Managing multi-valued horizons within a structural interpretation framework

Joseph Baudrillard

UNIVERSITÉ GRENOBLE ALPES

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Signal, Image, Parole, Télécommunications**

Arrêté ministériel : 7 août 2006

Présentée par

Joseph BAUDRILLARD

Thèse dirigée par **Michèle ROMBAUT** et
co-dirigée par **Sébastien GUILLON**

préparée au sein du
Laboratoire Grenoble Image Parole Signal Automatique (GIPSA-Lab)
dans l'école doctorale **Electronique, Electrotechnique, Automatique et Traitement du Signal (EEATS)**

Gestion des Horizons Multivalués dans le Cadre d'une Interprétation Structurale

Thèse soutenue publiquement le **12/12/2018**,
devant le jury composé de:

Guillaume CAUMON
Laboratoire GeoRessources, Rapporteur, Professeur
Florent DUPONT
LIRIS, Rapporteur, Professeur
Géraldine MORIN
IRIT, Examinatrice, Professeur, Présidente du jury
Michèle ROMBAUT
GIPSA-Lab, Directrice de thèse, Professeur
Stefano FRAMBATI
TOTAL SA, Examinateur, Ingénieur de recherche
Kai WANG
GIPSA-Lab, Examinateur, Chargé de recherche
Jean-Marc CHASSERY
GIPSA-Lab, Invité, Directeur de recherche émérite

Thèse CIFRE avec TOTAL SA co-encadrée par Kai WANG

GRENOBLE ALPES UNIVERSITY

EEATS DOCTORAL SCHOOL

Electronics, Electrotechnics, Automatics, Signal Processing

T H E S I S

in preparation for the title of

DOCTOR IN SCIENCES

of Grenoble Alpes University

Specialty: Signal, Image and Speech Processing, Telecommunications

Defended by

Joseph BAUDRILLARD

Managing Multivalued Horizons within a Structural Interpretation Framework

Thesis directed by Michèle ROMBAUT

prepared at the Grenoble Image Parole Signal Automatique laboratory (GIPSA-Lab)

defended the $12^{th} of December, 2018$

in CIFRE partnership with TOTAL SA

co-directed by Sébastien Guillon, co-advised by Kai Wang

Examination panel:

Guillaume CAUMON	-	GeoRessources, Reviewer, Professor
Florent DUPONT	-	LIRIS, Reviewer, Professor
Géraldine MORIN	-	IRIT, Examiner, Professor, Chairwoman
Michèle ROMBAUT	-	GIPSA-Lab, Supervisor, Professor
Stefano FRAMBATI	-	TOTAL SA, Examiner, Research engineer
Kai WANG	-	GIPSA-Lab, Examiner, CNRS
Jean-Marc CHASSERY	-	GIPSA-Lab, Guest, Senior researcher

Acknowledgements

I would like to thank Michèle ROMBAUT for directing my PhD thesis at the GIPSA-Lab. By her regular and constructive feedback, she helped me keep a good direction during these three years. In the laboratory, I also had the chance to be accompanied by Kai WANG, whose commitment and rigorous proof readings of our publications proved invaluable. His bibliographical knowledge and many ideas were always welcome and fruitful. I am also very grateful for Jean-Marc CHASSERY who contributed regularly to our research, and I paid close attention to his experienced advices. Working with the GIPSA-Lab was a very enjoyable experience, and I feel lucky to have benefited from such a respected research institution.

This CIFRE PhD thesis was conducted at TOTAL SA under the direction of Sébastien GUILLON. I want to express sincere and grateful thanks to him for his dedication and benevolent guidance. His help allowed me to work in collaboration with TOTAL within a most pleasant environnement. I learned a lot from his patience and experience in conducting research, and wish him the best in his future projects. I also have thoughts for Stefano FRAMBATI, whose debugging expertise was challenged but eventually managed to clean my sometimes spaghetti code. I also thank Erwann HOUZAY for the valuable help he provided as the thesis was reaching its end. I really appreciated these three years at TOTAL in the Centre Scientifique et Technique Jean Féger, a comfortable working environnement complemented by the warm and ever-welcoming Sismage team – thanks to everyone for the good times spent around coffee and meals, and best of luck to my PhD student collegues Mehdi and Moctar.

I end these acknowledgments by a word for my familly who helped me on my way to my PhD thesis – a project I have been told to have since my very first years, watching my doctoral student father starting up the prehistoric computer to write his own manuscript. I thank them all, first amongst which my significant-other Marie, who provided precious support and endured some late night article writing, refueling coffee and checking equations.

Contents

List of Figures

List of Tables

List of symbols

AABB	*Axis-Aligned Bounding Box*
AOS	*Array Of Structure*
AVX	*Advanced Vector Extensions*
BOXCQP	*Box-Constrained Quadratic Programing*
BSP	*Binary Space Partitioning*
BVH	*Bounding Volume Hierarchy*
CAD	*Computer-Aided Design*
DEM	*Digital Elevation Model*
ECC	*Error Correcting Code*
EDM	*Euclidean Distance Map*
GC	*Garbage Collector*
GIS	*Geographical Information System*
GPGPU	*General purpose Programing on GPU*
GPU	*Graphical Processing Unit*
HDD	*Hard Disk Drive*
IO	*Input/Output operation*
ISA	*Instruction Set Architecture*
JVM	*Java Virtual Machine*
LAPACK	*Linear Algebra Package*
LIDAR	*Light Detection And Ranging*
LLC	*Last Level Cache*
LNG	*Liquid Natural Gas*
MBV	*Minimum Bounding Volume*
NUMA	*Non Uniform Memory Access*
QC	*Quality Control*

RAID	*Redundant Array of Independent Disks*
RAM	*Random Access Memory*
RLE	*Run Length Encoding*
SIMD	*Single Instruction Multiple Data*
SMT	*Simultaneous Multi-Threading*
SNR	*Signal-to-Noise Ratio*
SOA	*Structure Of Array*
SSD	*Solid State Drive*
SSE	*Streaming SIMD Extensions*
UML	*Unified Modeling Language*

Introduction

A key component of Oil & Gas exploration workflows is the structural model. It is built by interpreting geological objects on three-dimensional digital images of the underground called seismic cubes. Amongst the modeled objects, sediment deposition surfaces (horizons) are especially relevant. They are typically interpreted as sparse polylines or heightmap fragments by geologists, and then interpolated into a denser representation such as a heightmap or a triangle mesh. As hydrocarbon resources get harder to locate and put into production, horizons of complex shape are increasingly considered. Namely, horizons with reverse faults and horizons modeling salt domes are frequently found, amongst other places, in compressive domain. Their shape can be so distorted as to prevent the use of heightmaps: they then become cumbersome or even outright impossible to manage within traditional Oil & Gas software applications. The correct handling of these two types of "multivalued" horizons is therefore mandatory for a structural model to be correctly built.

The first difficulty is that no software package currently provides a unified representation of standard monovalued horizons along with their multivalued counterparts. This is notably due to the fact that few reviews of typical spatial data structures have been conducted in the field of geoscience and hydrocarbon exploration, especially targeting structures arising in compressive domain. The initial objective of the thesis is therefore to find candidate new models and assess their potential to represent multivalued horizons. This will be the subject of the first chapter. The search for new data structures is subjected to various constraints, from user methodologies to software and hardware limitations, along with pragmatic engineering issues such as software legacy and ease of deployment. Once the specifications will be established, potential models will be qualitatively reviewed, then quantitative performance measurements in a representative benchmark will be conducted. At this point, relevant new models will be chosen to represent reverse-faulted and salt dome horizons. Two different models will eventually be used because their associated horizons typically have different geometry and processing needs.

Once data structures are chosen, multivalued horizons have to be reconstructed from sparse interpretation data (polylines and heightmap fragments) to the target multivalued horizon model. A state of the art will show that the subject of surface reconstruction from sparse and unorganized polylines and heightmap fragments representing open multivalued surfaces has not been the subject of much academical interest. Consequently, interpolation methods will be proposed so that both reverse-faulted and salt dome horizons can be efficiently reconstructed from interpretation data, as reported in the second and third chapters, respectively.

The reconstruction of reverse-faulted horizons will target a model naturally extending the heightmap, that we call a patch system. In essence, it is a set of connected heightmaps, using as many heightmaps as required to model a multivalued horizon. In addition to its simplicity, this model will allow for standard monovalued horizon reconstruction methods to be elegantly

adapted. Our reconstruction approach will therefore benefit from the speed and robustness of image-based, two dimensional interpolation methods while modeling a three dimensional object. This will be done at the cost of a preparation stage based on graph labeling, whose complexity is kept low by a unified handling of input interpretation data and a heuristic graph propagation algorithm.

Another reconstruction method will then be presented, in order to interpolate polylines representing a salt dome into a triangle mesh. Once again, for simplicity and performance reasons, we will propose a two dimensional approach. The idea is to parameterize polylines to the plane, to enable easy two dimensional interpolation and triangulation. Once formed in the plane, the triangle mesh can then be transformed back into space. Standard mono-valued interpolation approaches will again be leveraged, enabling real-time reconstruction on our target hardware platform. Most of our work lies in the construction of a polyline parameterization method, using a barycentric approach. These simple parameterizations are defined by a neighborhood and a set of coefficients, and we will show how typical neighbors and coefficients in the literature are not suited to polylines. Instead, we will present an interpolated neighborhood along with a set of optimal coefficients, and show their efficiency when parameterizing polylines.

Chapter 1

New Models for Multivalued Horizons

Contents

We will start by providing the reader with an introduction to the Oil & Gas industry in general and exploration in particular, notably the construction of a structural model. In a second time, we will see that new models are required for multivalued horizons, as well as new reconstruction methods. This first chapter will finally compare and eventually select the new models, while the following chapters (see chapters 2 and 3) will present reconstruction methods for the two main families of multivalued horizons.

1.1 Oil & Gas Exploration Primer

1.1.1 Hydrocarbons

1.1.1.1 Definition

A *hydrocarbon* is a molecule including only carbon and hydrogen atoms. When bonded to other atoms, hydrocarbons are turned into a variety of organic compounds, rich enough to form the entire branch of organic chemistry [Sil06]. By varying the added atoms and the molecule shapes (linear alkanes, cyclocalkanes, alkynes, arenes, etc.), organic compounds can be used in an impressive number of applications. Table 1.1 reports some of the most common hydrocarbons.

C atoms	Phase	Typical form
1-4	Gas	Heating and cooking gas
5-18	Liquid	Solvents, gasoline
12-24	Liquid	Kerosene
18-50	Liquid	Diesel fuel, heating oil, fuel oil, lubricants
50+	Solid	Petroleum jelly, paraffin wax, tar, asphalt, polymers

Table 1.1: Common hydrocarbons sorted by increasing carbon atom count, and their phase at standard conditions of temperature and pressure (293°K, 1 atm).

1.1.1.2 Uses

Hydrocarbons were initially used mostly for lighting in the form of "rock oil" – *petroleum*, from *petra* (rock) and *oleum* (oil) [Dal15]. Today, hydrocarbons take many other shapes and are of primary importance in most aspects of our modern society, amongst which:

- Static energy production. Hydrocarbons are used in massive quantities as fuel for static power plants, especially in countries that ruled out nuclear power (Germany or Poland for example). They are available in whatever phase is most suitable, as solid (coal), liquid (diesel fuel, kerosene, gasoline, marine fuel) or gas (natural gas). They are cheaper than most other energy sources, and easy to transport by rail or pipe;
- Mobile energy production (fuel for motor vehicle / mobile power plants). The energy density of hydrocarbon fuels is significant, and can be leveraged by combustion in many mature engine designs (mainly piston engines or gas turbines, following various thermodynamic cycles) that are competitive in most mechanical environments, such as high-torque, high-speed, high-efficiency or small form factor;

- Chemical industry. Hydrocarbons can be found in most industrial processes, having organic chemistry, an entire subfield of chemistry, dedicated to them. They are precursors, catalysts, solvents, reactants and can be turned into a range of useful products, from plastics, waxes, asphalts and lubricants to drugs. Sulfuric acid, a work-horse of the chemical industry, is made out of elemental sulfur also coming from hydrocarbons.

Because of all these beneficial applications for a booming industrial world, explaining *how* and *where* hydrocarbons form has been the subject of intense geological and geophysical studies from the end of the XIXth century.

1.1.1.3 Formation, Migration, Traps

Hydrocarbons are organic compounds and as their name suggests, they often have a biological origin. They are indeed formed by the deposition of micron-scale decomposed organic material within sediments that turn into rocks on geological time scales [Sch13] – a typical hydrocarbon field is made of rock formations 20 to 400 million years old. Under favorable conditions (high temperature and pressure) that naturally occur at depths of several kilometers, these biological products are cooked into droplets of hydrocarbons, from heavy *crude oil* to light *natural gas* depending on the exact local conditions. Note that we will not consider coal here and focus on oil and natural gas, though coal is formed in a somewhat similar process and is indeed a fossil fuel. Being a solid, coal is mined by other companies than typical oil & gas industrials, using processes from the mining industry.

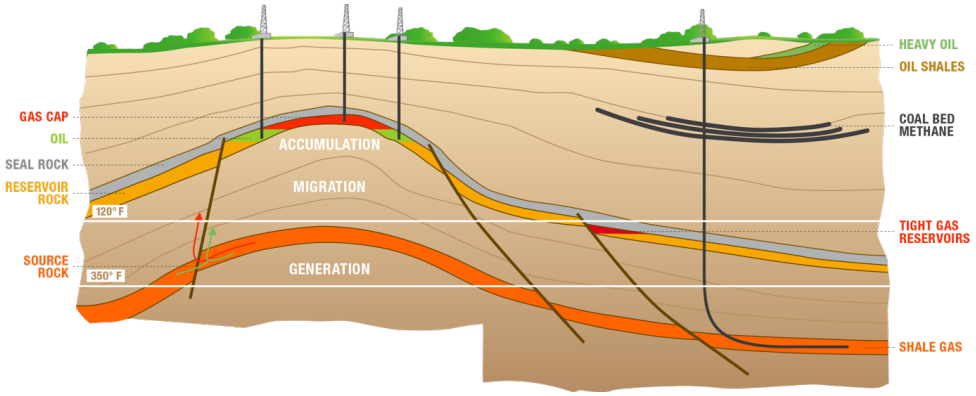

Figure 1.1: An example situation where hydrocarbons form and migrate using a conducting fault until stopped by a seal rock: they are now trapped in the reservoir rock, where they accumulate. The lighter gas can be found at lesser depths than heavier oil. Other hydrocarbon deposits called "unconventionnals" are increasingly produced, for example shale gas, tight gas, coal bed methane or oil shales but will not be considered here. *Image courtesy of TOTAL SA.*

Once formed, hydrocarbons can move within porous rock formations, until they are stuck between non permeable objects called "seals", ranging from salt formations to non-conducting

faults. Hydrocarbons are then trapped in a porous rock that is called a *reservoir*, as seen in figure 1.1.

Easily accessible hydrocarbons were discovered quickly, e.g. open-air oil spills. These low-cost resources are now long gone, and newly discovered reservoirs are buried deep beneath the surface or even the sea floor. Finding reservoirs (exploration) and extracting hydrocarbons from them (production) is therefore required, a significant investment in both time and resources.

1.1.2 The Oil & Gas Industry

1.1.2.1 Domains of Activity

The Oil & Gas industry involves companies and countries that produce economic value by activities related to hydrocarbons. Lands potentially containing hydrocarbons are owned by states, who provide exploitation licenses to companies and take an interest in return. Exploring and producing hydrocarbons on a field typically is a multi-billion dollar project, and companies tend to group together in joint ventures to minimize risks. Around companies lives an ecosystem of contractors and specialists that sell assistance and expertise. As illustrated in figure 1.2, the three main activities (also known as streams) in the oil & gas industry are:

- Exploration and Production (Upstream). Each country is partitioned into "blocks", and states can authorize companies (often specialized contractors) to explore for hydrocarbons. If reservoirs are found, and they are estimated to be economically viable to exploit, the field is put into production. We will detail the processes of exploration in section 1.1.3. At this point production wells are drilled and connected to local treatment facilities (gas liquefaction, monitoring) that are linked to a transport system, be it a pipeline or a tanker filling station[1];
- Refining (Midstream). Extracted raw hydrocarbons (crude oil or natural gas) take their real value after processing in refineries. Each reservoir produces a unique cocktail of hydrocarbons, with varying properties that must be taken into account, such as sulfur content. In a few words, they are separated by distillation and some of the products may be cracked into lighter compounds that typically have a higher economic value, e.g. fuels;
- Selling Refined Products (Downstream). At this point, refined products are sold to consumers in the form of chemicals, fuel at gas stations, etc.

Some industrials focus on a single activity while other "integrated" companies have capabilities in every oil & gas sector, from exploration to selling. This is challenging and prevents

[1] Unconventionals, oil sands, coal liquefaction or thermal depolymerization require different processes and are not discussed here.

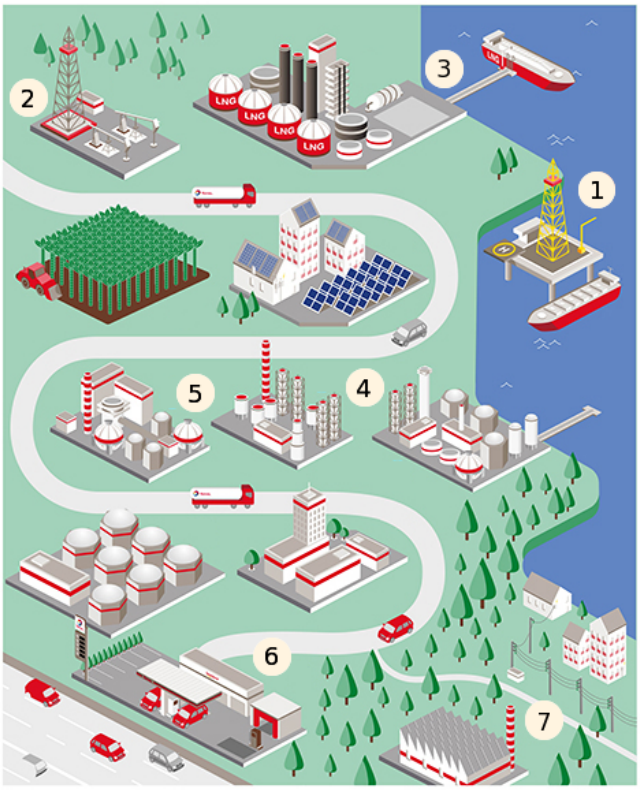

Figure 1.2: Main oil & gas activities. Once discovered, hydrocarbons are produced, be it offshore (1) or on land (2). Natural gas requires liquefaction for cheaper transport (3). Hydrocarbons are then processed in refineries (4) where they are separated and turned into higher value products, as compounds for the chemical industry (5) and fuel for motor vehicles (6) or power plants (7). *Image courtesy of TOTAL SA.*

extreme specialization, while providing protection against economical hazards – a pure exploration company is indeed in great danger when oil price is low, and no exploration is conducted to reduce costs.

1.1.2.2 Climate Change and Sustainability

We discussed the many uses and merits of hydrocarbons. Climate scientists have however accumulated massive evidence of the impact fossil fuels have on human health and the world's climate, mostly by releasing pollutants (nitrogen oxides, particles, sulfur compounds) and huge quantities of carbon dioxide in the air [Cli]; [Bat+12]. In response to this global threat, some oil & gas companies have changed their course of action in order to promote the cleanest fossil fuel: natural gas. Burning natural gas, especially in high efficiency combined-cycle power plants, releases less than half the CO_2 of coal-fired turbines for an equivalent energy output. With no spoil tips and proper filtering, this also prevents the emission of polluting particles, sulfur compounds and nitrogen oxides.

Renewables are undeniably to become a significant proportion of the energy mix [Ren]. Meanwhile, the growing energy demand must be pragmatically met by baseline power plants powered by nuclear fuels or natural gas. Moreover, most renewables are intermittent energy sources, relying on rains to fill dams, wind to blow on turbines or clear clouds from the sky for solar panels [JW16]. As of today, with no solution for country scale energy storage, peak reactors are required when renewables are not enough. This can only be done with hydrocarbon based plants, that can be turned on at second or minute time scales. Hydrocarbons will therefore likely have a lesser influence in the static energy sector for baseline power, while still remaining a major energy source for peak power if no large scale power storage solution is devised.

Concerning mobile power, electric engines and their many advantages are starting to be competitive against the rampant heat engines powering motor vehicles. They have still to solve the energy storage problem, relying for now on difficult to recycle lead-acid or lithium-ion batteries and (potentially polluting) electricity production. Hydrogen fuel cells also improve in efficiency and are starting to be deployed in personal vehicles and trains, yet hydrogen has to be produced in the first place, requiring energy. What will not change however is that the energy density of hydrocarbons is only beaten by nuclear fuels, which are of course reserved for specialist applications (e.g. military submarines and aircraft carriers, icebreakers, seaborne power plants). This means there is for now no serious alternative to hydrocarbons for heavy-duty applications, as in trucks, tankers and airplanes. Indeed, a switch to the cleaner Liquid Natural Gas (LNG) seems the only reasonable path when it comes to replacing the diesel and marine fuels powering trucks and tankers.

In the chemical industry, hydrocarbons will remain a force to reckon with even in the far future: many chemicals and products simply cannot be obtained without hydrocarbons for now, and these activities have negligible CO_2 emission compared to the burning of hydrocarbons as fuel for energy.

All this being said, there is strong evidence hydrocarbons (especially natural gas) will still be needed at a large scale in the future, even when taking a responsible course towards a renewable, minimal CO_2 future[2].

1.1.3 Exploration, Structural Interpretation

The big picture of the oil & gas industry being set, we will now focus on hydrocarbon exploration, and more precisely on the structural interpretation stage.

[2]This is even more relevant when considering other issues such as carbon capture and storage, geothermal energy and hydrogen gas storage for example.

1.1.3.1 Seismic Reflection Surveys

As previously said, countries suspecting the presence of hydrocarbons in blocks overlapping their lands can authorize contractors to explore them, often in collaboration with the country's national Oil & Gas company. The objectives of exploration are to locate potential reservoirs, and assess their economical potential: can hydrocarbons be extracted cheaper than they can be sold? In order to do this, two information sources are typically leveraged:

- Exploration wells. By drilling wells (holes dozens of centimeters wide, thousands of meters deep) and lowering measurement instruments in them, a high resolution (inferior to the centimeter) but very localized insight on the field is obtained. Drilling wells is expensive, in the millions of dollars per well or even way more in complex environment such as deep offshore. This means only a handful of wells will be drilled, in a few carefully thought places;

- Seismic reflection surveys. The main source of information on a field (and the way to decide where to drill exploration wells to be sure about hydrocarbon presence) is to conduct what are called *seismic reflection surveys*. Though they are often even more expensive than several wells combined, they provide a very large scale (dozens of square kilometers), but low resolution (around twenty five meters most of the time) picture of the subsurface. The process is to create acoustic waves using air guns or vibrating trucks, and let the acoustic waves propagate and reflect in the underground[3]. Huge arrays of sensors are set up at the surface, and an inversion problem is solved using supercomputers in order to reconstruct an image (actually an echography) of the field. The digital result is a geolocalized 3D array, called a *seismic cube*.

1.1.3.2 Interpretation

Because wells are very localized, the big picture of the survey is obtained using the seismic cubes. On a cube, geologists will interpret geological objects in a coherent, plausible scenario explaining what is visible on the cube, and how it got there. In figure 1.3, a section of the seismic cube was used to pick a polyline representing a salt dome.

Amongst the objects picked by geologists are reflectors, called "horizons", which model a difference of acoustic impedance often associated with a change in rock type. This means the interface between different rock types is modeled by a horizon surface. Other important objects are faults, which are the result of mechanical failure and displacement within a rock formation subject to stress. When rocks are compressed or elongated by natural geological processes, they deform up to a point where they eventually form faults, and are displaced along the fault surfaces. Once horizons and faults (as well as other objects) are detected, they are grouped in a so-called structural model.

[3]It is also possible to use electromagnetic waves instead of acoustic ones, but will not be discussed here.

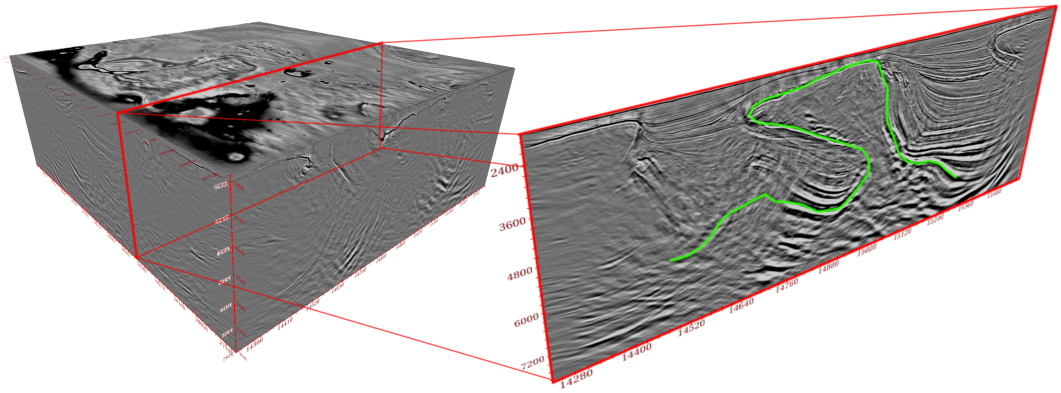

Figure 1.3: An example seismic cube in which a section was considered by a geologist, who picked a polyline representing a salt dome.

1.1.3.3 Reservoir Modeling

The structural model, whose realism is locally verified by exploration wells, only gives a rough idea about the reservoir. A reservoir model is constructed at this point, to localize hydrocarbons and model the fluid movements that will happen if production wells are placed here and there. This enables optimal placement of production wells, which are also very expensive. At this point, a business case can be made and the field might be put into production if deemed economically viable. There are many uncertainties in all this process, meaning sometimes the production wells yield no hydrocarbons, and stay "dry". While some are statistically bound to be so, having too much dry wells and therefore not producing as much as expected is obviously a huge set back, hence the close attention payed all along the exploration chain in order to correctly assess the hydrocarbon content of the reservoir.

1.2 Limits of Current Horizon Model

As an exploration project is conducted, geological objects will be given various representations within Computer-Aided Design (CAD) software processes. The trend is to first interpret objects on the cube using sparse interpretation data (polylines or fragments of heightmap), and move towards denser representation as the project moves on. These denser representations are built by interpolation from sparse interpretation data, and will be used to construct the structural model as described in section 1.1.3.2.

1.2.1 Interpretation Data

The whole problem takes place in a digital seismic survey of horizontal dimensions $W \cdot H$ pixels, the horizontal plane being associated to the first two coordinates (x, y) of a 3D point (x, y, z). We will furthermore note N_S the number of pixels in the survey grid, i.e. $N_S \doteq W \cdot H$. The

survey is illustrated in figure 1.4. Using double brackets for integer intervals, i.e. $\forall (a,b) \in \mathbb{N}^2, a < b, [\![a,b]\!] \doteq \{n \in \mathbb{N}, a \leq n, n \leq b\}$, we therefore define the *survey domain* $\mathcal{D}$ such as:

$$\mathcal{D} \doteq [\![0, W-1]\!] \times [\![0, H-1]\!] \tag{1.1}$$

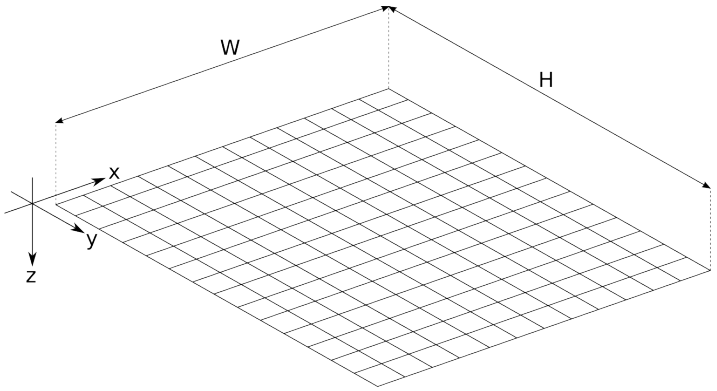

Figure 1.4: The survey 2D grid within 3D space. First two axes span the horizontal plane at a given depth. Note the third axis is directed downwards, as is customary in exploration. Even though we speak of heightmaps, they actually are "depthmaps".

Recall horizons are interpreted by geologist on seismic cubes. Using CAD software, they look at planar sections of the seismic cube (horizontal maps, vertical sections, or arbitrary intersections) and pick horizons using either polylines (series of joined segments) or heightmap fragments (images whose pixels are an elevation distance, with not all pixels being valued). This is illustrated by figure 1.5. The idea is to pick as few interpretation data as necessary in order to save time. The standard process for horizons is to pick interpretation data either:

- By propagation. In regions of the seismic cube where Signal-to-Noise Ratio (SNR) is high, automatic methods can be deployed to quickly pick horizons. The user clicks on points in the section, and the horizon is propagated in 3D towards similar seismic cube values. This is fast but will not work in noisy cube areas. The output is a heightmap fragment, i.e. a heightmap whose pixels are not all valued. It is in general a 2D manifold providing information on the local shape of the horizon;
- By manual picking. In low SNR areas where propagation fails, manual picking is required. This is a tedious but necessary process. Moreover, geologists with experience can use analogues and reasoning to pick objects not directly visible in the seismic cube, and no automatic method can hope to achieve this at the moment. Manual picking produces polylines, that are 1D manifold that locally describe the shape of the horizon.

1.2.2 Interpolated Data

Once picked, either automatically or manually, interpretation data is then typically interpolated into a denser representation: the heightmap. Because horizons represent variations

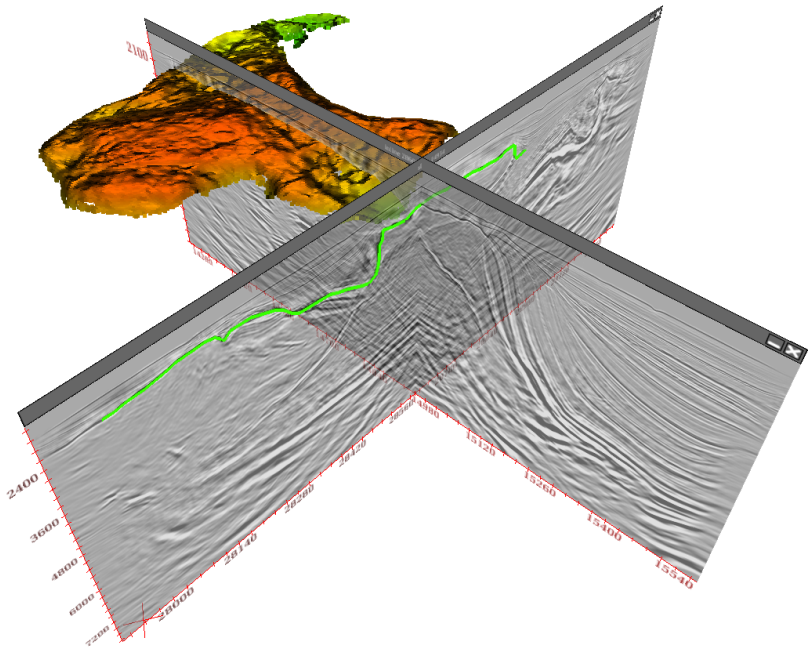

Figure 1.5: Two orthogonal cross-sections of the seismic cube. From the first section, a heightmap fragment (displayed as a point cloud) was propagated from seed pixels. A polyline was picked on the second section.

in acoustic impedance, often associated with different rock types, they are typically located at the interface between layers of sediments. Sediments are usually deposited on sea beds by a gravity-driven process, i.e. they are initially mostly flat. Using an explicit representation such as a Digital Elevation Model (DEM), i.e. a heightmap, is therefore perfectly valid. Heightmaps can moreover be converted easily to point clouds or triangulated surfaces (meshes) for hardware-accelerated display using Graphical Processing Units (GPUs). For these reasons, heightmaps and triangulated surfaces are core objects in the interpretation workflow [Cau+09]; [Nat+13].

1.2.3 Multivalued Horizons

We said horizons are explicit surfaces in the mathematical sense, i.e. it can be projected unambiguously on the plane. This is valid most of the time, however in some situations, namely in compressive domain, horizons may take complex shapes. When subjected to compressive forces, horizons are indeed deformed or broken by faults, and cannot be represented by an explicit model such as a heightmap. As illustrated in figure 1.6, the two main geological situations where this is observed are:

- Horizons with reverse faults. Recall faults are mechanical failures within a rock formation under stress. In extensive domain, where a rock is elongated, faults blocks slide in a way that a hole is created within a horizon. This does not prevent the use of heightmaps.

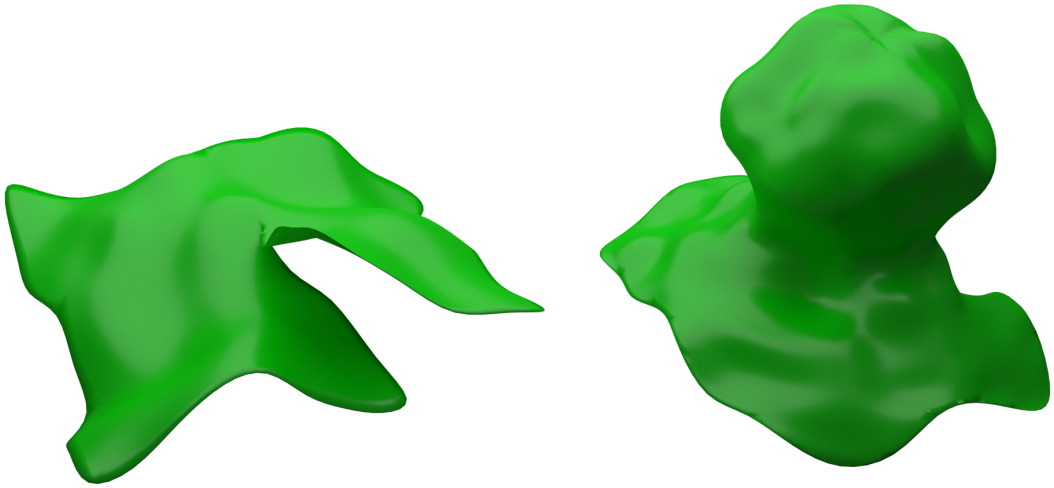

Figure 1.6: The two main types of multivalued horizons. Horizon with reverse-faults (left) and salt dome (right). The former is only multivalued when some horizon parts are superposed because of the displacement along fault planes, while the latter is an intrinsically 3D object.

However in compressive domain, one part of the horizon (the hanging wall) goes on top of the other (the foot wall), making the horizon not explicit anymore;

- Horizons representing salt domes. Huge salt deposits can form when seas dry up on geological time scales. Salt being less dense and more ductile than most rocks, it reacts to compression the way a toothpaste would when the tube is pressed on. In compressive domain, salt formations therefore form ascending bodies called salt domes, that sometimes eventually detach from the salt sheet by forming tear-shaped diapirs. The horizon depicting the limits of a salt dome can be folded to a point where it is not explicit anymore, preventing the use of a heightmap.

For these two types of horizon, a new model is required to replace the heightmap. However, horizons are a core object of oil & gas CAD software, meaning there is no hope to replace heightmaps by a completely different model in practice. This would introduce intolerable software refactoring costs, and would break many features built on the assumption that horizons are explicit surfaces.

1.2.4 Thesis Objectives Reformulation

At this point we can fully understand the objectives of this thesis. We want to provide tools that help to handle multivalued horizons (either reverse-faulted horizons or horizons representing salt domes) when building the structural model. This means we have to:

- Find new models that will replace the heightmap (without changing the model too much for software legacy reasons). This is the subject of section 1.3;

- Develop interpolation methods in order to reconstruct multivalued horizons from interpretation data (polylines or heightmap fragments) to the chosen new models, for:
  - Reverse-faulted horizons. This is done using the method exposed in section 2;
  - Horizons representing salt domes. We present a way to achieve this in section 3.

1.3 New Models

1.3.1 Current Handling of Multivalued Horizons

While heightmaps cannot be used to represent multivalued horizons, current interpretation software has working, if limited and cumbersome, approaches to handle multivalued surfaces. In theory, triangulated surfaces can be obtained using implicit methods [CCC18] or explicit picking that use information from the seismic cube [WFH18] – or both, as they provide complementary insights on the field [CC17]. In practice however, these methods may struggle with the memory requirements required for production fields. While out-of-core approaches have been developed to mitigate this issue [LDD14], there still are massive fields making a case for more adapted approaches.

In other words, current software packages typically only allow a sparse representation of multivalued horizons (polylines along with heightmap fragments), or are very costly to evaluate dense representation (mesh) that often cannot even be computed because the seismic quality is too poor.

All these limitations show the need for new models that would allow a proper representation of multivalued horizons, with efficient (non-implicit, i.e. purely geometrical) reconstruction methods to get a dense representation even in noisy cube areas.

1.3.2 Specifications for New Models

We will start by providing specifications for potential multivalued horizon models. We will enumerate the user requirements, and see how they translate into software requirements. A review of potential models found in the literature will give some candidates to start with, that we will compare qualitatively and then quantitatively in a benchmark, before concluding on which models are to be used for multivalued horizons.

1.3.2.1 User Requirements

We said horizons are picked by geologists in order to build the structural model. In more details, horizons are manipulated during the following processes:

- Modeling. Sparse interpretation data is used to reconstruct the surface by interpolation. This must be fast, so that interpretation data can be modified in order to have a satisfying final surface – the model;
- Saving. Once built, the model must be saved to be accessed in the future without having to reconstruct it once again from interpretation data;
- Quality Control (QC) and Display. Once reconstructed, the model will be visualized:
  - On horizontal cube sections (map view);
  - On vertical cube sections (section view);
  - In 3D (3D view).
- Seismic attributes. A standard processing applied to horizons is the computation of seismic attributes, i.e. quantities located on the horizon surface that have some physical or geometrical meaning. Typical examples are the seismic cube's chaotism or the horizon curvature. While salt domes are modeled mostly to be displayed, reverse-faulted horizons must have attributes computed on them.

1.3.2.2 Corresponding Software Requirements

The user requirements being made clear, we can now translate them into software specifications, that will guide our search for the best multivalued horizon models. Indeed, we are looking for a 3D database that will undergo the following actions:

- Reconstruction. The model will be populated by reconstruction from sparse polylines and heightmap fragments. This must be very fast to allow interpretation data to be modified in order to get a satisfying final model;
- Memory layout and Serialization. Once built, the model must be saved:
  - In volatile Random Access Memory (RAM) for fast access, notably to allow conversion and processing, see below. RAM is faster than non-volatile memory but has a more limited capacity, in the dozens of GB on a typical workstation. This means the model must have a low memory footprint, preferably compatible with compact sequential Inputs-Outputs (IOs) in order to benefit from the full memory bandwidth;
  - In non-volatile memory, e.g. on a remote file system through the network. Non-volatile memory is typically slower than RAM, though the traditional Hard-Disk Drives (HDDs) are increasingly replaced by Solid-State Drives (SSDs) and higher bandwidth connection protocols, reducing the performance gap between volatile and non-volatile memory. Remote storage has a virtually unlimited capacity but a low memory usage is of course welcome. Anyway the main point is that network traffic will be required, meaning the model will have to support random access to some of its parts (block streaming).

- Display format conversion:
  - To point clouds, that are sometimes used in map and 3D view. This means the model must be easy to sample, in order to generate a point cloud at the desired resolution;
  - To triangulated surfaces, the preferred format for 3D manipulation leveraging hardware acceleration with GPUs. Triangulation of the model is not typically done on the fly, but is expected to be nearly instantaneous.
- Processing. The computation of seismic attributes basically requires the model to be a spatial database supporting efficient spatial queries. For example, in order to compute a curvature seismic attribute, the local manifold support of the surface must be accessible everywhere, by taking the intersection of a small ball and the surface at some location.

Meanwhile, recall that the more distant new models are from a heightmap, the more software refactoring will be required. A great emphasis is therefore put on finding an extension of the existing model, instead of a radically new one. Specifications being set, we will now review the various models proposed by the literature.

1.3.3 Spatial Data Structures: State of the Art

Many models are available in the literature when it comes to representing objects in space. They are used in a variety of application, e.g. for Geographical Information Systems (GIS) and CAD. We will focus on four standard models, and also provide an *ad hoc* data structure that extends the heightmap to multivalued horizons.

1.3.3.1 Acceleration Structure vs Isochron Representation

Recall horizons model deposition surfaces. This means a horizon corresponds to a surface of constant geological time when considering the age of rock formations within a seismic cube. For this reason, a point on a horizon is called an *isochron*. We can now remark that many multivalued horizon models can be seen as an *isochron representation*, whose performance is improved by an *acceleration structure*, where:

- The isochron representation provides a local description of a horizon surface element. For example, we can use a triangulated surface to locally represent the horizon;
- The acceleration structure enables good performance when considering the entire horizon. Continuing the previous example, the whole triangulated surface can for example be manipulated faster by using an octree to locate the triangulated surface vertices in space.

By separating isochron representation from acceleration structure, we can now review them separately and study the various combinations in a second time.

1.3.3.2 State of the Art of Isochron Representations

The properties we are interested in when considering isochron representations are the following:

- Geometry, i.e. how well the locations of isochron representations are preserved;
- Topology, i.e. how well the connectivity information between isochron representations is preserved;
- Explicit character, i.e. how many coordinates are used to embed isochron representations in space?

We are *a priori* interested in non-explicit, i.e. implicit, isochron representations because they require less coordinates per isochron. In other words, they have a smaller memory footprint. We will see that implicit representations however have other drawbacks that can mitigate the advantage of not being explicit. Anyway, we will consider the most common models when it comes to representing isochrons, by increasing degree of implicit character. They are illustrated in the case of a salt dome horizon in figure 1.8.

Point Clouds. Point clouds are ubiquitous models for surface representation [LW85]; [RL00]. The main reasons probably are the sheer simplicity of the model, and the fact that any measurement device that sample a real world surface will eventually produce a point cloud [SHG06]. Moreover, many point cloud surface reconstruction methods can be found in the literature [Ber+17]; [Sol+17]. A 3D point cloud is a very explicit model, as 3 coordinates are required per point. It stores the geometry of the sampled surface into its points, while no topology is preserved at all. Most often, the topology is extracted by considering close enough points as neighbors, which can become problematic when the sampling is sparse and non-uniformly distributed.

Triangulated Surfaces. Surfaces can be modeled by a set of connected polygons. Triangles are generally used as they are always convex, leading to a triangulated surface – also called trimesh or mesh. Meshes benefit from an extensive literature on reconstruction, processing and compression [Bot+10]; [HLS07]; [Mag+15]. They also are a standard display format in order to benefit from hardware acceleration via GPUs. As for point clouds, meshes are explicit, i.e. all 3 coordinates must be used for each mesh vertex. Triangulated surfaces can represent connectivity: indeed, they store the surface geometry in vertices, and the topology in edges. A varying triangle density may be used, as low-curvature areas can be modeled by fewer, larger triangles than high curvature zones.

Parameterization. We call parameterization any mapping from a surface to the plane. Parameterization is a vast research field, with many applications in the discrete world [FH54];

[HLS07]. If we are given such a parallelization of the horizon surface to the plane, only 2 coordinates are required for parameterized isochrons. Such a parameterization would therefore be less explicit than a point cloud or a mesh, reducing the memory footprint. However, finding and storing the parameterization is a significant challenge on its own, as the mapping from space to the plane must be saved. Moreover, while mesh parameterization benefits from a mature literature, mapping sparse and non-uniformly distributed polylines to the plane has not been extensively studied to our knowledge.

Patch System. In contrast with explicit models (point clouds & meshes and, to a lesser extent, parameterizations), the standard monovalued model provided a regular support. In a heightmap, pixel locations are indeed implicit. This means their coordinates do not have to be stored, only the pixel value. In this context, we can say a heightmap only requires 1 vertical coordinate (the elevation) per isochron, further reducing the requirement from 3 (point clouds and meshes) and 2 (parameterization).

We are therefore interested in still using heightmaps to model multivalued horizons. The need for heightmaps supporting more than simple orthogonal elevation arose in computer graphics and more precisely in procedural terrain generation [Sme+09]; [TPB06]. When it comes to modeling cliffs with overhangs [Pey+09]; [GM03] or waves [FTR86], the heightmap is indeed not enough. Existing methods are targeted towards generation of a surface from mathematical formulas, whereas we are interested in describing a real-world surface: another approach must be considered. There are hybrid elevation map models coming from point cloud acquisition [Dou+10], but while they can handle multivalued surfaces, they would have a prohibitive memory cost on dense production data. On a side note, using computer graphics' normal maps for multivalued surface storage is tempting at this point, but is actually a form of parameterization and suffers from the same drawbacks.

A natural extension of heightmaps is instead to use a set of connected heightmaps, called a *patch system*. The idea is to use as many heightmaps as required: two vertically superposed points must indeed belong to two separate heightmaps. We also want to support connections between pixels of different heightmaps so that a complete connected surface can be described (see figure 1.7).

More formally, a patch system P made up of N_P patches P_i can be defined as:

$$P \doteq \{P_i, i \in [\![0, N_P - 1]\!]\} \tag{1.2}$$

Each patch P_i is defined as:

$$P_i \doteq \{H_i, N_i\} \tag{1.3}$$

Where:

- H_i is the patch *heightmap*, storing the geometry of the patch:

$$H_i : \begin{cases} \mathcal{D} & \to \mathbb{R} \\ (u, v) & \mapsto \text{Patch height } z \text{ at } (u, v) \end{cases} \tag{1.4}$$

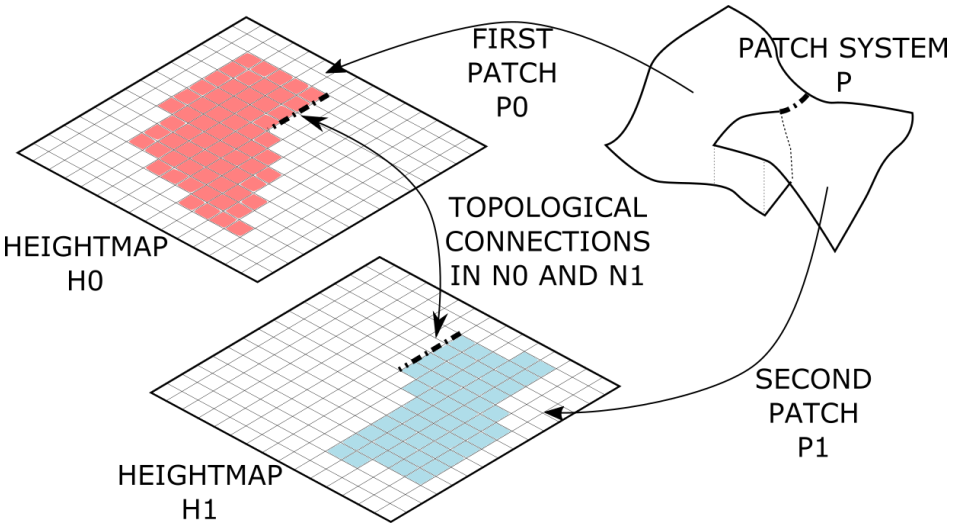

Figure 1.7: An example of multivalued surface described by a patch system made up of two patches P_0 and P_1. The geometry is stored in the two heightmaps H_0 and H_1, whereas the per-pixel topological connections between the two patches are stored in the two neighbor data structure N_0 and N_1.

- N_i is the so-called *neighbor data structure* containing the neighborhood information. Namely it provides the natural pixel neighbors of any pixel of heightmap H_i, and if necessary the pixel neighbors in another patch $P_j, j \neq i$. The latter only occurs for pixels touching another patch, i.e. on the "edges" of a patch. This structure stores the topology of the patch system, and can typically be constructed as a *neighbor patch index map* that provides for each pixel a list of neighbors. A neighbor in this list is a pair of patch index and neighbor index (for example, from 0 to 3 for the four Von Neumann neighbors). This is made space efficient by omitting neighbors that are in the same patch, in other words neighbors with the same patch index.

It must be noted that using a neighbor patch index map prevents from modeling salt dome horizons using a patch system, as it would lead to a prohibitive number of patches. This is acceptable, as salt domes do not require seismic attributes to be computed, therefore a patch system will not be the preferred format for them, but seems promising for reverse-faulted horizons (more details on this in the following sections).

It follows that a patch system is a piecewise-explicit representation of a multivalued horizon, and benefits from the efficiency of the heightmap model for both storage and access. This model elegantly extends the heightmap, and a monovalued horizon can be seen as a patch system with a single patch, without any useless overhead or complexity being introduced.

Voxels. We end this model review by considering the most implicit model: a set of volumetric pixels, also known as voxels. Voxels are used in many applications, ranging from medical imagery [Nov97] to computer graphics [Hug+13]. As for a heightmap, a voxel array

provides a regular support, removing the need for voxel coordinates. Moreover, because the voxel array is a tessellation of space, no elevation is required per voxel. In other words, only the presence or absence of isochron must be stored per voxel: loosely speaking, 0 coordinates are stored per voxel. A voxel array describes the surface geometry by the presence / absence of voxels, while the topology is implicit once a neighborhood type is chosen (e.g. the typical 6 Von Neumann neighbors). Voxels have however a significant drawback: because space must be regularly divided, naive voxel storage is prohibitive. Optimized implementations can however achieve impressive memory cost reductions [IS11], especially when their processing benefits from hardware acceleration via some form of General purpose Programing on GPU (GPGPU) [PKT04].

We have so far considered 5 potential isochron representations for multivalued horizons. Reducing the per-isochron memory footprint can be achieved using increasingly implicit isochron representations, but it has a cost, be it the computation of a polyline parameterization, the construction of a patch system or the storage of a voxel array.

1.3.3.3 State of the Art for Acceleration Structures

The performance of spatial databases (for both time and memory) can be improved using acceleration structures [ZD17]; [Azr+13]. They are also of increasing interest in high performance computer graphics, especially their adaption on the GPU [STL14]; [TSr05], as a promising path towards real-time global illumination. They are data structures that allow efficient access to objects placed in space, especially when a great number of them is considered. These objects can be either punctual, or having a spatial extent [GG98] – in our case they are the isochron representations we just presented. They are also known as *spatial indices*, as they are designed to index objects by taking their position in space into account. Without acceleration structures, a sequential scan of all objects is required when one of them is the subject of a *query*, e.g. it must be found, changed or removed. This means we want to provide better than $\mathcal{O}(n)$ algorithmic complexity with acceleration structures.

In order to provide fast access to objects, each acceleration structure has a particular inner mechanism to discard as many objects as possible from a final short list of candidates that must eventually be searched sequentially. This is a form of space partitioning, that is either regular (independent of data) or data-fitting (following data to avoid wasted space). We will present acceleration structures by increasing degree of data-fitting. There is always a compromise between data-fitting (which reduces wasted space and therefore provides faster queries) and the cost of such data-fitting, that can offset its benefits (implementation complexity and query performance reduction). Spatial indices often provide an $\mathcal{O}(n \log n)$ building complexity, and an $\mathcal{O}(\log n)$ query complexity that exponentially beats the $\mathcal{O}(n)$ naive sequential search. With spatial indices, we are intersted in the compromise between:

- Data-fitting, i.e. how close to the data the acceleration structure is, namely how much wasted space is avoided;
- Traversal speed, i.e. how much overhead does the spatial index introduces for internal

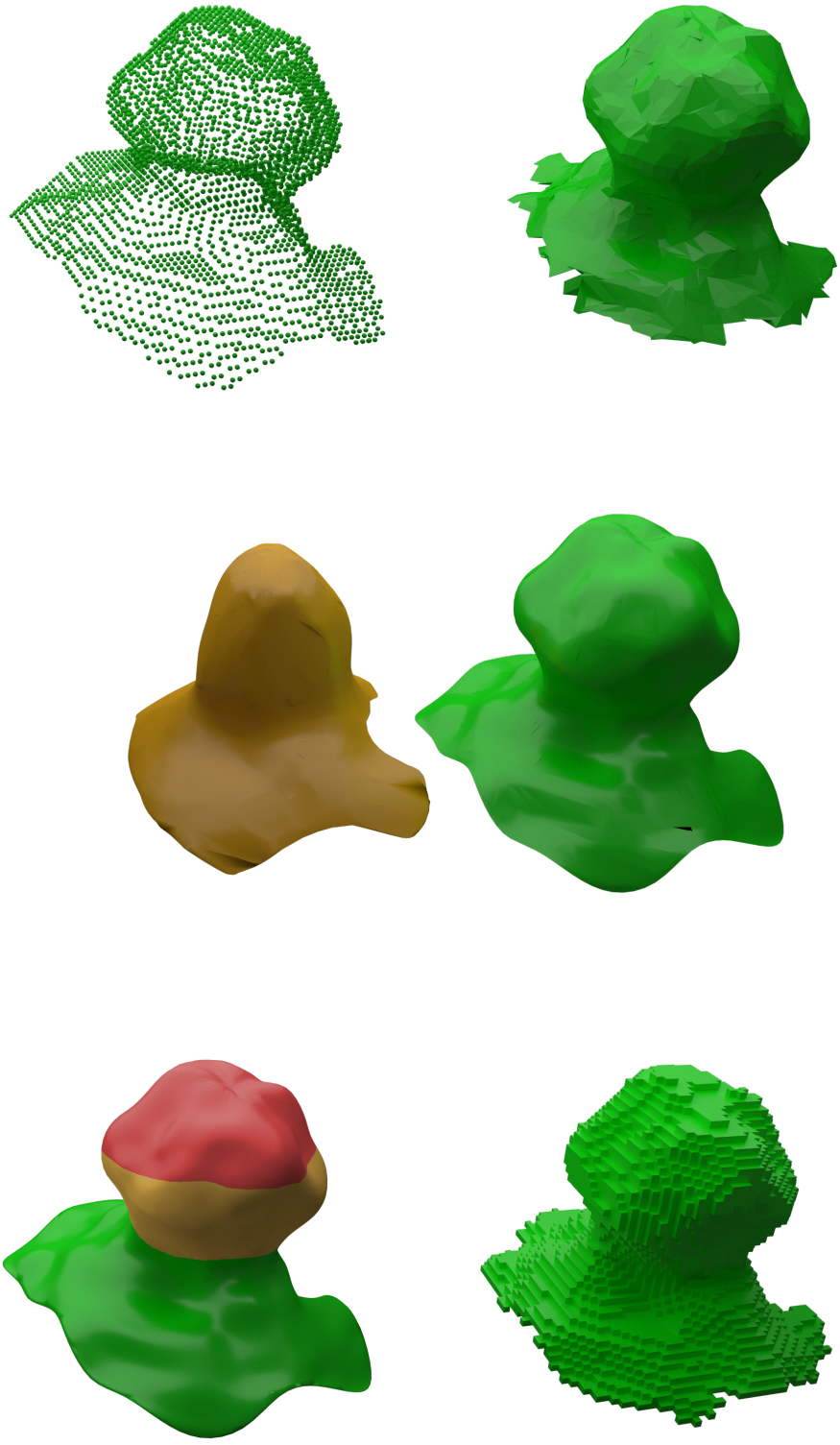

Figure 1.8: A salt dome horizon, as represented by the various models discussed in this section. Top row: point cloud (left), triangulated surface (right). Middle row: parameterization (parameterized monovalued surface on the left, 3D surface on the right). Bottom row: patch system (left, 3 patches are required here because there are at most 3 vertically superposed surface points), voxels (right). Note that in this simple case, a patch system can be used to model a salt dome, however in general it is not the case (to be discussed at the end of this chapter).

management?

The following list is by no means exhaustive, as many specialist data structures can be used in other contexts (hash tables [Knu97], Morton codes [Bia69], skip lists [Pug90] or radix-trees [ZZN14] to name but a few). Moreover, there are many variants of the proposed approaches that will not be discussed here for the sake of brevity. More detailed reviews of spatial indices can be found in the literature [OSdH]; [Pan+16]. The acceleration structures we will review are illustrated in figure 1.9.

Grid. An easy planar acceleration structure is the grid, a regular implicit subdivision of the plane into rectangular cells [RSV01]; [MM06]. This is naturally extended into any finite dimension metric space. Other cell shapes can also be used, as long as they can tessellate the space. Each grid cell contains a list of objects, and a global list of non-empty cells is maintained. When a query is performed, this list of cells is traversed to narrow down the search to at most 4 cells (or 8 in space), whose object lists are sequentially searched. Grids are regular acceleration structure that do not fit to contained data at all. This means some data set can exhibit poor performance, and some pathological data sets can be constructed to defeat the grid entirely (as an example, if the cell size is too big and all the data set is contained in a single cell, the grid performs no better than a naive list). However, the cost of maintaining the grid is minimal, meaning this acceleration structure has a very low overhead.

Regular Tree (Binary Tree, Quadtree and Octree). In order to have an adaptive resolution (versus the fixed grid cell dimensions), a hierarchical subdivision of grid cells can be envisioned. A typical approach is the binary tree (or quad-tree [FB74] and octree [Mea80] in 2 and 3 dimensions), that subdivide a given domain into 2 halves along each axis, each potentially subdivided further if necessary. The main parameter is an optional maximum tree depth, if a few large leaf nodes are preferred over many smaller leaf nodes at various depths. This acceleration structure is therefore more data-fitting than the grid, however the leaf nodes' spatial extent can still be considerably larger than the contained objects.

KD-Tree. By allowing the binary tree to subdivide into two parts but not necessarily at the middle (but still along the axes), the kd-tree (and relaxed quadtree and octree variants) can be obtained [Knu97]. This acceleration structure is even more data fitting, but has a bigger overhead as the separation is not always at the middle, preventing some optimizations and parallelization opportunities.

Binary Space Partitioning (BSP). A Binary Space Partitioning (BSP) tree is a kd-tree that splits not necessarily along the axes [SD69]; [Fuc+80]; [TN87]. This further increases the acceleration structure overhead while making it more data-fitting. At this point, diminishing returns are obtained by making the space subdivision even more irregular. Another approach must be taken in order to increase data-fitting.

Rectangle-Tree (R-Tree). Instead of subdividing space with an increasing irregularity, we can group the objects into clusters that each has a Minimum Bounding Volume (MBV). The clusters can then be organized themselves into a tree, in a so-called Bounding Volume Hierarchy (BVH). Depending on how the MBV and BVH are built, a great variety of acceleration structure can be constructed, amongst which the rectangle-tree (R-tree) [Gut84] and others (R+ [SRF87], R* [Bec+90], Hilbert [KF94], VP [Uhl91]; [Yia93] -trees to name but a few). These acceleration structures are very data-fitting, as by construction, the MBV is kept very close to data. However, a significant overhead is introduced to compute MBVs and manage the BVH, especially to keep it balanced. Various insertion and removal policies can be implemented depending on the data type. On a side note, the relevance of BVHs in computer graphics is likely to increase on the GPU, especially as GPU architectures recently started providing some direct hardware acceleration for BVH traversal and triangle-ray intersections.

Let us wrap up our review of acceleration structures. A first idea is to partition space: with grid acceleration structures, we started with a regular subdivision of space. By further reducing the regularity of the subdivision, more data-fitting was obtained, at the cost of increased overhead. In the end, a BSP tree indeed makes for an irregular subdivision of space. Another data-driven approach is then to group objects into bounding volumes, and manage them in a tree. This is the most data-fitting method, but it also presents a significant overhead.

1.3.4 Evaluating the Best Models

1.3.4.1 Qualitative Comparison

We provided a list of isochron representations. Some of them (point clouds, meshes and voxels) can be greatly improved using acceleration structures. We hence reviewed the most common of them. At this point we can create many models by combining them, for example we can envision point clouds in a R-tree or voxels in an octree. For concision, we will focus on the grid, octree and R-tree, a representative subset of the mentioned acceleration structures. This means we now have to compare the following 11 models:

- Point cloud (in a Grid / Octree / R-tree);
- Mesh (in a Grid / Octree / R-tree);
- Parameterization;
- Patch System;
- Voxels (in a Grid / Octree / R-tree).

An exhaustive comparison by hand would be cumbersome. We will instead first compare isochron representations between them, and then proceed similarly with acceleration structures. A more detailed quantitative comparison will follow next (see section 1.3.4.2).

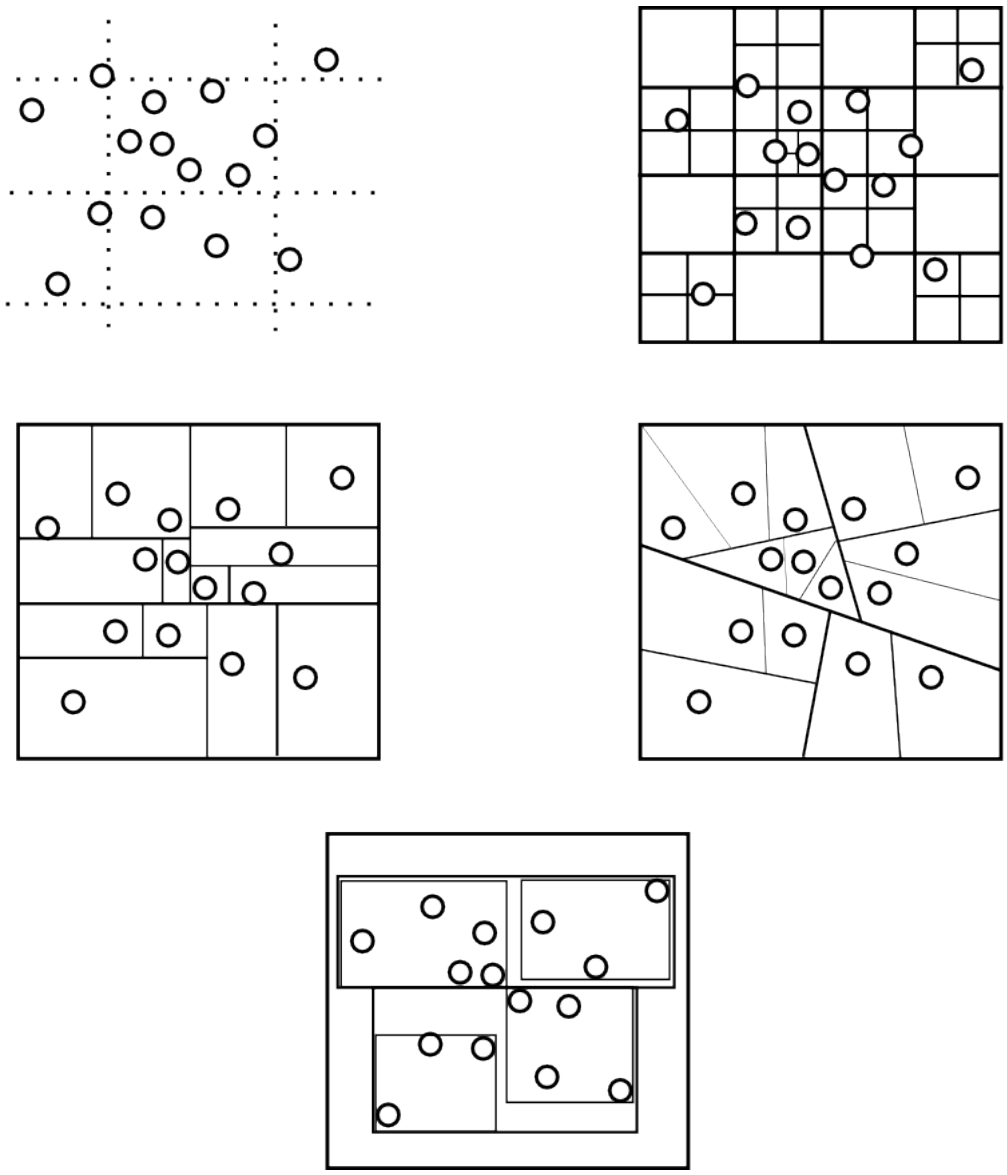

Figure 1.9: Some 2D points, as stored by the acceleration structures discussed in this section. Top row: a Grid implicitly divides the plane and stores points in lists for each cell (left), while an octree regularly subdivides the plane (right). Middle row: a kd-tree is an octree where subdivisions can occur not necessarily at the middle of a cell (left), and a BSP tree even allows the subdivisions not to be along the axes. Bottom row: another approach is to group point clusters into a hierarchy of bounding volumes, as done by a R-tree.

First let us compare isochron representations, as reported in table 1.2. An implicit representation is interesting, because it removes the need from storing coordinates per isochron. Moreover, a connected representation is desirable as it makes the topology unambiguous and easy to access. A low memory footprint is also preferred in order to make storage and transfer faster, while allowing bigger horizons to be managed given a fixed memory budget. Finally, the complexity (of implementation, as well as other costs) should be kept low.

Is. Rep.	Connected	Memory footprint	Complexity
Point cloud	- -	+	++
Mesh	+	+	++
Parameterization	-	++/-	- -
Patch System	++	++	-
Voxels	+	- -	+

Table 1.2: Qualitative comparison of isochron representations. Each property is noted from very bad (- -) to very good (++).

We see that point clouds are disconnected and have a noticeable memory footprint (because they are explicit). Triangulated surfaces are connected but require maintaining an edge list (or a half-edge data structure), and also have a significant memory cost. Parameterization has a low memory impact if the mapping is not stored, though it will likely have to be. While it is connected in parameter space, this is distorted in normal. It also requires some efforts to be computed. Patch systems seem promising, being connected and compact, though some machinery is required to construct them. As for voxels, they are connected and simple to implement, but have a prohibitive memory cost. Overall, point clouds and voxels seem not suited for our needs. Patch systems look more interesting, by construction, while meshes and parameterizations could potentially do.

1.3.4.2 Quantitative Comparison

Complementing the previous qualitative comparison, we can conduct a quantitative measurement of each model performance. Performance comparison has already been presented on many occasions [MK03]; [RSDB11]; [Sar+08], but not with seismic interpretation data. The idea is to construct a benchmark program in which each model will be implemented and subjected to a battery of tests, representing real workloads. The memory and time footprint will then be measured, and this can be done against several realistic data sets at various scales. So to speak, we are looking for the best model in a "4D optimization problem" that tests:

- all models (most being a combination of an isochron representation within an acceleration structure);
- against all situations (i.e. within a test representing a realistic use case of the model);

- against all data types (data source, for example a flat horizon, a folded one, salt domes, reverse-faulted horizons, etc);
- at all scales (various data size to test performance and measure algorithmic complexity).

We will now provide a brief presentation of the benchmark target platform, and then discuss the models, tests, data types and scales. If needed, appendix A can be used to get a more detailed presentation of the benchmark.

Implementation Platform. In order for our benchmark to be as representative as possible, it will be implemented on the very platforms that will run production code and Oil & Gas software. At the time this is written, a workstation typically has two CPU sockets, sporting in total around 32 CPU cores, i.e. software threads on a Simultaneous Multi-Threading (SMT) machine. Each core implements the x86-64 64 bits Instruction Set Architecture (ISA), and has access to around 64GB of DDR3 Registered RAM with Error Correcting Code (ECC). Cold storage takes the form of several TB of SSD storage, complemented by remote file systems on PB-scale filers. The machine is running an enterprise Linux distribution, and uses a professional graphics card. Our target CAD software being written in Java 8 code, we will implement the benchmark in this language.

Models. We have initially selected 11 models, most of which are a combination of acceleration structure and isochron representation. Using the abstraction mechanism of object-oriented programming, we can reduce implementation efforts by separating acceleration structure and isochron representation as two interfaces, implemented by various concrete classes. A model is then either a custom built one (patch system, parameterization) or a combination of an implementation of isochron representation and an implementation of acceleration structure.

As mentioned previously, there is not much literature on the parameterization of sparse, non-uniformly distributed polylines. We thus removed the parameterization from the candidate models (not knowing at the time that we would eventually develop one, see chapter 3). The benchmark therefore compares 10 models, only one (patch system) being custom built.

Tests. We have now to define some tests in order to represent the various situations in which a multivalued horizon is typically used. The performance of the models will be measured during each test, namely the memory and time required to complete the test. The list of implemented scenarios is reported in table 1.3.

While being synthetic, these tests are a good representation of a typical model behavior in a CAD workflow, from the storage aspects to the navigation in a seismic cube (fetching sections, maps), the update of a model after interpretation data is modified, and eventually the conversion to display formats such as point cloud or mesh. Note that each test boils down to a series of queries performed on the model. For example, the `ReadSection` test is a query

Test	Description
`Serialize`	Serialize data from model to binary file
`Unserialize`	Unserialize data from binary file to model
`ReadAll`	Read entire model
`ReadSection`	Read data (only a section) from model
`ReadMap`	Read data (only a map) from model
`WriteAll`	Overwrite entire model
`WriteSection`	Overwrite model (only a section)
`WriteMap`	Overwrite model (only a map)
`ConvertToPointCloud`	Convert model to point cloud
`ConvetToMesh`	Convert model to mesh

Table 1.3: List of tests. The intent of the test and the target properties that are measured are also reported.

that finds all isochrons in a particular section of the cube, and represent a user wanting this section for display.

Data Types. We need some data sets to be loaded into our candidate models. Because we are developing new tools for the handling of multivalued horizons, now multivalued model is readily available from production data: instead, we have to rely on synthetic data sets. This is not problematic as synthetic data allows for a focus on specific properties we want to test, and it can be made as realistic as possible. We will use the following data sets (they are reported in figure 1.10):

- `MonovaluedPlane`. A tilted plane, representing a monovalued horizon with negligible height variations. It will ensure that legacy monovalued horizons are correctly handled;
- `MonovaluedDisk`. A tilted disk, similar to a plane but with non-valued isochrons, i.e. areas of the survey grid were the horizon is not defined. This is to test the correct management of non-valued isochrons;
- `MonovaluedFractal`. A tilted plane with an added height, computed using procedural noise algorithms. Because several frequencies are found in this noise, representing several scales, we call it a fractal noise. With this data set we will see how well details are preserved and how much the high frequency content is increasing the memory requirements;
- `MultivaluedSawTeeth`. A multivalued horizon that could have been generated by reverse faults. This will show how well reverse-faulted horizons are handled by our models;
- `MultivaluedDome`. A multivalued horizon that model a symmetrical salt dome. We will test how such horizons are processed using this data set.

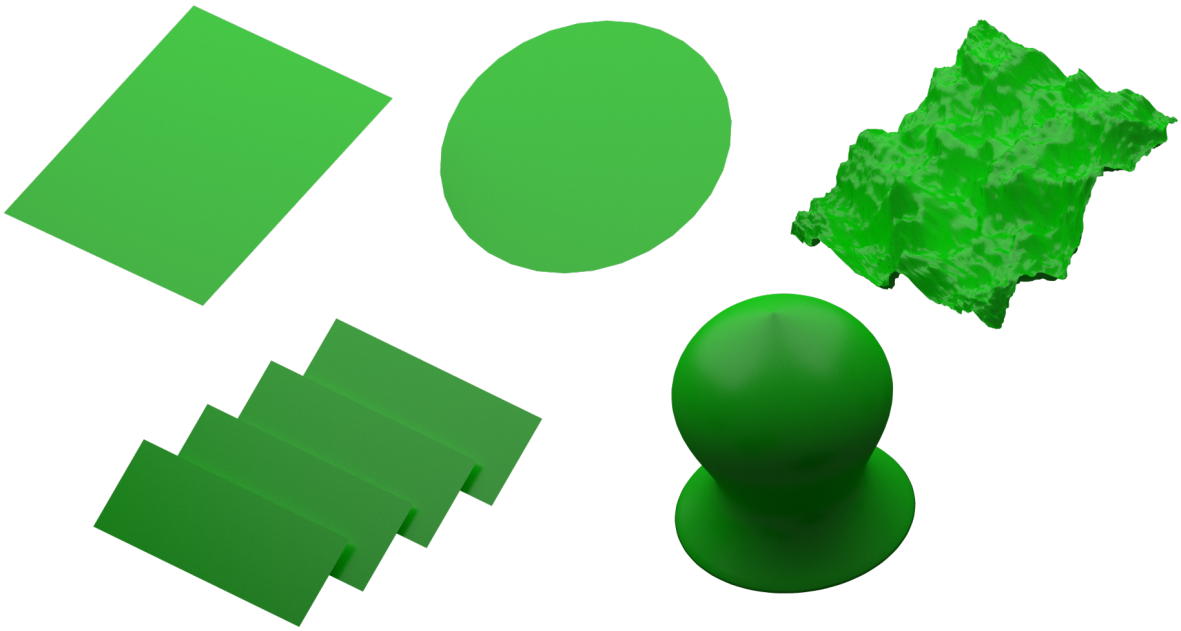

Figure 1.10: The synthetic and scalable data sets used for our tests. `MonovaluedPlane` (top left), `MonovaluedDisk` (top middle) and `MonovaluedFractal` (top right) are monovalued and used to check that new models can still represent standard horizons correctly. `MonovaluedFractal` also introduces high frequency shape variations that will be used to monitor how complex local shapes are handled by the models. `MultivaluedSawTeeth` (bottom left) and `MultivaluedDome` (bottom right) are multivalued and model the two main types of multivalued horizons, namely reverse-faulted horizons and salt domes.

Because these data sets must be available at several scales, they all are procedural, i.e. are obtained by evaluating some parametric function at the desired resolution. They are not loaded from a digital surface format, such as a mesh file.

Data Scales. The objective of having several scales is to observe the influence of data size on performances. Indeed, production data can range from small 2D exploration cubes (having a single section) to huge high-resolution 3D surveys. This means horizons can count from thousands to hundreds of millions of isochrons. We will hence consider square seismic surveys (recall figure 1.4) of side dimension 10^3, 10^4 and 10^5 isochrons. This means that data sets we just presented will have, depending on their exact shape, around 10^6, 10^8 and 10^{10} isochrons. The latter survey certainly is huge even by todays' standards, but will provide some future-proofing for our performance measurement.

Results. Many comments are given on the results in appendix A. We will focus here on the big picture, by considering the performance (in time) of all tests for each model, at the biggest scale, on the `MultivaluedSawTeeth` data set. Results are presented in figure 1.11.

Intuitively, point clouds and meshes benefit from the fact they already are a display format, in other words the conversion to point cloud or mesh for display is just a fast copy. Being explicit model, they are the slowest to serialize. The point cloud model is compromised by the high sampling density required to correctly model the saw teeth, which leads to a high memory footprint and slow operations. Conversely, very implicit models such as voxels do require a data-fitting acceleration structure (octree, R-tree) to reduce their memory footprint to something manageable. This introduces overhead, especially compared to an intrinsically 2D model such as the patch system. Patch systems are indeed the definitive overall winner of this benchmark, performing good in all tests. They require more work than just a copy for display format conversion, but can be easily sampled or triangulated.

The relative performances measured in the benchmark are moreover complemented by other aspects. Software backwards compatibility clearly favors the patch system and mesh models, as the former is a natural extension of the monovalued model, and the latter is already used to model geological objects. These two models are moreover relatively straightforward, not requiring a complex acceleration structure to shine. For all these reasons, we can now choose a model for each multivalued horizon type:

- Reverse-faulted horizons require seismic attributes to be computed, and the patch system seems the most reasonable format: there is a cost associated to the computation of such patch system, as well as for display format conversion, but it is the fastest format for processing, being simpler than a completely 3D model such as voxels;

- Salt dome horizons will be best represented with triangulated surfaces. No processing will be done on them, and this standard format has many advantages: it is connected and can be made reasonably fast with a simple acceleration structure if need be. Moreover,

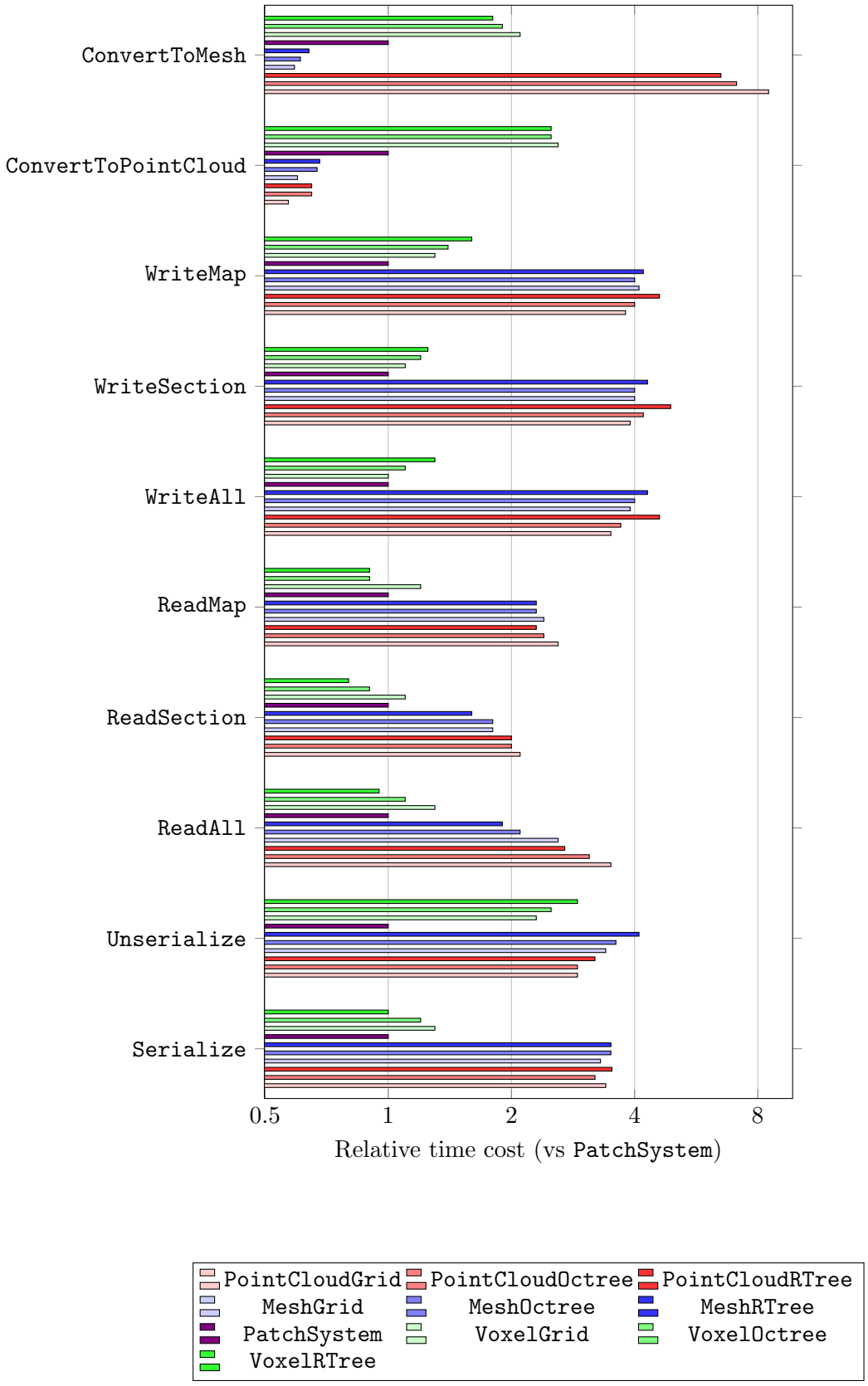

Figure 1.11: Performance comparison of candidate multivalued horizon models.

many software packages target the manipulation of meshes, meaning most operations will not have to be coded from scratch.

1.3.5 Summary

Although this benchmark is by no means comprehensive, it provides a good insight on how typical spatial data structures perform on seismic interpretation data. Such a comparison was found lacking when we gathered the candidate models, and we hope it will benefit others. Its realism and extent can still be improved by adding new isochron representations (notably parameterization, for example as presented in section 3), acceleration structures (BSP trees or others), but also more tests and data types. Using real production data instead of synthetic sources is also possible, though this data will not be provided at several scales. We did not consider the thread safety and parallelization potential of our models, which can however be a great optimization opportunity for some tasks.

Models now being selected, we must develop a reconstruction method for each one, starting only from interpretation data (polylines and heightmap fragments). The reconstruction of reverse-faulted horizons into a patch system is the subject of chapter 2. Reconstructing salt domes from polylines into a mesh will be discussed in chapter 3.

Chapter 2

Reconstructing Horizons with Reverse Faults

Contents

Before anything else we will provide a brief overview of the surface reconstruction literature. It will be apparent that no adequate method is available for the reconstruction of both reverse faulted and salt dome horizons. Thus, we will propose in this section our own reconstruction method for reverse-faulted horizons. As we chose the patch system in order to represent such horizons, we need to find a way to interpolate sparse interpretation data (polylines and heightmap fragments) into a patch system. In section 2.3 we will present gridding, the standard monovalued horizons' reconstruction method, and see how it can be extended to multivalued horizons. This will be achieved very naturally, because the patch system model is itself an intuitive extension of the heightmap model. However, we will see that a preparation stage is required (see section 2.4) in order to construct a patch system ready for gridding. Results, performance considerations and optimizations will also be given.

2.1 Surface Reconstruction: a Brief State of the Art

The subject of surface reconstruction benefits from a vast literature targeting various input formats, from point clouds to triangle meshes. It is a very active field with an increasing relevance, as data acquisition gets facilitated by popular measurement devices such as Light Detection And Ranging (LIDAR) systems. We will briefly expose the big picture of surface reconstruction for our input data type of interest (polylines and heightmap fragments), as more specialized approaches will be discussed in more details for each multivalued horizon type. The literature can be consulted for a more extensive review of numerical reconstruction and interpolation methods [Pre07]; [LH14].

2.1.1 Surface Reconstruction from Polylines

While the literature on surface reconstruction from point clouds is plentiful (see e.g. [LPG12]; [FR01]; [Ber+17]), there are fewer known methods capable of reconstructing surfaces from just a set of polylines. A polyline network can be seen as a very inhomogeneous point cloud, with very high density subsets (the lines) separated by big voids. Consequently, point cloud surface reconstruction methods do not generally perform well on polylines. A specific literature has thus developed to tackle the latter problem, often with industrial design (CAD) or medical imaging applications in mind [BG93]; [VRS11]; [Bes+12]; [BM07]; [SS14]; [BVG11]; [Iji+14]; [BBS08]; [AJA12]. However, strong assumptions are frequently made on the regularity of the input lines, for example, requiring regularly spaced parallel cross-sections [BG93] or convex or otherwise simple patches [VRS11]; [Bes+12]. Other approaches impose constraints on the surface topology [BM07]; [SS14]; [BVG11], which has to be closed in most cases. These assumptions are not realized in our case as our polylines depend on seismic signal quality and user input. Finally, some algorithms depend on application-specific properties [Iji+14] or require extensive user interaction [BBS08]; [AJA12], which is not practical outside of CAD software. Image processing applications for the modeling of organs on medical devices is another source of literature on the subject, but it is mostly focused on the reconstruction of closed surfaces in order to use implicit volumetric methods [ZJC13]; [Zou+15]. As we target open surfaces, these methods cannot be easily adapted, and their computational cost would be prohibitive at the resolutions required for us anyway.

2.1.2 Surface Reconstruction from Heightmap Fragments

The field of geostatistics is notably interested in the reconstruction of dense heightmaps from a few fragments, ranging from single pixels to small heightmap parts. Historically, the objective was to construct map from just a few measurements on the field (be it a digital elevation model from punctual altitude values, or an ore richness indicator from a few exploration mining wells). Generic signal processing operations such as spline interpolation [BPT07]; [Boo89] come first to mind but are not adapted to very sparse real world data. Purpose-built approaches such as kriging [Zim+98]; [Eme05] or inverse-distance weighting [She68] are more

adapted but *gridding*, a variational approach [Bri74]; [SW90] is preferably deployed because it is straightforward, efficient and robust towards constraint density anisotropy which can be severe in our case. However, gridding requires a regular support (a grid) to work, and must be adapted for multivalued surfaces represented by a patch system.

2.1.3 A Need for New Reconstruction Methods

As illustrated by this brief state of the art, the subject of surface reconstruction from either polylines or heightmap fragments (with our constraints) has not received much academical attention – interpolating both of them simultaneoulsy even less so, to the extent of our knowledge. We will therefore have to develop our own reconstruction methods, targeting either reverse-faulted horizons (patch systems) or triangle meshes (salt domes). In this chapter we will start by extending the gridding process to patch systems.

2.2 Sketch of Proposed Method

We use patch systems to represent reverse-faulted horizons. We need to devise a reconstruction method that, from polylines and heightmap fragments, leads to a dense surface representation using patches. We propose to achieve this in two steps:

- Multivalued gridding preparation. In this stage we will find how many patches are required, and somehow turn input polylines and heightmap fragments into patch pixels (this is the subject of section 2.4);
- Multivalued gridding. Once the patch system has some pixels associated with input data, it can be interpolated using an extension of gridding (see section 2.3).

In order to properly understand how to prepare for multivalued gridding, we will start by presenting the gridding method, and how it can be extended to multivalued surfaces modeled as patch systems.

2.3 Interpolating Horizons by Gridding

2.3.1 Monovalued Case

As mentioned in section 2.1.2, gridding is a very popular interpolation method for monovalued horizons. We will now present how the gridding process reconstructs horizons in the monovalued case.

2.3.1.1 Objectives

As presented in section 1.2.1, we consider a digital survey $\mathcal{D}$ of horizontal dimensions $N_S \doteq W \cdot H$ pixels. In the monovalued case, the heightmap we are interested in is also the size of the survey. Given a set of constraint height values $f_{u,v}$ to respect at positions $(u, v) \in \mathcal{D}$, the objective of gridding is to find the unknown heights elsewhere on the heightmap, while creating a smooth surface. This can be formulated as the search for an unknown function $f : \mathcal{D} \mapsto \mathbb{R}$ that takes the values $f_{u,v}$ at locations (u, v) while being smooth.

2.3.1.2 Variational Formulation

Let Ω be the set of locations where the height is known. Horizon gridding can be seen as a minimization problem of a quantity $J(f)$ defined by two components $D(f)$ and $L(f)$, the first quantifying how close the surface is to the constraint heights, the second measuring the "smoothness" of the final surface:

$$J(f) \doteq D(f) + L(f) \tag{2.1}$$

Where:

- $D(f)$ is the constraint term that imposes f to pass through known values at locations in Ω. It is defined as:

$$D(f) \doteq ||f \cdot \delta - \overline{f}||^2 \tag{2.2}$$

  With:

  - δ being the selection function such as:

$$\delta : \begin{cases} \mathcal{D} & \to \{0, 1\} \\ (u, v) & \mapsto \begin{cases} 1 & \text{if } (u, v) \in \Omega \\ 0 & \text{otherwise} \end{cases} \end{cases} \tag{2.3}$$

  - $\overline{f}$ is f evaluated at locations in Ω:

$$\overline{f} : \begin{cases} \mathcal{D} & \to \mathbb{R} \\ (u, v) & \mapsto \begin{cases} f_{u,v} & \text{if } (u, v) \in \Omega \\ 0 & \text{otherwise} \end{cases} \end{cases} \tag{2.4}$$

- $L(f)$ is the smoothness term. In our case we want to minimize the variations and curvature of f which is expressed using a linear combination of the gradient and Laplace operators, as they provide an image of the local slope and curvature respectively[1]:

$$L(f) \doteq \alpha||\nabla f||^2 + \beta||\Delta f||^2 \tag{2.5}$$

[1] As written, $L(f)$ prevents an exact passage through constraints but this is acceptable in our context.

2.3.1.3 The Gridding Equation

We want to minimize the quantity $J(f) = ||f \cdot \delta - \overline{f}||^2 + \alpha||\nabla f||^2 + \beta||\Delta f||^2$. This is reached when $\frac{\partial J}{\partial f} = 0$, which leads to:

$$(\alpha\Delta + \beta\Delta^2 + \delta)f = \overline{f} \tag{2.6}$$

The parameters α and β in equations 2.5 and 2.6 can be made small in order to have a surface closer to input constraints. Conversely, big values lead to a very smooth surface that might respect constraints more loosely. Control over uncertainties can therefore be obtained using relevant parameter values.

2.3.1.4 Numerical Implementation

Recall we note N_S the number of pixels in the survey grid. When evaluated numerically using the finite difference method, by mapping the survey grid on a vector of $\mathbb{R}^{N_S}$, it can be shown that this leads to the definition of a matrix equation in the form:

$$A \cdot \mathbf{x} = \mathbf{b} \tag{2.7}$$

where:

- A is an $N_S \times N_S$ matrix representing the action of operator $(\alpha\Delta + \beta\Delta^2 + \delta)$, i.e. the access to neighbor pixels
- $\mathbf{x}$ is a vector of $\mathbb{R}^{N_S}$ containing the unknown height pixels f
- $\mathbf{b}$ is a vector of $\mathbb{R}^{N_S}$ containing 0 for pixels without constraints, and the known height $\overline{f}$ for constraint pixels

Monovalued gridding is performed by first rasterizing the polylines onto the heightmap using standard algorithms [Bre65]. Pixel positions and heights are interpolated between polyline vertices in this process. As for heightmap fragments, they are projected on the complete heightmap. Equation 2.7 can then be solved by a direct or iterative method (Jacobi, Gauss-Seidel, conjugate gradient, etc.). Implementations are presented in details in the literature [Wal14].

Figure 2.1 illustrates the gridding of some projected polylines and a heightmap fragment. Note that not all pixels of the heightmap become valued (the surface does not take all the image). Indeed, gridding only takes place in what we call the *envelope* of the horizon, i.e. the pixel locations where it should be defined. Outside envelope, extrapolation would occur instead of interpolation[2].

[2]The actual definition of an envelope for monovalued horizons and how it prevents pixels from being gridded are not detailed here for the sake of brevity.

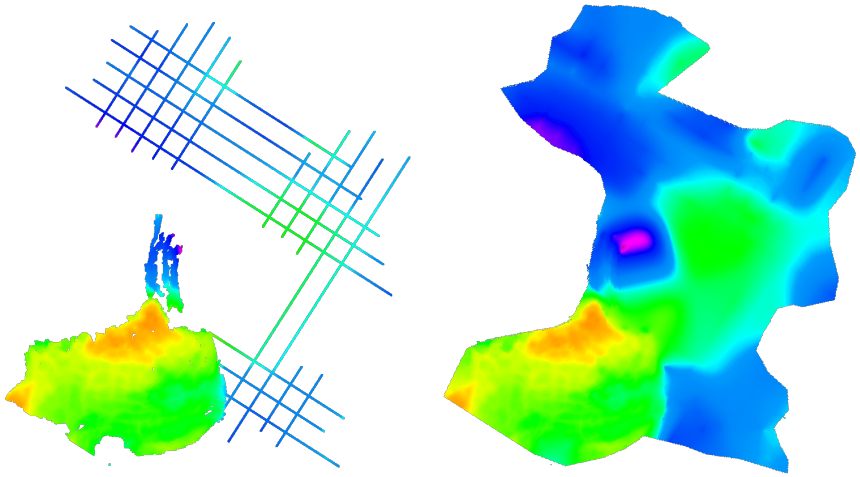

Figure 2.1: An example of monovalued gridding shown on a map (i.e. viewed from top). A color ramp is used to represent elevation. Sparse constraint pixels from rasterized polylines and heightmap fragments (left) are interpolated into a dense surface (right).

2.3.2 Multivalued Case

When looking at equation 2.7, we see that only the connectivity information stored in A depends on the horizon type: it is a simple access to natural pixel neighbors in the case of a monovalued horizon, and becomes slightly more complex in the case of a multivalued horizon – a patch pixel can have neighbors in another patch. Using the notations introduced in section 1.3.3.2, in order to grid multivalued horizons, we define an extended unknown vector $\mathbf{x_E}$ that contains the unknown heights $\mathbf{x_i}$ associated to all N_P patches P_i simultaneously:

$$\mathbf{x_E} \doteq \begin{pmatrix} \mathbf{x_1} \\ \vdots \\ \mathbf{x_{N_P}} \end{pmatrix} \tag{2.8}$$

Using the connectivity information given by the neighbor data structure presented in section 1.3.3.2, it is possible to construct the extended operator matrix A_E and the extended constraint vector $\mathbf{b_E}$ in a similar manner and therefore interpolate each patch correctly. Equation 2.7 then becomes:

$$A_E \cdot \mathbf{x_E} = \mathbf{b_E} \tag{2.9}$$

Interpolation of a multivalued horizon in the form of a patch system is then a natural extension of monovalued gridding. However, given just a set of input polylines and heightmap fragments, in order to prepare a patch system for gridding, we must first convert the input interpretation data into a unified graph (section 2.4.1), then provide a solution to a partitioning problem (section 2.4.2) and finally solve an envelope computation problem (section 2.4.3).

2.4 Multivalued Gridding Preparation

2.4.1 A Graph View of Input Interpretation Data

2.4.1.1 Input Polylines

Recall geologists pick polylines in the low SNR areas of the seismic cube. An example of such a polyline is illustrated by figure 2.2. We can therefore define p, the set of N_p input polylines, as:

$$p \doteq \{p_i, i \in [\![0, N_p - 1]\!]\} \tag{2.10}$$

- Each polyline p_i can be written as follow:

$$p_i \doteq \{V_j = (x_j, y_j, z_j), j \in [\![0, N_{p_i} - 1]\!]\} \tag{2.11}$$

  i.e. it is a set of N_{p_i} 3D points. Those points are scattered within the survey. They are considered to form an open polyline, each being connected by an edge to the previous and next vertices in the set p_i – except for the first and last vertices.

- Each polyline vertex is placed along a regular grid in the first two dimensions, as it belongs to the survey of size $W \times H$ pixels. The third axis use real numbers for better vertical precision. This can be summarized as:

$$\forall i \in [\![0, N_p - 1]\!], \forall (x, y, z) \in p_i, \begin{cases} x \in [\![0, W - 1]\!] \\ y \in [\![0, H - 1]\!] \\ z \in \mathbb{R} \end{cases} \tag{2.12}$$

- Note we use uppercase P for the set of patches P_i in a patch system, while lowercase p is the set of input polylines p_i.

2.4.1.2 Input Heightmap Fragments

In high SNR regions of the seismic cube, automatic picking of horizons can be performed (for example by propagation from seed seismic samples [Kes+82]). This leads to the creation of heightmap fragments in parts of the survey. We define h, a set of N_h heightmaps such as:

$$h \doteq \{h_i, i \in [\![0, N_h - 1]\!]\} \tag{2.13}$$

- Each heightmap is an image of the same size than the survey:

$$h_i : \begin{cases} \mathcal{D} & \to \mathbb{R} \\ (u, v) & \mapsto h_i(u, v) = \begin{cases} z & \text{if pixel is valued} \\ \gamma & \text{(outside surface) otherwise} \end{cases} \end{cases} \tag{2.14}$$

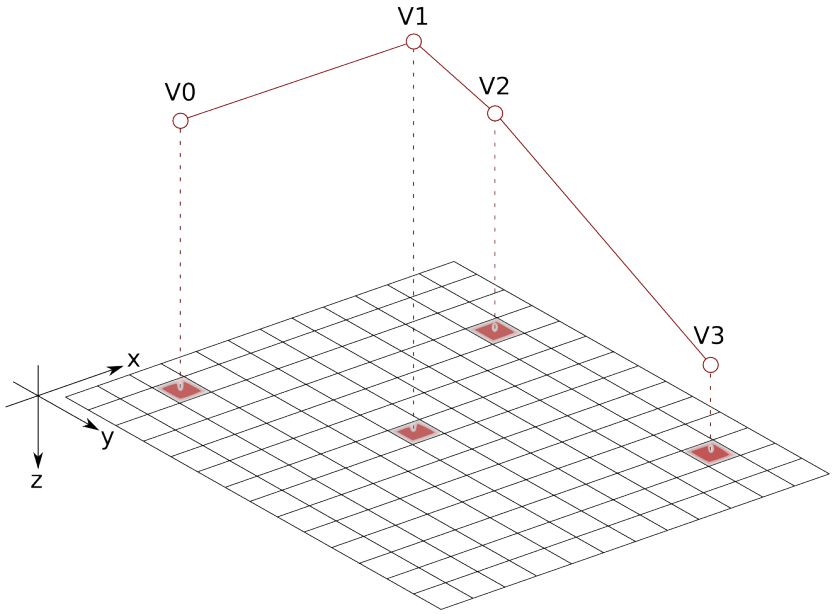

Figure 2.2: A polyline made up of 4 vertices, and their horizontal location on the survey grid.

- γ (outside surface) is the magic pixel value meaning that at this location, the heightmap is not defined (e.g. it is outside of or a hole in the surface);
- It is therefore possible for heightmaps to have holes in them, associated with a zone of one or more γ valued pixels. It means that the heightmap pixels at the boundary of such holes have less than 4 valued neighbor pixels;
- By definition each heightmap, hence each heightmap connected component, describes a monovalued (explicit) surface;
- Once again, note we use uppercase H for the heightmaps H_i of a patch system, while lowercase h is the set of input heightmap fragments h_i, leading to the input heightmap connected components $h_{i,j}$ (see below).

Each heightmap h_i can be made up of several connected components $h_{i,j}$ as depicted in figure 2.3. As for polylines (each being a single connected component), we will rather more consider the set h^* of heightmap connected components, instead of the set of heightmaps h. We can now grant that each object in h^* is, by construction, made up of a single connected component, i.e. from any pixel we can reach any other. This is not incompatible with holes or recursive inclusion of valued pixels inside a hole, for example as in figure 2.3.

2.4.1.3 An Abstract Graph of Interpretation Data

An elegant and practical way to handle both polylines and heightmap fragments as unified inputs for our reconstruction scheme is to create an intermediary abstract graph G as illustrated in figure 2.4, constructed in two steps:

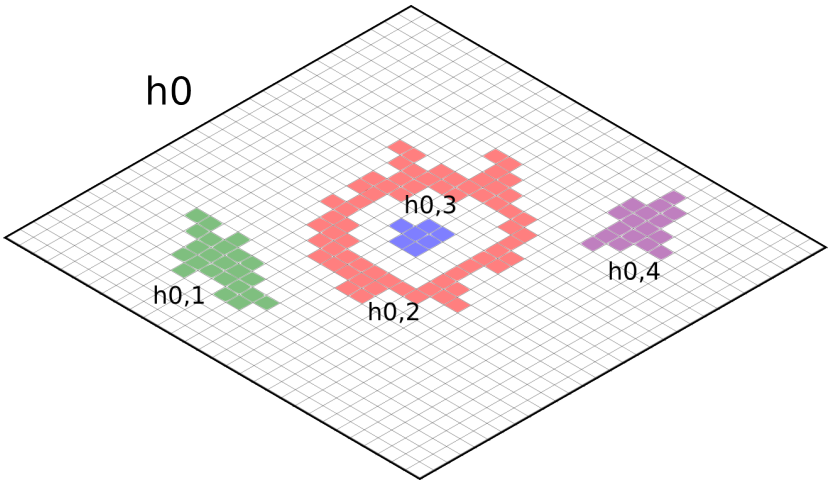

Figure 2.3: This heightmap h_0 is made up of 4 connected components. There is moreover a recursive inclusion of pixels in a hole, i.e. $h_{0,3}$ is within $h_{0,2}$, which is tolerated as long as the abstract graph is connected (see next section).

- A *geometry pass* to create the set of graph vertices V, representing input interpretation data (be it a polyline or a heightmap connected component);
- A *topology pass* to create the set of graph edges E, associated with a topological link between two interpretation data instances (for example, two polylines intersecting or a polyline ending on a heightmap).

2.4.1.4 Geometry Pass: Creating Graph Vertices

The first step is to gather both polylines in p and heightmap connected components in h^* into a single set of input interpretation data, namely V, that will be the vertices of the abstract graph G. By construction, each element in V will be either a polyline or a heightmap connected component, i.e. it will be monovalued (explicit) and made up of a single connected component. Figure 2.5 provides an example of such abstract graph vertices, coming from the input data in figure 2.4.

2.4.1.5 Topology Pass: Creating Graph Edges

The second step is to detect the junctions between any two interpretation data, and create a graph edge in E for each junction. This also leads to the possible splitting of graph vertices (for example two polylines snapped in the middle lead to four polylines connected at an end). This is done in order to get connections only at the border of interpretation data, be it a polyline end vertex or a heightmap connected component's border pixel. Two graph vertices will be moreover added in order to represent the two new polylines. Because we have two classes of interpretation data (polyline or heightmap connected component), we have three

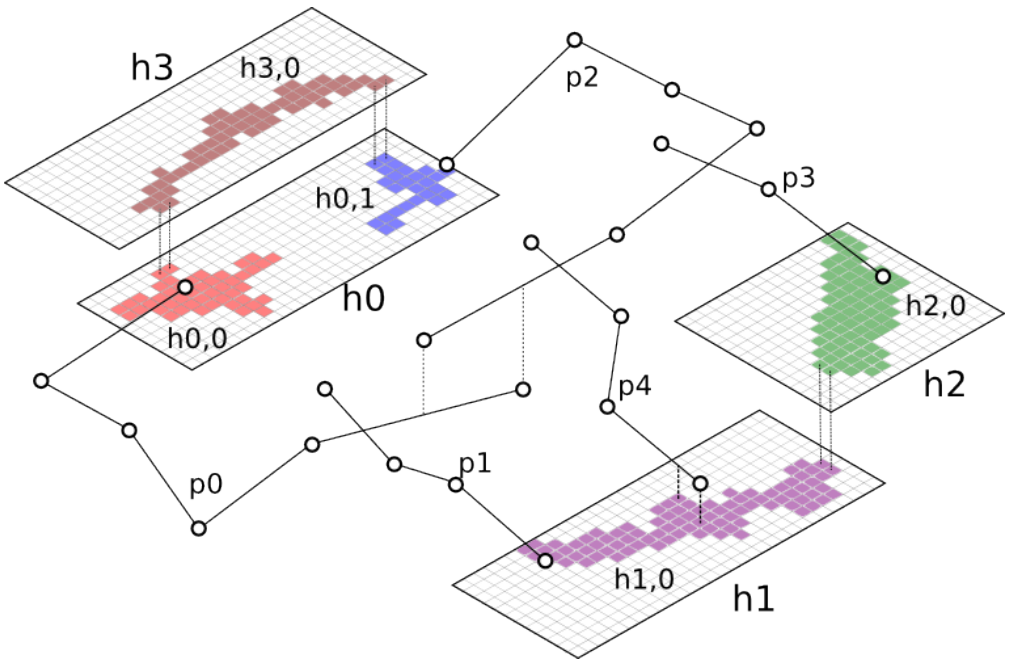

Figure 2.4: An example of sparse input interpreptation data graph, made up of polylines and heightmap fragments, each being composed of potentially several connected components.

possible junctions (the Polyline-Heightmap junction being identical to the Heightmap-Polyline junction):

- Polyline-Polyline junction (PP). This happens when two edges coming from two distinct polylines are very close – recall polylines are hand-picked on computer screens, so they cannot intersect exactly numerically speaking. Given a so-called snapping threshold d_S, we can therefore snap together the polyline edges and introduce a common polyline vertex, as illustrated in figure 2.6;

- Heightmap-Heightmap junction (HH). Automatic propagation of heightmap fragments from several starting points can lead to superpositions between two or more heightmap connected components. By a process similar to polyline snapping, we can join two superposed heightmap connected components along a boundary curve made of pixels (see figure 2.7)[3];

- Polyline-Heightmap junction (PH). Polylines are often picked in low SNR areas, whereas heightmaps can be propagated in high SNR zones. This means they provide two complementary types of input interpretation data, that must be eventually connected together. As for PP and HH junctions, a polyline passing close enough to a heightmap connected component will lead to a connection between a border pixel and a polyline vertex, as shown in figure 2.8.

[3]This junction process is not detailed here but is in the spirit of the dilated envelope restriction in section 2.4.3.3.

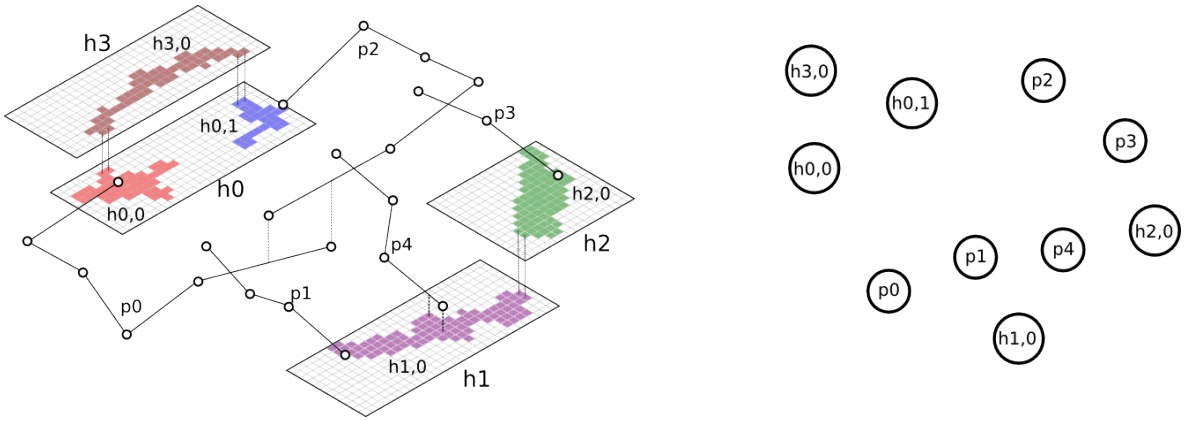

Figure 2.5: An abstract graph representing interpretation data (either heightmap connected component or polyline) in its vertices. Junctions (topological connections between interpretation data) will later be represented with graph edges.

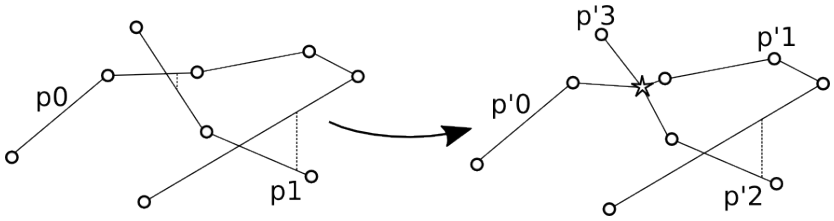

Figure 2.6: Close edges from different polylines are snapped at a common polyline vertex. Edges farther than d_S are not snapped and stay superposed. This leads to four polylines, connected together at an end vertex represented by a five branched star.

We now have a graph G of interpretation data, whose vertices are interpretation data geometry (polyline or heightmap connected component). The graph vertices have an "intra" topology (polyline edges joined by polyline vertices, or implicit connectivity between valued pixels of heightmap connected components), but are by construction monovalued and made up of a single connected component. Figure 2.9 provides an example of the graph at this stage. The graph edges represent "inter" topology, arising in the three situations previously described. At this point the graph G can be partitionned into monovalued sub-graphs that we will interpolate.

For the rest of our worklow, we can assume without loss of generality that G is a connected graph, i.e. all input interpretation data was picked to represent a single connected surface. If not, each connected component can be handled independently as a separate connected graph.

2.4.2 Partitioning Problem

The objective is now to find a decomposition of G into a set of monovalued sub-graphs G_i, i.e. sub-graphs where no internal vertical overlap occurs. Such decomposition is not unique, therefore some criteria must be defined in order to choose a suitable partition.

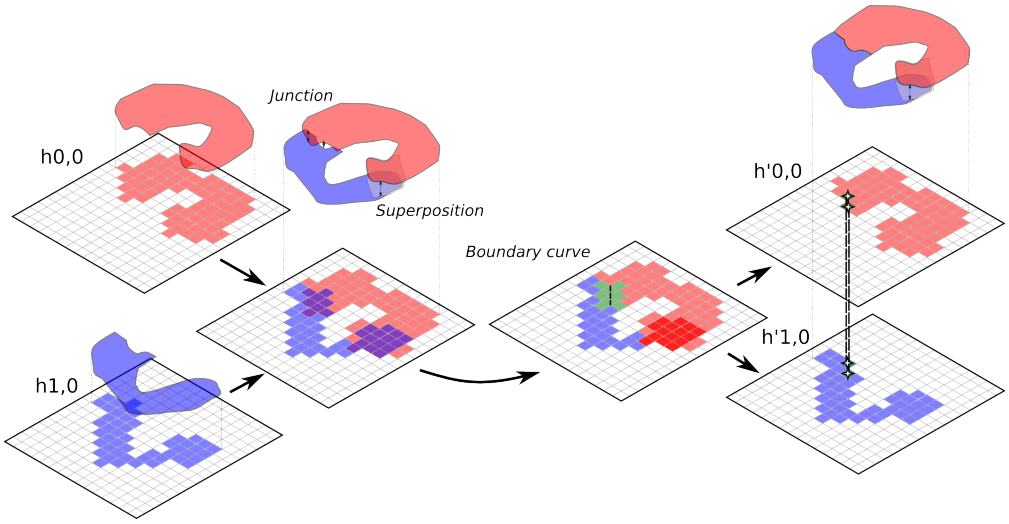

Figure 2.7: Two heightmap connected components that have both superposed parts and "almost joined" parts. Superposed parts are shown in red and are too far away to be joined. Close enough parts (in green) are made to nicely join each other along a boundary curve made of pixels, represented by a four branched star.

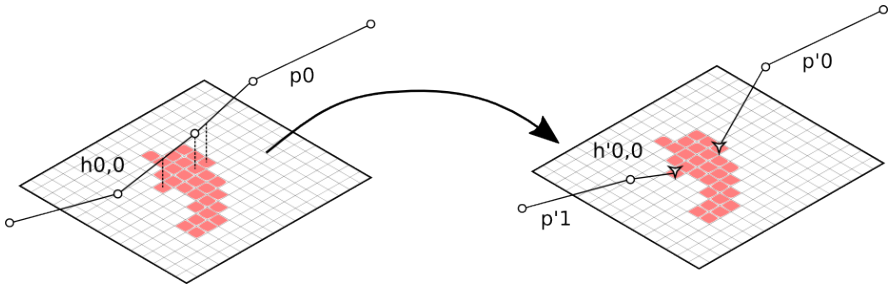

Figure 2.8: A polyline close to a heightmap is snapped on two border pixels, represented by a three branched star.

After a partition is found, each sub-graph will be turned into a patch heightmap in the envelope computation stage, and will then be gridded. Both these steps have a computational complexity of $\mathcal{O}(N_P \cdot W \cdot H)$ where N_P is the number of patch in the patch system, W and H are the width and height of the patch (here we assume patches the size of the survey for simplicity). This means we want to reduce the number of patch (so the number of sub-graphs G_i) as much as possible, and large patch size must be avoided – this is a second order concern though as only the envelope will be considered, not the entire heightmap image.

Considering the partitioning problem in a variational framework could provide an optimal combination of patch count and size [Bul+13], but we favored a simpler method based on a heuristic. The graph having a relatively small size, this easy-to-maintain approach is actually preferable from an engineering standpoint. In this context, we propose a constructive method that leads to an acceptable compromise between patch count and size. It is based on three steps:

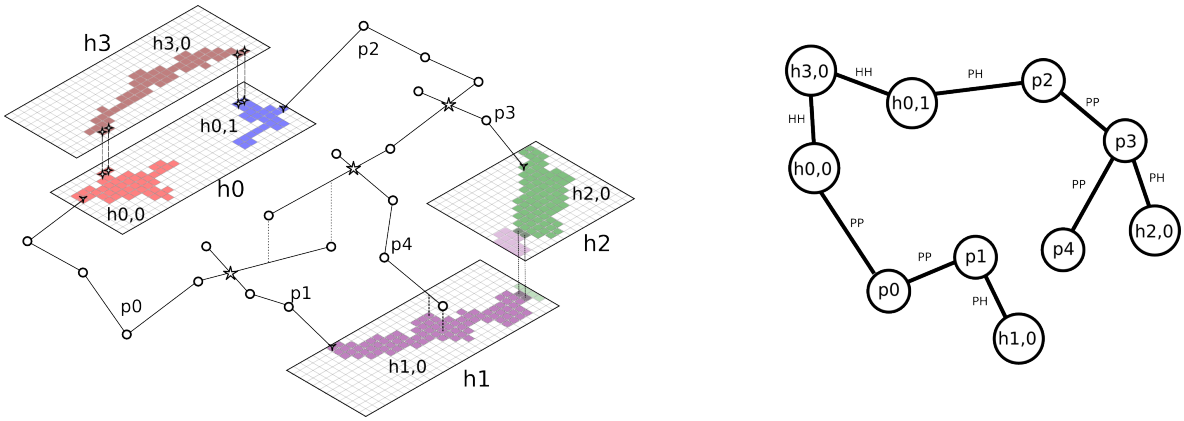

Figure 2.9: The abstract graph after junctions are handled: there are now graph edges representing topological connections between input interpretation data. Stars with 3, 4 and 5 branches represent PH, HH and PP junctions respectively.

- **Multivalued scan**. Vertically superposed interpretation data of G are detected, grouped into superposed zones, and graph vertices are introduced in order to avoid having half-superposed interpretation data;
- **Sub-graph index propagation**. Simultaneous propagation of sub-graph index from superposed zones leads to the definition of monovalued sub-graphs G_i;
- **Merge**. Reduce sub-graph count by merging together those that can be. The sub-graphs after merge are noted $\tilde{G}_i$.

2.4.2.1 Multivalued Scan: Detecting Superpositions in the Graph

The objective of this section is first to detect when the interpretation data associated with two graph vertices are vertically superposed, and second to split them such that any two graph vertices are either completely superposed or not at all. As for the three possible junction situations, there are three different kind of superposition:

- Polyline-Polyline superposition (PP). Polylines are picked in planar sections of the cube (vertical, horizontal or arbitrary). This means some polyline edges can be vertically superposed. There is no reason for two edges to be entirely superposed though, so we introduce a polyline vertex whenever necessary so that a polyline edge is either completely superposed with another, or is not at all (see figure 2.10);
- Heightmap-Heightmap superposition (HH). Some heightmap connected components can vertically overlap, while having a significant distance between them: they will not be joined as in section 2.4.1.5. Instead, each must be split into several heightmap connected components such that each is either completely superposed with another, or not at all. This is illustrated in figure 2.11;

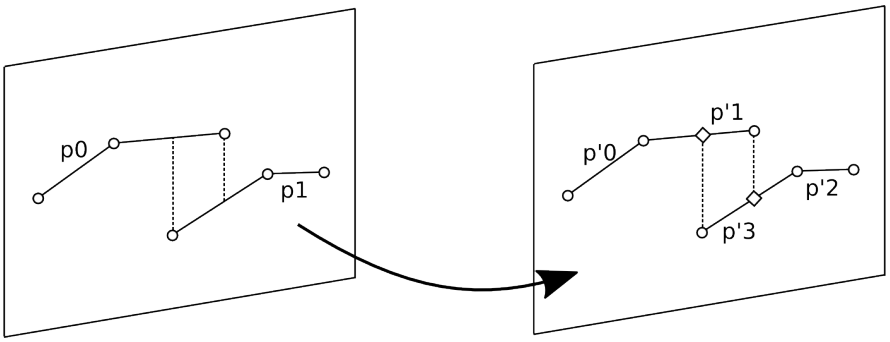

Figure 2.10: Polyline vertices (symbolized here by diamonds) are introduced and polylines split in order to only have polylines that are totally overlapping, or not at all.

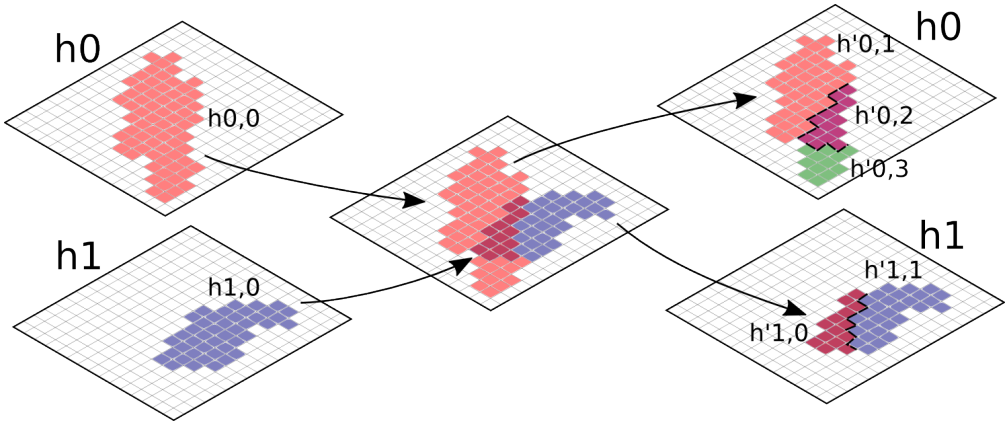

Figure 2.11: Superposed heightmap pixels lead to three new heightmap connected components, joining the previous ones along boundary curves.

- Polyline-Heightmap superposition (PH). By seeing a polyline as a heigtmap (for example by rasterization [Bre65]), a PH superposition is nothing but a degenerate HH superposition and can be solved as shown previously (see figure 2.12).

Whatever the superposition type, additional graph vertices will be added in order to enforce total or zero superposition for any two graph vertices. Once these vertices are introduced, detecting vertically superposed graph vertices is a simple geometric problem. Figure 2.13 shows the final graph after superpositions are handled.

2.4.2.2 Sub-Graph Index Propagation

At this point, we can give an index i for each graph vertex that is superposed, and for each we initialize a *monovalued sub-graph* G_i with the graph vertex. The sub-graphs G_i are called monovalued as by construction, each is made of graph vertices that do not overlap, each one being also monovalued by construction. All the graph vertices can then be indexed using a propagation method illustrated by algorithm 1.

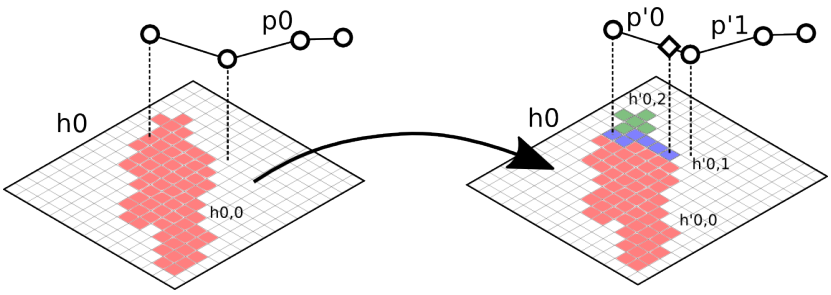

Figure 2.12: A polyline superposed with a heightmap, leading to the creation of a new heightmap along superposed pixels. The polyline must also be split in two and a polyline vertex (drawn as a diamond) is introduced.

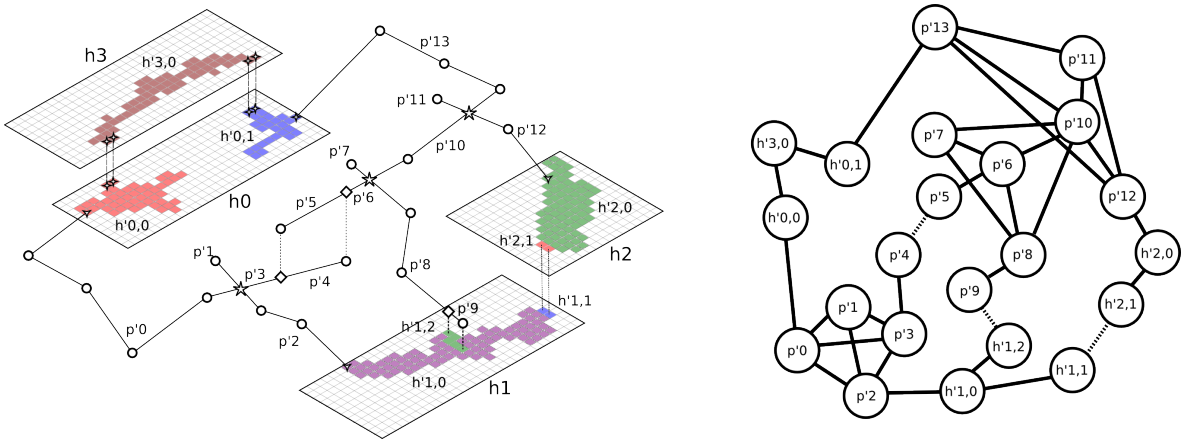

Figure 2.13: Abstract graph after superpositions were detected. Extra graph vertices were added as described previously, and superpositions between graph vertices are noted with a dashed line.

This ensures that all graph vertices will have a sub-graph index, but more importantly that each index will be associated with a similar number of graph vertices[4]. Following the previous example, figure 2.14 shows the evolution of the sub-graph index propagation in the graph.

2.4.2.3 Merge

By starting from superposed graph vertices, sub-graph index propagation ensures that enough monovalued sub-graphs will be used. However it can lead to a massive over-estimation of the number of required sub-graphs, especially when the input interpretation data is dense. This being said, it occurs that many of the sub-graphs can be merged together.

[4]Exact same number is not reached as it depends on the graph shape for propagation. Moreover, two graph vertices can have a different geometrical extent, e.g. polylines being often smaller than heightmap connected components.

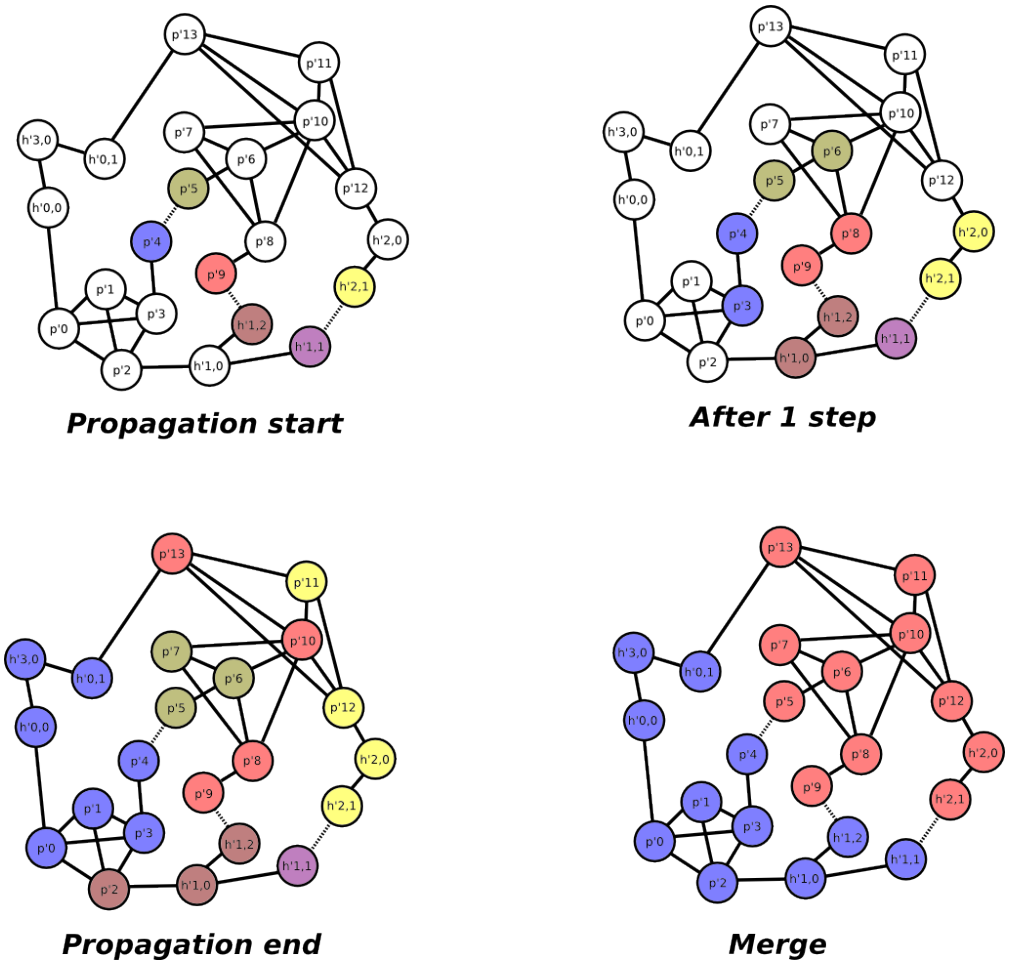

Figure 2.14: Before propagation, only superposed graph vertices are given an index, symbolized here by a color. At each propagation step, each color gets propagated in the graph until all graph vertices have a color. After propagation end, sub-graphs (i.e. graph vertices of the same color) are merged together to reduce the sub-graph count, as explained in the following section.

Indeed, let us consider a pair of sub-graphs $(G_i, G_j), i \neq j$. If they are connected by a graph edge and do not have graph vertices that overlap vertically, then they are merged together. An example of merging can be found in figure 2.14. We call $\tilde{G}_i$ the merged sub-graphs.

2.4.3 Envelope Computation

At this point the partitioning problem is solved as we found a partition of G into a relatively low number of monovalued sub-graphs $\tilde{G}_i$. In order for a sub-graph $\tilde{G}_i$ to be gridded, it is however necessary to convert it to a patch P_i and compute its envelope. As detailed in the next section, each sub-graph will indeed be converted into a heightmap, and its polylines and heightmap connected components will be rasterized into *constraint pixels*.

Algorithm 1 Sub-graph index propagation

Procedure propagate $(G, \{G_i\})$
Input:
G ▷ Connected abstract graph
$\{G_i\}$ ▷ Indexed sub-graphs (superposed graph vertices only at start)
Algorithm:
1: *vertices* ← FIFO list with all vertices of $\{G_i\}$
2: **while** *vertices* is not empty **do**
3: Pop a, the first vertex of *vertices*
4: **for** Each unindexed vertex b touching a **do**
5: *index* ← a's index
6: Index b with *index*
7: Add b to *vertices*
8: **end for**
9: Add a to G_{index}
10: **end while**
End procedure

For a patch P_i, the envelope is the combination of two objects:

- A *mask* indicating for each pixel of its heightmap H_i whether it is to be gridded or not. This mask will be encoded in the heightmap H_i using a boolean value, for example *true* if inside envelopes, $false$ otherwise;
- A set of *junction points*, i.e. pixels that have neighbor pixels in another patch. This will be encoded in the neighbor data structure N_i.

The envelope will therefore be the domain around constraint pixels, i.e. pixels coming from interpretation data. There are methods to compute the envelope (or "hull") of a set of pixels: one could consider using the pixels' convex shape [KS86] or alpha shape [EKS83], but in our case this leads to masks that are too large and hence does not prevent extrapolation.

An efficient and intuitive way to construct this mask is instead to use the closing morphological operator against the constraint pixels of each patch heightmap. Closing is actually the succession of a dilatation and an erosion, both using a structural element of size $d_C \in \mathbb{N}^*$ pixels. When a relevant value of d_C is chosen, holes between constraint pixels are closed by the dilatation while extrapolation is avoided because of the erosion. We therefore propose the following steps to find the envelope of each patch:

- **Heigtmap conversion**. Turn each sub-graph $\tilde{G}_i$ into a patch heightmap H_i initialized with constraint pixels;
- **Dilatation**. A dilated envelope is created independently for each patch;

- **Dilated envelope restriction and junction point location**. For each patch pair that is connected topologically by an edge of G, restrict dilated envelopes to ensure smooth connection along a set of junction points;
- **Joint erosion**. Each dilated envelope is eroded to prevent extrapolation. This is done simultaneously, i.e. on the multivalued surface.

2.4.3.1 Heightmap Conversion

Each sub-graph can now be converted into an image of size $W \times H$, the survey size (see figure 2.15). We therefore associate each sub-graph $\tilde{G}_i$ with its corresponding patch P_i of heightmap H_i whose pixel (u, v) contains the height z for any interpretation data (u, v, z) in $\tilde{G}_i$:

$$H_i : \begin{cases} \mathcal{D} & \to \mathbb{R} \\ (u,v) & \mapsto \begin{cases} z' & \text{if } \exists M' = (x', y', z') \in \tilde{G}_i, \\ & \quad (u,v) = (x', y') \\ \nu & \text{(null value) otherwise} \end{cases} \end{cases} \tag{2.15}$$

Remarks:

- ν (null value) is a magic value designating a patch pixel that is not yet valued (it is not a constraint pixel). The value of such a pixel will be set during gridding. It must not be confounded with γ pixels (recall figure 2.2) which define locations outside the heightmap, i.e. pixels that will *never* be valued;
- Concretely, H_i is obtained by rasterizing any polyline in $\tilde{G}_i$ and projecting any heightmap connected component in $\tilde{G}_i$;
- The "intra-patch" connectivity information once stored explicitly in the vertices and edges of $\tilde{G}_i$ is now replaced by the natural neighborhood of the pixels in P_i. The "inter-patch" connectivity, i.e. the topological connection between $\tilde{G}_i$ and its potential neighbor sub-graphs is for now lost though, but it will be stored in the neighbor data structure N_i when computing the dilated envelope restriction and the joint erosion.

2.4.3.2 Dilatation

Although image morphological operators are typically defined by kernels associated with structural elements, numerical implementations are faster when using Euclidean Distance Maps (EDM). It can be shown that both dilatation and erosion are equivalent to the thresholding of an EDM[5] [Rus98]. Using an EDM is faster than using masks because there are

[5]This is for disk-shaped structural elements and distance maps based on the $L2$ norm, because the disk is the topological ball associated with the $L2$ norm in $\mathbb{R}^2$.

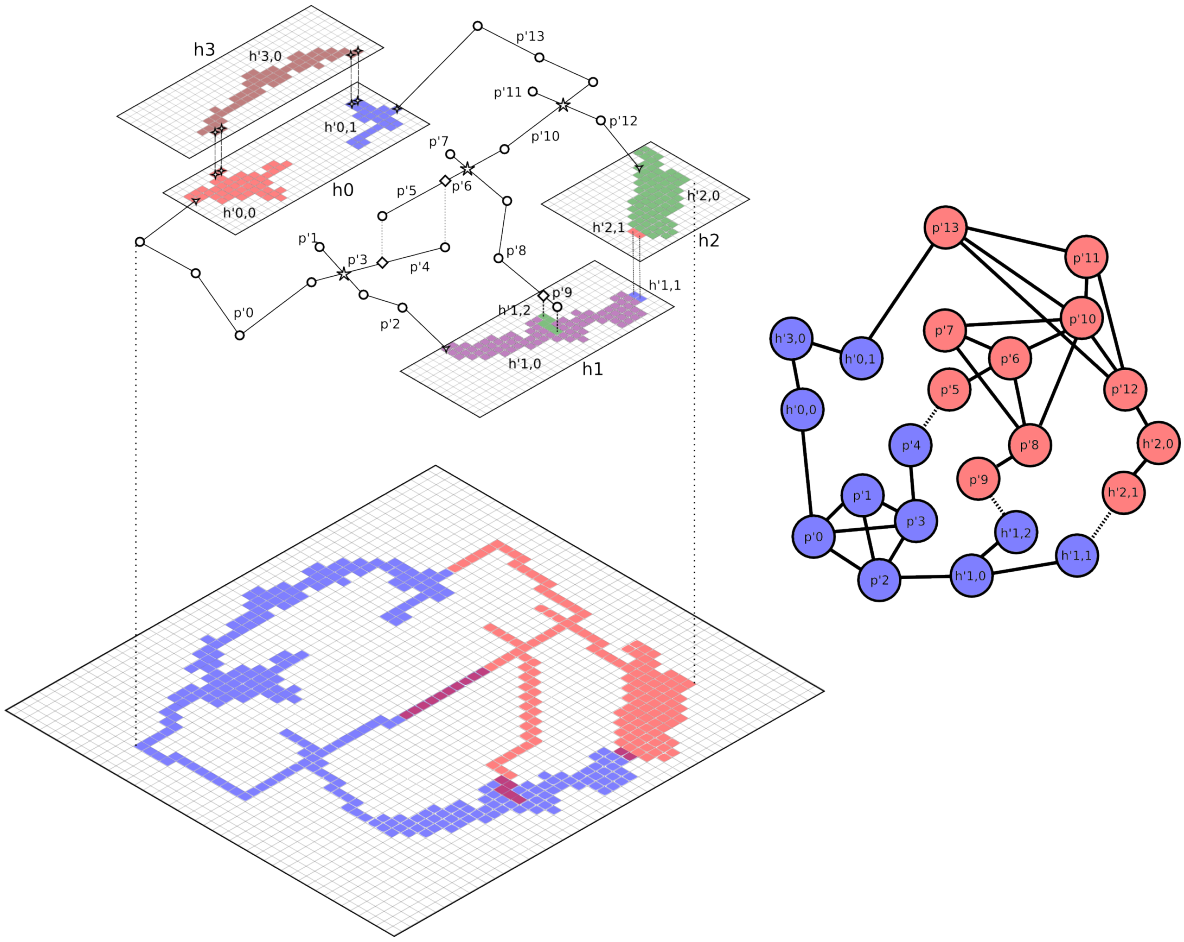

Figure 2.15: Following the example in figure 2.14, each subgraph G_i is converted into a heightmap H_i. The superposed result is displayed here. Overlapping graph vertices in G lead to overlapping pixels in this image. Polylines are rasterized and heightmap connected components are projected in order to get this raster representation.

efficient $\mathcal{O}(W \cdot H)$ algorithms to compute an approximated distance map [Dan80]; [TH16]. Using approximations is tolerable in our case as the envelope does not require pixel-perfect precision and those errors are small [Gre04].

Recall constraints are the non-ν pixels of heightmap H_i. We therefore construct the map of *distance to constraints* DC_i. Being a distance map, each pixel of DC_i has a positive value and is only zero on the location of constraints, i.e. non-ν pixels. DC_i is defined by:

$$DC_i : \begin{cases} \mathcal{D} & \to \mathbb{R}^+ \\ (u, v) & \mapsto \text{distance to closest non-}\nu\text{ pixel} \end{cases} \tag{2.16}$$

We then define the *dilated envelope* DE_i by thresholding the distance map DC_i:

$$DE_i \doteq \{(u, v) \in \mathcal{D}, DC_i(u, v) \leq d_C\} \tag{2.17}$$

Using this thresholding method, it is possible to obtain the dilated envelopes, as depicted by figure 2.16.

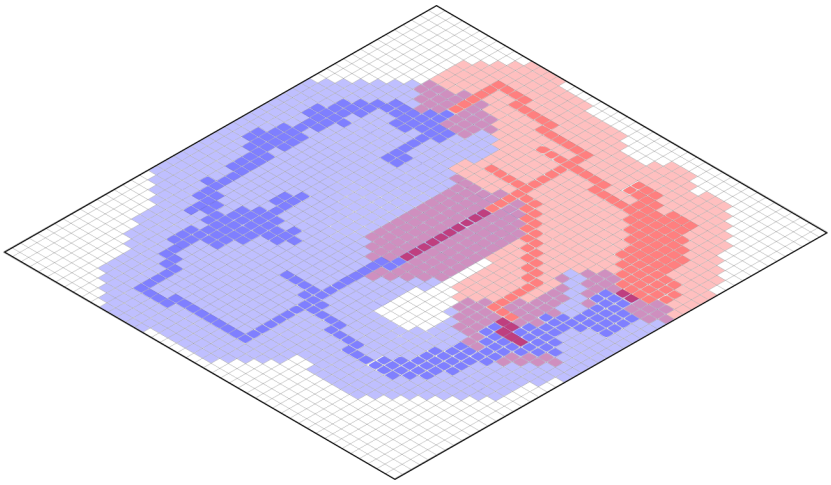

Figure 2.16: An example of dilated envelopes. They overlap when constraints of the two patches are both closer than d_C.

2.4.3.3 Dilated Envelope Restriction and Junction Point Location

Before computing the erosion, we want dilated envelopes to join along a *boundary curve* without overlapping around the known topological connections between two patches, i.e. near edges of G that have vertices from two sub-graphs (G_i, G_j), $i \neq j$. Meanwhile, we also want to allow and preserve dilated envelopes overlapping around superposed constraints (for example in figure 2.16, superposed polyline edges must eventually lead to superposed parts of the final multivalued surface). This can be handled simultaneously by a criteria map using the following procedure.

Compute Criteria Map: Each boundary between two patches P_i and P_j should be located "in the middle" of the two patches' dilated envelopes. For this reason we compute a *criteria map* $C_{i,j}$ derived from distance maps DC_i and DC_j: see figure 2.17 for an example. The criteria map can be defined as:

$$C_{i,j} : \begin{cases} \mathcal{D} & \to \mathbb{R} \\ (u,v) & \mapsto DC_i(u,v) - DC_j(u,v) \end{cases} \tag{2.18}$$

Remarks:

- A pixel in criteria map $C_{i,j}$ has negative value when closer to patch i than patch j;
- A pixel in criteria map $C_{i,j}$ has positive value when closer to patch j than patch i;

- We want the boundary curve between patches i and j to be defined the location of sign change in $C_{i,j}$;
- However, the boundary curve should not be defined around superposed constraints, i.e. on pixels valued 0.

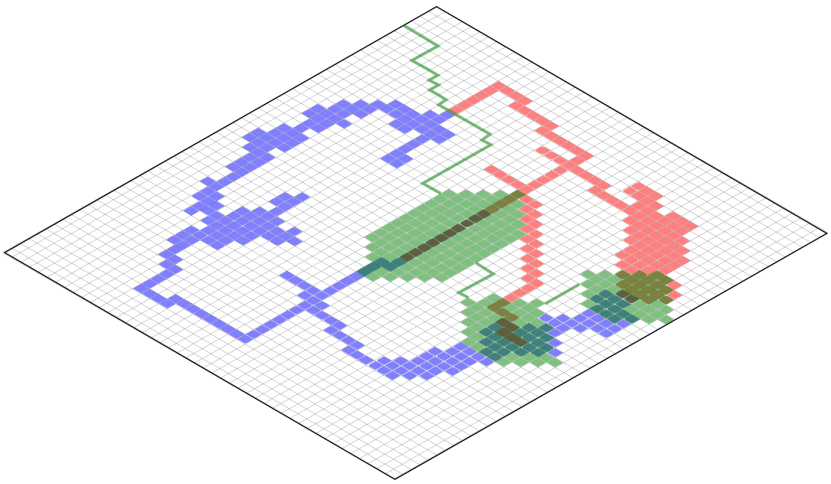

Figure 2.17: In green is depicted the isovalue 0 in the criteria map used in order to define the boundary shape. It is "between" the pixels unless on the "0 areas" associated with superposed constraint pixels, where the boundary curve should not be defined.

On a side note, the use of EDM thresholding and criteria maps shows how 3D implicit methods, whose performance was commented in section 1.3.1, can be somehow used on a 2D scalar field that is built only from geometry, not from seismic cube samples. This makes our approach both faster and independent from seismic cube SNR (though noisy cubes often introduce uncertainties on picked polylines and prevent the propagation of heightmap fragments).

Restrict Envelopes: In order to restrict the dilated envelopes, we introduce the set of locations where the criteria map is positive and negative (excluding locations where criteria is zero):

$$\begin{cases} C_{i,j}^{+} \doteq \{(u,v) \in \mathcal{D}, C_{i,j}(u,v) > 0\} \\ C_{i,j}^{-} \doteq \{(u,v) \in \mathcal{D}, C_{i,j}(u,v) < 0\} \end{cases} \tag{2.19}$$

We then define the *restricted dilated envelopes* RDE_i and RDE_j of patches i and j as depicted by figures 2.18 and 2.19: the restricted dilated envelope of patch i is the dilated envelope of patch i, but deprived of areas where the criteria map $C_{i,j}$ is strictly positive, i.e. when closer to patch j. "0 areas" are kept on the restricted dilated envelope so superposed envelopes can exist. This can be noted as:

$$\begin{cases} RDE_i \doteq DE_i \setminus C_{i,j}^{+} \\ RDE_j \doteq DE_j \setminus C_{i,j}^{-} \end{cases} \tag{2.20}$$

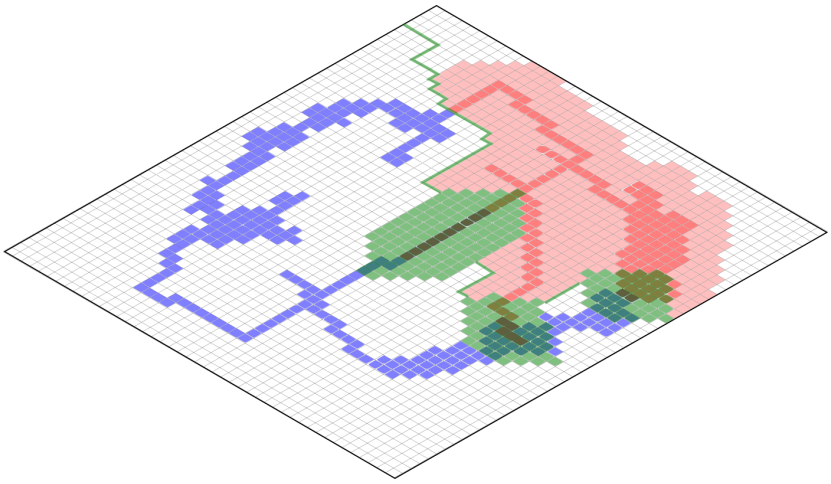

Figure 2.18: By keeping "0 areas" while removing envelope beyond the location of sign change in the criteria map, it is possible to define the restricted dilated envelope, here for the right patch as an example.

Locate Junction Points: Once restricted, the dilated envelopes will perfectly join at the boundary. At this point, the neighbor data structure N_i of each patch P_i can be updated as locally around the boundary, the per-pixel connections between any two patches are known.

2.4.3.4 Joint Erosion.

Whereas dilatation could be computed independently for each patch in previous section, it is required to consider the patch system as a whole during erosion. Once again, using kernel-based morphological operators works but is extremely slow. Using EDM thresholding still speeds up the process, but the fast two-pass algorithm previously used [Dan80] cannot be easily adapted to a non-manifold support, in our case the patch system.

Instead we propose to use a fast-marching algorithm [TH16] that propagates pixel by pixel the *distance from outside* the restricted dilated envelope on a "multivalued EDM" DO_i, i.e. a patch system whose heightmaps are EDM. As our patch system model clearly defines neighborhood relations in the entire horizon using the neighbor data structures N_i, fast-marching implementation is straightforward.

Once computed, the multivalued EDM DO_i can be thresholded using the closing distance d_C, leading to the definition of the *eroded envelope* EE_i. The erosion of the example patches is shown in figures 2.19 and 2.20, depicting the envelopes respectively before and after erosion.

The eroded envelope EE_i is therefore:

$$EE_i \doteq \{(u,v) \in \mathcal{D}, DO_i(u,v) \geq d_C\} \tag{2.21}$$

Along with restricted dilated envelopes, boundaries are also eroded. The neighbor data

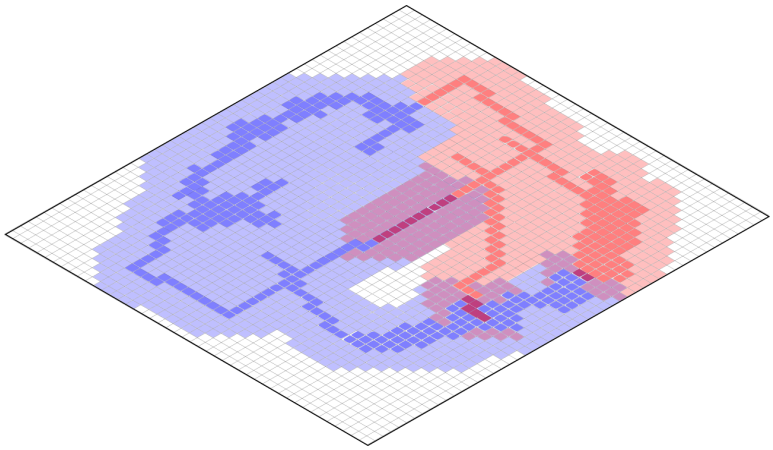

Figure 2.19: Restricted dilated envelopes before erosion. Notice the areas "outside" constraint pixels where extrapolation would occur if no erosion was performed.

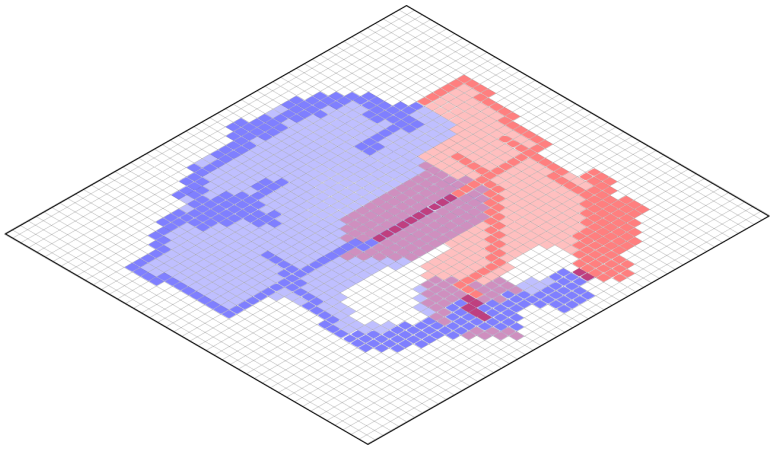

Figure 2.20: Cut envelopes after erosion. They still connect along a neat boundary curve, but erosion removed envelope "outside" where extrapolation would have occurred.

structures N_i needs therefore to be updated again at this point to only connect together points that are still on the envelope. By construction, we now have an eroded envelope EE_i for each patch P_i, and all eroded envelopes join nicely along the eroded boundaries.

It is now time to update the patch heightmaps with the envelope information: from now on, each pixel of H_i outside of the eroded envelope EE_i is associated with a boolean value $false$ (outside patch) in the envelope mask. At this point the patch system is ready for gridding as described in section 2.3.

2.5 Results

Examples of reconstructed surfaces using both synthetic and real data are presented here to illustrate the action of our method on polylines and heightmap fragments interpreted by geologists. At the time this is written, the handling of heightmaps as input is still under implementation. Quantitative results will therefore only be provided for polyline inputs.

2.5.1 Illustration on Synthetic Data

The proposed model of patch system as well as the multivalued gridding approach we developed provide good results on both synthetic and real data. First, we will comment here the interpolation of sparse synthetic polylines and heightmap fragments into a patch system.

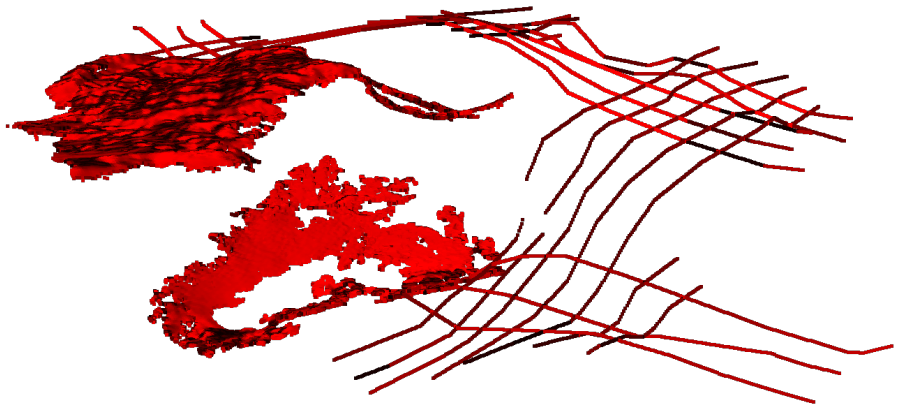

Figure 2.21: Example of synthetic data: sparse polylines and heightmap fragments.

Figure 2.21 shows the input polylines as they would be picked by geologists on a survey. Many superposed zones will be detected in the multivalued scan, leading to the indexation of a lot of sub-graphs in the index propagation stage. The final sub-graph count will not be excessive though, because of the merge step. Eventually, only 2 patches are necessary (see figure 2.22).

After conversion to heightmaps, envelope computation begins. This will lead to the definition of valid envelope masks and neighbor data structures, used by the gridding process. The resulting patch system is depicted in figure 2.23. The global surface is smooth even along patch boundaries.

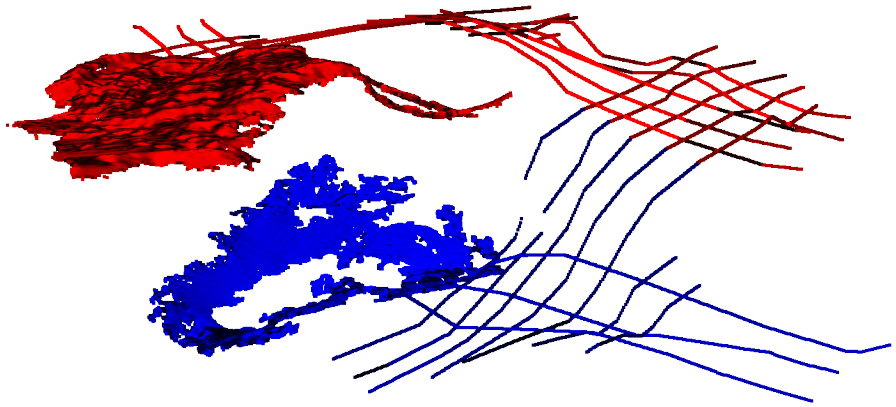

Figure 2.22: After patch index propagation and merging, two patch indices have been attributed.

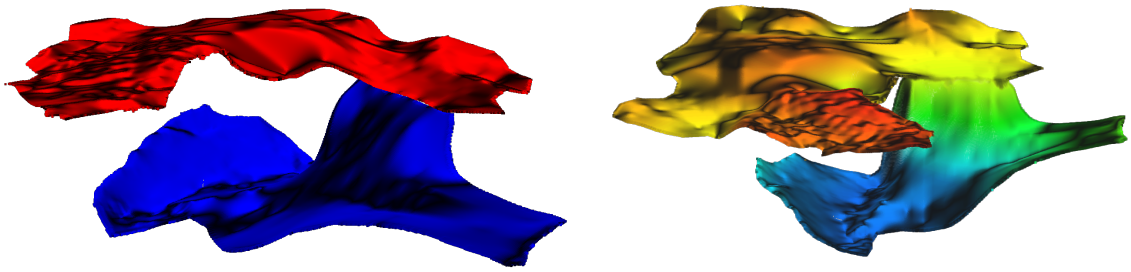

Figure 2.23: Following envelope computation and gridding, a patch system is created (left). Patches join smoothly along a boundary curve (right, with another viewing angle and a color map indicating elevation).

2.5.2 Illustration on Real Data

After this simple and readable example, we will show that complex real data can be addressed with our approach. In figure 2.24, a sparse set of polylines describing a complex reverse-faulted horizons are reported. There are at most 3 vertical superpositions in the data: in other words, at least 3 patches will be used to represent the multivalued surface with a patch system.

In practice, as illustrated in figure 2.25, an extra patch will be created in this case. Indeed, recall we use a heuristic graph propagation algorithm for the sake of speed and simplicity, which means we cannot guarantee that the number of produced patch is the minimum one. This is not an issue as no data set was found to generate a pathological number of patches, and the additional patches that are sometimes produced are a very low price to pay in order to have a fast and straightforward approach.

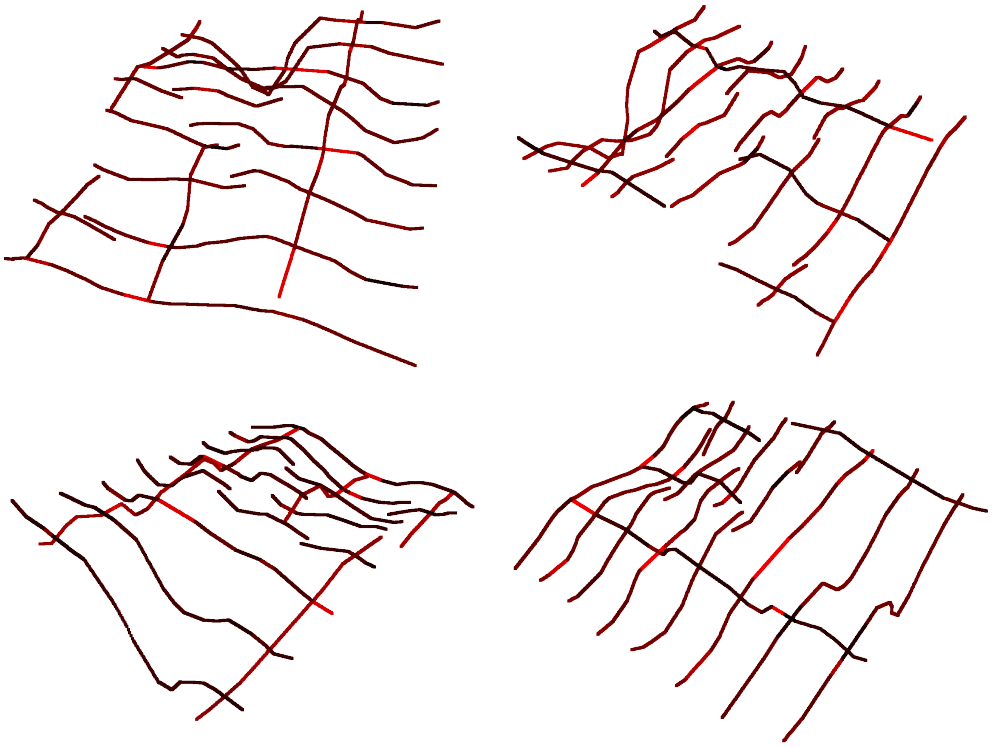

Figure 2.24: A set of 12 input polylines describing a horizon made multivalued by several faults. This input is sparse despite representing a complex geometry, making it quite difficult to tackle.

2.5.3 Performance Considerations

Our multivalued gridding scheme is based on a preparation stage, where the number of patches is found by a heuristic graph propagation algorithm. The graph vertices represent high-level interpretation data, i.e. polylines or heightmap fragments, instead of, say, polyline vertices or heightmap pixels. This means the graph has a relatively small size, making the propagation very fast. Interpretation data is then projected on patches, using either straightforward vertical projection (for heightmap fragments) or rasterization (for polylines).

This quick preparation step is then followed by a standard 2D interpolation, namely gridding. Once again, this image-based method is fast and robust, making the overall multivalued gridding almost instantaneous for human users. Table 2.1 reports the run-time of our multivalued gridding method for the real data set we just presented. We considered varying levels of input density in order to show that the algorithm is scalable. Indeed, the run-time of the algorithm is linear in the number of input polyline vertices, and quadratic in survey resolution for EDM computation and gridding – compared to the sometimes exponential costs of graph partitioning approaches, and the cubic complexity inherent to any 3D interpolation approach. Another advantage of using gridding is that control over surface smoothness and input data fitting is obtained through the α and β coefficients presented in section 2.3.

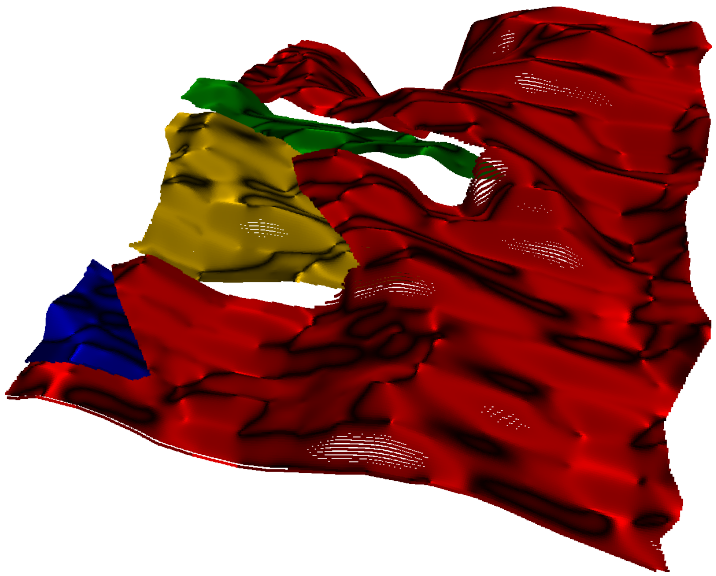

Figure 2.25: Reconstructed surface displayed as a point cloud, using one color per patch. Note that 4 patches were created for this horizon, though there are at most 3 superpositions in the data: this is because our approach is based on a heuristic.

Polylines	Vertices	Run-time (ms)
12	194	207
25	401	423
51	937	892
73	1392	1203

Table 2.1: Run-time of the multivalued gridding method on the real data set, at varying polyline densities. Linear scaling in performance is observed. The survey size was used for resolution, in our case $1137 \cdot 2227$ pixels, i.e. a bit more than 2 megapixels.

2.5.4 Summary

This multivalued gridding method is an extension of existing work on the reconstruction of horizons from polylines [Bau+18b], and had lead to an article currently being submitted to the Communications in Computer and Information Sciences. This method is a natural extension of monovalued models and reconstruction methods, meaning our approach will induce only minimal software refactoring efforts. Moreover, it is very fast and easy to implement. The cost that must be paid for this performance is the machinery of multivalued gridding preparation, i.e. the unified graph construction, partitioning and envelope computation.

An interesting feature would be the handling of faults during the gridding stage. Faults are the result of mechanical failure within a geological object. These discontinuities can displace rock formations on a wide range of distances, some largely visible even at the seismic scale.

Horizons can be for example cut and displaced by faults – normal faults being one of the primary source of multivalued horizons. For this reason it makes sense to prevent access to neighbor pixels on the opposite side of a fault while gridding. This is a standard feature of current monovalued gridding implementations in modern geophysics software, and would be appreciated for multivalued horizons as well. Beyond new features, many optimizations could also be conducted in order to reduce the run-time and memory footprint of the algorithmic chain. From multi-grid schemes, multi-threading and compression strategies to constraints for the sub-graph index propagation, a lot of progress can be made to support ever larger horizons.

Chapter 3

Reconstructing Horizons with Salt Domes

Contents

It is now time to design a reconstruction method for horizons representing salt domes. They will be modeled by a triangle mesh, so we need to eventually triangulate our input polylines. The idea summarized in section 3.1 is to first parameterize the polylines to the plane, i.e. "flatten" them. This is the subject of section 3.2. Once in 2D, the polylines can be rasterized on a regular grid, and monovalued gridding can once again be used for interpolation. As explained in section 3.3, at this point, triangulation is straightforward, and the triangle mesh can then be transformed back into 3D space, yielding the reconstructed surface. We will also comment the performance issues and optimizations related to this reconstruction scheme.

3.1 Sketch of Proposed Method

3.1.1 Problem Presentation

The problem we intend to solve, illustrated by figure 3.1, is the following: how do we reconstruct salt domes from sparse, non-uniformly distributed polylines picked in unorganized cross-sections?

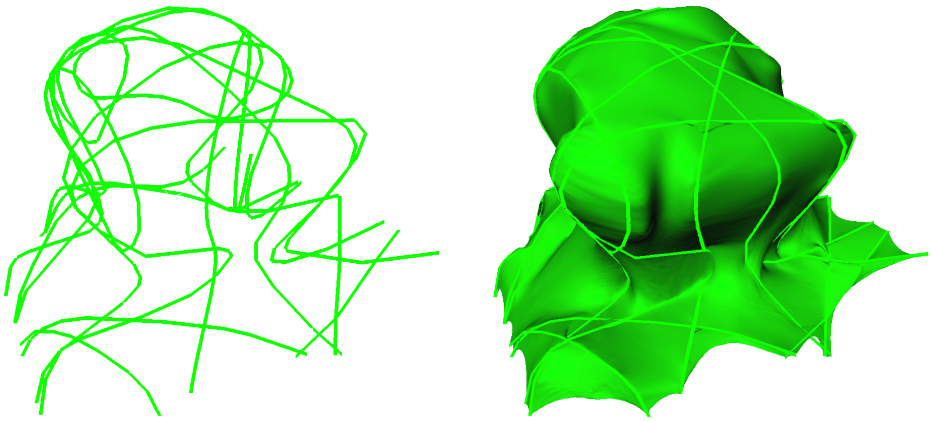

Figure 3.1: Sparse set of input polylines picked in unorganized cross-sections (left). We want to reconstruct the surface described by the polylines (right).

3.1.2 Proposed Method, Motivations

The main difficulty we face is that surface reconstruction is often difficult in three dimensions, especially for sparse and non uniformly distributed data. In the spirit of previous point-cloud triangulation methods [FR01], we therefore propose to transform the problem to a two-dimensional one. This is moreover an improvement upon a previous proposal [Bau+18a]. Namely, we use the following workflow (illustrated by figure 3.2):

1. The polylines are parameterized over a subset of the plane, in other words we "flatten" them. This way, when viewed from the top, polylines are not superposed anymore except at their intersection points. The parameters u, v along with the z coordinate form the so-called *deformed* space, in which the polylines (and the reconstructed surface) are monovalued. This process is explained in section 3.2;
2. The flattened polylines are then interpolated in the plane. We propose to use gridding, a robust method frequently used for monovalued horizon interpolation (this is explained in section 2.3).

As a general assumption, we consider that the input polylines have clean intersection, i.e., do not "hover" on top of each other. In practical applications, a preprocessing step to snap together close polylines might be needed in order to satisfy this requirement.

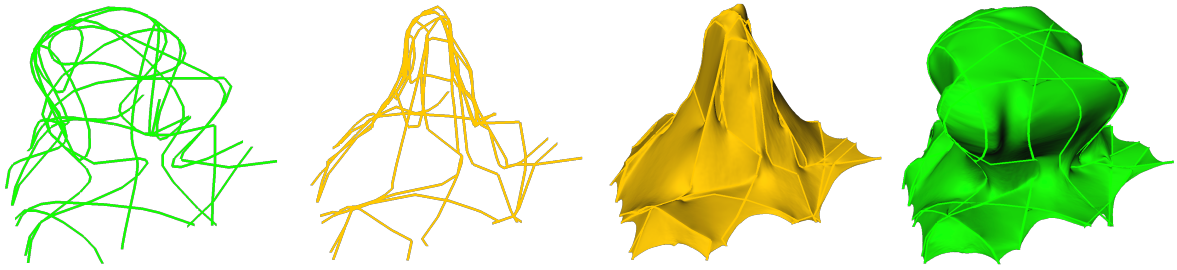

Figure 3.2: Our proposed workflow, from left to right. Input polylines, picked in unorganized cross-sections, in three-dimensional space (x, y, z) – Polylines in deformed space (u, v, z), i.e., as a z-valued heightmap. They are now monovalued – Interpolated surface in deformed space – Interpolated surface projected back in three-dimensional space.

3.2 Polylines Parameterization: a Barycentric Approach

Our objective is to deform the space where input polylines are picked so that when viewed from the top, they do not overlap anymore. In this abstract deformed space, the horizon depicting a salt dome is hence monovalued, and can be interpolated using standard two-dimensional interpolation methods, for example gridding (see section 2.3). The interpolated surface can then be transformed back from deformed space to normal space, yielding the reconstructed three-dimensional surface.

3.2.1 Parameterization and Discretization

3.2.1.1 Smooth Case

In the smooth theory, *conformal* (angle-preserving) parameterizations of surfaces are widely studied and posses a large body of literature. Conformal parametrizations can be proved to exist for any three-dimensional surface havig disk-like topology, and they have interesting properties that enable their efficient calculation [HLS07]; [FH54]; [SC17]. In particular, if we fix the target boundary of the parameterization, a conformal map is *harmonic*, i.e., every point is in some sense the average of its neighbors, using the Laplace-Beltrami operator Δ_{LB} which takes the shape of the 3D surface into account.

3.2.1.2 Discrete Case

We are interested here in the implementation of our method in a discrete setting, where surfaces are often represented as a collection of elements, often triangles. In this restricted setting, strict conformal maps, meaning maps that preserve exactly all the angles in the mesh, do not exist unless the surface is developable [Cra15]. For example, a closed 3D surface is completely determined (up to scale) by giving the angles of each triangle, while an open disk-like surface is completely determined by giving the angles and one triangle. Indeed,

consider the example of a 3D conical fan made up of a set of triangles meeting at one common summit. In this case, the sum of 3D angles at the common vertex is in general different than 2π (depending on the local Gaussian curvature) so that no strict angle-preserving mapping to the plane is possible since then the sum of angles would have to be exactly 2π. Thus, a different, less rigid approach such as *barycentric mappings* will be used.

3.2.1.3 Barycentric Maps: State of the Art

Barycentric methods find their origin in graph drawing problems [Tut60], but are well suited for mesh parameterization as well, being easy to understand, implement and often yielding satisfying results [HLS07]. Barycentric methods can also be adapted for point clouds [FR01], or even for other supports such as arbitrary polygons [Flo03].

However, to the extent of our knowledge, adapting barycentric methods to polyline parameterization has not been the subject of much academical interest. In the following section, we will therefore propose a natural extension of barycentric methods for polylines.

3.2.2 A Barycentric Mapping for Polylines

Let us consider a set of $N \in \mathbb{N}$ points $\mathbf{x}_i \in \mathbb{R}^3$. We want to find, for each 3D point $\mathbf{x}_i$, a 2D point in parameter space $\mathbf{u}_i \in \mathbb{R}^2$, while preserving the distribution of points from space to the plane. Such a map will represent the parameterization, or flattening, we are looking for.

Barycentric methods solve this problem by enforcing a local equilibrium, i.e. by making each 2D point $\mathbf{u}_i$ the barycenter of its 2D neighbors, with weights based on the relative distribution of their correspondent 3D points $\mathbf{x}_i$. As previously shown [FR01], by fixing the exterior *boundary points* in space and under some sufficient conditions that do apply here, the system of local barycentric equations leads to a sparse global linear system that has a unique solution.

Without loss of generality, we can assume that the first $n_B < N$ points, in the list of input points $\mathbf{x}_i, i \in [\![0, n_B - 1]\!]$ are the *boundary points*, that is, the endpoints of the input polylines. For each boundary point (x, y, z), we associate a parameter point $(u, v) \equiv (x, y)$, i.e., we parameterize the boundary points by simple vertical projection[1]. It follows that the $n_I = N - n_B$ remaining 3D points are *interior points*, and have to be associated with a parameter point $\mathbf{u}_i \in \mathbb{R}^2$ using a barycentric equation:

$$\mathbf{u}_i \doteq \sum_{j \in \mathcal{N}_i} \lambda_{i,j} \mathbf{u}_j \tag{3.1}$$

Where:

[1] We make the assumption that they are not vertically superposed and are indeed on the convex outside boundary of our input data set when viewed from top. Vertical superposition on the boundary can be resolved via a small perturbation of the point coordinates.

- $\mathcal{N}_i$ is the *neighborhood* of point i, namely the set containing the indices of the points considered close to point i in space;
- $\lambda_{i,j} \in \mathbb{R}$ are the *coefficients* associated with each ordered pair of points i and j. In order to be interpreted as barycentric coordinates, they are required to be strictly positive and sum to one for each point i [HLS07]:

$$\begin{cases} \forall j \in \mathcal{N}_i, \lambda_{i,j} > 0 \\ \sum\limits_{j \in \mathcal{N}_i} \lambda_{i,j} = 1 \end{cases} \tag{3.2}$$

Note that in general $\lambda_{i,j} \neq \lambda_{j,i}$. Barycentric maps are therefore defined by giving, for each point, its neighborhood as well as the associated set of coefficients. Various choices of neighborhoods (see section 3.2.3) and coefficients (see section 3.2.4) have been explored in order to find a barycentric map parameterizing the polylines on the plane, while introducing as few distortions as possible.

3.2.3 Barycenter Neighborhoods

In this section, we assume that we have a method that produces a valid coefficient $\lambda_{i,j}$ for any ordered pair of points i and j. We will focus on the definition of a set of neighborhoods suitable for polyline parameterization.

3.2.3.1 Topological Neighborhood

When considering connected data such as a triangle mesh, a very natural neighborhood for every vertex i is available: the set of vertices connected by an edge to i [HLS07]. The same definition can be naturally transposed to our situation, by considering the set of polylines as a graph. After snapping, the network of input polylines can be seen as a connected graph, whose edges can be used to define a *topological neighborhood* for each vertex i, denoted $\mathcal{N}_i^T$ (see figure 3.3 for an example).

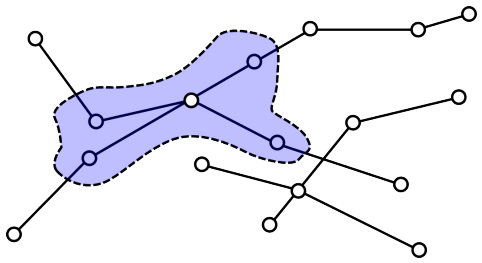

Figure 3.3: Topological neighbors are graph vertices connected by an edge.

This neighborhood definition takes polyline edges into account and provides balance at polyline crossings. In fact, the coordinates of vertices with more than two connected points

can be represented as a linear combination of the coordinates of their topological neighbors, satisfying the barycentric equation (3.1). However, the topological neighborhood does not take into account polylines that are very close geometrically without touching. This means that the distribution of points in the plane will in general look very different from the distribution in 3D space. It also will not unfold loops, i.e. cycles in the graph containing exactly two crossings (see figure 3.4), which are bound to become aligned by the barycentric equation. This will prevent polylines from being separated by a discrete interpolation process such as gridding, and must be avoided. The topological neighborhood is therefore not well adapted to polyline parameterization.

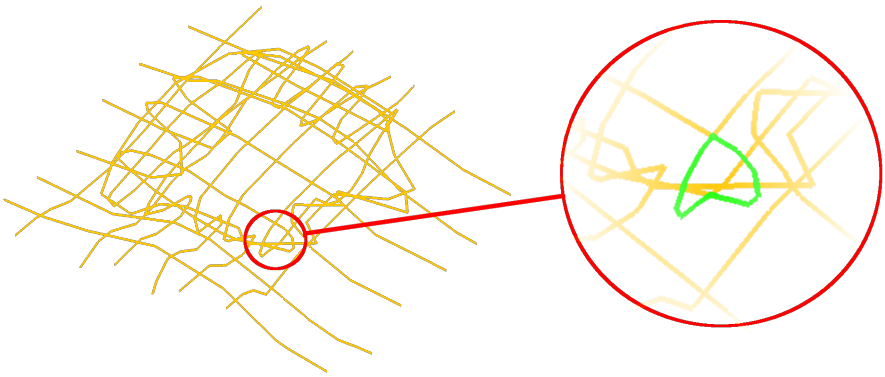

Figure 3.4: An example of loop, namely a cycle in the graph containing exactly two polyline crossings. Because parameterized vertices along the loop must be a linear combination of end points, they will end up aligned in the plane and the loop will not be unfolded.

3.2.3.2 Geometrical Neighborhood

In contrast with triangle meshes, point clouds do not contain any connectivity information. A simple neighborhood for any vertex i can however be defined by considering any point within a sphere of given radius $r \in \mathbb{R}_+$ centered on i [FR01], producing a *geometrical neighborhood* that we will denote $\mathcal{N}_i^G$. This is illustrated by figure 3.5.

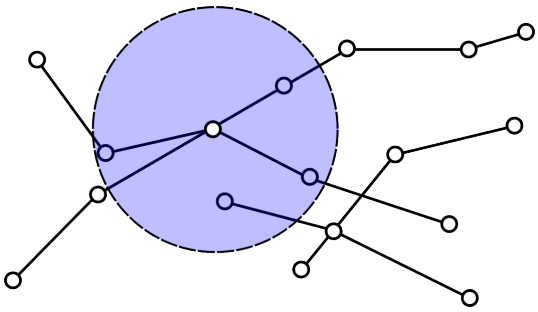

Figure 3.5: Geometrical neighbors are constructed from vertices within a given radius r.

This neighborhood takes geometrically close, but disconnected, polylines into account, making it easier to satisfy the barycentric equation (3.1). It also unfolds loops by adding

neighbors on the sides. This being said, it completely ignores the existence of polyline edges, and can have a shrinking effect when a large sphere radius is used[2]. It also has an intrinsic resolution limit on the reconstructed topology, as any structure of the final salt dome surface smaller than the geometrical radius will not be resolved. Similarly to the topological neighborhood, the geometrical neighborhood alone is not sufficient for a correct parameterization of polylines.

It should be noted that it is possible to mitigate the drawbacks of the shrinking effect and intrinsic resolution of the geometrical neighborhood by estimating locally an appropriate sphere radius r_i. This can be done iteratively, by looking for the smallest radius that puts the current point i inside the 3D convex hull of its neighbors. This will automatically result in a balancing barycentric equilibrium, satisfying equation (3.1) without increasing the size of neighborhoods too much and suffering from the detrimental effects of a larger-than-necessary radius.

3.2.3.3 Our Proposal

As discussed previously, topological or geometrical neighborhoods alone are not enough for a correct parameterization of polylines. This was to be expected, as polyline data falls somewhere in between the 0D disconnected point clouds and the 2D manifold-connected triangle meshes. In order to leverage the advantages of both neighborhoods, we propose to interpolate the barycenter produced by each method. To this end, recall that we have denoted $\lambda_{i,j}$ the coefficient between vertex i and vertex j. We can therefore compute the topological and geometrical barycenters as:

$$\begin{cases} \mathbf{u}_i^T \doteq \sum\limits_{j \in \mathcal{N}_i^T} \lambda_{i,j} \mathbf{u}_j \\ \mathbf{u}_i^G \doteq \sum\limits_{j \in \mathcal{N}_i^G} \lambda_{i,j} \mathbf{u}_j \end{cases} \tag{3.3}$$

An interpolation between the two can be obtained via an interpolation coefficient $\alpha \in [0, 1]$, yielding the parameterized barycenter:

$$\mathbf{u}_i \doteq (1 - \alpha)\mathbf{u}_i^T + \alpha \mathbf{u}_i^G \tag{3.4}$$

We have found through numerical experiments that the addition of a global interpolation coefficient is already enough to greatly improve the quality of the parameterization. It can however present some local problems, where a higher or lower coefficient value would have been more desirable. To this end, it was proposed in [Bau+18a] to compute a local coefficient α_i for every vertex i, namely the coefficient that leads to a more appropriate 3D barycenter:

$$\alpha_i \doteq \arg\min_{\alpha} \left\| \mathbf{x}_i - \left[(1 - \alpha) \sum_{j \in \mathcal{N}_i^T} \lambda_{i,j} \mathbf{x}_j + \alpha \sum_{j \in \mathcal{N}_i^G} \lambda_{i,j} \mathbf{x}_j \right] \right\|^2 \tag{3.5}$$

[2]The barycentric equation makes each point the weighted average of its neighbors. In other words, it has a low-pass filtering effect on data, namely it attracts points towards the center of mass hence the shrinking effect. This only becomes detrimental for big values of r though.

In this sense the interpolation of neighborhoods is optimal and indeed yields very good results, especially compared to the use of a single neighborhood. This choice of neighborhood is very well suited to polyline parameterization, producing fully flattened polylines with unfolded loops, while preserving distribution of points from space to the plane. Results will be discussed in more details in section 3.4.

3.2.4 Barycenter Coefficients

We will now assume that the neighborhood $\mathcal{N}_i$ of each point i is given, and consider the various sets of coefficients $\lambda_{i,j}$ used to define the barycentric map, in order to select an appropriate method for the parameterization of the input polylines.

3.2.4.1 Constant Coefficients

As previously mentioned, the origins of barycentric methods can be traced back to the problem of graph drawing [Tut60], that is, how to draw a given (planar) graph on paper without overlapping edges. A barycentric approach with constant coefficients is enough in this context. For polyline parameterization, such a choice would not take polyline edge lengths into account, which makes constant coefficients unsuitable.

3.2.4.2 Inverse-Distance Coefficients

The relationship between graph drawing and mesh parameterization is clear: by considering a triangle mesh as a graph of its vertices, drawing a graph on paper is equivalent to provide a two-dimensional parameterization of the mesh. In order to take polyline edge lengths into account, the coordinates of input points $\mathbf{x}_i \in \mathbb{R}^3$ must be somehow considered. In order to do this, many methods have been devised [GH97]; [Wac75]; [Flo03]. We will focus here on approaches based on distance between points [FR01], mostly because polylines, unlike triangle meshes, do not give much connectivity information, making it difficult to take full consideration of the local shape (for example using angles).

Using the inverse-distance coefficients effectively means that the farther two points are from each other, the smaller the influence is between them, which makes it easier to balance the local equilibrium and satisfy equation (3.1). Thus, we can define the coefficient between any two distinct points i and j by normalizing the inverse-distances as:

$$\lambda_{i,j} \doteq \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^{-1}}{\sum\limits_{k \in \mathcal{N}_i} \|\mathbf{x}_i - \mathbf{x}_k\|^{-1}} \tag{3.6}$$

The main drawback of inverse-distance coefficients is that they perform poorly on sparse and non-uniformly distributed data, which is often encountered in geoscience applications.

Indeed, given a point of index i, all points on a sphere centered on point i will provide the same inverse-distance, without taking the exact point distribution into account. Though they can work for point clouds, they are not always sufficient for polyline parameterization.

3.2.4.3 Our Proposal

In order to overcome the limitations of the inverse-distance coefficients, we propose once again a locally optimal approach. The idea behind it is to locally compute the optimal coefficients, that is, to compute for each vertex the neighbor coefficients that lead to the "best" barycenter. This can be formulated as a constrained optimization problem over the neighborhood of each point i:

$$\begin{aligned} &\text{Minimize } \|\mathbf{x}_i - \sum_{j \in \mathcal{N}_i} \lambda_{i,j} \mathbf{x}_j\|^2 \\ &\text{subject to } \forall j \in \mathcal{N}_i, \lambda_{i,j} \geq 0 \\ &\text{and } \sum_{j \in \mathcal{N}_i} \lambda_{i,j} = 1 \end{aligned} \tag{3.7}$$

We can reformulate this problem as a constrained quadratic optimization problem. The energy to minimize is just the difference between the point coordinates and the barycentric combination of its neighbors. This can be simply refactored using the inner product in $\mathbb{R}^3$ as follows:

$$\begin{aligned} \|\mathbf{x}_i - \sum_{j \in N_i} \lambda_{i,j} \mathbf{x}_j\|^2 &= (\mathbf{x}_i - \sum_{j \in \mathcal{N}_i} \lambda_{i,j} \mathbf{x}_j)^T (\mathbf{x}_i - \sum_{k \in \mathcal{N}_i} \lambda_{i,k} \mathbf{x}_k) \\ &= \mathbf{x}_i^T \mathbf{x}_i - 2 \sum_{j \in \mathcal{N}_i} \lambda_{i,j} \mathbf{x}_i^T \mathbf{x}_j + \sum_{j \in \mathcal{N}_i} \sum_{k \in \mathcal{N}_i} \lambda_{i,j} \lambda_{i,k} \mathbf{x}_j^T \mathbf{x}_k \end{aligned} \tag{3.8}$$

The term $\mathbf{x}_i^T \mathbf{x}_i$ is constant and can therefore be discarded. Since minimizations need to be carried out independently for each vertex i, for the sake of clarity we will drop the subscript i from all equations. The position of the treated vertex will therefore simply be denoted as $\mathbf{x}$, and its neighborhood $\mathcal{N}$. With these adjustments, our problem is equivalent to the following box-constrained quadratic program (QP), with an additional equality constraint:

$$\begin{aligned} &\text{Minimize } \frac{1}{2} \lambda^T Q \lambda + \mathbf{p}^T \lambda \\ &\text{subject to } \forall j \in \mathcal{N}, \lambda_j \geq 0 \\ &\text{and } \sum_{j \in \mathcal{N}} \lambda_j = 1 \end{aligned} \tag{3.9}$$

Where:

- $\lambda \in \mathbb{R}^{|\mathcal{N}|}$ is the column vector containing the unknown optimal coefficient values for vertex i, noting $|\mathcal{N}|$ the cardinality of $\mathcal{N}$;

- Q is a real, symmetric $|\mathcal{N}| \times |\mathcal{N}|$ matrix corresponding to the *Gram matrix*[3] of the coordinate vectors $\mathbf{x}_j, j \in \mathcal{N}$. As such, Q is positive semi-definite, making the optimization problem convex;

- $\mathbf{p} \in \mathbb{R}^{|\mathcal{N}|}$ is the column vector defined by $p_j \doteq -\mathbf{x}_j^T \mathbf{x}_j$.

As a constrained quadratic programming (CQP) problem, our minimization can be solved by a variety of methods. We have chosen to follow the exterior point active-set approach presented in [VL04], called BOXCQP. Active set methods for inequality-constrained optimizations are based on the realization that, according to the standard Karush-Kuhn-Tucker (KKT) conditions, on the optimum a constraint is either satisfied as an equality, or it is inactive (i.e., the corresponding Lagrange multiplier is zero). Consequently, constraints can iteratively be activated and deactivated, based on the sign of the corresponding Lagrange multiplier, until the solution is reached. Once the set of active constraints is known, a simple optimization on the unconstrained unknowns yields the solution. BOXCQP specializes this approach to box-constrained problems, i.e., CQP problems with constraints $a_i \leq v_i \leq b_i$ for all variables i, and it allows updating multiple constraints in a single iteration, sacrificing the monotonicity of the convergence for superior performance.

Following [VL04], we can add Lagrange multipliers μ_j for inequality constraints. An extra multiplier α is also added for the equality constraint, leading to the following Lagrangian for our minimization (3.9):

$$\mathcal{L} \doteq \frac{1}{2}\lambda^T Q \lambda + \mathbf{p}^T \lambda - \mu^T \lambda + \alpha(\lambda^T \mathbf{1} - 1) \tag{3.10}$$

where $\mathbf{1}$ is a vector containing all ones. From this Lagrangian, the following KKT conditions can be deduced:

$$\begin{cases} Q\lambda + \mathbf{p} - \mu + \alpha\mathbf{1} & = 0 \\ \mu_j \lambda_j & = 0 \\ \lambda^T \mathbf{1} - 1 & = 0 \\ \lambda_j, \mu_j & \geq 0 \end{cases} \tag{3.11}$$

At each iteration k, the BOXCQP algorithm divides the constraints into two sets, inactive $A^{(k)}$ and active $B^{(k)}$ constraints. In practice, if we have $\lambda_j < 0$ at iteration $k-1$, we put j in $B^{(k)}$ at the next iteration, and correspondingly if $\lambda_j \geq 0$ at iteration $k-1$, we put $j \in A^{(k)}$ at iteration k. This means that at iteration k we have to set the Lagrange multipliers $\mu_j = 0$ for the unconstrained vertices $j \in A^{(k)}$, and we set $\lambda_j = 0$ for the constrained ones $j \in B^{(k)}$. Accordingly, we split the coefficient vector λ, the multiplier vector μ and the right hand side vector $\mathbf{p}$ into the unconstrained $\lambda_A, \mu_A = \mathbf{0}_A, \mathbf{p}_A$ and constrained $\lambda_B = \mathbf{0}_B, \mu_B, \mathbf{p}_B$ parts ($\mathbf{0}_A$

[3] Recall that the *Gram matrix* G of a set of vectors $\mathbf{y}_i \in \mathbb{C}^m, i = 1, \ldots, n$ is the $n \times n$ Hermitian matrix of inner products defined by $G_{i,j} \doteq \mathbf{y}_i^\dagger \mathbf{y}_j$.

and $\mathbf{0}_B$ are null column vectors of respective size $|A^{(k)}|$ and $|B^{(k)}|$). The KKT conditions (3.11) for iteration k can then be reformulated as the following system of equations:

$$\begin{bmatrix} Q_{AA}^{(k)} & 0_{AB} & \mathbf{1}_A \\ Q_{BA}^{(k)} & -I_B & \mathbf{1}_B \\ \mathbf{1}_A^T & \mathbf{0}_B^T & 0 \end{bmatrix} \begin{bmatrix} \lambda_A \\ \mu_B \\ \alpha \end{bmatrix} = \begin{bmatrix} -\mathbf{p}_A \\ -\mathbf{p}_B \\ 1 \end{bmatrix} \tag{3.12}$$

where:

- $Q_{AA}^{(k)}$ is the $|A^{(k)}| \times |A^{(k)}|$ submatrix of Q containing rows and columns $r, c \in A^{(k)}$;
- $Q_{BA}^{(k)}$ is the $|B^{(k)}| \times |A^{(k)}|$ submatrix containing rows and columns $r \in B^{(k)}, c \in A^{(k)}$;
- $\mathbf{I}_B$ is the $|B^{(k)}| \times |B^{(k)}|$ identity matrix;
- $\mathbf{1}_A, \mathbf{1}_B$ are column vectors, of lengths $|A^{(k)}|$ and $|B^{(k)}|$ respectively, containing all ones;
- 0_{AB} is the null matrix of size $|A^{(k)}| \times |B^{(k)}|$ and $\mathbf{0}_B$ is the column vector of length $|B^{(k)}|$ containing zeros.

It should be noted that, due to the normalization constraint, we can only achieve a system of size $|\mathcal{N}| + 1$ instead of the smaller size $|A^{(k)}|$. However in all conceivable applications, with only a few dozen vertices in a neighborhood, this augmented size is not an issue.

Using this approach, a set of locally optimal coefficients can be computed. Numerical experiments show that these locally optimal coefficients lead to a parameterization of superior quality compared to inverse-distance coefficients, and are relatively cheap to evaluate numerically – in practice, the size of the local system to solve is no more than a few dozen points on real data, and only a handful of BOXCQP iterations are usually required for convergence. Results will be discussed more thoroughly in section 3.4.

3.3 Polylines Interpolation by Gridding and Triangulation

We showed how polylines can be parameterized from 3D space to the plane, through a barycentric approach. Indeed, we proposed a mixed neighborhood interpolation with locally optimal coefficients. Now we will briefly explain how two-dimensional gridding (as presented in chapter 2) can be used to interpolate polylines in the plane on a regular support, which then enables straightforward triangulation.

3.3.1 Gridding Parameterized Polylines

The interpolation process described in the previous section is carried out in parameter space $(u, v) \in \mathcal{D}$. The 3D, real-world coordinates of the surface can then be seen as an immersion

$f : \mathcal{D} \mapsto \mathbb{R}^3$. Thus, equation 2.6 needs to be solved once for every coordinate x, y and z, each time setting the constrained values f_i to the corresponding real-world coordinates of the polyline points x_i, y_i and z_i. Thus, every point in parameter space $\mathcal{D}$ is associated with its three-dimensional coordinate, leading to a dense immersion of the whole surface, with polyline points getting mapped close to their original positions.

3.3.2 Triangulation of a Gridded Surface

Once the parameterized polylines are gridded, they can easily be triangulated. A naive triangulation (e.g. using two triangles for each group of four pixels) can be used, but this can lead to very dense meshes irrespective of the smoothness of the surface, eventually introducing significant storage or visualization problems. Adaptive approaches can be used, for example by only using a high triangle density where local surface curvature is high.

Whatever the triangulation method used, once the triangle mesh is computed in deformed space, its vertices can be transformed into three-dimensional space using the previously computed map, and the reconstruction process is complete.

3.4 Results

3.4.1 Illustration on Synthetic Data

In order to construct an efficient barycentric map for polylines, we proposed an interpolation method for each vertex neighborhood with local estimation of the geometrical radius and the interpolation parameter, combined with a set of locally optimal coefficients. We will now show examples that illustrate how this custom barycentric map performs better than existing methods, that were originally designed with point cloud or mesh parameterization in mind.

The first input dataset we will consider is shown in figure 3.6. It is a sparse set of polylines, representing a typical mushroom-shaped salt dome, commonly found in geosciences. The objective is to flatten the polylines so that they become monovalued.

We will first of all consider the interpolated neighborhood method for this example, in order to illustrate the influence of the choice of coefficients. To this end, figure 3.7 depicts how optimal coefficients lead to a better parameterization. It can be seen that the distribution of points is indeed better preserved when using the optimal coefficients. In this case, the inverse-distance coefficients were moreover unable to unfold all loops, making the use of optimal coefficients mandatory. Gridding and triangulation in the plane are then applied. The reconstructed surface using optimal coefficients and interpolated neighborhoods is reported in figure 3.8.

Let us now introduce a quality metric Δ_{θ_a} that quantifies the average of angle[4] distortions

[4]Absolute value of angles is actually considered as angle distortions can be negative: an average of signed

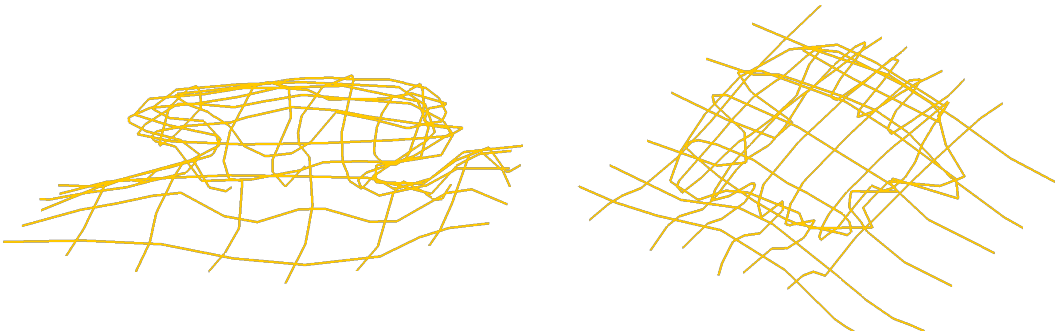

Figure 3.6: The input polylines for this simple example, viewed from the side and in perspective.

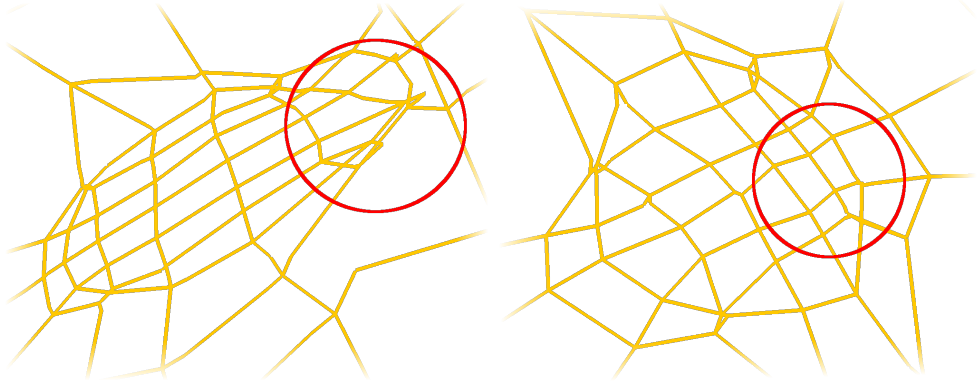

Figure 3.7: Parameterized polylines (in the plane, or equivalently polylines in deformed space and viewed from top with orthogonal projection). Using inverse-distance coefficients (left) and optimal coefficients (right). Optimal coefficients better preserve the distribution of polylines in parameter space, and unfold all loops (highlighted by the red circles in the figure), whereas inverse-distance coefficients cannot cope with local issues and do not unfold all loops.

between 3D polylines and their parameterized 2D counterparts. For length distortions, we note Δ_l the normalized dimensionless edge distortion error used in [YBS04]. Note that in general, we are interested in reducing the gradient of the length distortions, as there will always be some length distortions where the Gaussian curvature does not vanish. In our case, because we fixed the boundaries and we will eventually use a regular triangulation in parameter space (instead of an adaptive one as for example in [Los17]), we can directly consider the length distortions. We reported the quality metric values in table 3.1 for various combinations of neighborhoods and coefficients, applied on the input polylines reported in figure 3.6. These measurements highlight the efficiency of our proposed interpolated neighborhood and optimal set of coefficients. They also show that on average, simple neighborhoods and coefficients work relatively decently. However, they cannot unfold loops and therefore totally prevent the use of discrete interpolation methods. In this particular case the inverse-distance coefficients (when using the interpolated neighborhood) managed to unfold all loops, but this is not always the case. For this reason, optimal coefficients are preferably used in practice.

angles would often be around zero.

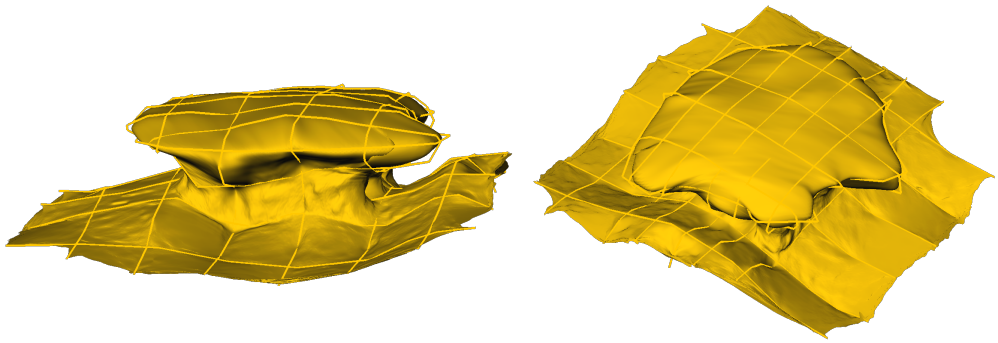

Figure 3.8: Reconstructed surface, fitting input polylines. Fitting is done using $\alpha = 0.02$ and $\beta = 0.2$ with polylines coordinates normalized in $[0, 1]$. The gridding parameters are explained in section 2.3.

3.4.2 Illustration on Real Data

3.4.2.1 First Example: Handling Complex Shapes

Real-world data can be much more challenging than the previous synthetic example in both size and complexity (survey size i.e. gridding resolution, polyline count, target surface curvature, etc). Indeed, in compressive domains, salt formations can reach extremely convoluted shapes. As an example, figure 3.9 illustrates the top surface of a salt body in compressive domain. It shows both the input polylines and the surface reconstructed with our interpolated neighborhood and optimal coefficients.

3.4.2.2 Second Example: Influence of Picking Density and Uniformity

Another top salt horizon was picked using various polyline distributions, and the surface was once again reconstructed by the interpolated neighborhood and optimal coefficients. The results are reported in figure 3.10, and show how robust our method is towards anisotropy, that increases as polylines are picked in a less organized fashion. Quality metric measurements are also reported in table 3.2, noting Δ_{θ_m} and Δ_{θ_M} the minimum and maximum absolute angle distortions. As expected, a dense and organized set of input polylines leads to the best output surface. However, the visualization results also show that a relatively sparse and unorganized set of polylines is sufficient to capture the shape of the target surface. Even on very sparse input data sets, the angle and length distortions still remain relatively low.

3.4.3 Discussion

We believe that our method possesses significant advantages: it is conceptually simple, relatively straightforward to implement, and fast. Namely, numerical tests show that triangle meshes of several million triangles can be reconstructed in under a second on a typical oil

Neighborhood	Coefficients	Δ_{θ_a} (°)	Δ_l (-)	Loops unfolded
T	Inverse-distance	19	0.44	✗
T	Optimal	16	0.42	✗
G (fixed radius)	Inverse-distance	23	0.62	✗
G (fixed radius)	Optimal	24	0.56	✗
G (optimal radius)	Inverse-distance	24	0.56	✗
G (optimal radius)	Optimal	25	0.51	✗
T+G (optimal radius)	Inverse-distance	9	0.41	✓
T+G (optimal radius)	Optimal	8	0.39	✓

Table 3.1: Performance of various neighborhoods and coefficients. The topological neighborhood (T) is more conformal and isometric than the geometrical neighborhood (G), as it is balanced at polyline crossings. The geometrical neighborhood is better at unfolding loops, though it did not manage it on its own in this case. Indeed, only the interpolated neighborhood (T+G) had loops correctly unfolded. Optimal coefficients improve the parameterization if an adequate neighborhood is used. Overall, our method (interpolated neighborhoods and optimal coefficients) works best, reducing both angle and length distortions.

Polylines	Points	Angle distortions			Length distortions
		Δ_{θ_m} (°)	Δ_{θ_a} (°)	Δ_{θ_M} (°)	Δ_l (-)
59	2454	0.05	8	152	0.36
30	661	0.09	11	139	0.42
10	246	0.004	13	160	0.47

Table 3.2: Quality metric measurements for the picking types illustrated in figure 3.10. Angles and lengths are well preserved in average. Moreover, no trend in minimum and maximum distortion values can be observed, another indication of the robustness of our method.

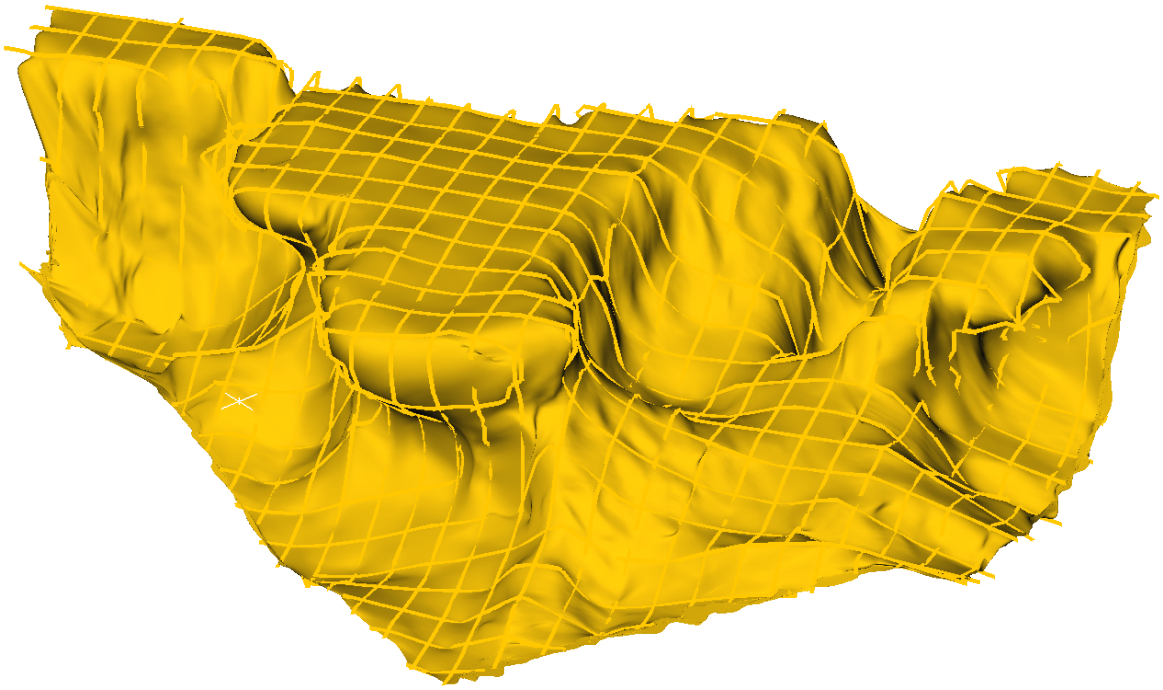

Figure 3.9: Production data for the top of salt in compressive domain. Input polylines and reconstructed surface are shown. Several domes are joined, forming significant overhangs over the salt sheet. Highly curved areas are well resolved, despite the relative sparsity of the picking. The entire reconstruction process (parameterization, gridding and triangulation) took less than a second on a typical oil & gas workstation.

& gas workstation[5]. The speed with which a parameterization can be obtained means that many tests can be done quickly, allowing the iterative tuning of the parameters to achieve the best result. The good performance of the algorithm is due to fact that the method is based on solving a few sparse linear systems for the gridding part and dense, but small, linear systems for the determination of the locally optimal neighborhood coefficients. Parallelization is also straightforward, notably in the construction of the linear system. Gridding and triangulation could also be made more efficient using parallel linear algebra and adaptive meshing libraries, which however we have not done in this work. Finally, we believe that the underlying philosophy of our method, which solves a 3D problem in 2D, has an intrinsically lower time and memory footprint compared to other reconstruction methods that are completely formulated in three dimensions, such as implicit approaches.

[5]See section 1.3.4.2. We used Java linear algebra front-ends towards LAPACK [And+99] for dense matrices and CSPARSE [Dav06] for direct sparse solvers.

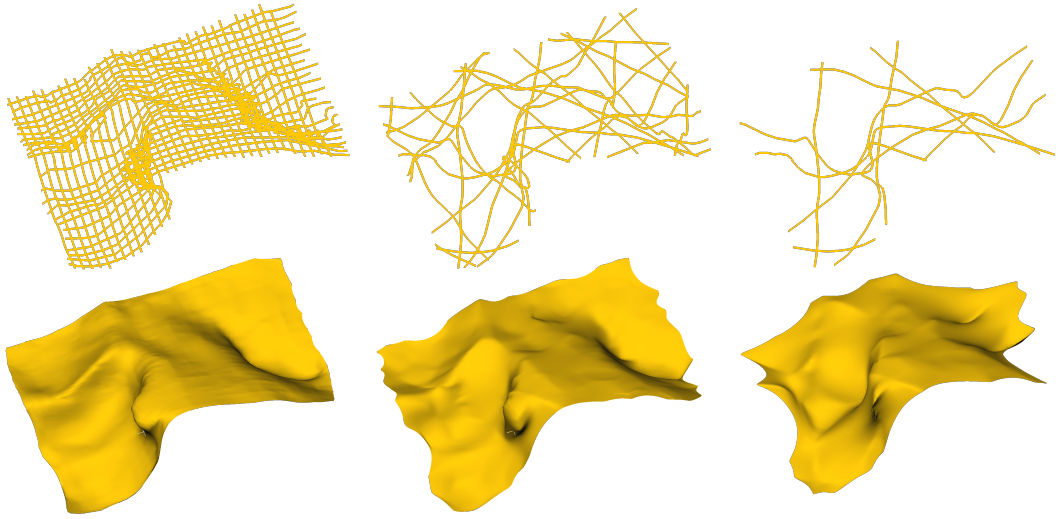

Figure 3.10: Influence of input polyline density and distribution (top row) on the reconstructed surface (bottom row). From left to right, a dense organized input, a sparser unorganized input and a very sparse unorganized input. The middle example shows how similar reconstructed surfaces can be obtained from fewer and unorganized polylines.

3.4.4 Summary

We showed how a barycentric mapping can be developed in order to parameterize 3D polylines representing a surface to the plane. This mapping can handle sparse, non uniformly distributed polylines that were picked in unorganized cross-sections. Once flattened, the polylines describe a monovalued horizon in an abstract, deformed space. At this point, standard gridding can be performed in order to interpolate the polylines. From a regular grid, it is then easy to create a triangle mesh to produce a discrete version of the reconstructed surface. The method presented here, and especially the locally optimal set of barycenter coefficients, is an improvement upon a previous proposal [Bau+18a], and a research paper presenting it is undergoing submission to the journal Computers & Geosciences.

Further advances could be made, for example by developing a globally optimal set of coefficients instead on relying on local minimization problems, that play the role of a surface-wide Laplace-Beltrami operator. Another possible improvement is the use of more recent conformal parameterization algorithm, for example as presented in [SC17], to automatically determine the position of the surface boundary in parameter space, minimizing locally the area distortion and thus further improving the quality of the reconstructed triangulated surface. Moreover, a richer set of inputs could be tackled by relaxing the assumption that all polyline endpoints need to be boundary points for the parameterization, allowing partial polylines as an input: a manual flag on polylines could be used to this end. An automatic detection would be even more preferable, saving time for other picking tasks and lowering the risk of human errors. As for performance, further parallelization of our method could be envisioned, notably in the gridding and triangulation stages.

Conclusion and Perspectives

It is now possible to handle multivalued horizons within a structural interpretation framework. Amongst the possible data structures from the literature, we selected the most appropriate to model multivalued horizons: reverse-faulted horizons are most efficiently represented by patch systems, a natural and effective extension of the monovalued model. Salt domes would not benefit as much from a patch system representation because the sub-vertical parts they often have would require too many patches. Instead, a simple triangle mesh is adequate, especially since seismic attribute are rarely computed on salt dome surfaces – the regular support of the patch system is therefore not needed.

These two models are close enough to existing ones that software refactoring efforts are minimal. They can moreover be reconstructed from sparse interpretation data using our proposed two methods. Namely, reverse-faulted horizons can be gridded in a natural extension of the monovalued gridding, at the cost of a preparation stage whose complexity is kept low by a unified handling of input interpretation data, and a heuristic graph propagation algorithm that partitions the input data into what will eventually become the patches. This approach is straightforward, fast and resource-efficient because it is intrinsically two dimensional, despite the fact that we reconstruct a three dimensional surface.

As for salt domes, they can also be reconstructed from polylines using a two dimensional approach. A polyline parameterization method is however required. Because we once again use a regular planar interpolation (gridding), a distortion-minimizing parameterization is preferred. We proposed a barycentric approach by developing a neighborhood and coefficients that are well suited to polyline parameterization. The simplicity and efficiency of both our barycentric mapping and the gridding method makes this reconstruction almost instantaneous for human interpreters, significantly out-pacing most other triangulation methods (notably implicit approaches) while yielding a reconstructed surface of measurably good quality.

Using our models and reconstruction methods, multivalued horizons can become a first-class feature of any Oil & Gas interpretation software package. We chose models that are identical (triangle mesh) or very close (patch system) to existing data structures, making it easy to address software legacy issues. They can be efficiently populated by automatic reconstruction from interpretation data, and they are by construction well suited to typical horizon processing tasks: the patch system requires almost no change in seismic attribute evaluation code and can be easily converted for display. Triangle meshes modeling salt domes can be directly visualized by rasterization while benefiting from hardware acceleration via GPUs. Both models and reconstruction methods can be (and most often have been) parallelized to benefit from the full power of a typical modern workstation.

This being said, many improvements remain for a better handling of multivalued horizons. The most elegant and striking feature would be a totally unified handling of any multivalued surface, namely a single model for both reverse-faulted and salt dome horizons, that can be reconstructed from any interpretation input, be it polylines or heightmap fragments. One

could for example envision the parameterization of the abstract input graph of a reverse-faulted horizon, by "sliding" the data along the fault surfaces. If such a unified model could not be devised, another area of research would be the improvement of the multivalued gridding pipeline, for example by a more balanced graph propagation algorithm that takes the size of the interpretation data into account. As for polyline parameterization, BOXCQP is perhaps not the most adequate solver for our box-constrained QP problem. Moreover, instead of having a regular triangulation in parameter space, one could use an adaptive triangulation that would not suffer from distortions as much as our naive method does. Beyond the industrial issue of multivalued horizon management, this thesis allowed the benchmarking of standard data structures in an Oil & Gas context, and the development of innovative surface reconstruction method from polylines, under conditions of sparsity and non-uniform distribution not often considered by the literature. This benchmark could be extended to more data types, supporting other models and tests to improve its realism.

Appendix A

Multivalued Horizon Models: a Benchmark

A.1 Overview

A.1.1 Objectives

This appendix will provide a more technical insight into the benchmark used to evaluate new models for multivalued horizons. We will first establish a measurement protocol and present the benchmark software architecture. In a second time, implementation details will be given, notably the data structures used for isochron representations and acceleration structure. Optimization will also be discussed, as well as the definition of the data sets. Results will eventually be reported and commented, especially the influence of various parameters such as the model access pattern (the context of each test), the data type and the data scale. The benchmark's most significant result is the model comparison, that was given in section 1.3.4.2.

A.1.2 Protocol

Recall that for software legacy reasons, this benchmark is implemented in Java on a x86-64 Linux workstation. This means the benchmark code will be interpreted by a Java Virtual Machine (JVM), while some of its bytecode will likely be compiled "just in time" for optimization [Arn+18]. In other words, care has been taken to "warm-up" the JVM so that the run-time optimization noise is reduced. Moreover, all measurements have been taken several times to reduce variance, as the operating system kernel and other user-space programs might have been interfering with the benchmark's allocated CPU time.

Because exact timing and memory usage is likely to change with platform evolutions, we will only present normalized results. In other words, all results are rescaled so that the patch system model always scores 1 in the benchmark. This makes for an easier comparison between models.

In order to fully benefit from our target implementation platform (presented in section 1.3.4.2), we will have to maximize:

- Memory bandwidth. This is obtained via buffered file system access – be it on the local SSDs organized in Redundant Array of Independent Disks (RAID) or on a remote filer – compact and aligned RAM usage, cache-aware memory handling and minimum strain on the Garbage Collector (GC). Most performance benefits will be leveraged by a correct handling of the CPU caches. In our case we benefit from the typically significant (a few dozen megabytes) L3 cache of workstation CPUs, in which sizable parts of our data will mostly reside. High RAM capacities also allow us to consider memory hungry approaches – if there is a benefit to it of course;

- CPU usage. We can achieve this using parallelization via multi-threading and Single Instruction Multiple Data (SIMD) operations and data structures. In our context, multi-threading is often easily achieved using the Java 8 Stream API on containers: because processing tasks are often very demanding (requiring from milliseconds to seconds), there is no need for fine-grain (micro-second scale or less) multi-threading that would typically require a job system, or even a lock-free implementation for minimum contention[1]. As for SIMD operations, we only have to use SIMD-aware data structures and operations, and the JVM will hopefully produce the corresponding x86-64 vector instructions. Typical workstations will benefit from the top-end x86-64 Streaming SIMD Extensions (SSE) or the more recent Advanced Vector Extensions (AVX) [Int]. As an example, our machine even had AVX2 available, meaning 256 bits vector registers could be used. In other words, each CPU core can simultaneously operate on 6 8-bits values per instruction, or 4 32-bits values. At this point memory bandwidth must be heavily optimized though, lest the CPU cores quickly starve for data.

Note that GPGPU approaches are not considered here for deployment and compatibility reasons: all processing will be done on CPUs. A typical workstation is a powerful machine, but it requires some care to be fully used. It also has some specialist hardware design into it, be it several CPU sockets and the corresponding Non Uniform Memory Access (NUMA) issues [Sol15], or very high bandwidth but high latency Quad Channel ECC Registered memory. All this being said, the machine is expected to be pushed to its limits, Oil & Gas applications being typically compute and memory intensive. They are indeed very often given as an example of how well performance can scale with CPU core count and RAM capacity. For more details about the hardware aspects of optimization, comprehensive reviews can be found in the literature [Fog18a]; [Fog18b].

A.1.3 Benchmark Architecture

The benchmark is a console Java program made of around 40 classes organized in a dozen packages (see figure A.1). In order to reduce implementation work, a few abstract interfaces were defined, notably for acceleration structures, isochron representations, models (as

[1] To be more precise, the duration of typical processing tasks far exceeds the standard CPU burst time on modern operating systems, which lies around a few dozens milliseconds. There is therefore no need for particular approaches that minimize process or thread level preemption.

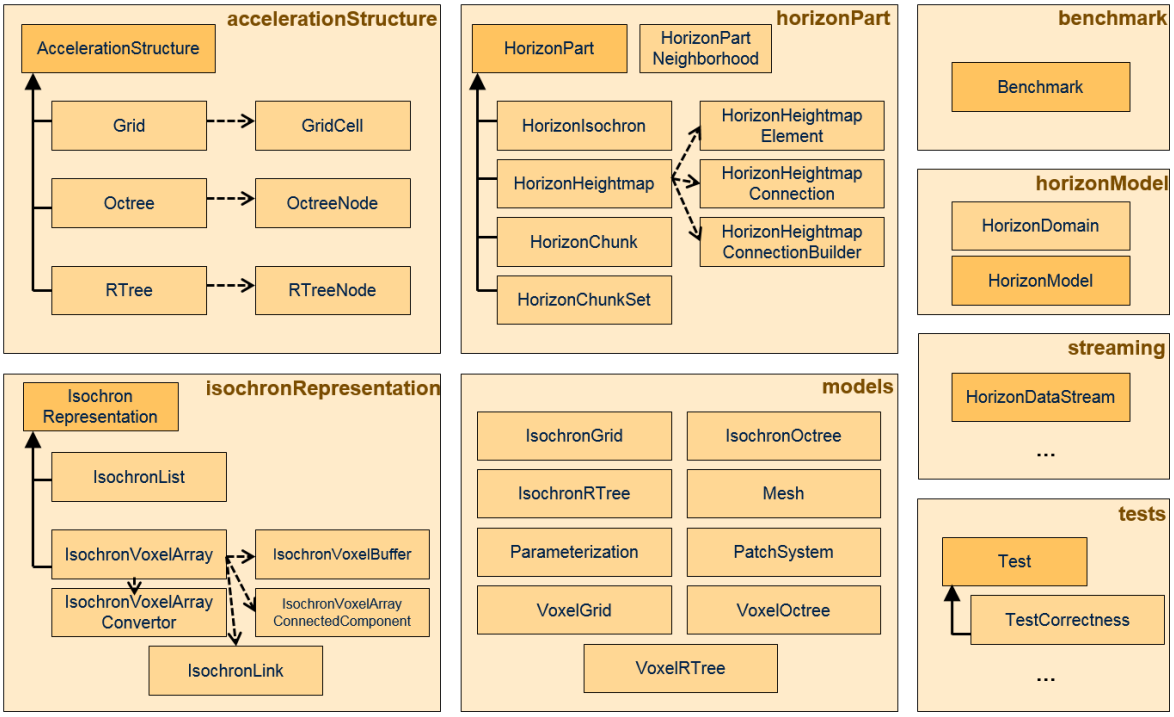

Figure A.1: A Unified Modeling Language (UML) view of the benchmark program. Isochron representations and acceleration structures are all implementations of an abstract interface. Models are then built by combining those, or are custom (patch system). Similarly, tests all relate to an abstract test interface. `HorizonPart` provides an intermediate data structure between data sets and models.

shown by figure A.2), tests (see figure A.3) and data sets. For each interface, various implementations were provided. For example, the `Grid`, `Octree` and `RTree` classes are implementing the `AccelerationStructure` interface. Most models are then a combination of `IsochronRepresentation` within an `AccelerationStructure`. The `PatchSystem` required a custom implementation, being *ad hoc* by construction.

One thing of note is the necessity of an intermediary data structure between models and data sets. Indeed, some tests require the insertion, deletion or retrieval of a local fragment of a horizon surface. The class `HorizonPart` was therefore introduced, and is a local description of a multivalued surface. It is heavily based on a patch system for practical reasons (see figures A.4 and A.5). Great care was taken in order to avoid measurement bias because of the similarity between the `PatchSystem` model and the very close `HorizonPart`[2]. In other words, this similarity is not artificially improving or reducing the patch system performance.

[2]For example, heightmaps could be shared between `PatchSystem` and `HorizonPart` instances but were fully copied.

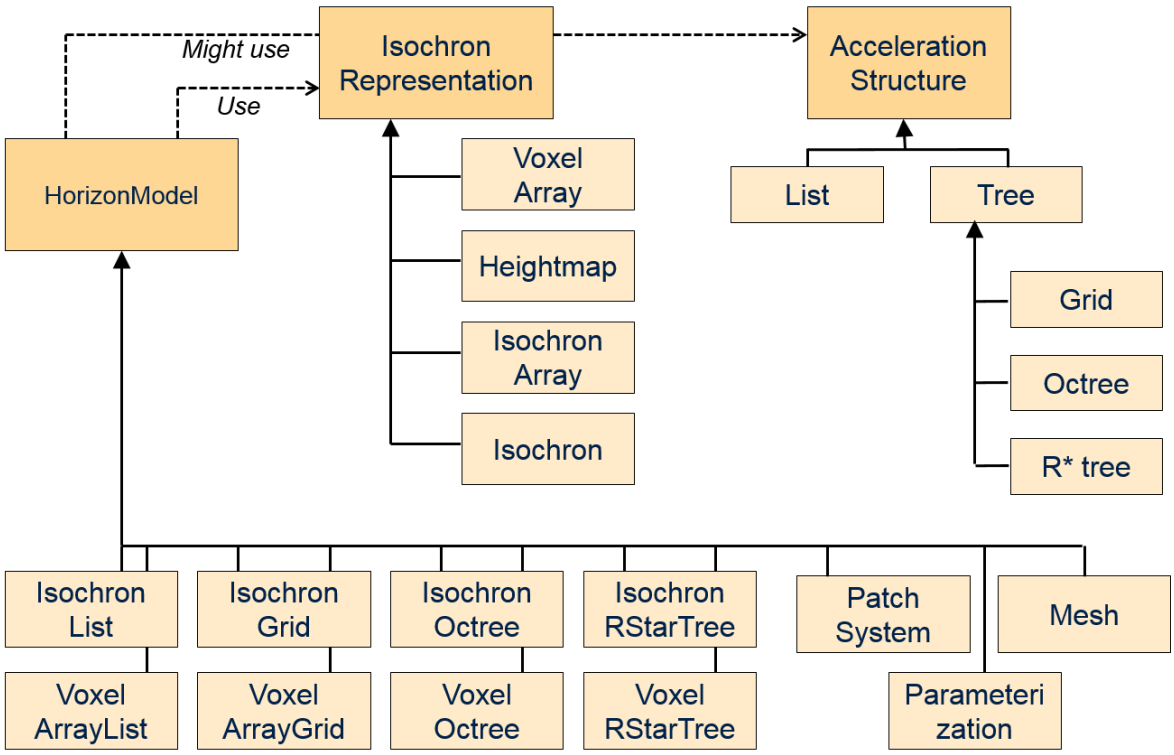

Figure A.2: Models are either a combination of an isochron representation and an acceleration structure, or custom built.

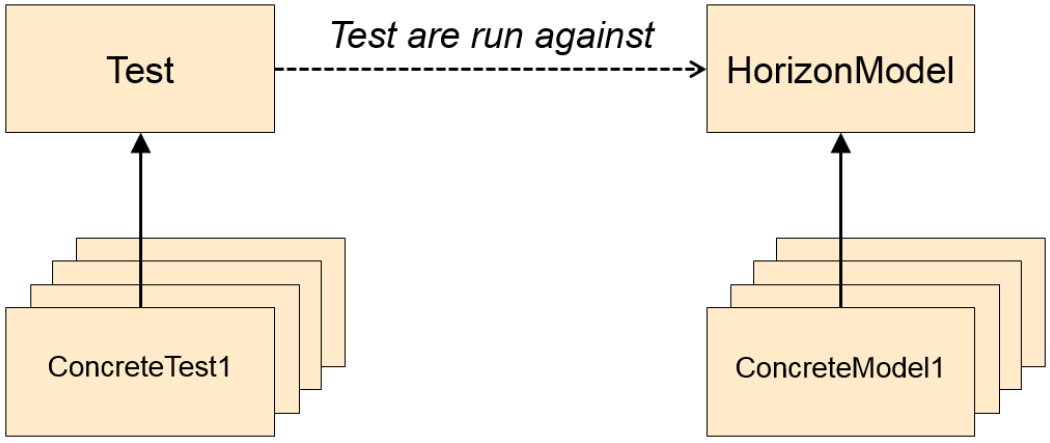

Figure A.3: Tests all implement an abstract `Test` interface, and target each model through the abstract `HorizonModel` interface.

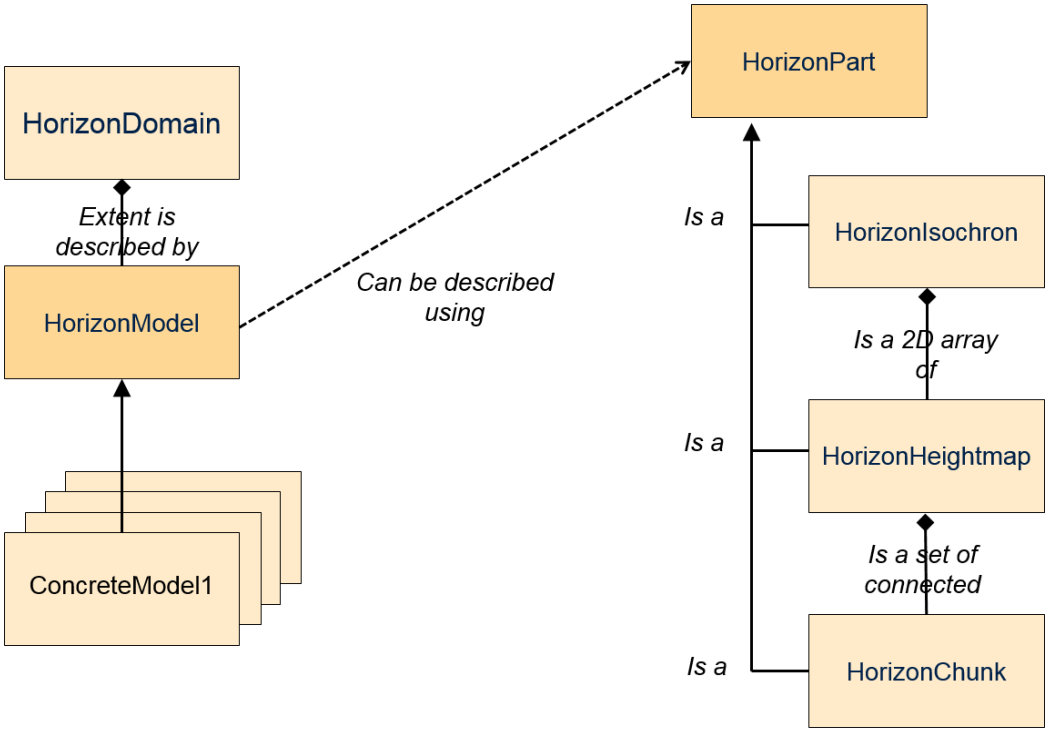

Figure A.4: `HorizonPart` provides a common ground between models and tests. It is built quite similarly to a patch system, because it makes it easier to extract a local manifold representation of a surface using this model.

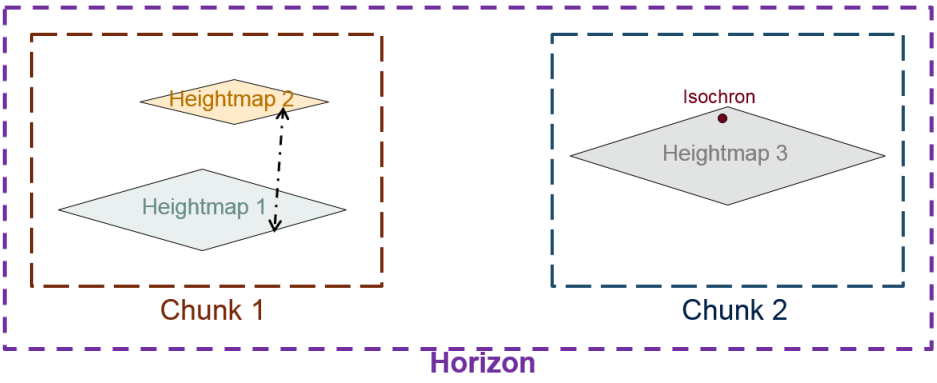

Figure A.5: `HorizonPart` is a hierarchical representation of a surface. Each connected component is modeled by a potentially multivalued `Chunk`, itself composed of several monovalued heightmaps. Isochrons are therefore pixels of those heightmaps.

A.2 Implementation

A.2.1 Isochron Representations

Isochron representations have already been presented, but we will provide here some implementation details that the reader might find interesting. Some design choices were arbitrary, while others were made with our specifications in mind. In order to use isochron representations in the acceleration structures, a base interface `IsochronRepresentation` was required. We then implemented the following isochron representations:

- `PointCloud`. A list of 3D point coordinates, using a 32-bit floating point representation for each coordinate. It is implemented in the Structure Of Array (SOA) fashion, i.e. one list for each axis. We preferred SOA instead of Array Of Structure (AOS) because the former is likely to benefit from packed SIMD instructions that the JVM will eventually produce. Such vector operations will for example be used when rescaling coordinates, or any other large scale modification. While it has the downside of requiring more cache ways for efficient use of the CPU caches, our target x86-64 platform typically sports 8-ways L2, L3 and sometimes even L4 caches that will manage this SOA without problems. Moreover, an AOS approach would lead to heavy use of the GC, leading to prohibitive memory management;

- `Mesh`. An half-edge mesh implementation was used in order to enable efficient traversal [Bot+10]. The downside of using doubly linked list is however that an AOS approach must be used. Because Java does not allow for function inlining or other small class mitigation strategies, this means a significant work for the GC when mesh vertices and edges are added or removed. Memory thrashing is also possible after many modifications on such a mesh, as new object pointers are allocated in ever more distant virtual memory locations, defeating the purpose of having CPU caches. Vertices use a 32-bits floating point number for each coordinate;

- `PatchSystem`. Our *ad hoc* model was implemented in a similar way than the `HorizonPart` class: it is a list of `HorizonChunk`, each being a single connected component. Each is made up of potentially several `Heightmap`, stored in a list. Each heightmap has its corresponding neighbor data structure as described in section 1.3.3.2. We implemented the latter as a neighbor patch index map. In order to increase the spatial locality of reference (pixels close in pixel coordinate distance must be close in memory), the patch heightmap is a 2D array stored using a Z-order curve. This technique is commonly used for texture storage [Mor66], and allows pixel blocks to fall into close cache lines, improving the CPU cache efficiency. The same goes for the neighbor patch index map. The latter can however benefit from an additional processing as for cold storage: it can indeed be saved in compressed form, for example using Run-Length-Encoding (RLE), as it will be made up of mostly non valued pixels in practice [RC67]. As for quantization, heightmaps use a 32-bits floating point number for each pixel, while a smaller 8-bit integer was chosen for the neighbor patch index map. In practice, this

limits the number of patch per chunk to 256, which proved sufficient in both tests and production;

- `Voxel`. As for patch systems, a block storage through Z-order curves is used. This is further improved by bit-packing, by storing $4 \times 4 \times 4 = 64$ voxels into 64-bits integers. Compression schemes could have been deployed, but would have reduced the speed at which voxels are manipulated, and were hence not considered.

A.2.2 Acceleration Structures

Now that isochron representations are detailed, we can focus on the implementation of the selected acceleration structures. Recall each isochron representation implements an abstract `IsochronRepresentation` interface. By creating a generic interface `AccelerationStructure< IsochronRepresentation >` that accepts an implementation of isochron representation, we can implement each acceleration structure, and it will work for each isochron representation – though `PatchSystem` is left on its own. This object-oriented approach significantly reduces the number of required classes, leading to an easier and more elegant implementation.

State-of-the-art acceleration structures are delicate constructs that require some significant effort to implement, tune and debug. To this end, let it be noted that various tests have been developed to check that they each behaved correctly, and each design decision was confirmed by performance measurements. The benchmark provides the following implementations of acceleration structures:

- `Grid`. The `Grid` class is a list of `GridCell< IsochronRepresentation >`. The cells can therefore be specialized for each type of isochron representation, be it a point cloud, mesh vertices or a voxel array. The grid cell was tuned so that it is small enough to avoid having large isochron representations in each cell, while being large enough to significantly reduce the number of discarded isochron representations if a cell is nothing in a query. On a side note, a `GridCell` is also an `IsochronRepresentation`, so that a fixed depth greater than 1 can be used for the `Grid` acceleration structure. In practice, diminishing returns are quickly reached when using depths higher than 2 or 3;

- `Octree`. Being trees, octrees can be naively implemented using pointers to node instances. This typically leads to many allocations and deallocations that significantly stress the GC, and this is not even considering the additional problems on NUMA systems. However, this can be mitigated by having, for each node, an index towards a block of 8 son nodes instead of 8 indices towards one son node each. Despite reducing the strain on the GC, it still uses a lot of memory and some pointers. A more compact representation is the linear hashed (pointer-free) octree, that relies instead on node integer indices and Z-order curves [Gar82]. On the down side, linear octrees have more overhead when modified, and can use more memory when representing sparse data – which is our case, as we only represent surfaces embedded in space, instead of volumes for example.

Once again, there is a compromise between memory footprint reduction and traversal speed. All things considered, we chose a pointer-based octree implementation grouping nodes by blocks of 8;

- `RTree`. Amongst the many possible variants of R-tree available in the literature, we chose the R*-tree variant with an MBV in the form of an Axis-Aligned Bounding Box (AABB). It is a R-tree with an improved topological splitting strategy – a heuristic that reduces node overlap and therefore enhances performances. It has a higher internal maintenance cost however, and is harder to implement. When bulk-loading the tree, i.e. when constructing it in a single insertion, Z-order curves or other packing schemes can be used to produce a balanced tree [LEL97]. In theory, the R*-tree is one of the most data-fitting, hence memory efficient, data structures. In practice, this is significantly mitigated by the use of pointers and their detrimental effects on modern x86-64 platforms whose performance increasingly relies on the correct use of CPU caches [CHL00]. Indeed, recall typical workstations are multi-socket NUMA machines with high core count. This means they can suffer from many problems with pointers: cache coherency and conflict misses, cache thrashing, pointer aliasing and false sharing, or memory fragmentation to name but a few [SHW11].

A.2.3 Data Types

Recall several scales of data will be considered. So to speak, a data set is the sampling of an ideal surface at a given resolution, i.e. scale. This means the data sets, also called data types, must be somehow parametric: given a scale, an intermediary representation must be produced, in the form of a `HorizonPart` instance. Most data types are simple enough not to require any comment. The `MonovaluedFractal` data set is obtained by computing some Perlin value noise, a very common algorithm in procedural texture generation. As for the `MultivaluedDome`, it is a warped half-sphere whose border circle is mapped to a square: the deformation gives the desired multivalued overhang in order to model a salt dome.

A.3 Results

Before commenting various results, it must be noted that given 10 models, each subjected to 10 tests, using 5 data sets at 3 scales, 1500 measurements were conducted. Conducting the full benchmark takes several minutes and produces a dozen result files, that can be formatted using command-line options. The necessity and advantage of an automatic benchmark is made very clear at this point. There is no hope of reporting or discussing all the results here however, therefore we will only consider the influence of the last 3 parameters (test, data set and scale), while section 1.3.4.2 comments the influence of the first parameter, i.e. models. This is enough to draw conclusions as to which models are most suited for representing multivalued horizons, performance wise.

A.3.1 Influence of Context

We will consider the performance of all models in each test, using the `MultivaluedSawTeeth` data set at the largest scale. Were smaller scales used, we might have the data set fully contained in Last Level Cache (LLC), which would bias the memory intensive tests. Anyway by using this protocol, we will see how models perform depending on the context, i.e. the access patterns. Results are reported in figure A.6.

As expected, the point cloud and mesh models perform measurably better when it comes to display format conversion (though triangulating point clouds is costly). They are not the most suited for reading / writing operations, as well as serialization, mostly because it is more difficult to leverage the full benefits of coherent memory access with such explicit models, as compared to the regular voxels or patch systems – for them, looping through pixels / voxels can benefit from caching if done linearly. This comparison shows that patch systems, and to a lesser extent voxels, are adequate models for processing (reading, writing, conversion), while point clouds and especially meshes are more adapted for display.

A.3.2 Influence of Data Type

Let us now observe the `ReadAll` test with each model on all data sets at the largest scale. This tests represent a situation where the entire horizon is read from the model, for example for global processing or in preparation for a conversion or serialization. Results are reported in figure A.7.

A first observation is that the simpler the data set, the more similar the models perform. This means that for smooth monovalued surfaces, there is no need for complex models – hence the current use of heightmaps. When data sets are multivalued, the good processing performance of patch systems and voxels is clear. Moreover, intrinsically 3D objects such as salt domes are difficult for patch systems to model, because they can require a lot of patches where the surface tangent plane is almost vertical. For them, voxels are best for processing, and to a lesser extent triangulated surfaces (though they are better used for display, as shown in the previous section).

A.3.3 Influence of Scale

Finally, we will consider for all models the `MultivaluedSawTeeth` data set, at each scale in the `WriteAll` test. This will provide a measurement of the algorithmic complexity of each model, i.e. its asymptotic performances, as the data to handle gets larger and larger. It must be noted that asymptotic behavior is perturbed for smaller scales due to the presence of a large 20MB last-level cache (LLC). In other words, the smallest data scale (survey of side dimension 10^3 isochrons) fits entirely in the cache, which dramatically speeds-up any processing on it. This explains the big difference in time performance between this data set size and the others, although this is not visible on the relative scale we use (see figure A.8).

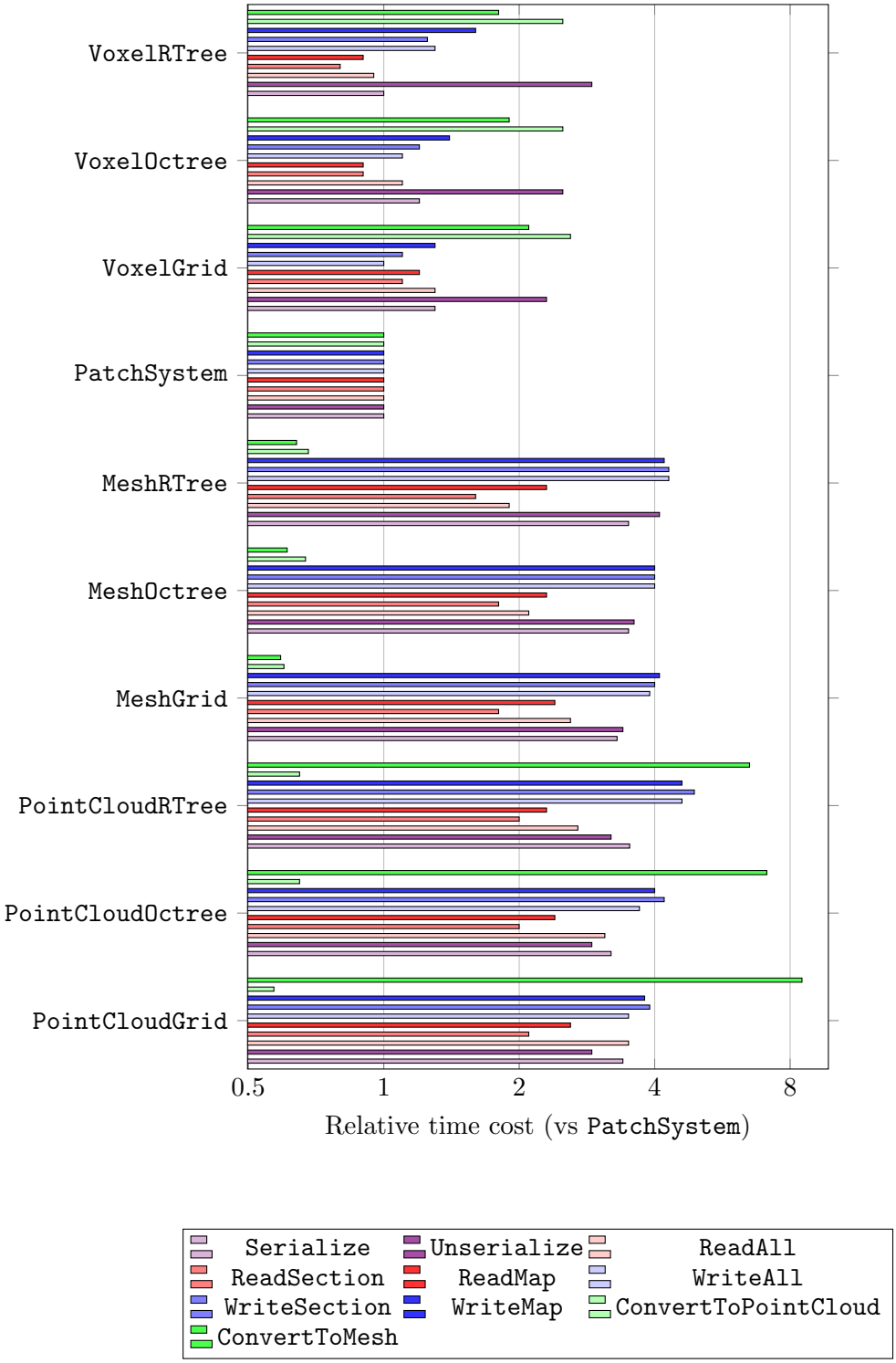

Figure A.6: Performance of models in each tests, with the `MultivaluedSawTeeth` data set at the largest scale.

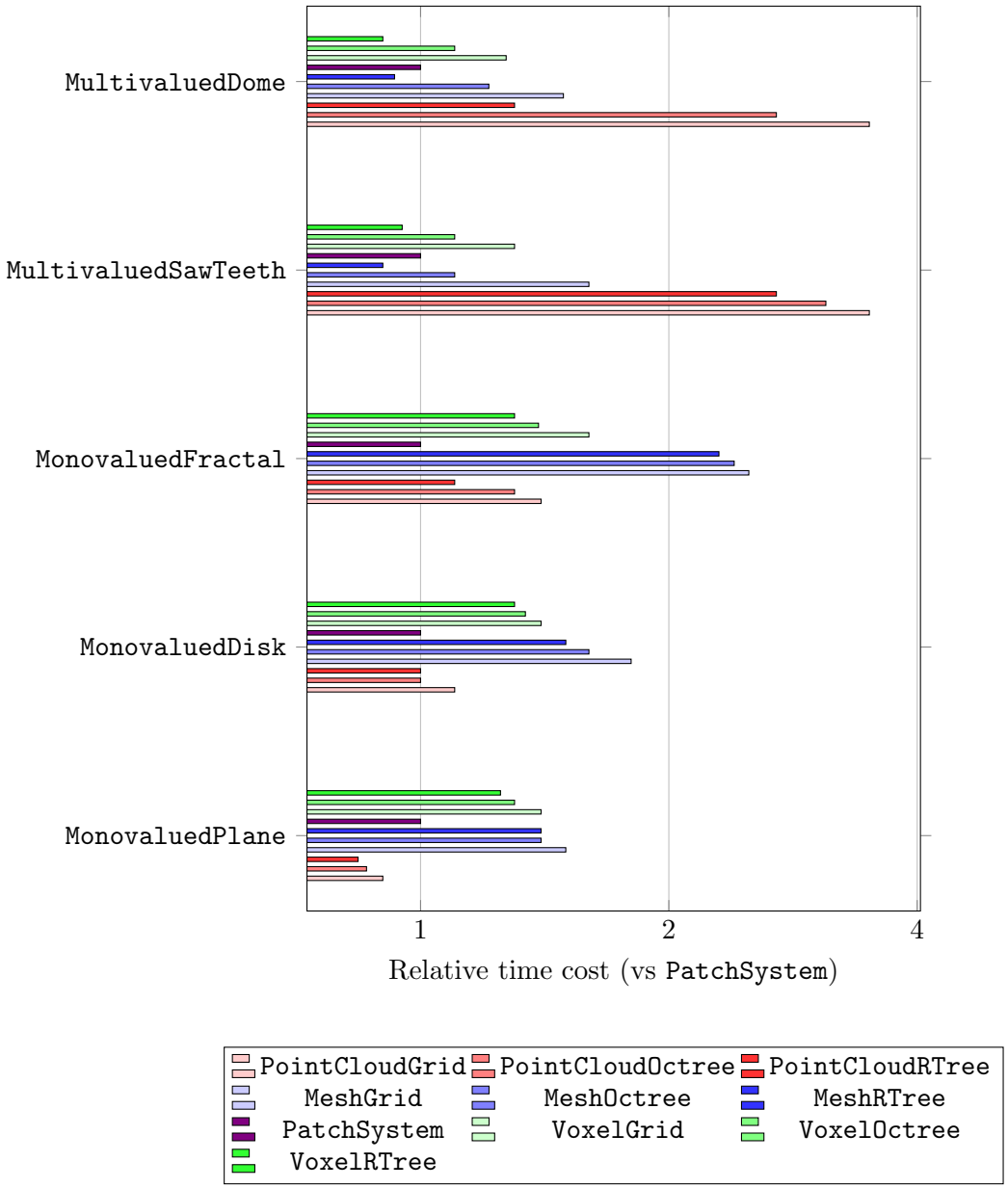

Figure A.7: Performance of models in the `ReadAll` test, on each data set at the largest scale.

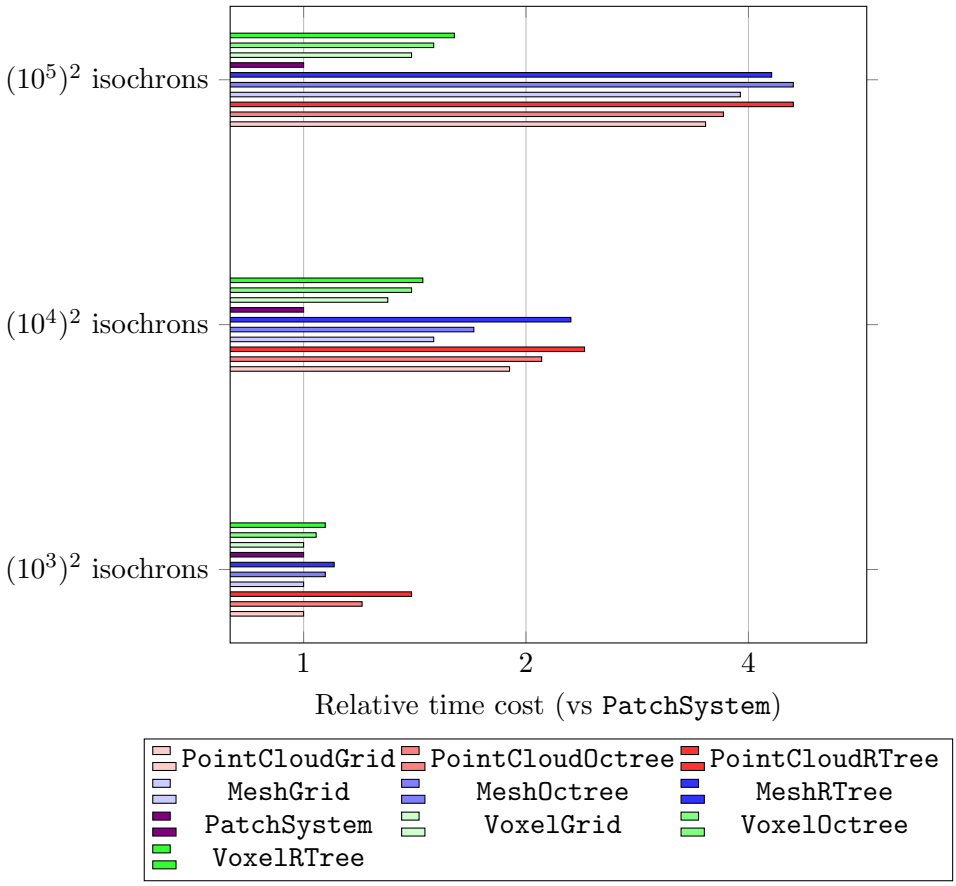

Figure A.8: Performance of models in the `WriteAll` test, on the `MultivaluedSawTeeth` data set at various scale.

On a side note, another disturbance was avoided by using isochron counts that are not big powers of 2, which could have introduced cache trashing on the typically 8-way associative LLC. We used the `WriteAll` test instead of `ReadAll`, because the cache write policies are often the source of performance issues, especially in multi-socket systems such as ours.

As explained, most models perform the same for small data scales, as memory traffic is made easy by having the entire data into the LLC. As survey size gets bigger, the performance difference between models increases, and especially between acceleration structures for the same isochron representation. The maintenance cost of each acceleration structure, i.e. its "overhead", is made more visible as more isochrons have to be considered. This will result in better performance for reads, but has to be payed whenever there is a change in the acceleration structure, as for writes. The patch system performs really well in this situation because it is an intrinsically 2D representation. Triangulated surfaces are also efficient to some extent.

A.3.4 Summary

We have seen the influence of our 4 parameters of study, namely model type, test, data set and scale. We are looking for a model that performs relatively well on average, and can cope with the bigger data sets for future proofing. It was also mentioned that our two main multivalued horizon types, namely reverse-faulted horizons and salt domes, have different needs. The former will be the subject of a lot of processing, mostly seismic attributes computation, while the latter is essentially modeled for display.

With this in mind, the patch system appears to be very suited to the modeling of reverse-faulted horizons. With excellent performance in processing, it only suffers from having the need to be converted to point cloud or mesh for display, though the cost is manageable. This model is adequate because it represents a 3D surface with 2D elements, and a reverse-faulted horizon is indeed such a kind of object, being essentially a 2D surface teared and distorted at some places.

As for salt domes, the patch system has detrimental performance issues with highly vertical surface tangent planes, what geologists would call "sub-vertical" areas. As processing needs are lighter for salt domes, a display oriented model seems more relevant. In other words, the mesh is an excellent model for salt domes.

As discussed in chapter 1, these favorable performance arguments are complemented by other aspects, such as limiting software refactoring and implementation simplicity. Patch systems are indeed a natural extension of the heightmap model, and triangulated surface are standard objects of Oil & Gas software packages, already used to model monovalued horizons, faults, geological bodies and many other objects. We moreover observed that there is no need for excessively complex acceleration structures with these models, avoiding the cost of implementing, tuning and maintaining them. For all these reasons, patch systems and meshes will be used for the multivalued horizons' data model, representing reverse-faulted horizons and salt domes, respectively.

Bibliography

[AJA12] Fatemeh Abbasinejad, Pushkar Joshi, and Nina Amenta. "Surface patches from unorganized space curves". In: *Proceedings of the Twenty-eighth Annual Symposium on Computational Geometry*. Chapel Hill, North Carolina, USA: ACM, 2012, pp. 417–418 (cit. on p. 34).

[And+99] Edward Anderson et al. *LAPACK Users' Guide*. Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999 (cit. on p. 76).

[Arn+18] Matthew Arnold et al. "Architecture and policy for adaptive optimization in virtual machines". In: *IBM Research Report* (Aug. 2018), pp. 1–66 (cit. on p. 81).

[Azr+13] Suhaibah Azri et al. *Review of Spatial Indexing Techniques for Large Urban Data Management*. Jan. 2013 (cit. on p. 20).

[Bat+12] Bryson Bates et al. *Change and Water - IPCC Technical Paper VI*. Tech. rep. Intergovernmental Panel on Climate Change, 2012 (cit. on p. 7).

[Bau+18a] Joseph Baudrillard et al. "Parameterization of polylines in unorganized cross-sections and open surface reconstruction". In: *9th International Conference on Curves and Surfaces*. Arcachon, France, 2018, pp. 24–24 (cit. on pp. 62, 67, 77).

[Bau+18b] Joseph Baudrillard et al. "Reconstruction of piecewise-explicit surfaces from three-dimensional polylines". In: *International Conference on Geographical Information Systems Theory, Applications and Management (Doctoral Consortium)*. Funchal, Madeira, Portugal, 2018, pp. 8–18 (cit. on p. 59).

[BBS08] Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. "ILoveSketch: as-natural-as-possible sketching system for creating 3D curve models". In: *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*. Monterey, CA, USA: ACM, 2008, pp. 151–160 (cit. on p. 34).

[Bec+90] Norbert Beckmann et al. "The r*-tree:an efficient and robust access method forpoints and rectangles". In: *ACM Transactions on Graphics* 19.2 (1990), pp. 322–331 (cit. on p. 23).

[Ber+17] Matthew Berger et al. "A survey of surface reconstruction from point clouds". In: *Computer Graphics Forum* 36.1 (2017), pp. 301–329 (cit. on pp. 17, 34).

[Bes+12] Mikhail Bessmeltsev et al. "Design-driven quadrangulation of closed 3D curves". In: *ACM Transactions on Graphics* 31.6 (2012), 178:1–178:11 (cit. on p. 34).

[BG93] Jean-Daniel Boissonnat and Bernhard Geiger. "Three-dimensional reconstruction of complex shapes based on the Delaunay triangulation". In: *Biomedical Image Processing and Biomedical Visualization*. Vol. 1905. International Society for Optics and Photonics. 1993, pp. 964–976 (cit. on p. 34).

[Bia69] Theodore Bially. "Space-filling curves: their generation and their application to bandwidth reduction". In: *IEEE Transactions on Information Theory* (1969), pp. 658–664 (cit. on p. 22).

[BM07] Jean-Daniel Boissonnat and Pooran Memari. "Shape reconstruction from unorganized cross-sections". In: *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*. Barcelona, Spain: Eurographics Association, 2007, pp. 89–98 (cit. on p. 34).

[Boo89] Fred Bookstein. "Principal warps: thin-plate splines and the decomposition of deformations". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11.6 (1989), pp. 567–585 (cit. on p. 34).

[Bot+10] Mario Botsch et al. *Polygon Mesh Processing*. AK Peters, 2010 (cit. on pp. 17, 86).

[BPT07] Rick Beatson, Michael James David Powell, and Alex Tan. "Fast evaluation of polyharmonic splines in three dimensions". In: *IMA Journal of Numerical Analysis* 27.3 (2007), pp. 427–450 (cit. on p. 34).

[Bre65] Jack E. Bresenham. "Algorithm for computer control of a digital plotter". In: *IBM Systems Journal* 4.1 (1965), pp. 25–30 (cit. on pp. 37, 46).

[Bri74] Ian C. Briggs. "Machine contouring using minimum curvature". In: *Geophysics* 39 (1974), pp. 39–48 (cit. on p. 35).

[Bul+13] Aydin Buluc et al. "Recent advances in graph partitioning". In: *Lecture Notes in Computer Science* 9220 (2013), pp. 1–37 (cit. on p. 44).

[BVG11] Amit Bermano, Amir Vaxman, and Craig Gotsman. "Online reconstruction of 3D objects from arbitrary cross-sections". In: *ACM Transactions on Graphics* 30.5 (2011), 113:1–113:11 (cit. on p. 34).

[Cau+09] Guillaume Caumon et al. "Surface-based 3D modeling of geological structures". In: *Mathematical Geosciences* 41.8 (2009), pp. 927–945 (cit. on p. 12).

[CC17] Pauline Collon and Guillaume Caumon. "3D geomodelling in structurally complex areas: implicit vs. explicit representations". In: *79th EAGE Conference Exhibition*. Paris, 2017, pp. 1–5 (cit. on p. 14).

[CCC18] Nicolas Clausolles, Pauline Collon, and Guillaume Caumon. "A workflow for 3D stochastic modeling of salt from seismic images". In: *80th EAGE Conference and Exhibition*. Copenhagen, 2018, pp. 1–6 (cit. on p. 14).

[CHL00] Trishul M. Chilimbi, Mark D. Hill, and James R. Larus. "Making pointer-based data structures cache conscious". In: *Computer* 33.12 (2000), pp. 67–74 (cit. on p. 88).

[Cli] *Advancing the Science of Climate Change : America's Climate Choices*. Washington, D.C: National Academies Press, 2010 (cit. on p. 7).

[Cra15] Keenan Crane. *Discrete Differential Geometry: An Applied Introduction*. 2015 (cit. on p. 63).

[Dal15] Samir Dalvi. *Fundamentals of Oil and Gas Industry for Beginners*. Chennai: Notion Press, 2015 (cit. on p. 4).

[Dan80] Erik Danielsson. "Euclidian distance mapping". In: *Computer Graphics and Image Processing* 14 (1980), pp. 227–248 (cit. on pp. 51, 54).

[Dav06] Timothy A Davis. *Direct Methods for Sparse Linear Systems*. Vol. 2. Siam, 2006 (cit. on p. 76).

[Dou+10] Bertrand Douillard et al. "Hybrid elevation maps: 3D surface models for segmentation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2010), pp. 1532–1538 (cit. on p. 18).

[EKS83] Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. "On the shape of a set of points in the plane". In: *IEEE Transactions on Information Theory* 29 (1983), pp. 551–559 (cit. on p. 49).

[Eme05] Xavier Emery. "Simple and ordinary multigaussian kriging for estimating recoverable reserves". In: *Mathematical Geology* 37.3 (2005), pp. 295–319 (cit. on p. 34).

[FB74] Raphael Finkel and Jon Louis Bentley. "Quad trees: a data structure for retrieval on composite keys". In: *Acta Informatica* 4.1 (1974), pp. 1–9 (cit. on p. 22).

[FH54] Michael Floater and Kai Hormann. "Surface parameterization: a tutorial and survey". In: *Advances in Multiresolution for Geometric Modeling* (2054), pp. 157–186 (cit. on pp. 17, 63).

[Flo03] Michael Floater. "Mean value coordinates". In: *Computer-Aided Geometric Design* 20.1 (2003), pp. 19–27 (cit. on pp. 64, 68).

[Fog18a] Agner Fog. *Optimizing Subroutines in Assembly Language: An Optimization Guide for x86 Platforms*. 2018 (cit. on p. 82).

[Fog18b] Agner Fog. *The Microarchitecture of Intel, AMD and VIA CPUs: An Optimization Guide for Assembly Programmers and Compiler Makers*. 2018 (cit. on p. 82).

[FR01] Michael Floater and Martin Reimers. "Meshless parameterization and surface reconstruction". In: *Computer-Aided Geometric Design* 18.2 (2001), pp. 77–92 (cit. on pp. 34, 62, 64, 66, 68).

[FTR86] Alain Fournier and William T. Reeves. "A simple model of ocean waves". In: *ACM Siggraph Computer Graphics* (1986), pp. 75–84 (cit. on p. 18).

[Fuc+80] Henry Fuchs et al. "On visible surface generation by a priori tree structures". In: *SIGGRAPH '80 Conference Proceedings* (1980), pp. 124–133 (cit. on p. 22).

[Gar82] Irene Gargantini. "An effective way to represent quadtrees". In: *ACM Communications on Graphics* 25.12 (1982), pp. 905–910 (cit. on p. 87).

[GG98] Volker Gaede and Oliver Günther. "Multidimensional access methods". In: *ACM Computing Surveys* 30.2 (1998), pp. 170–231 (cit. on p. 20).

[GH97] Günther Greiner and Kai Hormann. "Interpolating and approximating scattered 3D-data with hierarchical tensor product b-splines". In: *Surface Fitting and Multiresolution Methods*. Vanderbilt University Press, 1997, pp. 163–172 (cit. on p. 68).

[GM03] Manuel Gamito and Forest Kenton Musgrave. "Procedural Landscapes with Overhangs". In: (2003), pp. 1–10 (cit. on p. 18).

[Gre04] George J. Grevera. *Distance Transform Algorithms and their Implementation and Evaluation*. Tech. rep. Saint Joseph's University, 2004 (cit. on p. 51).

[Gut84] Antonin Guttman. "R-trees: a dynamic index structure for spatial searching". In: *SIGMOD Rec.* 14.2 (1984), pp. 47–57 (cit. on p. 23).

[HLS07] Kai Hormann, Bruno Levy, and Alla Sheffer. *Mesh Parameterization: Theory and Practice*. Siggraph Course Notes, 2007 (cit. on pp. 17, 63–65).

[Hug+13] John F. Hughes et al. *Computer Graphics: Principles and Practice*. Third. Boston, MA, USA: Addison-Wesley Professional, 2013, p. 1264 (cit. on p. 19).

[Iji+14] Takashi Ijiri et al. "Flower modeling via x-ray computed tomography". In: *ACM Transactions on Graphics* 33.4 (2014), 48:1–48:10 (cit. on p. 34).

[Int] *Intel® 64 and IA-32 Architectures Software Developer's Manual*. 2015 (cit. on p. 82).

[IS11] Martin Isenburg and Jonathan Richard Shewchuk. *Streaming Connected Component Computation for Trillion Voxel Images*. 2011 (cit. on p. 20).

[JW16] Arnulf Jäger-Waldau. *PV Status Report 2016*. 2016 (cit. on p. 8).

[Kes+82] Noomane Keskes et al. "Image analysis techniques for seismic data". In: *SEG Technical Program Expanded Abstract* 1.1 (1982), pp. 221–222 (cit. on p. 39).

[KF94] Ibrahim Kamel and Christos Faloutsos. "Hilbert r-tree: an improved r-tree using fractals". In: *Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 500–509 (cit. on p. 23).

[Knu97] Donald Knuth. *The Art of Computer Programming*. Reading, Mass: Addison-Wesley, 1997 (cit. on p. 22).

[KS86] David G Kirkpatrick and Raimund Seidel. "The ultimate planar convex hull algorithm". In: *SIAM Journal on Computing* 15 (1986), pp. 287–299 (cit. on p. 49).

[LDD14] Ricardo Uribe Lobello, Florent Dupont, and Florence Denis. "Out-of-core adaptive iso-surface extraction from binary volume data". In: *Graphical Models* 76.6 (2014), pp. 593–608 (cit. on p. 14).

[LEL97] Scott T. Leutenegger, Jeffrey M. Edgington, and Mario A. Lopez. *STR: A Simple and Efficient Algorithm for R-tree Packing*. Tech. rep. 1997 (cit. on p. 88).

[LH14] Jin Li and Andrew D. Heap. "Spatial Interpolation Methods Applied in the Environmental Sciences". In: *Environmental Modeling & Software* 53.C (Mar. 2014), pp. 173–189 (cit. on p. 34).

[Los17] Adrien Loseille. "Unstructured mesh generation and adaptation". In: *Handbook of Numerical Analysis*. Vol. 18. Elsevier, 2017, pp. 263–302 (cit. on p. 73).

[LPG12] Yang Liu, Balakrishnan Prabhakaran, and Xiaohu Guo. "Point-based manifold harmonics". In: *IEEE Transactions on Visualization and Computer Graphics* 18.10 (2012), pp. 1693–1703 (cit. on p. 34).

[LW85] Marc Levoy and Turner Whitted. *The Use of Points as a Display Primitive*. Tech. rep. University of North Carolina at Chapel Hill, Computer Science Department, 1985 (cit. on p. 17).

[Mag+15] Adrien Maglo et al. "3D mesh compression: survey, comparisons, and emerging trends". In: *ACM Computation Survey* 47.3 (2015), 44:1–44:41 (cit. on p. 17).

[Mea80] Donald Meagher. *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer (Technical Report)*. Oct. 1980 (cit. on p. 22).

[MK03] Jussi Myllymaki and James H. Kaufman. "DynaMark: a benchmark for dynamic spatial indexing". In: 2003 (cit. on p. 25).

[MM06] Lukasz A. Machowski and Tshilidzi Marwala. "Using images to create a hierarchical grid spatial index". In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. Taiwan, 2006, pp. 1974–1979 (cit. on p. 22).

[Mor66] Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. Tech. rep. 1966 (cit. on p. 86).

[Nat+13] Mattia Natali et al. "Modeling terrains and subsurface geology". In: *EuroGraphics 2013 State of the Art Reports (STARs)*. 2013, pp. 155–173 (cit. on p. 12).

[Nov97] Robert Novelline. *Squire's Fundamentals of Radiology*. Cambridge, Mass: Harvard University Press, 1997 (cit. on p. 19).

[OSdH] Beng Chin Ooi, Ron Sacks-davis, and Jiawei Han. *Indexing in Spatial Databases (Technical Report)* (cit. on p. 22).

[Pan+16] Neelabh Pant et al. *Performance Comparison of Spatial Indexing Structures for Different Query Types (Technical Report)*. June 2016 (cit. on p. 22).

[Pey+09] Adrien Peytavie et al. "Arches: a framework for modeling complex terrains". In: *Computer Graphics Forum* 28 (2009), pp. 457–467 (cit. on p. 18).

[PKT04] Georgios Passalis, Ioannis A. Kakadiaris, and Theoharis Theoharis. "Efficient hardware voxelization". In: *Proceedings of Computer Graphics International* (2004), pp. 374–377 (cit. on p. 20).

[Pre07] William Press. *Numerical Recipes : the Art of Scientific Computing*. Cambridge, UK New York: Cambridge University Press, 2007 (cit. on p. 34).

[Pug90] William Pugh. "Skip lists: a probabilistic alternative to balanced trees". In: *Communications of the ACM* 33.6 (1990), pp. 668–676 (cit. on p. 22).

[RC67] Adam Robinson and Colin Cherry. "Results of a prototype television bandwidth compression scheme". In: *Proceedings of the IEEE* 55.3 (1967), pp. 356–364 (cit. on p. 86).

[Ren] *Deploying Renewables 2011: Best and Future Policy Practice*. Paris: OECD/IEA, 2011 (cit. on p. 8).

[RL00] Szymon Rusinkiewicz and Marc Levoy. "QSplat: A multiresolution point rendering system for large meshes". In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 343–352 (cit. on p. 17).

[RSDB11] Suprio Ray, Bogdan Simion, and Angela Demke Brown. "Jackpine: a benchmark to evaluate spatial database performance". In: *Proceedings of the IEEE 27th International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1139–1150 (cit. on p. 25).

[RSV01] Philippe Rigaux, Michel Scholl, and Agnes Voisard. 2001 (cit. on p. 22).

[Rus98] John C. Russ. *The Image Processing Handbook*. CRC Press, 1998 (cit. on p. 50).

[Sar+08] Maruto Masserie Sardadi et al. "Choosing R-tree or Quadtree Spatial Data Indexing in One Oracle Spatial Database System to Make Faster Showing Geographical Map in Mobile Geographical Information System Technology". In: *International Journal of Science, Engineering and Information Technology* 2.10 (2008) (cit. on p. 25).

[SC17] Rohan Sawhney and Keenan Crane. "Boundary first flattening". In: *ACM Transactions on Graphics* 37.1 (2017), 5:1–5:14 (cit. on pp. 63, 77).

[Sch13] Harold Schobert. *Chemistry of Fossil Fuels and Biofuels*. Cambridge England New York: Cambridge University Press, 2013 (cit. on p. 5).

[SD69] Robert Schumacher and Air Force Human Resources Laboratory. Training Research Division. *Study for applying computer-generated images to visual simulation*. Air Force Human Resources Laboratory, Air Force Systems Command, 1969 (cit. on p. 22).

[She68] Donald Shepard. "A two-dimensional interpolation function for irregularly-spaced data". In: *Proceedings of the 23rd ACM National Conference*. New York, NY, USA: ACM, 1968, pp. 517–524 (cit. on p. 34).

[SHG06] Arkadiusz Sitek, Ron Huesman, and Grant Gullberg. "Tomographic reconstruction using an adaptive tetrahedral mesh defined by a point cloud". In: 25 (2006), pp. 1172–9 (cit. on p. 17).

[SHW11] Daniel J. Sorin, Mark D. Hill, and David A. Wood. *A Primer on Memory Consistency and Cache Coherence*. 1st. Morgan & Claypool Publishers, 2011 (cit. on p. 88).

[Sil06] Martin Silberberg. *Chemistry : the Molecular Nature of Matter and Change*. Boston: McGraw-Hill, 2006 (cit. on p. 4).

[Sme+09] Ruben Smelik et al. "A survey of procedural methods for terrain modeling". In: 2009, pp. 1–10 (cit. on p. 18).

[Sol+17] Amir Arsalan Soltani et al. "Synthesizing 3D shapes via modeling multi-view depth maps and silhouettes with deep generative networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 2511–2519 (cit. on p. 17).

[Sol15] Yan Solihin. *Fundamentals of Parallel Multicore Architecture*. 1st. Chapman & Hall/CRC, 2015 (cit. on p. 82).

[SRF87] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. "The r+-tree: a dynamic index for multi-dimensional objects". In: *Proceedings of the 20th International Conference on Very Large Data Bases*. 1987, pp. 507–518 (cit. on p. 23).

[SS14] Bardia Sadri and Karan Singh. "Flow-complex-based shape reconstruction from 3D curves". In: *ACM Transaction on Graphics* 33.2 (2014), 20:1–20:15 (cit. on p. 34).

[STL14] Artur Lira dos Santos, Veronica Teichrieb, and Jorge Lindoso. "Review and comparative study of ray traversal algorithms on a modern GPU architecture". In: *Proceedings of the World Society for Computer Graphics*. 2014, pp. 1–10 (cit. on p. 20).

[SW90] W. H. F. Smith and P. Wessel. "Gridding with continuous curvature splines in tension". In: *Geophysics* 55 (1990), pp. 293–305 (cit. on p. 35).

[TH16] Eran Treister and Eldad Haber. "A fast marching algorithm for the factored eikonal equation". In: *Journal of Computational Physics* 324.C (2016), pp. 210–225 (cit. on pp. 51, 54).

[TN87] William C. Thibault and Bruce F. Naylor. "Set operations on polyhedra using binary space partitioning trees". In: *SIGGRAPH Computer Graphics* 21.4 (1987), pp. 153–162 (cit. on p. 22).

[TPB06] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. "Multi-level surface maps for outdoor terrain mapping and loop closing". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2006), pp. 2276–2282 (cit. on p. 18).

[TSr05] Niels Thrane, Lars Ole Simonsen, and Advisor Peter Ørbæk. *A Comparison of Acceleration Structures for GPU Assisted Ray Tracing*. Tech. rep. 2005 (cit. on p. 20).

[Tut60] William Thomas Tutte. "How to draw a graph". In: *Proceedings of the London Mathematical Society*. 1960, pp. 743–767 (cit. on pp. 64, 68).

[Uhl91] Jeffrey K. Uhlmann. "Satisfying general proximity/similarity queries with metric trees". In: *Information Processing Letters* 40 (1991), pp. 175–179 (cit. on p. 23).

[VL04] Costas Voglis and Isaac Lagaris. "BOXCQP: an algorithm for bound constrained convex quadratic problems". In: *First International Conference "From Scientific Computing to Computational Engineering"*. Athens, Greece, 2004, pp. 1–8 (cit. on p. 70).

[VRS11] Tamás Várady, Alyn Rockwood, and Péter Salvi. "Transfinite surface interpolation over irregular n-sided domains". In: *Computer-Aided Geometric Design* 43.11 (2011), pp. 1330–1340 (cit. on p. 34).

[Wac75] E. L. Wachspress. *A Rational Finite Element Basis*. New York: Academic Press, 1975 (cit. on p. 68).

[Wal14] Eric Walter. *Numerical Methods and Optimization, a Consumer Guide*. Springer, 2014 (cit. on p. 37).

[WFH18] Xinming Wu, Sergey Fomel, and Michael Hudec. "Fast salt boundary interpretation with optimal path picking". In: *Geophysics* 83.3 (2018), pp. 045–053 (cit. on p. 14).

[YBS04] Shin Yoshizawa, Alexander Belyaev, and Hans-Peter Seidel. "A fast and simple stretch-minimizing mesh parameterization". In: *Proceedings of the Shape Modeling International 2004*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 200–208 (cit. on p. 73).

[Yia93] Peter N. Yianilos. "Data structures and algorithms for nearest neighbor search in general metric spaces". In: *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*. Austin, Texas, USA: Society for Industrial and Applied Mathematics, 1993, pp. 311–321 (cit. on p. 23).

[ZD17] X. Zhang and Z. Du. "Spatial indexing". In: *The Geographic Information Science and Technology Body of Knowledge (4th Quarter 2017 Edition)* (2017) (cit. on p. 20).

[Zim+98] Dean A. Zimmerman et al. "A comparison of seven geostatistically based inverse approaches to estimate transmissivities for modeling advective transport by groundwater flow". In: *Water Resources Research*. Vol. 34. 6. 1998, pp. 1373–1413 (cit. on p. 34).

[ZJC13] Ming Zou, Tao Ju, and Nathan Carr. "An algorithm for triangulating multiple 3D polygons". In: *Proceedings of the Eleventh Eurographics/ACMSIGGRAPH Symposium on Geometry Processing*. Genova, Italy: Eurographics Association, 2013, pp. 157–166 (cit. on p. 34).

[Zou+15] Ming Zou et al. "Topology-constrained surface reconstruction from cross-sections". In: *ACM Transactions on Graphics* 34.4 (2015), 128:1–128:10 (cit. on p. 34).

[ZZN14] Tilmann Zäschke, Christoph Zimmerli, and Moira C. Norrie. "The ph-tree: a space-efficient storage structure and multi-dimensional index". In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. Snowbird, Utah, USA: ACM, 2014, pp. 397–408 (cit. on p. 22).

Résumé — Les horizons sont des éléments indispensables à la création d'un modèle structural pour l'exploration d'hydrocarbures. Ces horizons peuvent avoir des formes complexes, les rendant difficiles voir impossibles à manipuler par les logiciels d'exploration. La gestion de ces horizons "multivalués" dans l'interprétation structurale passe par deux axes de travail. Il faut premièrement trouver un modèle pour les représenter. Un état de l'art des modèles de données et un programme de comparaison automatique montrent ainsi comment les systèmes de patch et les surfaces triangulées sont les plus adaptés pour représenter les horizons avec failles inverses et les dômes de sel, respectivement. Dans un second temps, il faut pouvoir reconstruire les surfaces de ces horizons multivalués à partir de données d'interprétation, qui sont à la fois peu denses et non-uniformément distribuées. Nous proposons deux algorithmes de reconstruction, un pour chaque type d'horizon. Ces méthodes sont basées sur les approches classiques en géoscience, et forment donc une extension naturelle de l'état de l'art. Avec ces modèles et leurs méthodes de reconstruction, la gestion des horizons multivalués est désormais possible et pratique, permettant de considérer des champs plus complexes.
Mots clés : Horizons Multivalués, Modèle de Données, Indexation Spatiale, Structure d'Accélération, Reconstruction de Surface, Gridding, Paramétrisation, Polyline, Carte d'Élévation.

Abstract — Horizons are first-class elements of the structural models used for Oil & Gas exploration. When they have a complex shape, they become difficult or even outright impossible to manage within typical software packages. Handling those so-called "multivalued" horizons indeed requires tackling two problems. First, data models must be found to represent them. A review of the state of the art, complemented by a benchmark, led us to promote the use of patch systems and triangulated surfaces in order to represent reverse-faulted horizons and salt domes, respectively. Second, multivalued surfaces must be reconstructed from sparse and non-uniformly distributed interpretation data. We developed two reconstruction methods, each targeting a multivalued horizon type. They are based on standard geoscience interpolation methods, making them both fast and natural extensions of the state of the art. Using the proposed models along with the reconstruction methods, it is now possible to efficiently tackle the more complex fields where they are typically found.
Keywords: Multivalued Horizons, Data Model, Spatial Index, Acceleration Structures, Surface Reconstruction, Gridding, Parameterization, Polyline, Digital Elevation Model.

GIPSA-Lab, 11 rue des Mathématiques, Grenoble Campus BP46, F-38402
SAINT MARTIN D'HERES CEDEX