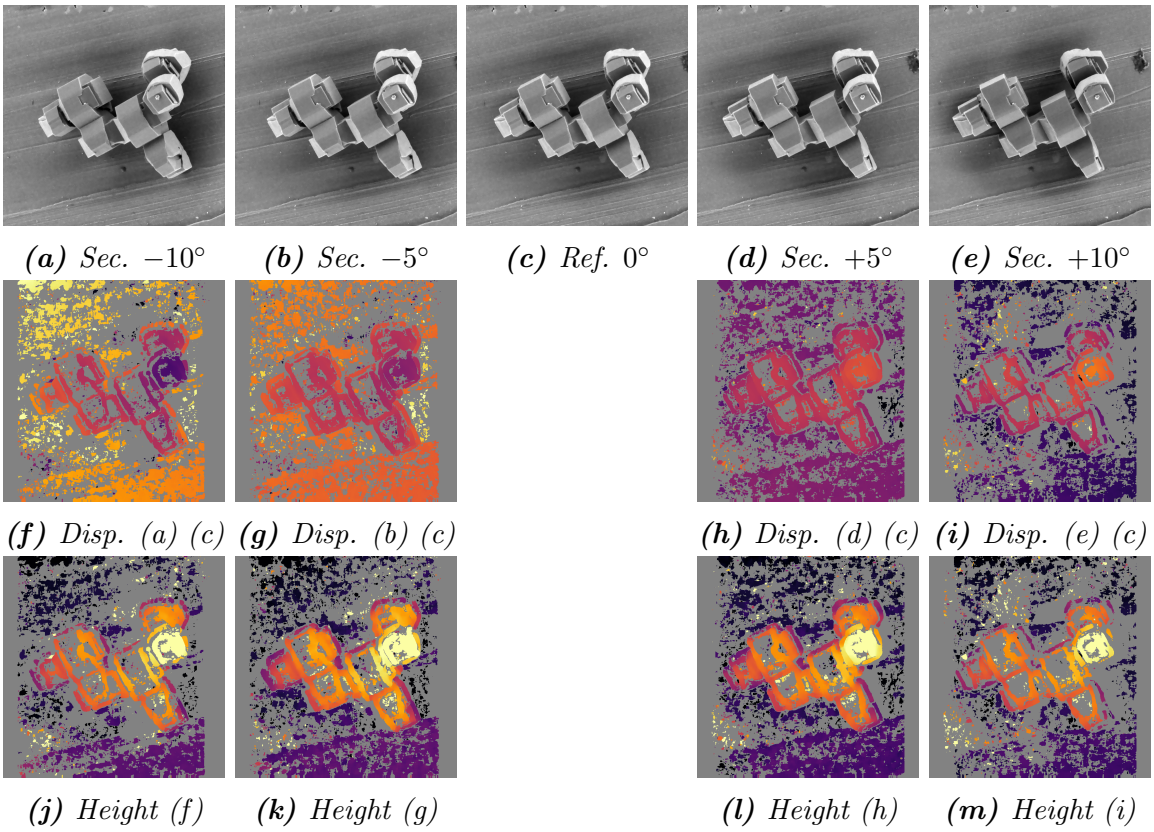
$$h_f = \text{median}(h_1, h_2, ..., h_m)$$

Results are shown in Figure 4.30c. This approach, although less robust, doesn't require any matching score, meeting the criteria we stated earlier. Without any post-filtering, the method produces a more complete but more erroneous height map. These errors especially appear on low textured areas, where only one or two pairs have estimated an height on these pixels.

We can therefore create a confidence map indicating for each pixel the number of defined evaluations m (see Figure 4.30b). We can also create another confidence value being the difference between the maximum and the minimum of the set $(h_1, h_2, ..., h_m)$. When retaining only high confidence values, most errors are removed (see Figure 4.30e).

4.6 Error quantification

As the 3D topography estimations of stereo methods will be used to infer physical and chemical properties, it is important to move beyond qualitative analysis and quantitatively measure their accuracy.



(a) *Sec. −10°* ***(b)*** *Sec. −5°* ***(c)*** *Ref. 0°* ***(d)*** *Sec. +5°* ***(e)*** *Sec. +10°*

(f) *Disp. (a) (c)* ***(g)*** *Disp. (b) (c)* ***(h)*** *Disp. (d) (c)* ***(i)*** *Disp. (e) (c)*

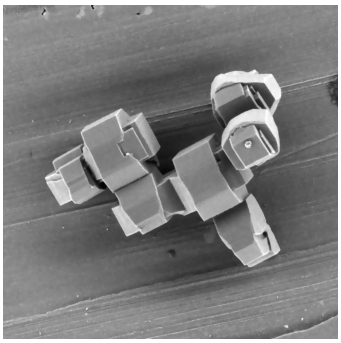
(j) *Height (f)* ***(k)*** *Height (g)* ***(l)*** *Height (h)* ***(m)*** *Height (i)*

Figure 4.29: Disparity map and height map estimations for each pair of a sequence. *Sec.: Secondary image. Ref.: Reference image. Disp. (1) (2): Disparity map between (1) and (2). Height (1): height map from (1). SGBM algorithm. Block size: 21px. Uniqueness ratio: 5.*

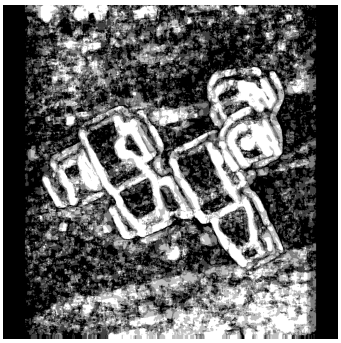
Our first approach was to project, using the disparity map, the reference image on the secondary image: we move each pixel of the reference image according to their disparity values, and compare the projection to the secondary image. If it can be used to check the overall quality of the disparity map, this test can be misleading as very inaccurate disparity maps can produce good projections (see Figure 4.31). Indeed, inside homogeneous areas, there won't any differences in the projected image between pixels with a correct match and those with an incorrect match: the gray level will be the same as those pixels are inside an homogenous area.

A second possibility would be to use the relationship between the gray intensities and the topographic properties of the sample as described in Chapter 2. However, as shown in Section 2.5, this relationship is complex and difficult to model. Furthermore, even using secondary electrons, the physical properties of the scanned materials can also influence the intensities. Any quantitative metric would therefore bring limited input on the quality of the reconstruction.

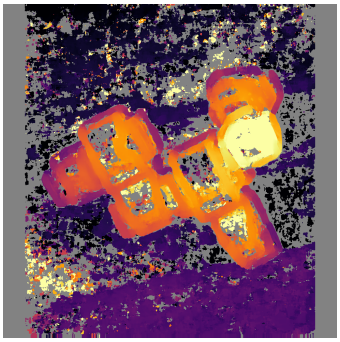
The ideal is to have a set of samples associated with ground truths that we can compare the height map estimations with. Multiple databases exist [90] [91], though not for SEM images.



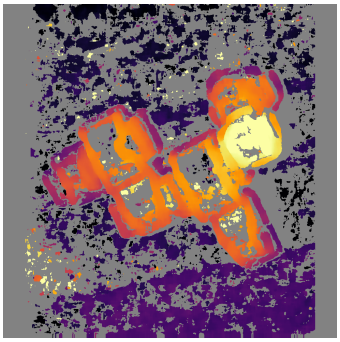
(a) Reference image



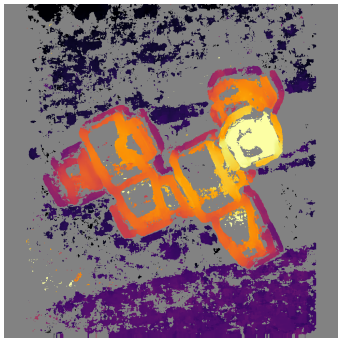
(b) m confidence



(c) Combined height map

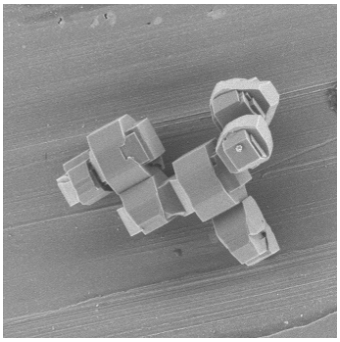


(d) Height map from pair

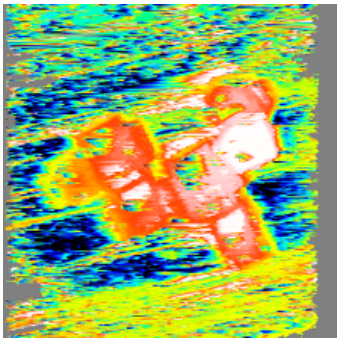


(e) (c) only high confidence

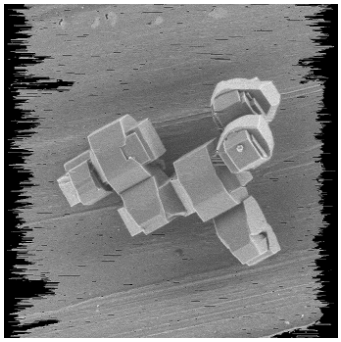
Figure 4.30: Height map fusion. *The final height maps have been combined from Figure 4.29. High confidence pixels are those with $m \geq 2$.*



(a) Secondary image



(b) Disparity map



(c) Projected reference image

Figure 4.31: A bad disparity map can lead to a good reference image projection.

One of the most popular is the Middlebury dataset [90], supported by Middlebury College (Vermont, USA), Microsoft Research and NSF. It provides rectified pairs of images of static indoor scenes with disparity map ground truths (an example is shown in Figure 4.32). The official website ranks the disparity maps of state of the art methods on multiple criteria. Two of the most used metrics are the average error and the percentage of pixels whose error is greater than 2px.

(a) Reference image *(b) Secondary image* *(c) Ground truth*

Figure 4.32: Example of Middlebury pair with disparity map ground truth.

Using the Middlebury dataset can facilitate the evaluation of the overall methodology and its comparison with other stereo methods. However, images are very different from our SEM images; the Middlebury dataset contains color images, their resolution are much larger, the noise is different, the objects and shapes are different.

It would therefore be better to generate a dataset with SEM images. However, generating relevant ground truths is not simple as we can't rely on alternative topography acquisition systems such as LiDAR [91].

A first solution is to use calibration samples with known topographies. An example is shown in Figure 4.33. The problem is that the shapes are generally simpler and different from the shapes we need to analyze.

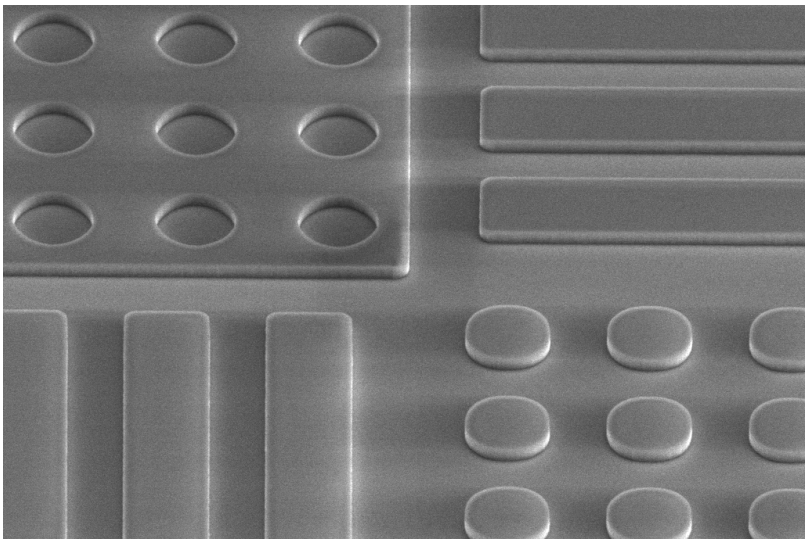


Figure 4.33: Calibration sample. *TGXYZ calibration grating [92].*

A second solution is to determine manually the disparity maps / height maps of the samples. However, this is time-consuming, limiting the number of samples in the dataset, and more prone to errors.

A general comment we can formulate about stereo datasets - whether they are produced with classical or SEM images - is that they generally contain too few examples. The consequence is that it is difficult to know if the evaluated method will have a similar performance on other images or if they just perform well on the dataset.

Hence the idea of simulating SEM images; in these cases, we are able to choose the 3D model we want to display, so we know its topography and can compare our estimations with it. There are two possible approaches for producing these simulated images.

Either we use a process that simulates the path of each electron according to the laws of physics. This **strong simulation** produces realistic images, but is very slow and only simple models - far from the complex geometrical shapes we have to estimate - can be rendered at low resolution. See Figure 4.34.

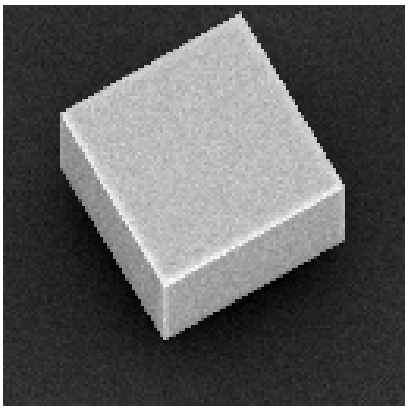
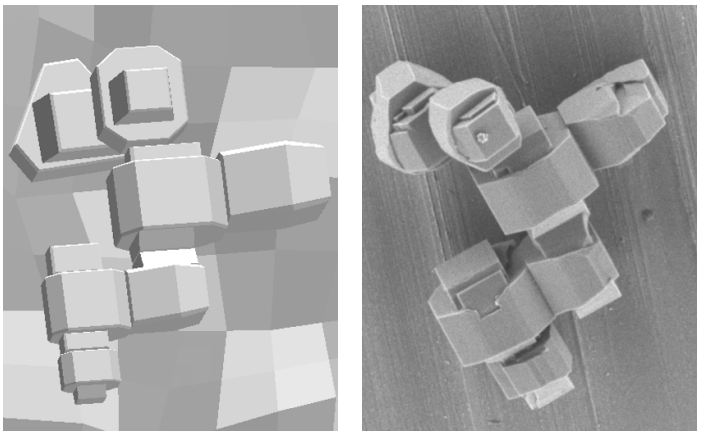


Figure 4.34: Strong simulation of a cube. *This image was simulated by the Monte-Carlo method using an home-made software based on the general purpose package PENELOPE [93] [94]. 10000 primary electrons trajectories were simulated on each pixel. Electrons reaching a virtual annular detector of 1cm inner radius and 5cm outer radius placed 1cm above the z=0 plane were counted to produce the image.*

Either we use classical 3D engines to render objects using shaders that approximate SEM images. An example of such a shader can be the Lambertian reflectance [95]. This **weak simulation** produces less realistic images, but is much faster - allowing us to produce a lot of images - and we can use models that are similar to the samples we have to analyze. An example of weak simulation is shown in Figure 4.35.



(a) *Weak simulation* ***(b)*** *Real sample*

Figure 4.35: Weak simulation vs real sample. *The model was rendered using Unity 3D [96].*

In a nutshell, there is no ideal error quantification method. A recap of advantages and drawbacks of the listed methods is shown in Table 4.1.

Method	Is reliable ?	High precision ?	Similar shapes ?	Similar rendering ?	Easy comparison ?	High variety ?
Comparison between Sec. and proj. ref. images	✗	✗	✓	✓	✗	N/A
Relationship between gray intensities and 3D	✗	✗	✓	✓	✗	N/A
Ground truth / Middlebury dataset	✓	✓	✗	✗	✓	✗
Ground truth / SEM calibration samples	✓	✓	✗	✓	✗	✗
Ground truth / Estimated manually (SEM)	✓	✗	✓	✓	✗	✗
Ground truth / Strong simulation	✓	✓	✗	✓	✗	✗
Ground truth / Weak simulation	✓	✓	✓	✗	✗	✓

Table 4.1: Recap of the advantages and drawbacks of the different error quantification methods.
Is reliable? → Does a great score necessary means a great estimation?
High precision? → Is the margin of error low?
Similar shapes? → Are the shapes in the images similar to the shapes of target samples?
Similar rendering? → Do the gray levels in the images follow similar rules than in SEM images?
Easy comparison? → Are we able to easily compare developed methods to state of the art methods?
High variety? → Are we able to produce a sufficiently high variety of examples to be confident that most cases are covered?
N/A: Not applicable.
Sec. and proj. ref. images: Secondary and projected reference images.
The two first methods do not require ground truth

4.7 Conclusion

We have described in this chapter the main stereo methods used to evaluate the topography of objects. These methods estimate a 3D map from a set of images in two steps. First, they estimate a disparity map which evaluates for each pixel of a reference image its displacement in a secondary image. Then, using geometry, they transform the disparity map into a height map.

Estimating the disparity map is however a challenging task: low-textured areas and occlusions are often-misevaluated because the displacement of those areas are difficult to assess. We have presented several methods allowing to remove most of these erroneous values, as well as methods that propagate high confidence values.

However, these methods don't produce results that are accurate enough for our use cases. The following chapters present some solutions we developed to improve their performance.

Part II

Proposed solutions and results

Chapter 5

Top-down segmented matching

French summary / Résumé en français

Une des limitations des méthodes stéréoscopiques est que la disparité de chaque point est évaluée en s'appuyant sur un voisinage de taille fixe, généralement un carré autour du point à estimer. Nous avons vu précédemment que cela conduit à des erreurs d'estimations dans les régions homogènes, faute de texture permettant de précisément suivre les points. Augmenter la taille du voisinage observé permet de limiter ce type d'erreur, mais entraîne d'autres effets négatifs. Les outils de segmentation permettent justement de détecter les régions homogènes : au lieu s'appuyer sur un voisinage de taille fixe pour suivre les points entre deux images, nous proposons ici une méthode qui estime la disparité pour chaque région homogène d'une image. Afin de choisir une région de taille adaptée, nous utilisons un algorithme de segmentation hiérarchique qui divise une image en région, sous régions et ainsi de suite.

We have seen in Chapter 4 that most stereo techniques evaluate the disparity of each point of an images' pair by looking at a fixed neighborhood, generally a square block. In this case, disparity values of points in homogeneous regions are often misevaluated. Increasing the neighborhood's size can alleviate this problem, but often at the cost of more fattening effects. Semi-global and global methods can also help at reducing the issue, but the improvement is insufficient in terms of accuracy and computation time for the standard we aim to attain.

There is no general and correct block size that can be used in our SEM images. They contain large homogeneous regions, where using large block sizes would be necessary. They also contain a lot of small and textured regions, where using large block sizes would create inaccuracy in the disparity evaluation. In addition, occlusions are often depicted, so the fattening effect created by large block sizes would produce high inaccuracies in these cases.

Segmentation seems like a perfect solution to this problem. Indeed, the watershed algorithm separates the image in different homogenous regions that we can use for our matching process. It is not a new idea as several methods [29][70][97][98] already use image segmentation for stereo reconstruction purposes. However, the watershed has a tendency to oversegment the image, especially when they are noisy, like our SEM images. Furthermore, it is difficult to create a perfect segmentation of the image : by removing oversegmentation in some areas, we might lose some important edges in the image.

We have therefore created a solution that uses the hierarchical segmentation of an image in a top-down manner; it first processes the main regions of the image, then - if necessary - its subregions, its (sub) subregions and so on. The benefit of the method is that it doesn't require a perfect segmentation: oversegmentation is tolerated as it rarely occurs at the upper levels of the hierarchical segmentation.

5.1 Assumptions

The presented algorithm relies on the following assumptions:

- We will apply our algorithm on a pair of images.
- This pair has been processed so that the matching search area is limited to the horizontal axis. In general, it means that the images are rectified. In our SEM images, we will suppose an orthographic projection model (since the magnification is larger than 1000×), rotate the images 90°, and vertically align the pair using the process explained in Section A. Our algorithm could handle 2D search, but at the cost of a much higher computation time.
- We know the lower bound and upper bound of the matching search range.
- Images contain an acceptable amount of noise. For our SEM images, we will suppose that they have been filtered as explained in Section 7.3.1.
- The shape of the studied objects didn't change between the two acquisitions. In our SEM images, that means that there was no thinning or contamination effects due to the electron beam.
- We know the tilt angle of both images, though we don't need to know where the rotation axis is positioned compared to the sample.

5.2 Overall idea

The final goal of our solution is to obtain a depth map. We have seen in Section 4.1 that once the geometry and camera matrices have been determined, the main challenge lies in the disparity map computation.

In real terms, if we have a pair of images A and B, the disparity map represents the transformation between the image A and the image B. See an example in Figure 5.1.

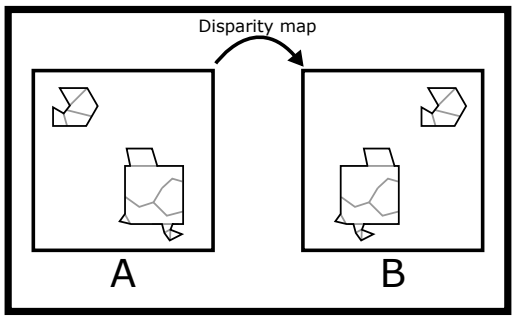
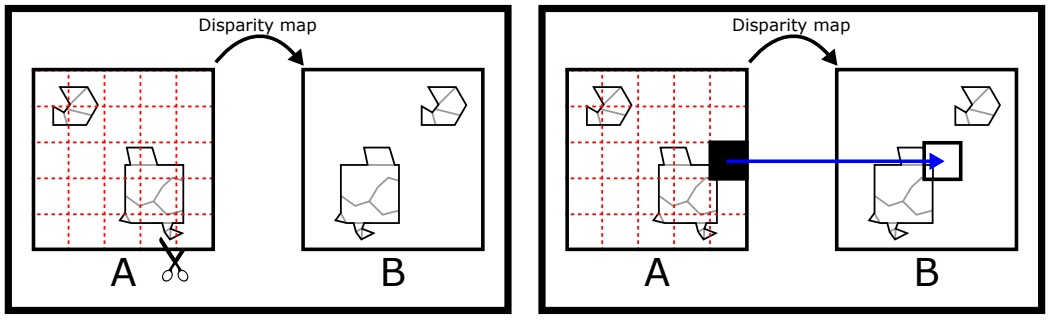


Figure 5.1: Pictures A and B. *The disparity map evaluates the transformations that are necessary to change A into B.*

It is therefore interesting to take a step back and imagine how, if we were to measure this disparity map manually, we would solve the problem. Concretely, we suppose that we have two similar but not identical photographs A and B. We want to apply changes to the picture A to make it look like the picture B. For doing so, we can only cut and move regions of the picture A.

If we transpose classical stereo methods to our example, they would cut the picture A into a set of small squares and then move each of these squares where they best match the picture B. See Figure 5.2. It is not however how most people would do it manually, and, as we have seen, it assumes that there is enough texture.

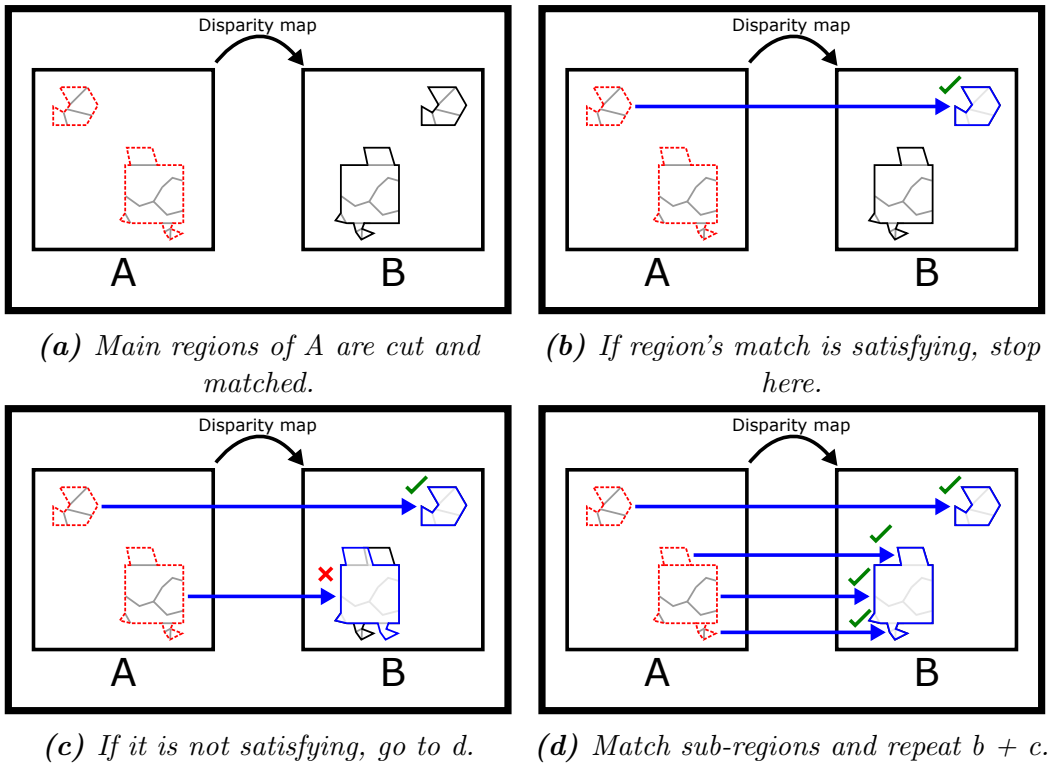


(a) *Image A is cut in squares.* ***(b)*** *Each square is matched in B.*

Figure 5.2: How classical methods generally evaluate the disparity map.

Most people start by cutting the picture A on its most obvious edges. Once it is done, the picture A is separated into a set of regions. They then try to move each region where they best match in the picture B. For each region, if the match is satisfying, the process stops here. If it is not, they cut the most obvious edges inside the region. They obtain a new set of regions, on which they apply the same process as described in this paragraph until the result is satisfactory. See Figure 5.3.

We propose to evaluate disparity maps using this same region-cutting process.



(a) *Main regions of A are cut and matched.*

(b) *If region's match is satisfying, stop here.*

(c) *If it is not satisfying, go to d.*

(d) *Match sub-regions and repeat b + c.*

Figure 5.3: How we propose to evaluate disparity maps.

5.3 General process

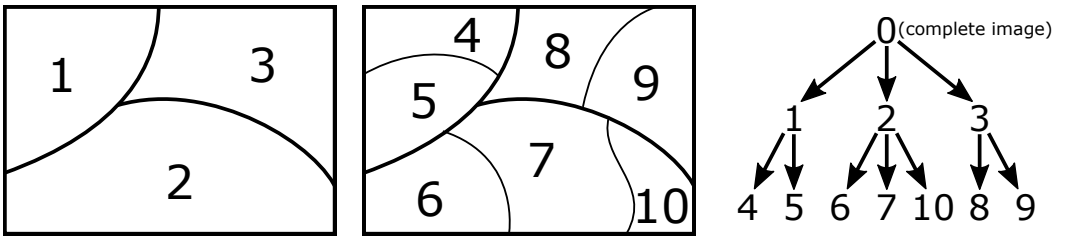
We will describe here in more concrete terms the process of our algorithm.

As input, the algorithm receives two images : a reference image and a secondary image. It outputs a disparity map, that can then be easily converted into a depth map using the equations shown in Section 4.1.2.

The algorithm will first compute the hierarchical segmentation of the reference image. The hierarchical segmentation of an image can be represented by a **partition tree**. The concept is very similar to the binary partition tree[99] presented by Salembier and Garrido, except that a node can have more than two children. The top node represents the whole image, its children represent the most important regions of the image according to the hierarchical segmentation algorithm used, their children represent secondary regions, and so on until a node can't be separated into more sub regions. An example of partition tree is illustrated in Figure 5.4.

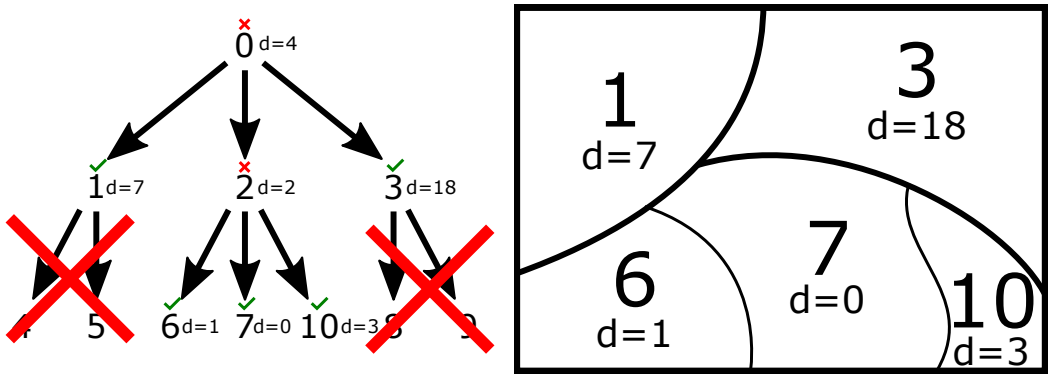
Once this partition tree has been constructed, we start from the top node, and compute the disparity of the associated region. If the match is satisfying or if the node doesn't have any children, we stop here, associate the disparity to the node, and delete all the node's children. If it is not satisfying, we retrieve the node's children, and apply the same process individually to each node. See Figure 5.5a.

If we combine all the leafs of the resulting tree, we can obtain a segmentation of the reference image where each region has been affected a disparity: this will be the disparity map that our algorithm will output. See Figure 5.5b.



(a) *1st level of hier. seg.* ***(b)*** *2nd level of hier. seg.* ***(c)*** *Resulting partition tree*

Figure 5.4: Hierarchical segmentation to partition tree. *Hier. seg.: Hierarchical segmentation.*



(a) *Processed partition tree.* ***(b)*** *Resulting seg. and disparity map.*

Figure 5.5: Partition tree processing and resulting disparity map. *In (a), we evaluate the disparity d of each node of the partition tree, assess if the match is satisfying, and delete the node's children if it is. In (b), we combine all the leafs of the processed partition tree displayed in (a) to obtain a final disparity map.*

In order to concretize this general concept, we need to define the following: the partition tree H, the method used to get the region's disparity and the conditions defining when the match of the node is deemed to be satisfactory.

5.3.1 The Partition Tree H

We first compute the mosaic gradient of the reference image. We used the mosaic gradient instead of the standard watershed operator because of its much thinner nature (see Section 3.3.4).

We then compute the enhanced waterfall (see Section 3.3.5) on this mosaic gradient. The result is a grayscale image S_h. If $S_h(x, y) = n$ and $n \neq 0$, then the pixel is a border between two or more regions at the level n or higher of the hierarchical segmentation. The higher the level, the coarser the segmentation.

Algorithm 10 shows how we transform S_h to a partition tree:

Algorithm 10: From S_h to a partition tree

Function *getPartitionTreeFromSh(S_h)*

```
N ← max(S_h) ;          // Set N as the highest level of segmentation
T ← newPartitionTree() ;
foreach n in [N, N − 1, ..., 2, 1] do
    S_h^n ← S_h ≥ n ;          // set S_h^n as the segmentation at level n
    L_h^n ← label(S_h^n) ;                  // Label S_h^n and save it in L_h^n
    L_h^n ← unique(L_h^n) ;  /* Make each label unique in L_h^n: no similar
     labels in L_h^n should exist in previously processed levels */
    foreach label l in L_h^n do
        R ← L_h^n == l ;  // Set R as the region in L_h^n where label = l
        // Set p as parent id of label l
        if n == N then
            p ← T.topNode ;              // The parent is the top node
        else
            p ← Get label in L_h^{n+1} inside the region R ; /* Note: no region
              in L_h^n can belong to two regions in L_h^{n+1} */
        T.addNode(id=l, parentId=p, region=R) ;
return T ;
```

5.3.2 Disparity computation of a node

For determining the disparity of a node, we evaluate the disparity of the associated region between the reference image and the secondary image according to the process described in Section 4.2. As the matching score, we use the Sum of Squared Differences (SSD) discussed in Section 4.2.3. The matching score is not illumination-independent, so we suppose that gray intensities of objects remain similar between the two images.

5.3.3 Conditions of a satisfactory match

The match of the node n is considered satisfactory if the node has no children or if processing the node's direct children independently does not produce better results.

For doing that, we retrieve the children of the node n. We independently evaluate their disparity according to the same process described in the previous section 5.3.2. If all the children have the same evaluated disparity than their parent node n, then there wasn't any improvement and the match is satisfactory. Otherwise, there is value added in processing the children independently, so the match is not satisfactory.

With such a system in place, we hope to get a satisfactory match before getting to the over-segmented levels of the hierarchical segmentation.

5.4 Discussion

Some results of the proposed algorithm on SEM images are shown in Figure 5.6.

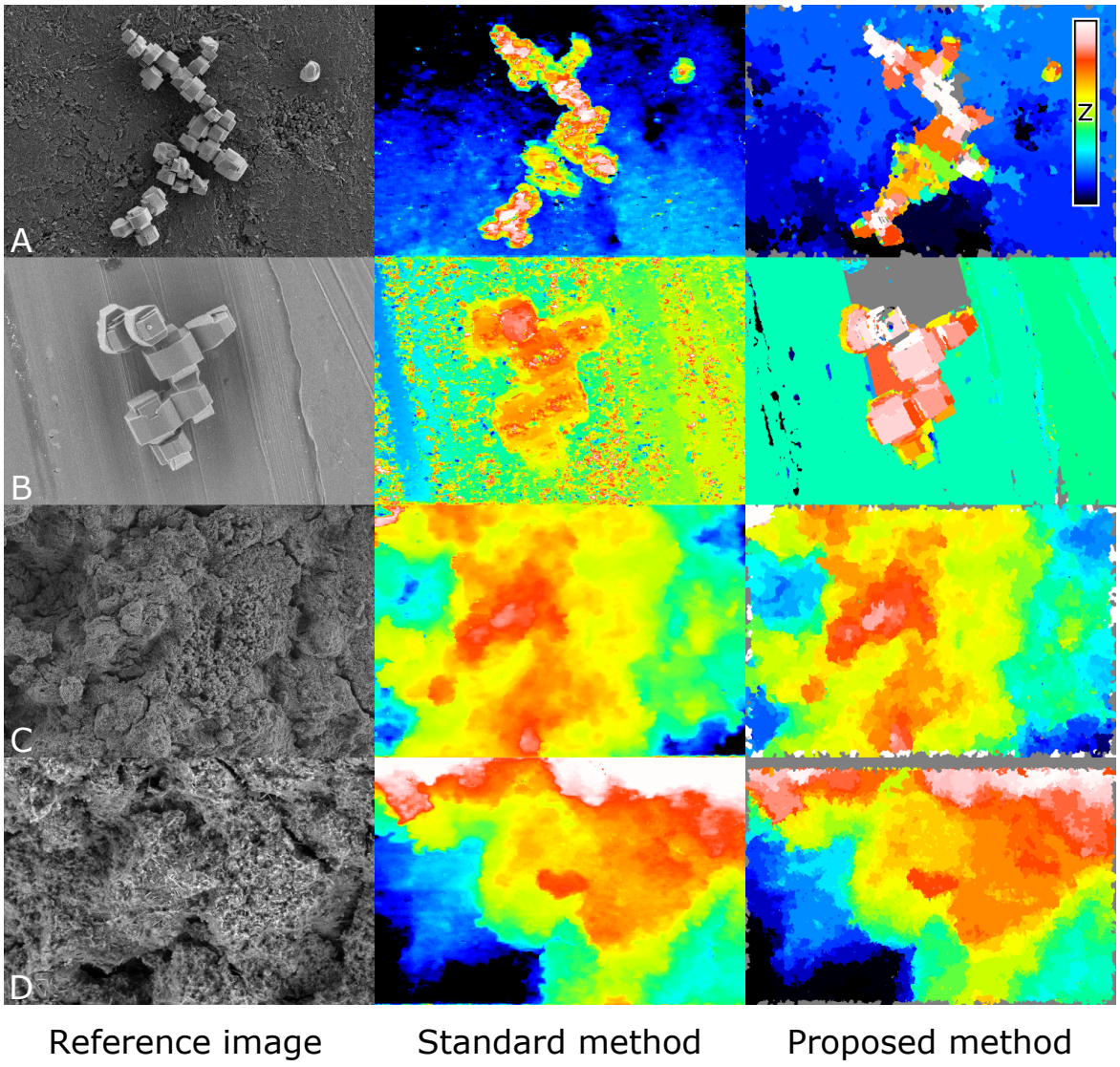
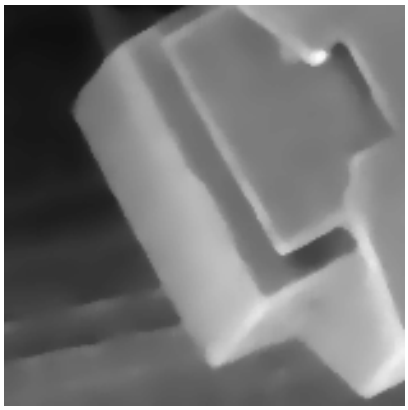


Figure 5.6: Comparison of depth maps obtained using a standard stereo method and our proposed method. *We used the MountainsMap [44] application to produce the depth maps for the standard method column. The acquisitions tilts were adapted to the analyzed sample: when the sample was flat, the difference in tilt angles between the reference image and the secondary image was high. For A, the tilt angles used to acquire the reference image was 0° and 5° for the secondary image. For B, 0° and 5°. For C: 0° and 4°. For D: 0° and 2°.*

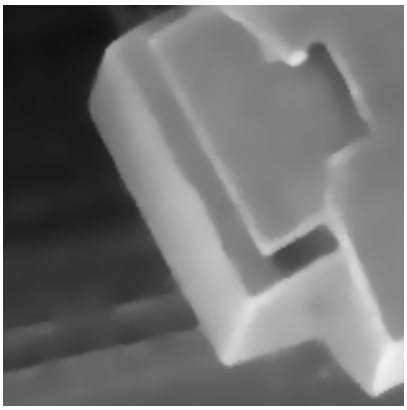
On the good side, our algorithm has produced less noisy depth maps. It is because large homogeneous regions are processed as a whole. As fattening effects have been strongly reduced, the edges of the images are also much more respected in the final depth map, thanks to the segmentation that has been heavily used by our approach. The results are therefore perceptually better than standard stereo approaches when evaluating low textured images, and they are similar when evaluating textured images.

However, our algorithm suffers from a number of flaws that limit its performance.

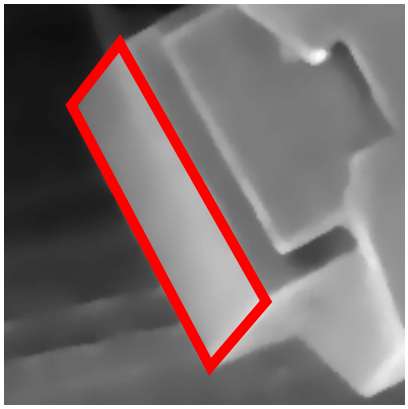
First, evaluating the disparity of a region as a whole supposes that the shape it represents in the reference image will not be distorted in the secondary image. This assumption is often not verified, particularly on very oblique surfaces. In these cases, the match is not satisfactory: the algorithm will therefore evaluate the disparity of sub-regions, often created by over-segmentation, leading to depth misevaluations (see Figure 5.7). This also makes the use of more than two images difficult as this increases the probability that the region will be distorted.



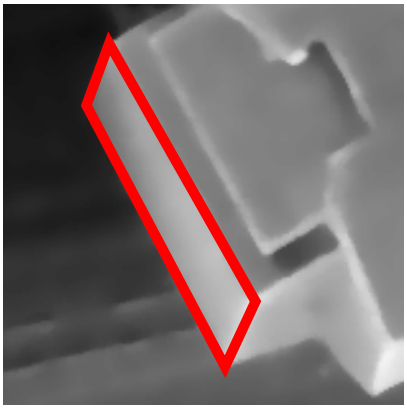
(a) *Extract of reference image (tilt = 0°).*



(b) *Extract of secondary image (tilt = 5°).*



(c) *(a) with oblique region highlighted.*

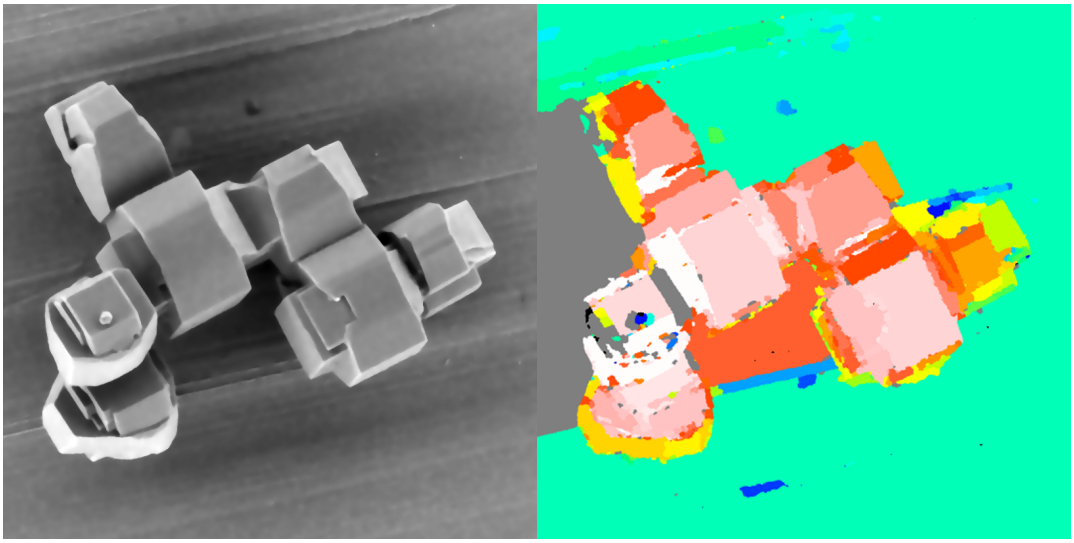


(d) *(b) with oblique region highlighted.*

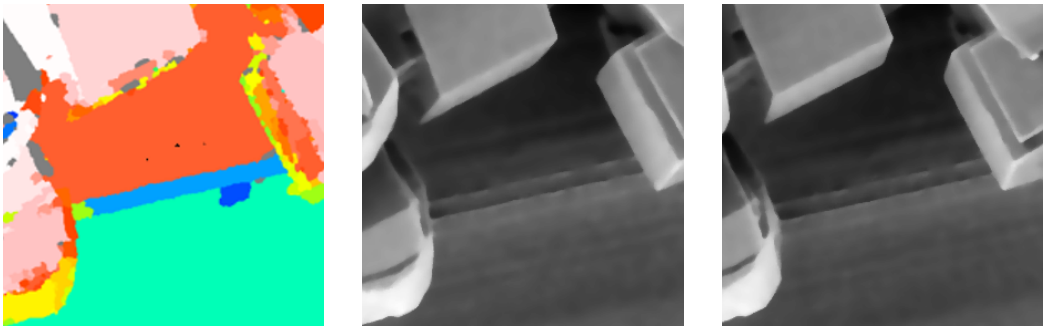
(e) *Resulting depth map.*

Figure 5.7: Deformation of an oblique surface between two images.

Secondly, occlusions are not well managed. Indeed, it can happen that some regions are partially occluded by a foreground object. When computing the disparity of the region, the result will be influenced by the foreground object as it will interact with the matching cost. See Figure 5.8.



(a) *Extract of reference image.* ***(b)*** *Extract of disparity map.*



(c) *Zoom in (disp.)* ***(d)*** *Zoom in (ref.)* ***(e)*** *Zoom in (sec.)*

Figure 5.8: Illustration of an occlusion effect.

The processing speed is very dependent on whether we are dealing with textured or low textured images. On low textured images, it is acceptable - around one minute -, but on highly textured images it can take up to 5 minutes. Indeed, most of the time is spent during the disparity computation of the nodes, so the overall processing time will be highly dependent on the number of nodes in the partition tree. Table 5.1 gives an indication on the partition tree's structure for textured or low textured images. However, the processing time could be speeded up as the matching process was not optimized nor parallelized.

Finally, the way the hierarchical segmentation is currently used doesn't always prevent processing too small and untextured regions. Indeed, when evaluating whether a node's match is satisfying, its children could already over-segment too much the region for being meaningfully matched.

Depth	Low textured	Textured
1 (main regions)	191	2760
2 (subregions)	301	3932
3	677	4964
4	1212	7439
5 (leafs)	2484	13268

Table 5.1: Number of nodes in the hierarchical tree depending on depth. *The Low textured image is shown in Figure 5.6.B and the Textured image is shown in Figure 5.6.D. Both images have been filtered following the process shown in Section 7.3.1.*

5.5 Summary

We have presented our first segmentation based approach for stereo reconstruction. Though it features some advantages, there are too many shortcomings to produce an accurate 3D representation.

We have however decided to briefly describe this method as it constitutes the basis of the next presented method, Top-Down Segmented Regression (TDSR). This method keeps the top-down approach of TDSM while addressing its most critical issue: the way regions are matched between the reference image and the secondary image.

Indeed, though the matching process is straightforward, it relies on one limiting assumption. It supposes that regions won't be distorted between the two images, and, therefore, that they are all parallel to the camera plan (fronto-parallel hypothesis). The main focus of the next method will be to provide a better way to match these regions and measure their distortion.

Chapter 6

Top-down segmented regression

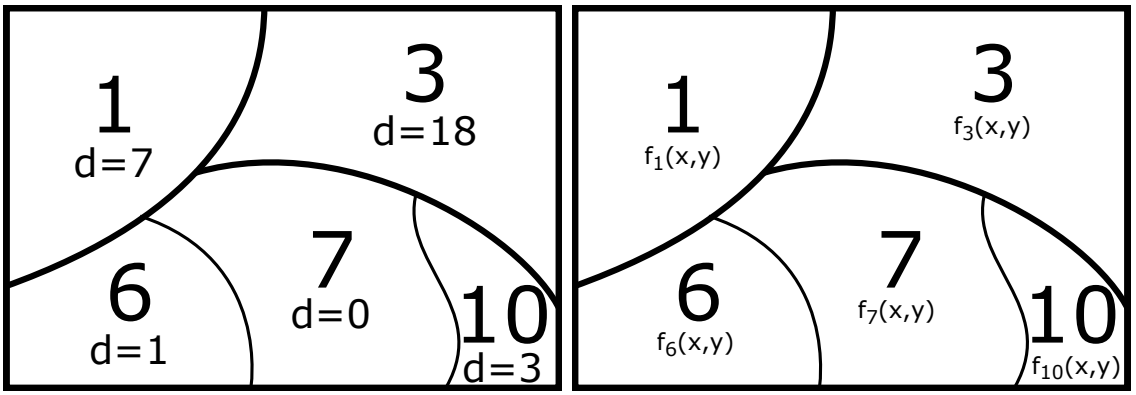
French summary / Résumé en français

Bien que produisant des résultats plus satisfaisant que les méthodes classiques, la méthode présentée lors du chapitre précédent est encore insuffisante pour une analyse précise des catalyseurs. La cause principale est que l'on suppose que la disparité est constante au sein de chaque région, ce qui n'est pas le cas des régions inclinées. La méthode décrite dans ce chapitre a été développée pour tenir compte de ces inclinaisons. Bien qu'elle s'appuie de manière similaire que précédemment sur la segmentation hiérarchique pour diviser les images en régions, un changement de paradigme a lieu dans le traitement du problème. La méthode présentée ici n'estime pas une carte de disparité, mais affine et complète une carte de disparité existante, pouvant être estimée par des méthodes classiques. Pour évaluer l'inclinaison de chaque région, elle utilise le principe de régression sur les points définis dans la carte de disparité initiale et à l'intérieur de la région.

We have proposed in Chapter 5 a solution, TDSM, that relies on hierarchical segmentation to compute the disparity map of a pair of images. Though promising, it has several drawbacks that impede its performance. One of the main drawbacks is that it matches regions as a whole, assigning a single value of disparity for each region it processes, leading to mis-evaluations.

The new proposed algorithm, Top Down Segmented Regression (TDSR)[100], carry on the general framework of TDSM, but instead of setting a single disparity value for the region it processes, it sets a mathematical function, or model, that depends on x and y. See Figure 6.1.

This new way of processing regions allows to better deal with the complex structures we have to analyze, particularly oblique surfaces. It also leads to a paradigm shift compared to the previous algorithm: TDSM is a matching algorithm that computes a disparity map, TDSR refines and completes an existing disparity map, possibly created by any matching algorithm.



(a) TDSM's leafs of the partition tree *(b) TDSR's leafs of the partition tree*

Figure 6.1: Instead of assigning a single disparity value for each region, TDSR assigns a model.

Here, we will not only tune and check the performance of our algorithm on SEM images, but also on the Middlebury database presented in Section 4.6. Though the Middlebury images are different than SEM images in many aspects, it will allow us, in Chapter 7, to easily compare our results with 60 state of the art stereo matching methods.

6.1 Assumptions

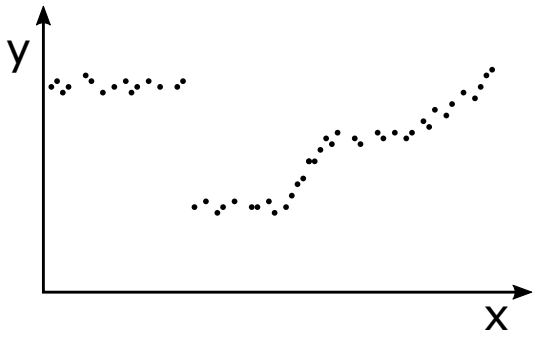
The assumptions are similar as those presented for the previous algorithm TDSM. See section 5.1

6.2 Overall idea

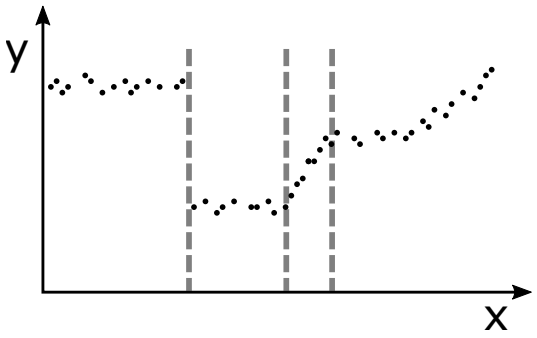
We will transpose here the idea to a 2D problem as it is much easier to visualize, but it can be generalized to 3D or n-D problems.

Our objective is to obtain a complete and accurate depth map of a scene. For this purpose, we acquired an initial depth map evaluation S that is incomplete, noisy and subject to outliers (see Figure 6.2a). We also acquired a partition tree that separates this initial depth map into regions, sub-regions and so on (see figures 6.2b and 6.2c).

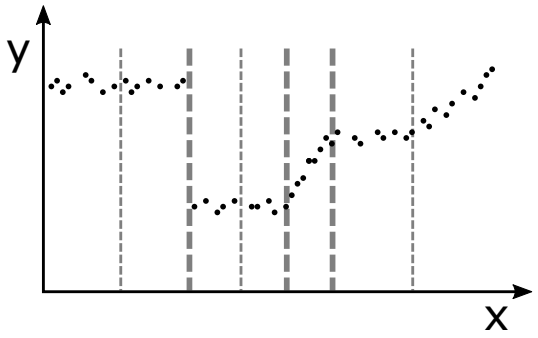
First, we try to find the line that best fits all the points in S (Figure 6.2d). If the fit is considered satisfying the process stops here. Otherwise, we separate S into different main regions using the partition tree, and try to fit a line on each region (Figure 6.2e). For each region, if the fit isn't satisfying, we separate it again into different sub-regions and repeat the process until the fit is considered satisfying (Figure 6.2f).



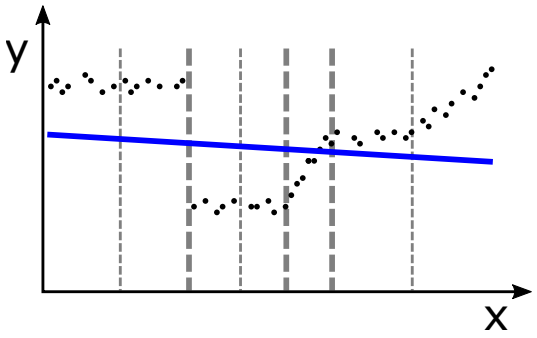
(a) The initial depth map evaluation S



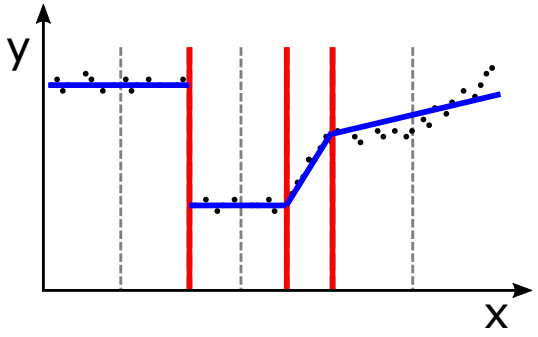
(b) Main regions of the partition tree



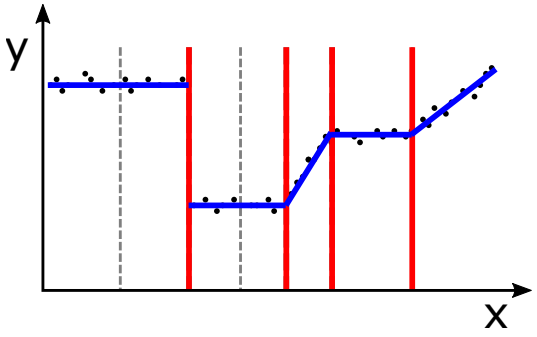
(c) Sub-regions of the partition tree



(d) If we try to fit all S with a line, it isn't satisfying.



(e) So we fit each main region. It is better but the last region is still not a good fit.



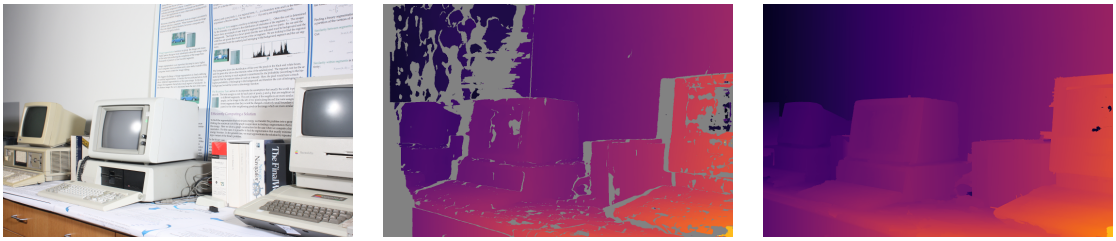
(f) We therefore process each sub-regions of the problematic region, until satisfaction.

Figure 6.2: Main idea of the TDSR algorithm.

6.3 General process

We will now transform this overall idea to a more formalized process.

We have two images depicting the same scene (see example in Figure 6.3a). We also have a sparse and / or noised disparity map between the reference and the secondary images obtained using an existing stereo matching algorithm (see Figure 6.3b). Large areas of the disparity map can be undefined (gray pixels), and defined areas may contain errors. Furthermore, we know the camera matrix: if a point is defined in the sparse disparity map we can know its 3D relative position in the scene, and vice versa. From these information, we want to obtain a complete and accurate disparity map (see Figure 6.3c).



(a) *Reference image* ***(b)*** *Sparse disparity map* ***(c)*** *Objective*

Figure 6.3: From sparse disparity map to complete one, illustrated by the Vintage pair of the Middlebury dataset[90].

We will suppose that the scene is a collection of different **objects**. Each object's surface is composed of one or multiple **simple shapes**. We will consider here that those shapes can be represented by a plane: all the 3D points inside a shape can be modeled using a bivariate linear polynomial, that we will call **model**.

Our method will therefore try to separate all the simple shapes in the reference image. If each region is a simple shape, we can obtain a 3D point cloud estimate of the corresponding region from the sparse disparity map (by converting each disparity value to a 3D point) and model it by a bivariate linear polynomial. The model can then be used to correct existing disparity values and fill undefined values in the region, by converting back each 3D point of the model to a disparity value. However, this is not automatically feasible by a segmentation algorithm, so we will instead rely on hierarchical segmentation.

After computing the partition tree of the reference image, we start from the top node, and try to model its associated 3D point cloud. If the modelization is satisfying or if the node doesn't have any children, we stop here, associate the model to the node, and delete all the node's children. If it is not satisfying, we retrieve the node's children, and apply the same process individually to each node.

If we combine all the leafs of the resulting tree, we can obtain a **modeled segmentation** - or **model map** - of the image, where each region is associated to a model. Some post-processing can be done on this model map that we will describe in Section 6.3.5. It can then be converted to a complete disparity map.

In a nutshell (see Figure 6.4), the TDSR algorithm takes two inputs:

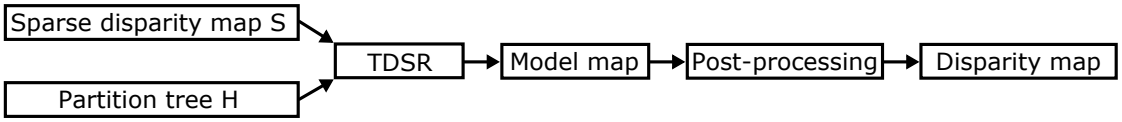


Figure 6.4: *General process*

- The initial sparse disparity map S
- The partition tree H that divides S into regions, subregions and so on

We will first describe how these two inputs are generated.

From these two inputs, the TDSR produces a model map. We already explained the general top-down process allowing to produce this model map, but we need to answer the two following questions:

- How the disparity values of a region are modeled into a bivariate linear polynomial
- What are the conditions that define a satisfying modelization

Finally, if the model map is the end product of the TDSR algorithm, some work can still be done to refine the result. We will lastly describe the post-processing methods we developed to further refine the model map.

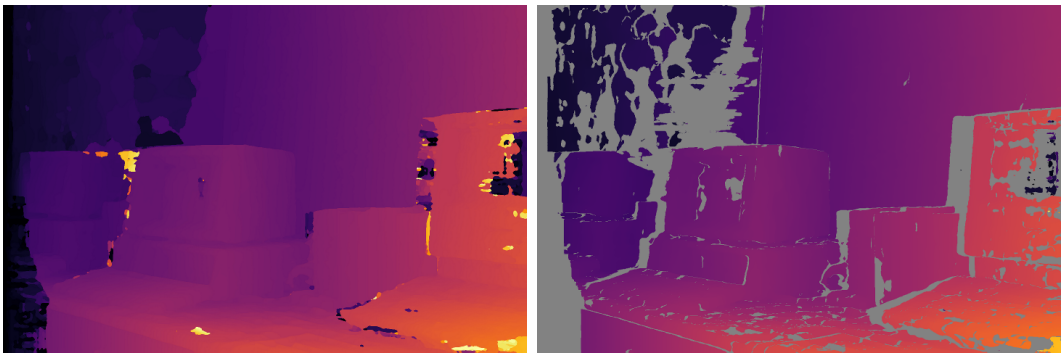
6.3.1 The Sparse Disparity Map S

In order to compute a complete disparity map, we first need an initial sparse disparity map. The quality of our complete disparity map is highly correlated to the quality of the initial disparity map, so choosing an accurate algorithm at this stage is crucial.

For the Middlebury database

For the Middlebury database, we chose to compute this disparity map between the left and right images using the MC-CNN algorithm[54] as it evaluates high quality disparity maps. The inner workings of MC-CNN is described in section 4.2.3. The authors provide two sets of architectures: one optimized for speed, and one for accuracy. We chose the one for accuracy. We applied on the disparity map a LRC check [59] (threshold is set to 1), explained in Section 4.2.5, since the MC-CNN algorithm doesn't handle well occlusions. See Figure 6.5.

To give a sense on how critical the quality of the sparse disparity map is for the performance of our algorithm, we previously used the semi-global MGM [68] algorithm for producing our initial maps. When using MGM and TDSR, the final disparity maps' average error on the Middlebury dataset[90] was more than 50% higher than using MC-CNN followed by TDSR.



(a) Disparity map without LRC *(b) Disparity map with LRC*

Figure 6.5: Effect of the LRC check on the disparity maps produced by MC-CNN.

For the SEM images

For SEM images, we can't use the MC-CNN algorithm due to its slowness and GPU requirement. Instead, we use the much simpler block matching algorithm, since a very fast implementation is available in the OpenCV library [13]. The algorithm is described more thoroughly in section 4.2.3.

However, as all block matching methods misevaluate disparities on homogeneous regions, we adopted a multiscale approach.

We know that choosing a small block size is likely to create a noisy disparity map and that choosing a large block size will reduce its noise but also its accuracy, notably on edges. The idea is therefore to evaluate the disparity map using different block sizes. For each pixel, we will assess its disparity using the smallest block size that meet enough requirements to almost ensure a correct match.

For defining these requirements, we will use different criteria.

First, small block sizes are sufficient around strong edges whereas on homogeneous regions we need larger block sizes. One of the criteria will therefore be the average amount of texture inside the block : a higher amount should allow smaller block sizes to be used.

Secondly, if when evaluating the disparity of a pixel there are matches that have a similar score than the best one, it is a sign of a low confidence evaluation. The used algorithm features the **Uniqueness** parameter that allows to remove such low confidence values. On smaller block size, we will therefore use a high uniqueness threshold to filter out most mis-evaluations.

Finally, the disparity evaluated on a pixel must verify the LRC [59] check (threshold is set to 1).

Concretely, we computed 4 disparity maps using the OpenCV algorithm. Table 6.1 shows the parameters that have been used to generate each one.

Block size	Texture threshold	Uniqueness ratio
51	1	2
35	2	5
21	4	15
11	8	30

Table 6.1: *Parameter used to create the 4 disparity maps.*

We then combined these disparity maps into a single one using the process shown by the Algorithm 11 and illustrated in Figure 6.6. We also track the block size used for each pixel in a **block size map**.

Algorithm 11: Combination of several disparity maps to a single one

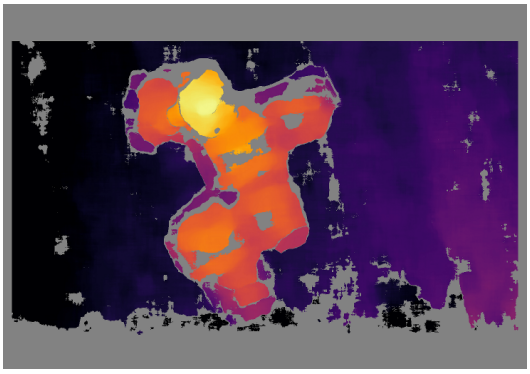
Function *combineDisparityMaps(L_{disp}, $L_{blockSize}$)*

```
/* L_disp is a list of disparity maps that should be ordered from
   the biggest to the smallest block size. Low confidence values
   of the disparity maps in L_disp should have already been
   filtered out. L_blockSize is the list of block sizes used in
   L_disp.                                                      */
N ← length of L_disp ;
size ← size of first disparity map of L_disp ;
D_f ← newMatrixWithSize(size) ;              // Final disparity map
B_f ← newMatrixWithSize(size) ;                   // Block size map
foreach n in [0, 1, ..., N − 1] do
    D ← L_disp[n] ;
    blockSize ← L_blockSize[n] ;
    defined ← positions where D is defined ;
    D_f[defined] ← D[defined] ; // D overrides D_f where D is defined
    B_f[defined] ← blockSize ;
return D_f, B_f ;
```

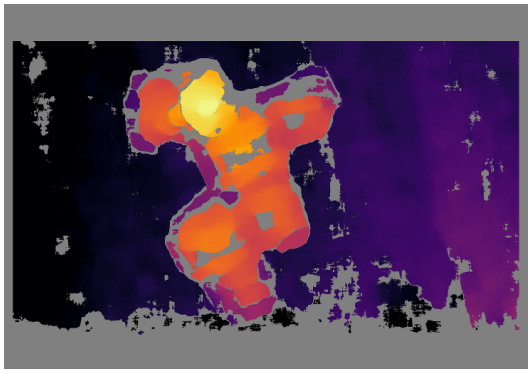
At the global scale, the merged disparity map seems to be composed essentially by the disparity map evaluated with the largest block size. For understanding the added value of the disparity maps evaluated with lower block sizes, we have zoomed-in on a specific zone of the image in Figure 6.7. The disparity map evaluated with the largest block size produces a unnoised but too smooth result. The merged disparity map is more noisy, but respects more the edges of the reference image.

A final filtering is done on the merged disparity map. On some area, some speckles remain with extreme mis-evaluations. We remove these speckles with the process described by the Algorithm 12.

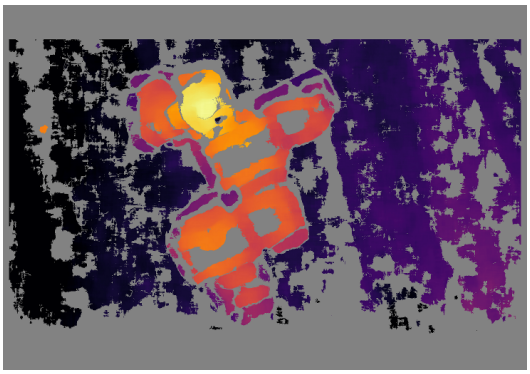
The resulting disparity map and block size map are shown in Figure 6.8.



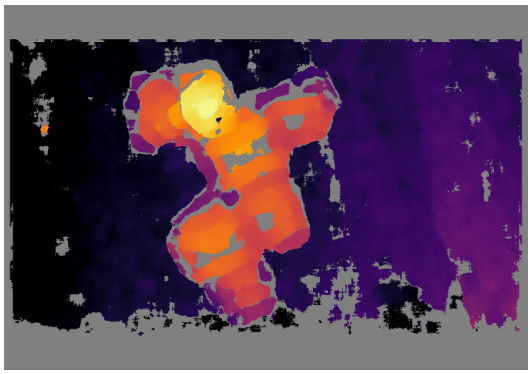
(a) *Disparity map with block size 51*



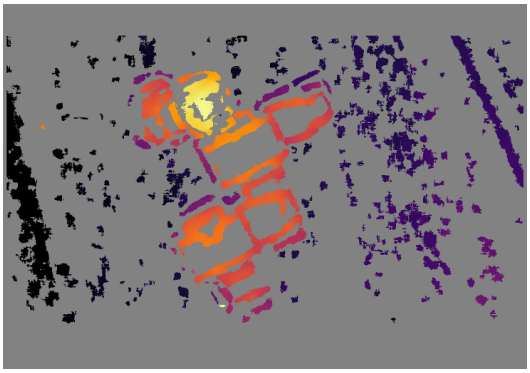
(b) *Merged disparity map*



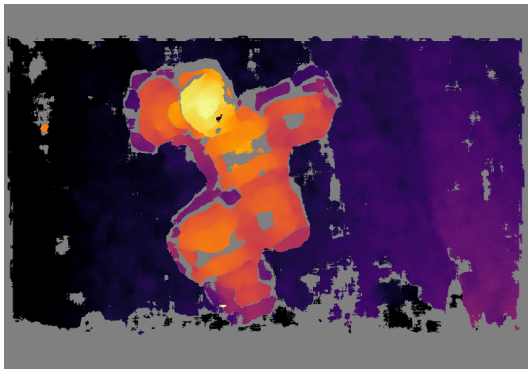
(c) *Disparity map with block size 35*



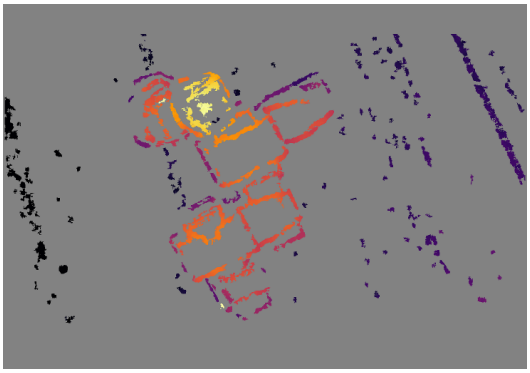
(d) *Merged disparity map: (c) over (b)*



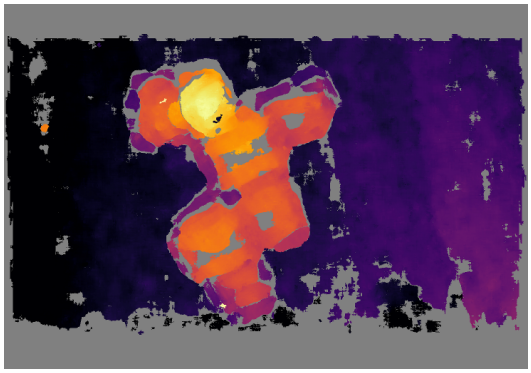
(e) *Disparity map with block size 21*



(f) *Merged disparity map: (e) over (d)*

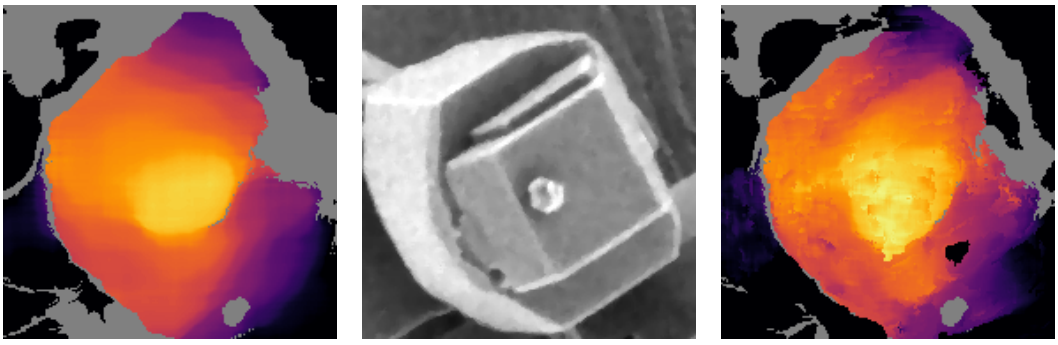


(g) *Disparity map with block size 11*



(h) *Merged disparity map: (g) over (f)*

Figure 6.6: Illustration of the process of merging maps of different scales.



(a) Disparity map (BS=51px) *(b) Reference image* *(c) Merged disparity map*

Figure 6.7: Detailed comparison between the disparity map evaluated with a large block size (BS) and the merged disparity map.

Algorithm 12: Removing speckles from a disparity map.

Function *removeSpeckles(D, B, discThresh, ccSize)*

```
/* D is the merged disparity map, B is the block size map,
   discThresh is the amount of gradient of D defining a
   discontinuity, ccSize defines until which size, in pixels, a
   connected components will be removed.                        */
```

gradient ← MorphologicalGradient(D) ; `/* D is not defined everywhere, if there are undefined values around a pixel, the gradient will be undefined too. */`

discPositions ← positions where *gradient* is not defined or *gradient* > *discThresh* ;

Remove values in D and B on *discPositions* ;

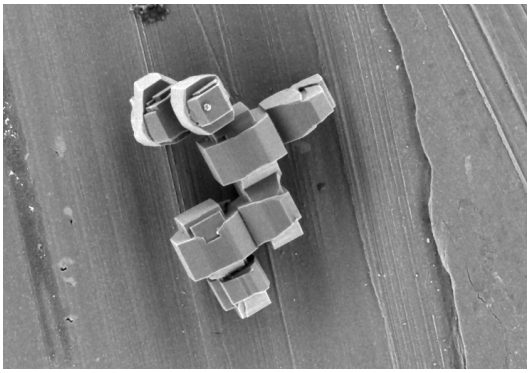
Remove connected components in D and B whose size is less than *ccSize* pixels.

return D, B ;

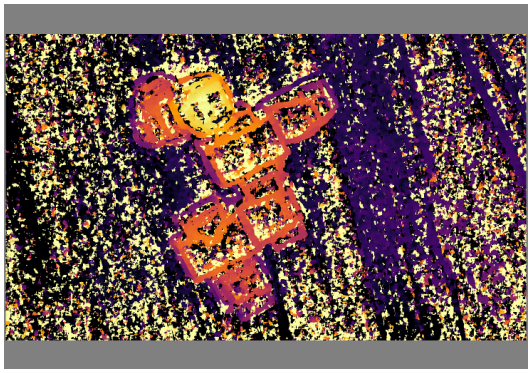
6.3.2 The Partition Tree H

Obtaining an adapted partition tree for our reference images is not a straightforward task.

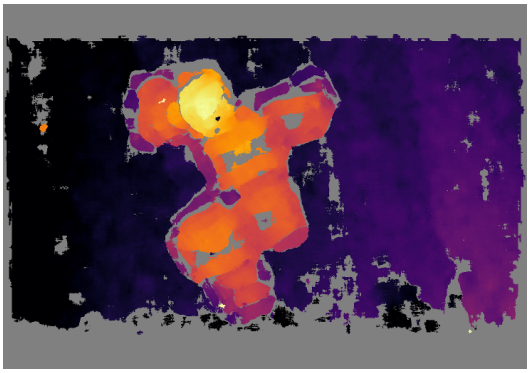
First, the assumption of having simple shapes separated at least at the leafs of the partition tree can be difficult to attain on some images. Segmentation, whether it provides hierarchical information or not, relies on the image gradient. An object occluding another one with similar color can, in the image, have indistinguishable borders with very low or even null gradient.



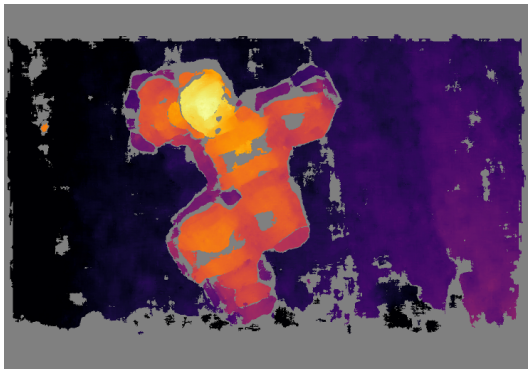
(a) *Reference image*



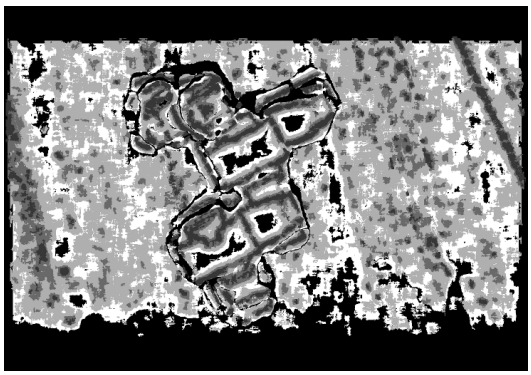
(b) *Disparity map (BS=11)*



(c) *Merged disparity map*



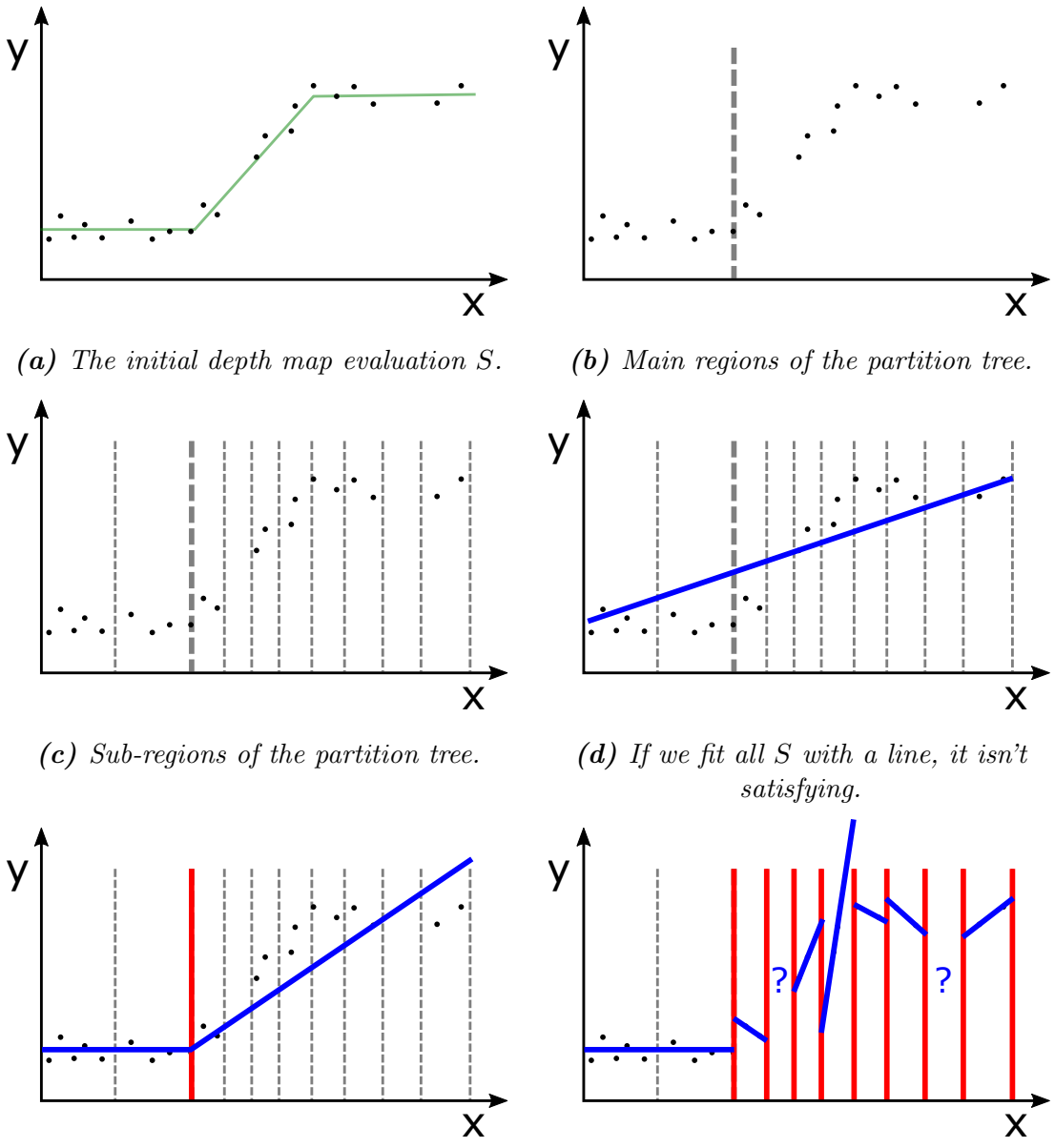
(d) *Merged disparity map + speckle removal*



(e) *Block size map (light: large BS, dark: low BS, black: no defined disparity)*

Figure 6.8: Comparison between a disparity map without low confidence removal, a merged disparity map without and with speckle removal. *Final block size map is also displayed. BS: Block Size.*

Secondly, the partition tree must divide the image in a progressive manner. If it does not - for instance a simple shape is merged with another one at a level and divided into twenty parts at the next level of the tree - it can cause our approach to poorly perform in many ways. Since the input disparity map is sparse, it is possible that some regions at the over-segmented level would contain no disparity values making them impossible to be modeled. Since regions are much smaller and contain fewer disparity points, the modelization would be more sensitive to errors in the sparse disparity map. See Figure 6.9.



(a) *The initial depth map evaluation S.*

(b) *Main regions of the partition tree.*

(c) *Sub-regions of the partition tree.*

(d) *If we fit all S with a line, it isn't satisfying.*

(e) *So we fit each main region. It is better but the last region is still not a good fit.*

(f) *We therefore process each sub-regions of the problematic region, here leading to overfitting.*

Figure 6.9: Illustration of the TDSR algorithm when a unsuitable partition tree is used. *Green: ground truth.*

We took these problems into account when constructing our partition tree.

One of the challenges of our images, especially in the Middlebury dataset, is that borders are sometimes blurry (due to the initial filtering process or due to the image being captured slightly out of focus). If we compute a morphological gradient [14] of size 1 in those borders, their gradient will have low values, preventing an accurate segmentation. There is a change of color that is occurring, but we need to look at a larger scale. However, we don't want to simply use a morphological gradient with a larger size since we might lose out on small details. That is why we used the multi-scale gradient presented in Section 3.3.2. We computed this gradient from scale 1 to scale 6.

For markers, we initially used the h-minima transformation of the gradient, with $h = 5$. A second problem is that, when two adjacent regions have similar colors, the gradient can be less than h at some parts of the border between them. This effect is known as gradient leakage [101]. A way to detect these leakage is that markers will get thinner at these locations. Therefore, we want to encourage a split when the markers get thinner. That is why we applied on these markers the adaptive erosion presented in Section 3.3.3 with $\alpha = 0.25$.

We then compute the valued watershed segmentation from the gradient and markers.

We finally apply on it the waterfall algorithm and transform it to a partition tree as explained in section 5.3.1.

6.3.3 Region Modelization

When we process each node of the partition tree, we want to obtain a model that best reflects the real geometry of the region. For doing that, we have an estimated 3D point cloud extracted and converted from the sparse disparity map which contains errors and bias. We need to address two points. First, we have to define whether we take all the 3D points into account or we select a specific subset. Then, we have to specify how the model is computed from this subset.

Points Selection

A common issue is the well-known fattening effect discussed in Section 4.2.4. The worst consequences of this effect take place around occlusions: near the border separating two objects, pixels on the occluded object will inherit the disparity of pixels on the occluding object. Such an issue can seriously impede the modelization process.

Since disparities around borders are not reliable, we decided not to take them into account when computing the model. For doing that, we first separate the region A into two separate regions: A_b being the border of A and A_i being the inside of A. More specifically $A_i = \varepsilon(A), A_b = A \setminus A_i$, $\varepsilon(A)$ being the morphological erosion of A of size 1.

We want here to select all the pixels in A_i whose matching blocks don't intersect with the border. Therefore the retained pixels $\hat{A}_i$ belong to $\hat{A}_i = \varepsilon_{\frac{B}{2}}(A)$, where $\varepsilon_{\frac{B}{2}}$ is the morphological erosion of size $\lceil \frac{B}{2} \rceil$ (on a square grid) and B is the matching block size.

For merged disparity maps, that have been constructed using multiple block sizes, the process is similar and detailed in Algorithm 13.

Algorithm 13: How A_i is built on multiscale disparity maps

Function *getInsideModelPoints(B, R)*

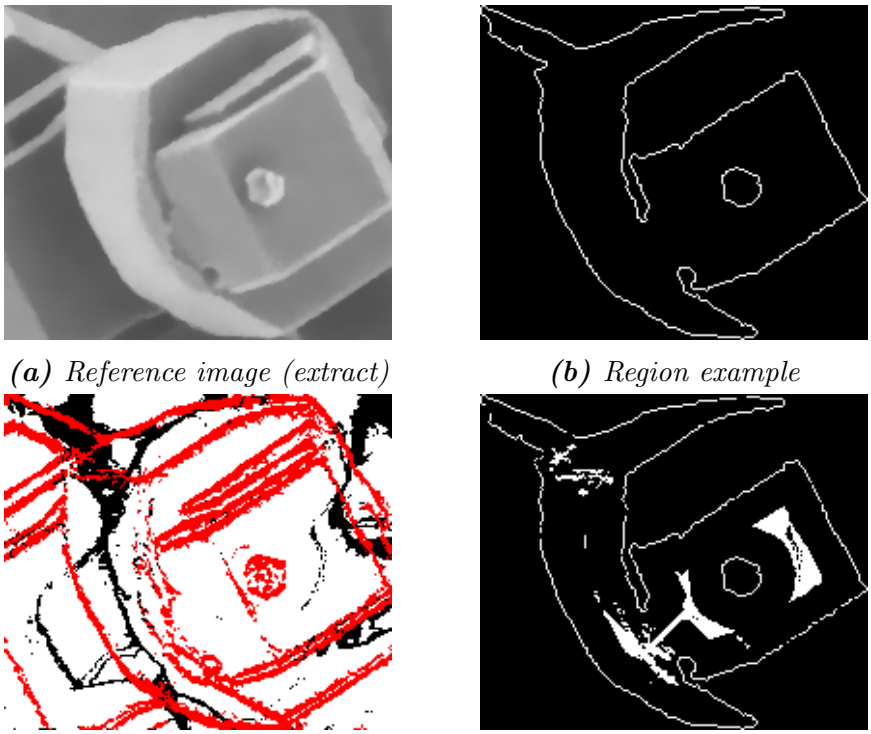
```
/* B is the block size map, R is a map filled with boolean
   values: the pixel's value is true if inside the region, false
   if it is not.                                                  */
sizes ← get unique values of B where R == true ;
sortFromSmallestToLargest(sizes) ;
lastSize ← 0 ;
workingArea ← copy(R) ;
modelPoints ← newBinaryMatrix0FilledWithSize(size(B)) ;
foreach size in sizes do
    workingArea ← ε_{(size−lastSize)/2}(workingArea) ;
    modelPointsForSize ← workingArea ∩ (B == size) ;
    modelPoints ← modelPoints ∪ modelPointsForSize ;
    lastSize ← size ;
return modelPoints ;
```

We will keep all pixels in A_b in the subset where the models will be fitted, as $\hat{A}_i$ might contain an insufficient number of disparity values for an accurate fitting. For instance, the inside of a region might be completely textureless resulting in an absence of disparity values. Though there might be some outliers in the borders, due notably to occlusions, we are relying on the model computation method we will describe later to filter them out.

As a result, we will model only the 3D points from the sparse disparity map in the region $\hat{A} = (\hat{A}_i \cup A_b)$. The selection process is illustrated in Figure 6.10.

Points Modelization

Now that we have selected the 3D points, we can compute our model. There are however some issues that need to be addressed.



(a) Reference image (extract) *(b) Region example*

(c) Block size map around region *(d) Selected points*

Figure 6.10: Example of point selection process. *For the block size map, it was computed from only 2 block sizes : 41px (white), 11px (red). Black areas represent null disparity values.*

First, we have seen previously that some points from A_b can contain outliers which disparity have been contaminated by a nearby object. Another issue is the presence of small areas with disparity values very different from the ground truth. The problem generally appears on areas where there is very little or repetitive texture, as the stereo matching algorithms generally fail on these areas.

We will therefore proceed the following way. We first model the points using a linear regression: we try to fit $z = A + Bx + Cy$. If the model is satisfying (see Section 6.3.4), we stop here and the linear equation is affected as the model of the region. If it isn't, we use RANSAC[102] to perform a robust regression on these 3D points: if the proportion of outliers isn't too large, they should be automatically excluded from the model evaluation process thanks to RANSAC. The obtained linear equation is then affected as the model of the region. We don't directly use RANSAC as it is computationally expensive and linear regressions produce satisfying results in most cases.

If there are no points in the sparse disparity map, we can't compute any model: we will affect to the region an undefined model.

6.3.4 Conditions Defining a Satisfying Modelization

We have constructed a model from a set of 3D points, and now we want to know if the model is satisfying. By satisfying, we mean that we want a large proportion of points that are close to the model.

First, we convert the model to disparity values thanks to the camera matrix. For each point i of the set, we know therefore its value in the sparse disparity map d_i and we also know its disparity value according to the model $\widetilde{d_i}$.

A point is an outlier if $|d_i - \widetilde{d_i}| > t_d$, t_d being a custom threshold (we chose $t_d = 2$). We define the fitting score s as the percentage of points in the set that are not outliers. A model is satisfying if $s > t_s$, with $t_s = 70\%$, and if the number of outliers $n_o < 100$.

6.3.5 Model map post-processing

We have at this stage a model map M. This model map can be transformed to a disparity map by applying on each pixel its associated model: we call this transformation **rasterization**.

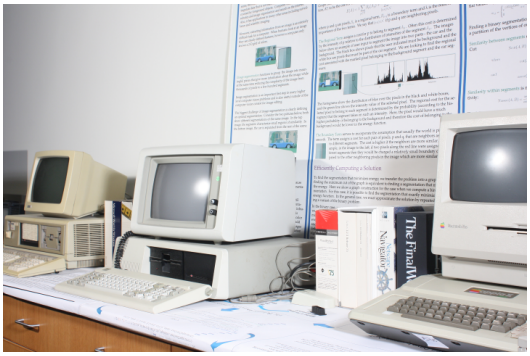
If we rasterize the model map directly after the TDSR process, there will be some undefined areas as well as some remaining errors. See Figure 6.11.

We will mitigate these issue by applying custom operations on the model map.

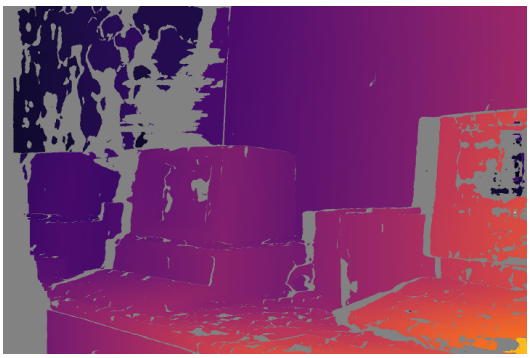
Model propagation on undefined areas

Some regions have an undefined model because of the absence of points in the sparse disparity map. We have to affect a valid model to these regions if we want to evaluate a complete disparity map. These regions are surrounded by regions associated with a model: the main idea is to fill the undefined region with the most probable model in the adjacent regions.

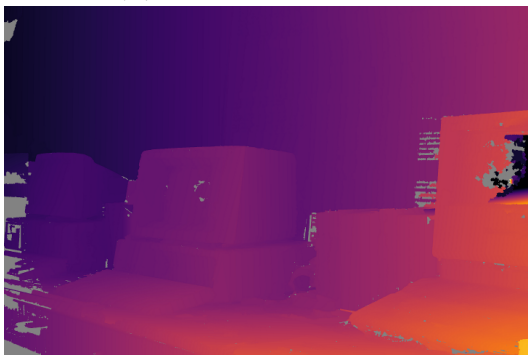
However, since each undefined region might cover multiple simple shapes, we will cut these regions the following way. First, we create a label map ℓ_1 labeling all regions with an undefined model. We create a second label map ℓ_2 labeling all the regions of a similar watershed segmentation than the one computed in Section 6.3.2 but with $h = 12$. We then compute ℓ_f such as $\ell_f(x) = \ell_f(y) \iff ((\ell_1(x) = \ell_1(y)) \wedge (\ell_2(x) = \ell_2(y)))$. We call this operation **intersection**. See Figure 6.12. Each region labeled in ℓ_f will then be processed independently.



(a) Reference image

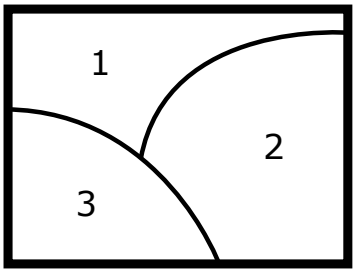


(b) Sparse disparity map

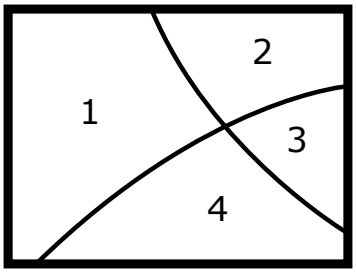


(c) Disparity map after TDSR

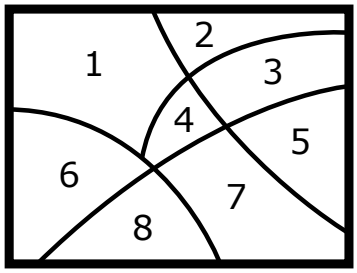
Figure 6.11: Disparity map obtained from the model map produced by the TDSR process without any post-processing, illustrated by the Vintage pair of the Middlebury dataset[90].



(a) Label map ℓ_1



(b) Label map ℓ_2



(c) Intersect. of ℓ_1 and ℓ_2

Figure 6.12: Illustration of the intersection operation on two label maps.
Intersect.: Intersection.

For each label l and associated area A_l, we define its external border as $B_l = (\delta A_l) - A_l$, δA_l being the morphological dilation of A_l of size 1. We list all different models in $M(B_l)$. We want to choose the model that creates the largest consensus across adjacent regions separated to the current region by a low gradient. For doing that, we create a list l_p of all positions where $g(B_l) < \min(g(B_l)) + t_g$, g being the gradient of the left image obtained following the process explained in Section 6.3.2. t_g is a threshold to be defined (we chose 10). We affect to $M(A_l)$ the model M that maximize the number of points p in l_p such as $(M(p) - D(p)) < t_d$, t_d being the same threshold than the one used in Section 6.3.4 ($t_d = 2$). See Figure 6.13.

The labels are processed from the one that contain the lowest proportion of undefined models in B_l to the one that contain the highest.

Model map LRC check

It is possible to apply an LRC check on two model maps - reference to secondary images and the reverse - by rasterizing them, computing a classical LRC on the resulting disparity maps, and set an undefined model on the model maps where the LRC threshold (1) is exceeded.

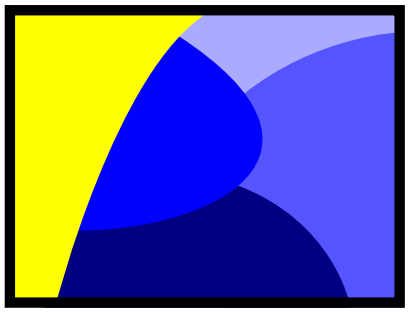
Model map simplification

It is frequent that two neighboring regions in the model map have similar models, meaning that merging both regions would cause little to no change in the final disparity map estimation. Even though TDSR tries to prevent these cases with its top-down approach, it can happen when there is no level in the partition tree that separates two simple shapes in exactly two regions.

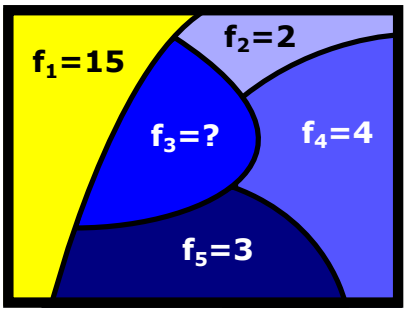
To allow a more robust estimation, the best would be to merge these two regions into a single one and estimate a new model. By doing so, the model map is also simplified, making future manipulations on this model map easier. The objective of the operator is illustrated in Figure 6.14.

The first tried approach was to list all pairs of regions separated by a low gradient (< 10) sorted from the lowest to the highest gradient (two regions separated by a low gradient are more likely to be part of the same simple shape). For each pair, we then tested if both regions had similar models (this similarity criterion will be explained later). If it was the case, both regions were merged and a new model was estimated as described in Section 6.3.3.

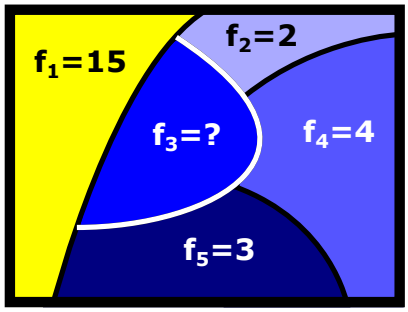
However, the number of mergers can be high and estimating a new model for each merge is costly. To limit the amount of time this post-processing takes, we divide the method in two steps. The first step merges regions with similar models but without updating the model of the merged region: it inherits the model of the largest region being merged. The second step re-estimates the models of the merged regions.



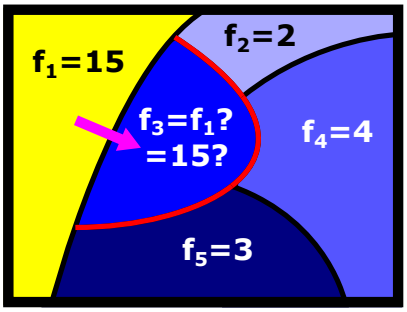
(a) Reference image



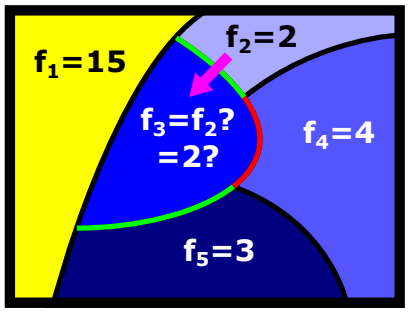
(b) Model map over reference image



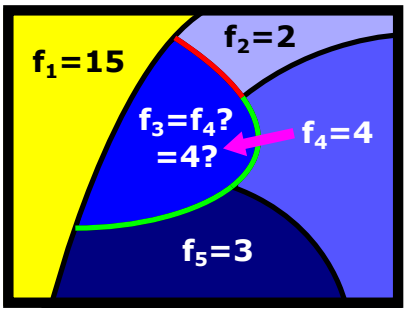
(c) Low gradient borders around f_3 are depicted in white



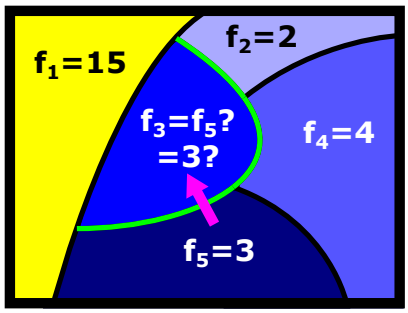
(d) Testing propagation of f_1



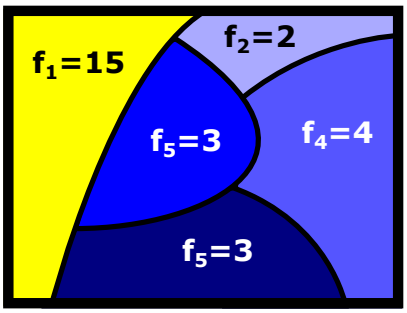
(e) Testing propagation of f_2



(f) Testing propagation of f_4



(g) Testing propagation of f_5



(h) $f_3 = f_5$, as f_5 is the most consensual model

Figure 6.13: Illustration of the model propagation process. *We have the reference image in (a) and a model map previously computed in (b). For clarity, we will consider here that models are constant: $f(x, y) = c$. f_3 is undefined because there were not enough points in the sparse disparity map. We want to fill values of the region represented by f_3. First, we detect lower gradient borders around the region, depicted in white in (c). We first try to propagate the model f_1 on the region, in (d). In this case, all the lower gradient borders would have a high discontinuity in disparity (colorized in red), so this model is not a good candidate. We then try to propagate f_2 (see (e)): there are less discontinuities (acceptable borders are colorized in green), but some remain. Same remark for f_4 (see (f)). Only f_5 achieves a full consensus (see (g)), it is therefore propagated to the undefined region (see (h)).*

This new approach allows to dramatically reduce the number of model estimations as they happen only once all regions have been merged. However, as the models are not updated during the merging step, they can diverge from the final model. The consequence is that it is necessary to repeat the post-processing operator until idempotence. Yet, the iterations that follow the first one need to only update few regions; these iterations are therefore much faster and only a small number of iterations is necessary (most often less than 5).

The process is formalized in Algorithm 14.

Algorithm 14: How a model map is simplified

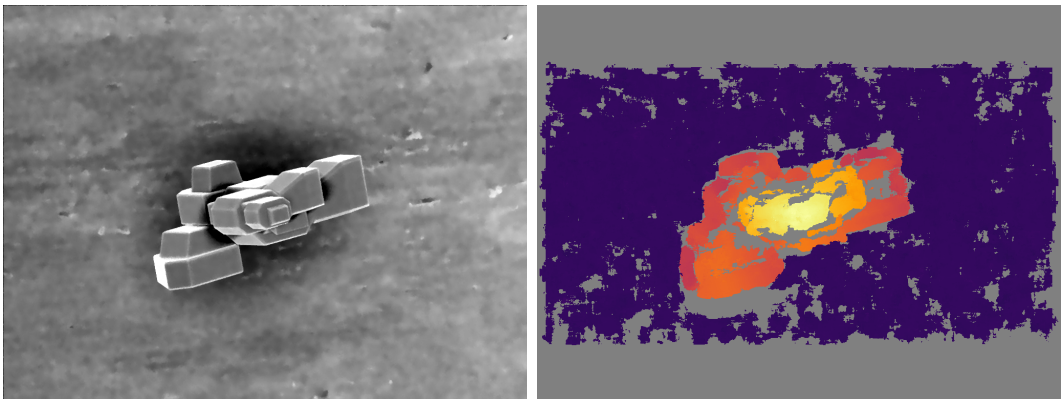
```
Function SimplifyModelMap(MM, isSimilar, gradientThreshold)
    /* MM is the model map, isSimilar is a similarity criterion
       that will be later described. gradientThreshold is a threshold
       (by default set to 10): a pair of regions will be tested only
       if they are separated by an average gradient less than
       gradientThreshold.                                          */
    pairs ← list of adjacent regions in MM separated by a low gradient
     (< gradientThreshold) sorted from the lowest to the highest gradient ;
    mustIterate ← True ;
    while mustIterate do
        mustIterate ← False ;
        mergedRegions ← new list ;
        foreach pair in pairs do
            if isSimilar(pair) then
                model ← model of the region in pair with the largest area ;
                Merge both regions in pair and set model to the merged region ;
                Add merged region to mergedRegions ;
                mustIterate ← True ;
        foreach mergedRegion in mergedRegions do
            Reestimate the model in mergedRegion ;
    return MM ;
```

The post-processing method is called two times using different similarity criteria.

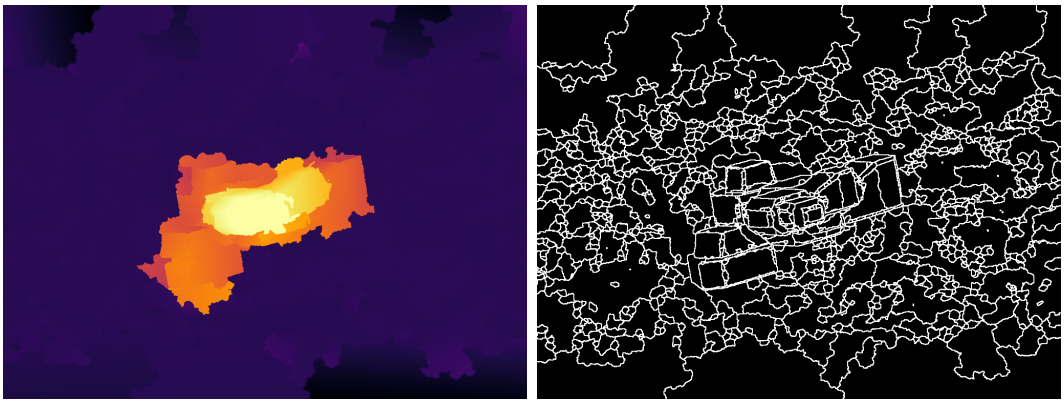
The first similarity criterion is designed for speed. We suppose that we want to know if two regions A and B have similar models M_A and M_B. First, we extract the 4 edge points of the bounding box of A. We estimate their disparity using the models M_A and M_B and measure the maximum offset between the two set of estimations. If this maximum offset is less than a threshold t (set to 2), we consider that A and B have a similar model. If it is not, we repeat the process with B. If the maximum offset is still too large, the two models are not deemed similar.

The second similarity criterion is slower, so it is used to merge regions whose model similarity was not detected with the first criterion. The idea here is to look at the initial data and check if this data can be satisfyingly explained by both models. We suppose that we want to know if two regions A and B have similar models M_A and M_B. We also have access to the disparity values and positions used for estimating their models, D_A and D_B. We count the number n_A of points in D_A whose offset comparing to the model M_A is larger than the threshold t, and do the same with the model M_B (the number is n_B). If $n_B \leq n_A$, then both models are seen as similar. If it is not, we repeat the process with D_B. If $n_A > n_B$, the two models are not deemed similar.

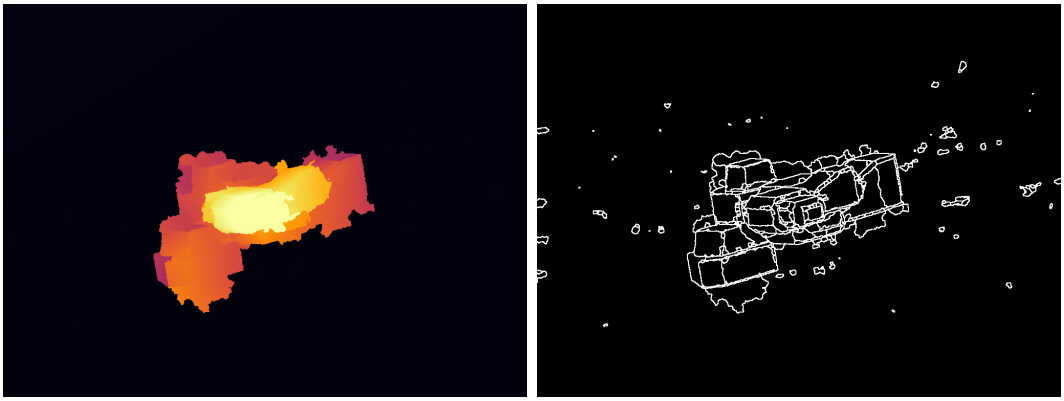
The effect of the model map simplification operator is illustrated in Figure 6.14.



(a) *Reference image* ***(b)*** *Initial disparity map*



(c) *Model map obtained using TDSR* ***(d)*** *Regions in (c)*



(e) *Model map after simplification* ***(f)*** *Regions in (e)*

Figure 6.14: Effects of the model map simplification operator. *The number of regions has been highly decreased. The estimation also seems more exact, especially near the image's borders. Indeed, regions on these parts are misevaluated because they contained too few defined disparity points for estimating a correct model. Using the first similarity criterion alone wouldn't correct these regions because their models are very different from the neighboring models. The second similarity criterion detects these regions as similar as they compare the models to the initial disparity map which remains constant near the border.*

Guided model map correction

Edge preserving diffusion algorithms such as FGS (described in Section 4.4) complete and refine initial disparity maps differently than the TDSR algorithm. Even though they achieve worse performance than TDSR as shown in sections 7.2 and 7.3, their errors have different properties.

FGS and diffusion algorithms tend to flatten disparity values and overflow effects can appear near objects in the front (see Figure 6.15c). As for TDSR, when the image is very noisy with low signal to noise ratio, some regions near objects can be artificially created. These regions are sometimes mis-evaluated, most often acquiring the disparity values of the neighboring object (see Figure 6.15d). As their results present different characteristics, the idea is to combine the two methods to further improve the final result.

The main idea is to measure the difference between the disparity map generated by TDSR and a guide disparity map (obtained by FGS for instance). Regions where this difference is important are more likely to be incorrect, so we replace their models with the neighboring ones using a similar process than model propagation.

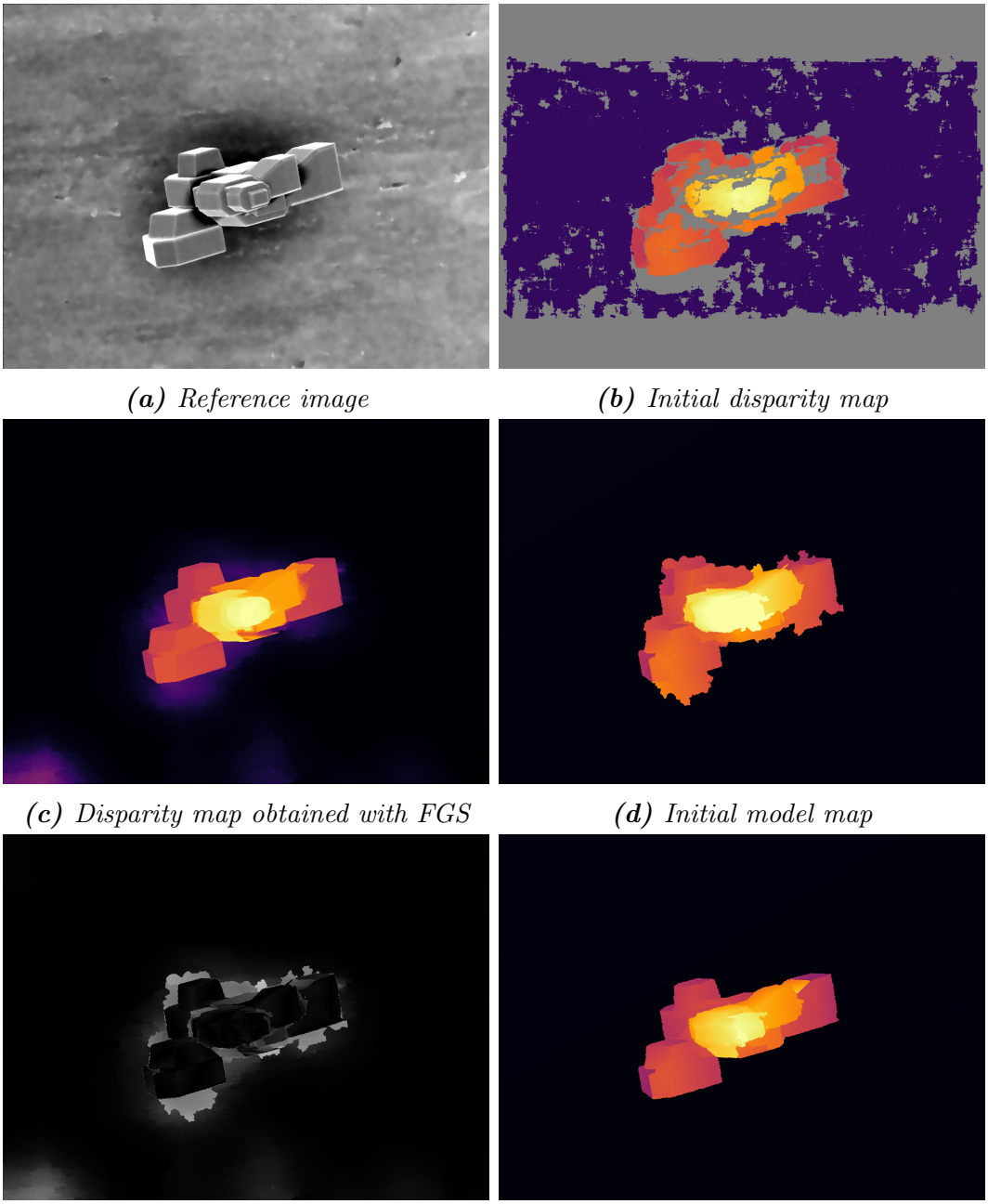
The process is formalized in Algorithm 15.

The effect of the guided model map correction operator is illustrated in Figure 6.15.

Algorithm 15: How is a model map corrected using a guide

Function *CorrectModelMap(MM, guide, diffThreshold, maxGradientThreshold)*

```
/* MM is the model map, guide is the guide disparity map.
   diffThreshold is a threshold: we will only try to correct
   regions with an average difference to the guide disparity map
   greater than diffThreshold. maxGradientThreshold is a threshold:
   when looking for which model to propagate on the region to be
   corrected, we will only look at neighboring regions separated
   by a gradient lower than the
   maxGradientThreshold + minGradient, minGradient being the
   minimum gradient in the border of the region to be corrected.
   */
raster ← disparity map produced by MM ;
diffMap ← absolute difference between raster and guide ;
regions ← list of regions in MM ordered from those with the highest average
 in diffMap to the lowest ;
Remove regions in regions whose average in diffMap is lower than
 diffThreshold ;
foreach region in regions do
    neighbors ← neighboring regions of region ;
    /* We want to propagate neighbors' models that are separated
       from region by a relatively low gradient and are closer to
       the guide disparity map.                                    */
    minGradient ← minimum gradient separating region and its neighbors ;
    minDiff ← minimum average diffMap among neighbors ;
    selectedNeighbors ← subset of neighbors with average diffMap lower
     than minDiff + diffThreshold and separated from region by a gradient
     < minGradient + maxGradientThreshold ;
    if length of selectedNeighbors = 0 then
        Change nothing ;
    else if length of selectedNeighbors = 1 then
        Replace the model of region with the one of selectedNeighbors ;
    else
        Propagate models of selectedNeighbors according to the same process
         illustrated in Figure 6.13 as if region was undefined ;
return MM ;
```



(a) *Reference image* ***(b)*** *Initial disparity map*

(c) *Disparity map obtained with FGS* ***(d)*** *Initial model map*

(e) *Abs. difference between (c) and (d)* ***(f)*** *Model map after guided correction*

Figure 6.15: Effects of the guided model map correction operator.

Post-processing for the SEM images

On SEM images, we developed two approaches.

The default approach is to first evaluate the model map with TDSR and then simply propagate models on undefined areas. This approach generally produces satisfying results. When not specified, this is the post-processing approach used on SEM images.

If results are not satisfying, because images are very noisy or there is a low noise to signal ratio for instance, the second approach is to apply on the result of the first approach the model simplification operator and do a guided correction. This approach is not used by default because there is a performance overhead and more parameters need to be defined. Moreover, it makes the results less predictable because of the guide disparity map: if this guide contains too many errors, it could negatively affect the results of the TDSR algorithm.

On both approaches, we do not apply an LRC check because we would need to apply twice the TDSR algorithm (one for the left-right pair, and one for the right-left pair), doubling the necessary time to produce results.

Post-processing for the Middlebury database

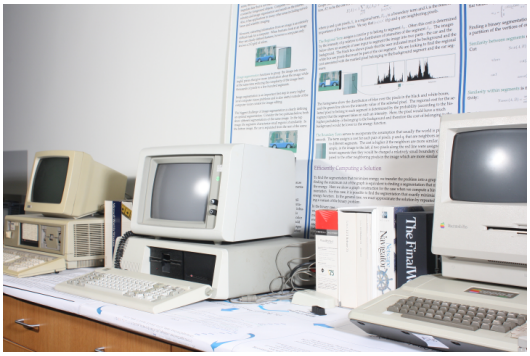
Since there are less time constraints for the Middlebury database, we first compute the model map M between the reference and secondary images and the model map M' between the secondary and reference images. On both model maps, we propagate models on undefined areas. We then apply an LRC check. Finally, we propagate models again undefined areas. The process is shown in Figure 6.16.

This chain of operations is not idempotent: we can apply it another time on the processed model map, and it will produce another result. However, experimentally, there is little to no added value in the final result when adding other iterations.

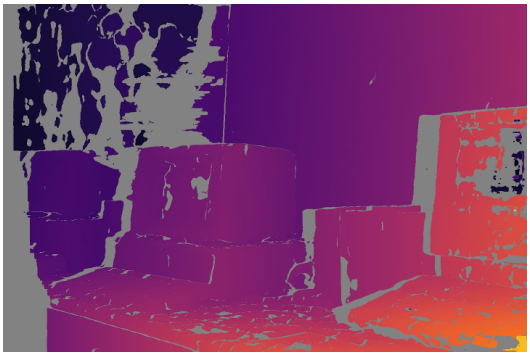
6.4 Discussion

Some results of the proposed algorithm on SEM and Middlebury images are shown in figures 6.17 and 6.18.

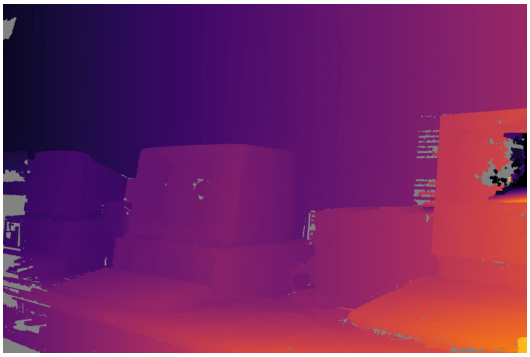
Results have much improved compared to the previous algorithm presented in Chapter 5. TDSR processes much better oblique surfaces. Occlusions are also better addressed, though there can still be some mis-evaluations due to lack of data. Moreover, some homogeneous regions don't respect the **simple shape** hypothesis: their shape are too complex to be successfully fitted by a plane. See Figure 6.19.



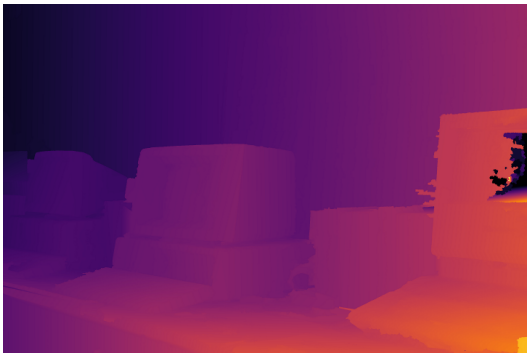
(a) *Reference image*



(b) *Sparse disparity map*

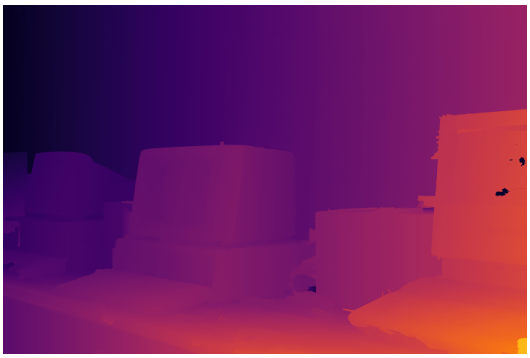


(c) *(b) after TDSR*



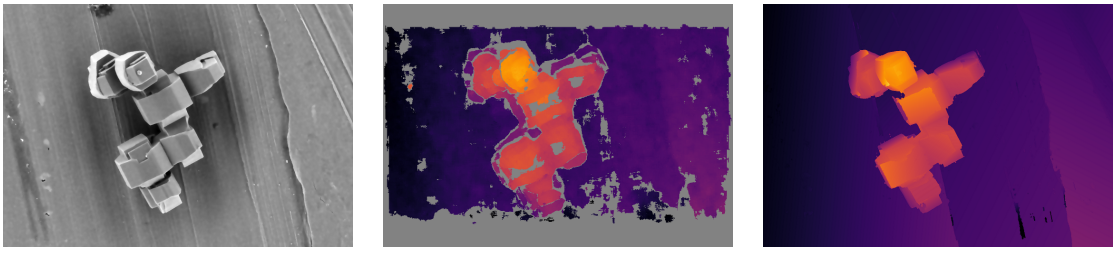
(d) *(c) after filling undefined values*

(e) *(d) after LRC check*

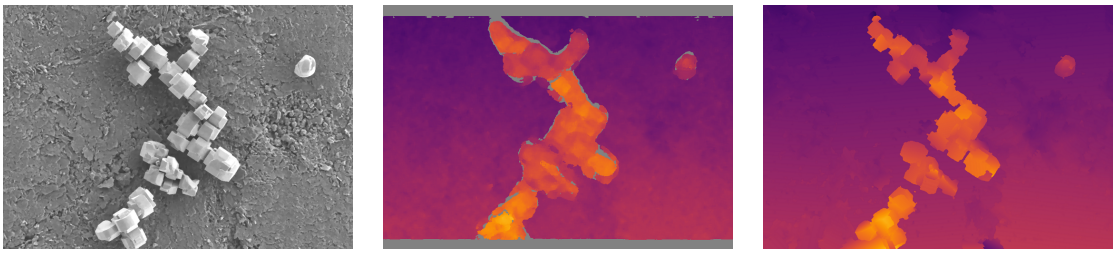


(f) *(e) after filling undef. values: final result*

Figure 6.16: *Post-processing chain of the model map produced by TDSR.*

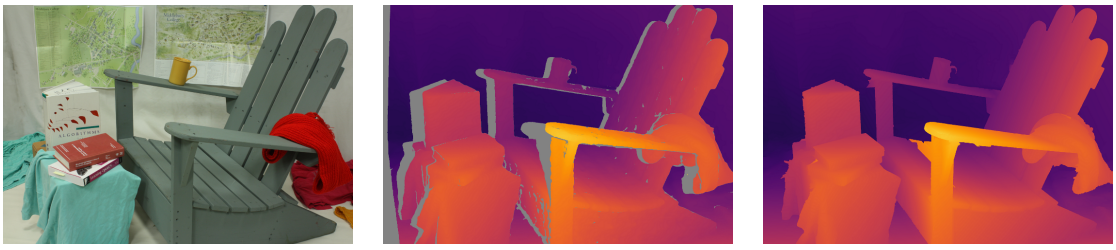


(a) Reference image *(b) Sparse disparity map* *(c) TDSR result*

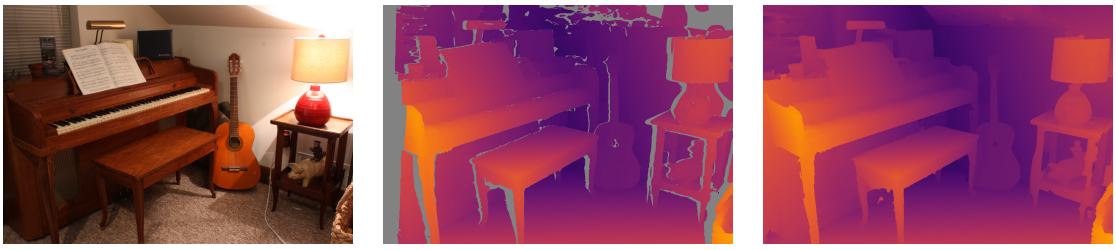


(d) Reference image *(e) Sparse disparity map* *(f) TDSR result*

Figure 6.17: *TDSR results on some SEM pairs of images*

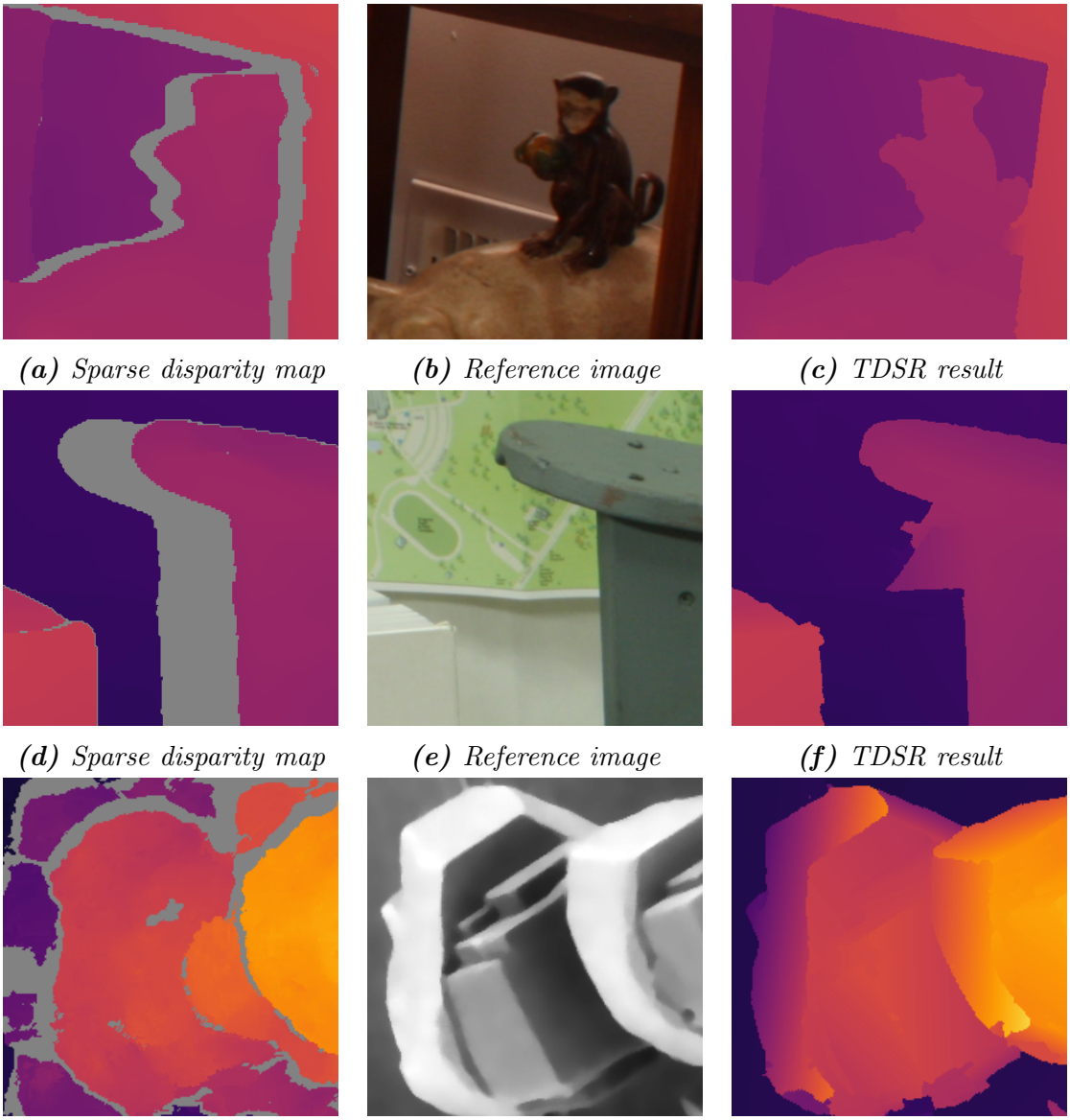


(a) Reference image *(b) Sparse disparity map* *(c) TDSR result*



(d) Reference image *(e) Sparse disparity map* *(f) TDSR result*

Figure 6.18: *TDSR results on some Middlebury pairs*



(a) *Sparse disparity map* ***(b)*** *Reference image* ***(c)*** *TDSR result*

(d) *Sparse disparity map* ***(e)*** *Reference image* ***(f)*** *TDSR result*

(g) *Sparse disparity map* ***(h)*** *Reference image* ***(i)*** *TDSR result*

Figure 6.19: Some characteristics of the TDSR's results. *(a), (b), (c) demonstrate how the shapes of the images are better respected after the TDSR post-processing. (d), (e) and (f) show some occlusions that are still not well processed by the algorithm. (g), (h) and (i) illustrate some homogenous region that have a too complex shape to be successfully fitted by a plane.*

The process is slightly faster (20%) than the previous algorithm. Although the structure of the partition trees are similar as well as the total number of nodes, the disparity computation is externalized to the OpenCV library which is much faster. The region modelization continues though to take most of the time, mainly because of the regressions. However, this process could be easily parallelized.

As shown in Section 7.3.1, the algorithm can be easily extended to 3 views or more by combining the initial disparity maps into a single one. Apart from the disparity map estimation, the process doesn't change, so there is very little overhead in computation time.

We restricted the models we used to linear models. Though we could easily use more complex models, it is important to note that the more complex a model is, the more likely it will overfit disparity values in the regions, leading to errors and potentially extreme values. When choosing the model used, it is therefore important to check that the initial disparity map is exact and complete enough for the complexity of the model.

As for the running time, there are multiple factors that can influence it. The method can be separated into two main parts - the disparity map estimation and the TDSR post-processing method - that depend on different factors. The necessary time for the disparity map estimation will mainly depend on $w \times h \times s$, w being the width of the image, h being the height of the image, and s being the search area's size (that is related to the height of the sample). Although, since this part is well optimized, the necessary time to complete it constitutes only a small portion of the overall running time on standard SEM images (1536 $\times$ 1024 pixels). The TDSR post-processing method takes around 80% of the total running time: its completion time is not really dependent on the size of the image, but has a linear relationship with the number of detected homogeneous regions in the image. This number will be related to the number of homogeneous faces in the sample, but also on the amount of noise: if the image is noisy, a lot of regions will be detected, hence increasing the running time. Applying an adapted filter on the SEM images to reduce the noise is therefore very important: we will address this issue in Section 7.3. Furthermore, in Section 6.5, we suggest ways to reduce the running time.

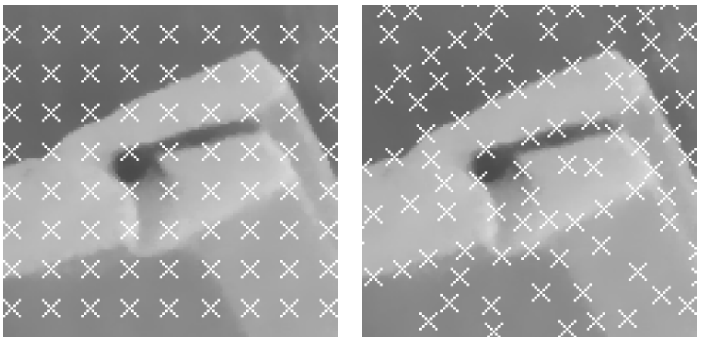
Finally, our algorithm depends on a number of parameters (h, t_d, t_s...). Those parameters were chosen so that the average error between the disparity map produced by our algorithm and the ground truth in the Middlebury database is minimized. Though those parameters play an important role, the performance of TDSR is not very sensitive to the parameters around the optimal settings: for instance choosing $t_s = 90\%$ or $t_s = 60\%$ instead of $t_s = 70\%$ doesn't dramatically change results. To further improve results, we suggest to focus instead on the post-processing steps of the algorithm.

6.5 Speed up of TDSR using adaptive subsampling

As a reminder, we need to return an estimation of our sample between 30 to 60 seconds. One of the longest steps is the modelization of the regions, where points are fitted using the RANSAC regression algorithm. A way we can speed up the process is by reducing the number of points to be fitted.

We know that the higher a point's gradient is, the greater is the probability that its disparity value will be correct. Instead of subsampling these points using a naive approach, we propose a method that uses this property to our advantage. The selected points should therefore be equally spaced, but in priority over the high gradient of the reference image. See Figure 6.20.

This concept is implemented in Algorithm 16. As shown in Figure 6.21, provided that a reasonable amount of points are removed (a low size), the adaptive subsampling can produce similar results while providing a 25% speed up.



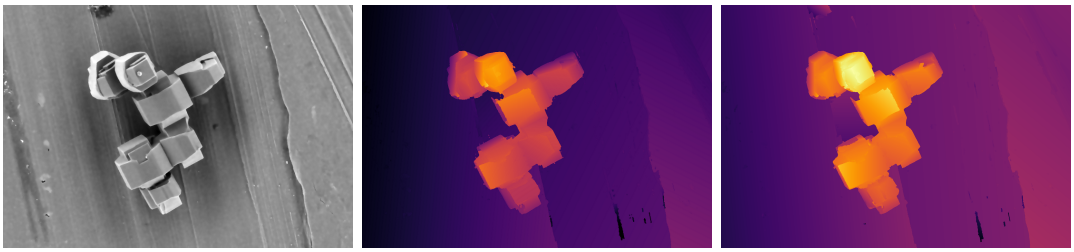
(a) *Naive subsampling* ***(b)*** *Objective*

Figure 6.20: Objective of the adaptive subsampling method.

Algorithm 16: Adaptive subsampling

Function *adaptiveSubsampling(g, size)*

```
/* g is a gradient the reference image, size is the block size
   used during the disparity map estimation.                    */
listPoints ← empty list ;
sortedPoints ← points in g ordered from the highest to the lowest value ;
covered ← binary matrix initialized to False with same size of g;
foreach point in sortedPoints do
    if covered(point) is False then
        add point to listPoints ;
        Set covered to True on the block of size size centered on point ;
return listPoints ;
```



(a) Reference image *(b) TDSR (50s)* *(c) TDSR + AS (38s)*

Figure 6.21: TDSR results with and without adaptive subsampling. *AS: Adaptive Subsampling. Size chosen for adaptive sampling: 7px. CPU: 2.7Ghz i7-4800MQ. RAM used: 200-400 MB.*

6.6 Summary

We have presented a disparity map post-processing method that takes advantage of the hierarchical segmentation of the reference and secondary images. The results are much more promising than the previous method presented in Chapter 5. There are however still some discrepancies around strong borders, and the quality of the produced disparity map is of course very dependent on the quality of the input sparse disparity map.

We have also introduced a new concept: the **model map**. Instead of defining on each pixel of an image a disparity value, we separate the image into different regions and we define for each region a model, a mathematical function $f(x, y)$ that represents the disparity on the whole region. We have also created custom operations on model maps that allow to propagate models on undefined values for instance.

We will now present in more detail the results obtained by our algorithms.

Chapter 7

Applications and results

French summary / Résumé en français

Nous présentons dans ce chapitre différents résultats obtenus par les méthodes que nous avons développés. En première partie, nous présentons nos travaux réalisés dans le cadre du concours Multi-View Stereo 3D Mapping Challenge : à partir d'un ensemble pouvant compter jusqu'à 50 images satellitaires, l'objectif était de reconstruire la topographie de zones urbaines. En seconde partie, nous présentons les résultats obtenus sur la base de donnée Middlebury, qui contient des paires d'images stéréoscopique de scènes d'intérieur associées à des vérités terrains : il s'agit de la base de donnée de référence utilisée pour comparer les méthodes de reconstruction stéréoscopique. Enfin, nous estimerons notre méthode sur l'objectif initial de la thèse : la reconstruction stéréoscopique à partir d'images de catalyseur acquises à l'aide d'un microscope électronique à balayage.

As a reminder, the main objective of the thesis is to provide a solution that automatically estimates the 3D topography of a microscopic sample. The methodology used, stereo reconstruction, is however general: there are multiple possible applications that go beyond the microscopic scale.

In this chapter, we will present three applications of stereo reconstruction we worked on during this thesis. We will also show qualitative and quantitative results for each case. See Figure 7.1.

First, we will present our work on the Multi-View Stereo 3D Mapping Challenge organized by the Intelligence Advanced Research Projects Activity (IARPA) and Topcoder. From a set of up to 50 satellite images, the objective was to estimate the topography of urban areas.

Secondly, we will present the results we obtained on the Middlebury dataset, which contains multiple pairs of images of indoor scenes associated with disparity maps ground truths. As this is one of the standard datasets used to benchmark stereo reconstruction algorithms, it will allow us to easily compare our solution with the state of the art.

Finally, we will present the retained methodology for estimating the topography from SEM images, as well as some results.

(a) Satellite image $1px \approx 30cm$ | *(b) Middlebury image* $1px \approx 0.5\text{-}2mm$ | *(c) SEM image* $1px \approx 20nm$

Figure 7.1: Images from the different stereo reconstruction applications we will present in this chapter.

7.1 The Multi-View Stereo 3D Mapping Challenge

The Multi-View Stereo 3D Mapping Challenge was organized by the Intelligence Advanced Research Projects Activity (IARPA) and Topcoder and took place from July 2016 to December 2016. The objective of the challenge was to foster and stimulate research on multiview stereo reconstruction applied on satellite images.

There is a growing interest in obtaining the 3D topography of cities. Mapping services, urban planning, disaster management, or defense: these are only a few examples of applications where having a precise representation of cities brings an added value to decision-making.

Though obtaining the topography of cities can be done using technologies such as LiDAR [103], the acquisition cost is expensive. On the other hand, there are various satellites that orbit earth and continuously acquire images of the overflown areas. It is therefore possible to get lots of different images of the same area taken from different viewpoints. See Figure 7.2. Using multiview stereo reconstruction algorithms can therefore be a promising lead to obtain topographies at lower cost.

7.1.1 Overview of the challenge

The objective of the contest was to obtain the most accurate and complete topography of an area from up to 50 images acquired using the WorldView-3 satellite. The reconstruction had to be completed in less than 2 hours.

(a) *Satellite image 1* ***(b)*** *Satellite image 2* ***(c)*** *Satellite image 3*

Figure 7.2: Three satellite images of the same area.

The contest was separated in two independent parts, the **Explorer** contest and the **Master** contest. The submitted methods were quantitatively compared on multiple urban areas with a known ground truth (acquired using the LiDAR technology). Two following criteria were used to compare the methods:

- Accuracy (α). The accuracy was computed using the median error in meters during the Explorer contest and using the root mean square error in meters during the Master contest.
- Completeness (β): the fraction of points whose error is less than 1 meter. Points that are not defined count as if their error is more than 1 meter.

These two criteria are then relativized compared to the best solution:

$$\alpha_r = \frac{\alpha^*}{\alpha} \text{ and } \beta_r = \frac{\beta}{\beta^*}$$

α^* and β^* being respectively the best α and β obtained among all submissions.

The final score used to rank submitted solutions was:

$$score = 500,000 \times (\alpha_r + \beta_r)$$

Contestants had therefore to submit a solution that maximized this score.

Finally, in the Master contest, a training set was also provided: the ground truth of one area was available, allowing us to quantitatively optimize the solution's score.

7.1.2 Approach taken

We will address the problem in two steps. First, we will describe the algorithm estimating the height map from a pair of images. Then, we will extend this pair-wise stereo algorithm to a multiview stereo algorithm. Methods used in the first step are standard whereas we developed a new approach in the second step.

The overall algorithm for obtaining a 3D topography from a pair of images follows the same pipeline described in Chapter 4: first we compute a disparity map, and then, using geometry, we transform this disparity map into a height map.

The acquisition system of the WorldView-3 satellite differs from standard cameras or SEMs. Indeed, images are acquired using a push broom scanner. Instead of acquiring the whole image in a single shot, light sensors in a push broom scanner are positioned on a single line and the 2D image acquisition is made by moving the line of sensors in the perpendicular direction (similarly to standard scanners). See Figure 7.3. The consequence is that, since the satellite moves during the acquisition, the projection model is different from the pinhole and orthographic models.

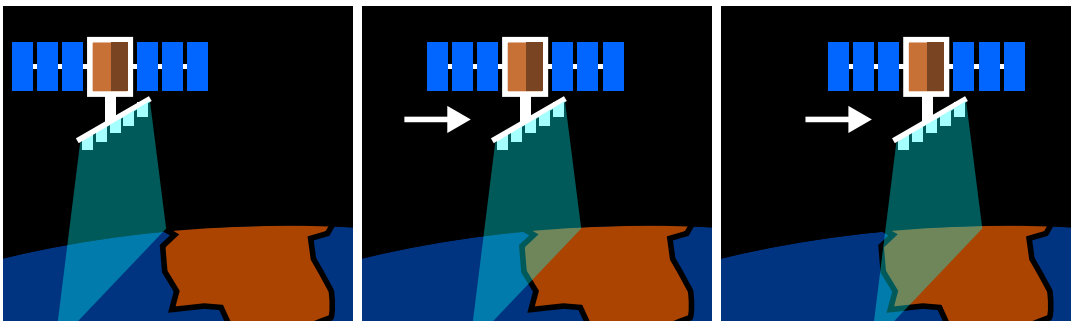


Figure 7.3: How a 2D image is acquired using the a push broom scanner.

This specific projection model makes the rectification process difficult [104]. In addition, images provided for the contest covered a much larger area that what was needed, so a specific crop, using Rational Polynomial Coefficients (RPCs) [105], was necessary. As these steps were specific for satellite images and were not the focus of this thesis, we decided to rely on the NASA Ames Stereo Pipeline [106].

We propose the overall process shown in Figure 7.4.

First, we use the NASA Ames Stereo Pipeline to do some preprocessing on the pair of images: extraction, bundle adjustment, image filtering and rectification.

We then estimate a disparity map using the SGBM method (block size: 15px) described in Section 4.3 and refine it using the FGS diffusion algorithm ($\lambda = 8000$, $\sigma = 0.25$, $DDR = 3$) described in Section 4.4.

Then, using the NASA Ames Stereo Pipeline, we transform this disparity map into a 3D map.

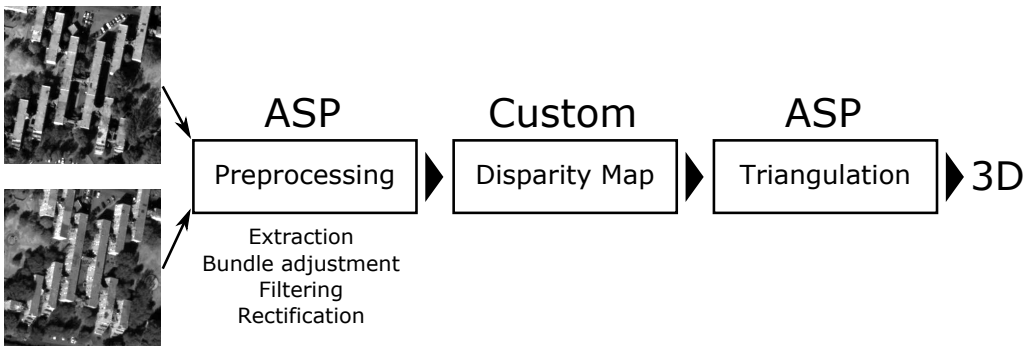


Figure 7.4: Overall proposed process. *ASP: NASA Ames Stereo Pipeline.*

This is how we process a pair of images, but there can be up to 50 images of the same area, that is to say more than 2000 potential pairs. We had to provide an estimation within a limited amount of time (2 hours), so we had to choose a subset of pairs.

It is important to note that most pairs do not provide satisfying results. A major difference in the provided dataset compared to more classical dataset is that images are not acquired at the same time; there is more than one year difference between the earliest and the latest images.

This creates additional challenges for the matching process. See Figure 7.5. Indeed, images taken on different seasons feature various differences: for instance, trees have leaves in summer whereas they don't have leaves in winter, the length and orientation of the buildings' shadows is also different depending on the acquisition date and hour. These differences make the matching process fail, so it is important to choose pairs of images that minimize errors. Therefore, pairs were chosen so that both images were acquired during the same season and approximately the same hour of the day. This selection was done manually but it could be done automatically as these information are available inside the images' metadata.

Once this selection is done, our solution generates the 3D map for each selected pair. If this generation process takes more than one hour, the process is stopped to avoid exceeding the time limit. Then, the algorithm merges all these maps following the consensus process described in Algorithm 17. The process also returns a confidence map which measures the consensus size. See results in Figure 7.6.

We have explained in the previous section (7.1.1) that two criteria affect the score on which submissions were ranked during the contest: accuracy and completeness. Pixels that are misevaluated by our algorithm have generally a low confidence value. We can therefore assume that, while removing low confidence values reduces completeness, it also improves accuracy. We verified this hypothesis by plotting the relationship between accuracy, completeness and confidence thresholds (maps where heights with a confidence lower than a threshold are removed). See Figure 7.7.

For optimizing the score of our submission we chose to remove values whose confidence were lower than 70% of the number of processed pairs.

(a) *Trees with leaves* ***(b)*** *Trees without leaves*

(c) *Short shadow* ***(d)*** *Long shadow*

Figure 7.5: Problems appearing on pairs of images taken on different seasons.

Algorithm 17: Combination of several height maps into a single one.

Function *mergeHeightMaps(heightMaps, t)*

```
/* heightMaps is a list of height maps obtained for the same
   area.  t is the threshold under which we consider that two
   heights are sufficiently close to be aggregated (we chose
   2.0m).                                                    */
```

Align all the height maps in $heightMaps$ using correlation ;

```
/* Height maps are aligned because there might still be some
   errors in the evaluation of the position and orientation of
   the satellite, even after bundle adjustment.              */
```

$heightMap \leftarrow$ new matrix of same size of $heightMaps[1]$;

$confidenceMap \leftarrow$ new matrix of same size of $heightMap$;

foreach *point (x, y) in heightMaps* **do**

```
// For each point, we have multiple heights estimations.
```

$heights \leftarrow$ list of heights in $heightMaps$ on (x, y) ;

```
// We select the largest consensus of the heights estimations.
```

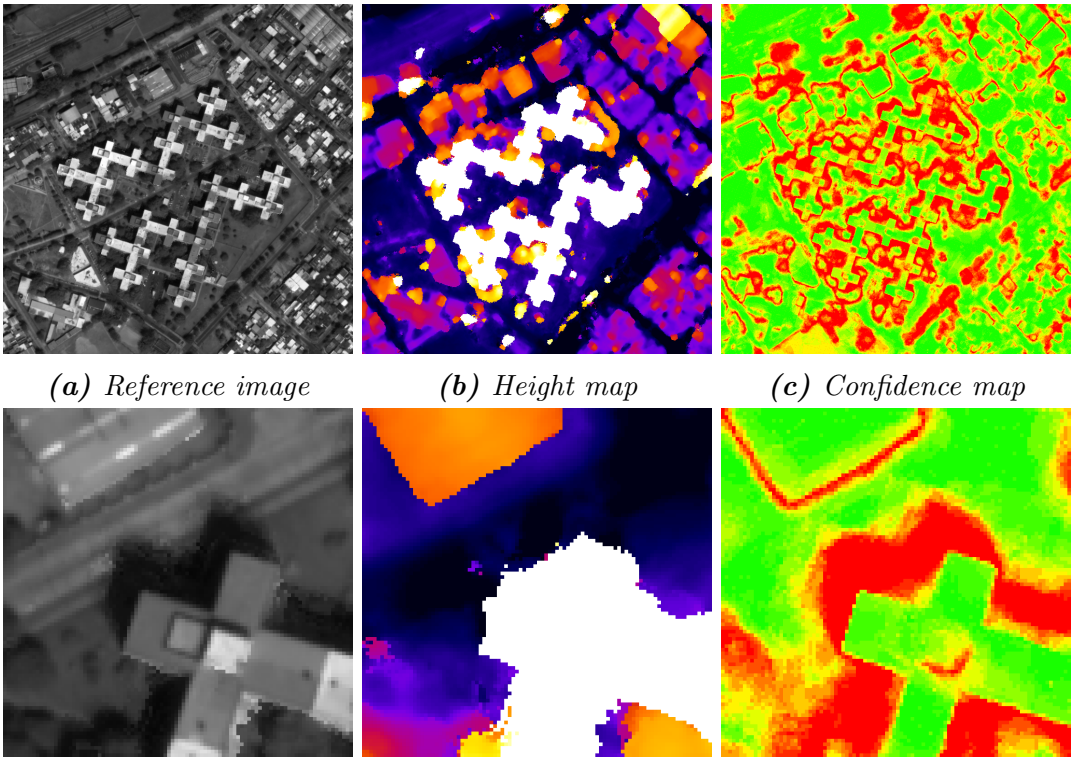
$S \leftarrow$ largest set in $heights$ with a range $< t$;

$heightMap(x, y) \leftarrow$ average of S ;

$confidenceMap(x, y) \leftarrow$ number of elements inside S ;

```
// The largest the consensus is, the higher the confidence.
```

return $heightMap$, $confidenceMap$;



(a) Reference image *(b) Height map* *(c) Confidence map*

(d) Extract of (a) *(e) Extract of (b)* *(f) Extract of (c)*

Figure 7.6: Example of an height map and confidence map obtained by our solution.

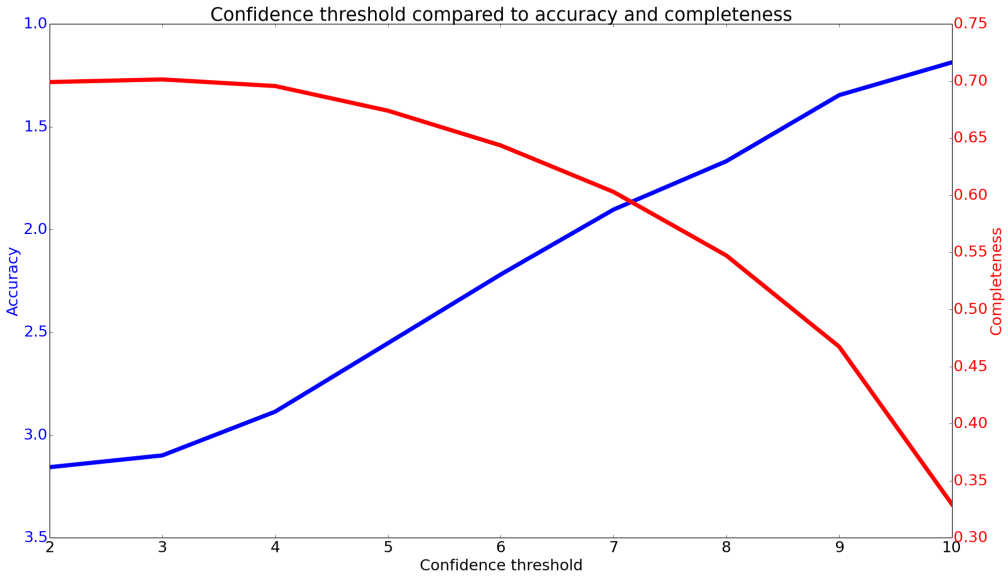


Figure 7.7: Relationship between accuracy, completeness and confidence. *The merging algorithm was applied on 10 pairs of images, so possible confidence values are between 2 and 10.*

7.1.3 Results

Our solution was ranked 1st in the Explorer challenge and 3rd in the Master Challenge. 34 teams took part in the contest, and 369 subscribed.

Our submission also won the open-source incentive and was published at the following url:

`http://github.com/sdrdis/iarpa`

We were invited to present our solution in a workshop organized for the challenge at Washington on November 30th 2016.

Results of the different submissions for the Master Challenge are further discussed in [107]. A quantitative comparison of top submissions is shown in Table 7.1 and a qualitative comparison is shown in Figure 7.8 (our submitted solution is named "ASP-SDRDIS"). Though we achieve comparable quantitative results, our submission suffers from the fattening effect as discussed in Section 4.2.4: foreground objects tend to be fattened in the height map estimation. The overflow effect created by the FGS diffusion algorithm and discussed in Section 4.4 is also visible.

The combination of these two effects is clearly visible when the height maps are compared with the building blueprints of the areas in Figure 7.8c: our results clearly tends to overestimate the width of the buildings. These drawbacks are not due to the merging algorithm - the overflown area are generally detected as they are associated with low confidence values - but to the quality of the algorithms used for the pairs' disparity map estimation - SGBM and FGS - which are optimized for speed and not accuracy. Better algorithms used at this step might produce better results.

However, these errors are generally associated with low confidence values, so the accuracy score can be improved by removing those points.

7.1.4 Summary

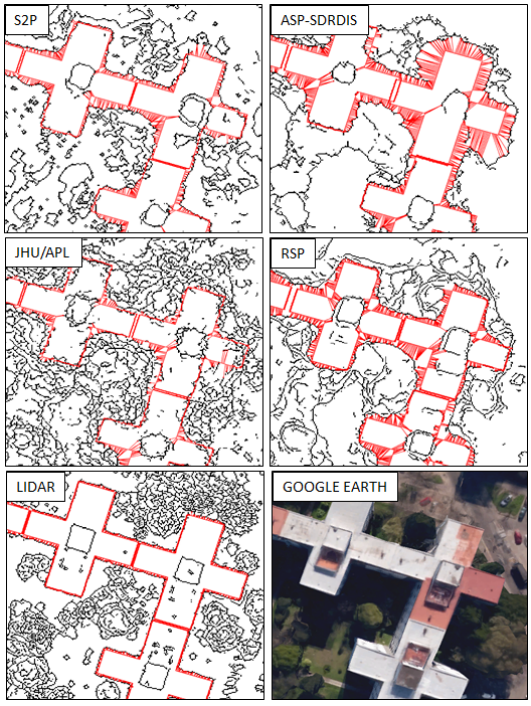
The multiview stereo method we developed for the contest has various advantages:

- It is general: it could be applied on other images such as our SEM images.

Metric	S2P	JHU/APL	ASP SDRDIS	RSP
Median Z Error (m)	0.37	0.47	0.39	0.35
Z RMSE (m)	2.59	2.20	2.31	2.27
Completeness	73.2%	64.1%	68.7%	69.4%

Table 7.1: Quantitative results for the top submissions results for the IARPA Master Challenge. *Our submitted solution is named "ASP-SDRDIS". Table from [107].*

(a) *Top submissions results* ***(b)*** *Zoomed top submission results*



(c) *Offset between building blueprints and height maps*

Figure 7.8: Top submissions results for the IARPA Master challenge. *Our submitted solution is named "ASP-SDRDIS". Images from [107].*

- It is easily extendable: it can work on a single pair of images as well as 50 pairs of images. It is a matter of choosing the performance - processing time trade-off.
- It provides a confidence map allowing us to choose the accuracy - completeness trade-off.
- It doesn't require a lot of memory (2-3 Gb on satellite images).

There are however several possible improvements:

- The process could be easily optimized as disparity map estimations are not parallelized.
- Low-confidence heights are simply removed but we could fill these values using diffusion algorithms.
- Finally, we could use more advanced algorithms for the disparity map evaluations than SGBM and FGS. However, as the TDSR method was not mature enough at the time of contest and there were strong time constraints, there was no time to research more adapted solutions.

7.2 The Middlebury dataset

We benchmarked our TDSR method using the well-known Middlebury dataset[90] (discussed in Section 4.6). As a reminder, the dataset contains several stereo pairs of indoor scenes associated with disparity map ground truths. There is also an official website that independently ranks state of the art stereo methods on multiple criteria. As provided images are already rectified and the algorithms are evaluated on their produced disparity map, there is no need to evaluate the camera's parameters and to estimate the scenes' height maps.

We chose the following two commonly used criteria in the Middlebury ranking: "Bad 2.0", which computes the percentage of disparity values farther than 2.0 pixels from their associated ground truth, and "Avg. error", which computes the average absolute difference between the disparity values and their associated ground truth. They were computed on full resolution (F) images.

We compared the results of our TDSR method with three interpolations techniques directly applied on the sparse disparity map: nearest, linear, and cubic.

We also compared our results with the FGS disparity map post filtering method, described in Section 4.4. However, this method has three parameters -λ, σ, DDR - bringing additional challenges to the comparison.

Therefore, we computed three complete maps: one with the default parameters, one where the parameters have been optimized to minimize "Bad 2.0", and one where the parameters have been optimized to minimize "Avg. error". The parameters have been optimized using a grid search. Tested parameters are shown in Table 7.2.

Parameter	Tested	Default	"Bad 2.0" optimized	"Avg. error" optimized
DDR	1, 3, 5, 7, 10	5	3	5
λ ($\times$1000)	0, 0.1, 0.3, 0.5, 1, 2, 4, 6, 8, 10, 12, 14, 16, 20	8	0.1	6
σ (/10)	1, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20, 30	10	10	15

Table 7.2: Tested parameters for FGS.

Finally, we compared our results with the RBS [78] method, which uses a variant of the bilateral algorithm to propagate disparity values while preserving edges.

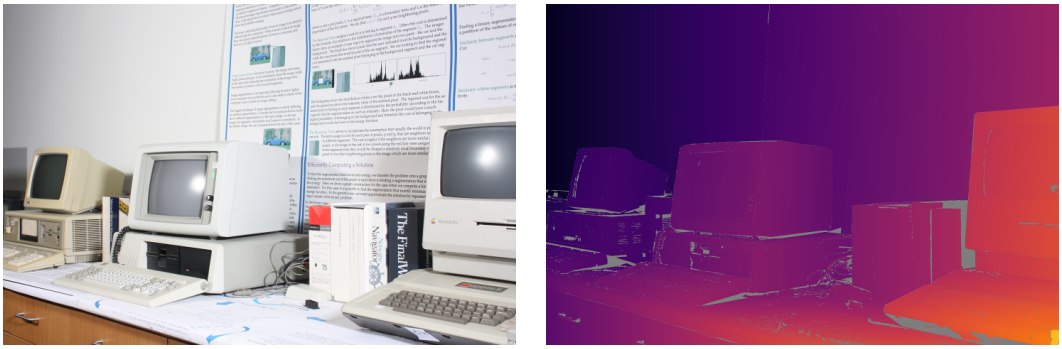
Produced disparity maps can be qualitatively compared on the "Vintage" pair of the Middlebury dataset in Figure 7.9. They can be quantitatively compared on the whole training set in Table 7.3.

Method	Bad 2.0 (no occ)	Avg. error (no occ)	Bad 2.0 (all)	Avg. error (all)
Nearest interpolation	10.1	2.83	18.3	7.53
Linear interpolation	10.6	2.71	20.2	7.61
Cubic interpolation	10.6	2.75	20.1	7.70
FGS (Optimized on Bad 2.0)	10.4	2.62	17.4	8.20
FGS (Default)	16.5	2.60	23.1	6.80
FGS (Optimized on Avg. error)	21.3	2.85	27.7	6.61
MC-CNN+RBS[78]	10.8	2.60	19.3	6.66
MC-CNN+TDSR	***10.2***	***2.28***	***15.6***	***4.56***
$\rightarrow$ *Improvement to best interp.*	*-1%*	*16%*	*15%*	*40%*
$\rightarrow$ *Improvement to best FGS*	*2%*	*12%*	*10%*	*31%*

Table 7.3: Quantitative comparison between interpolation, FGS, RBS, and our proposed TDSR method. *all: Taking all points into account. no occ: Taking only non-occluded points into account.*

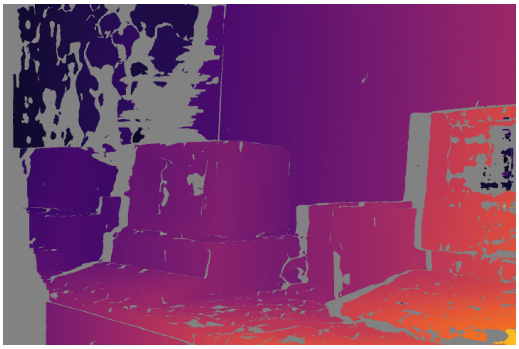
Interpolation methods are very sensitive to the noise in the sparse disparity map and they don't work well on occlusions. Depending on the chosen parameters, the Fast Global Smoothing method can either produce a less noisy disparity map but with the risk of having the disparity of some objects contaminated by neighboring ones (high λ, optimized for "Avg. error"), or more noise but less contamination (low λ, optimized for "Bad 2.0"). Default parameters give a good tradeoff between both effects.

Qualitatively, compared to the interpolation, FGS and RBS methods, our approach appears robust to noise and seems to globally manage occlusions. However, some occlusions, notably near the computer screen in the foreground (see Figure 7.9f), are not well evaluated due to lack of data. Some regions can also have their disparity contaminated by nearby objects with similar color.

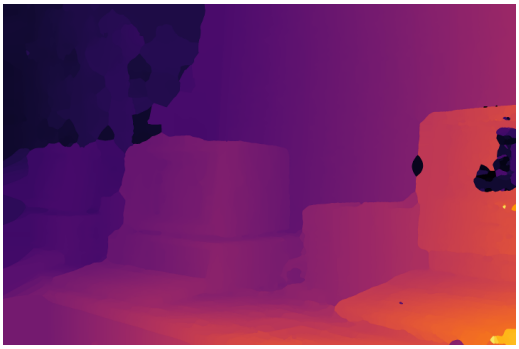


(a) Left image

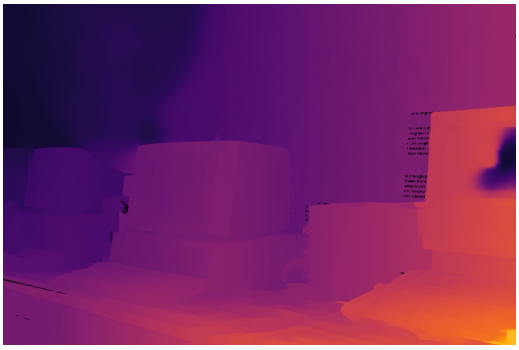
(b) Ground truth



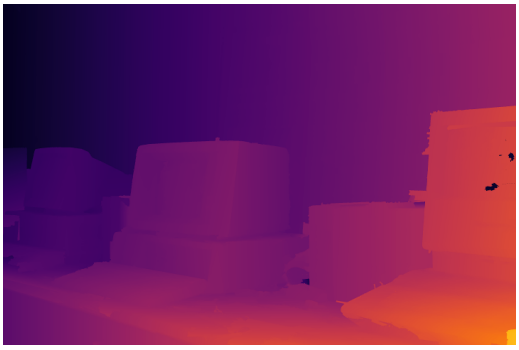
(c) Sparse disparity map



(d) Nearest interpolation



(e) FGS with default parameters



(f) TDSR

Figure 7.9: Image and disparity map produced for the Vintage pair of the Middlebury dataset.

Set	Metric	Mask	Ranking (/ 59)
training dense	bad 2.0	all	4
training dense	avgerr	all	1
test dense	bad 2.0	all	1
test dense	avgerr	all	3

Table 7.4: Rankings of MC-CNN+TDSR as of March 10, 2017.

Quantitatively, compared to the interpolation, FGS and RBS methods, whether we choose to only look at non occluded pixels or to look at all the pixels, our method produces at least similar results according to all chosen criteria. It is important to note that our algorithm's parameters were fixed, though we optimized the parameters of the FGS algorithm for each criterion and compared our algorithm's results to the best cases of the interpolation and FGS methods.

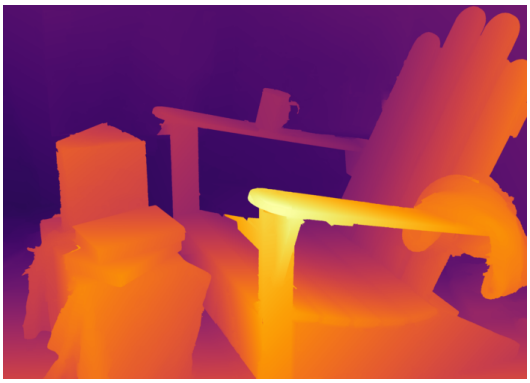
The solution has been submitted to the Middlebury ranking website on March 10, 2017. Our entry is named **MC-CNN+TDSR**, and the official ranking is available at the following url:

`http://vision.middlebury.edu/stereo/eval3/`

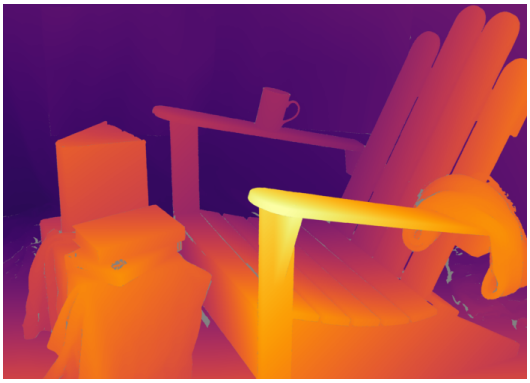
Please note that the ranking is continuously updated. As of March 10, 2017, our method is comparable and sometimes better than state of the art. See Table 7.4.

Results of the algorithm for all pairs of the official dataset can be found in figures 7.10 and 7.11. Quantitative results on the chosen criteria for each pair can be found in Figure 7.12.

(a) Adirondack ref. im.



(b) TDSR



(c) Ground truth

(d) ArtL ref. im.

(e) TDSR

(f) Ground truth

Figure 7.10: Results obtained on the training set of the Middlebury dataset. *ref. im.: reference image.*

(g) Jadeplant ref. im.

(h) TDSR

(i) Ground truth

(j) Motorcycle ref. im.

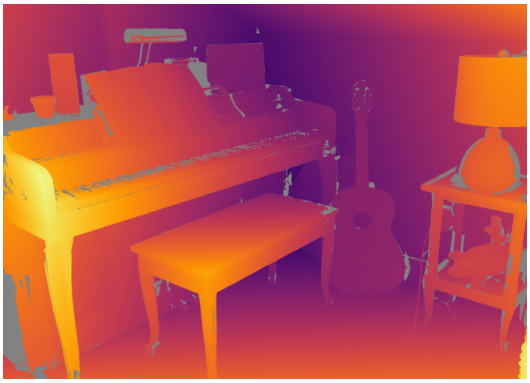
(k) TDSR

(l) Ground truth

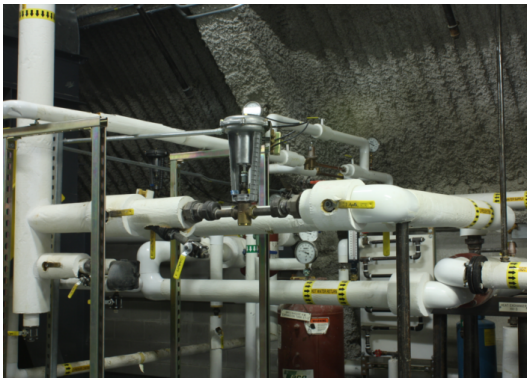
Figure 7.10: Results obtained on the training set of the Middlebury dataset (cont.). *ref. im.: reference image.*

(m) *Piano ref. im.*

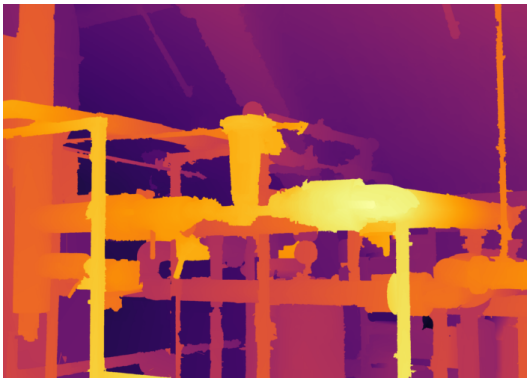
(n) *TDSR*



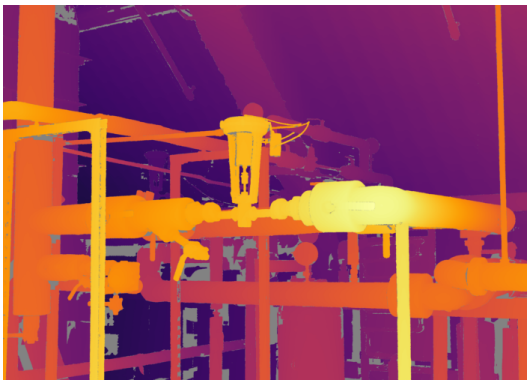
(o) *Ground truth*



(p) *Pipes ref. im.*



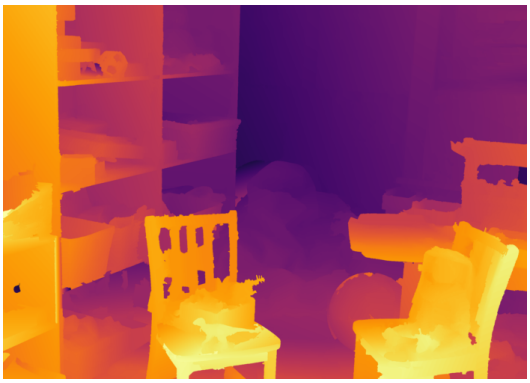
(q) *TDSR*



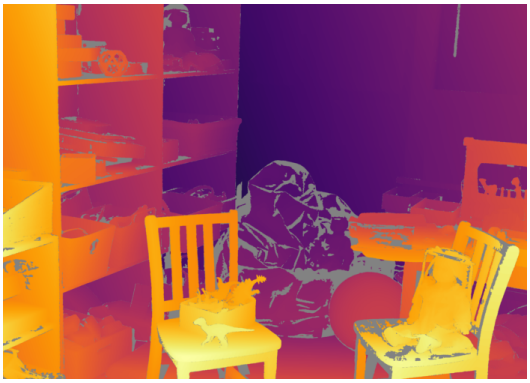
(r) *Ground truth*

Figure 7.10: Results obtained on the training set of the Middlebury dataset (cont.). *ref. im.: reference image.*

(s) *Playroom ref. im.*

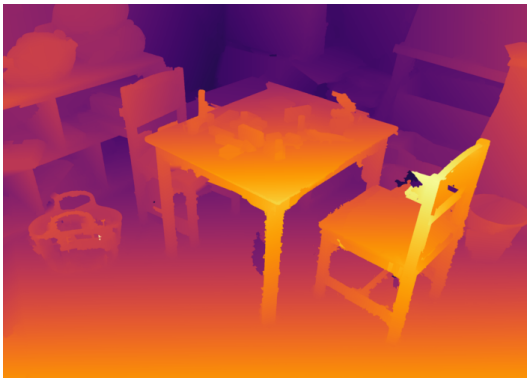


(t) *TDSR*

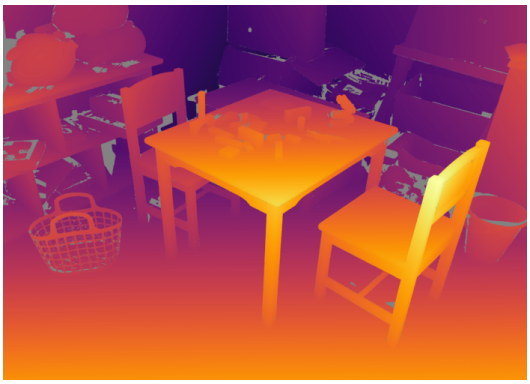


(u) *Ground truth*

(v) *PlaytableP ref. im.*



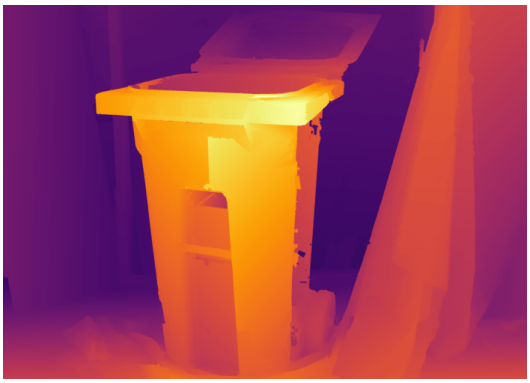
(w) *TDSR*



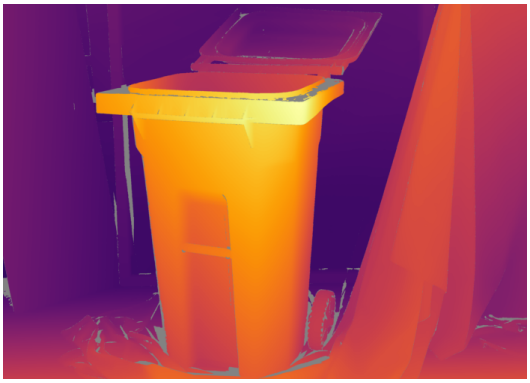
(x) *Ground truth*

Figure 7.10: Results obtained on the training set of the Middlebury dataset (cont.). *ref. im.: reference image.*

(y) *Recycle ref. im.*

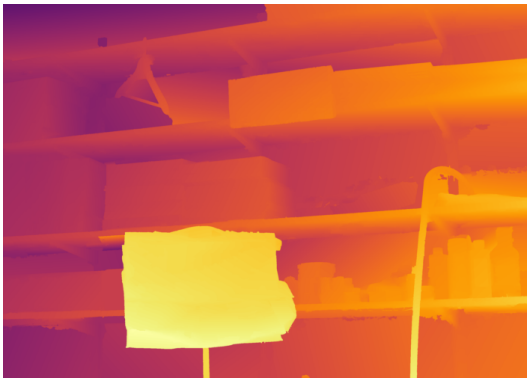


(z) *TDSR*

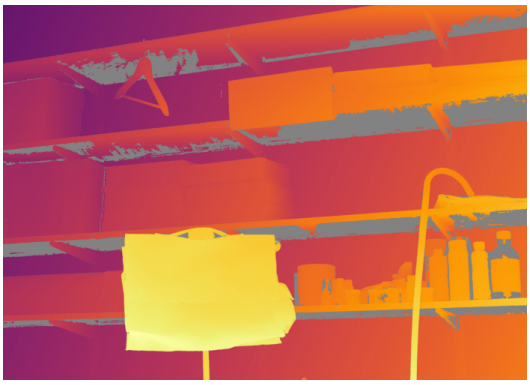


(aa) *Ground truth*

(ab) *Shelves ref. im.*



(ac) *TDSR*



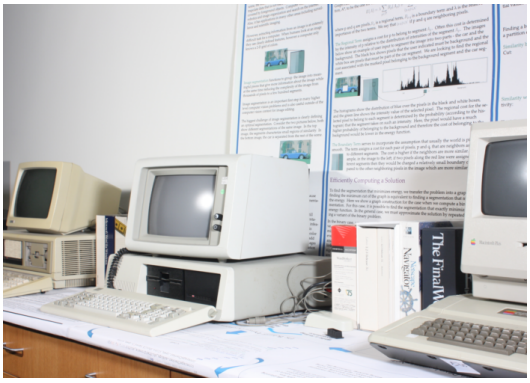
(ad) *Ground truth*

Figure 7.10: Results obtained on the training set of the Middlebury dataset (cont.). *ref. im.: reference image.*

(ae) *Teddy ref. im.*

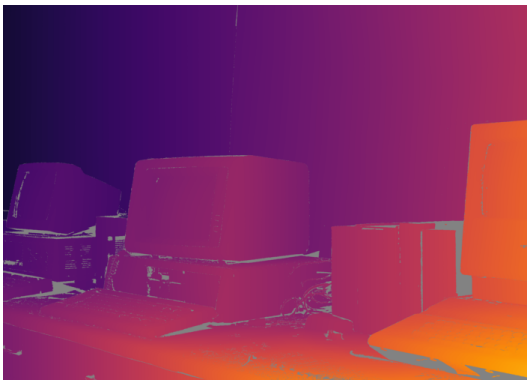
(af) *TDSR*

(ag) *Ground truth*



(ah) *Vintage ref. im.*

(ai) *TDSR*



(aj) *Ground truth*

Figure 7.10: Results obtained on the training set of the Middlebury dataset (cont.). *ref. im.: reference image.*

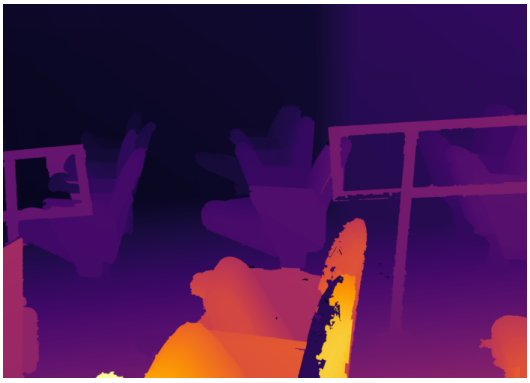
(a) AustraliaP ref. im.

(b) TDSR

(c) Bicycle2 ref. im.

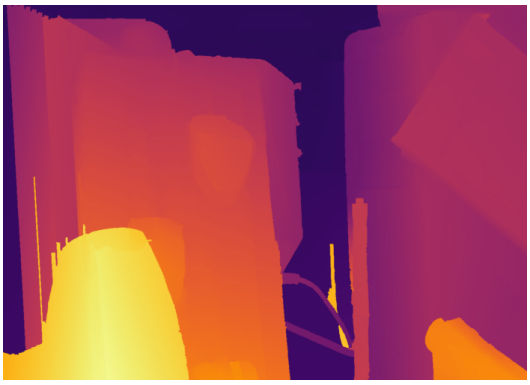
(d) TDSR

(e) Classroom2 ref. im.



(f) TDSR

(g) Computer ref. im.



(h) TDSR

Figure 7.11: Results obtained on the testing set of the Middlebury dataset.
ref. im.: reference image. Ground truths are not available to prevent overfitting.

(i) *CrusadeP ref. im.*

(j) *TDSR*

(k) *Djembe ref. im.*

(l) *TDSR*

(m) *Hoops ref. im.*

(n) *TDSR*

(o) *Livingroom ref. im.*

(p) *TDSR*

Figure 7.11: Results obtained on the testing set of the Middlebury dataset (cont.). *ref. im.: reference image. Ground truths are not available to prevent overfitting.*

(q) *Newkuba ref. im.*

(r) *TDSR*

(s) *Plants ref. im.*

(t) *TDSR*

(u) *Staircase ref. im.*

(v) *TDSR*

Figure 7.11: Results obtained on the testing set of the Middlebury dataset (cont.). *ref. im.: reference image. Ground truths are not available to prevent overfitting.*

Pair	bad 2.0	avgerr
Adiron	4.75	1.28
ArtL	19.8	6.55
Jadepl	30.2	18.4
Motor	8.94	3.53
MotorE	8.61	3.46
Piano	14.9	2.03
PianoL	19.8	4.76
Pipes	15.5	7.23
Playrm	26.1	4.12
Playt	18.3	3.27
PlaytP	16.8	2.49
Recyc	12.3	1.74
Shelvs	29.0	3.38
Teddy	9.43	1.55
Vintge	15.3	*2.07**
Avg	**15.6**	**4.56**

(a) *Training set*

Pair	bad 2.0	avgerr
Austr	8.68	5.40
AustrP	7.41	5.04
Bicyc2	9.71	3.13
Class	*10.0**	6.74
ClassE	17.4	13.3
Compu	18.0	*5.32**
Crusa	10.7	6.36
CrusaP	12.7	5.56
Djemb	5.17	*1.18**
DjembL	13.0	2.29
Hoops	*21.1**	*8.19**
Livgrm	12.6	4.48
Nkuba	*17.4**	*5.55**
Plants	12.0	*9.77**
Stairs	*11.0**	*5.99**
Avg	**12.1**	**5.66**

(b) *Test set*

Figure 7.12: Quantitative results of our method on the Middlebury dataset (extracted from the Middlebury website). *When a result is appended with a star, our solution ranks first on this criterion and pair. Selected mask: all.*

7.3 SEM images

In the previous section, we have validated our approach on the standard stereo dataset Middlebury. Our objective is however to provide a solution working on SEM images. There are two major differences:

- First, we need to return a height map and not a disparity map. We must therefore determine the projection model.
- Our SEM images are not rectified: input images might be misaligned or there might be distortions. Points also mainly move on the Y axis between the reference and the secondary images, and most stereo methods work on the X axis. Furthermore, SEM images contain a lot of noise that might disturb the matching process. We need therefore to transform the images so that we are able to estimate correct disparity maps.

In Appendix A, we study the projection model and the distortions observed in SEM images at a similar magnification than our samples' acquisitions. We also show how the camera and distortions parameters can be recalibrated using a sequence of images. As this part doesn't represent the core of our work, we will suppose that these factors are known in the following sections. We will also suppose that images have been preliminarily aligned following the process described in Appendix A.

7.3.1 Height map estimation process

As there is an important amount of Poisson noise in the SEM images (discussed in Section 2.4), we reduce it using the combination of a median filter followed by a bilateral filter both discussed in Section 3.2.1. We produce two sets of denoised images: one that will be used for the disparity map estimation, and one for the image segmentation. Filters parameters for the two sets were specified manually and their values are shown in Table 7.5.

The filters used for the segmentation are stronger than the ones used for the disparity map estimation because we want to simplify the image for the segmentation, whereas for the disparity map we want to remove noise but still retain some details for the matching process.

Image set	s_m	σ_r	σ_s
Disparity estimation	3	9	3
Segmentation	5	17	5

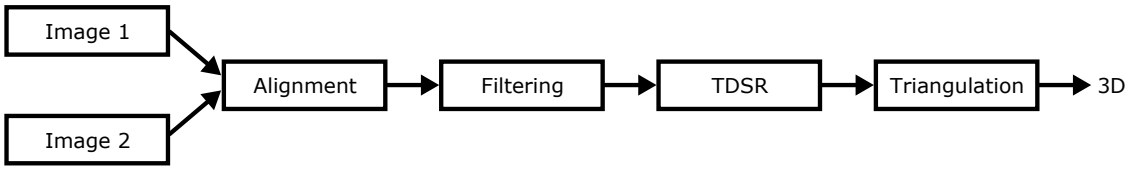
Table 7.5: Filters' parameters for the two sets of images. *s_m: size of the median filter. σ_r and σ_s are both parameters of the bilateral filter.*

Images are then rotated 90° and the disparity map is then estimated using the TDSR algorithm as described in Chapter 6.

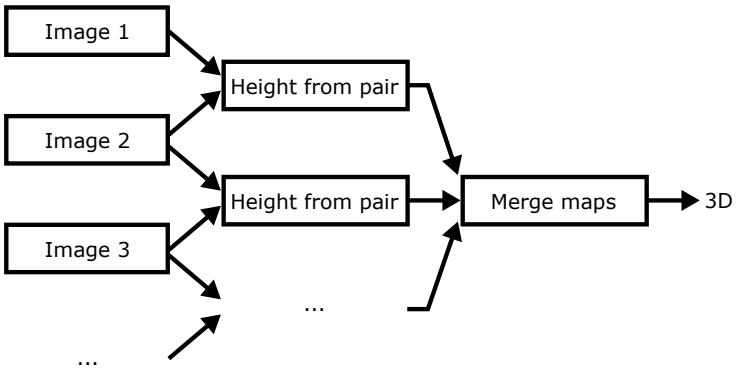
The height map can then be retrieved using triangulation (discussed in Section 4.1).

The initial disparity map used as input by the TDSR algorithm can be refined if more than two views are available. If we have a reference image r acquired with a tilt of 0° and n secondary images $(s_1, s_2, ..., s_n)$ acquired with different tilts, then we estimate n initial disparity maps from the pairs $((r, s_1), (r, s_2), ..., (r, s_n))$. We then merge them with a similar process than the one developed for the IARPA challenge described in Section 7.1. We remove low confidence values ($< 50\%$), and we use the resulting disparity map as input for the TDSR algorithm.

The overall processes for estimating the height map from a pair of images and from several images are shown in Figure 7.13.



(a) *Height map estimation process for a pair of images.*



(b) *Height map estimation process for several (> 2) images.*

Figure 7.13: Height map estimation process.

7.3.2 Quantitative results

As our algorithm is going to be used to precisely measure the topography of catalysts in order to infer mechanical and chemical properties, we should quantify its accuracy. In Section 4.6, we have listed different methods that allow to quantify the accuracy of a stereo algorithm. We already used the Middlebury database for comparing TDSR to the state of the art stereo methods. However, our SEM images contain unique properties - noise, illumination and shapes are different than in standard stereo databases - and therefore we need to create a specific database to compare our results to.

In order to generate this database, we need, for each sample, a height map ground truth in addition of its images. The problem is that there is no alternative to obtain a precise topography of catalysts. It is possible to use calibration samples with a known topography. However, their shapes are too different from the catalysts we must estimate; as for the Middlebury database, the quantitative analysis wouldn't be representative. Manually estimating the topography of our samples would be too much time consuming: the database we would be able to produce would contain too few items for the analysis to be representative.

We therefore decided to rely on simulated images. A **strong simulation**, simulating the electrons paths according to the laws of physics, requires too much computing power for complex shapes similar to our catalysts. We rely therefore on **weak simulation**, that creates images that look sufficiently similar to SEM images, though the laws of physics are not all taken into account. This allows us to simulate enough samples with realistic shapes to get a statistically relevant analysis.

In Section 7.3.2, we will first describe the process we created to obtain realistic enough SEM images. We will then explain how we generated the benchmark database in Section 7.3.2. Finally, in Section 7.3.2 we will quantify the quality of the height maps obtained by our TDSR algorithm and compare it with state of the art methods.

SEM image simulation

In order to generate realistic enough SEM images, we need to address multiple issues.

First, we have to generate 3D structures that are similar to our catalysts. Luckily, similar shapes can be easily modeled using a 3D computer graphics software. For this purpose, we used the open source software Blender [108].

In order to make the image more realistic, we also retrieved textures from original SEM images and applied them on the sample's support. The catalyst and the support are then bundled in a single scene in the Unity 3D software [96] (see Figure 7.14) and a screenshot can be taken. An example of screenshot is shown in Figure 7.15.

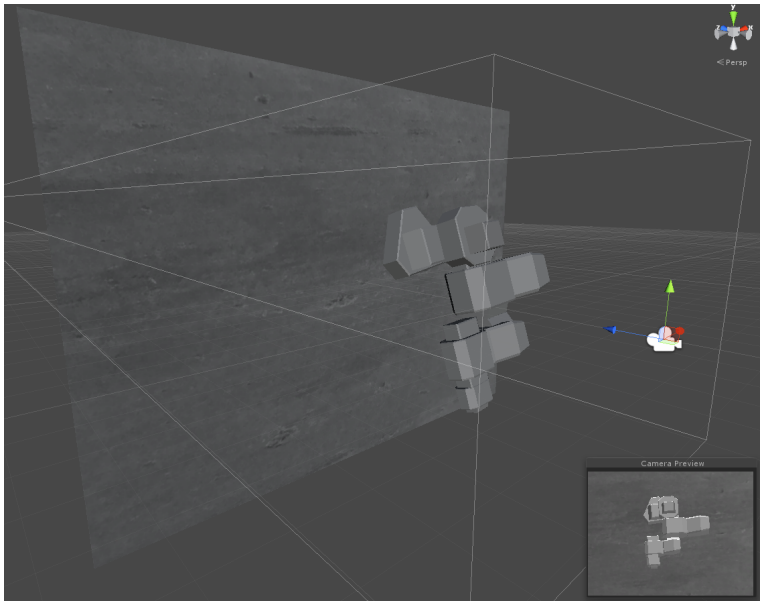


Figure 7.14: Simulation's scene in Unity 3D.

Unfortunately, this simulation is not realistic enough because the reflectance of all the objects in the image follow a simple Lambertian model. We have seen in Chapter 2 that the gray levels in the images follow several known rules that greatly differ from this model. We created therefore a new reflectance model that takes the three following rules into account:

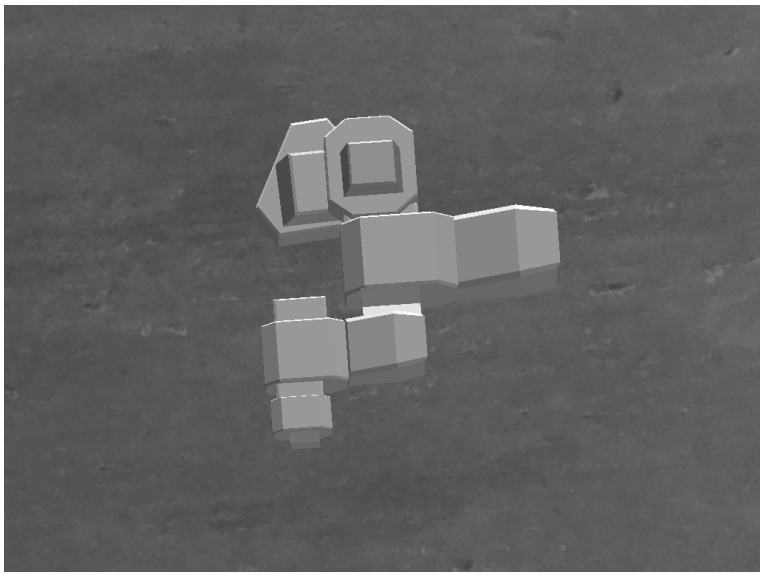


Figure 7.15: Example of simulation screenshot.

- For a pixel, if the angle between the surface and the simulated electron beam is θ, its gray level mainly depend on $1/\cos(\theta)$ for $|\theta| < 80°$ ([1] p. 156).
- A shadow effect appears near an object that occludes another one.
- Spikes appear lighter and dips appear darker.

For simulating these various effects, we need 3 maps:

- A **texture map**: for each pixel, we retrieve the gray value of the nearest point to the camera without any lighting effect.
- A **height map**: for each pixel, we retrieve the height of the nearest point to the camera.
- A **normal map**: for each pixel, we retrieve the normal vector of the nearest point to the camera.

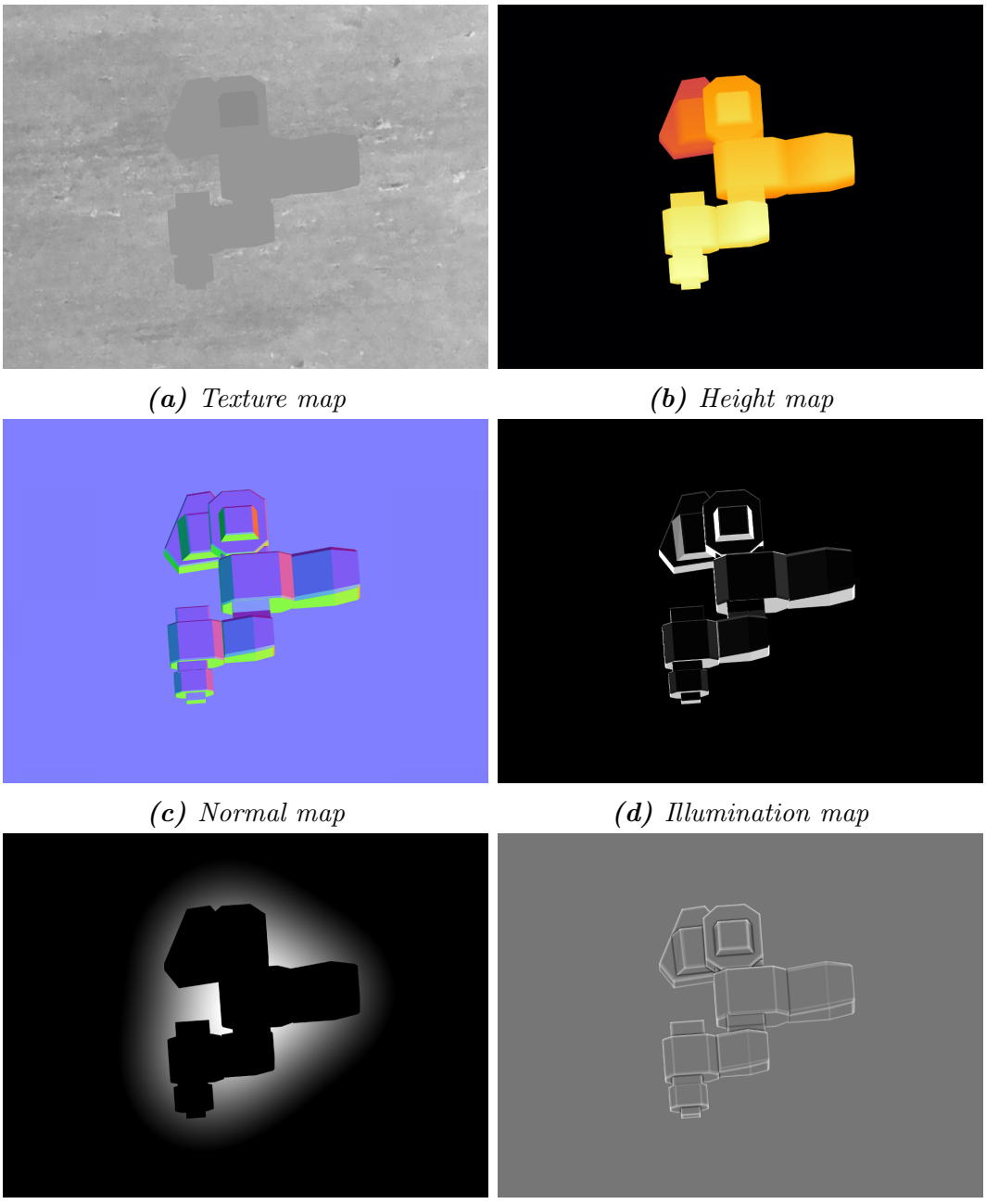
These three maps are shown in Figure 7.16.

We then generate three maps: the **illumination map**, the **shadow map**, and the **SD map** (Spike and Dips).

The **illumination map** estimates the illumination effect due to the angle between the surface and the electron beam. For doing that, we suppose that the electron beam is vertical; its direction vector is $d_{beam} = [0, 0, -1]$. For each point, we compute the angle from the dot product between the surface's normal n_s and d_{beam}:

$$\theta = \arccos(n_s \dot{d}_{beam})$$

We then get an illumination value from the following function:



(a) *Texture map* ***(b)*** *Height map*

(c) *Normal map* ***(d)*** *Illumination map*

(e) *Shadow map (inverted)* ***(f)*** *SD Map*

Figure 7.16: Maps generated for producing the new reflectance model. *For the normal map, the color red represents the x axis, green represents the y axis, and blue represents the z axis.*

$$f_{illumination}(\theta) = \begin{cases} \frac{1}{\cos(\theta)}, \text{for } |\theta| < 80° \\ a\theta + b, \text{otherwise} \end{cases}$$

Where $a = \frac{\partial \frac{1}{\cos(\theta)}}{\partial \theta}(\theta^*)$, $b = f(\theta^*) - a\theta^*$, and $\theta^* = 80°$. The function is plotted in Figure 7.17. The resulting **illumination map** is shown in Figure 7.16d.

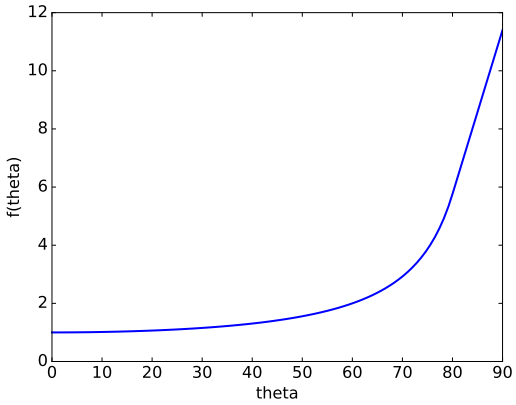


Figure 7.17: Plot of $f_{illumination}(\theta)$

The **shadow map** estimates the shadow effect due to higher neighboring objects. It is simulated by the following formula :

$$m_{shadow}(x) = \min(0, h(x) - G(h(x)) + k)$$

h being the height map, G being a Gaussian blur filter (with a kernel size of 301px), k being a constant (set to 50). The resulting **shadow map** is shown in Figure 7.16e.

The **SD map** estimates the lighting and shadow effect caused by spikes and dips. The main idea is to model for each pixel a sphere centered at the surface and to measure the percentage of volume covered by the sample. See Figure 7.18. If this percentage is lower than 50%, then the pixel is probably localized on a spike, and if it is higher than 50%, then it should be localized on a dip.

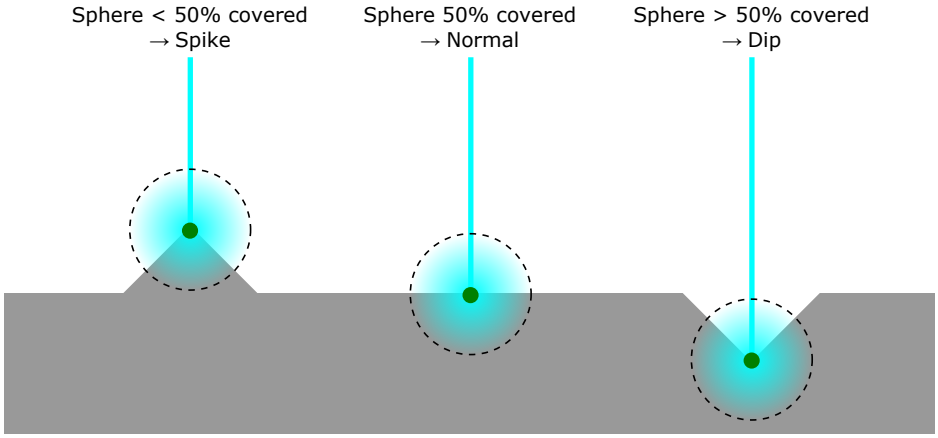


Figure 7.18: How spikes and dips are found.

The following formula is used to implement this idea:

$$m_{SD}(x,y) = \frac{\sum_{(x',y') \in \mathcal{C}(x,y)} v_{covered}(x,y,x',y')}{\sum_{(x',y') \in \mathcal{C}(x,y)} v(x,y,x',y')}$$

Where $\mathcal{C}(x, y)$ is the set of points in the circle of size r centered on (x, y). $v(x, y, x', y')$ and $v_{covered}(x, y, x', y')$ are defined as follow:

$$v(x, y, x', y') = 2v_{sphere}(x, y, x', y') \times r(x, y, x', y')$$

$$v_{covered}(x, y, x', y') = bounded(h'_{(x,y)}(x', y') - h(x', y') + v_{sphere}(x, y, x', y')) \times r(x, y, x', y')$$

With:

$$bounded(v) = \max(0, \min(2v_{sphere}(x, y, x', y'), v))$$

$$v_{sphere}(x, y, x', y') = \sqrt{r^2 - (x' - x)^2 - (y' - y)^2}$$

$$r(x, y, x', y') = \max(0, \min(1, 1 - \frac{h(x', y') - h(x, y) - \Delta_{max}}{\Delta_{max}}))$$

$h(x, y)$ being the height map, $h'_{(x,y)}(x', y')$ being the interpolated height in (x', y') from (x, y) assuming the normal in (x, y) remains constant, Δ_{max} being a constant ($\Delta_{max} = 60$). r has been added to prevent an additional shadowing effect where great changes of height occur, as it is already taken into account by the **shadow map**.

The new simulating image is then obtained by a normalized linear combinations of the different maps:

$$f_{simulated} = normalize(m_{texture} + a \times m_{illumination} + b \times m_{shadow} + c \times m_S + d \times m_D)$$

$m_{texture}$ being the texture map, $m_{illumination}$ being the illumination map, m_{shadow} being the shadow map, $m_S(x, y) = \max(0, m_{SD}(x, y))$ and $m_D(x, y) = \min(0, m_{SD}(x, y))$. The parameters a, b, c and d were determined manually. A Poisson noise is also added to the resulting image to match the noise of SEM images. An example of simulated image is shown in Figure 7.19.

Database creation

The generated database is composed of multiple **scenes**. Each scene represent the combination of a sample and a support.

For each scene, the height map is saved as well as a **sequence** of images. The sequence represents the same scene rotated at different tilts. The most prevalent sequence is: -10°, -5°, 0°, +5°, +10°. See Figure 7.20.

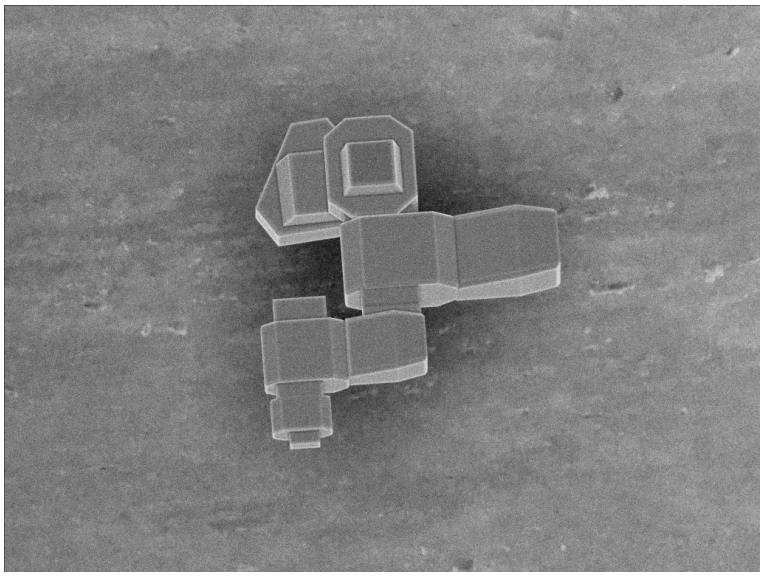
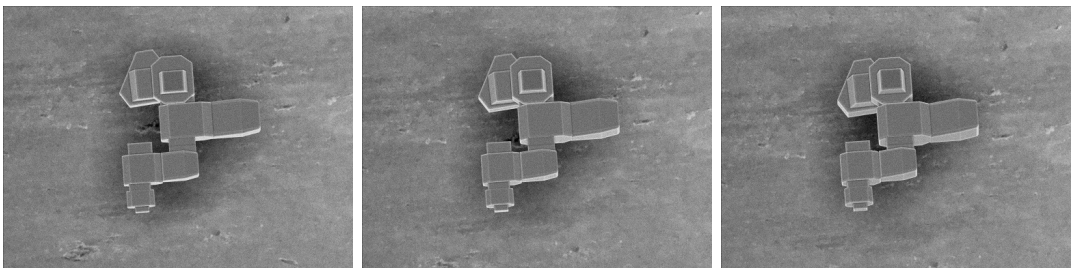
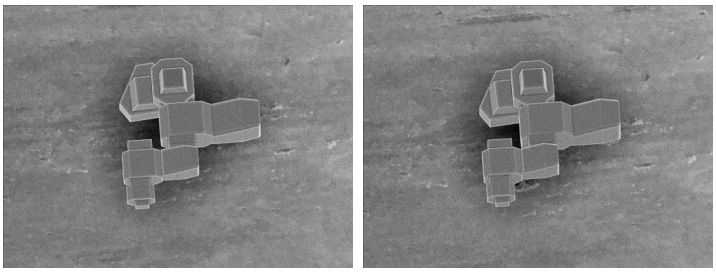


Figure 7.19: New simulated image.



(a) *-10°* ***(b)*** *-5°* ***(c)*** *0°*

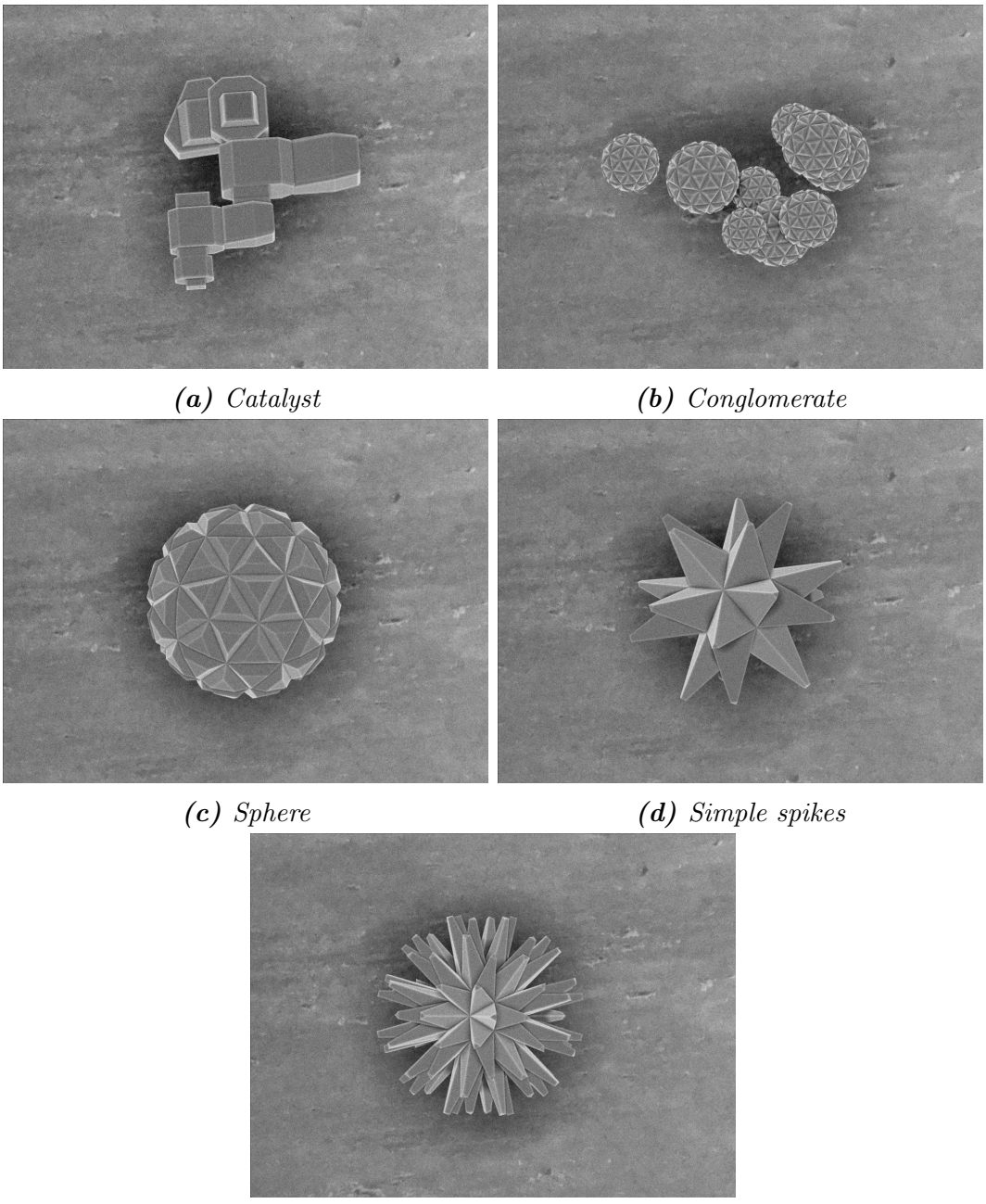


(d) *+5°* ***(e)*** *+10°*

Figure 7.20: Sequence of image.

Five samples have been created for the database. They are named *Catalyst*, *Conglomerate*, *Sphere*, *Simple spikes*, and *Complex spikes*. An image is shown for each of them in Figure 7.21.

We divided our database into 7 categories, described in Table 7.6. Each category has a specific combination of sample and sequence acquisition tilts. As the shape of *Catalyst* was inspired from our samples, we also test on it different sequence tilts intervals in order to find the most adapted one.



(a) *Catalyst* ***(b)*** *Conglomerate*

(c) *Sphere* ***(d)*** *Simple spikes*

(e) *Complex spikes*

Figure 7.21: Samples created for the database.

For generating the database, we handle each category separately. For each category, we create a scene with the associated sample. We then acquire the height on each pixel to generate the height map ground truth. Finally, we acquire several images on the different sequence tilts. For each image, the intensity of the shadows, spikes and dips are randomly and slightly changed to simulate the variation of acquisition properties.

We then randomly rotate the sample, and repeat the previous process several time. See Figure 7.22. These random rotations allows us to generate multiple scenes using each sample (this is therefore a kind of data augmentation). We repeat the process more for complex shapes than for simple or repetitive shapes.

Name	Sample	Sequence tilts	Nb.
Catalyst	Catalyst	-10°, -5°, 0°, +5°, +10°	46
Catalyst narrow	Catalyst	-5°, -2.5°, 0°, +2.5°, +5°	50
Catalyst narrower	Catalyst	-2.5°, -1.25°, 0°, +1.25°, +2.5°	52
Conglomerate	Conglomerate	-10°, -5°, 0°, +5°, +10°	30
Sphere	Sphere	-10°, -5°, 0°, +5°, +10°	15
Simple spikes	Simple spikes	-10°, -5°, 0°, +5°, +10°	16
Complex spikes	Complex spikes	-10°, -5°, 0°, +5°, +10°	17
Total			**226**

Table 7.6: Categories in the database.

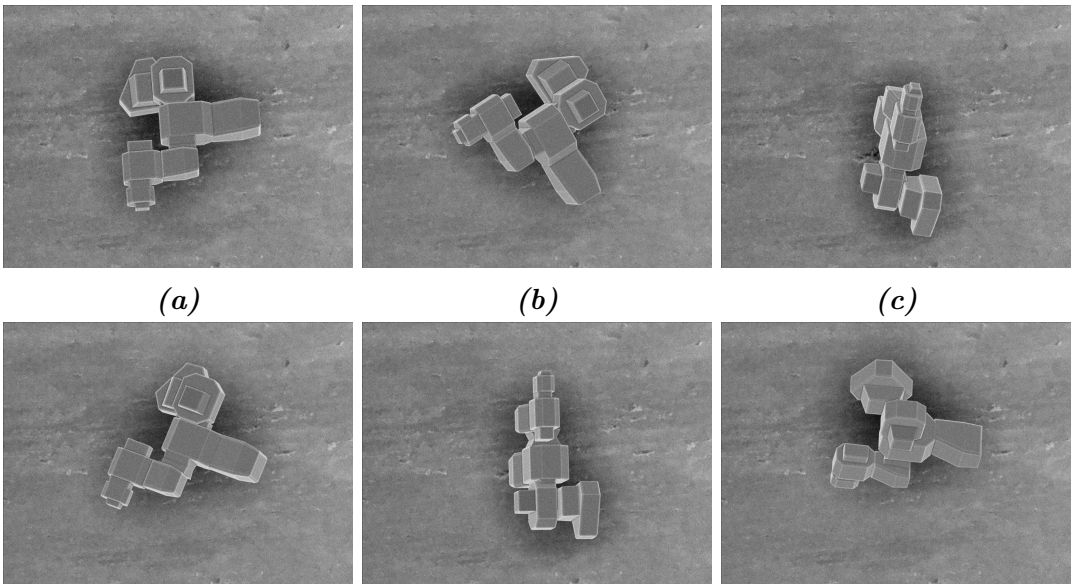


Figure 7.22: Multiple examples of scenes.

Results

Our pipeline then processes each sequence of the database. We used here the full model map post-processing pipeline (model propagation, model map simplification and guided correction with the FGS method).

The produced height maps can then be compared to their associated ground truth. Both height maps' unit is the voxel, a cube of 1px large in the image. As the estimated heights are relative values compared to a chosen origin point, we align the height map to the ground truth using the following formula :

$$h_{aligned} = h - median(h) + median(h_{gt})$$

h being the height map and h_{gt} being the ground truth height map.

For each pixel, the error term $e(x, y) = |h_{aligned}(x, y) - h_{gt}(x, y)|$ can then be computed. Tables 7.7 and 7.8 and Figure 7.23 show various statistics on this error term, for our method as well as state of the art diffusion and interpolation methods.

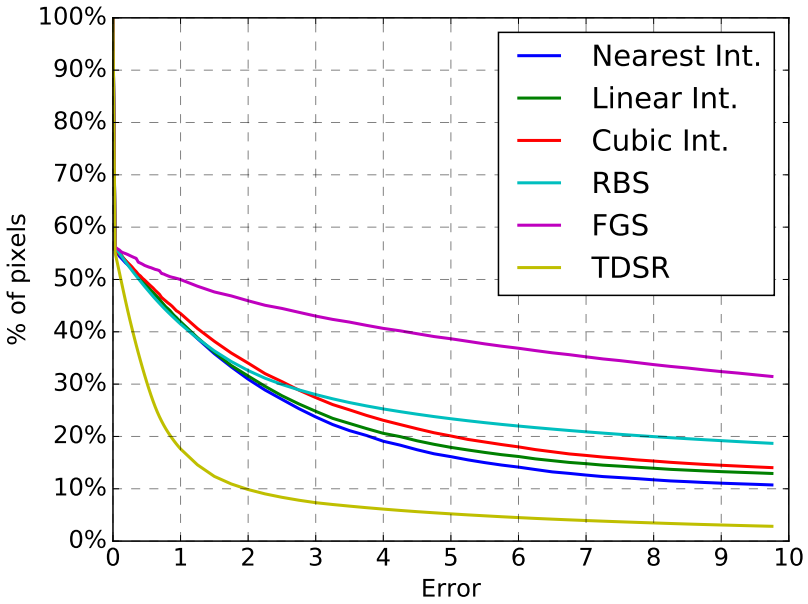


Figure 7.23: Distribution of the error for the Catalyst category. *Percentage of pixel with an error greater than the x axis. A perfect curve would follow the y and then the x axis. How to read: for the TDSR method, 10% of the points have an error larger than 2 voxels.*

Interpolation methods produce height maps with a higher average error than diffusion methods, but contain fewer points with a large error. It is consistent with our analysis on the Middlebury database discussed in Section 7.2, where interpolation methods produced disparity maps with a higher average error but lower "Bad 2.0".

Whether we look at the average error, the proportion of large errors, or the distribution of the errors, the quantitative results show that our solution is much more adapted to the database than interpolation and diffusion methods, no matter the sample used.

Category	Method	Avg. Error	% of large errors (> 10)
Catalyst	Nearest Int.	34.64 ± 10.63	$10.6 \pm 2.4\%$
	Linear Int.	33.86 ± 10.6	$12.8 \pm 2.5\%$
	Cubic Int.	34.42 ± 10.7	$13.9 \pm 2.6\%$
	RBS	30.36 ± 9.33	$18.5 \pm 5.5\%$
	FGS	26.5 ± 16.93	$31.2 \pm 6.7\%$
	TDSR	2.52 ± 1.97	$2.8 \pm 1.5\%$
Catalyst narrow	Nearest Int.	21.48 ± 3.37	$11.6 \pm 1.6\%$
	Linear Int.	20.92 ± 3.34	$12.5 \pm 1.7\%$
	Cubic Int.	21.5 ± 3.4	$14.7 \pm 1.9\%$
	RBS	19.37 ± 2.92	$11.9 \pm 2.0\%$
	FGS	15.52 ± 2.98	$27.1 \pm 4.6\%$
	TDSR	2.35 ± 0.8	$3.1 \pm 1.1\%$
Catalyst narrower	Nearest Int.	21.25 ± 2.79	$23.6 \pm 1.4\%$
	Linear Int.	20.71 ± 2.78	$23.4 \pm 1.6\%$
	Cubic Int.	21.72 ± 2.83	$25.6 \pm 1.5\%$
	RBS	18.83 ± 2.5	$22.7 \pm 1.8\%$
	FGS	13.63 ± 2.64	$24.0 \pm 4.3\%$
	TDSR	3.29 ± 1.3	$4.1 \pm 2.7\%$
Conglomerate	Nearest Int.	33.88 ± 9.2	$7.5 \pm 1.5\%$
	Linear Int.	33.34 ± 9.08	$11.3 \pm 1.5\%$
	Cubic Int.	33.76 ± 9.18	$11.7 \pm 1.5\%$
	RBS	29.38 ± 6.62	$15.7 \pm 3.7\%$
	FGS	17.31 ± 4.72	$24.9 \pm 4.6\%$
	TDSR	3.1 ± 1.56	$1.6 \pm 0.4\%$
Sphere	Nearest Int.	34.65 ± 4.47	$10.9 \pm 1.0\%$
	Linear Int.	33.03 ± 4.46	$13.9 \pm 0.8\%$
	Cubic Int.	33.71 ± 4.52	$14.2 \pm 0.8\%$
	RBS	28.21 ± 0.91	$18.5 \pm 1.1\%$
	FGS	14.13 ± 2.64	$31.3 \pm 3.2\%$
	TDSR	1.96 ± 0.38	$5.0 \pm 1.1\%$
Simple spikes	Nearest Int.	43.49 ± 2.31	$14.0 \pm 0.9\%$
	Linear Int.	41.9 ± 2.33	$17.3 \pm 0.7\%$
	Cubic Int.	42.5 ± 2.37	$17.9 \pm 0.8\%$
	RBS	37.65 ± 1.66	$23.5 \pm 2.2\%$
	FGS	27.93 ± 3.62	$34.5 \pm 2.2\%$
	TDSR	4.71 ± 2.02	$4.1 \pm 0.6\%$
Complex spikes	Nearest Int.	53.59 ± 4.51	$16.7 \pm 0.8\%$
	Linear Int.	52.62 ± 4.62	$20.8 \pm 0.7\%$
	Cubic Int.	53.1 ± 4.68	$21.0 \pm 0.7\%$
	RBS	44.98 ± 1.89	$27.6 \pm 0.8\%$
	FGS	23.7 ± 3.02	$35.5 \pm 2.3\%$
	TDSR	5.13 ± 0.82	$7.2 \pm 1.7\%$

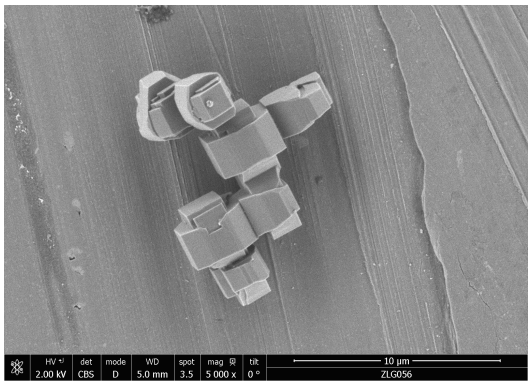
Table 7.7: Average error and percentage of large error (more than 10 voxels).

Method	10th	25th	50th	75th	90th
Nearest Int.	0.0 ± 0.0	0.0 ± 0.0	0.42 ± 0.14	2.88 ± 0.59	31.75 ± 80.72
Linear Int.	0.0 ± 0.0	0.0 ± 0.0	0.44 ± 0.16	3.08 ± 0.74	64.23 ± 68.25
Cubic Int.	0.0 ± 0.0	0.0 ± 0.0	0.48 ± 0.15	3.64 ± 0.88	58.06 ± 65.02
RBS	0.0 ± 0.0	0.0 ± 0.0	0.43 ± 0.19	5.51 ± 3.73	76.94 ± 50.45
FGS	0.0 ± 0.0	0.0 ± 0.0	1.51 ± 1.38	18.63 ± 11.4	89.95 ± 90.36
TDSR	0.0 ± 0.0	0.0 ± 0.0	0.17 ± 0.11	0.76 ± 0.46	1.8 ± 1.55

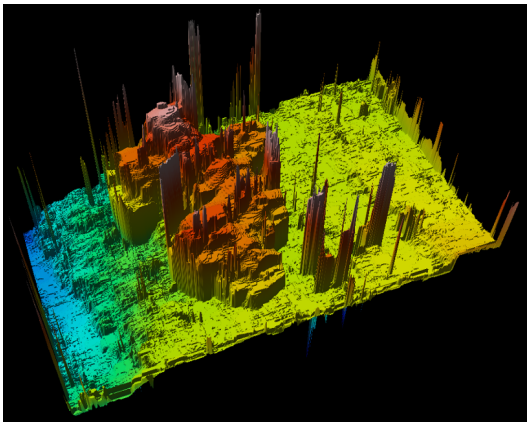
Table 7.8: Errors' percentiles for the Catalyst category. *How to read: for the TDSR method, 75% of the points have an error lower than 1.13 voxel.*

7.3.3 Qualitative results

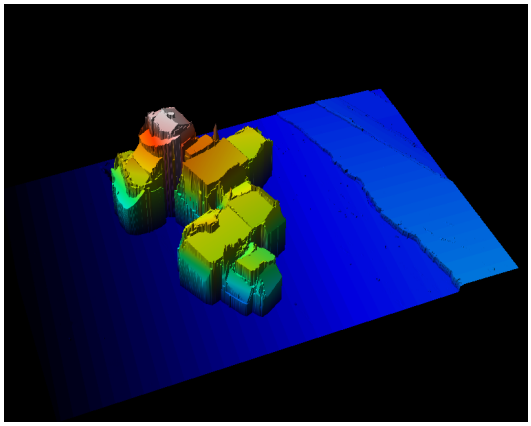
Results on SEM images are shown and described in Chapter 6. Some additional 3D visualization are also shown in figures 7.24 to 7.30. The 3D visualizations confirm our algorithm's advantages on catalysts' images; qualitative inspections of the resulting height maps show that they are less noisy and that the overall shapes of the samples seem to be much more respected. However, due to lack of texture, the TDSR algorithm can sometimes oversimplify the samples' shapes as shown in figure 7.28.



(a) *Reference image*



(b) *Block matching (block size: 31px)*



(c) *TDSR*

Figure 7.24: Height map estimation on a pair of image using the block matching and the TDSR algorithms.

(a) Reference image

(b) Obtained 3D map

(c) (b) + overlaid by reference image

Figure 7.25: Stereo reconstruction example using TDSR (from simulated images). (1 / 6)

(a) *Reference image*

(b) *Obtained 3D map*

(c) *(b) + overlaid by reference image*

Figure 7.26: Stereo reconstruction example using TDSR. (2 / 6)

(a) Reference image

(c) (b) + overlaid by reference image

(b) Obtained 3D map

Figure 7.27: Stereo reconstruction example using TDSR. (3 / 6)

(a) *Reference image*

(b) *Obtained 3D map*

(c) *(b) + overlaid by reference image*

Figure 7.28: Stereo reconstruction example using TDSR. (4 / 6)

(a) Reference image

(b) Obtained 3D map

(c) (b) + overlaid by reference image

Figure 7.29: Stereo reconstruction example using TDSR. (5 / 6)

(a) Reference image

(b) Obtained 3D map

(c) (b) + overlaid by reference image

Figure 7.30: Stereo reconstruction example using TDSR. (6 / 6)

7.4 Conclusion

We have presented in this chapter various applications of stereo reconstruction algorithms: urban topography reconstruction from satellite images, indoor scenes reconstruction, and samples surface reconstruction from SEM images. As each application had its own set of challenges, we developed adapted solution for each case, and achieved results near and sometimes better than state of the art methods.

Qualitative and quantitative results both suggest that the TDSR algorithm, combined with the multi-scale disparity map and multi-view stereo approaches developed in this thesis, is much more adapted to the analysis of catalysts than state of the art methods.

However, the developed approach is versatile and can be used with various inputs and in various applications. For instance, in the Middlebury database, the TDSR algorithm combined with results of the MC-CNN stereo reconstruction algorithms allows us to achieve results that are near and sometimes better than state of the art methods.

Chapter 8

Conclusion and perspectives

French summary / Résumé en français

Ce chapitre conclut sur les méthodes proposées et leur pertinence pour la reconstruction 3D de catalyseurs à partir d'images de Microscopes Electroniques à Balayage. Nous discutons également des extensions potentielles des algorithmes développés ainsi que de quelques perspectives.

The aim of this work was to estimate the 3D topography of microscopic samples from images acquired using Scanning Electron Microscopes. For achieving this goal, multiple techniques already exist.

A first method, photoclinometry, consists in acquiring multiple images of the sample on different lightning conditions, and using the illumination information in each pixel to estimate its normal: the topography is then estimated by integrating the normal of all pixels. Though extensively used in the past, it performs significantly worse than the second method, stereo reconstruction.

In stereo reconstruction, multiple images of the sample at different orientations are acquired. By evaluating the displacement of a pixel between the different images, its height can be estimated. An height map can then be constructed by evaluating the displacement of all pixels in the images.

Though standard stereo methods perform very well on textured samples, they fail to evaluate adequate 3D reconstructions on low textured areas because the displacement of the pixels inside those areas are difficult to estimate. The samples whose topography we aim to evaluate, catalysts, contain a lot of low textured areas, so we needed to adapt stereo methods to our use case.

We proposed two solutions in this work.

The first method, Top-Down Segmented Matching (TDSM), used hierarchical segmentation methods to divide images of the samples into regions and subregions. Instead of estimating the displacement of pixels, as standard stereo methods do, TDSM estimates the displacement of regions. If the estimation of a region is deemed unsatisfactory, the region is divided into subregions and the displacement of each subregion is then estimated following the same process.

This top-down approach allows a better processing of low textured areas as they are very likely to be processed as a single region. However, it suffers from the fronto-parallel hypothesis. Indeed, by matching regions as a whole, we make the hypothesis that they are parallel to the camera plane, so oblique surface are therefore not well processed.

This limitation led us to create a second method, Top-Down Segmented Regression (TDSR). Though we are still using hierarchical segmentation to process regions and subregions in a top-down manner, there is a paradigm shift in how the problem is perceived. Instead of estimating an height map, our algorithm refines and completes an existing height map obtained in a preliminary step. The reference image is first divided into main regions. For each region, we retrieve the values of the initial height map inside the region, and try to model them by a plane. If the model is satisfying, we stop here. If it is not, we divide the region into subregions and try to model each of them following the same process. This approach produces 3D reconstructions that are less noisy and more accurate than state of the art stereo methods on our catalysts samples.

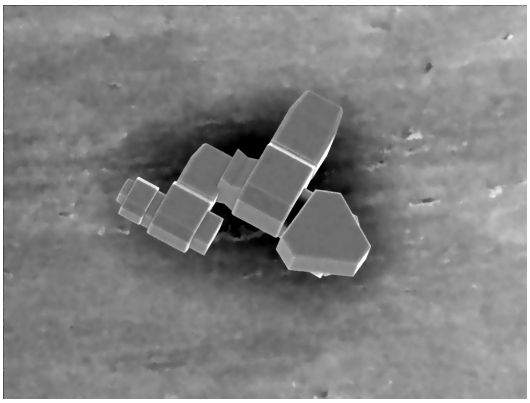
Key strengths of our approach and axes of improvements

As it will allow us to formulate areas of improvements, it is important to understand which aspects makes TDSR an efficient solution to stereo reconstruction problems.

The first strength is that our approach doesn't try to process pixels independently, but to aggregate their estimations into regions observed in the reference image. For choosing these regions, the algorithm promotes, through the use of the hierarchical segmentation, large and most obvious regions over small and less obvious ones. This leads to robust results as most regions contain enough points to provide an accurate model.

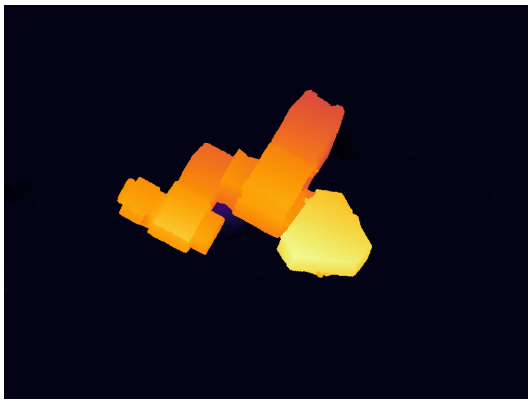
Hence, a first axis of improvement could be to improve final results by choosing a more adequate hierarchical segmentation. This problem involves multiple steps - image filtering, gradient estimation, segmentation, hierarchical segmentation - each of them being extensively researched to this day. In the implementation of TDSR, we provide the opportunity to choose a custom hierarchical segmentation algorithm so that new and potentially better methods can be used in the future.

The second strength lies in the model map produced by our algorithm. This model-based representation provides much more information than only the height maps produced by classical stereo methods. As shown in Figure 8.1, it can also produce a clean and accurate segmentation of the reference image, or it can provide the orientation of each pixel from the models parameters.

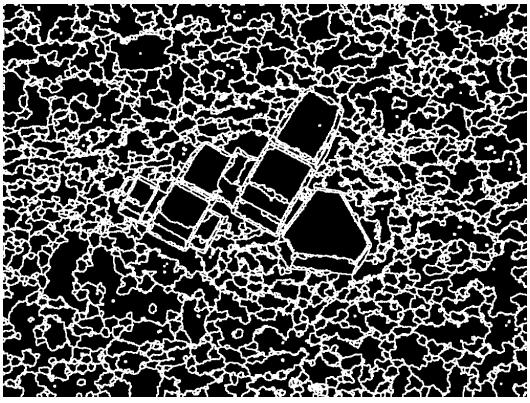


(a) Reference image

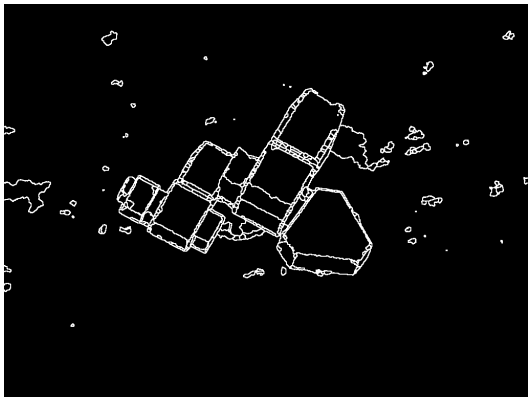
(b) Ground truth



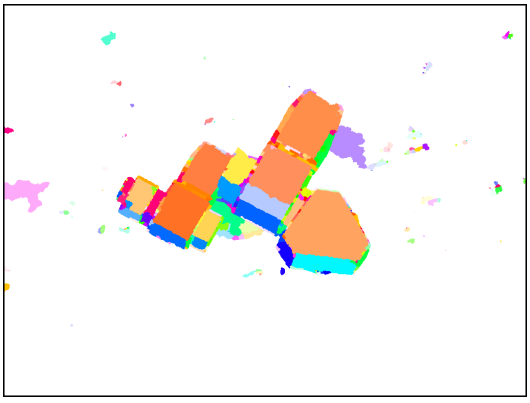
(c) Height map produced by TDSR



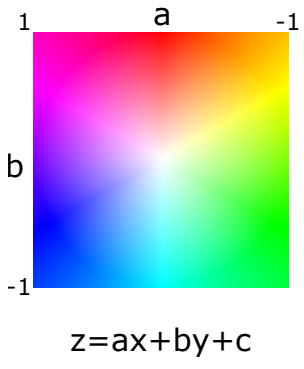
(d) Initial segmentation



(e) Retained segmentation by TDSR



(f) Orientation map



(g) Orientation map palette

Figure 8.1: Retained segmentation and orientation map produced by TDSR.

Moreover, this representation allows us to further refine the height map through the use of the model map post-processing operators. For instance, the model propagation operator discussed in Section 6.3.5 allowed us to complete undefined areas in the height map.

Therefore, the development of new model map operators constitutes the second axis of improvement. A lead that could be further explored is the combination of the results of our TDSR method with those of alternative stereo of diffusion techniques.

The third strength is the versatility of the TDSR method. Though the quality of the final height map is dependent on the quality of the initial height map, the stereo method estimating the latter is viewed as a separated process. The consequence is that TDSR can be used to refine the output of any stereo method. Therefore, though this thesis' purpose is very specific, the proposed approach is general.

A third axis of improvement could therefore lie in the further generalization and extendibility of the TDSR method. Though some work has already been done to this end, several parts could be improved. For instance, instead of only using linear models, the method could choose the most adapted model from a predefined list that could contain linear models, quadratic models, and splines for instance. Some work is needed to determine which model is the most adapted to the analyzed region, and how model map post-processing methods can propagate these more complex models (interpolation of quadratic models or splines outside their defined areas can lead to extreme values).

Practical application

This thesis also had a practical side as one of the objective was to provide a software that IFPEN could use to automatically estimate the topography of catalysts. A first version of the software was provided to the Physics and Analysis division of IFPEN on June 2016, one year before the end of the thesis. Thus, we had the time to take into account the various feedbacks formulated since then and improve the software accordingly. IFPEN has now a finished product that can be easily used by its chemists and physicists. The software is described in Appendix B and a screenshot is shown in Figure 8.2.

Perspectives

The proposed TDSR method is general and can therefore be applied to other uses.

First, we have shown that it can be applied to other stereo reconstruction problems. On the stereo pairs of the Middlebury database, which contains images of in-door scenes, we obtained results that were comparable and sometimes better than state of the art methods.

Secondly, as long as spatial data is combined with an image, our TDSR method can be easily adapted to refine it. RGB-D images and semantic segmentation are a few potential applications, as shown in Figure 8.3.

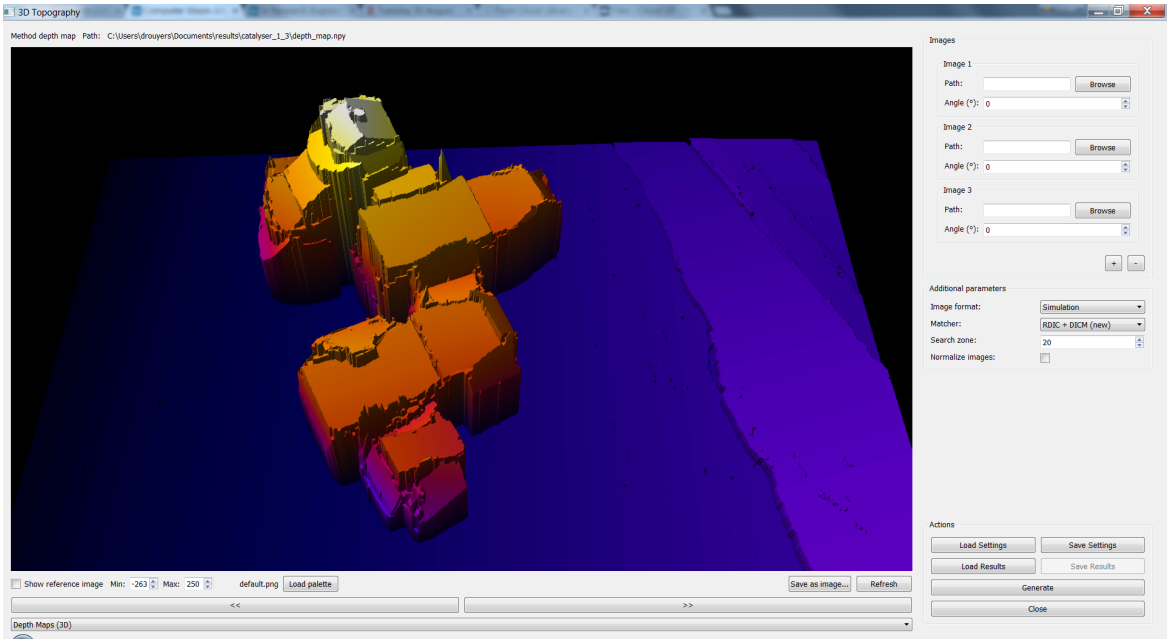
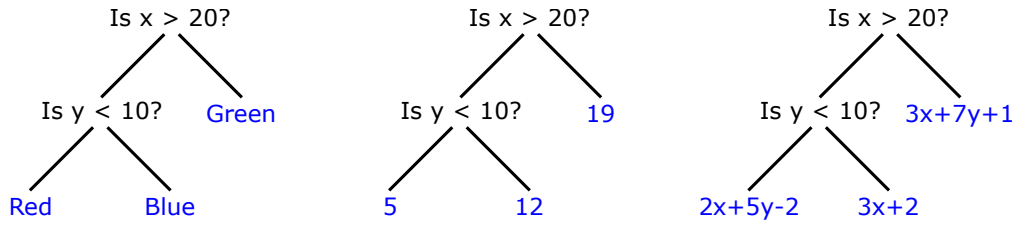


Figure 8.2: The GUI of the software provided to IFPEN.

(a) *Initial depth map* ***(b)*** *Reference image* ***(c)*** *(a) + TDSR*

Figure 8.3: Example of RGB-D image refinement using the TDSR algorithm. *The initial depth map and the reference image are from the RGB-D Object Dataset [109] and were acquired using a Kinect style 3D camera.*

Finally, the concept of TDSR could be further generalized to perform regressions on other types of data. The initial idea behind TDSR was drawn from decision tree learning. Decision trees are used in machine learning to create interpretable models to perform classifications or regressions. As TDSR, the algorithms generating these decision trees split the data in a top-down manner until each group is sufficiently consistent. The conditions used to separate the data are simple and based on a single attribute (for instance: "is $x > 20$?"), making the produced decision tree easily interpretable. Each leaf of the decision tree is generally assigned either a class (for classification trees) or a number (for regression trees) corresponding to the majority of the observed items in the leaf. Used less frequently, the M5 algorithm [110] allows to assign a model (such as a linear model) to each leaf: the resulting decision tree is also named **model tree**. These different concepts are illustrated in Figure 8.4.



(a) *Classification tree* ***(b)*** *Regression tree* ***(c)*** *Model tree*

Figure 8.4: Illustration of the different categories of decision trees.

In many ways, the end result of TDSR is similar to model trees. Developed concepts such as model propagation or model map simplification could be transposed to allow the pruning - or simplification - of model trees. A similar concept to TDSR could be also applied to generate model trees. Unlike M5, TDSR doesn't compute a hierarchy: the hierarchy is viewed as a parameter that can be customized. This can be regarded as an advantage, because it gives the choice to its user on how to construct it. For instance, the hierarchy could be obtained from state of the art methods producing a hierarchical clustering of the data.

Other perspectives also lie in the complementary tools developed during this work.

The Adaptative Subsampling algorithm presented in Section 6.5 for speeding up TDSR can also be used for calibration purpose as shown in Appendix A. More broadly, this tool can be used for selecting, from an initial list of points, a subset of spatially separated points.

Several applications could also arise from the SEM simulator developed in Section 7.3.2. First, the database generated for testing our TDSR method will be published online so that it can be used as a benchmark for new stereo methods. The relatively high number of samples also makes this database an acceptable candidate for machine learning or deep learning based stereo methods. Secondly, as these images are artificially generated, they could also constitute a benchmark for noise reduction methods applied to SEM images (as we are able to get the image with and without noise).

Appendix A

Calibration: projection model and distortions

We need to determine the projection model and potential distortions in the SEM images under the same conditions our samples will be acquired. For a chosen magnification, this process can be done once and separately of the samples acquisition. It returns a set of parameters, the camera's intrinsic parameters, that will be used when estimating the height maps of our samples.

We describe here the general process, and we will detail each step in the following sub-sections.

First, we select an appropriate sample for the calibration process with a known topography. We acquire two images of it: a reference image and a secondary image taken from a different tilt angle. Next, we detect key points - points that can be reliably matched - on the reference image and estimate their 2D displacement into the second image. We manually remove eventual mismatches. We then analyze these displacement vectors, and we find the hypothesis (projection model + distortion) that best explain them knowing the sample's topography.

A.1 Sample selection

For the calibration process, we need a sample that contains a lot of feature points, evenly distributed. We also need to know its topography. We chose to use here a silicium wafer covered with 10 nm diameter colloidal gold particles (Sigma Aldrich); the topography of the sample is flat, and the gold particles add feature points that can be easily matched. We acquire two images of the sample, a reference image, and a secondary image taken at a different tilt (10°). See the reference image we used for the calibration in Figure A.1.

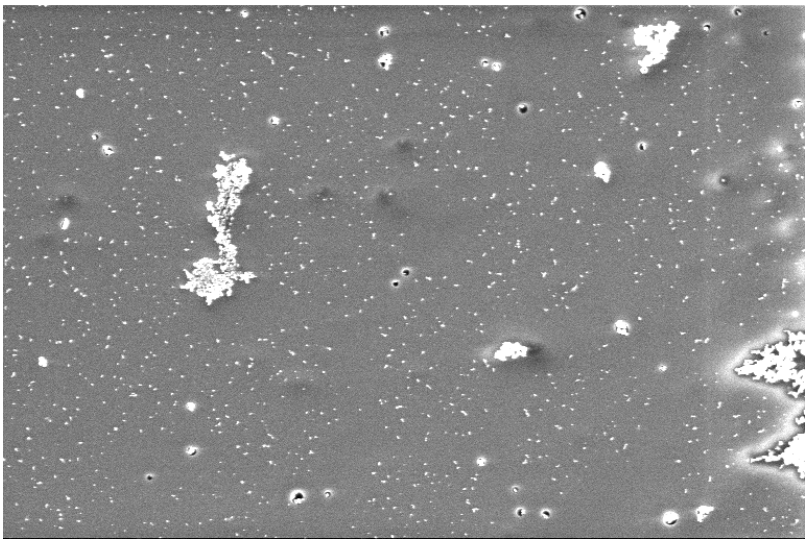


Figure A.1: Reference image of the calibration sample. *Magnification: 5000×.*

A.2 Key points detection

Once the sample has been acquired, we detect key points in the images. For achieving this, we first apply the Harris operator [58] on the reference image. The Harris operator produces a grayscale image where edges have a high gray values. See Figure A.2.

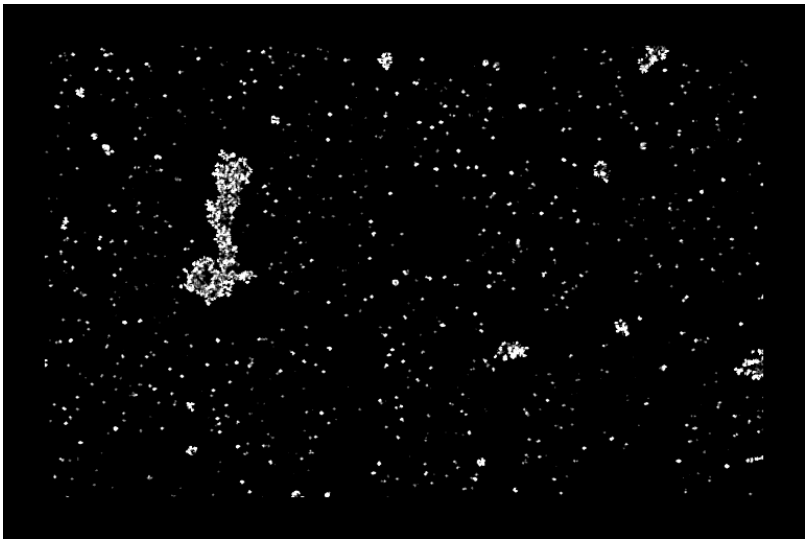
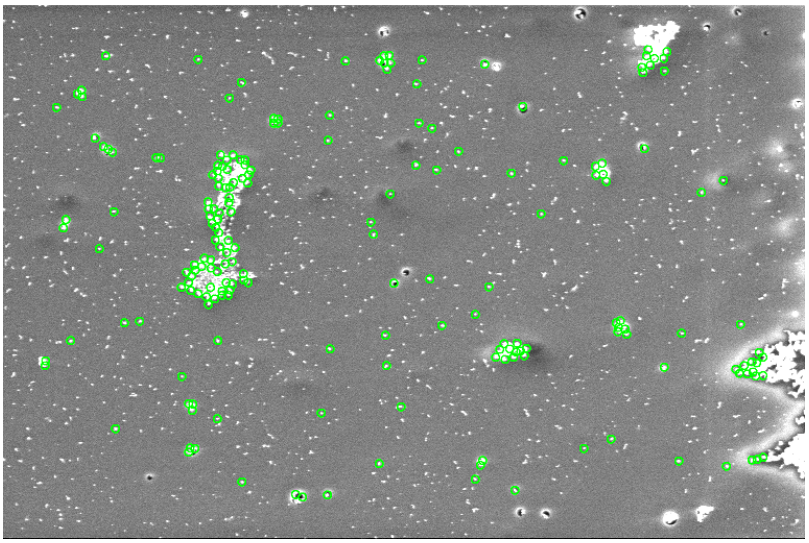
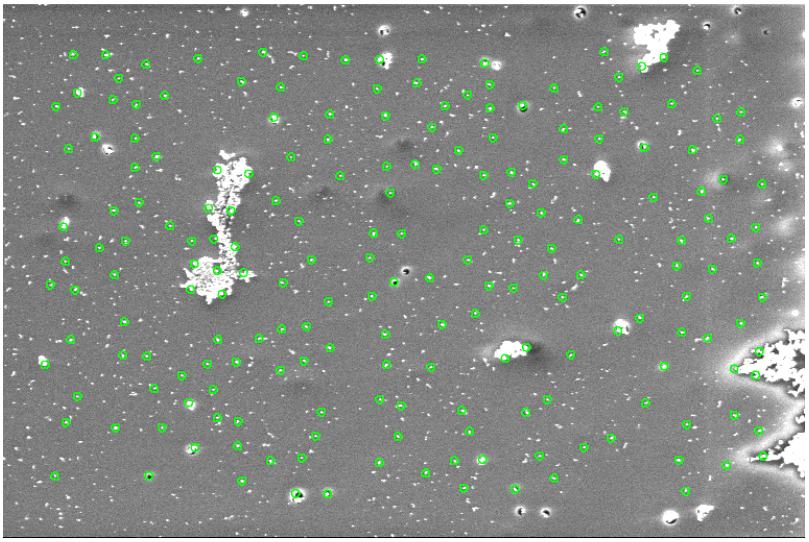


Figure A.2: Harris operator applied on the reference image.

On the resulting image, we then use the Adaptive Subsampling algorithm presented in Section 6.5 to retrieve the first N local maxima spatially separated. The chosen radius must be sufficiently large so that objects with a lot of strong edges don't dominate the chosen key points. We chose $N = 200$ and $size = 81px$ (radius). See Figure A.3.



(a) *The radius is too small: some objects collect a lot of key points.*



(b) *The chosen radius is correct.*

Figure A.3: Key points detected by the Adaptive Subsampling algorithm applied on the Harris transform of the reference image.

A.3 Key points matching

For the sake of accuracy, we need here a subpixel matching of the key points. For each key point (X, Y), we apply the following process.

First, we extract from the reference image the 9×9 block centered on (X, Y), called T. We also extract from the secondary image the 161×161 block centered on (X, Y), called E. We enlarge the blocks by a factor of 8 using bicubic interpolation. We then compute the following correlation score's matrix (robust to illumination changes):

$$s(x, y) = \sum_{x', y'} (T(x', y') \times E(x + x', y + y'))$$

The displacement vector v can then be deduced by finding the position of the global maximum in s.

Using a similar process than the LRC check described in Section 4.2.5, we then remove inconsistent checks. Out of the 200 key points, there are 175 consistent matches. We manually inspected the remaining matches, and removed 3 mismatches. See results in Figure A.4.

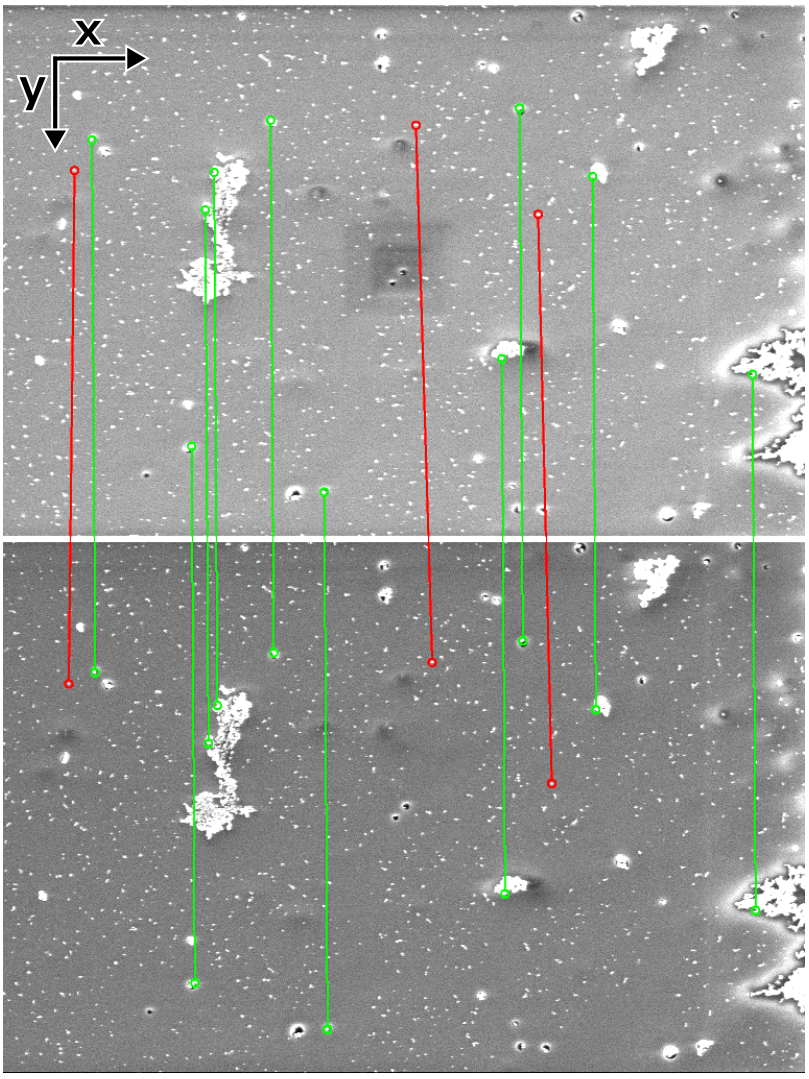


Figure A.4: Subset of detected matches. *Top: reference image. Bottom: secondary image. Green: correct match. Red: incorrect match manually detected by looking at outlier displacements vectors.*

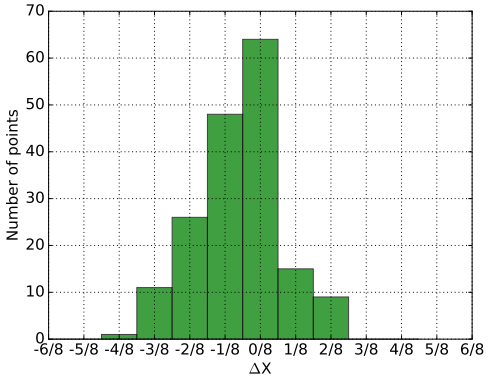
A.4 Image alignment

Due to the acquisition procedure, the reference image and the secondary image might not be horizontally aligned. Therefore, we displace the secondary image so that the median of the X component of the displacement vectors equals 0.

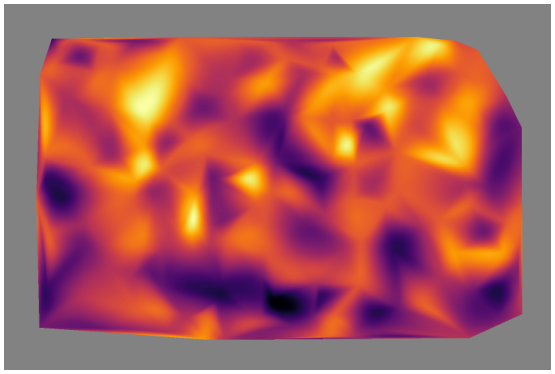
A.5 Distortion in the X axis

Now that the images are aligned, we can check if there are significant distortions in the X axis. For each valid key point, we measure ΔX, the X component of its displacement vector (after the images have been aligned).

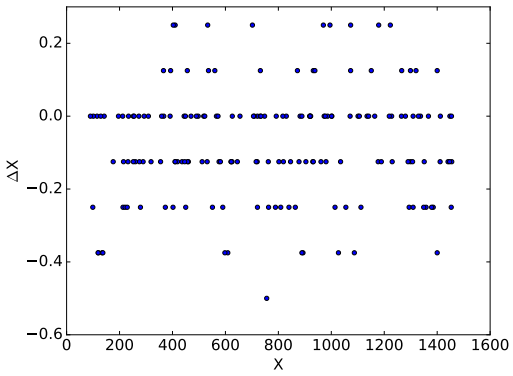
See Figure A.5. Overall, the ΔX values are quite low - most values lie in the $-\frac{1}{4}$px and $\frac{1}{4}$px range (Figure A.5a). If the positions of the key points are (X_i, Y_i), there is no clear dependence between ΔX and X_i (Figure A.5c), but there seems to be a weak correlation between ΔX and Y_i (Figure A.5d). However, as ΔX is very small, it won't significatively affect the matching process, as the smallest block we use when computing the disparity maps has a size of 9×9px.



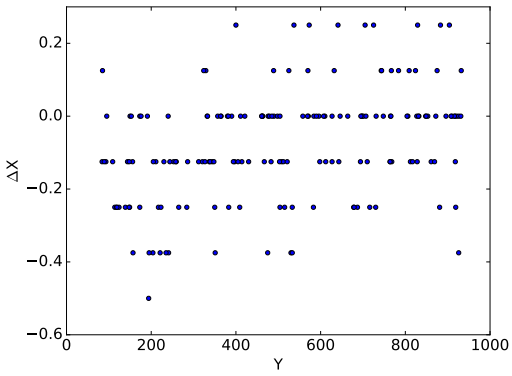
(a) Distribution of ΔX



(b) ΔX depending on the X and Y axes



(c) ΔX depending on the X axis



(d) ΔX depending on the Y axis

Figure A.5: Distribution of ΔX and relationship with X and Y.

A.6 Displacement vectors reevaluation

Now that the images have been aligned, we rotate both images 90°, and we can use the Block Matching algorithm discussed in Section 4.2.3. Though it is not necessary, it improves the accuracy of the evaluated displacement vector. First, the search is limited to one dimension (the X-axis), so the probability of a mismatch is greatly reduced. Secondly, the OpenCV [13] implementation allows us to obtain a disparity map with an accuracy of $\frac{1}{16}$ of pixel.

For computing the disparity map, the image is preprocessed using Normalization, we use 9×9px blocks, remove values with a uniqueness ratio lower than 10% and also those with a LRC check greater than 1.5px. See Figure A.6.

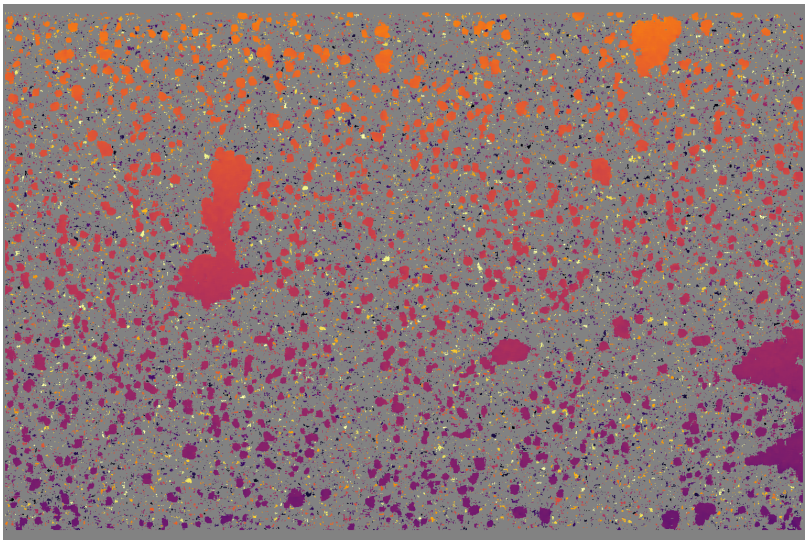


Figure A.6: Estimated disparity map between the reference and secondary images. *Images have been rotated 90° for the estimation.*

We then update the Y component of each key point with its disparity value. In Figure A.7, we show that there is no significant noise in the Y component of the displacement vectors.

A.7 Projection model + distortion benchmarking

At this stage, we have two sets of points: the key points K in the reference image, and the matched key points K' that we can obtain from K and their displacement vectors. We know that these points moved between both images because we rotated the sample by a known angle and we know that this sample is flat. From these information, we want to find the hypothesis - projection model + distortion - that best explain these displacements.

One of the main question is whether or not an orthographic projection model is sufficient or if we need to use the pinhole projection model (both models are discussed in Section 4.1.1).

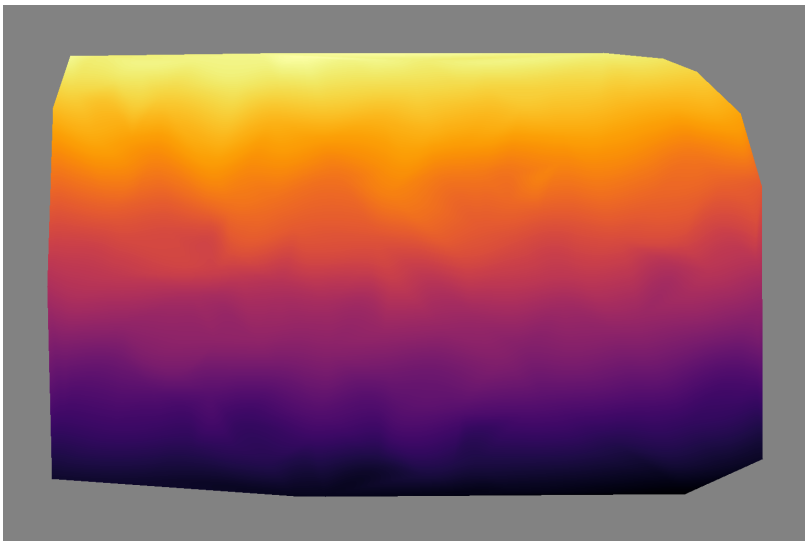


Figure A.7: Interpolation of the disparity values of the key points.
Interpolated with bicubic interpolation.

As a reminder, for the pinhole projection model, a 3D point $P = (X, Y, Z, 1)$ (homogeneous coordinates) is projected in the image into a 2D point $\tilde{p} = (\tilde{x}, \tilde{y}, \tilde{z})$ (homogeneous coordinates) using the following formula :

$$\tilde{p} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The orthographic projection model is simpler:

$$\tilde{p} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$r_{11}...r_{33}$ are known (as we know the rotation angle). However, if we use the pinhole projection model, we need to define the parameters f, t_x, t_y, t_z (c_x and c_y are set to the image center, considered as the optical center). For the orthographic projection model, we don't need to define any parameter, as t_x, t_y and t_z can simply be set to 0: in this case, the displacement of the point between the reference and the secondary images doesn't depend on the position of the sample nor the position of the rotation axis.

For using both models, we also need to define an origin point and its positions in the reference and secondary images. As all key points have an accurate enough displacement vector, we randomly select one of the key point as the origin. We take however into account the imprecision in the matching process. Let's suppose the position of the origin point is estimated to (x_{est}, y) in the secondary image whereas its true position should be located at (x_{true}, y). On both projection models, we define the additional parameter δ that estimates $x_{true} - x_{est}$.

We find those parameters by minimizing the scoring metric described in Algorithm 18. We use grid search first, and when the lowest score is found on this grid, we use the Nelder-Mead algorithm [111] to refine the parameters.

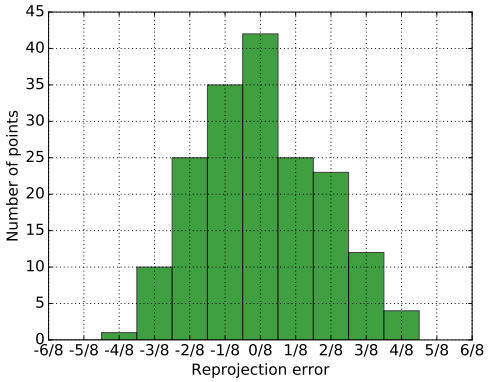
Algorithm 18: Calibration score to minimize.

Function *calibrationScoring(*K, K', α, α', *params)*

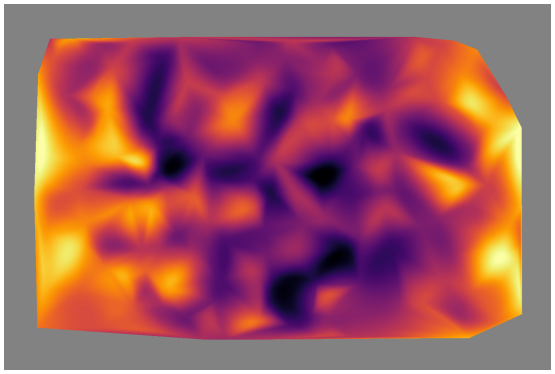
```
/* K is the list of key points. K' is the list of matched key
   points. α is the tilt angle of the reference image, α' the
   one of the secondary image. params contains the camera's
   parameters.                                                  */
/* First, we estimate for each key point its 3D position using
   triangulation as described in Section 4.1.2                  */
X, Y, Z ← triangulate(K, K', α, α', params) ;
/* We know that the sample is flat, so we model the key points'
   Z component by a linear regression depending on X and Y.     */
regression ← regressionFit(X, Y, Z) ;
/* We then get the modeled Z according to the regression.       */
mZ ← regression.predict(X, Y) ;
/* And we reproject the new points into the reference and
   secondary image.                                             */
K_p, K'_p ← project(X, Y, mZ, α, α', params) ;
/* We then compute the difference between the original positions
   and the reprojected positions: it is called the reprojection
   error.                                                       */
ε ← K_p − K ;
ε' ← K'_p − K' ;
/* The score we want to minimize is the mean of the absolute of
   these errors                                                 */
score ← ⟨ (||ε|| + ||ε'||) / 2 ⟩ ;
return score, ε' ;
```

The repartition of the reprojection errors of both models are shown in figures A.8 and A.9. There aren't any significant difference between the orthographic and pinhole projections models. However, there is a clear hyperbolic distortion in the X axis, though in the subpixel range.

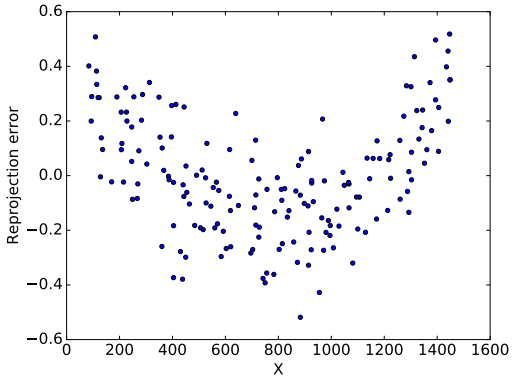
For correcting this effect, we consider the following distortion model, inspired by the Brown-Conrady model [112]:



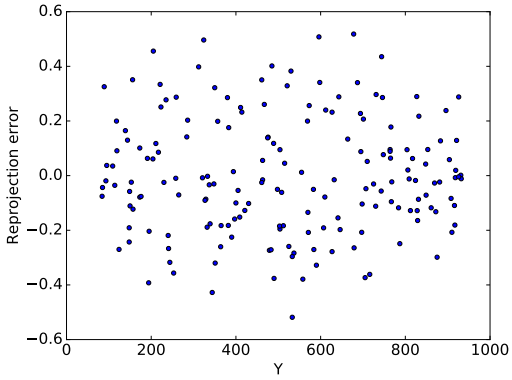
(a) Distribution of the error



(b) Error depending on the X and Y axes



(c) Error depending on the X axis



(d) Error depending on the Y axis

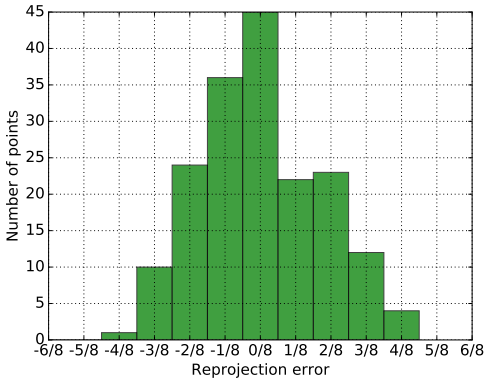
Figure A.8: Reprojection error of the orthographic model.

$$y_d = y_u(1 + K_1 * r^2)$$

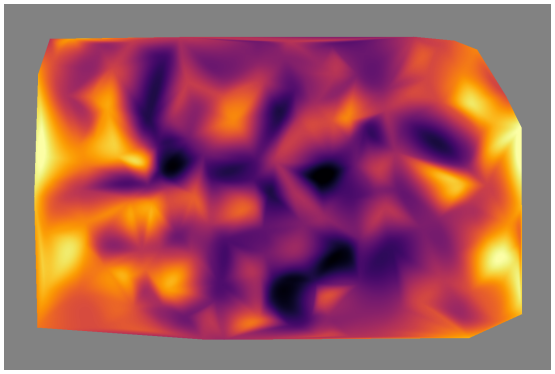
(x_d, y_d) being the distorted image point, (x_u, y_u) being the undistorted image point, K_1 being the radial distortion coefficient. We tried to use $r = \sqrt{(x_u - x_c)^2 + (y_u + y_c)^2}$, but it appeared that $r = |x_u|$ produced better results. (x_d, y_d) and (x_u, y_u) are coordinates relative to the center of the images (considered as the optical center). This parabolic distortion along a single axis is not compatible with the theoretical radial or spiral distortions expected for SEM images formation [113]. However, Sutton has also measured distortions on SEM images far from being radial [114].

As shown in figures A.10 and A.11, distortions effects were significantly reduced. As $K_1 > 0$ (see Table A.1), the observed distortion is close to a Pincushion distortion.

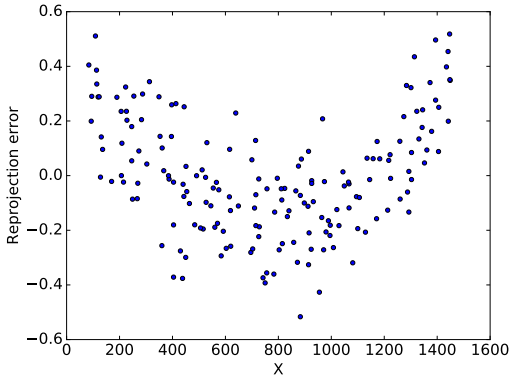
In order to choose the best projection model, we proceed the following way. We separate the key points into two random sets of equal size, the **train** set and the **test** set. We optimize the parameters of the model on the **train** set. We then quantify the error on the **test** set: we use the same Algorithm 18 on the points of the **test** set but using the parameters obtained for the **train** set. We repeat this experiment 100 times and get the average error and its standard deviation.



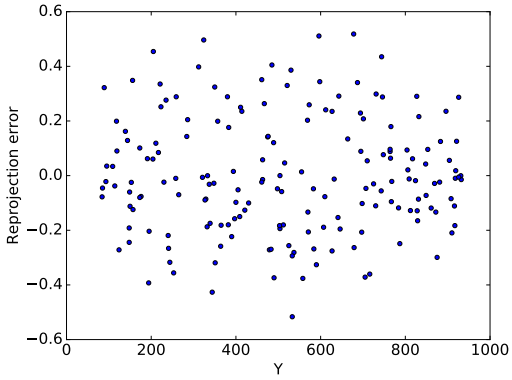
(a) Distribution of the error



(b) Error depending on the X and Y axes



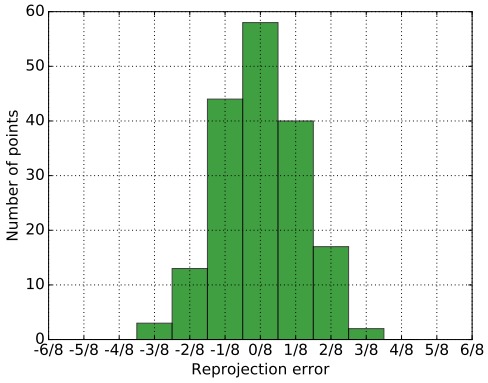
(c) Error depending on the X axis



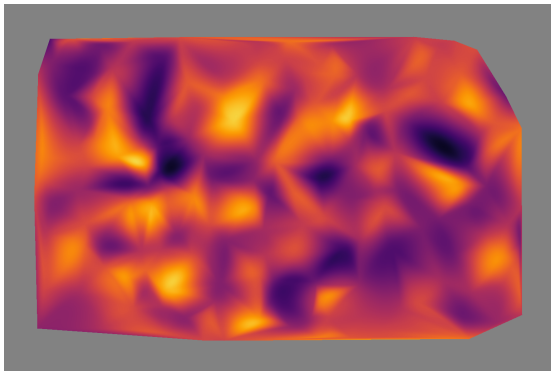
(d) Error depending on the Y axis

Figure A.9: Reprojection error of the pinhole model.

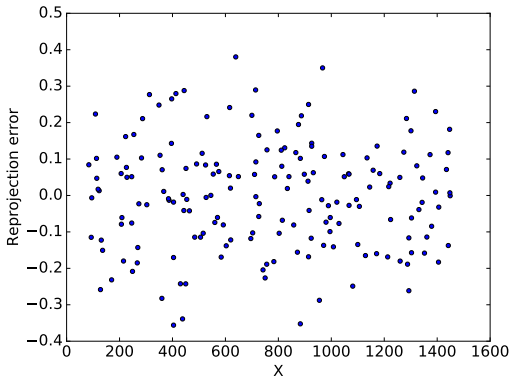
Obtained parameters and errors are shown in Table A.1. The Pinhole projection model, though obtaining a better score on the **train** set, doesn't reduce the errors on the **test** set. Taking into account distortion significantly reduces error. Therefore, the model chosen is the **orthographic projection model with distortion**.



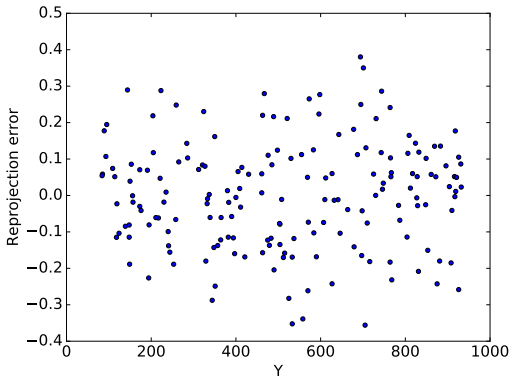
(a) Distribution of the error



(b) Error depending on the X and Y axes



(c) Error depending on the X axis

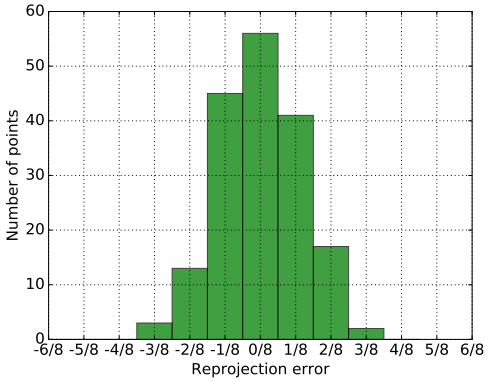


(d) Error depending on the Y axis

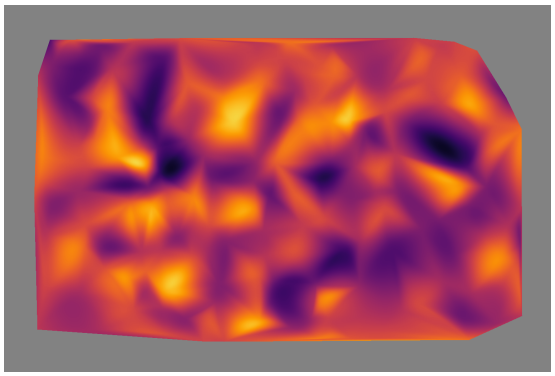
Figure A.10: Reprojection error of the orthographic model with distortion correction.

	Ortho.	**Pinhole**	**Ortho. + dist.**	**Pinhole + dist.**
δ	0.28	0.15	0.17	0.17
f	/	9.79e+08	/	1.09e+10
t_x	/	-4.69e-04	/	9.71e-04
t_y	/	5.43e+04	/	-1.57e-03
t_z	/	-1.70e-03	/	-7.84e+04
K_1	/	/	1.81e-07	1.81e-07
ϵ_{train}	8.55 ± 0.46	8.41 ± 0.55	5.79 ± 0.34	5.77 ± 0.30
ϵ_{test}	8.49 ± 0.46	8.62 ± 0.54	5.98 ± 0.36	5.98 ± 0.32

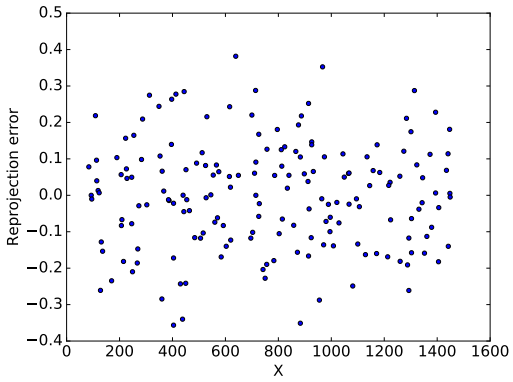
Table A.1: Error and parameters found for the different projection models.
Ortho.: Orthographic. Dist.: Distortion.



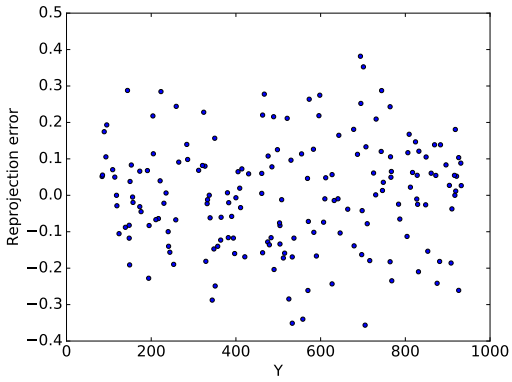
(a) Distribution of the error



(b) Error depending on the X and Y axes



(c) Error depending on the X axis



(d) Error depending on the Y axis

Figure A.11: Reprojection error of the pinhole model with distortion correction.

A.8 Calibration refinement using a sequence of images

If more than two views are available, the calibration process can also be refined. We will suppose that we have a reference image r acquired with a tilt of 0° and n secondary images $(s_1, s_2, ..., s_n)$ acquired with different tilts. We can then optimize the score in Algorithm 19. In that case, key points and their matches are obtained in a similar way than in sections A.2 and A.3. The origin point can also be chosen during recalibration (we choose the origin point that minimize the recalibration error).

Algorithm 19: Calibration refinement algorithm for multi-view stereo images.

Function *recalibration*(K, K'_1, K'_2, ..., K'_n, α, α'_1, α'_2, ..., α'_n, *params*)

```
/* K is the list of key points. K'_1 is the list of matched key
   points for the first secondary image, K'_2 for the second
   secondary image and so on. α is the tilt angle of the
   reference image, α'_1 the tilt angle of the first secondary
   image, α'_2 the tilt angle of the second secondary image and so
   on. params contains the parameters.                          */
/* First, we estimate for each key point and each pair
   (reference-secondary image) its 3D position using
   triangulation as described in Section 4.1.2                  */
foreach i ∈ [1, 2, ..., N] do
    X_i, Y_i, Z_i ← triangulate(K, K'_i, α, α'_i, params) ;
/* We then compute the average coordinate                       */
X ← ⟨X_i⟩ ;
Y ← ⟨Y_i⟩ ;
Z ← ⟨Z_i⟩ ;
/* And then reproject this average coordinate into each image and
   compute reprojection errors                                  */
foreach i ∈ [1, 2, ..., N] do
    K_p, K'_p ← project(X, Y, Z, α, α'_i, params) ;
    ε ← K_p − K ;
    ε' ← K'_p − K'_i ;
    /* We use median instead of the average of errors because
       matches are not manually checked so some of them might be
       erroneous.                                               */
    score_i ← median((||ε|| + ||ε'||) / 2) ;
score ← ⟨score_i⟩ ;
return score ;
```

Appendix B

Software provided

The software provided to IFPEN has been designed to be easily extendable and integrable to other software solutions. The architecture is similar to the NASA Ames Stereo Pipeline[106]; a collection of utilities have been developed that can be connected between each other. Each utility takes files as inputs and generates files as outputs. For instance, there is an utility that filters an image using the bilateral and median filters: it takes an input file (the original image) and generates an output file (the filtered image).

Some utilities, called **chain**, connect several utilities together. For instance, there is an utility called **chain_sem** that takes several images as input and generates a final height map as output : for doing so, it calls several utilities that will filter the images, estimate the disparity maps, use the TDSR algorithm and so on until the height map generation.

All those utilities are described in Appendix C and can be run using the command line. A GUI has also been implemented to allow chemists to easily use our solution without advanced computer skills. A screenshot is displayed in Figure B.1. In this GUI, the user only has to indicate the images paths as well as their tilt angle. He can also choose other parameters such as the image format, the method used, the search area for the stereo algorithms, and whether the images need to be normalized or not.

The software has also been integrated to IFPEN's internal image processing tool called INDIGO, with similar functionalities. A screenshot is displayed in Figure B.2.

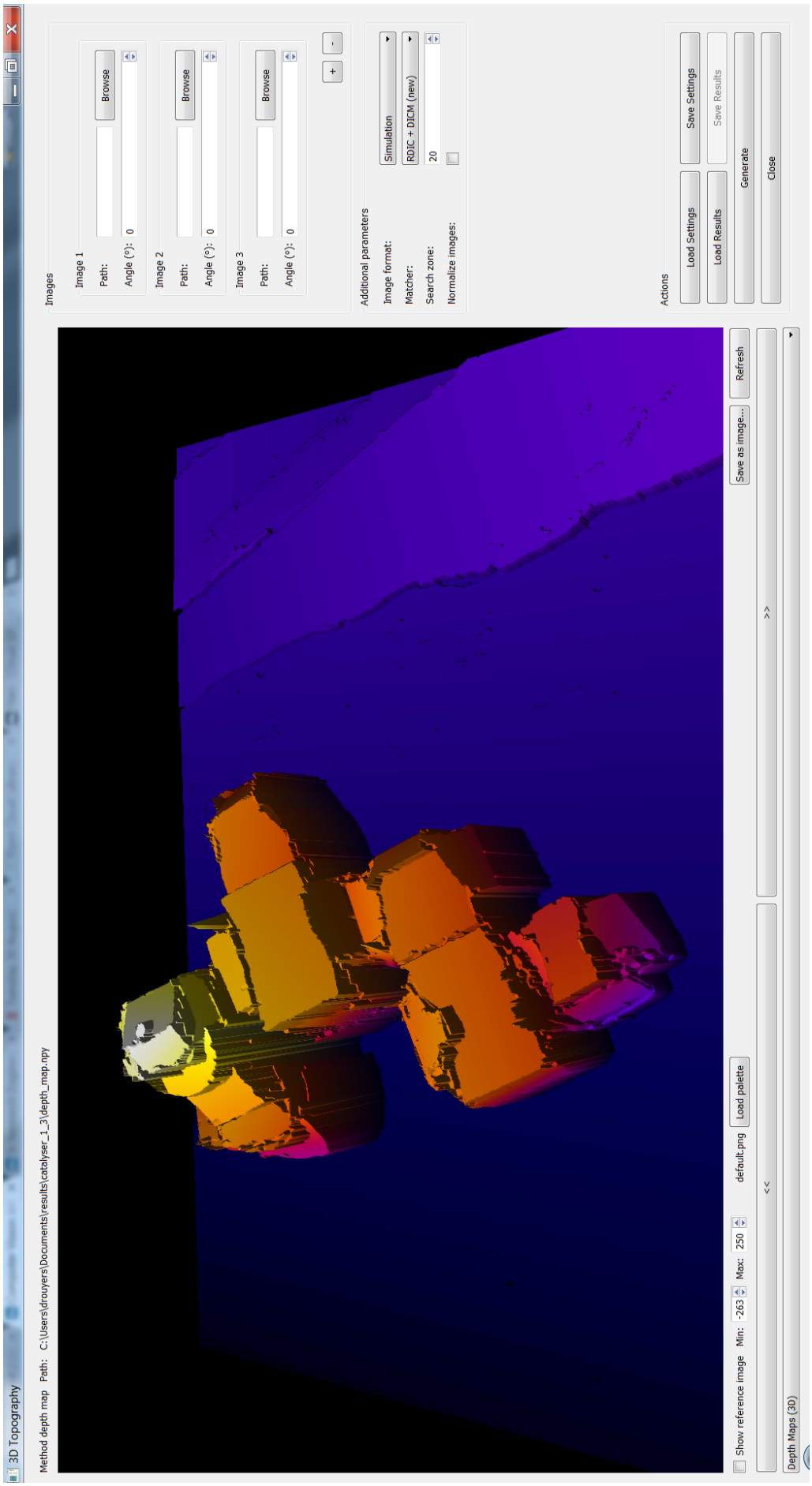


Figure B.1: The GUI of the software provided to IFPEN.

Figure B.2: The GUI of the software on INDIGO.

Appendix C

Software documentation

The software provided to IFPEN is a collection of utilities that can be connected between each other. We will describe them in this appendix.

List of utilities:

- chain_disp_with_init.py C.1
- chain_guided_disp_with_init.py C.2
- chain_sem.py C.3
- chain_simple_disp_with_init.py C.4
- compare_height_maps.py C.5
- convert.py C.6
- generate_confidence.py C.7
- generate_cropped_rotated.py C.8
- generate_diffused_rbs.py C.9
- generate_disp_opencv.py C.10
- generate_f_median_bilateral.py C.11
- generate_g_bricola.py C.12
- generate_g_classic.py C.13
- generate_g_classic_np.py C.14
- generate_g_mosaic.py C.15
- generate_height_from_disp.py C.16
- generate_h_waterfall.py C.17

C.1 chain_disp_with_init.py

```
usage: chain_disp_with_init.py

Enhances an existing disparity map using TDSR, with LRC check.

optional arguments:
  -h, --help            show this help message and exit
  --left_image_path L   Input path
  --right_image_path R  Output path
  --input_path IN       Input path
  --output_path OUT     Output path
  --start START         Starting step
  --stop STOP           Stopping step
  --config_path CONFIG_PATH
                        Configuration path
```

C.2 chain_guided_disp_with_init.py

```
usage: chain_guided_disp_with_init.py

Enhances an existing disparity map using TDSR, without LRC check, but
    by using FGS as a guide.

optional arguments:
  -h, --help            show this help message and exit
  --left_image_path L   Input path
  --input_path IN       Input path
  --output_path OUT     Output path
  --start START         Starting step
  --stop STOP           Stopping step
  --preview PREVIEW     Preview file
  --config_path CONFIG_PATH
                        Configuration path
```

C.3 chain_sem.py

```
usage: chain_sem.py

Computes a height map from a list of images acquired using a SEM.

optional arguments:
  -h, --help            show this help message and exit
  --input_path INPUT_PATH
                        Input path
  --output_path OUTPUT_PATH
                        Output path
  --output_file OUTPUT_FILE
                        Output file
  --tilts TILTS         Images tilt
  --start START         Starting step
  --stop STOP           Stopping step
  --simple_config_path SIMPLE_CONFIG_PATH
                        Configuration path
  --config_path CONFIG_PATH
                        Configuration path
  --default_config_path DEFAULT_CONFIG_PATH
                        Configuration path
```

C.4 chain_simple_disp_with_init.py

```
usage: chain_simple_disp_with_init.py

Enhances an existing disparity map using TDSR, without LRC check.
```

```
optional arguments:
  -h, --help            show this help message and exit
  --left_image_path L   Input path
  --input_path IN       Input path
  --output_path OUT     Output path
  --start START         Starting step
  --stop STOP           Stopping step
  --preview PREVIEW     Preview file
  --config_path CONFIG_PATH
                        Configuration path
```

C.5 compare_height_maps.py

```
usage: compare_height_maps.py height_map_path gt_path

Quantitatively compare a height map to a ground truth.

positional arguments:
  height_map_path       The height map path.
  gt_path               The image path.

optional arguments:
  -h, --help            show this help message and exit
  --large_error LARGE_ERROR
                        How much is considered a large error.
  --plot
  --no-plot
  --save_analysis_path SAVE_ANALYSIS_PATH
                        Where to save analysis.
```

C.6 convert.py

```
usage: convert.py IN OUT

Converts data (such as an image or a height map) from a format to
   another one. Supported formats include: jpeg, png, gif, tif, bmp,
   fda, pfm, npz (numpy zipped files).

positional arguments:
  IN          The input path.
  OUT         The output path.

optional arguments:
  -h, --help  show this help message and exit
```

C.7 generate_confidence.py

```
usage: generate_confidence.py MM CONF

Generate confidence map from model map.

positional arguments:
  MM          The model map path.
  CONF        The confidence map path.

optional arguments:
  -h, --help  show this help message and exit
```

C.8 generate_cropped_rotated.py

```
usage: generate_cropped_rotated.py IN OUT

Rotate the image and remove bottom strip.

positional arguments:
  IN                    The image path.
  OUT                   The rotated image path.

optional arguments:
  -h, --help            show this help message and exit
  --strip_height STRIP_HEIGHT
                        The height of the strip to be removed.
```

C.9 generate_diffused_rbs.py

```
usage: generate_diffused_rbs.py IM CONF DATA OUT

Diffuse data using the Robust Bilateral Solver.

positional arguments:
  IM          The image path.
  CONF        The confidence map path.
  DATA        The data map path.
  OUT         The diffused data map path.

optional arguments:
  -h, --help  show this help message and exit
```

C.10 generate_disp_opencv.py

```
usage: generate_disp_opencv.py L R DISP

Estimates the initial disparity maps between two images using the
   opencv library.

positional arguments:
  L                     The left image path
  R                     The right image path
  DISP                  The initial disparity maps path (LR and RL)

optional arguments:
  -h, --help            show this help message and exit
  --preview_path PREVIEW_PATH
                        Generates a preview of the initial disparity
   map
  --method METHOD       The method for computing disparity maps (BM
   or SGBM)
  --search_range SEARCH_RANGE
                        The search range (in px)
  --min_disparity MIN_DISPARITY
                        The minimum disparity value possible (by
   default:
                        -search_range/2)
  --lrc_threshold LRC_THRESHOLD
                        Remove values with LRC > this threshold
  --layers LAYERS       Layers (block size, uniqueness, texture)
  --wls WLS             Apply wls post-processing
  --wls_ddr WLS_DDR     WLS Disc Discontinuity Radius
  --wls_lambda WLS_LAMBDA
                        WLS Lambda parameter
  --wls_sigma WLS_SIGMA
                        WLS Sigma parameter
  --wls_lrc_threshold WLS_LRC_THRESHOLD
                        WLS LRC Threshold (x16)
  --recal_path RECAL_PATH
                        If defined, add back the borders removed for
   aligning
                        the images.
```

C.11 generate_f_median_bilateral.py

```
usage: generate_f_median_bilateral.py IN OUT

Filters an image using the median and bilateral filters. It also
   normalize the image.

positional arguments:
  IN                    Input image path
  OUT                   Output filtered image path

optional arguments:
```

```
  -h, --help            show this help message and exit
  --median_size MEDIAN_SIZE
                        Size of the median filter (disabled if <= 0)
  --bf_sigma_color BF_SIGMA_COLOR
                        Sigma color parameter of the bilateral filter
                        (disabled if <= 0)
  --bf_sigma_space BF_SIGMA_SPACE
                        Sigma space parameter of the bilateral filter
                        (disabled if <= 0)
  --std_amount STD_AMOUNT
                        Standardization of the image (disabled if <=
   0)
```

C.12 generate_g_bricola.py

```
usage: generate_g_bricola.py IN OUT

Computes the bricola gradient on an input image. /!\ Image size has
   to be a multiple of 64.

positional arguments:
  IN          Input image path
  OUT         Output gradient path

optional arguments:
  -h, --help  show this help message and exit
  --s1 S1     Bricola gradient lowest scale
  --s2 S2     Bricola gradient highest scale (should be higher than
   s1)
```

C.13 generate_g_classic.py

```
usage: generate_g_classic.py IN OUT

Computes the morphological gradient on an input image, using Mamba
   and an hexagonal grid. /!\ Image size has to be a multiple of 64.

positional arguments:
  IN           Input image path
  OUT          Output gradient path

optional arguments:
  -h, --help   show this help message and exit
  --size SIZE  Size of gradient
```

C.14 generate_g_classic_np.py

```
usage: generate_g_classic_np.py IN OUT

Computes the morphological gradient on an input image, using Numpy
   and a square grid.

positional arguments:
  IN            Input image path
  OUT           Output gradient path

optional arguments:
  -h, --help    show this help message and exit
  --size SIZE   Size of gradient (n x n block).
```

C.15 generate_g_mosaic.py

```
usage: generate_g_mosaic.py IN OUT

Computes the mosaic gradient of an input image.

positional arguments:
  IN                  Input image path
  OUT                 Output gradient path

optional arguments:
  -h, --help          show this help message and exit
  --size SIZE         Size of initial gradient (for computing h-minima
   )
  --hminima HMINIMA   h-minima for initial segmentation.
```

C.16 generate_height_from_disp.py

```
usage: generate_height_from_disp.py DISP IM A_REF A_SEC OUT

Converts disparity maps into height maps, assuming tilted
   acquisitions and orthographic projection.

positional arguments:
  DISP        The disparity maps path
  IM          The angle of the reference image
  A_REF       The tilt angle of the reference image
  A_SEC       The tilt angle of the secondary image
  OUT         The output height maps path

optional arguments:
  -h, --help  show this help message and exit
```

C.17 generate_h_waterfall.py

```
usage: generate_h_waterfall.py WS OUT

Computes the hierarchical segmentation using the enhanced waterfall
   algorithm.

positional arguments:
  WS          Input valued watershed path
  OUT         Output hierarchy file path

optional arguments:
  -h, --help  show this help message and exit
```

C.18 generate_init_merge.py

```
usage: generate_init_merge.py H [H ...] OUT

Merge height maps.

positional arguments:
  H                     The height maps paths
  OUT                   Output init

optional arguments:
  -h, --help            show this help message and exit
  --max_reprojection_error MAX_REPROJECTION_ERROR
                        Maximum reprojection error allowed (in px)
  --confidence_threshold CONFIDENCE_THRESHOLD
                        Proportion of height map consensus needed
  --merge_method MERGE_METHOD
                        Merge method used (smallest_majority:
   smallest block
                        size with consensus, highest_confidence:
   block size
                        with highest confidence, both)
  --preview PREVIEW     Preview file
```

C.19 generate_init_sparsify.py

```
usage: generate_init_sparsify.py G WS IN OUT

Sparsify a data map.

positional arguments:
  G             The gradient path
  WS            The watershed_path
  IN            The initial data map.
  OUT           The output data map.

optional arguments:
  -h, --help    show this help message and exit
  --size SIZE   The higher the parameter, the more points will be
   removed.
```

C.20 generate_mm_correct_with_guide.py

```
usage: generate_mm_correct_with_guide.py G MM GUIDE OUT

Corrects a model map according to a guide map.

positional arguments:
  G                     The reference image gradient path.
  MM                    The model map path.
  GUIDE                 The guide data path.
  OUT                   The output model map path

optional arguments:
  -h, --help            show this help message and exit
  --nb_iterations NB_ITERATIONS
                        Number of iterations
  --gradient_threshold GRADIENT_THRESHOLD
                        Max gradient between two regions for
   propagating a
                        model
  --max_diff MAX_DIFF   Max accepted difference between model map and
    guide
                        map
  --disc_threshold DISC_THRESHOLD
                        Low gradient relative to lowest border (
   default:
                        gradient_threshold / 2)
  --diff_threshold DIFF_THRESHOLD
                        Low difference relative to lowest neighboring
    region's
                        difference (default: max_diff / 3)
```

C.21 generate_mm_lrc_postprocessing.py

```
usage: generate_mm_lrc_postprocessing.py L R LMM RMM LPMM

Refines the model map using propagation and LRC checking.

positional arguments:
  L                     The left image path.
  R                     The right image path.
  LMM                   The left model map path.
  RMM                   The right model map path.
  LPMM                  The left model map path after post processing
    .

optional arguments:
  -h, --help            show this help message and exit
  --disc_seg_hminima HMIN
                        The hminima for the discontinuity
    segmentation.
```

C.22 generate_mm_postprocessing.py

```
usage: generate_mm_postprocessing.py IM MM OUT

Refines the model map using propagation.

positional arguments:
  IM                    The image path.
  MM                    The model map path.
  OUT                   The model map path after post processing.

optional arguments:
  -h, --help            show this help message and exit
  --disc_seg_hminima HMIN
                        The hminima for the discontinuity
    segmentation.
```

C.23 generate_mm_simplify.py

```
usage: generate_mm_simplify.py G MM INIT OUT

Simplifies the model map: similar models are merged together.

positional arguments:
  G                     The reference image gradient path.
  MM                    The model map path.
  INIT                  The init data map path.
```

```
  OUT                   The output model map path

optional arguments:
  -h, --help            show this help message and exit
  --tolerated_diff TOLERATED_DIFF
                        Tolerated diff from model
  --max_gradient MAX_GRADIENT
                        Until which gradient regions are merged
```

C.24 generate_mm_tdsr.py

```
usage: generate_mm_tdsr.py INIT H MM

Computes the model map using the TDSR algorithm.

positional arguments:
  INIT                  The initial data map path.
  H                     The hierarchy file path
  MM                    The output model map path

optional arguments:
  -h, --help            show this help message and exit
  --mg_config_path MG_CONFIG_PATH
                        The configuration path of the model getter (
   for
                        customizing models used)
  --models MODELS       The models class
  --selector SELECTOR   The selector class
  --preview PREVIEW     The preview file
  --preview_interval PREVIEW_INTERVAL
                        Interval between two generation of preview
   files
```

C.25 generate_mm_ws.py

```
usage: generate_mm_ws.py G INIT OUT

Computes the model map from segmentation and gradient (not using the
   hierarchy).

positional arguments:
  G                     Input gradient path
  INIT                  Initial data map path
  OUT                   Output model map path

optional arguments:
  -h, --help            show this help message and exit
  --hminima HMINIMA     Amount of hminima taken
```

```
  --alpha ALPHA         Bricola's adaptative erosion on hminima
  --initial_erosion_size INITIAL_EROSION_SIZE
                        Initial erosion on hminima
```

C.26 generate_raster.py

```
usage: generate_raster.py MM OUT

Generate raster data from model map. A post-processing removing small
    spikes is also added (but can be disabled).

positional arguments:
  MM                    The model map path.
  OUT                   The raster path.

optional arguments:
  -h, --help            show this help message and exit
  --rot90 ROT90         Rotations.
  --remove_spikes_change REMOVE_SPIKES_CHANGE
                        Remove spikes post-processing; remove when
   change is >
                        threshold. Disabled if <= 0.
  --remove_spikes_size REMOVE_SPIKES_SIZE
                        Remove spikes post-processing; remove when
   area is <
                        threshold. Disabled if <= 0.
```

C.27 generate_raster_from_init.py

```
usage: generate_raster_from_init.py INIT OUT

Get raster data from the initial data map.

positional arguments:
  INIT           The initial data map path.
  OUT            The output raster path.

optional arguments:
  -h, --help     show this help message and exit
  --rot90 ROT90  Number of 90 degrees rotations.
```

C.28 generate_recalled.py

```
usage: generate_recalled.py L R OUT_L OUT_R

Align two images on the Y axis by cropping them, using Digital Image
   Correlation.

positional arguments:
  L                     The left image path.
  R                     The right image path.
  OUT_L                 The recalled left image path.
  OUT_R                 The recalled right image path.

optional arguments:
  -h, --help            show this help message and exit
  --recal_path RECAL_PATH
                        If defined, save the recal information in
   this path.
  --search_zone_size SEARCH_ZONE_SIZE
                        Search zone size.
  --match_zone_size MATCH_ZONE_SIZE
                        Match block size.
  --left_linked_images_paths LEFT_LINKED_IMAGES_PATHS
                        Linked images to the left image (will be
   cropped the
                        same way as the left image).
  --right_linked_images_paths RIGHT_LINKED_IMAGES_PATHS
                        Linked images to the right image (will be
   cropped the
                        same way as the right image).
```

C.29 generate_seg_from_mm.py

```
usage: generate_seg_from_mm.py MM SEG

Generate the segmentation image from a model map.

positional arguments:
  MM          Input model map path
  SEG         Output segmentation path

optional arguments:
  -h, --help  show this help message and exit
```

C.30 generate_tangent_from_mm.py

```
usage: generate_tangent_from_mm.py MM TA

Generate the tangent map from a model map. y=ax+by+c, Red:a, Green:b,
    Blue:c
```

```
positional arguments:
  MM                    Input model map path
  TA                    Output tangent map path

optional arguments:
  -h, --help            show this help message and exit
  --extrema EXTREMA     If absolute value is over, truncate.
  --remove_axis REMOVE_AXIS
                        Remove axis number (a: 0, b: 1, c: 3)
```

C.31 generate_ws_bricola.py

```
usage: generate_ws_bricola.py G WS

Computes the valued watershed with bricola's adaptative erosion.

positional arguments:
  G                   Input gradient path
  WS                  Output watershed path

optional arguments:
  -h, --help          show this help message and exit
  --hminima HMINIMA   Amount of hminima taken
  --alpha ALPHA       Bricola's adaptative erosion on hminima
```

C.32 generate_ws_classic.py

```
usage: generate_ws_classic.py G WS

Computes the valued watershed of an image.

positional arguments:
  G                   Input gradient path
  WS                  Output watershed path

optional arguments:
  -h, --help          show this help message and exit
  --hminima HMINIMA   Amount of hminima taken
```

C.33 rotate_im.py

```
usage: rotate_im.py IN NB OUT

Rotates an image.

positional arguments:
  IN          Input image path
  NB          Number of 90 degrees rotations
  OUT         Output rotated image path

optional arguments:
  -h, --help  show this help message and exit
```

Bibliography

[1] L. Reimer, *Scanning Electron Microscopy*. 2nd edition, Springer Berlin Heidelberg, 1998.

[2] E. M. Le Cui, "Calibration of scanning electron microscope using a multi-image non-linear minimization process," in *2014 IEEE International Conference on Robotics & Automation*, 2014.

[3] N. Cornille, *Accurate 3D Shape and Displacement Measurement using a Scanning Electron Microscope*. PhD thesis, University of South Carolina, Institut National des Sciences Appliquées de Toulouse, Juin 2005.

[4] N. Cornille, D. Garcia, M. A. Sutton, S. McNeill, and J.-J. Orteu, "Automated 3-d reconstruction using a scanning electron microscope," in *SEM conference on experimental and applied mechanics*, 2003.

[5] E. Faber, D. Martinez-Martinez, C. Mansilla, V. Ocelík, and J. D. Hosson, "Calibration-free quantitative surface topography reconstruction in scanning electron microscopy," *Ultramicroscopy*, 2015.

[6] "TGF 11 webpage." `http://www.spmtips.com/test-structures-TGF11-series.html`. Accessed on 03/23/2017.

[7] J. Allen, "Perfect reconstruction filter banks for the hexagon grid," in *Information, Communications and Signal Processing, 2005 Fifth International Conference on*, pp. 73–76, IEEE, 2005.

[8] R. C. Staunton and N. Storey, "A comparison between square and hexagonal sampling methods for pipeline image processing," in *Proc. SPIE*, vol. 1194, pp. 142–151, 1989.

[9] J. Ohser and K. Schladitz, *3D images of materials structures: processing and analysis*. John Wiley & Sons, 2009.

[10] J. W. Tukey, "Exploratory data analysis," 1977.

[11] B. Chanda and D. D. Majumder, *Digital image processing and analysis*. PHI Learning Pvt. Ltd., 2011.

[12] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Computer Vision, 1998. Sixth International Conference on*, pp. 839–846, IEEE, 1998.

[13] Itseez, "Open source computer vision library." `https://github.com/itseez/opencv`, 2015.

[14] S. Beucher, *Segmentation d'images et Morphologie Mathématique*. PhD thesis, École Nationale Supérieur des Mines de Paris, Juin 1990.

[15] A. Rosenfeld and J. L. Pfaltz, "Distance functions on digital pictures," *Pattern recognition*, vol. 1, no. 1, pp. 33–61, 1968.

[16] P.-E. Danielsson, "Euclidean distance mapping," *Computer Graphics and image processing*, vol. 14, no. 3, pp. 227–248, 1980.

[17] L. Ikonen, "Priority pixel queue algorithm for geodesic distance transforms," *Image Vision Comput.*, vol. 25, pp. 1520–1529, Oct. 2007.

[18] F. Meyer, "Levelings, image simplification filters for segmentation," *Journal of Mathematical Imaging and Vision*, vol. 20, no. 1, pp. 59–72, 2004.

[19] F. Meyer, "Levelings: theory and practice," in *Handbook of Mathematical Models in Computer Vision*, pp. 63–78, Springer, 2006.

[20] S. Beucher and F. Meyer, "The morphological approach to segmentation: the watershed transformation," *Optical Engineering-New York-Marcel Dekker Incorporated-*, vol. 34, pp. 433–433, 1992.

[21] B. Marcotegui and S. Beucher, *Fast Implementation of Waterfall Based on Graphs*, pp. 177–186. Dordrecht: Springer Netherlands, 2005.

[22] N. Beucher and S. Beucher, "Hierarchical queues: general description and implementation in mamba image library," 2011.

[23] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 5, pp. 898–916, 2011.

[24] J. Angulo, *Mathematical morphology and image colour indexing : application in bio-medical microscopy*. Theses, École Nationale Supérieure des Mines de Paris, Dec. 2003.

[25] V. Risson, *Application de la morphologie mathématique à l'analyse des conditions d'éclairage des images couleur*. Theses, École Nationale Supérieure des Mines de Paris, Dec. 2001.

[26] J.-C. Bricola, M. Bilodeau, and S. Beucher, "A multi-scale and morphological gradient preserving contrast," in *14th International Congress for Stereology and Image Analysis*, (Liège, Belgium), Eric Pirard, July 2015.

[27] S. Beucher, "Numerical residues," *Image and Vision Computing*, vol. 25, no. 4, pp. 405–415, 2007.

[28] N. Beucher and S. Beucher, "Mamba image examples," tech. rep., 2015.

[29] J.-C. Bricola, M. Bilodeau, and S. Beucher, "A top-down methodology to depth map estimation controlled by morphological segmentation," *HAL*, 2014.

[30] S. Beucher, "Towards a unification of waterfalls, standard and P algorithms." working paper or preprint, Jan. 2013.

[31] S. Beucher and B. Marcotegui, "P algorithm, a dramatic enhancement of the waterfall transformation." working paper or preprint, Sept. 2009.

[32] S. Beucher and B. Marcotegui, "P algorithm, a dramatic enhancement of the waterfall transformation," *CMM/Mines Paristech publication*, 2009.

[33] P. A. Arbeláez and L. D. Cohen, "Energy partitions and image segmentation," *Journal of Mathematical Imaging and Vision*, vol. 20, no. 1, pp. 43–57, 2004.

[34] A. Vedaldi and S. Soatto, "Quick shift and kernel methods for mode seeking," *Computer vision–ECCV 2008*, pp. 705–718, 2008.

[35] G. Binnig, C. F. Quate, and C. Gerber, "Atomic force microscope," *Physical review letters*, vol. 56, no. 9, p. 930, 1986.

[36] R. Crowther, D. DeRosier, and A. Klug, "The reconstruction of a three-dimensional structure from projections and its application to electron microscopy," in *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 317, pp. 319–340, The Royal Society, 1970.

[37] Y. Liu, A. M. Kiss, D. H. Larsson, F. Yang, and P. Pianetta, "To get the most out of high resolution x-ray tomography: A review of the post-reconstruction analysis," *Spectrochimica Acta Part B: Atomic Spectroscopy*, vol. 117, pp. 29 – 41, 2016.

[38] B. K. Horn, "Obtaining shape from shading information," in *Shape from shading*, pp. 123–171, MIT press, 1989.

[39] T. P. Ellison and C. J. Taylor, "Calculating the surface topography of integrated circuit wafers from sem images," *Image and Vision Computing*, 1990.

[40] A. Jones and C. Taylor, "Automated interpretation of sem images," in *TENCON '94. IEEE Region 10's Ninth Annual International Conference. Theme: Frontiers of Computer Technology. Proceedings of 1994*, 1994.

[41] R. Pintus, S. Podda, F. Mighela, and M. Vanzi, "Quantitative 3D reconstruction from bs imaging," in *Microelectronics Reliability*, 2004.

[42] O. C. Wells, C. E. Murray, J. L. Rullan, and L. M. Gignac, "Top-down topography of deeply etched silicon in the scanning electron microscope," *Review of Scientific Instruments*, 2004.

[43] A. D. Kammers and S. Daly, "Small-scale patterning methods for digital image correlation under scanning electron microscopy," *Measurement Science and Technology*, 2011.

[44] "Mountainsmap website." `http://www.digitalsurf.com/en/mntspm.html`. Accessed on 05/17/2015.

[45] O. Sinram, M. Ritter, S. Kleindiek, A. Schertel, H. Hohenberg, and J. Albertz, "Calibration of an SEM, using a nano positioning tilting table and a microscopic calibration pyramid," *The International Archives of Photogrammetry and Remote Sensing. Vol. XXXIV, part 5, pp. 210-216*, 2002.

[46] O. Faugeras, *Three-dimensional computer vision: a geometric viewpoint*. MIT press, 1993.

[47] C. Loop and Z. Zhang, "Computing rectifying homographies for stereo vision," in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, vol. 1, pp. 125–131, IEEE, 1999.

[48] A. Fusiello, E. Trucco, and A. Verri, "A compact algorithm for rectification of stereo pairs," *Machine Vision and Applications*, vol. 12, no. 1, pp. 16–22, 2000.

[49] S. Kumar, C. Micheloni, C. Piciarelli, and G. L. Foresti, "Stereo rectification of uncalibrated and heterogeneous images," *Pattern Recognition Letters*, vol. 31, no. 11, pp. 1445–1452, 2010.

[50] A. A. Goshtasby, *Image registration: Principles, tools and methods*. Springer Science & Business Media, 2012.

[51] R. O. Duda and P. E. Hart, *Pattern classification and scene analysis*. New York: Wiley-Interscience Publication, 1973.

[52] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *European conference on computer vision*, pp. 151–158, Springer, 1994.

[53] R. W. Hamming, "Error detecting and error correcting codes," *Bell Labs Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[54] J. Zbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *CoRR*, vol. abs/1510.05970, 2015.

[55] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," *Shape, contour and grouping in computer vision*, pp. 823–823, 1999.

[56] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[57] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," 1974.

[58] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the 4th Alvey Visiion Conference*, 1988.

[59] P. Fua, "A parallel stereo algorithm that produces dense depth maps and preserves image features," *Machine Vision and Applications*, vol. 6, pp. 35–49, dec 1993.

[60] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 23, no. 11, pp. 1222–1239, 2001.

[61] V. Kolmogorov and R. Zabih, "Computing visual correspondence with occlusions using graph cuts," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 2, pp. 508–515, IEEE, 2001.

[62] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient belief propagation for early vision," *International journal of computer vision*, vol. 70, no. 1, pp. 41–54, 2006.

[63] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 10, pp. 1568–1583, 2006.

[64] M. J. Wainwright, T. Jaakkola, and A. S. Willsky, "Tree-based reparameterization for approximate inference on loopy graphs," in *Advances in neural information processing systems*, pp. 1001–1008, 2002.

[65] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, "A comparative study of energy minimization methods for markov random fields with smoothness-based priors," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 6, pp. 1068–1080, 2008.

[66] L. Li, S. Zhang, X. Yu, and L. Zhang, "PMSC: Patchmatch-based superpixel cut for accurate stereo matching," *IEEE Transactions on Circuits and Systems for Video Technology*, 2016.

[67] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 328–341, 2008.

[68] G. Facciolo, C. de Franchis, and E. Meinhardt, "MGM: A significantly more global matching for stereovision," in *BMVC*, 2015.

[69] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 2, pp. 504–511, 2013.

[70] J.-C. Bricola, *Depth map estimation from stereo images & mathematical morphology*. Theses, PSL Research University, Oct. 2016.

[71] S. N. Sinha, D. Scharstein, and R. Szeliski, "Efficient high-resolution stereo matching using local plane sweeps," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1582–1589, 2014.

[72] S. Hermann and R. Klette, "Iterative semi-global matching for robust driver assistance systems," in *Asian Conference on Computer Vision*, pp. 465–478, Springer, 2012.

[73] K. Zhu, P. d'Angelo, and M. Butenuth, "Evaluation of stereo matching costs on close range, aerial and satellite images," 2012.

[74] A. Banno and K. Ikeuchi, "Disparity map refinement and 3D surface smoothing via directed anisotropic diffusion," *Computer Vision and Image Understanding*, vol. 115, no. 5, pp. 611–619, 2011.

[75] D. A. Lima, G. B. Vitor, A. C. Victorino, and J. V. Ferreira, "A disparity map refinement to enhance weakly-textured urban environment data," in *Advanced Robotics (ICAR), 2013 16th International Conference on*, pp. 1–6, IEEE, 2013.

[76] J. Ralli, J. Díaz, and E. Ros, "A method for sparse disparity densification using voting mask propagation," *Journal of Visual Communication and Image Representation*, vol. 21, pp. 67–74, jan 2010.

[77] D. Min, S. Choi, J. Lu, B. Ham, K. Sohn, and M. N. Do, "Fast global image smoothing based on weighted least squares," *IEEE Transactions on Image Processing*, vol. 23, pp. 5638–5653, dec 2014.

[78] J. T. Barron and B. Poole, "The fast bilateral solver," in *Computer Vision – ECCV 2016*, pp. 617–632, Springer Nature, 2016.

[79] J. Ralli, F. Pelayo, and J. Diaz, "Increasing efficiency in disparity calculation," in *Lecture Notes in Computer Science*, pp. 298–307, Springer Nature.

[80] J. Weickert, *Anisotropic diffusion in image processing.* PhD thesis, 1998.

[81] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. M. Seitz, "Multi-view stereo for community photo collections," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, IEEE, 2007.

[82] Y. Furukawa and J. Ponce, "Accurate, dense, and robust multiview stereopsis," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 8, pp. 1362–1376, 2010.

[83] C. Wu, "Towards linear-time incremental structure from motion," in *3DTV-Conference, 2013 International Conference on*, pp. 127–134, IEEE, 2013.

[84] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157, Ieee, 1999.

[85] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *Computer vision and pattern recognition, 2006 IEEE Computer Society Conference on*, vol. 1, pp. 519–528, IEEE, 2006.

[86] Y. Furukawa, C. Hernández, *et al.*, "Multi-view stereo: A tutorial," *Foundations and Trends® in Computer Graphics and Vision*, vol. 9, no. 1-2, pp. 1–148, 2015.

[87] C. H. Esteban and F. Schmitt, "Silhouette and stereo fusion for 3D object modeling," *Computer Vision and Image Understanding*, vol. 96, no. 3, pp. 367–392, 2004.

[88] G. Vogiatzis, C. H. Esteban, P. H. Torr, and R. Cipolla, "Multiview stereo via volumetric graph-cuts and occlusion robust photo-consistency," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 12, pp. 2241–2246, 2007.

[89] N. Campbell, G. Vogiatzis, C. Hernández, and R. Cipolla, "Using multiple hypotheses to improve depth-maps for multi-view stereo," *Computer Vision–ECCV 2008*, pp. 766–779, 2008.

[90] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang, and P. Westling, "High-resolution stereo datasets with subpixel-accurate ground truth," in *Lecture Notes in Computer Science*, pp. 31–42, Springer Nature, 2014.

[91] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[92] "TGXYZ webpage." `http://www.spmtips.com/test-structures-TGXYZ-series.html`. Accessed on 03/23/2017.

[93] J. Sempau, E. Acosta, J. Baro, J. Fernández-Varea, and F. Salvat, "An algorithm for Monte Carlo simulation of coupled electron-photon transport," *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, vol. 132, no. 3, pp. 377–390, 1997.

[94] F. Salvat, J. M. Fernández-Varea, E. Acosta, and J. Sempau, "A code system for Monte Carlo simulation of electron and photon transport," 2001. NEA Report.

[95] S. J. Koppal, *Lambertian Reflectance*, pp. 441–443. Boston, MA: Springer US, 2014.

[96] "Official website of Unity 3D." `http://unity3d.com/`. Accessed on 05/20/2015.

[97] M. Bleyer and M. Gelautz, "A layered stereo matching algorithm using image segmentation and global visibility constraints," *ISPRS Journal of Photogrammetry and remote sensing*, vol. 59, no. 3, pp. 128–150, 2005.

[98] N. Ma, Y. Men, C. Men, and X. Li, "Accurate dense stereo matching based on image segmentation using an adaptive multi-cost approach," vol. 8, p. 159, 12 2016.

[99] P. Salembier and L. Garrido, "Binary partition tree as an efficient representation for filtering, segmentation and information retrieval," in *ICIP 98*.

[100] S. Drouyer, S. Beucher, M. Bilodeau, M. Moreaud, and L. Sorbier, "Sparse stereo disparity map densification using hierarchical image segmentation," in *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*, pp. 172–184, Springer, 2017.

[101] C. Vachier and F. Meyer, "The viscous watershed transform," *Journal of Mathematical Imaging and Vision*, vol. 22, pp. 251–267, may 2005.

[102] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, pp. 381–395, jun 1981.

[103] S. E. Reutebuch, H.-E. Andersen, and R. J. McGaughey, "Light detection and ranging (lidar): an emerging tool for multiple resource inventory," *Journal of Forestry*, vol. 103, no. 6, pp. 286–292, 2005.

[104] C. De Franchis, *Earth Observation and Stereo Vision*. PhD thesis, Université Paris-Saclay, 2015.

[105] S. K. Singh, S. D. Naidu, T. Srinivasan, B. G. Krishnaa, and P. Srivastava, "Rational polynomial modelling for cartosat-1 data," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 37, 2008.

[106] Z. Moratto, M. Broxton, R. Beyer, M. Lundy, and K. Husmann, "Ames stereo pipeline, NASA's open source automated stereogrammetry software," in *Lunar and Planetary Science Conference*, vol. 41, p. 2364, 2010.

[107] M. Bosch, A. Leichtman, D. Chilcott, H. Goldberg, and M. Brown, "Metric evaluation pipeline for 3D modeling of urban scenes.," *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 42, 2017.

[108] Blender Online Community, *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2017.

[109] K. Lai, L. Bo, X. Ren, and D. Fox, "A large-scale hierarchical multi-view RGB-D object dataset," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1817–1824, IEEE, 2011.

[110] J. R. Quinlan *et al.*, "Learning with continuous classes," in *5th Australian joint conference on artificial intelligence*, vol. 92, pp. 343–348, Singapore, 1992.

[111] J. Dennis and D. J. Woods, "Optimization on microcomputers: The Nelder-Mead simplex algorithm," *New computing environments: microcomputers in large-scale computing*, vol. 11, pp. 6–122, 1987.

[112] A. E. Conrady, "Decentred lens-systems," *Monthly notices of the royal astronomical society*, vol. 79, no. 5, pp. 384–390, 1919.

[113] L. Cui and E. Marchand, "Calibration of scanning electron microscope using a multi-image non-linear minimization process," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 5191–5196, IEEE, 2014.

[114] M. Sutton, "Accurate image distortion correction in high magnification imaging systems," *SPIE Newsroom*, 2007.

Résumé

Le but de ce travail est de fournir une méthode de reconstruction stéréoscopique capable d'estimer la topographie des catalyseurs à partir d'images MEB. Les méthodes stéréo standard ne permettent pas d'évaluer des reconstructions 3D de bonne qualité en raison de la surface homogène de ces échantillons. Bien que particulièrement prononcé sur nos catalyseurs, le manque de texture est un problème courant dans la reconstruction stéréo, et aucune solution idéale n'a encore été trouvée.

Notre approche principale à ce problème est de combiner les méthodes stéréo existantes avec la segmentation hiérarchique des images de l'échantillon. En effet, la morphologie mathématique fournit des outils efficaces permettant de diviser une image en régions et sous-régions. Nous avons utilisé ces outils pour affiner et compléter les reconstructions 3D.

La méthode que nous avons développé estime des reconstructions 3D moins bruitées et plus précises que les méthodes existantes. L'approche fournit également des informations supplémentaires: la segmentation finalement retenue ainsi que la carte indiquant l'orientation de chaque région sont des données intéressantes qui peuvent être utilisées pour affiner la compréhension des catalyseurs.

Bien que le but de cette thèse soit très spécifique, l'approche proposée est généraliste. Elle a été notamment testée sur la base Middlebury et les résultats obtenus sont comparables et parfois meilleurs que les méthodes de pointe.

L'approche pourrait aussi être étendue à d'autres cas d'utilisation. Tant que des données spatiales sont combinées avec une image, notre méthode TDSR peut être utilisée pour améliorer et compléter ces données spatiales. Les images RGBD et la segmentation sémantique sont quelques exemples d'applications potentielles.

Mots Clés

Morphologie mathématique, mesure stéréoscopique, segmentation, MEB.

Abstract

The aim of this work is to provide a stereo reconstruction method able to estimate the topography of catalysts from SEM images. Standard stereo methods fail to evaluate adequate 3D reconstructions because of the homogeneous surface of these samples. Though particularly pronounced on our catalysts, the lack of texture is a common issue in stereo reconstruction, and no ideal solution has yet been found.

Our main approach to this issue is to combine existing stereo methods with the hierarchical segmentation of the sample's images. Indeed, Mathematical Morphology provides efficient tools that divide an image into regions and subregions. We have used these tools to refine and complete the 3D reconstructions.

The method we have developed estimates 3D reconstructions that are less noisy and more precise than state of the art methods. The approach also provides additional information: the final segmentation as well as the normal map are interesting data that can be used to refine the understanding of the catalysts.

Though this thesis' purpose is very specific, the proposed approach is general.

It has been notably used in the Middlebury database which contains images of in-door scenes, and obtained results were comparable and sometimes better than state of the art methods.

It could also be extended to other uses. As long as spatial data is combined with an image, our TDSR method can be used to refine it. RGBD images and semantic segmentation are a few potential applications.

Keywords

Mathematical morphology, stereovision, segmentation, SEM.