Pontifícia Universidade Católica
do Rio de Janeiro

Axelle Dany Juliette Pochet

Modeling of Geobodies: AI for seismic fault detection and all-quadrilateral mesh generation

Tese de Doutorado

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências – Informática.

Advisor : Prof. Marcelo Gattass
Co-advisor: Prof. Hélio Côrtes Vieira Lopes

Rio de Janeiro
September 2018

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Axelle Dany Juliette Pochet

Modeling of Geobodies: AI for seismic fault detection and all-quadrilateral mesh generation

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências – Informática. Approved by the undersigned Examination Committee.

Rio de Janeiro, September 28th, 2018

Axelle Dany Juliette Pochet

Graduated in Geology at the Ecole Nationale Supérieure de Géologie (ENSG) in Nancy, France, in 2013. The same year, she obtained a master degree in Numerical Geology from the Institut National Polytechnique de Lorraine (INPL) in Nancy, France.

Bibliographic data

Pochet, Axelle Dany Juliette

Modeling of Geobodies: AI for seismic fault detection and all-quadrilateral mesh generation / Axelle Dany Juliette Pochet; advisor: Marcelo Gattass; co-advisor: Hélio Côrtes Vieira Lopes. – 2018.

128 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamente de Informática, 2018.

Inclui bibliografia

1. Informática – Teses. 2. Falhas Sísmicas;. 3. Redes Neurais Convolucionais;. 4. Transferência de aprendizado;. 5. Geração de Malha;. 6. Quadtree.. I. Gattass, Marcelo. II. Côrtes Vieira Lopes, Hélio. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

Acknowledgments

First, I would like to express my sincere gratitude to my advisor, Prof. Marcelo Gattass, for his continuous support and precious advices all along my Ph.D. Those years of inspiring research and study in this beautiful city would not have been possible without him. I would also like to give a special thanks to my co-advisor, Prof. Hélio Cortes Vieira Lopes, for his endless optimism and for giving me motivation whenever I needed it.

I would also like to thank my committee members, Prof. Waldemar Celes Filho, Prof. Luiz Fernando Campos Ramos Martha, Prof. Aristófanes Corrêa Silva and Prof. Paulo Cezar Pinto Carvalho, for their insightful comments and suggestions. Thanks to their kindness and sincere interest in my research, my defense was an enjoyable moment that I will not forget.

My thanks also go to my colleagues and friends at the Tecgraf Institute and PUC-Rio. Especially, thanks to Jeferson Coelho, Pedro Henrique Bandeira Diniz and William Paulo Ducca Fernandes for their help, their kind and motivating words, and their support in my research. Thanks to Erwan Renault and Murillo Santana for their mental support and good time we spent in Rio.

I would like to thank my family and friends for their long distance support and for visiting me many times. A special thanks goes to my parents whose happiness, humor and optimism are priceless. Thanks also to my long distance sisters all over the world: Inus, la Gaga, Rorchal la Rogette, Khakha, Lanlan, Hélène, Tatchianaou and Polypolya.

Last but not least, I would like to thank my beloved husband Rémy Junius for being the real MVP.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

Abstract

Axelle Dany Juliette Pochet; Marcelo Gattass (Advisor). **Modeling of Geobodies: AI for seismic fault detection and all-quadrilateral mesh generation**. Rio de Janeiro, 2018. 128p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Safe oil exploration requires good numerical modeling of the subsurface geobodies, which includes among other steps: seismic interpretation and mesh generation. This thesis presents a study in these two areas. The first study is a contribution to data interpretation, examining the possibilities of automatic seismic fault detection using deep learning methods. In particular, we use Convolutional Neural Networks (CNNs) on seismic amplitude maps, with the particularity to use synthetic data for training with the goal to classify real data. In the second study, we propose a new two-dimensional all-quadrilateral meshing algorithm for geomechanical domains, based on an innovative quadtree approach: we define new subdivision patterns to efficiently adapt the mesh to any input geometry. The resulting mesh is suited for Finite Element Method (FEM) simulations.

Keywords

Seismic fault; Convolutional Neural Network; Transfer Learning; Mesh generation; Quadtree.

Resumo

Axelle Dany Juliette Pochet; Marcelo Gattass. **Modelagem de objetos geológicos: IA para detecção automática de falhas e geração de malhas de quadriláteros**. Rio de Janeiro, 2018. 128p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A exploração segura de reservatórios de petróleo necessita uma boa modelagem numérica dos objetos geológicos da sub superfície, que inclui entre outras etapas: interpretação sísmica e geração de malha. Esta tese apresenta um estudo nessas duas áreas. O primeiro estudo é uma contribuição para interpretação de dados sísmicos, que se baseia na detecção automática de falhas sísmicas usando redes neurais profundas. Em particular, usamos Redes Neurais Convolucionais (RNCs) diretamente sobre mapas de amplitude sísmica, com a particularidade de usar dados sintéticos para treinar a rede com o objetivo final de classificar dados reais. Num segundo estudo, propomos um novo algoritmo para geração de malhas bidimensionais de quadrilaterais para estudos geomecânicos, baseado numa abordagem inovadora do método de quadtree: definimos novos padrões de subdivisão para adaptar a malha de maneira eficiente a qualquer geometria de entrada. As malhas obtidas podem ser usadas para simulações com o Método de Elementos Finitos (MEF).

Palavras-chave

Falhas Sísmicas; Redes Neurais Convolucionais; Transferência de aprendizado; Geração de Malha; Quadtree.

Table of contents

List of figures

List of tables

List of Abreviations

AUC – Area Under the Curve

CNN – Convolutional Neural Network

FE – Feature Extractor

FEM – Finite Element Method

FFT – Full Fine-Tuning

FN – False Negative

FP – False Positive

HE – Half Edge

MDS – Multi-Dimensional Scaling

MLP – Multi-Layer Perceptron

RBF – Radial Basis Function

RC – Reflectivity Coefficient

ROC – Receiver Operating Characteristic

SVM – Support Vector Machine

TL – Transfer Learning

TN – True Negative

TP – True Positive

1 Introduction

1.1 Motivations

The exploitation of oil and gas reservoirs is a process facing many uncertainties. Reservoirs are located kilometers under the ground surface, where the presence and quantity of resources can only be estimated. Complex geological structures react to resource extraction in ways that can cause severe environmental damages, as well as huge economic losses. The only available direct observation at such depth is through well drilling, a process extremely expensive, which only gives local information on areas that extend over several cubic kilometers.

Experts thus extensively rely on indirect methods to assess information on the sub-surface. Among them, seismic reflection allows building images covering the entire volume of interest, in which the important geological features, called here geobodies, can be interpreted. Horizons delimit different types of rocks; faults can block flow or on the contrary be preferential fluid paths, breaking the continuity of horizons; salt domes and channels can also be important to locate hydrocarbon traps.

The manual interpretation of geobodies in seismic data is a tedious task. In the past decades, the amount of data have increased from hundreds of megabytes to hundreds of gigabytes and seldom even terabytes today (Wang *et al*, 2018). The majority of recent seismic surveys are three-dimensional, comprising hundreds of seismic images organized in seismic cubes. For that reason, computational tools that allow automating or even assisting interpretation are of great value in the industry. Such tools face challenges related to the quality of the processed seismic signal and the complexity of the geological structures involved (Figueiredo, 2007).

The finality of the interpretation step is to build numerical models of the sub-surface, in which geobodies are represented as separation lines in 2D and surfaces in 3D models. Numerical simulations in those models aim at understanding the mechanical, hydraulic and thermal behavior of rocks in different scenarios, helping experts to take strategic decisions for production.

One important decision is, for example, the number and location of injection and production wells. Simulations are carried out using numerical methods that require the subdivision of the domains in small elements, forming a mesh over the interest area. Methods like the Finite Element Method imply a set of constraints on the mesh generation, including restrictions on the number of elements, their arrangement and their quality.

Data interpretation and mesh construction are thus two crucial steps for reservoir characterization. The accuracy of both processes as respect to the in-place geobodies is essential to reduce environmental and economical risks during reservoir production, overall allowing for a safe reservoir exploitation. In this thesis we propose a work in each of these two fields.

1.2 Goals

The first study presented in this work focuses on seismic faults. We investigate the use of Convolutional Neural Networks (CNNs) for automatic interpretation of fault location. CNN is a deep learning method that has been recently growing in interest due to its high performances in a great variety of object detection tasks (Rawat *et al*, 2017; Zhiqiang and Jun, 2017). Works applying such technique recently achieved state-of-the art results for faults and salt domes detection (Wang *et al*, 2018). One of the obstacles of using this method in the seismic area is the difficulty of obtaining a significant number of well-interpreted data to train the networks. To face this issue, we propose to work with a synthetic dataset, however our final goal remains the classification of real data.

In a second work, we try to develop a mesh generator adapted to any geobody configuration, and suited for Finite Element Method simulations. Mesh generation remains until today a challenge in both 2D and 3D, especially for quadrilateral (resp. hexahedral) elements, which are usually preferred in the industry (Zienkiewicz *et al*, 2008). We tackle the problem in two-dimension and try to develop an automatic tool which produces meshes of good quality even in complex geological domains. For clarity, we note here that this work does not imply any method related to artificial intelligence.

Both works aim at reducing the manual effort for building models of the sub-surface, through process automation.

1.3 Contributions

The first study, on automatic seismic fault detection, presents an innovative method, where a CNN trained with synthetic seismic images is used to classify real seismic images through the use of Transfer Learning techniques. This method presents an interesting solution for two common problems researchers face when applying CNNs to seismic data: the obtention of a great number of well-annotated data for training, along with the generalizability of the method to different types of seismic data. This contribution is presented in two articles submitted in 2018: (Pochet *et al*, 2018b) and (Pochet *et al*, 2018a).

The second study, on all-quadrilateral mesh generation for FEM simulations, proposes an algorithm based on a new data structure, the *extended quadtree*. This new structure allows the division of a quadrilateral element into six quadrilaterals, giving more flexibility than the classical *quadtree* for the adaptation to the domain geobodies. The application of the technique is not bounded to geological domains and is also interesting for meshing curves representing human-made objects. We published an article to present this innovation: (Pochet *et al*, 2017).

1.4 Document layout

This thesis is divided into two parts.

Part I includes chapters 2 to 6 and investigates the use of CNNs for seismic fault detection in seismic images. We base our redaction on the content of two articles submitted in 2018: (Pochet *et al*, 2018b) and (Pochet *et al*, 2018a).

In Chapter 2, we present the concepts needed for the understanding of the proposed methodology. First, we introduce the basics of seismic imagery and describe the method we use to build synthetic seismic data. Second, we explain the principles of deep learning and the specificities of CNNs. Chapter 3 presents the related works: we focus on fault detection in seismic images, and do not treat other geobodies or other types of seismic data. In Chapter 4, we describe our methodology to build a fault classifier on synthetic seismic data, and show examples on test images where we apply a post-process to extract the exact fault locations. In Chapter 5, we present, test and discuss different strategies to apply the synthetic classifier to real data. Finally, Chapter 6 draws the conclusions of this first part and proposes potential lines of research for future works.

Part II includes chapters 7 to 12 and details an innovative algorithm for adaptive all-quadrilateral mesh generation suited for Finite Element Method simulations. It is based on the content of a published article: (Pochet *et al*, 2017).

Similarly to Part I, we begin by introducing important concepts in Chapter 7, describing some aspects of quadrilateral meshes and the data structures we use to develop our meshing method. Chapter 8 then presents an overview of the works related to quadrilateral mesh generation. Chapter 9 describes our extended quadtree, a new data structure at the core of our mesh construction method. Then, in Chapter 10, we detail the mesh generation algorithm. In Chapter 11, we present and discuss results on a set of models, and examine the performances and the limitations of the method. We also present an application of the algorithm as part of a software currently in development. We conclude on the method in Chapter 12, and give suggestions for further research.

Finally, we close this thesis with a few general words on our work in Chapter 13.

Part I

Seismic fault detection using Convolutional Neural Networks

2
Concepts

2.1
Seismic Data

Seismic reflection is a technique widely used in the oil and gas reservoir exploration industry. It is an indirect method that permits the construction of images of the sub-surface, based on the study of the seismic waves' travel time, frequency and waveform (Sheriff and Geldart, 1995; Goldner, 2014). Compared to direct methods like well drilling, seismic reflection allows to cover large volumetric areas at a lower cost. The acquisition and processing of the seismic signal in the interest area result in the visualization of the sub-surface where geologists and geophysicists can interpret the principal geobodies, such as faults, horizons, channels or salt domes, in order to further construct a 3D model of the exploration volume.

In this section we briefly describe the steps of seismic imagery in order to establish the notation used in this thesis and provide some understanding of the overall process in which our automatic fault detection method is inserted. For a deeper understanding we suggest (Robinson and Treitel, 2000; Gerhardt, 1998; Yilmaz and Doherty, 1987).

2.1.1
Acquisition

Seismic acquisition begins with the emission of elastic waves from a source that can be a dynamite explosion for terrestrial surveys or an airgun detonation when performed in the sea. A wave propagates with a velocity depending on the medium it is passing through. In fluids, such as air or water, there is only a *compression wave* that is an oscillatory movement in the direction of the wave propagation. In solids, there is also a *shear wave* that is an oscillation in the direction perpendicular to the propagation. Here we will consider only the compression wave which, for simplicity, will be referred as *seismic wave*.

At the interface between two different mediums, part of the seismic wave is reflected to the surface and is registered by a receiver, while the other part refracts and keeps on propagating further in the sub-surface. Figure 2.1 illustrates this process.

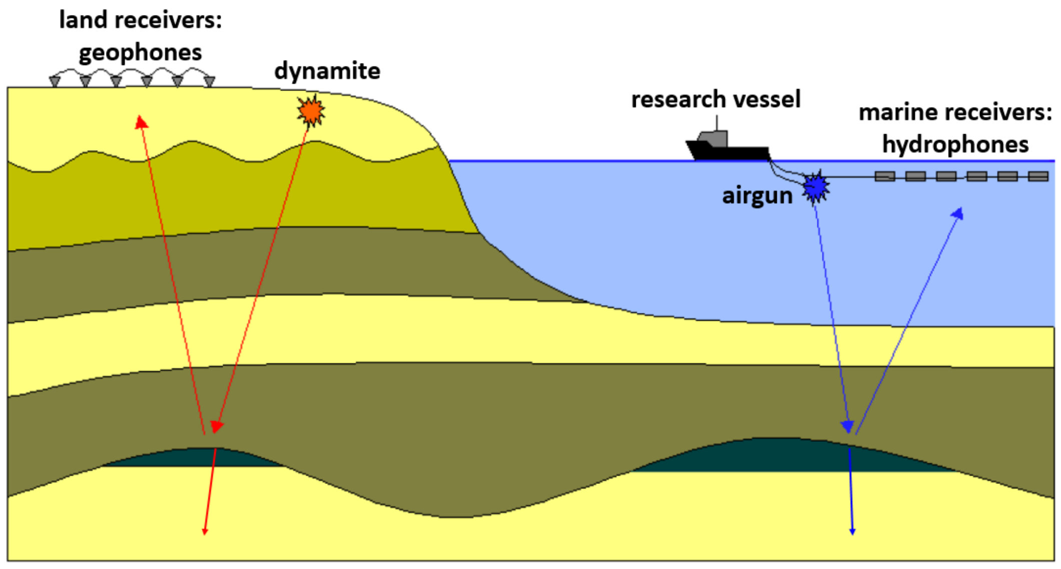


Figure 2.1: Seismic reflection acquisition. Figure adapted from (Gerhardt, 1998).

Receivers commonly register two quantities for each reflected wave: its *time of arrival*, which gives information on the position of the interface it reflected on, and its *intensity*, which gives the signal amplitude signature for this interface. This amplitude is referred here as the *seismic amplitude*, X.

As the wave hits an interface, the intensity of the reflected wave is proportional to the relative difference in *acoustic impedance*, Z, between the two types of rocks on each side of the interface. This is a property intrinsic to the rock that is defined as the product between its *density*, ρ, and the *velocity of the seismic wave* , V, through its medium:

$$Z = \rho V \tag{2-1}$$

In practice several explosions are performed, varying the position of the source and receivers. In 2D acquisitions, receivers are commonly positioned in a line with a fixed spacing between them, while in 3D acquisitions receivers are placed along parallel lines to create a grid, and each sensor can register a signal coming from various directions. The lateral and vertical resolution of the acquired data depends on the spacing between sensors and on the sampling frequency.

2.1.2 Processing

Signal processing aims at minimizing the signal noise and distortions, removing artifacts or unwanted seismic events in order to provide a visual result where the position of the geobodies is the most accurate as possible.

One important step in the process is the *seismic migration* that transforms recorded amplitudes to simulate an explosion where the source and receiver are at the same location. We can then consider that the signal propagated in the vertical direction and associate the reflected wave arrival time with depth. This vertical signal is called the *seismic trace*, and its maximal amplitude values (positive or negative peaks) correspond to the reflection events. It is important to this study to note that the actual value of the seismic amplitude is dependent of decisions made in the seismic migration step. The same data can be processed from different interpreters, yielding amplitudes in different scales. For this reason, when comparing different seismic images, it is better to put them in the same amplitude range. If the data is kept with float this may be values in the range between -1.0 and 1.0. If it is converted to a small integer (0 to 255) one must be carefull with the quantization procedure. The noise in the data may artificially increase the amplitude range, yielding many classes with no elements.

Figure 2.2(a) shows an example of a seismic trace. Its amplitude can be coded with color. By stacking seismic traces represented in a color scale along the acquisition lines, the seismic image appears. Figure 2.2(b) and 2.2(c) show the visual results for a 2D acquisition line (*seismic section*) and a 3D acquisition grid (*seismic cube*). All voxels of the grid (2D or 3D) contain a value of the trace amplitude, here painted with a grayscale. We assess the coordinates of each voxel through the in-line, cross-line and time (or equivalent depth) positions in the grid.

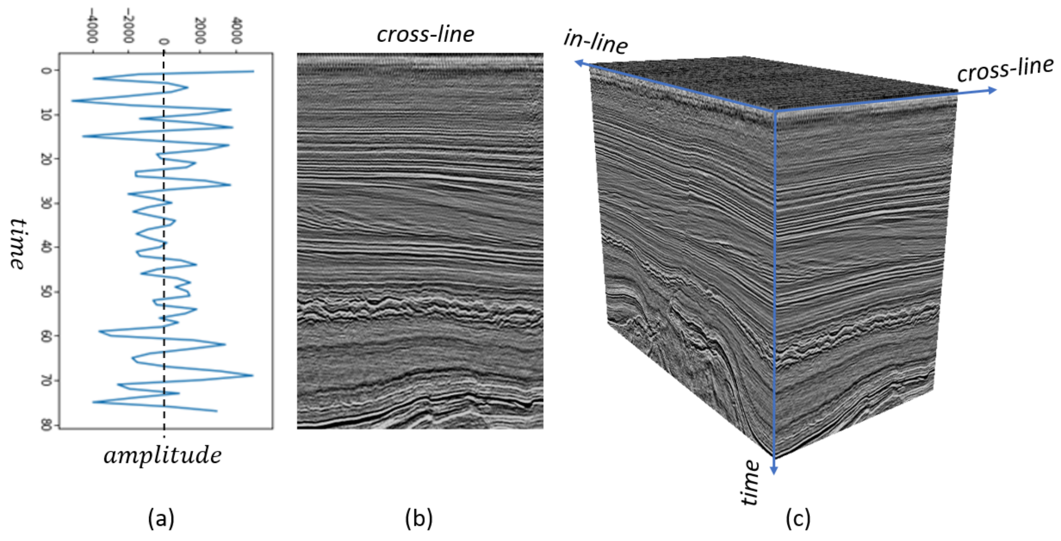


Figure 2.2: (a) Seismic trace; (b) seismic section; (c) seismic cube.

2.1.3 Interpretation

Theory on rock deformation and deposit behavior allows performing structural interpretation of the seismic image features. Figure 2.3 shows an example of the common organization of rocks and fluids around oil and gas reservoirs. Rock deformations are consequent to natural compression and traction forces applying in the Earth's crust, and can eventually result in break surfaces called *seismic faults*. Faults can trap the resources if filled with a sealing material, or on the contrary let them leak by opening a path through the surrounding rocks. The interface between rocks are called *seismic horizons*. As faults, sealing horizons can trap resources by blocking their natural upward propagation.

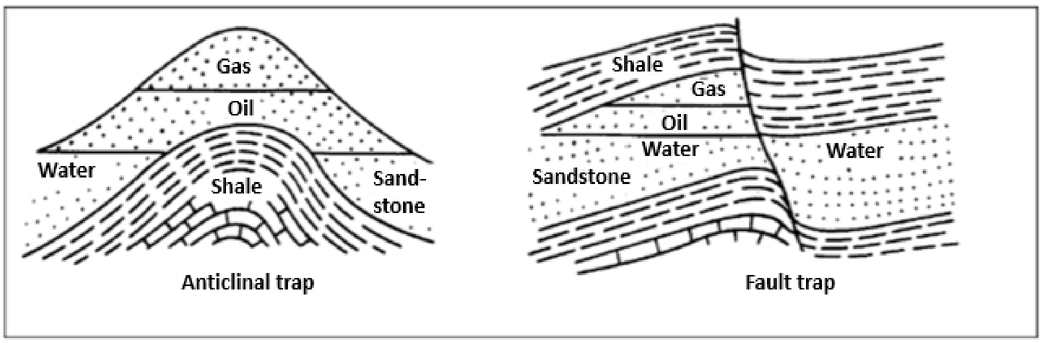


Figure 2.3: Common rock configuration around oil and gas reservoirs. Figure from (Robinson and Treitel, 2000).

In the seismic image, we can delineate horizons by following amplitude peak values of a same range in lateral traces (Sheriff, 2002; Goldner, 2014), since seismic waves reflected on a given interface with a fixed intensity. Faults then appear as a sudden loss in the lateral continuity of horizons. Figure 2.4 shows the interpretation of one horizon and two faults in a seismic image.

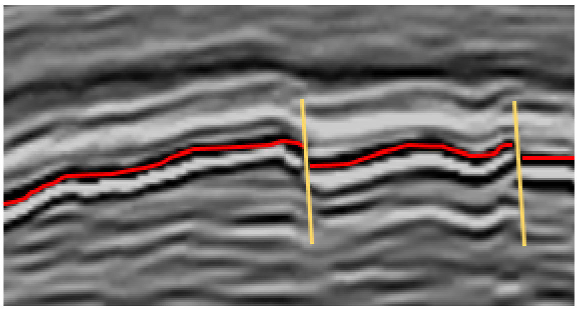


Figure 2.4: Simple interpretation of one horizon and two sub-vertical faults. Faults appear when the horizon's lateral continuity is lost.

As opposed to fractures, faults imply a relative displacement of the horizons on each side the breaking surface. They are generally sub-vertical, with a dip angle higher than 45° (Machado, 2008). They can curve with depth as shown in Figure 2.5. Fault scale goes from less than 1m to several km. They can branch to each other, organizing in networks. It is common to observe homogeneous orientations for groups of faults formed under the same mechanical conditions. Seismic surveys generally try to align acquisition lines perpendicular to the main faults' direction in order to make them appear clearly in the seismic sections.

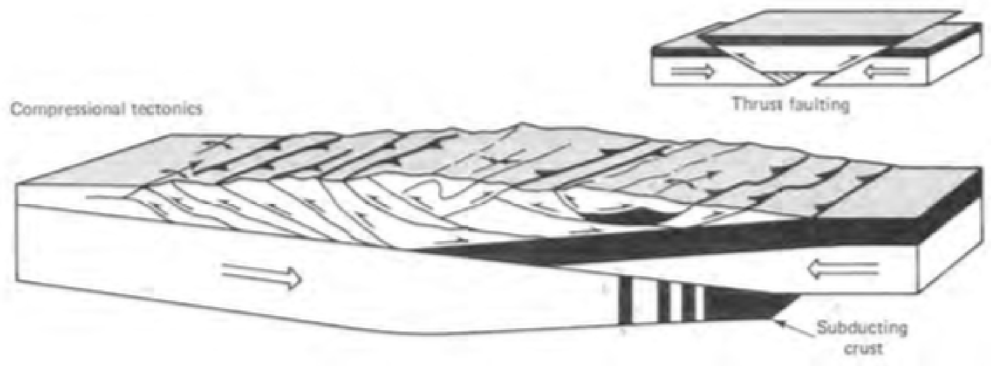


Figure 2.5: Typical fault network appearing in compressional conditions. Figure from (Suppe, 1985).

Because of noise and events location uncertainties in seismic images, a same input can result in many different interpretations: real seismic images have no ground truth.

2.1.4 Synthetic data

In our automatic fault detection method, part of the input are synthetic seismic data. We present here the basic concepts behind their construction.

There are two main possibilities when it comes to generating synthetic seismic images. The first approach is to simulate the seismic wave acquisition process: artificial source and receivers are placed upon a simplified sub-surface model made of rock layers with realistic properties, and the wave propagation is simulated using ray tracing (Fagin, 1991) or full waveform modeling (Hilterman, 1970; Kelly *et al*, 1976). The generated synthetic signal has to pass through all the seismic processing steps in order to produce a seismic section (or cube). Generating realistic seismic images with such technique requires substantial time and effort for both the acquisition simulation process and the seismic processing steps.

Consequently we consider here a second approach, which uses a simplified description of the seismic signal propagation process: the *convolutional model*. In this model, the reflection of the seismic wave at a rock interface is due to the contrast in acoustic impedance Z between the two mediums. This contrast is described through the *reflectivity coefficient*, RC:

$$RC_i = \frac{Z_i - Z_{i-1}}{Z_i + Z_{i+1}} \tag{2-2}$$

where i is the index of a rock in the stratigraphic column. This representation is only valid for a planar wave with normal incidence to the rock interface (Yilmaz and Doherty, 1987). The convolutional model then simply defines the seismic trace as the result of the convolution of a seismic impulse (*wavelet*, W) with the reflectivity coefficients:

$$Seismic\ trace = W * RC \tag{2-3}$$

It is common to simulate the seismic impulse as the *Ricker wavelet*:

$$W(t) = \left(1 - 2(\pi f_p t)^2\right) exp^{-(\pi f_p t)^2} \tag{2-4}$$

where f_p is the wavelet peak frequency. Figure 2.6 illustrates the whole process.

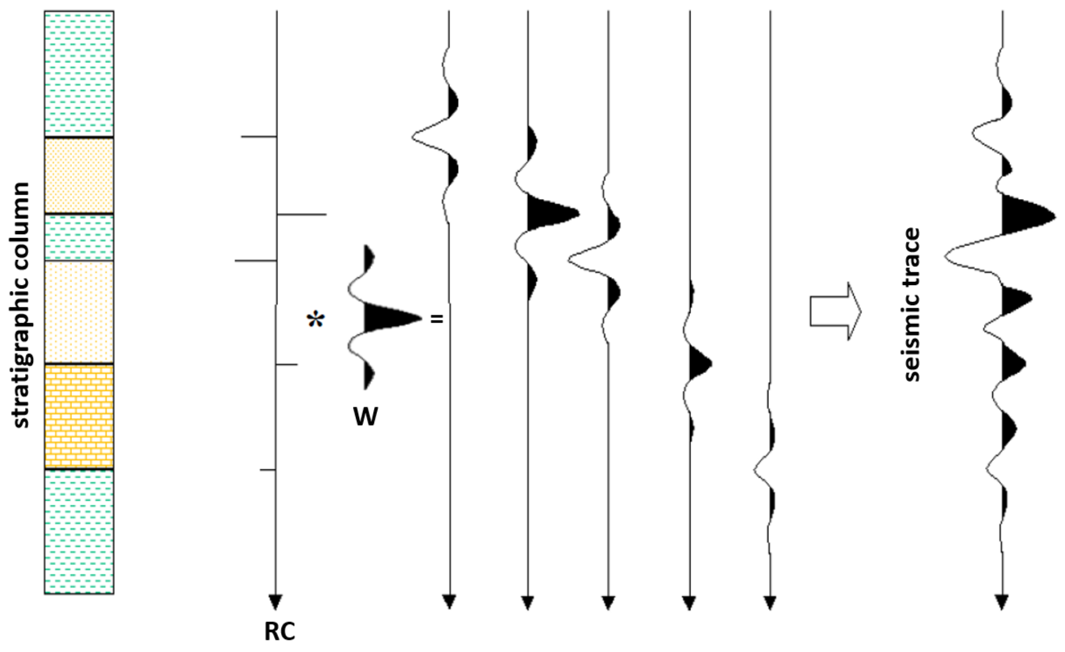


Figure 2.6: Convolutional model. Figure adapted from (Gerhardt, 1998).

Using this simple theory, Hale (2014) offers an open source code, *IPF* (Seismic Image Processing for Geological Faults), which allows the easy creation of synthetic seismic images. This tool first generates a random sequence of reflectivity coefficients RC. In the real world, rock layers present small variations in RC because rock properties are not perfectly homogeneous inside the layer. To mimic this effect, IPF adds a contrast parameter in the random RC function. This way, higher peaks can be seen as real horizons while smaller peaks are considered as RC noise:

$$f_{RC}(t) = [2(rand_{01} - 0.5)]^n \tag{2-5}$$

with f_{RC} the random reflectivity function, $rand_{01}$ a uniform random number between 0 and 1, and n the contrast parameter. Figure 2.7 shows an example of such signal and its corresponding amplitude map.

The RC column is then copied laterally to obtain a two-dimensional RC amplitude map representing horizontal rock layers, as shown in Figure 2.8.

Simple geometrical deformations are then applied to reproduce the effects of rock *shearing*, *folding* and eventually *faulting*. For each deformation type, the first amplitude column at position $x = 0$ is set as reference and all values indexed along t, from 0 to the number of samples in depth. Indices in the section are then updated according to the deformation type and final amplitude values are computed through interpolation, given the reference values and the

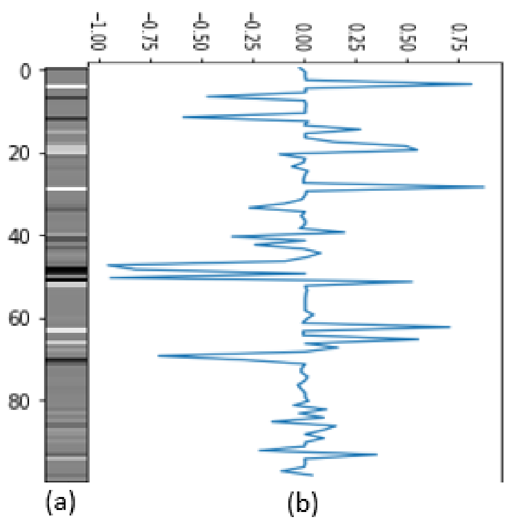


Figure 2.7: Random reflectivity for $n = 5$, with 100 samples. (a) discrete signal, corresponding to the amplitude map of the reflectivity signal, painted in grey scale from black = -1 to white = 1. (b) Corresponding continuous signal.

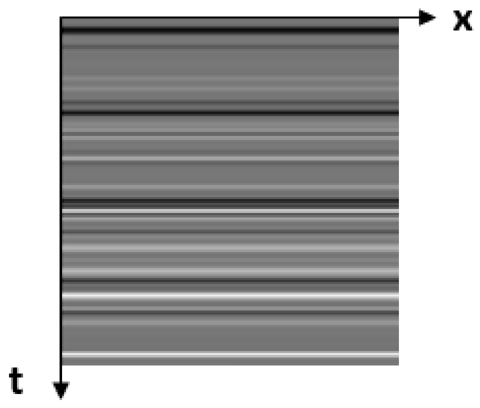


Figure 2.8: Flat section of size 100x100.

updated indices.

Shearing

Shearing performs a translation of amplitude values along the vertical direction, given the following index update:

$$t = t - (ax + b) \tag{2-6}$$

with a and b constant parameters. This vertical translation is called the throw, hence shearing applies a linear throw along x in the section. Figure 2.9 shows an example of applying shearing on the flat section.

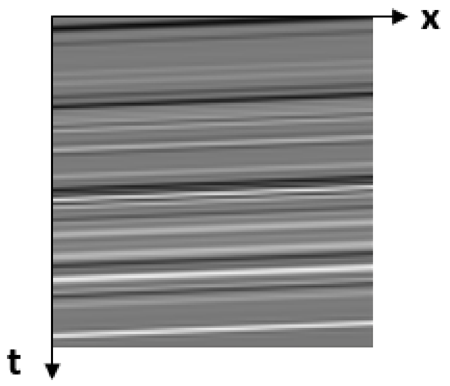


Figure 2.9: Shearing with $a = -0.05$ and $b = 0$.

Folding

Horizons' folding is performed using a squared sinus function:

$$t = t - f(t)sin(f(x)x) \tag{2-7}$$

where f is a linear function $f(k) = ak + b$. Here, $f(t)$ represents the variation in <u>*folding amplitude*</u> with depth, and $f(x)$ describes the variation in <u>*folding frequency*</u> along direction x. Figure 2.10 shows an example of applying folding on the flat section.

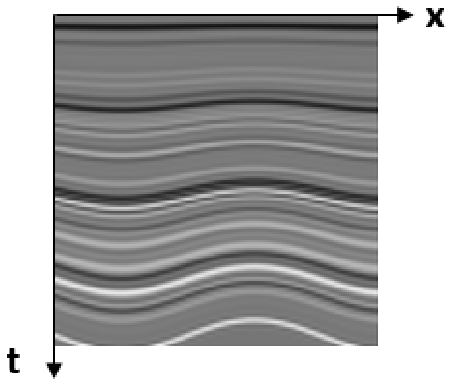


Figure 2.10: Folding with $a = 0.05$ and $b = 0$ for folding amplitude and $a = 0$ and $b = 0.1$ for folding frequency.

Faulting

IPF allows describing a fault as a straight line with a certain angle θ from the vertical. The general equation of a plane gives us the set of points (x, t) lying on the fault:

$$xn_x + tn_t + d = 0 \tag{2-8}$$

with n_x, n_t the two components of the fault's normal vector, and d the distance to the origin. Using simple trigonometry we can find the expression of the normal $\vec{n}$: Figure 2.11 shows the relation between angle θ and normal $\vec{n}$.

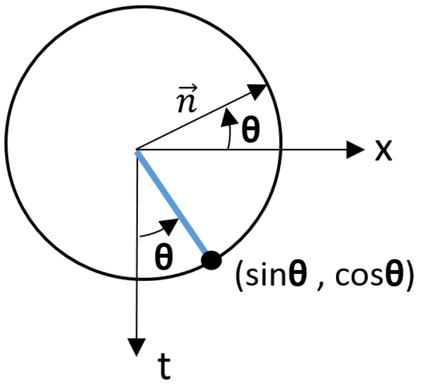


Figure 2.11: Relation between fault angle and normal on the unit circle. Thick blue line is the fault.

We can deduce that:

$$\vec{n} = (cos\theta, -sin\theta) \tag{2-9}$$

From Equation 2-8, we need one point (x, t) on the fault to obtain the distance to the origin, d. This point is an input parameter of the faulting process. To perform the index updates, the code first compares Equation 2-8 to 0: if the result is positive, the point is on the right side of the fault, and a throw is applied, constant along x but linear along t:

$$throw_{fault} = at + b \tag{2-10}$$

with a and b constant parameters. Figure 2.12 shows an example of applying faulting on the flat section.

After all deformations were performed, the synthetic sub-surface structure appears and the convolutional model can be applied. In order to maintain the validity of the convolution operation, the code keeps track of the horizons' normal along the deformation process. This way, the convolution can be performed considering the normal incidence of the synthetic seismic signal.

Finally, noise is added to make the resulting image more realistic. The amount of noise is controlled through a constant C which modifies the *signal-to-noise ratio* (SNR):

$$SI_{noise} = SI + I_{noise}SNR(SI, I_{noise})C \tag{2-11}$$

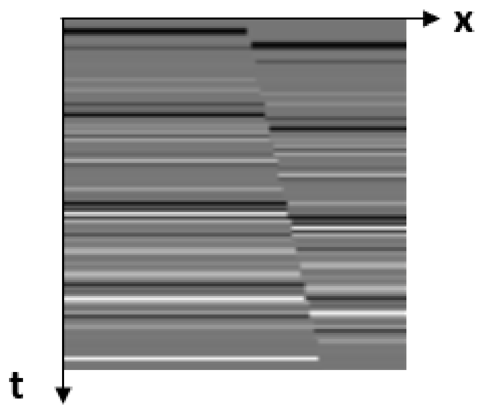


Figure 2.12: Faulting with reference point (50, 0), $\theta = 10°$, and throw parameters $a = 0$, $b = 4$.

where SI is the seismic image and I_{noise} is an image of equal dimension containing random values of high frequency between -1 and 1. SNR represents a way to scale the noise values with the image values, and is computed using the *root mean square* (RMS) of both images. Given a signal f with n values, the RMS is defined as:

$$RMS(f(x)) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \tag{2-12}$$

and the SNR is:

$$SNR(SI, I_{noise}) = \frac{RMS(SI)}{RMS(I_{noise})} \tag{2-13}$$

Figure 2.13 shows an example of the final image. In this synthetic section we applied the shearing, folding and faulting of Figures 2.9, 2.10 and 2.12, a Ricker wavelet with peak 0.5 and noise with $C = 0.5$.

This method presents the great advantage of being simple and fast. As every step of the process is parameterizable, we can create many seismic images automatically in few minutes. However, generated images only present simple configurations: faults are straight lines crossing the image entirely, it is difficult to generate more than a few faults per image, and other geobodies like channels or salt domes cannot be created. The code also lacks in documentation.

However, deep learning techniques used in this thesis for automatic seismic fault detection require large amounts of data where the position of the faults is known a priori: IPF remains thus a powerful tool to use with such methods, which concepts are presented in the next section.

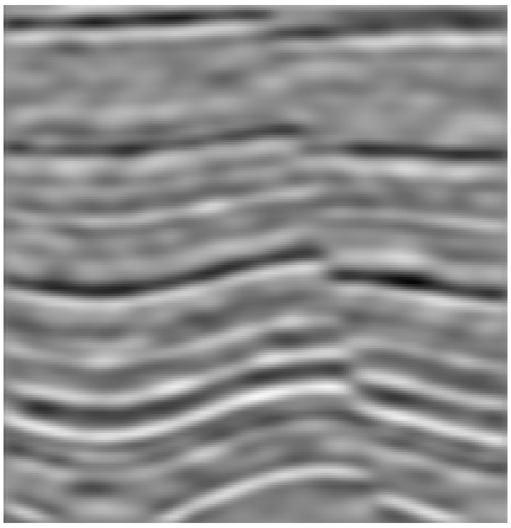


Figure 2.13: Example of a synthetic seismic image generated with IPF.

2.2
Convolutional Neural Networks

Convolutional neural networks (CNNs) are powerful mathematical models suited for image classification. In this work, we use CNNs to perform automatic fault detection in seismic images, for that reason we present here their basic concepts and features. We first present artificial neural networks' general principles, then explain the learning process, before introducing the peculiarities of CNNs and presenting the method to evaluate the model's performance.

2.2.1
Artificial neural networks

Artificial neural networks (ANNs) are biologically inspired computational models, which try to mimic processes occurring in animal brains for learning specific tasks. The first neural network was developed by McCulloch and Pitts (1943), motivated by the observation that brains perform better than computers at a couple of tasks (Haykin, 2009). This remains true today for object recognition. The field however really gained in momentum in the 1980's after researchers managed to overcome some of the main limitations of the technique (notably Rumelhart *et al* (1986) as a response to Minsky and Papert (1972) critics). Since then, the increase in computer power allowed building deep neural networks that can fully exploit the method's potential.

The base unit of an ANN is the *artificial neuron*. It is a function that receives a set of inputs, performs a weighted sum on them and produces an output, according to an *activation function*, σ:

$$f(x) = \sigma(\sum_{i=1}^{n} x_i w_i + b) \tag{2-14}$$

with x_i the input i, w_i the weight associated with x_i, n the number of inputs and b a bias term.

Activation functions perform a type of thresholding on the neuron's output. Figure 2.14 shows some popular non-linear activation functions. The Rectified Linear Unit (ReLU) is preferred in many applications since it can accelerate the learning process by up to 6 times compared to other activation functions (Krizhevsky *et al*, 2012). ReLU also avoids problems with vanishing gradient and promotes model sparsity (Glorot *et al*, 2011). It is also possible to use the simple linear activation function $f(x) = x$. However complex learning tasks can´t be achieved without non-linearity representations.

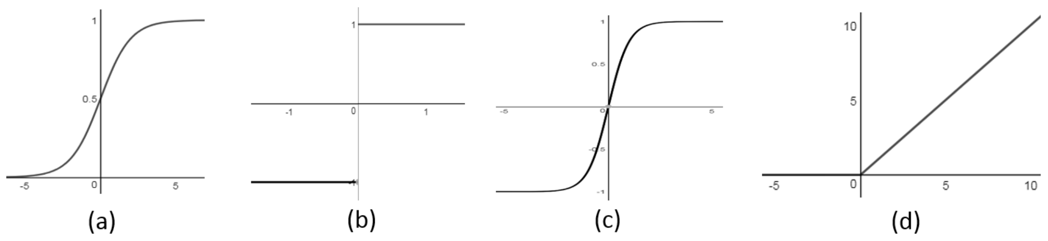


Figure 2.14: Activation functions. (a) sigmoid; (b) signal; (c) hyperbolic; (d) ReLU.

If it is possible to implement learning algorithms with one single neuron (Rosenblatt, 1957), achieving complex learning tasks requires organizing neurons in networks. ANNs are systems where neurons connect to each other such that the output of a set of neurons serves as input to the next neuron, according to given <u>*connection weights*</u>, w. For simplicity, we often describe the bias term as an input neuron of value 1, with connection weight w_0 (Figure 2.15).

The learning process then aims at updating connection weights in order to minimize the error between the network's prediction and the expected ground truth. We detail this process in the next section.

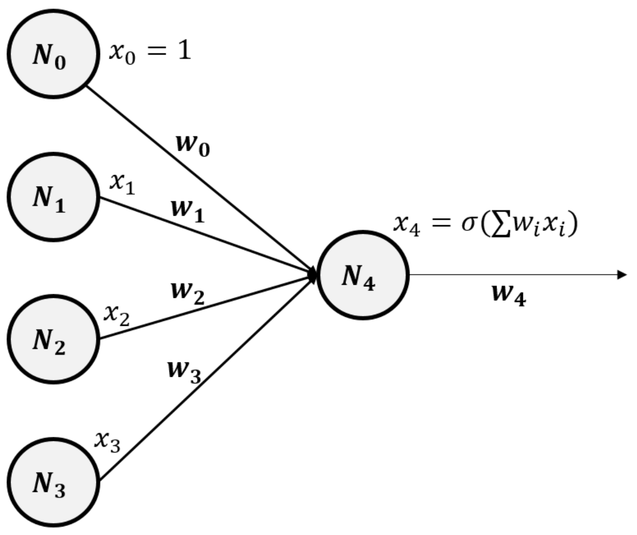


Figure 2.15: Structure of an artificial neural network. N_i are neurons, x_i their output, w_i the weight associated to the connections and σ the activation function.

2.2.2 Learning mechanism

Let us consider a common ANN, the *Multilayer Perceptron* (MLP). As shown in Figure 2.16, MLP are composed of three kinds of neuron layers: an *input layer*, which contains the input signal components, one or more *hidden layers*, and the final *output layer* which gives the final classification. With the exception of the bias terms, each neuron in a layer receives as input signal the output of all the neurons of the previous layer, so that the network is *fully connected*. In classification problems, the output layer contains a number of neurons equal to the number of classes to predict.

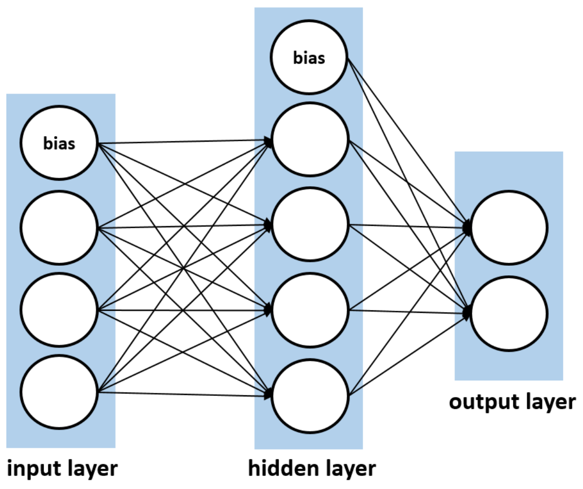


Figure 2.16: Structure of the Multilayer Perceptron.

Given a set of *training examples* $(x^{(i)}, y^{(i)})$, where $x^{(i)}$ represents the input signal and $y^{(i)}$ the expected output, the network performs classification in the following way.

First, all connection weights are initialized randomly. The input signal $x^{(i)}$ propagates forward in the network until reaching the output layer. The given output, called the *prediction*, is compared to the expected *ground truth* $y^{(i)}$ through an error calculated with a *loss function*. The error then propagates backward in the network and weight values are adjusted consequently. The network learns by repeating this process a number of times through all training examples, until reaching the desired minimal error. Connection weights are hence called *learnable parameters*, as opposed to a set of *fixed parameters* which describe the learning algorithm.

To explain those steps more in details, let us introduce some notations:

- i is the training example index, with N_i the total number of examples;
- c denotes the index of a neuron in the current layer, with N_c the number of neurons in that layer;
- p denotes the index of a neuron in the previous layer, with N_p the number of neurons in that layer;
- w_{ab} is the weight connection between neurons a and b. Bias terms have index 0.
- as the output of a layer is the input of its next layer, we denote by x the output of a neuron.

For the input layer, output $x_c^{(i)}$ are the components of the input signal. For the output layer:

$$x_c^{(i)} = \hat{y}_c^{(i)} \tag{2-15}$$

with $\hat{}$ denoting the prediction. For the intermediate layers, Figure 2.15 shows:

$$x_c^{(i)} = \sigma(\sum_{p=0}^{N_p} x_p^{(i)} w_{pc}) \tag{2-16}$$

Loss function

In this work we use the popular *cross-entropy* loss. To calculate it, we set the last layer activation function as the *Softmax* function, which outputs a normalized probability for each class (hence for each output neuron):

$$x_c^{(i)} = \hat{y}_c^{(i)} = \frac{exp^{\sum_{p=0}^{N_p} x_p^{(i)} w_{pc}}}{\sum_{k=1}^{N_c} exp^{\sum_{p=0}^{N_p} x_p^{(i)} w_{pk}}} \in [0,1] \tag{2-17}$$

The cross-entropy loss for training example i is then computed as follows:

$$L^{(i)} = -\sum_{k=1}^{N_c} y_k^{(i)} log(\hat{y}_k^{(i)}) \tag{2-18}$$

with $y^{(i)}$ the ground truth vector. The cross-entropy loss for a set of k examples is assessed through the mean:

$$L = \frac{1}{N_k} \sum_{i=1}^{N_k} L^{(i)} \tag{2-19}$$

Weight update

Updates of connection weights aim at minimizing the loss value. The common method to perform this optimization in neural networks is to use the *Gradient descent* algorithm coupled with *backpropagation*.

Gradient descent exploits the fact that a function's gradient reaches 0 at its optima. The gradient of a function is the vector of its partial derivatives. Given our loss function:

$$grad\ L^{(i)} = \left\{ \frac{\partial L^{(i)}}{\partial w_0}, \frac{\partial L^{(i)}}{\partial w_1}, ... \frac{\partial L^{(i)}}{\partial w_{N_w}} \right\} \tag{2-20}$$

with N_w the total number of weights in the network, initialized with random values. Gradient descent tries to reach the null gradient by taking small negative steps for all weights j:

$$w_j = w_j - \alpha \left(\frac{\partial L^{(i)}}{\partial w_j} \right) \tag{2-21}$$

with α the *learning rate*, a fixed parameter controlling the convergence speed. Negative steps ensure that the method converges to a minimum. If the loss function is convex (which is desirable), the gradient will find the unique (hence global) minimum of the function.

One problem with neural networks is that the loss of training example i is computed in the last layer, knowing only the output of its previous layer: it is not straightforward to access to the derivatives for all connection weights. The backpropagation method resolves this issue through the computation of neurons' local error δ_c, which can be seen as the contribution of each neuron to the global loss, and defined as:

$$\delta_c = \frac{\partial L^{(i)}}{\partial(\sum_{p=0}^{N_p} x_p^{(i)} w_{pc})} \tag{2-22}$$

Details on the computation of δ_c for each neuron can be found in (Haykin, 2009). We note here that computing such local error in layer l involves knowing local errors of all neurons in layer $(l + 1)$, so that the errors propagate layer by layer, backward in the network. Given the local error of a neuron in the current layer, the derivative of the loss function regarding this neuron's input weights can be computed as :

$$\frac{\partial L^{(i)}}{\partial w_{pc}} = \delta_c x_p^{(i)} \tag{2-23}$$

Backpropagation hence allows using the gradient descent algorithm in any neural network. In practice, weights are updated only after a set of training examples have passed through the network (forward and backward). These sets are called *batches*, and the update occurring after each batch is called an *iteration*. With this method, one iteration only estimates the gradient on a subset of training examples. As batches do not represent the entire dataset and can contain a varying number of data outliers, this gradient estimate may provide poor gradient steps. Adding a *momentum* term in the gradient update helps preventing this effect (Rumelhart *et al*, 1986; Qian, 1999). When all training examples have passed through the network once, an *epoch* was concluded. For example, a dataset of 2000 training examples with a batch size of 500 would result in 4 iterations per epoch. The process reaches convergence after several epochs. Batch size and number of epochs are fixed parameters.

Deep learning

The *depth* of a network is related to the number of non-linear operations it performs (Bengio, 2009). Deep networks consequently contain more neurons and more layers than *shallow* networks. The strength of neural networks is to extract new information dynamically during training through the use of the hidden layers. Each neuron corresponds to a *feature*, which can be seen as a puzzle piece used for building class discrimination. In neural networks, shallow features contain general information, while deep features are insights specific to the task. For example, in image classification, a shallow feature would allow recognizing edges, while a deep feature would hold for example the information of a particular stripe pattern on an animal's back.

The capacity of deep neural networks to dynamically learn features at different levels of abstraction allows the network to map complex inputs to the output directly from the data, without the need to manually select the features, as it would be the case with other learning techniques. This is particularly interesting for data with a high level of abstraction, like images, where it is difficult to specify features in terms of raw sensory input (Bengio, 2009).

Deep networks are thus better than shallow networks at differentiating classes. However, they can be difficult to train, as they are prone to *overfitting*. When the number of parameters in the network is high compared to the number of training data, some features will capture the dataset noise instead of meaningful task-related information (Srivastava *et al*, 2014). The training loss will be low, but the classifier will perform poorly when applied to new data: the network has a bad *generalization capacity*. A popular technique to avoid overfitting is the use of *dropout* (Srivastava *et al*, 2014). It consists in training different architectures of the network for each training example, by randomly "dropping" some neurons: the neuron and its input and output connections are temporarily removed from the network. This way, the training is averaging the predictions of different networks. Figure 2.17 shows an example of applying dropout on an MLP.

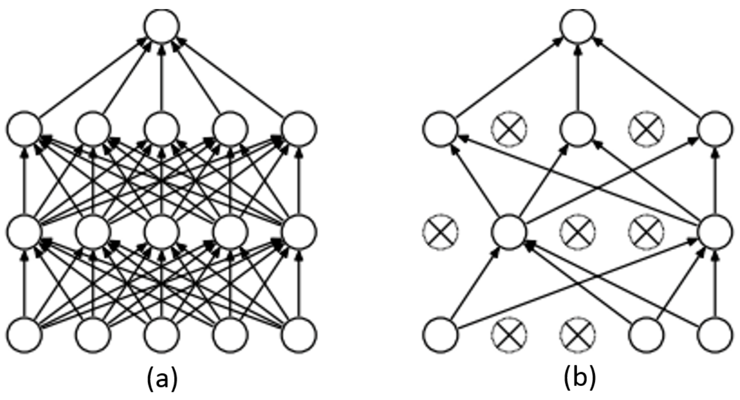


Figure 2.17: Applying dropout to a MLP. (a) A standard MLP with 2 hidden layers. (b) An example of dropout: cross units have been dropped. Figure from (Srivastava *et al*, 2014).

Transfer learning

We introduce here an interesting learning method which we use in our automatic fault detection process when training with real seismic data.

Transfer learning(TL) is a technique aiming at adapting a trained network to a different classification task, by using all or part of its learned features to train a new network. Here again, TL researches are based on the observation of a biological process: the presence of a knowledge learned previously can help solving new problems faster and / or better (Pan and Yang, 2010). For example, it may be simpler to learn playing the piano if we already know how to play the electric organ. In the neural network context, this translates as follows: the connection weights in the new network are not initialized randomly but come from a first training process. This initial network is called the *pre-trained model*. It can be any network trained on any classification task, but the method is more efficient if the two networks share a *similar* task.

In practice, there are many ways to apply TL, depending on each particular situation. First, part of the weights of the pre-trained model can be *frozen*, meaning that they will not be updated in the new training session. This makes sense if we consider that some features (neurons) already hold relevant information for the current classification task. The more similar the tasks are, the more pre-trained model weights can be frozen. Second, it is possible to use only a part of the pre-trained model's *architecture*: some layers can be removed, new ones can be added. This is particularly interesting when the pre-trained model is deep but the new network dataset is small. In this case some pre-trained model layers are removed to avoid overfitting.

TL is hence particularly suited for building powerful classifiers with small datasets. In addition, the drop in number of learnable parameters due to the freezing of parts of the network weights allows for fast training. Transfer learning is often used with CNNs, as the different layers of such networks have a rather good interpretability.

2.2.3 CNN architecture

CNNs are a type of ANN where the layers' organization is adapted to image classification. Images are basically a set of pixels organized in a matrix. This matrix is defined by its height, width and number of channels (for example channels R, G, B of colored images). In terms of neural networks, we can thus see the image as an input signal where all pixel values represent an input neuron. Using such input in the MLP structure would lead to a very large amount of learnable parameters. For example, let us consider the synthetic seismic image created in section 2.1.4, Figure 2.13. Its height and width is 100, and it has only one channel, the seismic amplitude. Considering a very simple network with only one hidden layer of 10 neurons and an output layer of two neurons (a binary class which indicates if the image contains a fault, or not), the number of learnable weights in the network would be: 100 x 100 x 10 + 10 x 2 = 100 020. In practice, we want to train with deeper networks.

CNNs exploits the spatial organization of the image to reduce the number of parameters while efficiently extracting relevant features. It does so through a sequence of specific layers illustrated in Figure 2.18.

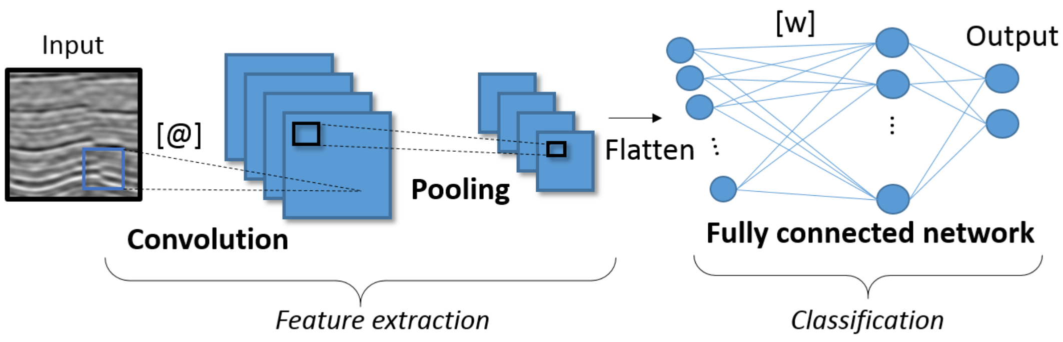
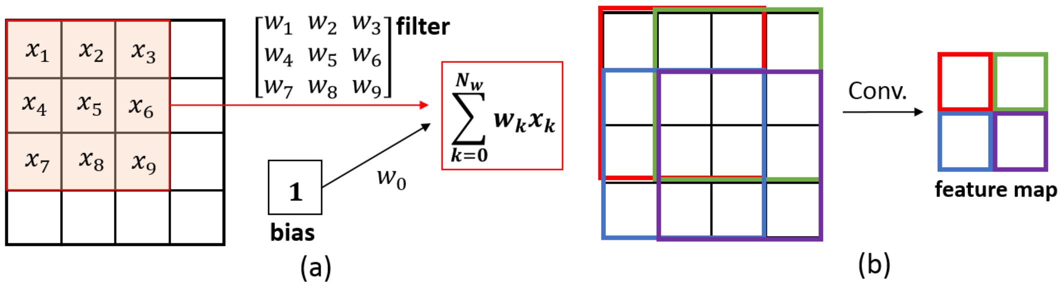


Figure 2.18: Architecture of a CNN, with: [@] = learnable filters; [w] = learnable weights. The example input is a seismic image, the output could be a binary class with value 1 if the image contains a seismic fault, 0 otherwise.

The first part of the network performs feature extraction through a series of *convolution* and *pooling* layers. In this part, layers are organized in *feature maps*, which are sets of neurons that share sets of weights called *filters*. Neurons in feature maps are connected to only a small region of the previous layer. Layer outputs are here again controlled by activation functions. At the end of the feature extraction part, neurons are flatten into a single vector and serve as input to a fully connected network. Here we describe in details each layer for the case of two-dimensional images (i.e with only one channel), which are the kind of images used in this thesis.

Convolutional layers

Convolutional layers are at the core of the feature extraction process. The convolution operation computes a weighted sum on a small region of the input neurons, as shown in Figure 2.19(a). The weight matrix, called filter, is applied throughout the input at different locations and outputs a feature map, where neurons' location roughly corresponds to their input region location (Figure 2.19(b)). Weights and bias are hence shared by all neurons of the feature map.



x_1 x_2 x_3
x_4 x_5 x_6
x_7 x_8 x_9
$\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix}$ filter
$\sum_{k=0}^{N_w} w_k x_k$
1
bias
w_0
(a)
Conv.
feature map
(b)

Figure 2.19: Convolutional layer. (a) Convolution operation on one region of the input. (b) Correspondence between position of neurons in the output feature map and position of input regions.

Convolutional filters are applied in the input in a sliding scheme. In this thesis, we apply filters on all possible pixels: filter *stride* is 1 (filter slides one pixel at a time), and there is no *padding* (a filter cannot have weights outside the input). This leads to a reduction of the size of the feature map compared to its input.

A convolutional layer is composed of n feature maps, each applying a filter of fixed size but different weights. All filters' weights are learnable parameters, while the number of filters and their size are fixed.

Compared to the MLP fully connected architecture, convolutions allow a drastic reduction of the number of learnable parameters. Taking as example the simple 4x4 input of Figure 2.19, we see that one filter of size 3x3 outputs 4 neurons. The number of learnable parameters involved is (3x3 + 1) = 10. In a fully connected architecture, all inputs are connected to all outputs, so that this number reaches (4x4x4 + 1) = 65. Moreover, with the convolutional architecture, increasing the size of the input and thus the size of the output does not change the number of learnable parameters involved: only the size of the filter matters. Simonyan and Zisserman (2014) showed that 3x3 filters are sufficient when training deep CNNs. As discussed before, the drop in number of learnable parameters is beneficial for computational efficiency and also helps preventing overfitting.

Pooling layers

Pooling layers perform downsampling of the feature maps. In this thesis we use the classic *max pooling* operation, which slides a filter on the input and keeps only the highest value within the filter (Figure 2.20).

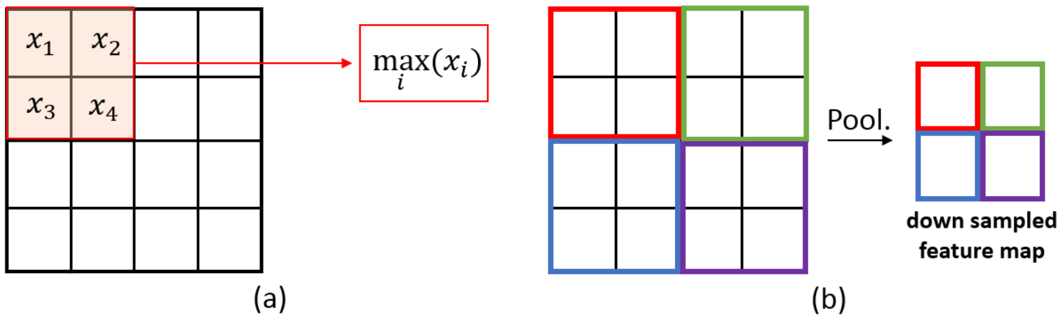


Figure 2.20: Pooling Layer. (a) Pooling operation on one region of the input. (b) Correspondence between position of neurons in the output feature map and position of input regions.

The idea behind max pooling is that important features will hold a high value after activation. We can thus discard neighboring features of lower value. Moreover, it suggests that the most valuable information is not the exact position of the features in the map, but rather their relative position. The pooling operation does not involve any learnable weight. Fixed parameters are the size and stride of the filters. In this work we use a classic configuration: 2x2 filters with stride 2, which reduce the size of the input feature map by 2 as illustrated in Figure 2.20(b). This operation reduces the spatial representation without losing important information, this leading to a global reduction of the network's learnable parameters.

Fully connected layers

The fully connected layers represent a classical MLP. In CNNs, this part is used exclusively for classification, as feature extraction was performed by the sequence of convolutions and pooling. Consequently, there is no need for deep networks and the number of layers can be low.

CNN with Support Vector Machine

As the CNN classification part does not need to involve deep learning, another popular option is to use the *Support Vector Machine* (SVM) classifier (Cortes and Vapnik, 1995), which is not a deep learning technique but often proved better at differentiating classes if the set of input features are well-designed (Tang, 2013).

SVM separates classes in a way that maximizes the confidence in a new prediction. Given two classes, there exist multiple hyperplanes which separate them correctly, as illustrated in 2.21(a). We see that moving slightly an hyperplane position can easily change the classification of the samples that are close to it. Intuitively we then understand that we can give more credit to a sample which is far from the decision line, than to a sample which is close to it. SVM aims at finding the hyperplane which maximizes the distance between the classes, as shown in Figure 2.21(b). This distance is called the *margin*. We detail here the mechanics of the SVM, beginning with the simple case of linearly separable classes, then explain how to adapt it to find non-linear solutions.

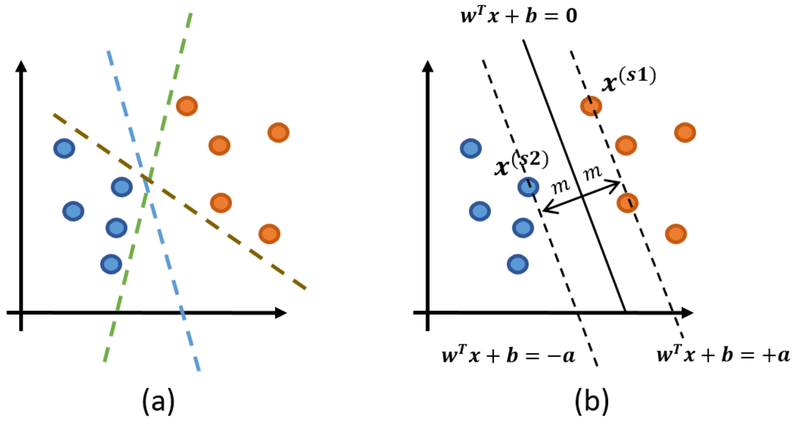


Figure 2.21: Separating two classes in 2-dimension. (a) Example of 3 hyperplanes correctly separating the two classes. (b) Best hyperplane maximizes the margin 2m.

We consider the case of two classes defined by the labels -1 and 1. The separation hyperplane is defined by the equation:

$$w^T x + b = 0 \tag{2-24}$$

where the normal vector w contains learnable weights and b is the bias term, as defined before. If we consider the optimal hyperplane of Figure 2.21(b), its closest points $x^{(s1)}$ and $x^{(s2)}$ are equidistant on each side of it. They are each support of a parallel hyperplane of equation:

$$w^T x^{(s1)} + b = a \tag{2-25}$$

$$w^T x^{(s2)} + b = -a \tag{2-26}$$

with a some constant > 0. Now, let us also define the distance from the hyperplane to any point $x^{(i)}$ of the dataset:

$$d = \frac{|w^T x^{(i)} + b|}{||w||} \tag{2-27}$$

with $|\cdot|$ the modulus and $||\cdot||$ the norm. We can easily see that d is invariant to the scale of vector w. Indeed, multiplying w (and hence b which is w_0) by a constant does not change the above expression. Hence, we can choose to scale it by $w' = w/a$, since $a > 0$. From equation 2-25, 2-26 and 2-27, at points $x^{(s1)}$ and $x^{(s2)}$ the distance then becomes:

$$d = m = \frac{1}{||w'||} \tag{2-28}$$

$x^{(s1)}$ and $x^{(s2)}$ are called *support vectors*. Geometrically we see that support vectors are few compared to the number of data examples: candidates lay exclusively on the convex hull of each class point set. As support vectors are the points which are the closest to the hyperplane, no point can enter the margin, in other words:

$$w^T x^{(i)} + b \geq 1 \; if \; y^{(i)} = 1 \tag{2-29}$$

$$w^T x^{(i)} + b \leq -1 \; if \; y^{(i)} = -1 \tag{2-30}$$

Finally, our optimization problem is to maximize the margin, that is to say minimize $1/2m$. And as minimizing $||w'|| = ||w/a||$ is the same as minimizing $||w||$, we can write:

$$\begin{aligned} &\min_{w,b} \frac{1}{2}||w||^2 \\ &s.t. \; y^{(i)}(w^T x^{(i)} + b) \geq 1 \end{aligned} \tag{2-31}$$

This optimization problem is a quadratic problem with linear restrictions and it can be solved with classic optimization techniques. The single restriction expresses both equation 2-29 and 2-30. This optimization problem defines the *hard margin* SVM, since no point can enter the margin. In practice, data present outliers, points that do not respect the general tendency of their class. Softening the margin constraints can help taking into account this noise. Hence, the *soft margin* SVM allows some errors in the classification, letting some points enter the margin. To do so, it introduces slack variables in a regularization term, scaled by a constant C which controls the penalization of outliers.

In order to solve problems where the data is not linearly separable, Cortes and Vapnik (1995) proposed to map the input vector to a higher dimensional feature space in which the data becomes linearly separable. An example of such mapping is shown in Figure 2.22.

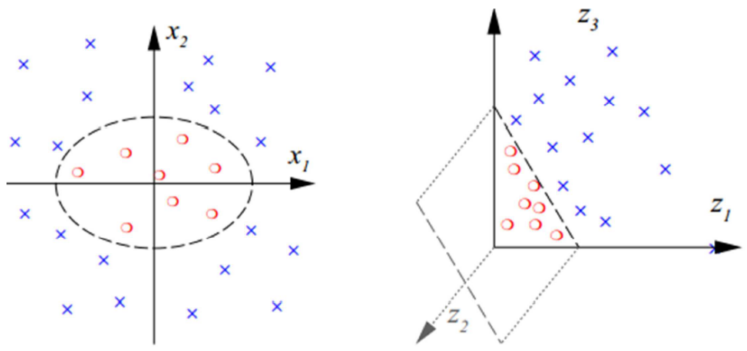
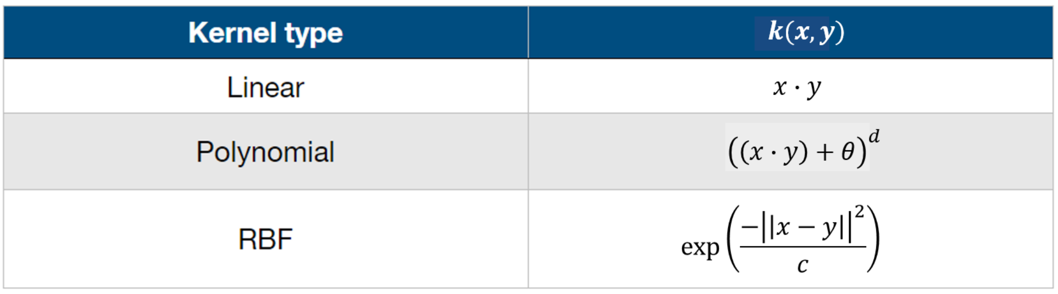


Figure 2.22: Example of a non-linear separation between classes in two dimensions (left). In the three-dimensional feature space $(x_1^2, \sqrt{2}x_1x_2\ , x_2^2)$, the classes become linearly separable (right). Figure from Müller *et al* (2001).

It is possible to explore different mappings efficiently, without explicitly defining the mapping function Φ, using *kernels*. Kernels compute inner products in high dimensionality without effort. In the example of Figure 2.22, taking two data points a and b in the initial feature space, we can write:

$$\begin{aligned}\Phi(a) \bullet \Phi(b) &= (x_{1,a}^2, \sqrt{2}x_{1,a}x_{2,a}, x_{2,a}^2)(x_{1,b}^2, \sqrt{2}x_{1,b}x_{2,b}, x_{2,b}^2)^T \\ &= x_{1,a}^2x_{1,b}^2 + 2x_{1,a}x_{2,a}x_{1,b}x_{2,b} + x_{2,a}^2x_{2,b}^2 \\ &= (x_{1,a}x_{1,b} + x_{2,a}x_{2,b})^2 \\ &= (ab^T)^2 := k(a,b)\end{aligned}$$

where k is here a polynomial kernel. In this thesis we use linear, polynomial and radial basis function kernels, describe in Table 2.1.



Kernel type	$k(x, y)$
Linear	$x \cdot y$
Polynomial	$((x \cdot y) + \theta)^d$
RBF	$\exp\left(\frac{-\|\|x-y\|\|^2}{c}\right)$

Table 2.1: Common kernels used in this thesis. $\theta \in \mathbb{R}, d \in \mathbb{N}, c \in \mathbb{R}$.

In order to apply kernels in SVM, it is necessary to use the dual form of the optimization problem, using the Lagrange multipliers. This form makes the optimization problem related to the data x through the inner product $x^{(i)}x^{(j)}$

only. SVM with *kernel trick* can then be defined as:

$$\max_{\alpha} \sum_{i=1}^{N_i} \alpha^{(i)} - \frac{1}{2} \sum_{i,j=1}^{N_i} \alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}k(x^{(i)}, x^{(j)})$$

$$s.t.\ 0 \leq \alpha^{(i)} \leq C\ and\ \sum_{i=1}^{N_i} \alpha^{(i)}y^{(i)} = 0 \tag{2-32}$$

$$with\ w = \sum_{i=1}^{N_i} \alpha^{(i)}y^{(i)}x^{(i)}\ and\ k(x^{(i)}, x^{(j)}) = \Phi(x^{(i)}) \bullet \Phi(x^{(j)})$$

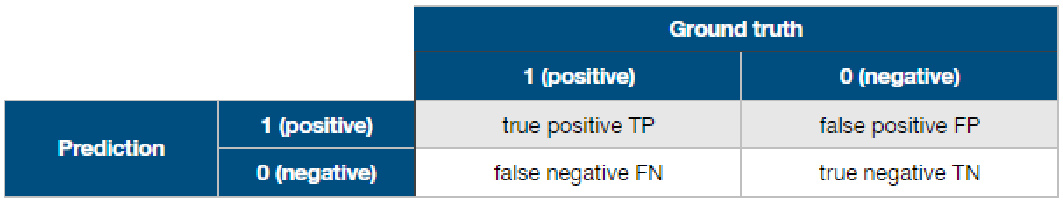
Using SVM with the kernel trick, complex classification tasks can be performed efficiently at the end of the CNN architecture.

2.2.4 Network's performance

When building neural networks, it is important to quantify the quality of the results in order to compare different parameterizations and / or architectures: quality metrics are essential to the network's *tuning*. In this section we present a set of metrics, then discuss some aspects of the proper way to perform dataset training and evaluation.

Quality metrics

We use a set of five metrics to evaluate the CNN results: accuracy, sensitivity, specificity, ROC AUC and F1-score. They each extract a different information from the *confusion matrix*, a table counting the number of true and false predictions for each class (Table 2.2).



		Ground truth	
		1 (positive)	0 (negative)
Prediction	1 (positive)	true positive TP	false positive FP
	0 (negative)	false negative FN	true negative TN

Table 2.2: Confusion matrix for a binary class problem.

Accuracy is a global quality metric that gives the total proportion of good predictions of the model.

$$accuracy = \frac{(TP + TN)}{(TP + FN + FP + TN)} \tag{2-33}$$

It allows quickly evaluating the model, as a high accuracy is desirable. However, it does not provide details on the performance of each class, consequently it should not be used alone.

Sensitivity is the probability for the model to predict class 1, knowing the ground truth is 1. This corresponds to the proportion of good predictions for the positive class:

$$sensitivity = \frac{TP}{(TP + FN)} = \frac{TP}{Number\ of\ positive\ samples} \tag{2-34}$$

A high sensitivity ensures the validity of negative predictions (FN are scarce). However, it does not guarantee that a positive prediction is true, because it does not take into account the FP samples. If the model predicts all samples as class 1, it will have a 100% sensitivity while being always wrong about class 0 samples.

Specificity is the counterpart of sensitivity, for the negative class. It corresponds to the probability to predict class 0 knowing that the ground truth is 0.

$$specificity = \frac{TN}{(TN + FP)} = \frac{TN}{Number\ of\ negative\ samples} \tag{2-35}$$

A combination of high sensitivity and high specificity for the model should ensure that both classes are predicted correctly. The ROC (Receiver Operating Characteristic) curve plots sensitivity against $(1 - specificity)$ and allows to quickly compare models (Figure 2.23). The ideal classifier would output 1 for each sensitivity and specificity, hence curves closer to that point represent better models. The diagonal of the graph represents a value of 0.5 for each metric, meaning the model did no better than chance. We can then assess the quality of a model numerically by computing the area under the curve, i.e the *ROC AUC*.

Finally, another interesting measure is the *F1-score*:

$$F1\text{-}score = \frac{2TP}{(2TP + FP + FN)} \tag{2-36}$$

This is a global measure of quality which, unlike accuracy, gives more importance to the results of the positive class. This is particularly interesting when the positive class is under represented. For example, consider seismic images: in this thesis, we develop a method in which seismic amplitude values are presented to a classifier, which outputs true if the values show the presence of a fault, false if no fault is detected. Consider a dataset of 10 faulted samples and 90 non-faulted samples. If the classifier outputs false for all samples: accuracy = 90%, F1-score = 0%. The F1-score thus emphasizes the error on

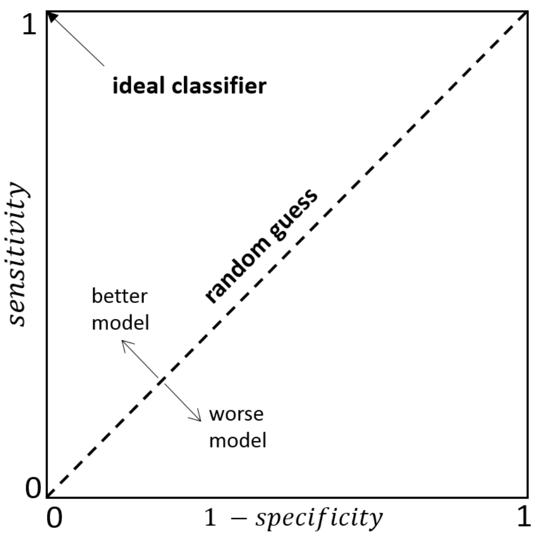


Figure 2.23: ROC space.

the positive class and allowed here to detect that the classifier was of very bad quality, since it detected no fault at all.

Training, validation and test

For good interpretability of the quality metrics, it is important to compute them on data that have not been seen by the network. Consequently, the dataset is divided into a *training set* and a *validation set*. The training set is used to train the network as described earlier, while the validation set serves specifically for metrics computation. Training and validation can occur many times before finding the configuration that gives the best results. Thus, there is a risk that the designed model overfits the validation data. To avoid this, a third set, the *test set*, is used only once at the end of the training, in order to test if the metrics are really representative.

In proportion, the training set should have the highest number of data, in order to learn complex relationships. The validation set should have enough data to represent well the entire dataset. The test set is generally the smallest.

Another important aspect to take into account when dividing the samples is the data *balance*. In the training set, all classes should be equally represented, while in the validation and test set the natural imbalance of the classes should be reproduced.

Finally, with small datasets it can be difficult to extract a good validation set, representative of the data. A popular technique for this matter is the *cross-validation*, in which each training session selects a different set of data for validation. First, the entire dataset is divided into n *folds*. Then, the dataset is trained n times using $(n-1)$ folds for training and 1 fold for validation. The final metrics are the mean of all n results. With this method, it is important to respect the proportion of each class in each fold.

3
Related works

The past decades have seen the development of many tools for computer-aided fault detection in migrated seismic data. The vast majority of methods is based on the use of seismic attributes, which can be defined as « the quantities that are measured, computed or implied from the seismic data » (Subrahmanyam and Rao, 2008). They mainly consist in a set of mathematical operations highlighting a certain type of information in the seismic image. Fault attribute maps allow visually enhancing possible fault location by looking at the local continuity of the seismic signal (coherence (Bahorich and Farmer, 1995; Luo *et al*, 1996), semblance (Marfurt *et al*, 1998), variance (Van Bemmel and Pepper, 2000), chaos (Randen *et al*, 2001), edge detection (Di and Gao, 2014)), or at the geometry of the reflectors (curvature (Lisle, 1994; Roberts, 2001; Al-Dossary and Marfurt, 2006), flexure (Gao, 2013)). Among them, the coherence is the best known attribute for highlighting faults (Wang *et al*, 2018). An alternative to seismic attributes is to use the information of interpreted horizons to find fault locations (horizons dip and azimuth maps, (Rijks and Jauffred, 1991)).

Each seismic attribute has its pros and cons, and fails at enhancing faults only; numerous artifacts remain, other structures appear. Seismic attributes usually require massive computation, and, alone, are not suited for efficient fault identification: a human interpreter must spend time finalizing the study manually. Consequently, many authors propose to post-process the attribute maps to extract fault location automatically, usually using some kind of image processing technique. For example, Pedersen *et al* (2002) used Artificial Ants tracking on one or more attribute cubes to extract fault surfaces. Gibson *et al* (2005) used semblance to extract a set of high faultiness points that they join in a multi-resolution scheme to build fault surfaces. Hale (2013) built surface meshes by connecting a selection of high fault likelihood points. Wang *et al* (2014) performed color transformations on semblance maps, followed by a skeletonization of the highlighted fault regions. Recently the same authors proposed the combination of the Hough Transform and tracking vector to extract faults from binarized coherence maps (Wang and AlRegib, 2017). Those methods generally fall into two categories: faults are well extracted but many artifacts remain, or the result is clean, but not all faults are detected (Wang *et al*, 2018; Di and Gao, 2017).

Another approach is to combine attributes using machine learning algorithms. Supervised methods extract meaningful information from a set of inputs (the features) and observations (here, the fault location), then apply acquired knowledge to predict new samples. Such methods allow building complex relationships within large amounts of seismic data. One of the first works in this direction is a study from Tingdahl *et al* (2005), where they used a set of 12 seismic attributes as input features of an Artificial Neural Network, generating fault probability maps that can be seen as a new attribute. More recently, Support Vector Machine (SVM) achieved promising results when applied on a selection of seismic attributes (Di *et al*, 2017) or image texture attributes (Guitton *et al*, 2017). These techniques imply a tedious step of attribute selection and computation, an operation which can be avoided with deep learning methods.

Compared to standard machine learning techniques, deep networks can learn new features dynamically during training, explaining their success in solving complex tasks (Voulodimos *et al*, 2018). Among them, image-oriented methods like CNNs are especially promising as they recently achieved state-of-the-art results in automatic fault detection (Wang *et al*, 2018). The use of CNNs is recent in the seismic field, as the first work for fault detection was by Huang *et al* (2017), in which the authors built many CNNs in parallel on a set of 9 seismic attribute cubes, then fed the extracted features to a single MLP. In other words, they did not take advantage of the ability of CNNs to automatically extract features from the data: as seismic attributes are derived from the seismic amplitude, a CNN should be able to compute relevant attributes from the amplitude input automatically during training. In this direction, Di *et al* (2018) proposed to classify amplitude patches on a real cube and showed promising results. Overall, despite very encouraging results, deep learning techniques present some drawbacks. As supervised learning methods, they require a large amount of marked seismic data as input; the generalizability of the proposed networks to other seismic cubes is hard to assess, as studies generally train and classify in the same field; building a CNN model on new data involves a long tuning step, due to the substantial amount of hyper-parameters to adjust.

In this thesis, we propose a two-step scheme to build a good fault classifier, addressing each of the aforementioned CNN drawbacks. First, we build a classifier on a set of synthetic data: a huge amount of error-free labeled data is thus easy to obtain. Second, we apply TL methods to adapt the classifier to real data. In this step, only a small number of marked data is needed, and the number of parameters to tune is also greatly reduced. TL methods also ensure the generalizability of the technique, by providing a way to adapt to any seismic cube.

4 Fault detection in synthetic amplitude maps using CNNs

Compared to real data, synthetic seismic data present several advantages for deep networks training. First, they allow total control of the ground truth and thus avoid marking errors. Second, the marking being automatic and fast, synthetic data provide an easy scaling regarding the number of training examples. Finally, they avoid problems with data privacy, allowing free distribution.

We propose a methodology in four steps. First, we generate synthetic seismic images where we control the location of the faults. Second, we extract Fault and Non-fault patches from the generated dataset. Then, we train and fine-tune different CNN architectures focusing on maximizing quality metrics. Finally, we classify pixels in new synthetic images and post process the results for fault segmentation.

4.1 Synthetic dataset generation

The open source code IPF from Hale (2014) allowed us to reproduce the results of migrated seismic data. Beginning with a randomly generated reflectivity model extended along the section, simple image transformations recreate sequential rock deformations along time: shearing, folding and faulting. We can then apply convolution with a Ricker wavelet and add random noise. Each step of the process can be parameterized. We built a dataset of 500 images of 572x572 pixels, all containing one straight fault crossing the section entirely, modifying randomly fault angle, position and throw, shearing slope, folding amplitude and frequency, wavelet peak and amount of noise. Resulting images present amplitude values between -1 and 1. Along with the seismic amplitude information, we generated for each image its corresponding binary mask, indicating in white the location of the fault. Figure 4.1 shows an example of such a pair.

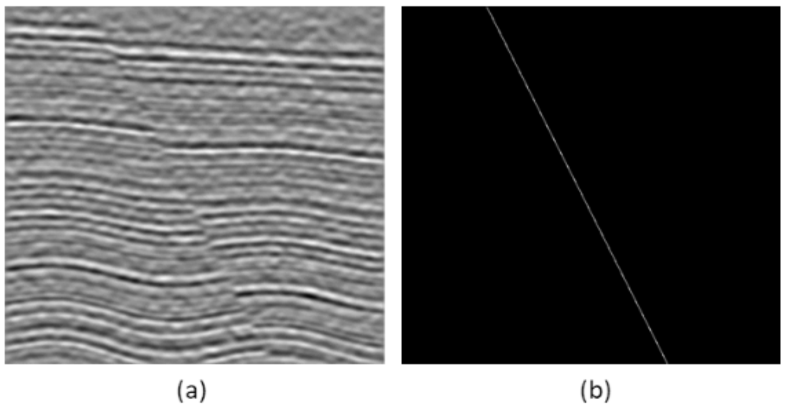


Figure 4.1: (a) Example of a synthetic seismic image from our dataset (b) Corresponding binary mask.

4.2 Patch extraction

Many machine learning techniques use features extracted from images as input. Features are relevant information that we think could be efficiently combined to achieve the desired classification. One of the advantages of the CNN is that it does not require an explicit feature extraction step. Instead, the neural network uses the image itself as input and attempts to extract the best features implicitly. Applying these principles to the seismic imagery area, good feature candidates are naturally any fault enhancing seismic attribute. Such attributes are computed using a small neighborhood of seismic amplitude values. Seismic amplitude is thus at the core of the fault detection problem, and a small neighborhood of amplitude values can be used as input to a CNN, that will hopefully find and compute the best seismic attributes dynamically, without the need of explicitly passing them as input. This small neighborhood is what we call here a patch.

Since faults may be located anywhere in the seismic image, all pixels are fault candidates. Our approach seeks to classify all pixels as fault or non-fault pixels. A patch is composed by the candidate pixel itself at the center and its neighbor pixels. The classification of a pixel is the classification of its patch. To separate the pixels in our two target classes, Fault and Non-fault, we use binary mask images, which contain the marking of the faults. If a pixel in the seismic image is masked by a white pixel in the binary image, this pixel is considered as Fault. Similarly, black pixels in the binary masks are considered Non-fault. Additionally, if a pixel is Non-fault but the fault passes somewhere in its patch (partial faulting), we discard the patch: in this work, such patches are simply not trained. Figure 4.2 shows the three types of patches in the binary domain. Figure 4.3 shows an example of one Fault patch and one Non-fault patch extracted from a synthetic seismic image.

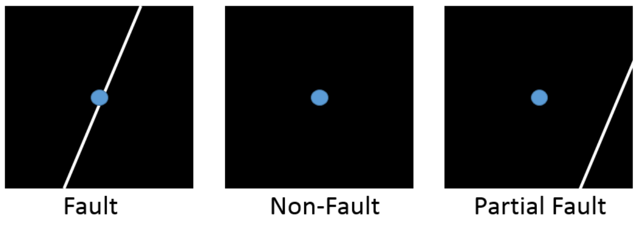


Figure 4.2: Types of binary patches. The center pixel is highlighted.

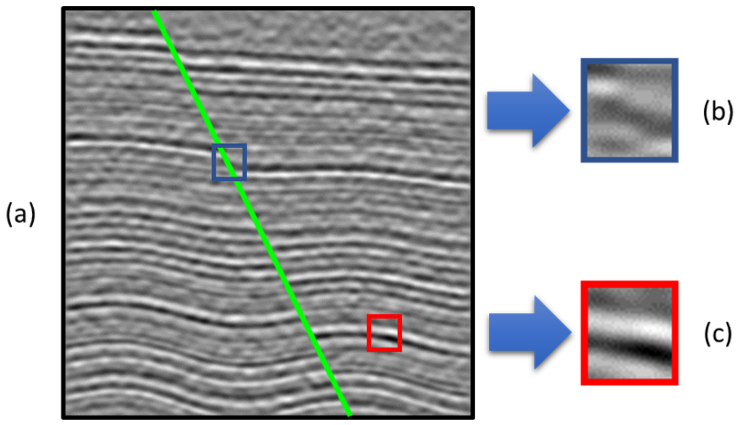


Figure 4.3: Extracting patches. (a) Seismic image crossed by a fault. The fault location is highlighted; (b) fault patch; (c) non-fault patch.

Follows how the sets of patches are used as input to the CNNs:

– For training images, we extract all possible Fault patches and one Non-Fault patch every 23 pixels. This generates a balanced number of patches for the two classes, which is desirable for training.

– For validation images, we extract all possible Fault patches and one Non-Fault patch every 10 pixels, to account for classes' natural imbalance in practice, and thus obtain interpretable quality metrics.

– For test images, we extract one patch every 3 pixels, regardless of the binary mask. This small pixel step ensures the fault will be crossed, while efficiently generating classifications suited for visualization.

4.3 CNN training

We applied and tested different CNN architectures and parameters, along with the input patch size and resolution.

Beginning with a common LeNet architecture (Lecun *et al*, 1998), we added complexity, applying results from the VGG-Net (Simonyan and Zisserman, 2014): deeper networks are better at differentiating classes, and with deeper networks small convolution filters of 3x3 pixels should give good results. However, adding too many layers to a network can have negative effects: the training time increases dramatically, and the classification can fall into overfitting the training data. Consequently, there is a trade-off to find. Following this method, we tuned the number of convolution, pooling and fully-connected layers, the number and size of learnable filters and the number of neurons in the MLP hidden layers.

All training sessions also shared some parameters: we used a learning rate value of 0.001, momentum value of 0.9 and input batch of 30 patches.

To choose the input patch size, we considered two aspects. First, large patches can contain more than one fault: using too large patches we may lose in precision. However, small patches contain less information and may lead to poor classifiers. We consequently tried to find the right trade-off.

We also tested two different patch resolutions:

– Integer values between 0 and 255, corresponding to common grey scale images;

– Floating point values.

We used 400 seismic images for training (381,079 patches), 50 images for validation (148,632 patches) and kept the remaining 50 images to perform tests.

For each configuration, we estimated the quality of the classifier on six common criteria, considering the Fault class as the positive class: accuracy, sensitivity, specificity, F1-score, Area Under the ROC curve (ROC AUC) and a visual evaluation on entire sections of the test set. Sensitivity, the capacity of the network to output true positives, should be high enough to underline the faults. Specificity should be as close to 1 as possible as even a small number of false positives tend to give poor visual results on the test sections.

We used Python with the Keras library (Chollet, 2015) to implement our CNNs.

4.4 CNN classification and post-processing

Once a good model is obtained, new images can be classified. Depending on the image size and fault scale, one can choose to classify all pixels or just a subset. In any case, the result will be a binary image with sets of white pixels at the predicted fault location. Extracting the fault thus requires a post-processing. We chose a simple sequence of morphological operations:

– *Dilation* adds white pixels to the boundary of detected objects. It tends to close holes and join adjacent parts of the objects: this operation aims at reducing the number of false negatives (pixels wrongly classified as Non-Fault).

– *Erosion* removes white pixels from the boundary of detected objects. It tends to clean the object's boundary and erase isolated white pixels: this operation aims at reducing the number of false positives (pixels wrongly classified as Fault).

– *Thinning* extracts the skeleton of the object. It reduces the number of Fault pixels without losing information.

Those steps clean the image and result in smaller Fault point sets. We then apply the Hough Transform (Hough, 1962) to finally extract the faults: the method allows detecting alignments of white pixels in our post-processed classification images. The method is here restricted to straight lines and we also give a restriction on fault angles, which should be sub-vertical.

4.5 Results

The network architecture which gave the highest metrics is described in Table 4.1. Input are patches of size 45x45, with a floating point precision. The network gave an accuracy of 0.98, a sensitivity of 0.95, a specificity of 0.99, F1-score of 0.97 and ROC AUC of 0.99.

Figure 4.4 shows the classification and fault extraction of sections from the test set. Classification is performed every three pixels. The fault is clearly highlighted, but still coarse because patches of type Partial fault are present and mainly classified as Fault. Since the training step did not include such patches, as stated in section 4.2, those results are unsurprising. Note also that such patches did not enter in the calculation of the quality metrics, over-estimating all values except sensitivity since the number of false positives was underestimated.

Layers	Conv	Conv	Conv	Max Pool	Conv	Conv	Conv	Max Pool	FC + dropout	FC
Number of filters	20	20	20		50	50	50			
Filter size	3x3	3x3	3x3	2x2	3x3	3x3	3x3	2x2		
Number of neurons									16	32

Table 4.1: CNN Architecture that achieved the best results on synthetic data, after 50 epochs. With: Conv = Convolution, Max Pool = Max-Pooling, FC = fully connected layer. Convolution steps are followed by a ReLU activation. The MLP part activation functions are ReLU. Last layer is a softmax classifier with 2 output neurons.

We observed similar results on all test sections, including sections with configurations that were not shown during training: varying fault throw (Figure 4.4(center)), faults crossing each other (Figure 4.4(bottom)). Indeed, as patches contain local image information, a variety of fault geometries can be detected even when not specifically trained.

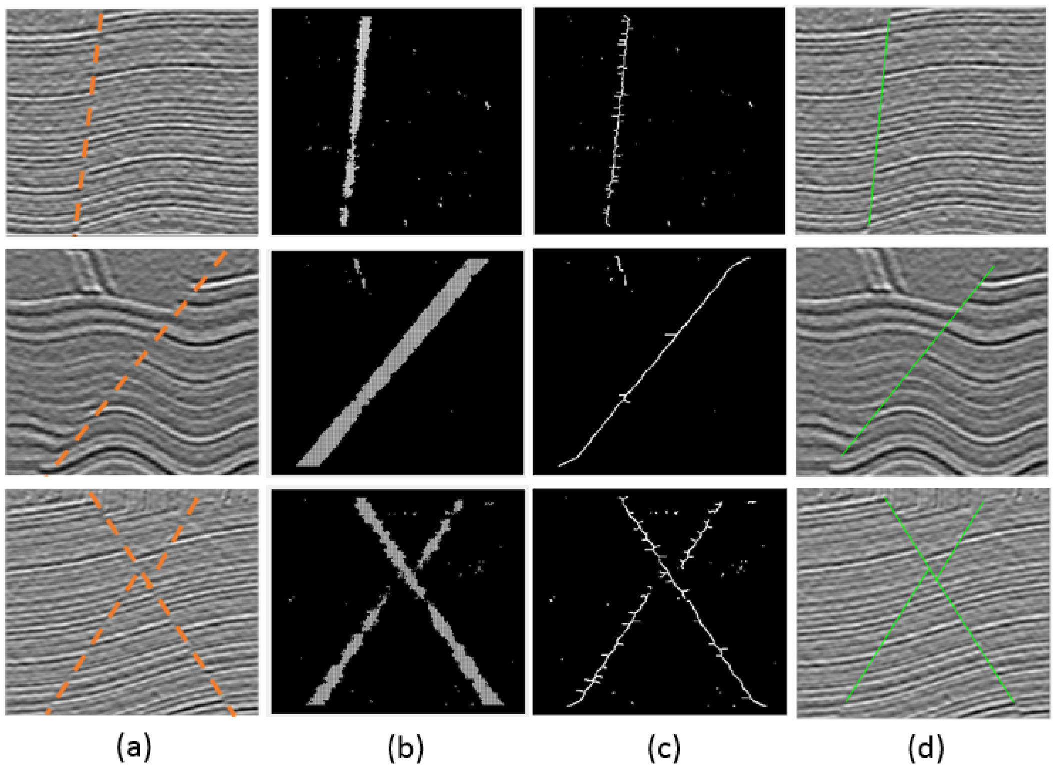


Figure 4.4: Classification and fault extraction on synthetic test sections. (a) Input section with expected fault marked in dashed lines; (b) raw classification of 1 over 3 pixels; (c) results of erosion, dilation and thinning on (b); (d) extracted faults using the Hough transform on (c).

5
Fault detection in real seismic data

5.1
Applying the synthetic classifier to real data

5.1.1
Synthetic and real data comparison

We use as case study a real data from the North Sea, the F3 cube (OpendTect, 1987). It contains 650 in-lines, 950 cross-lines and 462 time sections. We assessed the frequency content of the data by performing the Fourier transform on seismic traces. We compared with the frequencies of our synthetic dataset. It appears that our synthetic dataset does not match real data: while real seismic signal ranges between 20 and 50 Hz, the synthetic dataset does not present patches with a higher frequency than 20 Hz, as shown in Figure 5.1.

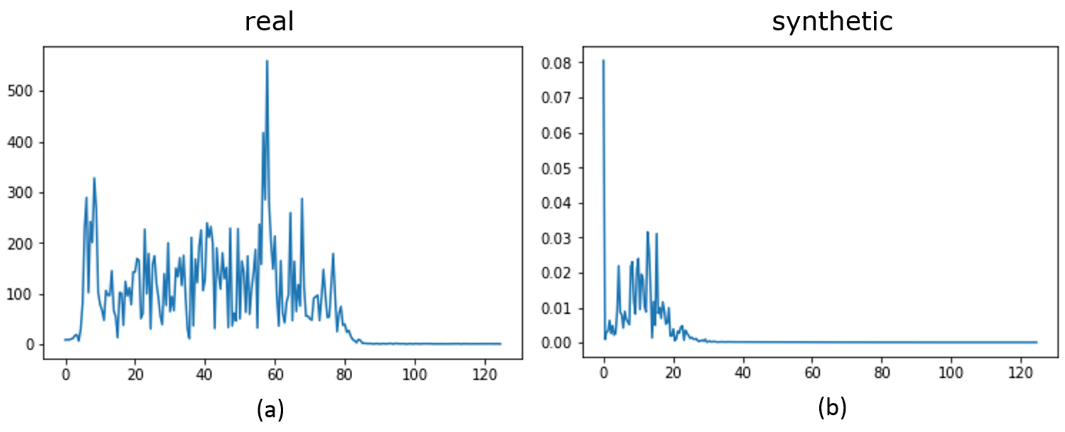


Figure 5.1: (a) Frequency range of the F3 cube; (b) frequency range of the synthetic dataset. Unit is Hz.

As a result, patches of size 45x45 present different amplitude patterns in the real and synthetic data. Figure 5.2 shows an example of patch extracted in both cases.

However, a proper scaling of real patches allows minimizing this difference: indeed, selecting smaller patches in the real data and resizing them to the proper input size of 45x45 should naturally lower their frequency content.

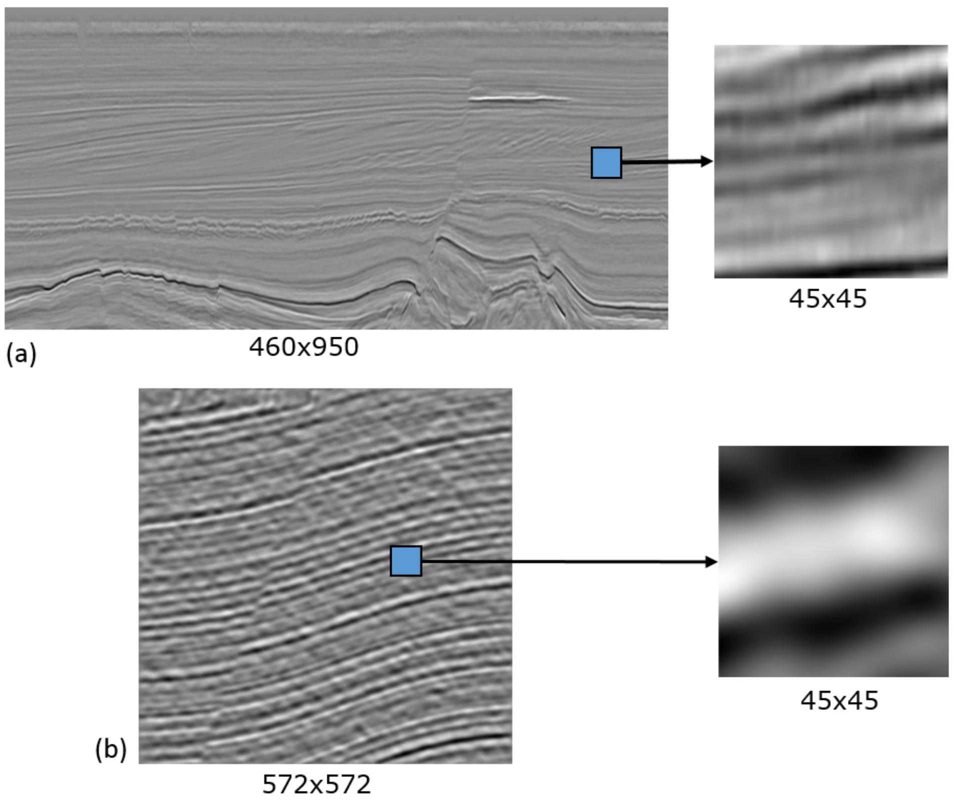


Figure 5.2: (a) Real section from the F3 cube and an extracted patch of size 45x45; (b) Synthetic section from our dataset and an extracted patch of size 45x45. We observe more thin horizons in the real patch, due to the higher frequency of seismic traces in real data.

We propose here a method to automatically select the proper patch size, based on patch texture comparison.

5.1.2 Automatic patch size extraction

We begin by randomly selecting 500 patches in the synthetic dataset, and extracting 500 patches of constant size in the real dataset. We clip the histogram of the real data to remove common outlier values. We smooth the real patches to minimize noise, and resize them to the expected size of 45x45 using a bicubic interpolation. Then, for each patch we compute six of the Haralick textural features (Haralick *et al*, 1973), a set suggested by Conners *et al* (1984): Inertia, Cluster Shade, Cluster Prominence, Local Homogeneity, Energy, and Entropy. For efficiency, we calculate them in the grey-level domain: amplitude values are normalized between 0 and 255.

Using these texture values as coordinates, we build the *proximity matrix*, which contains the Euclidean distance between each patch pair. This distance represents here the texture dissimilarity between patches inside and between sets, in 6-dimension.

We then use the classical Multidimensional Scaling (MDS) method (Kruskal, 1964) to represent computed distances in 2-dimension. Given the proximity matrix for dimension n, classical MDS allows visualizing the coordinates of the objects in a dimension $m \leq n$, by minimizing the *Stress function*:

$$Stress = \sqrt{\frac{\sum_{i<j}(d_{ij} - \hat{d}_{ij})^2}{\sum_{i<j} d_{ij}^2}} \tag{5-1}$$

where d_{ij} are the input distances between pairs of objects and $\hat{d}_{ij}$ are the predicted distances in the output space, called disparities. Resulting is a 2D graph in which each point corresponds to a patch and distance between points represent their texture dissimilarity. Figure 5.3 plots the synthetic patch set along with real patches initially of size 45x45, 20x20 and 8x8.

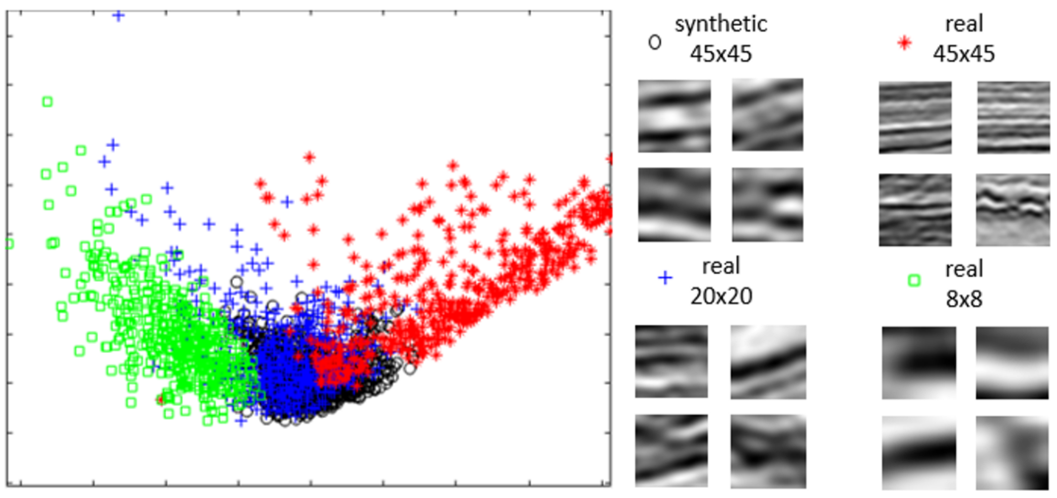


Figure 5.3: Relative texture dissimilarity between patches, using MDS to plot the computed 6-dimensional Haralick texture distances in 2-dimension. Each point corresponds to a patch. The different symbols distinguish sets of same size patches, synthetic or real.

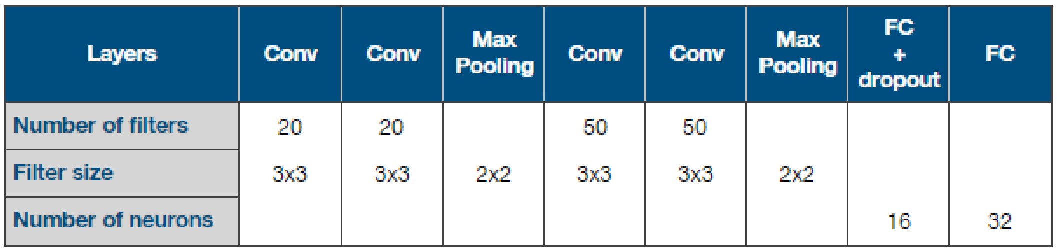
For each set, texture is roughly homogeneous and patches appear as clusters. The distance between cluster centroids give us the relative dissimilarity between patch sets. We found that the closest textures between synthetic and real data are obtained for real patches of size 20x20.

5.1.3 Best architecture

In the case of real data, we do not have access to the ground truth, and thus cannot compute quality metrics. We thus estimate the model performance visually, comparing the results on real sections where faults are interpreted. We select two time sections where some faults are clearly visible, and perform classification on all in-line sections around the faulty region. The time section allows assessing the continuity of the classification throughout in-lines.

Classification on in-lines follows a specific process: first, we clip the histogram under a threshold (+- 6000) to enhance visualization, and smooth the section to reduce noise. Then, we set the amplitude values of the entire section between -1 and 1 in order to be close to the training conditions. Next, we extract patches of size 20x20, and resize them to the expected classification size of 45x45 using bicubic interpolation. We finally classify every pixel in the images.

We test different CNN architectures, including Table 4.1 CNN. The best architecture was obtained with the CNN summarized in Table 5.1.



Layers	Conv	Conv	Max Pooling	Conv	Conv	Max Pooling	FC + dropout	FC
Number of filters	20	20		50	50			
Filter size	3x3	3x3	2x2	3x3	3x3	2x2		
Number of neurons							16	32

Table 5.1: CNN Architecture that achieved the best results on real data, after 60 epochs. With: Conv = Convolution, FC = fully connected layer. Convolution steps are followed by a ReLU activation. The FC activation functions are ReLU. Last layer is a softmax classifier with 2 output neurons.

Interestingly, the CNN with the highest performance in both synthetic and real case is different: Table 5.1 CNN reveals a greater generalization capacity than Table 4.1 CNN, despite poorest quality metrics if tested against the synthetic validation set (accuracy: 0.94, sensitivity: 0.69, specificity: 0.99, F1-score: 0.80, AUC : 0.96).

Figure 5.4 shows the results of this CNN on the two selected time sections, where we provide our fault interpretation. We see that the classifier manages to find parts of the faults, but is totally missing other parts. It also outputs few false positives, which is good. Overall, those visual results are promising but are not sufficient to make the method valuable in practice. In the next section, we show how we can use transfer learning methods to improve our classification.

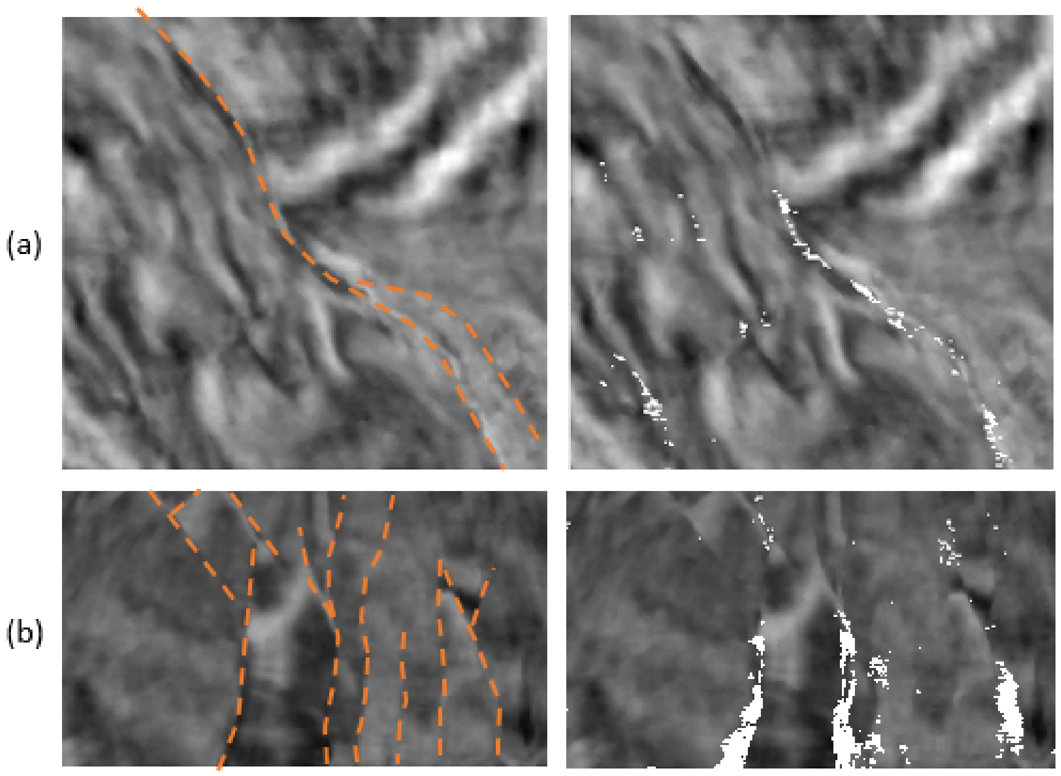


Figure 5.4: (a) (left) Time slice 924 (inlines 150 to 350, cross-lines 840 to 1050) with simple interpretation; (a) (right) classification results. (b) (left) Time slice 1736 with simple interpretation (inlines 110 to 240, cross-lines 390 to 590); (b) (right) classification results.

5.2 Using Transfer Learning

We propose to use our synthetic classifier as a pre-trained model for applying TL methods on real data. The classifier is a sequence of adjusted weights for each convolution filter and neuron connection. TL methods use all or part of those weights as the initial state for training a new dataset on a different classification task. While training a CNN from scratch requires a large amount of input data, TL allows working with small datasets, and usually performs well if the classification tasks are similar. Here, we extract a small dataset from the real data and apply TL for the same task: fault detection. Figure 5.5 illustrates the concept.

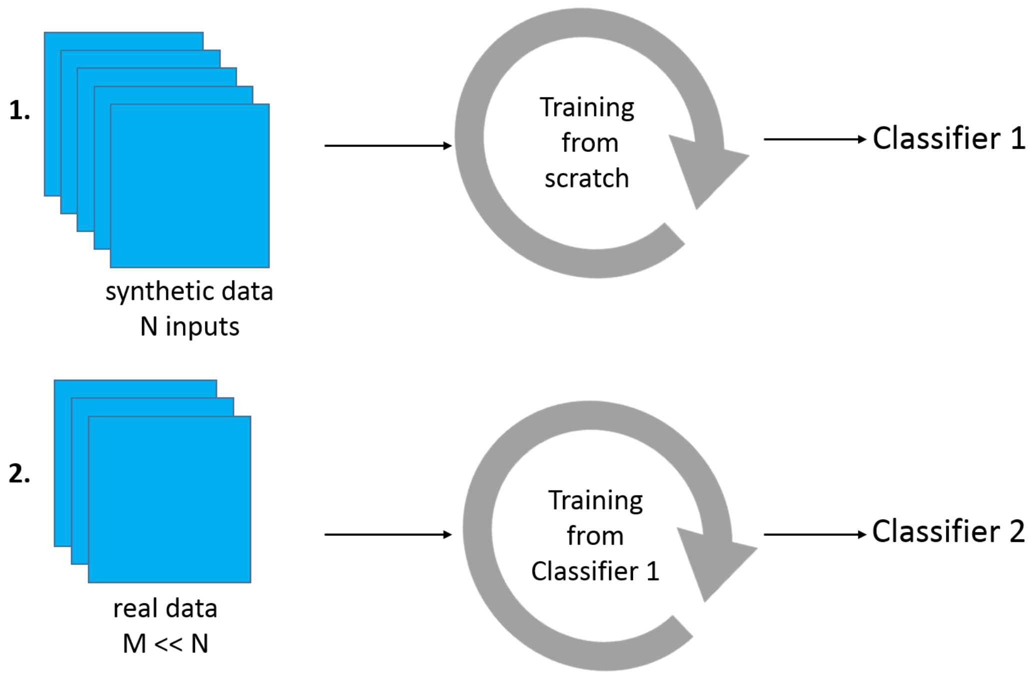


Figure 5.5: Process to build a good classifier for real seismic data. Step 1.: construction of a classifier on a huge amount of synthetic data, as presented in section 3. Step 2.: training of a small dataset of real patches, using classifier 1 as pre-trained model and applying TL.

5.2.1
Building the real dataset

In this step, we focus on minimizing the manual effort for real dataset patch extraction. First, we select a single, small section of the seismic cube where some faults are clearly visible: cross-line number 900. We then format the section to enhance fault visualization and to match the pre-trained model training conditions: we clip the histogram between amplitude values of +- 6000, then smooth the section to minimize the effect of noise, finally we normalize amplitude values between -1 and 1. We manually pick some visible faults and generate the binary mask for the section, as shown in Figure 5.6.

As we know that marking all faults is difficult, we suppose that all faults of the section were not marked. Consequently, if we extract Non-Fault patches directly using this mask, the dataset will contain errors: many Non-Fault patches may indeed contain faults. To minimize this error, we thus select some regions where we are confident that no fault exist.

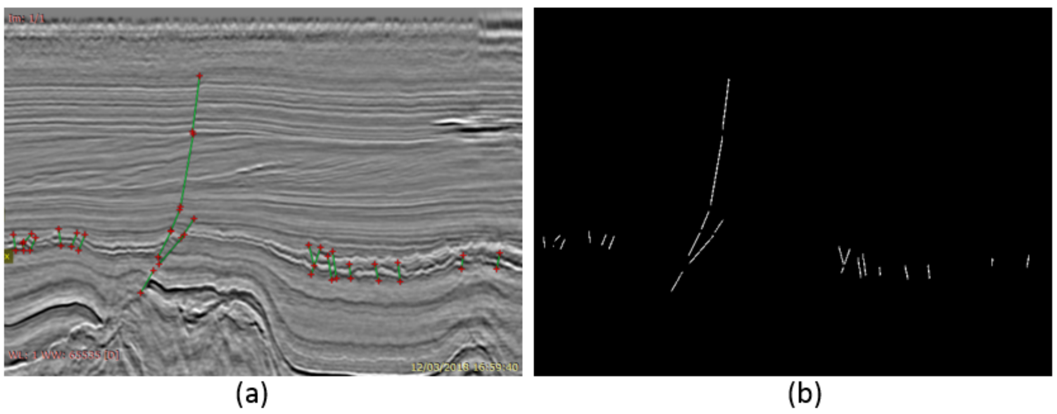


Figure 5.6: (a) Cross-section 900 with simple marking of the faults; (b) corresponding binary mask.

Finally, we extract all Fault patches using the binary mask and a set of Non-Fault patches using the defined non-faulted regions. The resulting dataset is highly imbalanced: extracting 1 Non-Fault patch every 7 pixels for example results in 3252 Non-Fault for 252 Fault patches. We thus performed data augmentation on the Fault class: this method consists in artificially increasing the dataset by applying some kind of operation on the existing data. One of the only relevant operation in the case of seismic images is horizontal flipping. Indeed, vertical flipping or rotation may lead to unrealistic configurations, while scaling would change the patch amplitude patterns as discussed in section 5.1.1.

We finally resize all patches from 20x20 to the expected input size of 45x45. The final dataset contains 504 Fault patches, resulting of a balance of 13.4% for the Fault class, which is still very low.

5.2.2 TL strategies

TL strategies rely on the observation that the different parts of the CNN learn different things. In particular, earlier convolutional layers learn general features such as edges or blobs, while deeper convolutional layers learn specific features related to the given classification task and dataset. As we ensured similarity between tasks and datasets, we assume that the pre-trained model specific features should be of good quality. Promising visual results of Figure 5.4 (section 5.1.3) tend to reinforce that guess.

We thus investigated two common TL strategies: fine tuning the whole network, and using the CNN as a Feature Extractor with a tuning of the classification layers. In the latter strategy, we test two types of classifiers: the

Multi-Layer Perceptron (MLP) and the Support Vector Machine (SVM).

Full fine tuning (FFT)

We train the entire network with initial weights the pre-trained model weights instead of random ones. In this context, the learning rate should be low in order to avoid a sudden change in weights during back-propagation. The number of epochs should also be low to avoid overfitting.

Feature extractor with MLP (FE-MLP)

We freeze the weights of the whole convolutional part of the CNN, in order to extract fixed features that we feed in the fully connected layers. We fix the number of hidden layers to 2 with dropout between layers, and the learning rate to 0.01. We tune the number of neurons in each layer, and the number of epochs.

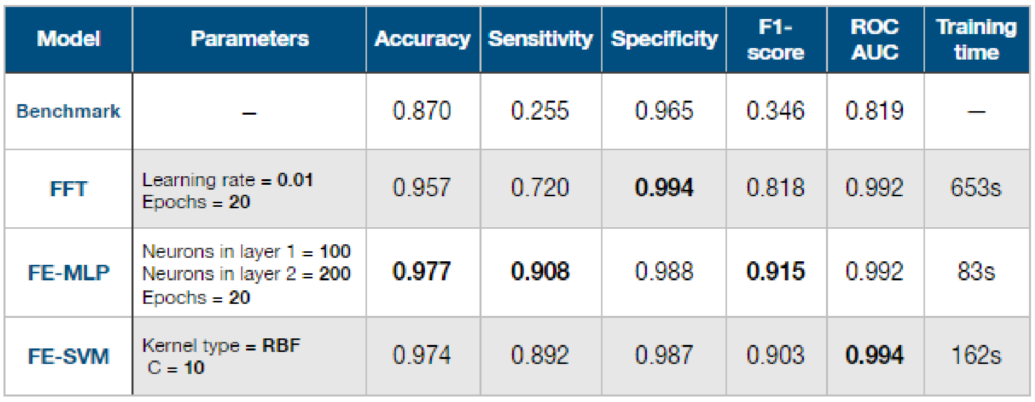
Feature extractor with SVM (FE-SVM)

In many deep learning applications, SVM proved better than MLP at predicting classes, given good input features (Tang, 2013). To find a good SVM classifier, we tune the SVM kernel type (linear, polynomial and RBF) and the value of constant C controlling the penalization of outliers and thus the softness of the SVM margins.

5.2.3 Results

We use as benchmark the pre-trained model applied on real data without TL, presented in section 5.1.3, Table 5.1. Table 5.2 shows the parameters and the numerical performances of each model. Benchmark metrics were calculated over the entire real dataset. For TL models, due to the small size and high imbalance of the dataset, metrics were averaged over a 5-fold cross-validation, respecting in each fold the class proportions. We considered the Fault class as the positive class.

The benchmark model shows a high specificity, confirming our visual conclusions that it outputs few false positives: it is good at discarding faults. However, it has a very low sensitivity, indeed as we saw it misses a lot of faults in practice.



Model	Parameters	Accuracy	Sensitivity	Specificity	F1-score	ROC AUC	Training time
Benchmark	–	0.870	0.255	0.965	0.346	0.819	–
FFT	Learning rate = **0.01** Epochs = **20**	0.957	0.720	**0.994**	0.818	0.992	653s
FE-MLP	Neurons in layer 1 = **100** Neurons in layer 2 = **200** Epochs = **20**	**0.977**	**0.908**	0.988	**0.915**	0.992	83s
FE-SVM	Kernel type = **RBF** C = **10**	0.974	0.892	0.987	0.903	**0.994**	162s

Table 5.2: Quantitative evaluation of benchmark and TL models. Parameters were tuned manually. Training times were obtained with an $Intel^{@}$ $Core^{TM}$ $i7 - 5960X$ Processor Extreme Edition, with 16 threads.

TL models all manage to drastically increase sensitivity without losing in specificity, despite the under representation of the Fault class in the dataset. Feature extractor strategies present overall better results than the Full Fine Tuning method. In particular, they have a better sensitivity, meaning they should highlight most of the faults.

All TL models were trained in few minutes on CPU. Note that the 5-folds cross-validation increased the training time by 5 compared to a single training and validation step.

In order to better assess the models' quality, Figure 5.7 and 5.8 show classification over the two study time sections of the F3 cube. Once again, we classified all inlines of the regions and painted to white the Fault patches' central pixels on the time slice. We provide our simple fault interpretation in order to help following the faults.

FFT model performs surprisingly well on the section of Figure 5.7, however it completely misses some of the faults of Figure 5.8 section. Both FE methods give similar classification, but FE-SVM seems to provide slightly cleaner results.

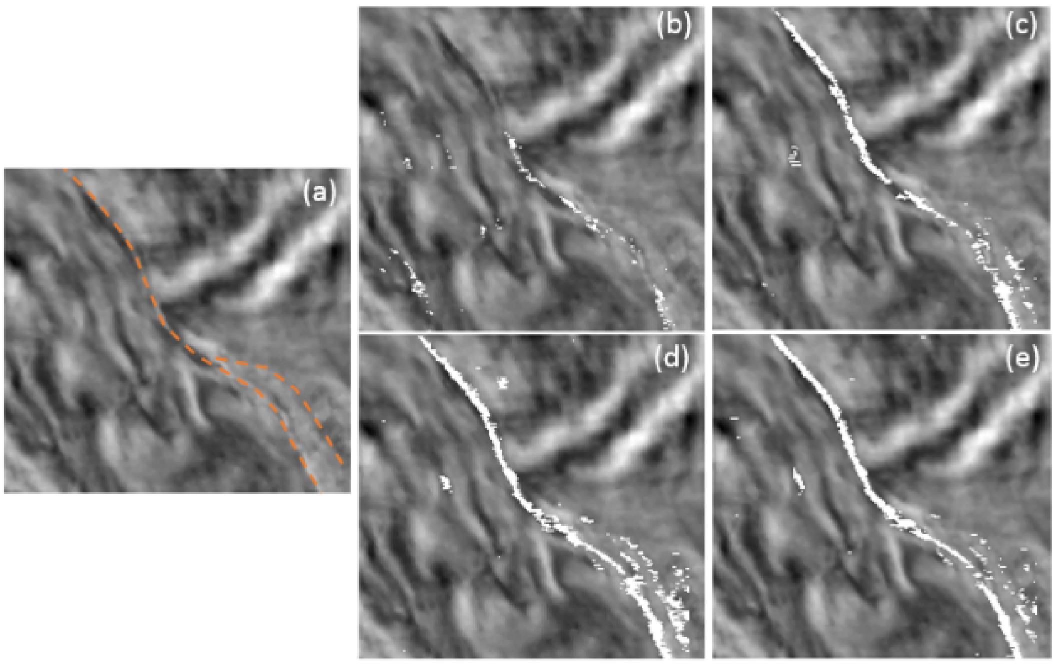


Figure 5.7: (a) Time slice 924 (inlines 150 to 350, cross-lines 840 to 1050) with simple interpretation. (b) Benchmark classification. (c) FFT. (d) FE-MLP. (e) FE-SVM.

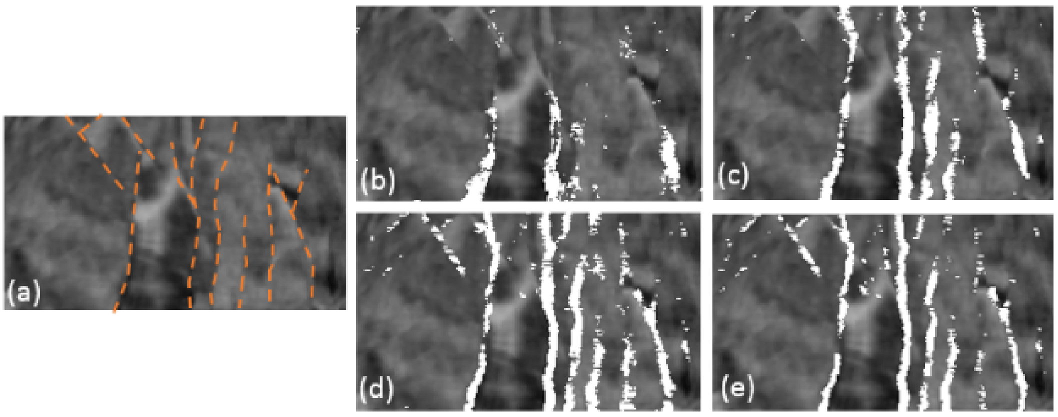


Figure 5.8: (a) Time slice 1736 with simple interpretation (inlines 110 to 240, cross-lines 390 to 590). (b) Benchmark classification. (c) FFT. (d) FE-MLP. (e) FE-SVM.

6
Conclusions for Part I

6.1
Summary

This part presented a methodology for the detection of faults in seismic amplitude images, using a patch-based CNN approach for training and classification. CNNs were first trained with synthetic data only and yielded good results when applied to new synthetic data, with a perfect match of the predicted and ground truth fault after applying the Hough Transform post process. They also revealed promising when classifying pre-processed real data, leading us to use our best CNN architecture as base model to TL processes in order to adapt the classifier to our real data case study. With a small dataset generated by marking only some of the faults on a single cross-section, we were able to build a satisfying classifier trained in a few minutes on CPU.

6.2
Conclusions

The use of CNN allowed us to train with the seismic amplitude map as the only input feature: explicit feature extraction through computation and selection of seismic attributes was unnecessary. However, we highlighted the large number of empirical parameters used in CNNs, which makes model fine-tuning difficult. We were able to overcome this drawback with the use of TL. Detecting real faults using TL from the synthetic seismic data classifier showed powerful despite the naturally high imbalance of classes in real data. The proposed strategies implied a tuning of a small number of parameters, the Feature Extractor with SVM model being the easiest to adjust. Using a CNN trained with synthetic data, this work shows that it is not necessary to have a large amount of well-marked real data to build a good fault detector with supervised techniques. However, results tend to mimic the interpretation and their quality thus depends on the quality of the marking.

6.3 Suggestions for further research

There are several research directions we can suggest to improve this work and go further.

First, we could improve the synthetic dataset by generating data with the commonly seen frequencies in real seismic signals. With this new data, we could investigate multi-scale CNNs, as in Schlegl *et al* (2015).

Second, we could improve the synthetic classifier in different ways. Patches of the type Partial Fault could be added as a new class during training. One idea could also be to assign a value of the distance of a patch to the fault instead of a fixed class. Patches far from the fault would receive a higher penalization if classified as Fault, taking as, for example, the work from Araya-Polo *et al* (2017). Also, the architecture of the CNN could be improved: we could try to visualize and better understand the generated features, for example using the Layer-Wise Relevance Propagation (LRP) method (Bach *et al*, 2015). We could also use evolutionary or other optimization algorithms to find the best architecture automatically (for example, see the work from Yamasaki *et al* (2017)).

Concerning TL results, a straightforward test to improve the classification is to increase the size of the real dataset, for example by classifying another section. Also, we could test other balance between the two classes. Other pre-trained models could also be studied. As shown in (Wang *et al*, 2018), there are many ways to build the main classifier, using synthetic or real data and even non-migrated data. Powerful pre-trained classification models like VGG-Net (Simonyan and Zisserman, 2014) are readily available online, and it would be interesting to compare the TL performances using such models, trained on a tremendous amount of images from the ImageNet database (Deng *et al*, 2009), with TL from models designed explicitly for fault detection as presented here.

Another line of research to improve the method is to work on the post-processing step used to extract the exact fault location from the classification results. In the synthetic case, faults are straight lines and simple morphological operations coupled with the Hough transform were enough to obtain a perfect match between predictions and ground truth. With real data, this process should be adapted to faults with any geometry, for example in the fashion proposed by Wang and AlRegib (2014).

One interesting research direction is also the exploitation of the three-dimensional information contained in seismic cubes. A simple way to do this would be to classify not only in-line sections but also cross-line sections: this should help us detecting some false positives, and faults in various directions.

More interesting, the IPF code used to generate synthetic data allows building 3D cubes: we could develop a CNN taking as input 3D patches.

Finally, during the few months of writing this thesis, many articles and expanded abstracts have been published on seismic fault detection using deep learning, showing the interest of the community in this approach. Two conferences in particular present many works which directions seem very similar to our CNN, patch-based approach: the *80th EAGE Annual Conference and Exhibition* (June 2018) and the *SEG International Exposition and 88th Annual Meeting*, which will be held in October 2018. All those works can indeed be an excellent source of inspiration for future research in our field. We also wish to cite two interesting published articles during this period. Xiong *et al* (2018) trained a CNN with seven annotated synthetic and real seismic cubes to compute fault probability maps which showed better results than coherence maps: the number and the diversity of their data ensure the generalization capacity of their method, but require a considerable effort for preparing the data. Lu *et al* (2018) proposed a pre-process based on a Generative Adversarial Networks (GAN) to enhance the performances of an existing fault detection network: this is an interesting alternative to our TL method for improving the results of a pre-existing CNN.

Part II

Adaptive all-quadrilateral FEM mesh generation

7
Concepts

7.1
FEM meshes

The *Finite Element Method* (FEM) is a numerical tool used to solve complex space and time-dependent physical problems, described by *partial differential equations* (PDEs). For example, in geomechanics it is used for the analysis, estimation, and investigation of the behavior of rocks regarding several problems such as reservoirs compaction and subsidence, reactivation of geological faults, hydraulic fracturing, well stability or well casing integrity. In these cases, the PDEs cannot be solved with analytical methods, and the FEM provides a numerical approximation by dividing the complex domain into smaller parts and solving simple equations in each of them. The solutions are then assembled in a larger system which describes the whole problem. The discretization of the domain in small elements is done through the construction of a *mesh*.

A mesh is a collection of connected entities described by their *topology* and *geometry*: the topology defines the neighboring relationships between the mesh entities, while the geometry stores their spatial position. Typically, a two-dimensional mesh is composed of *elements*, *edges* and *vertices*, though the appellation can vary with the type of data structure. Quadrilateral elements are usually preferred because they reduce the approximation errors and the number of needed elements compared to triangles for example (Zienkiewicz *et al*, 2008). The discretization of a domain with only these elements is referred here as *all-quadrilateral mesh generation*.

In order to ensure the quality of the FEM approximations, FEM meshes have to respect a series of constraints related here to the nature of geomechanical domains. Geomechanical domains depict a portion of the Earth's crust delimited by artificial frontiers on the bottom and sides, and the actual ground surface on the top. They usually extend on some kilometers. They contain geological discontinuities represented by curves (in our 2D case): *horizons* separate pseudo-horizontal rock layers, and *faults* are sub-vertical discontinuities which can be of various scales (from a few cm to several km). In this thesis, we call all curves the *domain boundaries*. Figure 7.1 shows an example of such a domain.

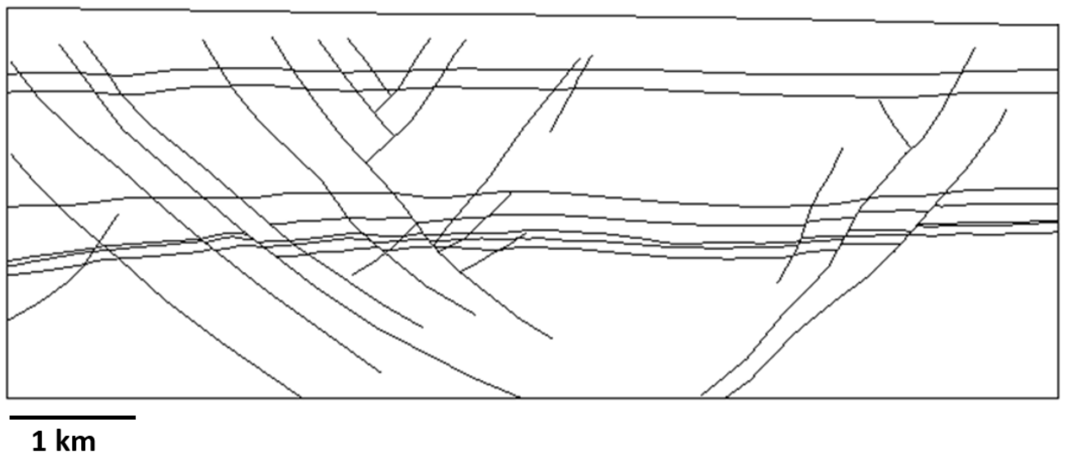


Figure 7.1: Example of a two-dimensional geomechanical domain.

Inside a rock layer, properties such as porosity or permeability are considered constant. Consequently, any flow or pressure propagation simulation will present the highest variability at the vicinity of faults and at the discontinuity between rock layers: the horizons. Those facts allow understanding some of the restrictions a FEM mesh has to respect in order to produce a good solution:

– *alignment with the discontinuities*: properties have to be constant at the scale of each finite element, so it cannot cross any curve in the domain. An element can touch a curve by a vertex or an entire edge only;
– *precision*: alignment with the discontinuities have to be as precise as possible in order to minimize the approximation errors at their vicinity;
– *adaptivity*: the size of the elements should be small enough near the discontinuities to catch the variations of the studied field. Far from them, they can be larger since properties vary less. Varying the size of the elements throughout the domain also allows for creating less elements, thus minimizing the FEM computational effort.

Additionally, any FEM mesh should respect some topological and geometrical constraints:

– *conformity*: neighboring elements must share entire edges and vertices;
– *element's quality*: the quality of the solution depends on the elements' quality, related to their shape. Ideally, elements must be close to perfect squares, as local deformations lower the method's accuracy and can even be a blocker to calculations if any element presents a null or negative area.

The ideal all-quadrilateral mesh should hence be only composed of vertices with a *valence* of 4, as shown in Figure 7.2. The valence of a vertex counts the number of its neighbors, i.e the number of vertices connected to it by an edge.

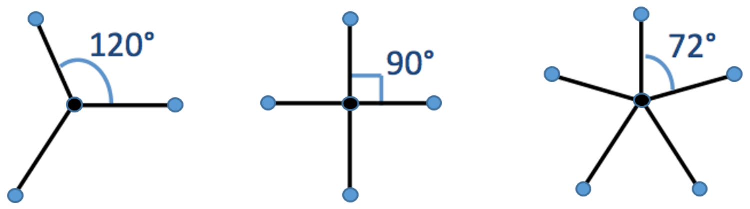


Figure 7.2: The valence of a mesh vertex describes its number of neighboring vertices, connected to it by an edge. Valence 4 (center) is the only way to obtain perfect squares for all involved elements.

This can only be achieved with *structured meshes*, where the connectivity is constant. However, this type of mesh usually lacks of flexibility when it comes to adapting to complex domains. In this thesis, we use *unstructured meshes*, where the valence is not constant in the mesh. Defining precise quality metrics is thus of great importance to evaluate the mesh viability. In the next section, we detail the quality metrics used in this thesis.

7.2 Quality metrics

In order to numerically assess the deformation of the elements as respect to perfect squares, we present three quality metrics that we use in this thesis to evaluate meshes.

7.2.1 The distortion factor

This factor described by El-Hamalawi (2000) estimates the angle distortion on the mesh element. For one element, the deviation of each angle i to 90° is calculated:

$$\delta\theta_i = |\frac{\pi}{2} - \theta_i| \tag{7-1}$$

A deviation vector is defined as the sum of the deviations of the four angles of the quadrilateral. Its norm is the *distortion factor* for the element:

$$||\vec{f_q}|| = \sqrt{\sum_{i=0}^{3}(\delta\theta_i)^2} \tag{7-2}$$

A perfect quadrilateral would have its four angles equal to 90°; therefore, the best distortion factor corresponds to $||\vec{f_q}|| = 0$. The element is flat if every angle has a deviation of 90°, i.e., when $||\vec{f_q}||$ reaches π; above this value it is concave. In the following, we thus consider π as being the upper bound of the quality factor. According to El-Hamalawi (2000), a good quality for an element is ensured if $||\vec{f_q}|| < 1.5$.

7.2.2 The Jacobian

The *jacobian* is a criterion often encountered in FEM mesh generators. It is also an estimation of the distortion of the elements, where we consider the transformation between the reference space (η, ξ) where elements are perfect squares, to the real space (x, y) where elements are distorted (Figure 7.3).

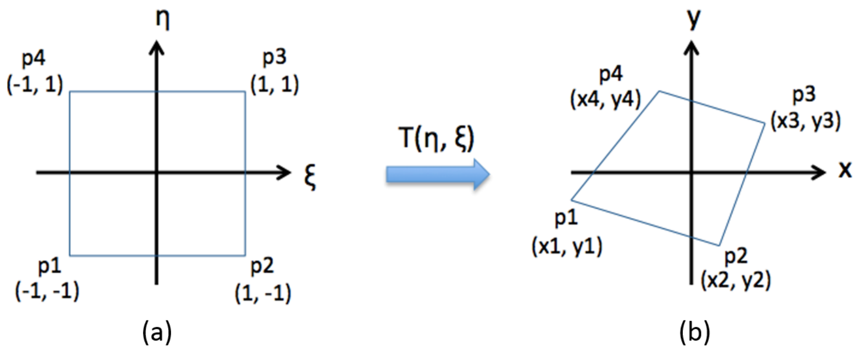


Figure 7.3: (a) Reference element in the reference space. The reference element has fixed coordinates. (b) Real element in the real space, after transformation T.

The transformation that maps the reference element into the real element can be written as follows (Nikishkov, 2004):

$$x(\eta, \xi) = \sum_{i=1}^{n} N_i(\eta, \xi) x_i \tag{7-3}$$

$$y(\eta, \xi) = \sum_{i=1}^{n} N_i(\eta, \xi) y_i \tag{7-4}$$

with η and ξ the coordinates of a point in the reference element, $x(\eta, \xi)$ and $y(\eta, \xi)$ the coordinates of a point in the real element, n the number of corners in the element, x_i and y_i the real coordinates of the i-th corner and $N_i(\eta, \xi)$ the so-called shape functions.

If this transformation is bijective, the ordination of points in the reference space will be the same in the real space. If the ordination is lost, this means that the element is highly distorted, as shown in Figure 7.4.

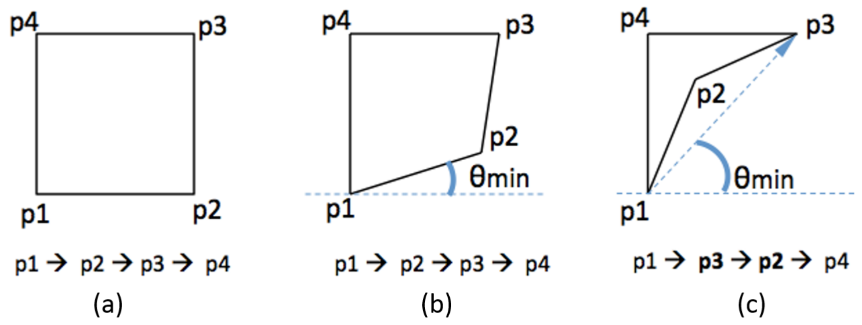


Figure 7.4: Taking $p1$ as reference point: (a) corners are ordered anti-clockwise; (b) a slight deformation does not affect the corners ordination; (c) when the element is highly distorted the ordination is lost as $p3$ comes in second in the increasing angle θ sequence.

The local bijectivity of the transformation can be assessed through the determinant of its Jacobian matrix: if the transformation is bijective, the determinant should be strictly positive. The Jacobian matrix of the transformation and its determinant can be written as follows:

$$J = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \dots & \frac{\partial N_i}{\partial \xi} & \dots & \frac{\partial N_n}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \dots & \frac{\partial N_i}{\partial \eta} & \dots & \frac{\partial N_n}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_i & y_i \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{7-5}$$

$$det\ J = J_{11}J_{22} - J_{12}J_{21} \tag{7-6}$$

In the case of a quadrilateral, the shape function and its derivative are given as:

$$[N]^T = \frac{1}{4} \begin{bmatrix} (1-\xi)(1-\eta) \\ (1+\xi)(1+\eta) \\ (1+\xi)(1+\eta) \\ (1-\xi)(1+\eta) \end{bmatrix} \tag{7-7}$$

$$\left[\frac{\partial N}{\partial \xi}\right]^T = \frac{1}{4} \begin{bmatrix} -(1-\eta) \\ 1-\eta \\ 1+\eta \\ -(1+\eta) \end{bmatrix} \tag{7-8}$$

$$\left[\frac{\partial N}{\partial \eta}\right]^T = \frac{1}{4} \begin{bmatrix} -(1-\xi) \\ -(1+\xi) \\ 1+\xi \\ 1-\xi \end{bmatrix} \tag{7-9}$$

In practice, the Jacobian factor corresponds to a single value for each element, calculated as the ratio between the minimum and the maximum value of the determinant on a set of integration points in the reference space. We use the four corners of the element as integration points. We scale the Jacobian criterion between -1 and 1, the latter corresponding to a perfect shaped element. The Jacobian factor reaches 0 when the area of the element is null and is negative for concave elements. For FEM simulations, it is important to have all mesh elements with a positive jacobian.

7.2.3 Aspect ratio

The two presented factors only consider the distortion of the elements as respect to their angles. However, the *aspect ratio* of the elements is also to consider, because an elongated element will not capture local flux variations as well as a perfect square: the elements' aspect ratio does not have an impact on the calculations accuracy but rather on their precision.

We simply calculate the aspect ratio of an element as the ratio between it shortest and longest edge. A good aspect ratio will be close to 1.

7.3 Data structures

In this section we present the different data structures used in this thesis for quadrilateral mesh construction.

7.3.1 The quadtree

Our mesh construction algorithm is based on a tree structure: the *quadtree*. It is a graph defined by its *root node*, *depth*, and *parent-child relationships* (Samet, 1982). The root is the initial node of the graph (it has no parent) and is recursively divided into four *nodes*. Nodes that are not subdivided are the *leaves* of the tree (they have no children). The level of a node is the number of subdivisions that were necessary to create it. The depth of the tree is the highest level. Figure 7.5 shows an example of a quadtree of depth 3, with 10 leaves.

By associating each node with four 2D coordinates called *corners*, we obtain a spatial representation of the tree. The four corners of a node implicitly define its four edges. Corners are shared by adjacent nodes (Figure 7.6).

The subdivision step, called *refinement*, consists in spatially splitting the nodes into quadrants of equal areas. The size of a node at refinement level i can be assessed through the formula:

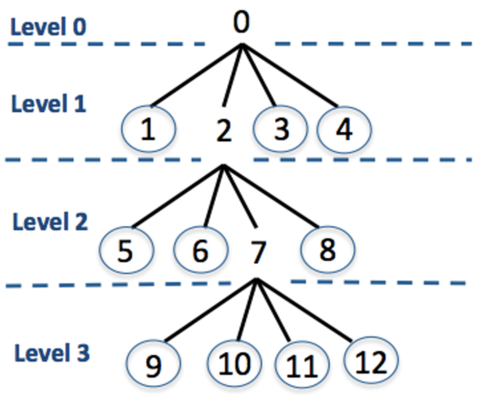


Figure 7.5: Graph representation of a quadtree of depth 3. Leaves are circled. Node 0 is the root of the tree.

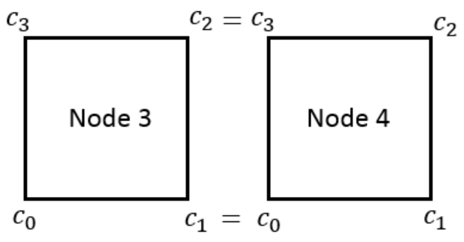


Figure 7.6: Nodes 3 and 4 of Figure 7.5 quadtree. Node corners c_i are ordered anticlockwise. Neighboring nodes share corners.

$$edge\ length\ level\ i = \frac{edge\ length\ root}{2^i} \tag{7-10}$$

Figure 7.7 shows the quadtree from Figure 7.5 in the spatial representation.

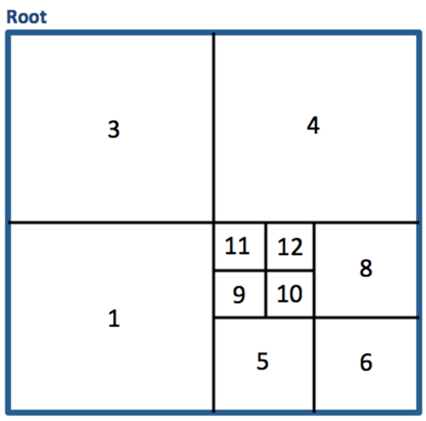


Figure 7.7: Quadtree of Figure 7.5 in the spatial representation. In this representation, only leaves are visible.

In practice, each non-leaf node stores a pointer to each of its children. The children are ordered using their relative position within the parent node, defined by the frontiers they have in common with the parent, using the four cardinal directions (Figure 7.8). The pointer list is always ordered as follows: {SW, SE, NW, NE}. The distinctive advantage of this structure is that the neighboring information between nodes is quickly recovered. As explained by Samet (1982), to find a horizontal or vertical neighbor to a node, a common ancestor is found by going up in the tree, then following a backtracking path mirroring the axis formed by the common boundary of the two nodes.

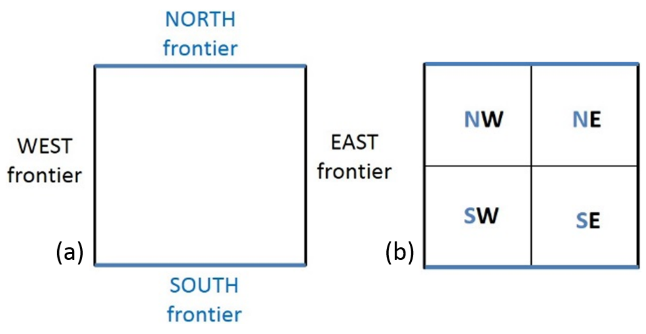


Figure 7.8: Parent-children relationship in a quadtree. (a) Orientation of parent frontiers; (b) Position of children correspond to the parent frontier they are in contact with.

The spatial quadtree can be seen as a quadrilateral mesh, however adjacent nodes of different refinement level present *hanging corners* and this is hence not a conform structure. Schneiders *et al* (1996) proposed a set of *transition patterns* shown in Figure 7.9 to conform the tree.

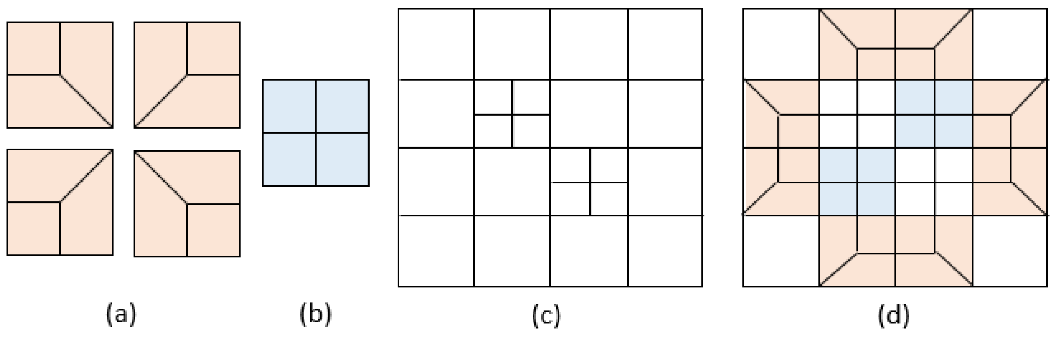


Figure 7.9: (a) Transition patterns from (Schneiders *et al*, 1996); (b) usual refinement pattern is also involved in the tree conformation; (c) unconform tree; (d) conform tree.

The conventional quadtree structure does not have the flexibility to include those patterns, since the children position and implicit neighborhood relationships are not preserved. However, in this thesis we develop a modification of the quadtree structure which allows including the conforming patterns. We then map the generated tree to another structure, the half-edge mesh.

7.3.2 The half-edge mesh

Another important data structure for our study is the *half-edge* mesh (Mäntylä, 1987). It is composed of a list of *faces*, *half-edges* and *vertices*. Figure 7.10 shows how these entities are related. A half-edge, HE, is an oriented edge specific to a face. Two neighboring faces share an edge composed of two HE that are opposite to each other. A face points at one of its HE. A HE points at its opposite HE, its previous and next HE, its face and its initial vertex. Each vertex points at one of the HE it is initial to.

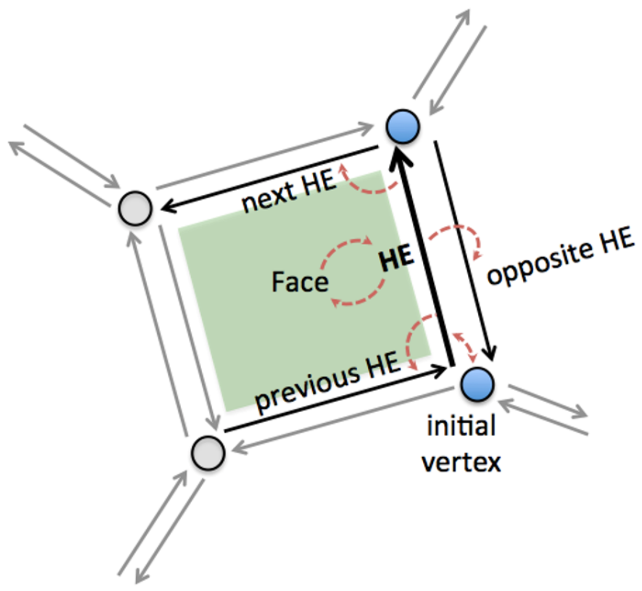


Figure 7.10: The half-edge data structure. "HE" for half-edge. The thick black arrow is the face's reference half-edge. Dotted arrows are pointers.

This structure allows traveling into the mesh very efficiently, knowing at any moment the local neighboring information.

7.3.3 Half-edge versus quadtree

First, for clarity, we sum up the relationships between the entities of each structure in Table 7.1.

Both structures are suited for different operations. The refinement process is easier in the quadtree structure, since creating new nodes is straightforward. It only implies the creation of corners and the assignment of children position. In the half-edge structure, adding a face would imply a series of op-

Quadtree	Half-Edge
Node	Face
Corner (duplicated)	Vertex (unique)
Edge implicitly defined by corners	Explicit half-edge

Table 7.1: Entities of the quadtree structure and their corresponding entities in the half-edge structure.

erations to maintain the consistency of the mesh: inserting new vertices, new half-edges, and updating the pointers of all neighboring entities.

The parent-children information of the quadtree structure makes it easy to find hanging corners and thus unconformities. In the half-edge structure, this information is lost.

Running through the entities of a structure is more efficient in the half-edge mesh, since it stores lists of each entity. For example, computing quality metrics for each face is straightforward. In the quadtree structure, this operation would first imply to recover all the leaves of the tree.

Finally, neighboring information is relatively fast to compute in quadtree only for nodes (through the relative position of children), while it is very efficient in half-edge for all entities (through the use of local pointers). For example, let us consider the operation of finding the star of a corner (resp. the star of a vertex), which is the set of corners linked to it by an edge. In the quadtree structure, this would imply to first find all leaves which contain the corner, then pick the next corner in the local corner list. However, this would only work for a conforming tree. In the half-edge structure, for any topology we can find the star by simply "walking around" the vertex, knowing its reference HE and following the pointers to local HE neighbors.

In this thesis, we use a tree structure for the mesh construction process. Once the topology is fixed, we use the half-edge structure to compute metrics and perform mesh *smoothing*.

7.4 Mesh smoothing

A common step in mesh generation is to optimize the mesh quality by relocating vertices in order to lower the elements distortion. In this thesis we perform an isoparametric Laplacian smoothing (Herrmann, 1976), where we try to move each internal vertex at the average of its star coordinates, applying the following formula:

$$v'_i = \frac{1}{N}\sum_{j=1}^{N} v_j \tag{7-11}$$

where v'_i is the new location of vertex v_i, N is the number of vertices in the star and v_j the coordinates of the j-th neighbor. This process is performed on each vertex of the mesh iteratively. This method is straightforward and efficient, the computation time varying linearly with the number of vertices in the mesh. However it can generate bad and even inverted quadrilaterals after a few iterations, and tends to make the elements size uniform. In our case it should hence be used with few iterations.

8
Related works

There are two categories of techniques to generate unstructured all-quadrilateral meshes: indirect and direct methods.

Indirect methods first create a triangle mesh and then convert it into a quadrilateral mesh. The triangle mesh is usually generated using the constrained Delaunay triangulation (Chew, 1987). Methods then differ in the step of converting the triangles into quadrilaterals. A common strategy is to subdivide the triangles as introduced by Catmull and Clark (1978): one triangle is divided into three quadrilaterals by adding a vertex at the centroid of the triangle and at the middle of each edge. Generated quadrilaterals tend to have a poor aspect ratio, and Liu *et al* (2011) improved the resulting mesh by proposing a post-processing method based on local topological operations. Another strategy to convert triangles into quadrilaterals is to combine them. Borouchaki and Frey (1998) joined pairs of triangles and divided the remaining ones using the Catmull subdivision. The Q-morph method (Owen *et al*, 1999) guides the combination of triangles using an advancing front strategy. Lee *et al* (2003) extended the Q-morph method to include open boundaries. Ebeida *et al* (2010b) introduced the Q-tran method, which joins a combination of triangles with a new subdivision scheme to generate good-quality quadrilaterals. Recently, Araújo and Celes (2014) proposed an algorithm to obtain good-quality mesh for complex crossing boundaries using a deferred constraint insertion strategy coupled with a new triangle mesh generation scheme. In general, indirect methods generate good-quality elements, but meshes suffer from a large number of vertices with a valence different from 4.

Direct methods generate quadrilaterals directly. We can group the direct methods into three categories: advancing front, domain decomposition, and grid-based methods.

Advancing front algorithms generate a first set of elements on the boundaries of the domain and then recursively project the elements toward the interior, moving the elements front until the domain is totally covered. As an example, the paving method by Blacker *et al* (1991) successfully generates meshes with good quality. However, advancing front methods tend to lack stability for complicated regions, especially at the meeting of the fronts, and usually require post cleanup operations.

Domain decomposition techniques subdivide the domain into a set of simple regions where conventional mapping methods can be applied to generate quadrilaterals. Talbert and Parkinson (1990) proposed a recursive algorithm that subdivides the complex domain until only basic shaped regions remain; the algorithm was then improved by Chae and Jeong (1997) and Nowottny (1997). Medial axis decomposition was introduced by Tam and Armstrong (1991) and uses the center of a maximal circle rolled through the area to subdivide the domain. Miranda and Martha (2013) proposed a template strategy for domain decomposition. These methods generate good-quality meshes but usually require heavy user intervention for complex domains.

Grid-based methods superimpose a grid to the domain and try to adapt it to the boundaries by modifying the grid geometry. It can be any type of grid, but quadtrees are often used because they allow a control on the local refinement, with a smooth size transition. To increase the refinement level around domain boundaries, Frey and Marechal (1998) proposed a simple size criterion. Ebeida *et al* (2010a) used a criterion based on the boundary curvature, assessed through an initial boundary discretization. The main challenge in such methods remains to adapt the grid points to the domain boundaries. For example, Yerry and Shephard (1983) intersected quadtree leaves with the boundaries, allowing the formation of non-quadrilateral elements. This is not a limitation to them since they aim at building a triangle mesh: they simply triangulate the elements at the end of the process. Rushdi *et al* (2015) recently presented a similar idea to create all-quadrilateral meshes: they first repel some quadtree grid points based on their distance to the boundary to guarantee a clean intersection; then they apply local midpoint subdivisions to turn elements into quadrilaterals. Another method is to remove all elements crossing the boundary and then reconstruct the missing layer with a post process. This can be done simply with a projection technique (Schneiders and Bünten, 1995) or using some reconstruction patterns (Liang *et al*, 2009). However, as stressed by Liang *et al* (2009), these techniques present some difficulties when the buffer layers face each other and must conform. In any case quadtrees have to undergo a conforming step before being usable as meshes (Schneiders *et al*, 1996). These methods are easy to implement and generate meshes of good quality. They also usually allow to bound maximum and minimum angles (Rushdi *et al*, 2015; Atalay *et al*, 2008). However, they can easily generate inverted elements and tend to gather lower quality elements on the boundaries.

All those methods tend to be dependent of the domain geometrical complexity. In the present work we aim at developing a method that would be robust for all kinds of boundaries. Moreover, the method should be usable in the industry and consequently have to be efficient and user-friendly. The quality of the resulting mesh should be adapted to FEM calculations requirements. We opted for a grid-based method using a new quadtree-based hierarchical data structure: the *extended quadtree*.

9
The extended quadtree

Our 2D quadrilateral mesh generation method is based on the creation of a new tree structure, called the extended quadtree: it includes new refinement patterns in the conventional quadtree structure to allow more flexibility for the domain boundary approximations.

In the extended quadtree, we allow splitting a node into six children. Two patterns are defined, and we call them S1 and S2 (“S” for “six”) (see Figure 9.1). The parent edges are still divided at their midpoints, so patterns S1 and S2 can easily neighbor the usual four-child pattern.

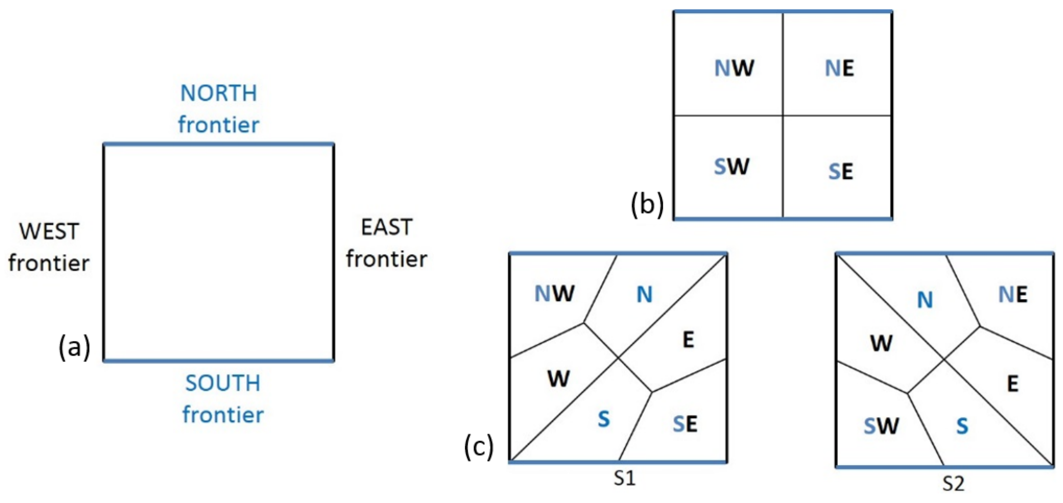


Figure 9.1: The different split patterns of the extended quadtree and the relative position of the children in the parent node. (a) Parent node with the definition of its frontiers; (b) usual four-child split pattern; (c) new six-child split patterns S1 and S2.

In our extended quadtree, each node stores eight pointers to its children, ordered as follows: {SW, SE, NW, NE, S, E, N, W}. Looking at Figure 9.1, we can see that neither the usual split pattern nor the new S1 and S2 patterns present all eight nodes. Any subdivided node will consequently present some null pointers. Nevertheless, the eight fixed positions keep the neighboring information organized. Inside the usual pattern, neighbor information is implicit: child NE has a west neighbor, NW, and vice versa. In the new S1 and S2 patterns, this duality is lost and thus we define neighbor information explicitly: in S1, child E has a west neighbor, S. However, S's east neighbor is SE in our definition. In other words, each child has its own local compass (Figure 9.2). Each node is given an attribute called the *Split_Pattern_Type* {USUAL, S1, S2} to help navigate through its children.

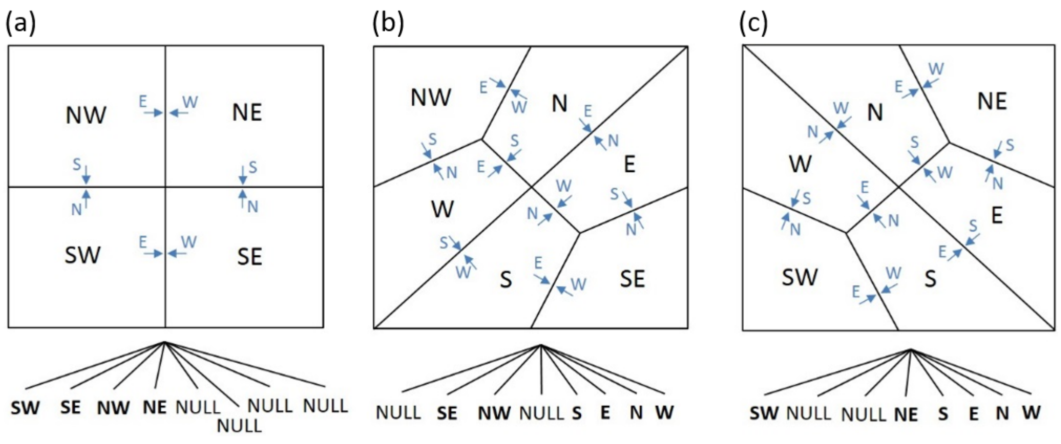


Figure 9.2: Definition of the split patterns' internal nodes with their neighboring relationships, and their corresponding pointer list. (a) Usual pattern; (b) pattern S1; (c) pattern S2.

The extended quadtree structure allows including new patterns very easily in the subdivision scheme. As for an example, we can define the transition patterns from Schneiders *et al* (1996) as extended quadtree nodes (Figure 9.3).

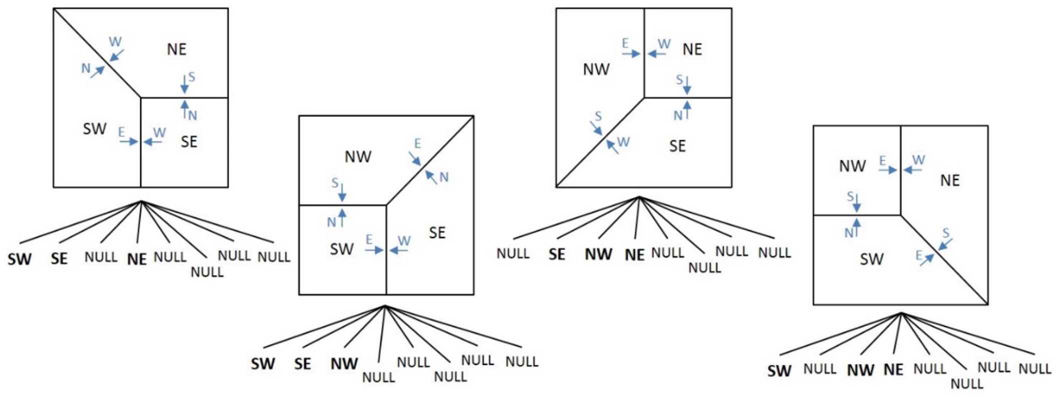

NE
SW
SE
SW SE NULL NE NULL NULL NULL NULL
NW
SW
SE
SW SE NW NULL NULL NULL NULL NULL
NW
NE
SE
NULL SE NW NE NULL NULL NULL NULL
NW
NE
SW
SW NULL NW NE NULL NULL NULL NULL

Figure 9.3: Definition of the extended quadtree transition patterns. From left to right, patterns T1, T2, T3, and T4 ("T" for "transition").

10
Adaptive mesh generation using the extended quadtree

We propose to exploit the flexibility of the extended quadtree to automatically build a quadrilateral mesh adapted to any geometrical constraint. The tree adapts to the boundaries dynamically during the refinement process. This section presents the whole procedure.

10.1
Overview

The input of the method is a set of polyline curves. Each curve corresponds to an ordered sequence of 2D points:

$$curve_i = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}, m \geq 2$$

If the curve is not closed, endpoints are considered as hard constraints: they must correspond to a corner in the final tree. We allow curves to cross each other, but we consider each part as a different curve and the intersection point as an endpoint shared by the intersecting curves. The polygonal geometry of each curve, defined by its sequence of 2D points, must correspond to a sequence of edges in the final tree, and this approximation has to meet a predefined error tolerance. The simple example in Figure 10.1 illustrates the approximation process sequence:

– We define the tree root as a square around the curves (Figure 10.1(a));
– We build a quadtree aligned to the curves, accommodating the tree geometry during its construction by moving the corners (Figure 10.1(b));
– We apply new split patterns S1 and S2 to ensure that the curves are approximated by edges only (Figure Figure 10.1(c));
– We make the tree conform (Figure 10.1(d)).

The extended quadtree refinement scheme depends on a tolerance regarding the accuracy of the final approximation. In Figure 10.1, this tolerance is high but already shows that the algorithm tends to refine more around high curvature areas. The final tree is then mapped into a half-edge mesh structure, then Laplacian smoothing is applied.

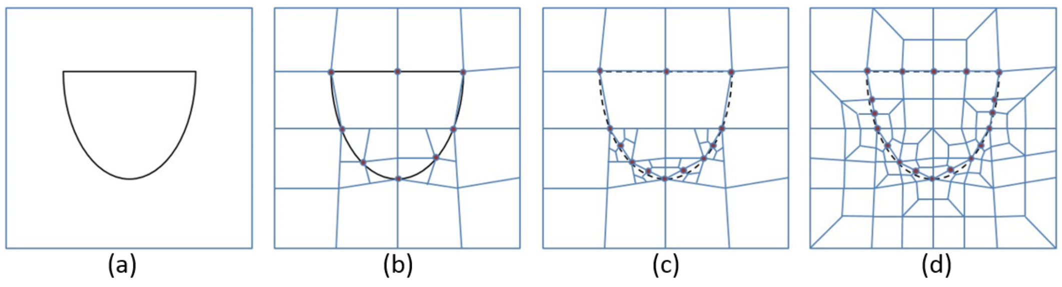


Figure 10.1: Workflow of the construction of the extended quadtree adapted to the given curve. (a) A curve and its bounding box; (b) construction of a quadtree with corners or edges on the curve; (c) addition of new refinement patterns to accommodate the whole curve geometry with node edges only; (d) balanced and conformed tree.

10.2 Tree geometry deformation

In this section we begin with defining the different destinations a corner can have onto a curve. Then we explain how we choose between those different possibilities and guarantee a smooth geometry deformation by introducing a local control parameter called the attraction zone. Finally, we explain the practical use of this parameter by presenting the global tree construction algorithm.

10.2.1 Destination points

Let us consider a single curve crossing a single node. For any corner of the node, there are three possible destinations on the curve:

– the intersection between the curve and one of the node edges (Figure 10.2, A);
– the projection of the corner on the curve (Figure 10.2, B);
– one of the curve's endpoints (Figure 10.2, C);

To choose which destination the corner should move to, we define a local controlling attraction zone.

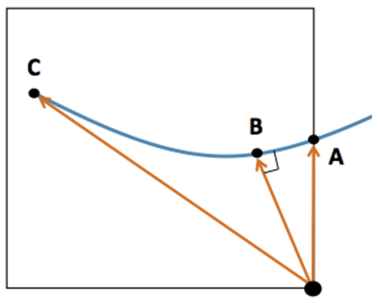


Figure 10.2: All considered destination points for a given corner onto a curve crossing a node.

10.2.2 The attraction zone

In order to ensure a smooth deformation of the tree geometry at each refinement step, we define a local parameter based on each node dimensions to control the movement of its corners. Indeed, the deformation of a node is transmitted to all of its children, and, thus, we must not allow big deformations at low refinement levels.

For each node corner, we define an area called the attraction zone: if the corner's destination point is inside the attraction zone, then the corner can move to its new position. This area corresponds to a proportion of the total node area around the corner: it is a quadrilateral which dimensions are given by a proportion of the two edges adjacent to the corner (Figure 10.3). This proportion is given by the <u>*ratio*</u> parameter:

$$ratio = ratio_{max} - \frac{ref_{level}(ratio_{max} - ratio_{min})}{ref_{max}} \tag{10-1}$$

where $ratio_{max}$ is the maximum value of $ratio$, defined by the user and superior to $ratio_{min}$; $ratio_{min}$ is the minimum value of $ratio$ and is fixed to 2, which cuts the edge at its mid length; ref_{level} is the node's current refinement level; and ref_{max} is the user-defined refinement level from which ratio reaches $ratio_{min}$.

$ratio_{max}$ defines the smallest proportion of the attraction zone, which will be considered at the lowest refinement level. The attraction zone then increases smoothly at each refinement step, reaching its maximum proportion for the refinement level ref_{max}. At this stage, the total area of the node is covered by its four corners' attraction zones. This means that, for any curve crossing the node, at least one corner should be able to move onto it. It is important to use the real length of the edges in the definition of the attraction zone to account for node deformation and avoid building an area that exceeds the limits of the node. Above the user-defined value of parameter ref_{max}, $ratio$ is maintained to $ratio_{min}$.

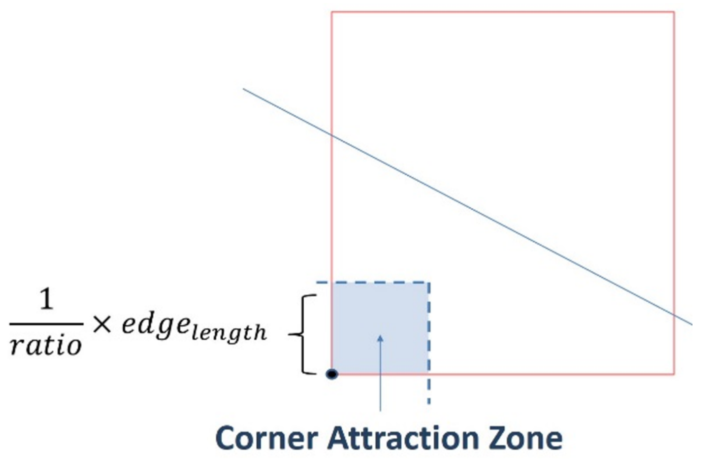


Figure 10.3: Definition of the attraction zone. The parameter ratio is defined in Equation 10-1.

When several destination points enter the attraction zone we use an order of priority to select the final destination:

1) curve endpoints. In practice we first treat all endpoints and construct a first tree. We then consider the curves' intermediate geometry;
2) corner's projection on the curve;
3) curve intersection with the node edges adjacent to the corner. In addition this destination is considered only when $ratio = ratio_{min} = 2$.

For each case, if several candidates enter the attraction zone we chose the closest one. Once a corner has been associated to a curve, its position is fixed and the corner is not allowed to move anymore.

The attraction zone is defined locally for each corner, in each node. Next section explains how we use it globally to control the tree refinement.

10.2.3 Tree construction algorithm

Using the definitions of the destination points and of the attraction zone we construct the tree in two main steps:

– Step 1: build a tree adapted to curve endpoints

The first step is to build a tree adapted to curve endpoints only. This includes curves' intersection points since we cut the curve in several sub-curves in this case. This first step ensures that all hard constraint points will be included as node corners in the tree. The pseudo-code for this step is presented in Figure 10.4.

```
Node list ← add tree Root
For each node in the list {
    If no endpoint inside the node: continue;
    Else {
        MoveCorners();
    }
}
```

Figure 10.4: Pseudo code for tree construction, step 1.

– Step 2: adapt the tree to the intermediate geometry

We adapt this first tree to the curve intermediate geometry, snapping corners and refining the tree when necessary. The procedure, described in Figure 10.5, starts with the root of the tree, then processes all children in a top-down scheme (low to high refinement levels) using a list with a pop_front/push_back strategy. If exactly 1 curve enters a leaf node, we try to move the corners to the curve. If several curves enter a node, we refine it. This ensures that two close curves will be properly separated. This also forces this step to be performed several times over sets of non-intersecting curves, otherwise the node containing the intersecting point would be endlessly refined.

```
For each curve set {
    Node list ← add tree Root
    For each node in the list {
        If the node has children: add children in node list and continue;
        If no curve intersects the node: continue;
        If more than 1 curve intersect the node: refine, add children in node list and
            continue;
        If 1 curve intersects the node {
            MoveCorners();
        }
    }
}
```

Figure 10.5: Pseudo code for tree construction, step 2.

Both steps use the MoveCorners function, presented in Figure 10.6. In this function, all four corners are candidates for moving onto the curve. We consider a heuristic method in which the best candidate:

– is free to move (no curve was already attributed to it);

– has its destination point inside its attraction zone;

– is the closest to the curve.

Corners move until no candidate remain for the node. At this point, there are two possibilities:

- no destination point remains in the node. In this case, the process is stopped for this node;
- at least one destination point remains in the node. In this case, the node is refined and its four children are added to the list.

Figure 10.7 illustrates the process for two different configurations.

```
(a) Find best candidate corner
(b) Move corner to its destination
(c) If any candidate remains: got to (a);
(d) If no candidate remains {
        If any intersection with curve(s) remains: refine, add children in node list and
            return;
        If no intersection with curve(s) remains: return;
}
```

Figure 10.6: Pseudo-code for function ***MoveCorners***.

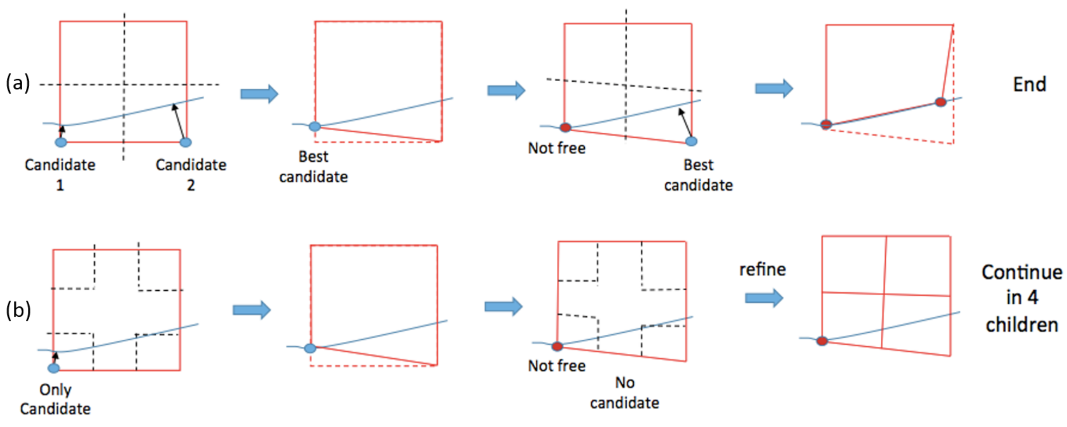


Figure 10.7: (a) local process for $ratio = ratio_{min} = 2$; (b) same situation for $ratio$=3.

In addition, an important aspect is that a corner can move only if all nodes sharing it have the same refinement level. This is to avoid creating non-quad elements, as illustrated in Figure 10.8(b). Before moving a corner, neighboring nodes are hence refined if necessary. If the current node has to be refined during this process, we add its children to the node list and stop processing it: the corner does not move until the moving parameter is verified in the concerned child. Figure 10.8(c) shows this process.

The refinement process stops when no corner can move in the tree, that is to say when the curves are entirely approximated by node edges or node diagonals. This implies that parameters $ratio_{max}$ and ref_{max} defining the local attraction zones are not strong constraints to the refinement process. A node can indeed be refined above ref_{max} if for example it still contains several curves

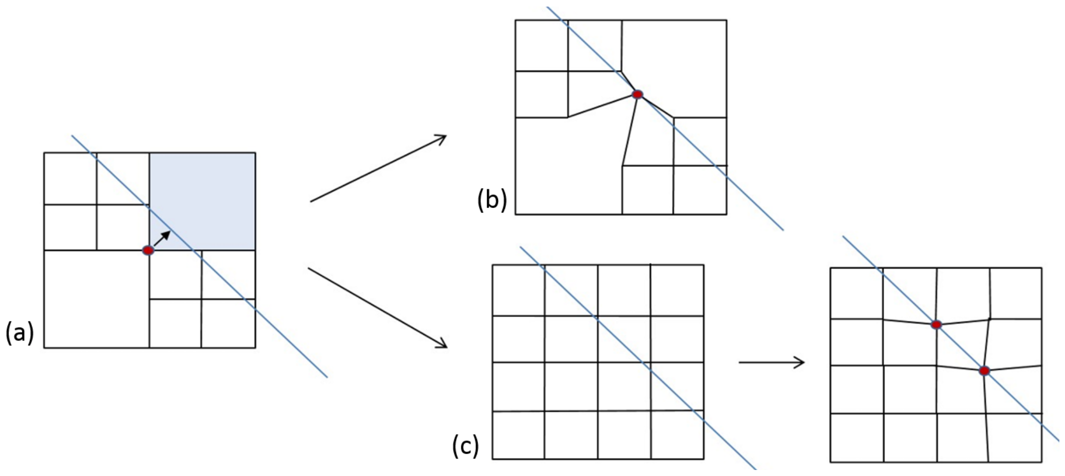


Figure 10.8: Local refinement process around a corner chosen to move on a curve. (a) Initial configuration: (blue) the current node; (red) the corner to move; (b) creation of non-quad elements; (c) our method: in the end, the initial candidate corner for moving did not move.

at this level. The process can also stop before reaching ref_{max} if the curves were entirely approximated in lower levels. Parameters $ratio_{max}$ and ref_{max} rather control the convergence of the procedure: the higher they are, the more refinement levels will be needed to reach large attraction zones, where corners are easily snapped to the curves. Thanks to this definition of the attraction zone and the dynamic snapping of the corners onto the curves, we do not need to pre-define a precise refinement level criterion.

In the end, there are two main configurations for a curve crossing a node:

– the curve corresponds to an edge (two adjacent corners are attributed to it);

– the curve cuts the node through its diagonal.

The second configuration is a problem because it will create two adjacent triangles in the final mesh. We use patterns S1 and S2 of our extended quadtree to resolve this configuration.

10.3 Application of patterns S1 and S2

Patterns S1 and S2 allow aligning children with the diagonal of the parent node and applying them at the end of the tree deformation process results in a tree where all geometrical constraints are approximated by node edges. We do not apply them during the tree deformation because it appears that deformation of those patterns leads to badly shaped elements.

However, there is a specific configuration where we have to apply S1 and S2 during the tree deformation process. Indeed, a refinement problem can appear at the intersection between curves: the algorithm can easily get involved in an endless refinement process illustrated in Figure 10.9, preventing the algorithm from reaching an end.

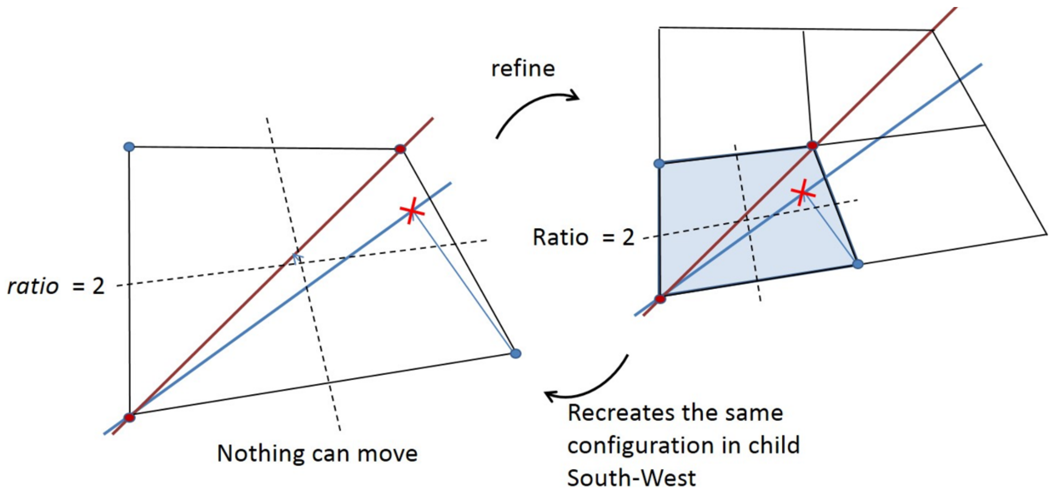


Figure 10.9: The curve–curve intersection degenerate configuration. In this configuration (left), no corner can move to the blue curve even for the parameter ratio set to $ratio_{min}$: the projection of the free corner to the curve will always fall outside its attraction zone. The node is consequently cut with the usual four-child pattern, and the left configuration is repeated endlessly in the SW child (blue node on the right).

This problem can be handled with refinement patterns S1 and S2, as explained in Figure 10.10. To simplify we show it in a non-deformed element. In this case, we apply pattern S1 or S2 as soon as we detect the degenerate configuration, to allow further refinement inside them as shown in Figure 10.10(b).

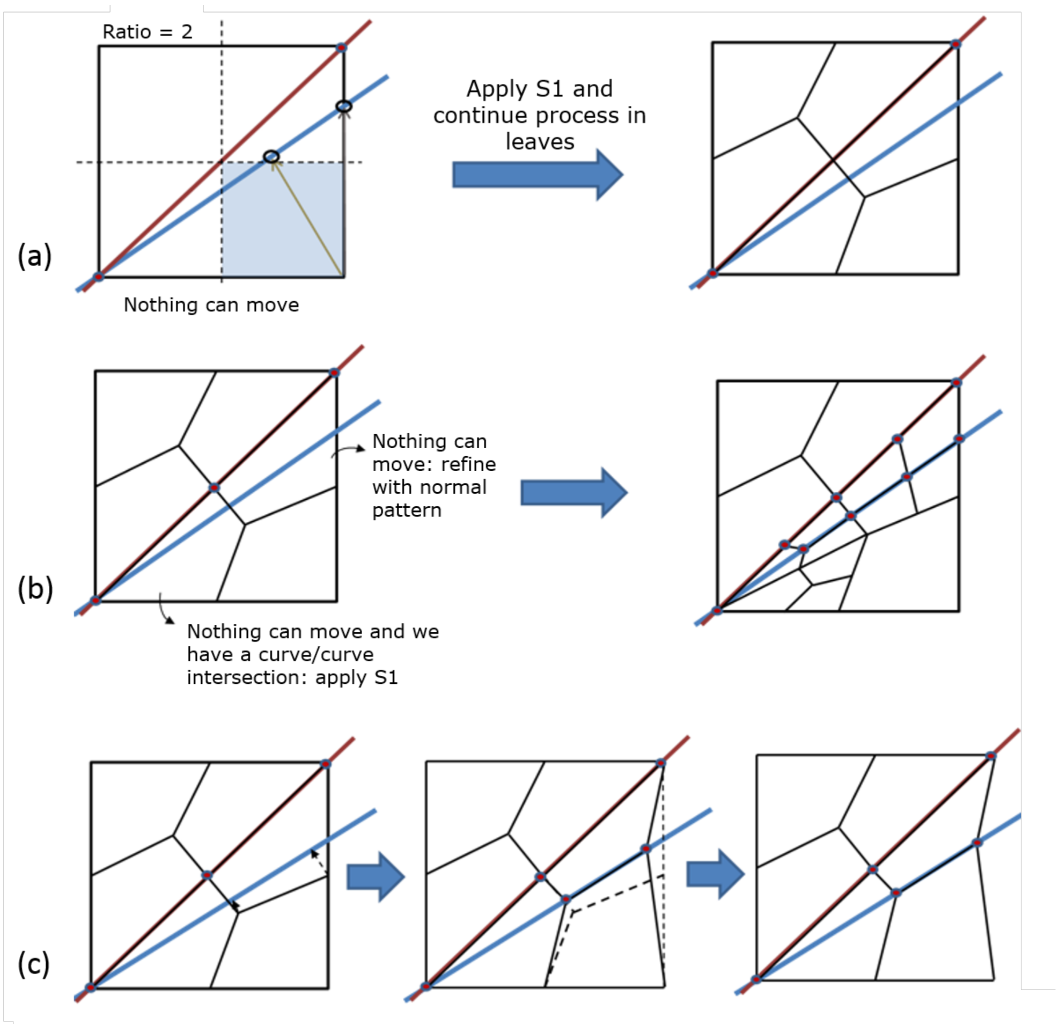

Ratio = 2
Apply S1 and continue process in leaves
(a)
Nothing can move
Nothing can move: refine with normal pattern
(b)
Nothing can move and we have a curve/curve intersection: apply S1
(c)

Figure 10.10: Curve-curve intersection configuration and its resolution. (a) The initial configuration presented in a perfect square to simplify. Pattern S1 is applied because no corner could be attracted to the blue curve. (b) The process continues in children where normal subdivision pattern or S1 or S2 patterns can be applied again. (c) If the angle between the curves was a little wider initially, there is no need to refine the children because corners can be directly attracted to the blue curve. (b), (c) The element presenting the blue curve in diagonal can be applied a pattern S1, we did not show it here for clarity of the figure.

10.4 Curve approximation accuracy

To ensure a good adaptation of the final tree edges with the curves, we can calculate a simple error as the maximum distance between the edge and the curve, as shown in Figure 10.11. If the error is higher than a user defined tolerance, the node has to be refined. This presents the great advantage to automatically refine more around high curvature areas. This error is calculated each time an edge or a node diagonal is associated with one curve, all along the process.

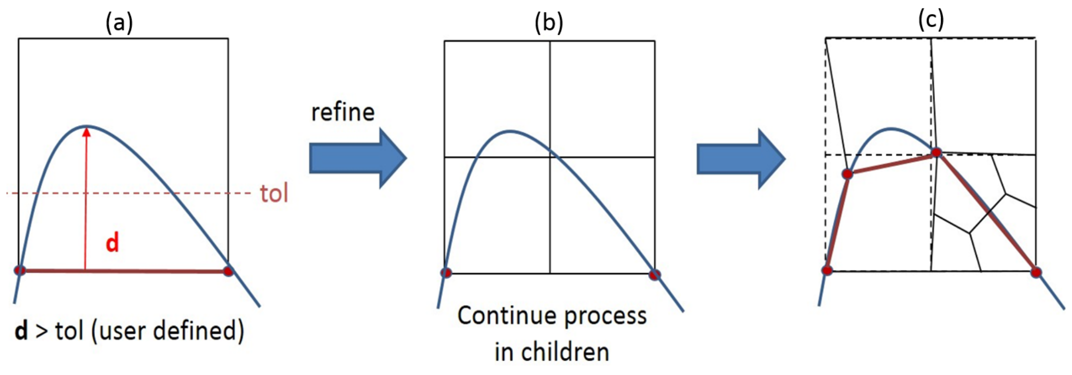


Figure 10.11: Process to handle curvature. (a) Initial approximation and definition of distance d; (b) refinement with the usual pattern; (c) final approximation.

In addition, we use this parameter to detect *fake intersections*: artifacts that can appear at lower resolutions. In computational geometry, two points are generally considered equal if their coordinates differ from less than $1.e^{-7}$m. However, in our approximation process, we sometimes do not want to differentiate points so precisely. Let us consider for example the configuration of Figure 10.12. If the user defined the tolerance parameter at 10m, the detected intersection must be considered as an artifact, the curve being approximated by the edge c4-c2: the intersection point is considered as being c4. However if the tolerance was set to 0.5m, the intersection point and c4 are considered as different and the process results in refining the node or maybe attracting c3 to the curve.

Consequently each time we detect an intersection between a curve and a node edge, we apply the following rule:

***If** the intersection point is close to an attracted corner **and** the error between the curve and the concerned edge is lower than the tolerance, **then** reject the intersection.*

An intersection point is "close to an attracted corner" if they share the same edge and the distance between the corner and the intersection is inferior to $edge_{length}/2$.

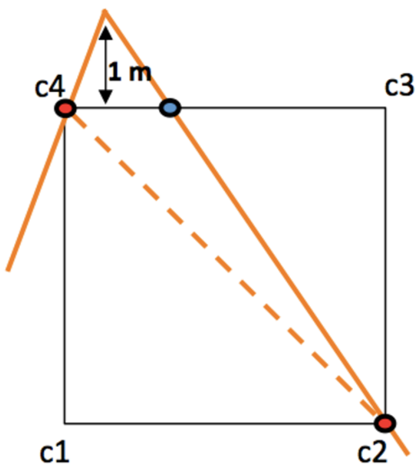


Figure 10.12: True intersection or artifact? Red points are the node's corners already attracted to the curve. The blue point is the remaining intersection curve-node. The dot line corresponds to the possible final approximation.

With the tolerance parameter, we guarantee that the curve will be approximated precisely. However, we do not yet have guarantees on the quality of the nodes. Next section tries to address this issue.

10.5 Node quality improvement

At this point of the process, the extended quadtree is capable of aligning itself automatically with all the curves, accommodating curves crossing a node diagonal and resolving curve–curve intersection problems using patterns S1 and S2, thus fully representing the curves' polygonal geometry with tree edges, with a satisfying tolerance.

However, one bad configuration appears when three consecutive corners are attracted to the same curve. This can easily happen when ratio reaches $ratio_{min}$. Indeed, the angle formed by those three corners can be very wide. This is a problem because once the corners are fixed, they will never have the possibility to move again; thus, this wide angle can never be improved by further smoothing operations. Consequently, we apply a post-processing technique to overcome this issue: at the end of the tree construction, when such a configuration is detected, the central corner can be freed and the curve is now approximated by the diagonal of the node. Pattern S1 or S2 can then be applied to fit the curve.

To ensure that the local approximation tolerance is respected, we apply the process of freeing the central corner only when the distance from the central corner to the concerned diagonal is lower than the set tolerance, and while it is not, the node is refined with the usual pattern (Figure 10.13).

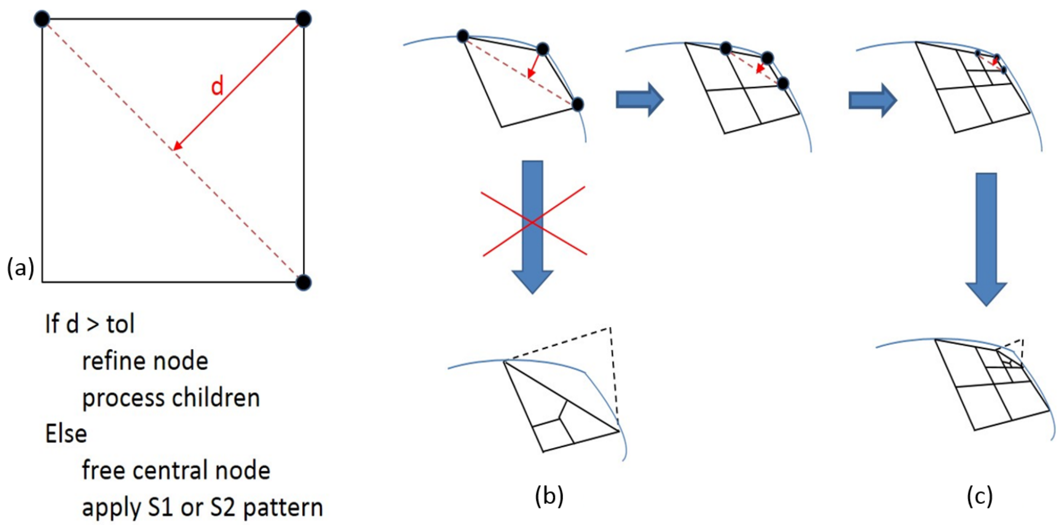


Figure 10.13: Keeping track of the approximation error. (a) Initial configuration where the three black corners are associated to a same curve, and pseudo-code; (b) the refinement with pattern S2 generates a bad approximation of the curve; (c) our process: the approximation of the curve meets the initial tolerance.

10.6 From the extended quadtree to the conform mesh

In the quadtree structure, hanging corners appear on edges shared by nodes of different refinement levels. In order to obtain a conform mesh we must get rid of those corners. To do this, we first balance the tree, meaning that we refine nodes locally until two adjacent nodes can only have 1 level of difference, so only one hanging corner will remain on their frontier. Then we apply common transition patterns described earlier (see Figure 9.3, chapter 9). We use an efficient algorithm proposed by Ebeida *et al* (2011), adapted from Schneiders *et al* (1996). The method first marks the corners of the transition patterns with a flag: *active* or *inactive* (Figure 10.14(a)). Then, three consecutive levels $\{n, n-1, n-2\}$ with $n \in [2, \infty]$ are processed, beginning with the highest levels and then recursively treating lower ones. This is valid since unconformity can only appear from level 2.

Level $n-1$ first marks the corners lying on level $n-2$ edges as *edge corners*. In the extended quadtree, we have to additionally mark the central corner of patterns S1 and S2 as edge corners (Figure 10.14(b)). Then, if an edge corner at level $n-1$ is part of a non-conforming edge at level n, the corner receives the *active* flag. Else, it is set *inactive* (Figure 10.14(c)). Finally, transition patterns are inserted by matching their active corners with the tree active corners (Figure 10.14(d)).

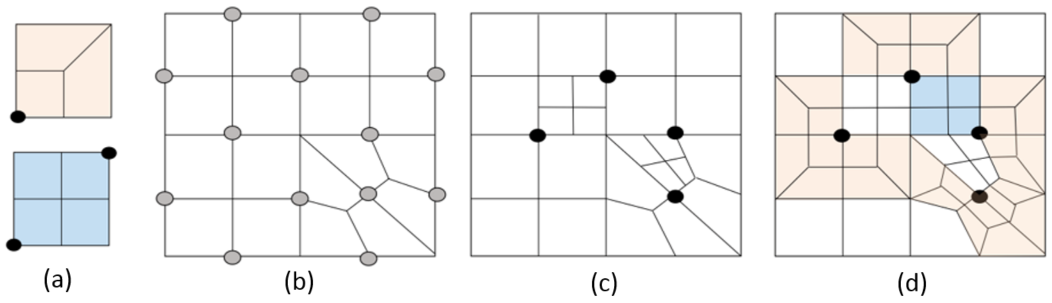


Figure 10.14: (a) Transition patterns with fixed active corners in black. (b) Tree configuration with edge corners in grey; (c) edge corners on non-conforming edges are set active (black points); (d) conform extended quadtree.

The final step of our method is to smooth the mesh in order to optimize the elements quality.

10.7 Mesh smoothing

Our method implies local deformations of the tree that can result in the creation of highly distorted elements. The distortion of an element lowers the FEM calculations accuracy and can invalidate the mesh, even if a few elements are concerned. Thus, we map the tree to the half-edge mesh structure in order to efficiently apply Laplacian smoothing on all the mesh vertices. The mapping is straightforward, since the tree topology and geometry correspond exactly to the mesh topology and geometry. To compute the opposite half-edge information we applied the algorithm from Lage *et al* (2017). In the Laplacian smoothing, we apply some restrictions specific to our problem:

– hard constraint points and root corners are fixed: they are not affected by the Laplacian smoothing;
– vertices on curves and on the root's edges can only move along their curve or edge.

Next section quickly summarizes the whole meshing process, before we present some applications.

10.8 The final algorithm

Figure 10.15 shows the algorithm for the proposed automatic all-quadrilateral mesh generation method.

```
1. Approximate hard constraint points only
        Input:
            - all hard constraint points
            - $ref_{max}$, $ratio_{max}$: specific for endpoints
        Output: a first extended quadtree
2. Separate curves in groups of non-intersecting curves
3. For each group {
      Add curve intermediate geometry
        Input:
            - current group point sequence
            - $ref_{max}$, $ratio_{max}$: specific for intermediate geometry
        Output: updated extended quadtree
4. Apply local node quality improvement
5. Apply patterns S1 and S2 on all nodes where a curve corresponds to a
   diagonal
6. Conform the tree
7. Map the extended quadtree into a half-edge mesh structure
8. Apply global Laplacian smoothing (10 iterations)
```

Figure 10.15: Algorithm for the proposed all-quadrilateral mesh generation method.

Steps 1 to 5 correspond to the construction of the tree. In step 2, we manually separate the curves into groups following this rule: curves in the same group do not intersect each other; curves from different groups can intersect each other. In step 3, we construct the extended quadtree by iteratively updating it, adding new constraints to the grid. We apply patterns S1 and S2 at the end of the tree construction for nodes where the curve corresponds to a diagonal (step 5). In steps 6 we conform the extended quadtree, before mapping it into a half-edge mesh structure in step 7. Finally step 8 applies a Laplacian smoothing to improve global element quality.

Next chapter presents and discusses our results.

11 Results

We present the application of our method on a set of geometries. First, we study the quality of meshes created on three hand-made models, each presenting a different type of geometrical complexity: varying curvature, curve intersections and multi-scale resolution. Second, we examine the robustness of the method on more complex models, involving many curves with intersections at sharp angles and thin regions. Then, we present the performances of the algorithm in terms of computation time. Finally, we discuss some aspects of the method before showing an example of its application in a geomechanical software currently in development.

11.1 Mesh quality on three hand-made models

We created three test cases to study the behavior of the method on different geometries. The *smooth star* (Figure 11.1) allows observing the refinement around varying curvature areas. It corresponds to one single closed curve (no endpoints). The *Olympic rings* (Figure 11.2) shows the behavior of the algorithm on curve intersections. Each ring was cut into different curves at the intersection points, and they were manually separated into four groups where curves do not intersect each other. The *Brazil map* (Figure 11.3) presents details at a small scale. To ensure the insertion of hinge points, we cut the contour into several curves: if three consecutive points form an angle inferior to 90° or superior to 270°, the central point is set as a new curve endpoint. Table 11.1 shows the mesh quality in the three cases.

We see that the quality of the elements is good as regards our quality criteria. For each model, a very small proportion of the elements are above the limit of $||\vec{f_q}|| = 1.5$. In addition, the quality is consistent at the different scales and remains good in the case of narrow regions as can be seen particularly in Figure 11.3, detail A. Lower quality elements are mainly due to the conforming transition patterns. Moreover, when neighboring, those patterns can create vertices with a high valence: we can see vertices with a valence 8 in Figure 11.1 for example. They also present an aspect ratio close to 0.5, explaining the mean aspect ratio of 0.6 for all meshes. Patterns S1 and S2 introduce elements

Model	Tree depth	Number of elements	Largest edge (m)	Smallest edge (m)	Mean quality			Worst element quality			% of ||fq|| above 1,5
					||fq||	Jacobian	aspect ratio	||fq||	Jacobian	aspect ratio	
Smooth star	8	2720	300	22	0.63	0.62	0.67	1.55	0.18	0.30	0.07
Olympic rings	8	4459	600	22	0.65	0.60	0.66	1.54	0.26	0.27	0.08
Brazil map	11	39116	600	2	0.65	0.60	0.67	1.86	0.26	0.26	0.08

Table 11.1: Quality results for the given mesh examples. The edge length of the biggest and smallest elements in the mesh is an approximation deduced from the tree depth and the domain size, considering the elements as perfect squares, to give an idea of the scale variation within models.

of good quality along the curves ($||\vec{f_q}|| < 1$), but when neighboring, they can create vertices with a valence of 6.

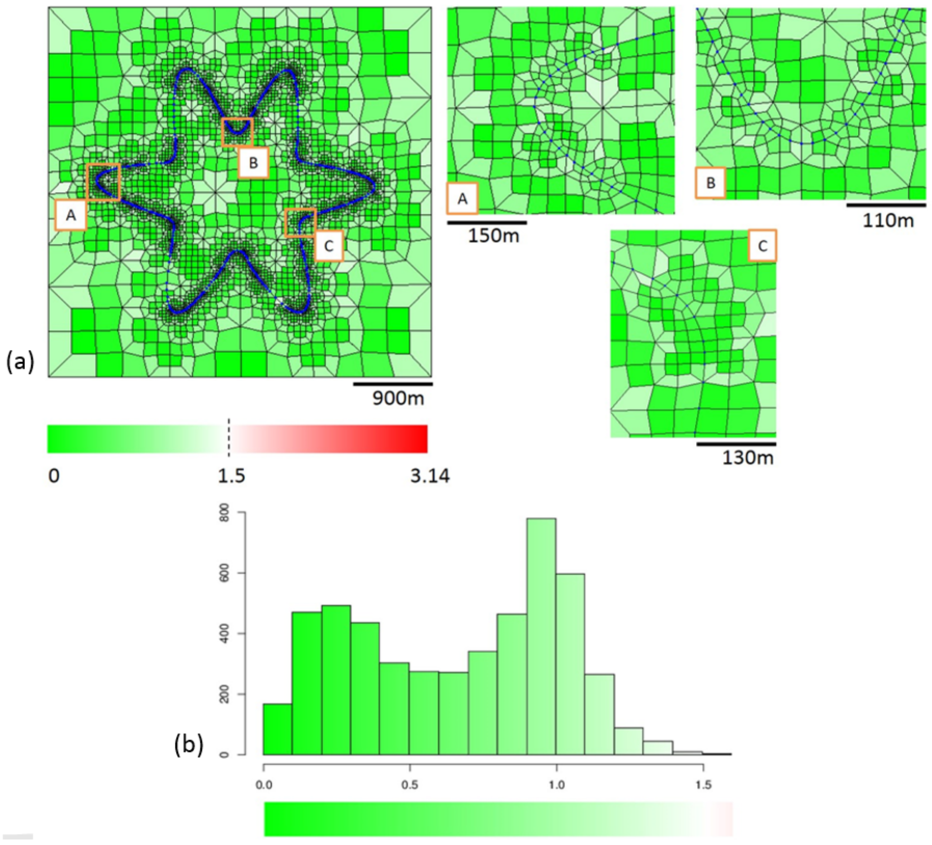


Figure 11.1: Smooth star. (a) Global view and details; (b) histogram of $||\vec{f_q}||$. There is only one closed curve and no endpoints. Parameters: $ref_{max} = 7$; $ratio_{max} = 10$; $tolerance = 5m$.

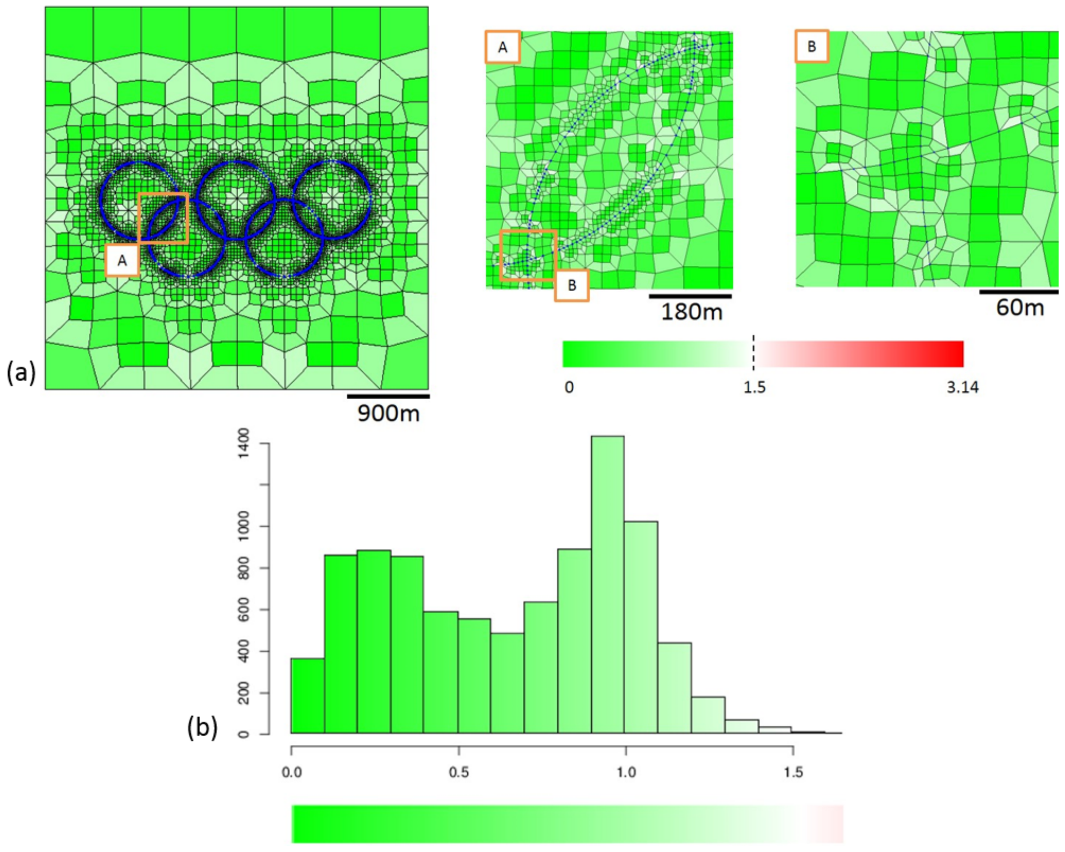


Figure 11.2: Olympics rings. (a) Global view and details; (b) histogram of $||\vec{f_q}||$. There are 15 curves divided into four groups where curves do not intersect each other. The parameters for the endpoints are the same as the parameters for the intermediate geometry: $ref_{max} = 8$; $ratio_{max} = 10$; $tolerance = 5m$.

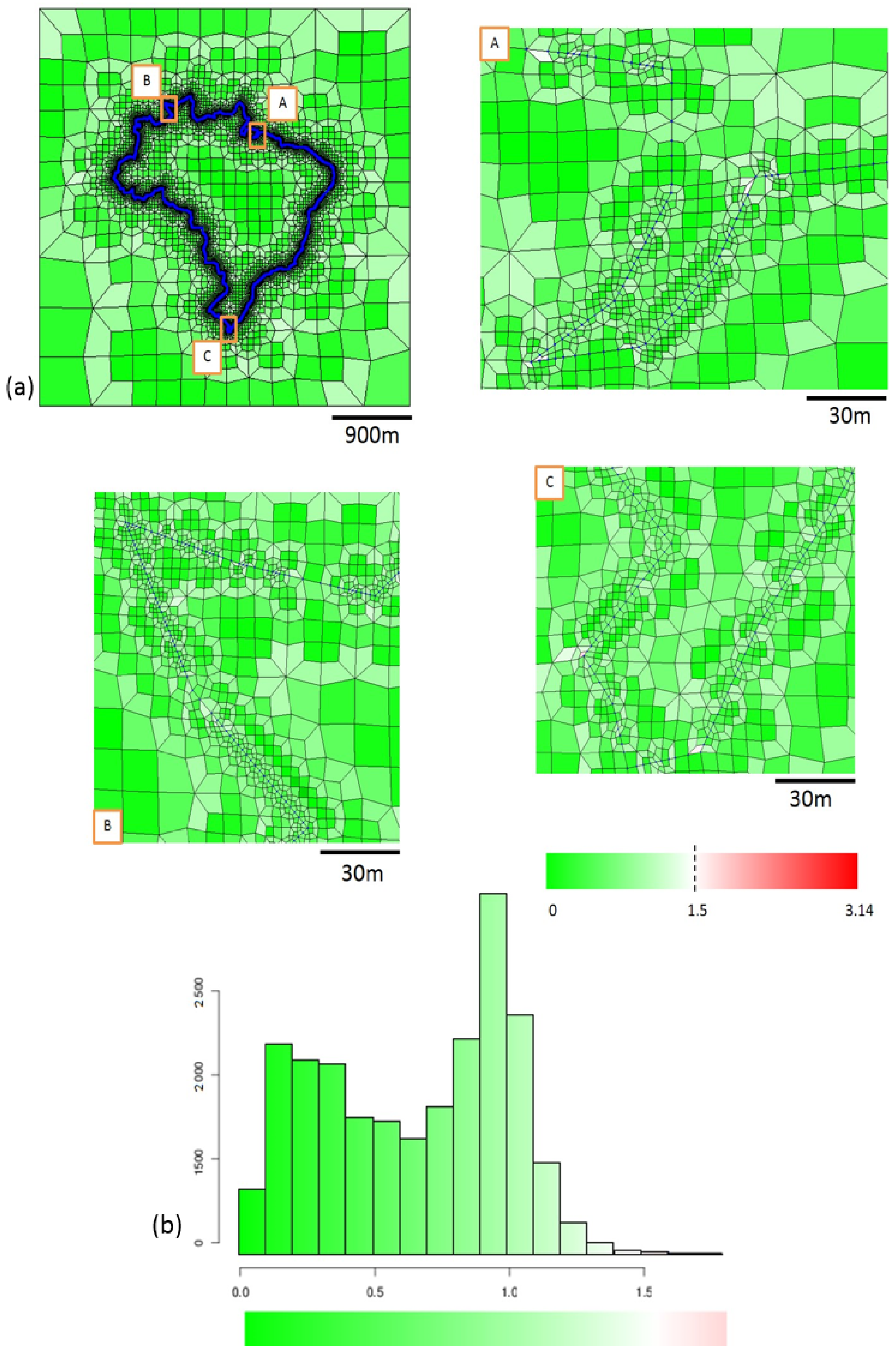


Figure 11.3: Brazil map. (a) Global view and details; (b) histogram of $||\vec{f_q}||$. There are 75 curves divided into two groups where curves do not intersect each other. The parameters for the endpoints are the same as the parameters for the intermediate geometry: $ref_{max} = 10$; $ratio_{max} = 20$; $tolerance = 5m$.

11.2 Complex geometries

To better assess the robustness of the process as regards to the curve geometry, we first generated a multiple curve intersection model. In Figure 11.4, we insert one after the other curves that intersect at a same point, with a random angle. By adding the curves one by one we guarantee that we can treat the intersection locally as only a two curve intersection: at the intersection point, we can look at the configuration as being the intersection of the new curve with any previous one. Patterns S1 and S2 help resolving degenerated cases as explained in Section 10.3, Figure 10.10. The algorithm finds a solution even in the case of very sharp angles.

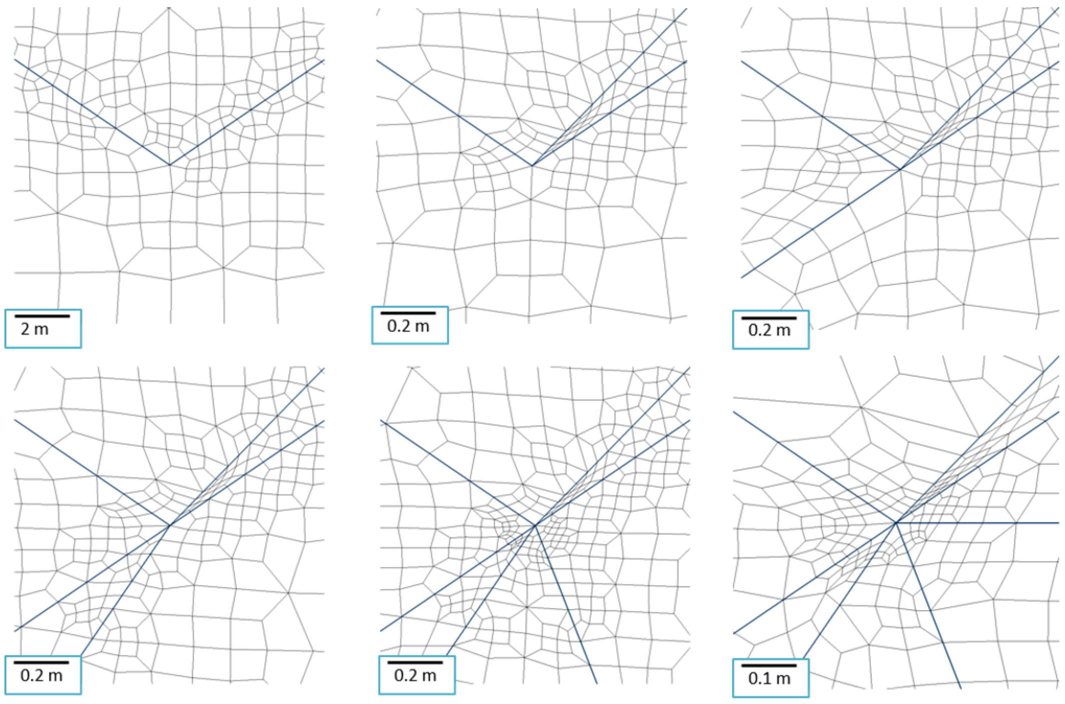


Figure 11.4: Multiple curve intersections. From top left to bottom right we added one curve after the other in the approximation process. Results show the final mesh.

We also applied our method on two structural models built from real interpretations of horizons and faults. Figure 11.5 shows the curves of each model. Following the classification from Pellerin *et al* (2015), we can describe the complexity of each model as follows:

- Model 1 presents *thin layers*: in its center, layers have a spacing of less than 5m, for a lateral extension of several km.
- Model 2 presents a complex network of *small displacement faults* and *Y faults*.

According to Pellerin *et al* (2015) the complexity of a structural domain also increases with the number of features, here the number of curves. Model 1 presents 56 curves and model 2 is made of 252 curves. This makes those models very challenging for any meshing method.

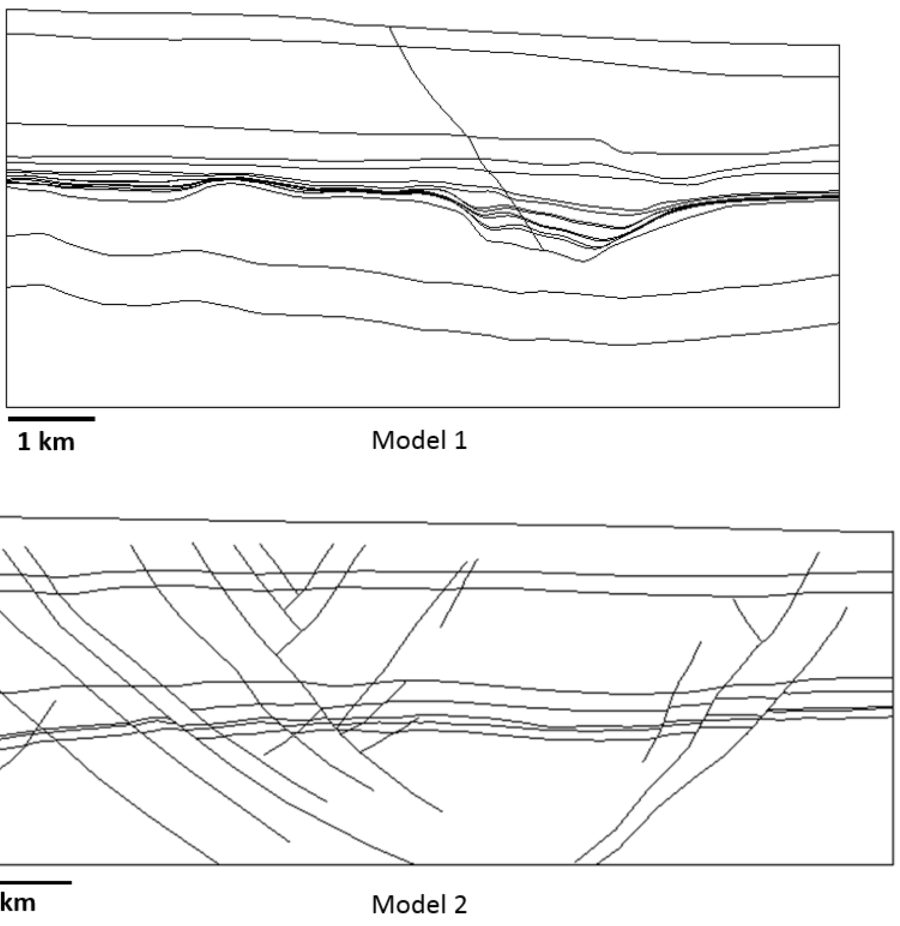


Figure 11.5: Models built from real interpretation of horizons and faults.

The number of curves in each model makes it difficult to tune the parameters for each curve, and also implies a substantial effort to separate the curves in groups of non-intersecting objects. Here, we simply gave the same parameterization for all curves and hard constraint points, and added each curve one by one in the tree. Figure 11.6 and 11.7 show the mesh generated for each model, along with some details. For a better visualization, we removed elements outside the model frontiers. We present the quality in Table 11.2.

The mean quality of both meshes is close to the one of simpler models of Table 11.1. However, the worst elements are more distorted, probably due to the sharp angles imposed by the complex geometries. Nevertheless, the jacobian remains positive for all elements. The thin layers configuration of Model 1 highlights one of the possible limitations of the method. Indeed, the algorithm refines the tree until properly separating the curves, maintaining a

Model	Number of elements	Largest edge (m)	Smallest edge (m)	Mean quality			Worst element quality			% of ||fq|| above 1,5
				||fq||	Jacobian	aspect ratio	||fq||	Jacobian	aspect ratio	
1	114,916	250	0.1	0.66	0.64	0.63	2.52	0.14	0.14	1.18
2	31,186	210	6.5	0.70	0.61	0.64	2.40	0.13	0.14	1.70

Table 11.2: Quality results for models 1 and 2.

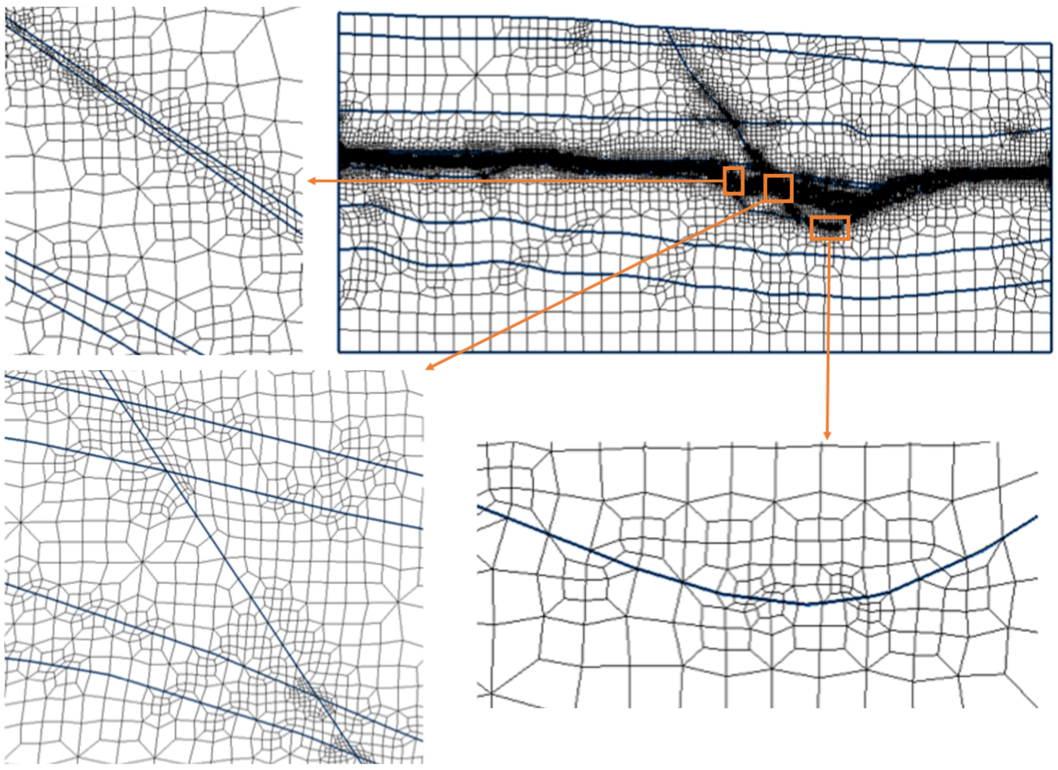


Figure 11.6: Mesh generated on Model 1.

good aspect ratio while refining: this can lead to the creation of a huge number of elements in thin regions. As thin layers extend in the whole model, the total number of elements is large in Model 1. As a consequence, the FEM simulation time can be too long to consider this mesh in practice. One of the drawbacks of the method is thus the lack of local control: in this situation, it could be advantageous to lose in elements' quality in the thin layers, while gaining in computation time when performing simulations.

Both meshes were created in a few seconds. In the next section, we study the relation between the number of elements and the mesh construction time.

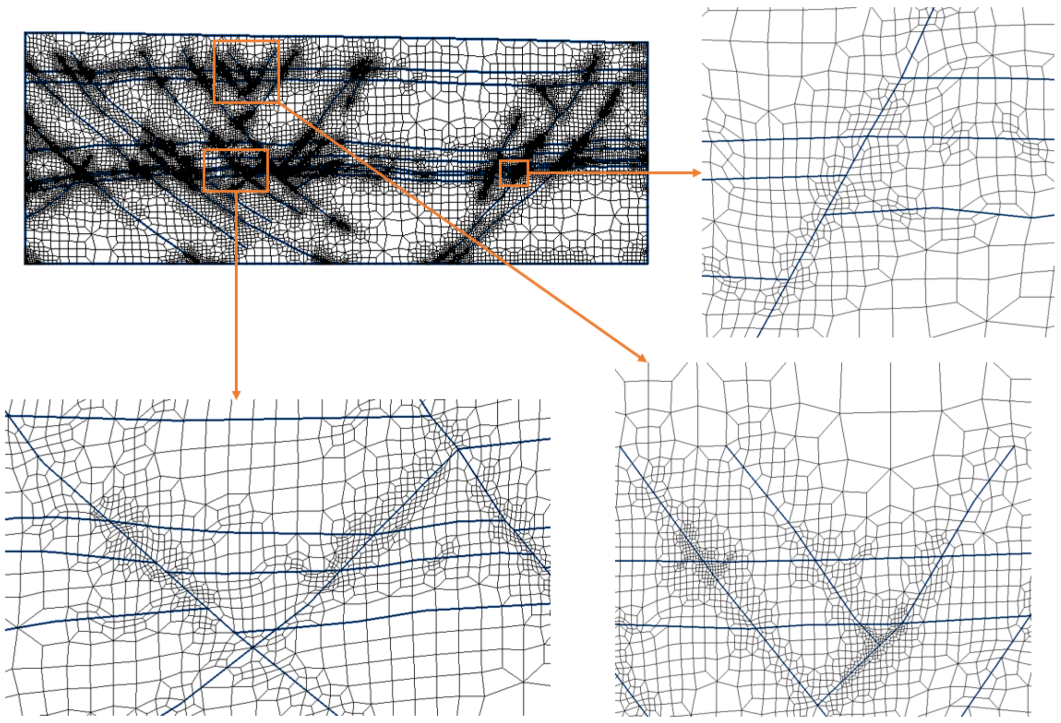


Figure 11.7: Mesh generated on Model 2.

11.3 Method's performances

We tested the method's performances on a test case made of test case made of 140 curves, all non-crossing arc circles with a random spacing (Figure 11.8(a)). We built different meshes on this model, modifying parameters ref_{max}, $ratio_{max}$, and the tolerance. In this way, we generated 6 meshes with an increasing number of elements.

We first measured the computation time of the approximation process, i.e., steps 1 to 5 in our workflow (Figure 10.15, section 10.8), corresponding to our contribution: the construction of the extended quadtree. We list the parameters used to generate the different trees in Table 11.3. We plot the performances in Figure 11.8(b). The experimental results show that the process is fast and that the computation time is linear with the increasing number of elements in the tree.

We then measured the computation time of the whole process, i.e., all steps of our workflow. Table 11.4 shows the details of the performance results for our 6 meshes. The steps of balancing and conforming the extended quadtree increase drastically the number of elements generated. Table 11.4 shows that the longest process is the final smoothing step. Figure 11.9 plots the results and shows that the final computation time is still linear with the increase in

Tree	ref_{max}	$ratio_{max}$	Tolerance	Number of generated elements
1	5	10	1	48,586
2	8	10	1	60,997
3	9	10	1	96,293
4	10	10	1	170,632
5	10	20	0.3	221,906
6	11	10	1	310,476

Table 11.3: Parameters used to generate the different extended quadtrees. Parameters are the same for the curve endpoints and for their intermediate geometry.

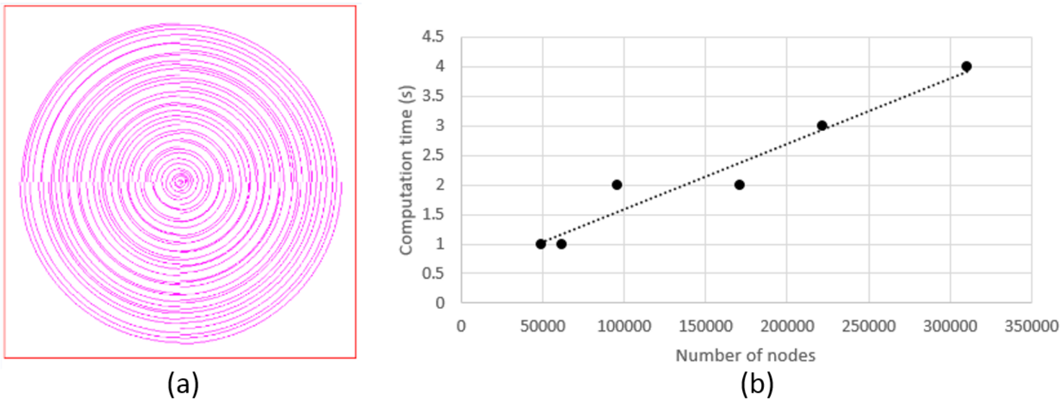


Figure 11.8: Test case and performance results. (a) The test case, made of 140 non-crossing arc circles with random spacing. (b) Computation time as a function of the number of elements generated in the different trees, showing linear evolution. Data points correspond to the trees presented in Table 11.3.

number of elements in the mesh. The whole process does not exceed 1min even for meshes with a large number of elements.

Mesh	Number of generated elements	Time for building the non-conforming extended quad tree (s)	Time for balancing and conforming the tree (s)	Time for smoothing the mesh (s)
1	93,950	1	1	3
2	122,677	1	2	4
3	227,885	2	2	8
4	470,378	2	6	13
5	569,769	3	6	17
6	944,885	4	9	30

Table 11.4: Details of the computation time for the different parts of our workflow. The parameters used to obtain these meshes are given in Table 11.3.

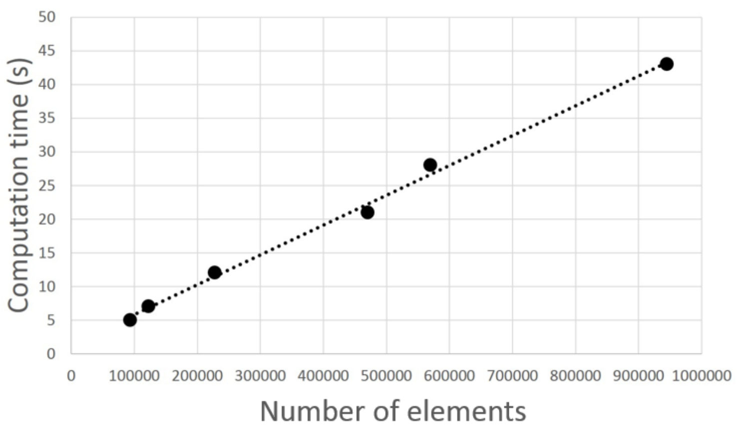


Figure 11.9: Performance results for the whole process show that the computation time is linear with the increase in the number of elements in the final mesh. Data points correspond to the meshes presented in Table 11.4.

11.4 Limitations

In this section we list some limitations of the method. First, the deformation of the tree nodes is not bounded to a certain angle, making it impossible to give robust guarantees on the final mesh quality. However, the node quality improvement step presented in section 10.5 should prevent the formation of degenerated elements. All our results showed positive jacobian, even for complex structural models.

The quadtree-based approach implies the use of conforming transition patterns, which present elements of low aspect ratio and vertices of valence

3, and valence 8 when neighboring. Patterns S1 and S2, meanwhile, introduce vertices of valence 6 when neighboring.

Another limitation of the method is the presence of heterogeneities in the refinement level that are not justified by the local curvature. This is because vertical, horizontal, and 45° lines are easy to approximate with our process and tend to require less refinement steps than lines with other angles to the horizontal. Figure 11.10 illustrates this: we built an extended quadtree on a circle to show the behavior of the algorithm along a curve with constant curvature. We can see that horizontal, vertical, and 45° lines are approximated with a lower refinement level than other inclinations. This can be a problem in simulations because the precision of the calculations will vary along the curve for a same level of details.

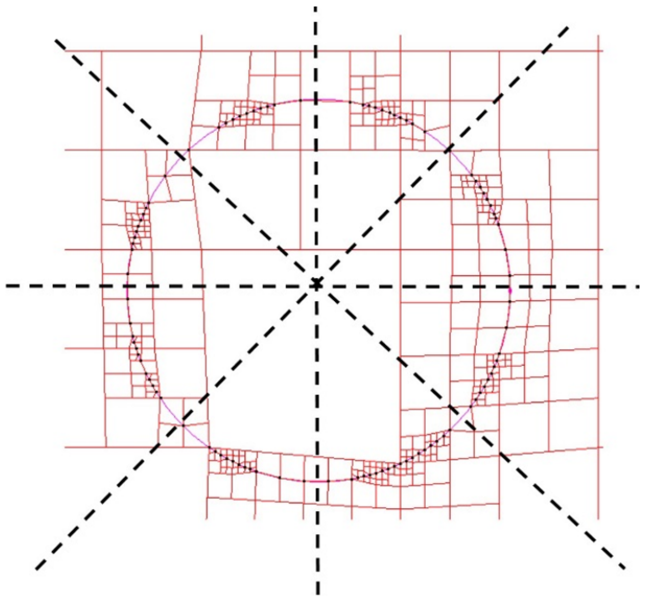


Figure 11.10: Extended quadtree built on a circle shows the preferential orientations where the process quickly approximated the curve: black dot lines, corresponding to the horizontal, vertical, and ±45° lines.

Our parameters can also be hard to tune on a large number of curves.

Finally, the lack in local control is both a strength and a limitation to the method. In one hand, the total automation of the refinement process avoids the laborious task of defining precise refinement parameters and allows a fast and easy construction of meshes even in complex structural domains. In the other hand, as shown in the results section 11.2, some configurations may trigger the creation of a large number of small elements, which can, in the worst case, make the mesh unusable in practice. This paradox highlights the difficulty of creating an automatic method which attends all kinds of configurations.

We present in the next section the application of the method in a software currently in development, to show how some of those limitations can be handled in practice.

11.5
Application: full-automatic mesh generation for the SABIAH software

SABIAH is a software currently in development for the modeling of reservoirs and the simulation of various problems like critical region deformation or other geomechanical threats. It provides support for stress and pore pressure non-uniform distributions, heterogeneous lithology and multiple depleted and injected regions, among others. The tool is built as a wizard where the user enters a series of data related to the problem, then runs the simulation without having to handle nor the mesh generation neither the solver management. This makes it easy to test many configurations without having to pass the problem between different softwares for each expertise field.

The mesh generation is meant to be full-automatic and uses the extended quadtree mesh presented in this thesis. First, the reservoir domain is defined through the importation of a set of polyline curves. Curves are automatically ordered to define regions, in which different properties can be attributed. Curves of the domain also receive a type: *reservoir*, *horizons*, or *sides*. Figure 11.11 shows such a model.

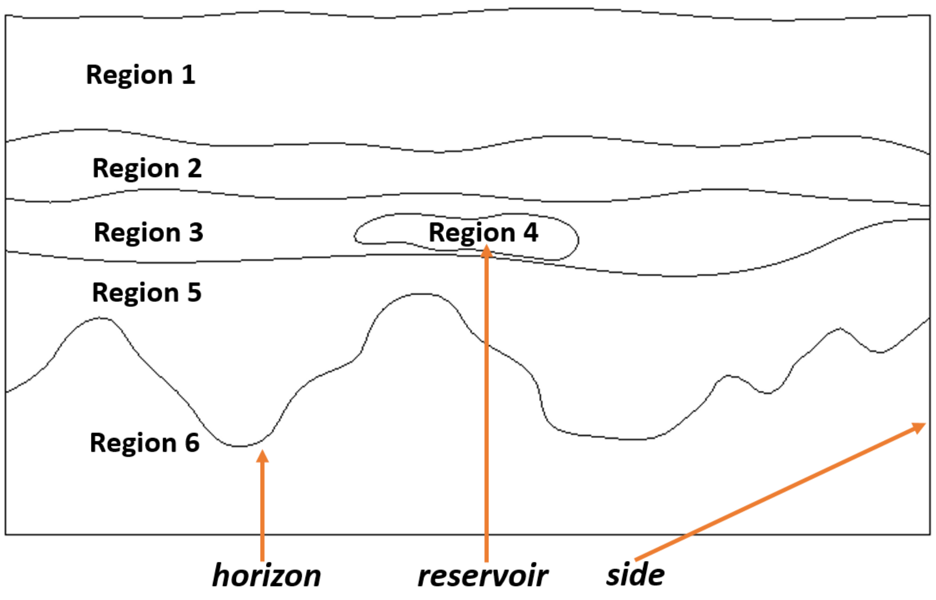


Figure 11.11: Structural model with several horizons and a single reservoir.

We divide curves at hinge points, then automatically set their refinement parameters as follows:

- $ratio_{max}$ is fixed to 10, this value was found empirically to always give satisfying results;
- ref_{max} is deduced from the default expected length of elements along the curves: 50m for reservoirs, 300m for others. We remind here the equation giving the size of an edge length in a quadtree as a function of its depth:

$$edge\ length\ level\ i = \frac{edge\ length\ root}{2^i} \tag{7-10}$$

  We can invert the equation to find level i, given the edge length:

$$i = log2\left(\frac{edge\ length\ root}{edge\ length\ level\ i}\right) \tag{11-1}$$

  We then set $ref_{max} = i$;
- the tolerance is set to the relatively high value of $tolerance = expected\ edge\ length/2$. Doing this, the refinement process is driven by the expected edge length and not by a tight tolerance to the curve, which would in many cases refine more than expected;
- the parameters for the curve endpoints are the same as the parameters for the intermediate geometry.

With this method, we can set all parameters for all curves in the domain very quickly. The mesh is expected to refine until ref_{max}. However, as explained, the algorithm makes local decisions that can lead to refining a little more, or a little less. The large tolerance should prevent the algorithm to refine a lot more, except in high curvature areas, which is good. To ensure that the process refines at least to the expected depth, we add a post-process in the tree construction: all nodes touching a curve with a refinement depth lower than the curve ref_{max} are refined until meeting ref_{max}.

We also give the possibility to make the refinement uniform along a curve. To do so, in the tree, we find the smallest refinement level along a curve and then refine all the elements touching the curve to this level. In the same fashion, we can make the refinement uniform for all elements in a region.

In the curve approximation process, curves are added one by one in order to avoid the step of building groups of non-intersecting objects.

Figure 11.12 shows the resulting mesh for the domain presented in Figure 11.11. The refinement along the reservoir curve is forced uniform. For this type of mesh, if horizons create thin layers, our process will generate a lot of small elements. We plan on detecting such configurations in a pre-processing.

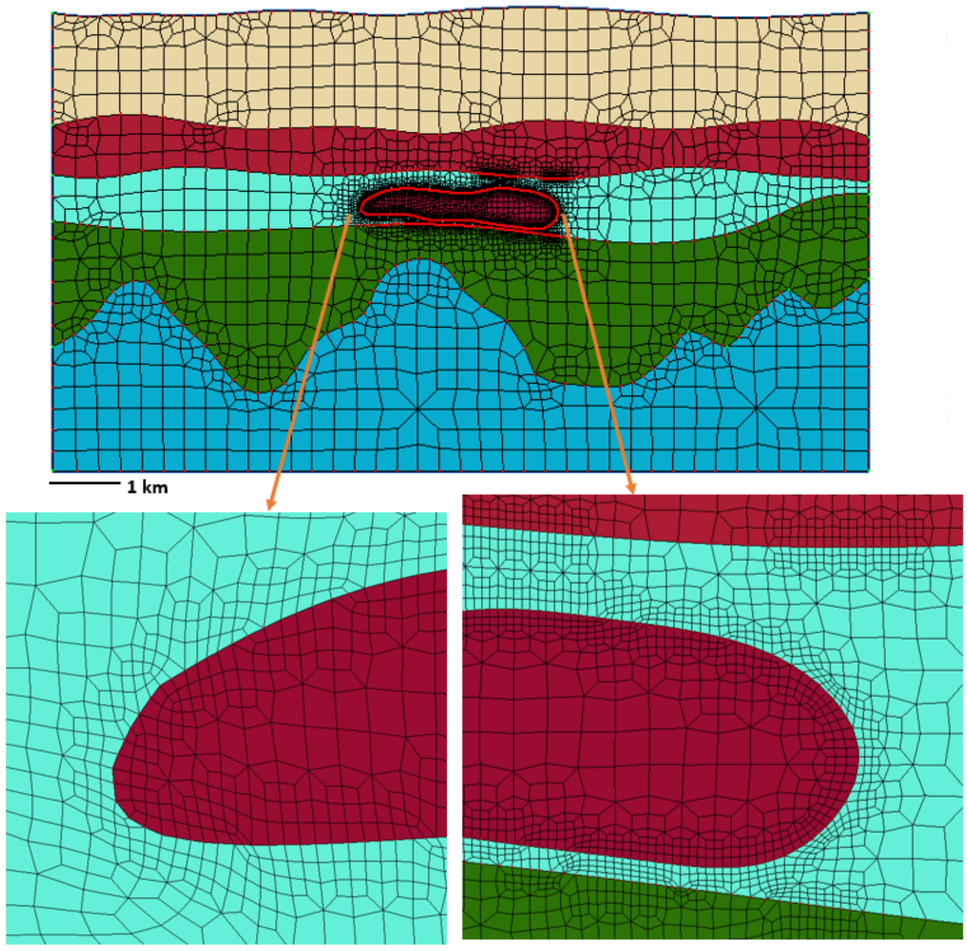


Figure 11.12: Example of a generated mesh for the model of Figure 11.11. Refinement is forced uniform along the reservoir curve.

Figure 11.13 shows another configuration, where the reservoir is viewed from top. There is no horizons, but two artificial circles to define the domain external boundaries. The central circle contains an extended quadtree mesh, while the region between the two circles is a structured mesh, showing an interesting mixture between two meshing methods. The refinement is made uniform along the internal circle only. The algorithm managed to catch the high curvature of the reservoir despite the large tolerance. This type of model is suited for building 3D models, by extrusion.

We plan on building several meshes for a same model, modifying slightly the parameters, then selecting the mesh with the best quality, in order to avoid the creation of models with bad or degenerated elements.

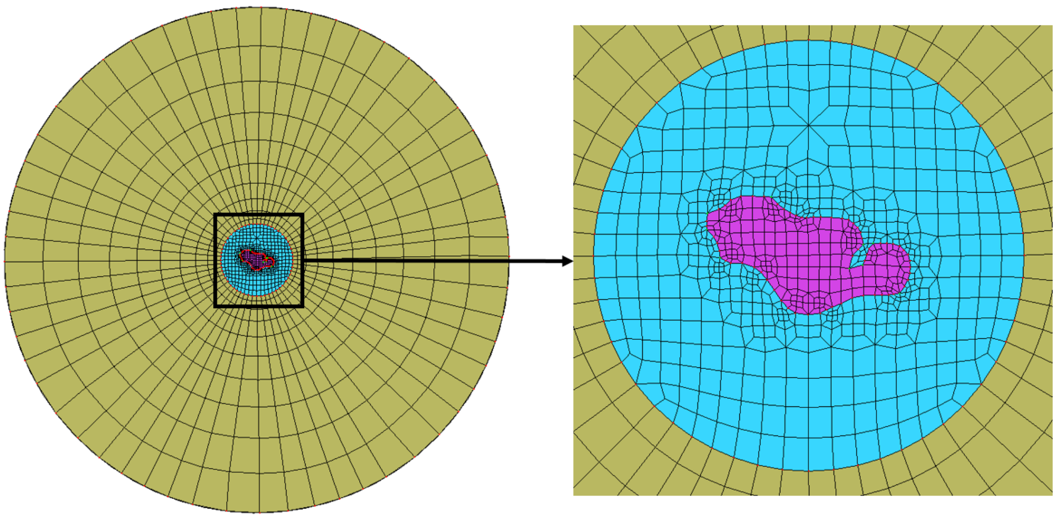


Figure 11.13: A single reservoir view from the top. Circles are artificial boundaries. Refinement is forced uniform along the internal circle only.

12
Conclusions for Part II

12.1
Summary

In this work, we presented a new method for 2D all-quadrilateral mesh generation. The method is based on our extended quadtree structure, which allows building a simple algorithm for the domain adaptation around any curve geometry, with few parameters involved and automatic refinement in high curvature areas. The key to this process was the inclusion of two new subdivision patterns, which we call S1 and S2, in the quadtree node refinement operation. The final mesh is obtained after a conforming step, followed by smoothing. The process proved efficient, building trees of over 400,000 elements in less than 5s, and generated meshes of reasonable quality even for complex structural domains.

12.2
Conclusions

The presented method is a fast and user-friendly mesh generator which finds a solution for any curve configuration. Local adaptation processes allow adaptive refinement without the need to define precise refinement criteria. Compared to other meshing methods, the extended quadtree mesh thus avoids heavy pre-processing of the input geometry and complex parameterization. In return, the user does not have control on local decisions such as, for example, the number of elements along the curves. To some extent, local control is possible through the underlying tree structure, which allows easy and fast refinement for local adjustments in a post-processing step. This can be advantageous in situations where the refinement must be uniform along curves or in specific mesh regions. However, coarsening is much more difficult than refining because of the tree corners adaptation to the curves. Thus, a problem appears in configurations where the extended quadtree tends to generate many elements, for example in the presence of thin layers extended throughout the domain. This situation remains the main limitation to the method, since it can generate unusable meshes. Another limitation of the method is the lack in robust quality guarantees, as the elements can deform freely. However, the generation of a degenerated element can be easily detected by quality metrics such as the jacobian, and the mesh can be reconstructed in few seconds with other parameters in order to attempt reaching a better quality.

12.3 Suggestions for further research

As discussed, the two main limitations of the method are the behavior of the algorithm in thin layer configurations and the lack of robust quality guarantees for the elements.

The first problem could be handled by detecting the configuration and defining a specific refinement scheme in the concerned nodes. For example, if two curves with a given spacing cross one node, we could snap corners directly without having to first separate the curves in different children.

The lack of quality guarantees could lead to locally bad shaped elements. We could investigate processes based on local topological operations to locally improve the mesh, taking as example the works from Kinney (1997), Canann *et al* (1998) and Anderson *et al* (2008).

Finally, a natural continuation to this work is the adaptation of the method to the 3D case: we could try to develop an extended octree. However, two major problems arise that have to be treated in parallel in the 3D case. First, the dynamic adaptation of tree corners to the geometrical constraints is much more complex in 3D. The number of configurations with which a surface can cross a cube is finite (Lorensen and Cline, 1987), however the subdivision of an hexahedron into smaller hexahedron aligned with the crossing surface and with a good quality for each child may be a real challenge. Second, supposing a set of subdivision patterns provides the desired adaptation, neighboring hexahedron will have to conform not only by matching edges and vertices, but also by matching faces, which is not trivial. Some authors have investigated 3D transition patterns (Mitchell, 1999; Yamakawa and Shimada, 2002). None of them guarantees the quality of the created elements, and transition patterns tend to create many small elements, overall presenting techniques which are not adapted to the FEM mesh constraints. The interesting work of Schneiders *et al* (1996) proposes octree conforming patterns in the case of a 27-children scheme octree, consisting in cutting a node in 27 hexahedron instead of 8 (resp. 9 quadrilateral instead of 4 in the 2D case). Owen *et al* (2017) adapted it to the usual 8-children pattern octree. These works may be interesting for mesh extrusion. Mesh extrusion allows easily creating 3D meshes from 2D meshes by projecting elements in the third dimension. In the case of geomechanical reservoirs, conformation to the geometrical constraints can be discarded in certain regions. 3D transition patterns could help reduce the number of elements in those regions by coarsening the mesh, without having to handle mesh alignment with boundaries.

13
Final words

This thesis is part of a research that aims to generate simulation models directly from the seismic data. The final goal is to construct the first model of a digital twin of the oil field as soon as the exploration begins to improve the safety of the operation. As one of the early work, we did not achieve this goal. Both parts of this thesis discuss relevant aspects of the problem. The use of a 2D model was enough to reveal challenges that automatic tools have to face to replace humans in tedious activities that are heavily dependent on interpretation and experience.

In the future, we hope to join the two parts of this work in a general tool. First, we could couple our fault detector with other geobodies' automatic extractors, like horizons or salt domes for example. Clean delineation of faults, horizons and salt domes would then build a structural model that could be the input to our mesh generator.

Merging the different steps of a complex workflow in a single tool is always of great value to reduce the accumulated errors and give the users a clear and general view of their models.

Bibliography

AL-DOSSARY, S.; MARFURT, K. J.. **3d volumetric multispectral estimates of reflector curvature and rotation**. GEOPHYSICS, 71(5):P41–P51, 2006.

ANDERSON, B. D.; SHEPHERD, J. F.; DANIELS, J. ; BENZLEY, S. E.. **Quadrilateral mesh improvement**. Research Note, 17th International Meshing Roundtable, 2008.

ARAYA-POLO, M.; DAHLKE, T.; FROGNER, C.; ZHANG, C.; POGGIO, T. ; HOHL, D.. **Automated fault detection without seismic processing**. The Leading Edge, 36(3):208–214, 2017.

ARAÚJO, C.; CELES, W.. **Quadrilateral mesh generation with deferred constraint insertion**. Procedia Engineering, 82:88 – 100. 23rd International Meshing Roundtable, 2014.

ATALAY, F. B.; RAMASWAMI, S. ; XU, D.. **Quadrilateral meshes with bounded minimum angle**. In: PROCEEDINGS OF THE 17TH INTERNATIONAL MESHING ROUNDTABLE, p. 73–91. Springer, 2008.

BACH, S.; BINDER, A.; MONTAVON, G.; KLAUSCHEN, F.; MÜLLER, K.-R. ; SAMEK, W.. **On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation**. PLOS ONE, 10(7):1–46, 2015.

BAHORICH, M.; FARMER, S.. **3-d seismic discontinuity for faults and stratigraphic features: The coherence cube**. The Leading Edge, 14(10):1053–1058, 1995.

BENGIO, Y.. **Learning deep architectures for AI**. Foundations and Trends in Machine Learning, 2(1):1–127, Jan. 2009.

BLACKER, T. D.; STEPHENSON, M. B.. **Paving: A new approach to automated quadrilateral mesh generation**. International Journal for Numerical Methods in Engineering, 32(4):811–847, 1991.

BOROUCHAKI, H.; FREY, P. J.. **Adaptive triangular–quadrilateral mesh generation**. International Journal for Numerical Methods in Engineering, 41(5):915–934, 1998.

CANANN, S. A.; MUTHUKRISHNAN, S. ; PHILLIPS, R.. **Topological improvement procedures for quadrilateral finite element meshes**. Engineering with Computers, 14(2):168–177, 1998.

CATMULL, E.; CLARK, J.. **Recursively generated b-spline surfaces on arbitrary topological meshes**. Computer-Aided Design, 10(6):350 – 355, 1978.

CHAE, S.-W.; JEONG, J.-H.. **Unstructured surface meshing using operators**. In: PROC. 6TH INT. MESHING ROUNDTABLE, p. 281–291, 1997.

CHEW, L. P.. **Constrained delaunay triangulations**. In: PROCEEDINGS OF THE THIRD ANNUAL SYMPOSIUM ON COMPUTATIONAL GEOMETRY, SCG '87, p. 215–222, 1987.

CHOLLET, F.. **Keras**. `https://keras.io`, 2015.

CONNERS, R. W.; TRIVEDI, M. M. ; HARLOW, C. A.. **Segmentation of a high-resolution urban scene using texture operators**. Computer Vision, Graphics, and Image Processing, 25(3):273 – 310, 1984.

CORTES, C.; VAPNIK, V.. **Support-vector networks**. Machine Learning, 20(3):273–297, Sept. 1995.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.; LI, K. ; FEI-FEI, L.. **Imagenet: A large-scale hierarchical image database**. In: 2009 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 248–255, June 2009.

DI, H.; GAO, D.. **Gray-level transformation and canny edge detection for 3d seismic discontinuity enhancement**. Computers & Geosciences, 72:192 – 200, 2014.

DI, H.; GAO, D.. **Fault detection: Methods, applications and technology**. chapter Seismic attribute aided fault detection in petroleum industry: A review, p. 53–80. Nova Science Publishers, 2017.

DI, H.; SHAFIQ, M. A. ; ALREGIB, G.. **Seismic-fault detection based on multiattribute support vector machine analysis**. In: SEG TECHNICAL PROGRAM EXPANDED ABSTRACTS 2017, p. 2039–2044, 2017.

DI, H.; WANG, Z. ; ALREGIB, G.. **Seismic fault detection from post-stack amplitude by convolutional neural networks**. In: 80TH EAGE CONFERENCE AND EXHIBITION 2018, 2018.

EBEIDA, M. S.; DAVIS, R. L. ; FREUND, R. W.. **A new fast hybrid adaptive grid generation technique for arbitrary two-dimensional domains**. International Journal for Numerical Methods in Engineering, 84(3):305–329, 2010.

EBEIDA, M. S.; KARAMETE, K.; MESTREAU, E. ; DEY, S.. **Q-tran: A new approach to transform triangular meshes into quadrilateral meshes locally**. In: PROCEEDINGS OF THE 19TH INTERNATIONAL MESHING ROUNDTABLE, p. 23–34. Springer Berlin Heidelberg, 2010.

EBEIDA, M. S.; PATNEY, A.; OWENS, J. D. ; MESTREAU, E.. **Isotropic conforming refinement of quadrilateral and hexahedral meshes using two-refinement templates**. International Journal for Numerical Methods in Engineering, 88(10):974–985, 2011.

EL-HAMALAWI, A.. **A simple and effective element distortion factor**. Computers & Structures, 75(5):507–513, 2000.

FAGIN, S.. **Seismic Modeling of Geologic Structures**. Society of Exploration Geophysicists, 1991.

FIGUEIREDO, A. M.. **Mapeamento automático de horizontes e falhas em dados sísmicos 3d baseado no algoritmo de gás neural evolutivo**. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2007.

FREY, P. J.; MARECHAL, L.. **Fast adaptive quadtree mesh generation**. In: 7TH INTERNATIONAL MESHING ROUNDTABLE, p. 211–224. Citeseer, 1998.

GAO, D.. **Integrating 3d seismic curvature and curvature gradient attributes for fracture characterization: Methodologies and interpretational implications**. Geophysics, 78(2):O21–O31, 2013.

GERHARDT, A. L. B.. **Aspectos da visualização volumétrica de dados sísmicos**. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, 1998.

GIBSON, D.; SPANN, M.; TURNER, J. ; WRIGHT, T.. **Fault surface detection in 3-d seismic data**. IEEE Transactions on Geoscience and Remote Sensing, 43(9):2094–2102, Sept 2005.

GLOROT, X.; BORDES, A. ; BENGIO, Y.. **Deep sparse rectifier neural networks**. In: PROCEEDINGS OF THE FOURTEENTH INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND STATISTICS, p. 315–323. PMLR, 2011.

GOLDNER, E. L.. **Avaliação de algoritmos de menor caminho em rastreamento de horizontes sísmicos**. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2014.

GUITTON, A.; WANG, H. ; TRAINOR-GUITTON, W.. **Statistical imaging of faults in 3d seismic volumes using a machine learning approach**. In: 2017 SEG INTERNATIONAL EXPOSITION AND ANNUAL MEETING, p. 24–29. Society of Exploration Geophysicists, 2017.

HALE, D.. **Methods to compute fault images, extract fault surfaces, and estimate fault throws from 3d seismic images**. Geophysics, 78(2):O33–O43, 2013.

HALE, D.. **Seismic image processing for geologic faults**. `https://github.com/dhale/ipf,2014`, 2014.

HARALICK, R. M.; SHANMUGAM, K. ; OTHERS. **Textural features for image classification**. IEEE Transactions on systems, man, and cybernetics, (6):610–621, 1973.

HAYKIN, S.. **Neural Networks and Learning Machines**. Prentice Hall, 2009.

HERRMANN, L. R.. **Laplacian-isoparametric grid generation scheme**. Journal of the Engineering Mechanics Division, 102(5):749–907, 1976.

HILTERMAN, F. J.. **Three-dimensional seismic modeling**. Geophysics, 35(6):1020–1037, 1970.

HOUGH, P.. **Method and means for recognizing complex patterns.**, 1965. US Patent 3069654A.

HUANG, L.; DONG, X. ; CLEE, T. E.. **A scalable deep learning platform for identifying geologic features from seismic attributes**. The Leading Edge, 36(3):249–256, 2017.

KELLY, K. R.; WARD, R. W.; TREITEL, S. ; ALFORD, R. M.. **Synthetic seismograms: A finite-difference approach**. Geophysics, 41(1):2–27, 1976.

KINNEY, P.. **Cleanup: Improving quadrilateral finite element meshes.** In: 6TH INTERNATIONAL MESHING ROUNDTABLE, p. 437–447, 1997.

KRIZHEVSKY, A.; SUTSKEVER, I. ; HINTON, G. E.. **Imagenet classification with deep convolutional neural networks.** In: PROCEEDINGS OF THE 25TH INTERNATIONAL CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS - VOLUME 1, NIPS'12, p. 1097–1105, 2012.

KRUSKAL, J. B.. **Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis.** Psychometrika, 29(1):1–27, Mar 1964.

LAGE, M.; MARTHA, L. F.; DE ALMEIDA, J. M. ; LOPES, H.. **Ibhm: index-based data structures for 2d and 3d hybrid meshes.** Engineering with Computers, 33(4):727–744, 2017.

LECUN, Y.; BOTTOU, L.; BENGIO, Y. ; HAFFNER, P.. **Gradient-based learning applied to document recognition.** Proceedings of the IEEE, 86(11):2278–2324, Nov 1998.

LEE, K.-Y.; KIM, I.-I.; CHO, D.-Y. ; WAN KIM, T.. **An algorithm for automatic 2d quadrilateral mesh generation with line constraints.** Computer-Aided Design, 35(12):1055 – 1068, 2003.

LIANG, X.; EBEIDA, M. S. ; ZHANG, Y.. **Guaranteed-quality all-quadrilateral mesh generation with feature preservation.** In: PROCEEDINGS OF THE 18TH INTERNATIONAL MESHING ROUNDTABLE, p. 45–63. Springer, 2009.

LISLE, R.. **Detection of zones of abnormal strains in structures using gaussian curvature analysis.** AAPG Bulletin, 78(12), 1994.

LIU, Y.; XING, H. L. ; GUAN, Z.. **An indirect approach for automatic generation of quadrilateral meshes with arbitrary line constraints.** International Journal for Numerical Methods in Engineering, 87(9):906–922.

LORENSEN, W. E.; CLINE, H. E.. **Marching cubes: A high resolution 3d surface construction algorithm.** In: ACM SIGGRAPH COMPUTER GRAPHICS, volumen 21, p. 163–169. ACM, 1987.

LU, P.; MORRIS, M.; BRAZELL, S.; COMISKEY, C. ; XIAO, Y.. **Using generative adversarial networks to improve deep-learning fault interpretation networks.** The Leading Edge, 37(8):578–583, 2018.

LUO, Y.; HIGGS, W. G. ; KOWALIK, W. S.. **Edge detection and stratigraphic analysis using 3d seismic data**. In: SEG TECHNICAL PROGRAM EXPANDED ABSTRACTS 1996, p. 324–327, 1996.

MACHADO, M. D. C.. **Determinação de Malhas de Falhas em Dados Sísmicos por Aprendizado Competitivo**. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2008.

MÄNTYLÄ, M.. **An Introduction to Solid Modeling**. Computer Science Press, Inc., 1987.

MARFURT, K. J.; KIRLIN, R. L.; FARMER, S. L. ; BAHORICH, M. S.. **3-d seismic attributes using a semblance-based coherency algorithm**. Geophysics, 63(4):1150–1165, 1998.

MCCULLOCH, W. S.; PITTS, W.. **A logical calculus of the ideas immanent in nervous activity**. The bulletin of mathematical biophysics, 5(4):115–133, 1943.

MINSKY, M.; PAPERT, S.. **Perceptrons: An Introduction to Computational Geometry**. Mit Press, 1972.

DE OLIVEIRA MIRANDA, A. C.; MARTHA, L. F.. **Quadrilateral mesh generation using hierarchical templates**. In: PROCEEDINGS OF THE 21ST INTERNATIONAL MESHING ROUNDTABLE, p. 279–296. Springer, 2013.

MITCHELL, S. A.. **The all-hex geode-template for conforming a diced tetrahedral mesh to any diced hexahedral mesh**. Engineering with Computers, 15(3):228–235, 1999.

MÜLLER, K. .; MIKA, S.; RATSCH, G.; TSUDA, K. ; SCHOLKOPF, B.. **An introduction to kernel-based learning algorithms**. IEEE Transactions on Neural Networks, 12(2):181–201, March 2001.

NIKISHKOV, G. P.. **Introduction to the finite element method**. University of Aizu, p. 1–70, 2004.

NOWOTTNY, D.. **Quadrilateral mesh generation via geometrically optimized domain decomposition**. In: PROC. 6TH INT. MESHING ROUNDTABLE, p. 309–320, 1997.

OPENDTECT. **Netherland offshore f3 block complete**. `https://www.opendtect.org/osr/Main/NetherlandsOffshoreF3BlockComplete4GB.`, 1987.

OWEN, S. J.; STATEN, M. L.; CANANN, S. A. ; SAIGAL, S.. **Q-morph: an indirect approach to advancing front quad meshing**. International Journal for Numerical Methods in Engineering, 44(9):1317–1340, 1999.

OWEN, S. J.; SHIH, R. M. ; ERNST, C. D.. **A template-based approach for parallel hexahedral two-refinement**. Computer-Aided Design, 85:34 – 52, 2017.

PAN, S. J.; YANG, Q.. **A survey on transfer learning**. IEEE Transactions on Knowledge and Data Engineering, 22(10):1345–1359, Oct. 2010.

PEDERSEN, S. I.; RANDEN, T.; SONNELAND, L. ; ØYVIND STEEN. **Automatic fault extraction using artificial ants**. In: SEG TECHNICAL PROGRAM EXPANDED ABSTRACTS 2002, p. 512–515, 2002.

PELLERIN, J.; CAUMON, G.; JULIO, C.; MEJIA-HERRERA, P. ; BOTELLA, A.. **Elements for measuring the complexity of 3d structural models: Connectivity and geometry**. Computers & Geosciences, 76:130–140, 2015.

POCHET, A.; CELES, W.; LOPES, H. ; GATTASS, M.. **A new quadtree-based approach for automatic quadrilateral mesh generation**. Engineering with Computers, 33(2):275–292, 2017.

POCHET, A.; CUNHA, A.; LOPES, H. ; GATTASS, M.. **Seismic fault detection in real data using transfer learning from a convolutional neural network pre-trained with synthetic seismic data**. Submitted to Computers & Geosciences, July 2018.

POCHET, A.; DINIZ, P. H. B.; LOPES, H. ; GATTASS, M.. **Seismic fault detection using convolutional neural networks trained on synthetic post-stacked amplitude maps**. Submitted to IEEE GRSL, April 2018.

QIAN, N.. **On the momentum term in gradient descent learning algorithms**. Neural Networks, 12(1):145–151, Jan. 1999.

RANDEN, T.; PEDERSEN, S. I. ; SØNNELAND, L.. **Automatic extraction of fault surfaces from three-dimensional seismic data**. In: SEG TECHNICAL PROGRAM EXPANDED ABSTRACTS 2001, p. 551–554, 2001.

RAWAT, W.; WANG, Z.. **Deep convolutional neural networks for image classification: A comprehensive review**. Neural Computation, 29(9):2352–2449, 2017.

RIJKS, E. J. H.; JAUFFRED, J. C. E. M.. **Attribute extraction: An important application in any detailed 3-d interpretation study**. The Leading Edge, 10(9):11–19, 1991.

ROBERTS, A.. **Curvature attributes and their application to 3d interpreted horizons**. First Break, 19(2):85–100, 2001.

ROBINSON, E.; TREITEL, S.. **Geophysical Signal Analysis**. Society of Exploration Geophysicists, 2000.

ROSENBLATT, F.. **The Perceptron, a Perceiving and Recognizing Automaton. Project Para**. Cornell Aeronautical Laboratory, 1957.

RUMELHART, D. E.; HINTON, G. E. ; WILLIAMS, R. J.. **Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1**. chapter Learning Internal Representations by Error Propagation, p. 318–362. MIT Press, 1986.

RUSHDI, A. A.; MITCHELL, S. A.; BAJAJ, C. L. ; EBEIDA, M. S.. **Robust all-quad meshing of domains with connected regions**. Procedia engineering, 124:96–108, 2015.

SAMET, H.. **Neighbor finding techniques for images represented by quadtrees**. Computer Graphics and Image Processing, 18(1):37 – 57, 1982.

SCHLEGL, T.; WALDSTEIN, S. M.; VOGL, W.-D.; SCHMIDT-ERFURTH, U. ; LANGS, G.. **Predicting semantic descriptions from medical images with convolutional neural networks**. Information processing in medical imaging : proceedings of the conference, 24:437–48, 2015.

SCHNEIDERS, R.; BÜNTEN, R.. **Automatic generation of hexahedral finite element meshes**. Computer Aided Geometric Design, 12(7):693–707, 1995.

SCHNEIDERS, R.; SCHINDLER, R. ; WEILER, F.. **Octree-based generation of hexahedral element meshes**. In: IN PROCEEDINGS OF THE 5TH INTERNATIONAL MESHING ROUNDTABLE, p. 205–215, 1996.

SHERIFF, R.. **Encyclopedic Dictionary of Applied Geophysics**. Society of Exploration Geophysicists, fourth edition, 2002.

SHERIFF, R.; GELDART, L.. **Exploration Seismology**. Cambridge University Press, 1995.

SIMONYAN, K.; ZISSERMAN, A.. **Very deep convolutional networks for large-scale image recognition**. CoRR, abs/1409.1556, 2014.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I. ; SALAKHUTDINOV, R.. **Dropout: A simple way to prevent neural networks from overfitting**. Journal of Machine Learning Research, 15(1):1929–1958, Jan. 2014.

SUBRAHMANYAM, D.; RAO, P. H.. **Seismic attributes-a review**. In: 7TH HYDERABAD INTERNATIONAL CONFERENCE AND EXPOSITION ON PETROLEUM GEOPHYSICS, p. 398–403, 2008.

SUPPE, J.. **Principles of Structural Geology**. Prentice-Hall, 1985.

TALBERT, J. A.; PARKINSON, A. R.. **Development of an automatic, two-dimensional finite element mesh generator using quadrilateral elements and bezier curve boundary definition**. International Journal for Numerical Methods in Engineering, 29(7):1551–1567, 1990.

TAM, T.; ARMSTRONG, C. G.. **2d finite element mesh generation by medial axis subdivision**. Advances in engineering software and workstations, 13(5-6):313–324, 1991.

TANG, Y.. **Deep learning using support vector machines**. CoRR, abs/1306.0239, 2013.

TINGDAHL, K. M.; DE ROOIJ, M.. **Semi-automatic detection of faults in 3d seismic data**. Geophysical Prospecting, 53(4):533–542, 2005.

VAN BEMMEL, P.; PEPPER, R.. **Seismic signal processing and apparatus for generating a cube of variance values**, 2000. UUS Patent 6151555.

VOULODIMOS, A.; DOULAMIS, N.; DOULAMIS, A. ; PROTOPAPADAKIS, E.. **Deep learning for computer vision: A brief review**. Computational intelligence and neuroscience, 2018, 2018.

WANG, Z.; ALREGIB, G.. **Fault detection in seismic datasets using hough transform**. In: ACOUSTICS, SPEECH AND SIGNAL PROCESSING (ICASSP), 2014 IEEE INTERNATIONAL CONFERENCE ON, p. 2372–2376, 2014.

WANG, Z.; ALREGIB, G.. **Interactive fault extraction in 3-d seismic data using the hough transform and tracking vectors**. IEEE Transactions on Computational Imaging, 3(1):99–109, March 2017.

WANG, Z.; TEMEL, D. ; ALREGIB, G.. **Fault detection using color blending and color transformations.** In: 2014 IEEE GLOBAL CONFERENCE ON SIGNAL AND INFORMATION PROCESSING (GLOBALSIP), p. 999–1003, Dec 2014.

WANG, Z.; DI, H.; SHAFIQ, M. A.; ALAUDAH, Y. ; ALREGIB, G.. **Successful leveraging of image processing and machine learning in seismic structural interpretation: A review.** The Leading Edge, 37(6):451–461, 2018.

XIONG, W.; JI, X.; MA, Y.; WANG, Y.; BENHASSAN, N. M.; ALI, M. N. ; LUO, Y.. **Seismic fault detection with convolutional neural network.** Geophysics, p. 1–28.

YAMAKAWA, S.; SHIMADA, K.. **Hexhoop: modular templates for converting a hex-dominant mesh to an all-hex mesh.** Engineering with computers, 18(3):211–228, 2002.

YAMASAKI, T.; HONMA, T. ; AIZAWA, K.. **Efficient optimization of convolutional neural networks using particle swarm optimization.** In: 2017 IEEE THIRD INTERNATIONAL CONFERENCE ON MULTIMEDIA BIG DATA (BIGMM), p. 70–73, April 2017.

YERRY, M. A.; SHEPHARD, M. S.. **A modified quadtree approach to finite element mesh generation.** IEEE Computer Graphics and Applications, 3(1):39–46, 1983.

YILMAZ, Ö.; DOHERTY, S.. **Seismic Data Processing.** Investigations in geophysics. Society of Exploration Geophysicists, 1987.

ZHIQIANG, W.; JUN, L.. **A review of object detection based on convolutional neural network.** In: 2017 36TH CHINESE CONTROL CONFERENCE (CCC), p. 11104–11109, July 2017.

ZIENKIEWICZ, O.; TAYLOR, R. ; ZHU, J.. **The Finite Element Method: Its Basis and Fundamentals.** Elsevier Butterworth-Heinemann, 2008.