SEQUENTIAL SIMULATION WITH PATTERNS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF PETROLEUM ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Guven Burc Arpat
January 2005

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Dr. Jef Caers Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Dr. Andre Journel

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Dr. Hamdi Tchelepi

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Dr. Roland Horne

Approved for the University Committee on Graduate Studies.

Abstract

Flow in a reservoir is mostly controlled by the connectivity of extreme permeabilities (both high and low) which are generally associated with marked, multiple-scale geological patterns such as fluvial channels. Accurate characterization of such patterns requires multiple-point correlations that are much beyond the reach of the two-point correlations provided by a traditional variogram model.

In this thesis, a pattern-based geostatistical sequential simulation algorithm (SIMPAT) is proposed that redefines reservoir characterization as an image construction problem. The approach utilizes the training image concept of multiple-point geostatistics (MPS) but is not developed through probability theory. Rather, it considers the training image as a collection of multiple-scale patterns from which patterns are selected and pasted into the reservoir model such that they match any local subsurface data. A training image pattern is defined as a multiple-pixel configuration identifying a meaningful geological structure believed to exist in the reservoir. The framework of sequential simulation is used to achieve the simulation and conditioning of patterns. During sequential simulation, at each visited grid location, the algorithm looks for a training image pattern that is most 'similar' to the data event (the neighborhood of the currently visited grid node), i.e. the traditional conditional probability calculations of MPS are replaced with similarity calculations of image processing. One way of conceptualizing the proposed algorithm is to envision it as a method for solving jigsaw puzzles: The technique builds images (reservoir models) by assembling puzzle pieces (training image patterns). The method works equally well with both continuous (such as permeability) and categorical (such as facies) variables while conditioning to a variety of local subsurface data such as well logs, local angles and 3D seismic.

Acknowledgments

I wonder whether Jef (Prof. Jef Caers) ever put his head between his hands and thought "Why did we accept this guy to Stanford in the first place?". May be he did and who could blame him if that is the case? ;-). But, even if he did, he never let me doubt myself and always, always fully supported me. Thus, if his illuminating suggestions constitute the brain of this thesis, then his faith in me is the soul. For that, I am truly thankful.

And, without Andre's (Prof. Andre Journel) advices, continuous support, always interesting philosophy and dedication to the 'art' and science of geostatistics, life in Stanford would not have been the same. Both Jef and Andre have been good friends through out my stay at Stanford, not only my professors. I can only hope I have been worthy of their company.

I would also like to thank my defense committee members, Prof. Hamdi Tchelepi, Prof. Roland Horne and the committee chair Prof. Biondo Biondi. All of them were incredibly tolerant to my (literally) last minute submissions and had very valuable suggestions. Hamdi was also a member of my reading committee and together with Jef, they made sure that I never forget the flow aspect of the problem.

Several other Stanford-affiliated people contributed to the thesis: Dr. Sebastien Strebelle from ChevronTexaco set the initial motivations for different parts of the study and endured long hours of discussions about the methodology. Without Seb's practical insights, not only this thesis, but probably the entire field of multiple-point geostatistics would have been set many years back. Tuanfeng Zhang from Stanford University spent several days with me, explaining the implementation details of SNESIM, sharing his ideas about possible improvements and some 100% home-made

nooddles. Let me note that his cooking abilities are on par with his scientific abilities; simply terrific. The SCRF folks, specifically, Jianbing Wu, Jenya Polyakova, Anuar Bitanov, Dr. Nicolas Remy, Dr. Sunderrajan Krishnan and Dr. Guillaume Caumon also contributed to different parts of the study. With another group of folks, the "mysterious history matchers", we have always engaged in heated discussions about pretty much everything PE. Through out the years, I kept their identities as a secret when I thanked them in presentations but it's probably time for me to finally name their names: Inanc Tureyen, Dr. Joe Voelker, Todd Hoffman, Herve Gross and Melton Hows. Thanx guys! Finally, during his visit to Stanford, Prof. Albert Tarantola patiently listened to my babblings over and over again and shared his ideas which allowed me to see the problem from a completely different point of view.

I also had some industry support: Dr. Renjun Wen from geomodeling technology corp. provided us with the SBED software which we used for several synthetic test cases. Dr. Yuhong Liu and Dr. Tingting Yao from ExxonMobil (both recent Stanford grads), Dr. Chris Grant and Dr. Genbao Shi from Landmark and Dr. Marco Pontiggia from Agip shared their ideas and insights, which allowed me not to lose touch with the industry.

Of course, it would have been almost impossible for me to complete this thesis without the never disappearing support of my family and my friends. It seems even an ocean in between stands no chance of making my family forget their little (not literally!) baby boy. What can I say? I am truly incredibly lucky when it comes to family. And, my friends, especially Inanc Tureyen, Deniz Cakici, Banu Alkaya, Baris Guyaguler, Baris Aksoy, Burak Yeten, Ozgur Karacali, Irtek Uraz and the very special s.b.p.d. made life at Stanford feel very much like home.

Finally, it is time for a confession: It is rumored that, on one of the many drafts that Jef gave to Andre for one of his very first papers at Stanford, Andre noted down the quote "Labor omnia vincit improbus" after Jef quoted "Quousque tandem abutere patientia mea". Now that I am done with my Ph.D., I can confess that I always wanted to join this little conversation. And, after four years at Stanford, what do I have to say? Why, "Veni, vidi, volo in domum redire", of course, what else? ;-)

Contents

List of Figures

Chapter 1

Introduction

Flow in a reservoir is mostly controlled by the connectivity of extreme permeabilities (both high and low) which are generally associated with geological patterns that create preferential flow paths such as formed by high permeability sand channels. Such structures might have a radical effect on the flow behavior of a reservoir. Strebelle (2000) shows that a correct reproduction of fluvial channel structures in a clastic reservoir model has a significant effect on the predictive ability of the model. Caers et al. (1999) conclude that the inclusion of stochastic channels in the model is necessary for the prediction of the field water breakthrough and subsequent water build-up. Other authors (Hoffman and Caers, 2004; Tureyen and Caers, 2004) show that using geologically realistic models for history matching results in better flow prediction.

The traditional geostatistical approach to property modeling is through sequential simulation of facies and/or petrophysical properties. Practical software implementations (Deutsch and Journel, 1998) of sequential Gaussian simulation (`SGSIM`) and sequential indicator simulation (`SISIM`) are widely used for stochastic reservoir modeling. The aim of sequential simulation, as it was originally proposed, is to reproduce the histogram and spatial covariance of the attributes being simulated through the sequential drawing from conditional distributions (ccdfs). A random path sequentially visits each node of the model and simulated values are drawn from the conditional distribution of the value at that node given the neighboring subsurface data and previously simulated values.

However, these traditional sequential simulation algorithms, limited to the reproduction of two-point statistics, such as a variogram model, cannot reproduce complex geological structures. Such structures, for example, meandering channels, are curvilinear and exhibit complex cross-scale interdependencies, and their modeling requires multiple-scale, multiple-point correlations, much beyond the reach of the two-point correlation provided by a variogram model.

In the early nineties, Srivastava (1992) and Guardiano and Srivastava (1993) introduced the concept of "training image" as a replacement of the variogram within an extended sequential simulation framework. This proposal led to the development of multiple-point simulation (MPS) and geostatistics. MPS considers the training image as the random function model itself, directly providing a quantification of the heterogeneities that the geologist believes to exist in the reservoir. Training images as used by MPS reflect a prior geological concept, they need not be conditioned to any local data. The original MPS algorithm proposed by Srivastava (1992) identifies the conditional probabilities to sampling proportions of similar conditioning data events found in the training image. This method proved to be extremely CPU demanding because it requires complete rescanning of the training image for every new data event informing a node to be simulated. The major contribution of Strebelle (2000) was to use a special data structure to store all relevant training image events for fast lookup. The training image scanning had to be done only once. The algorithm of Strebelle (2000), called `SNESIM`, represents a landmark development in multiple-point geostatistics, being the first truly practical algorithm. See Caers et al. (2001), Strebelle et al. (2002), Caers et al. (2003), Liu et al. (2004) and Harding et al. (2004) for applications to real reservoirs.

With the introduction of `SNESIM`, reservoir modeling studies could afford considering increasingly complex training images. Such training images aim to better represent the actual heterogeneities of a geological deposition, including their multiple-scale dependency and pronounced trends, both vertically and horizontally. The current research on MPS focuses on better reproducing the geological patterns found in complex training images and on modeling the non-stationarity of such patterns (Arpat and Caers, 2004; Zhang et al., 2004; Maharaja, 2004).

The characteristic non-stationary and multiple-scale nature of a realistic training image calls for a different approach, potentially less anchored on traditional probability tools. The focus of these new tools is not the direct and explicit reproduction of specific multiple-point statistics but the direct and explicit reproduction of multiple-scale training image patterns. Based on this paradigm shift, this thesis describes a sequential simulation algorithm (`SIMPAT`; SIMulation with PATterns) that can generate realizations such as the one shown in Figure 1.1b using highly complex training images such as the training image shown in Figure 1.1a.

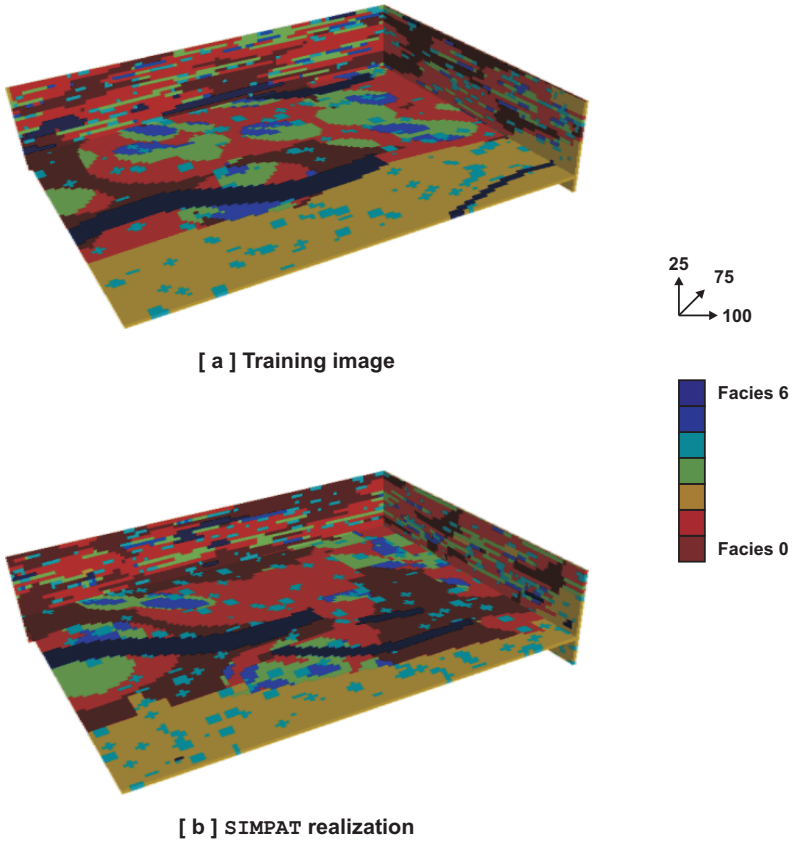


Figure 1.1: An unconditional realization (reservoir model) [b] generated using a highly complex and non-stationary training image [a] by applying the `SIMPAT` algorithm proposed in this thesis.

The proposed method is not developed through probability theory, rather it considers the training image as a collection of patterns from which one selects the patterns that locally match the reservoir well-log, core and seismic data. A training image pattern is defined as a multiple-pixel configuration identifying a meaningful geological structure believed to exist in the reservoir. During the simulation, the method looks for training image patterns that are 'similar' to the data event, i.e. the traditional probability calculations of MPS are replaced with similarity calculations. One way of conceptualizing the proposed algorithm is to envision it as a method for solving jigsaw puzzles: The technique builds images (reservoir models) by assembling puzzle pieces (training image patterns) that interlock with each other in a certain way while honoring the local data.

The thesis is structured as follows:

- Chapter 2 discusses the traditional modeling tools, the original MPS idea, the traditional method used to honor geological scales (called "multiple-grid"; Tran (1994)) and how data conditioning is performed within these approaches. The last section of the chapter motivates using an image construction approach (as found in the field of computer vision) as the basis of the reservoir modeling approach proposed in this thesis.

- Chapter 3 develops the details of the single-grid, unconditional algorithm using a simple, binary (sand/non-sand) training image. Several fundamental concepts of the algorithm such as patterns, data events and similarity are discussed and a formal notation to represent these concepts is introduced. This single-grid algorithm is then applied to different training images.

- Chapter 4 motivates the need for a method capable of reproducing large scale structures of training images. A modified version of the traditional multiple-grid approach is introduced as an extension of the single-grid algorithm described in the previous chapter. Additionally, a new multiple-grid approach, named "dual template simulation", is described which allows a better integration of the training image scale relations.

- Chapter 5 focuses on the key concept underlying the proposed algorithm, the similarity concept. First, the effects of using different similarity measures are discussed. Then, a multiple-point similarity measure is introduced. This measure is based on "proximity transforms", which relies on well-established concepts taken from the fields of computer vision and image processing.

- Chapter 6 discusses how data conditioning is performed within the proposed pattern-based framework. The chapter is divided into two parts: The first part discusses conditioning to hard data (such as well log or core data), the second part conditioning to soft data (such as seismic).

- Chapter 7 introduces the concept of region simulation. In region simulation, the reservoir is divided into several regions, each region with its own geological scenario (training image). Related topics such as conditioning to local angles are discussed in this chapter.

- Chapter 8 presents the software implementation of the proposed algorithm (named "`SIMPAT`") and additional utility software developed to assist `SIMPAT`. Several implementation details such as how the algorithm can be modified to run on multiple CPUs, are discussed in this chapter.

- Finally, Chapter 9 covers the conclusion and the future work.

Chapter 2

Motivation

Until the early eighties, most reservoir models consisted of simplistic layer cake models. In such models, homogeneous layers or fault blocks were given constant permeability values that did not reflect the smaller scale flow paths (high permeability conduits) and flow barriers (low permeability sheets) within these layers. Layer cake models were convenient for basic reservoir engineering calculations (such as mass balances) but could not provide the detail required by modern flow simulation applications.

Later, estimation methods that populate reservoir layers with continuously varying petrophysical values by means of interpolating the well data were implemented. One of such interpolation algorithms, kriging (Matheron, 1962), calls for a prior model of spatial correlation, namely the variogram model. Even today, kriging remains a popular method in the field of mining engineering. However, the limitations of kriging when applied to reservoir modeling are now well known. The spatial distribution of kriging estimates tend to be too smooth, in addition that smoothness is non-stationary, an artifact of well locations (Journel and Deutsch, 1993; Goovaerts, 1997). As a result of this artificial smoothing, kriging tends to overestimate the low petrophysical values and underestimate the high petrophysical values. This property of kriging can be a severe problem when the resulting models are used in flow-related calculations such as prediction of water breakthrough since the flow response of a reservoir is typically controlled by the connectivity of extreme permeabilities, and these cannot be reproduced by kriging or any other low-pass-type interpolation algorithm.

2.1 Review of Sequential Simulation

Simulation (Journel, 1974), and more specifically sequential Gaussian simulation (SGSIM; Deutsch and Journel (1998)), was introduced as a solution to the smoothing problem of kriging. Sequential simulation algorithms are 'globally' correct in that they reproduce a global structured statistics such as a variogram model, whereas kriging is 'locally' accurate in that it provides at each location a best estimate in a minimum error variance sense, regardless of estimates made at other locations. Since flow in a reservoir is controlled by the spatial disposition of permeability values in the reservoir, sequential simulation algorithms provide more relevant reservoir models which honor the global structure specified by the variogram.

The implementation of sequential simulation consists of reproducing the desired spatial properties through the sequential use of conditional distributions. Consider a set of N random variables $Z(\mathbf{u}_\alpha), \alpha = 1, \ldots, N$ defined at N locations $\mathbf{u}_\alpha$. The aim is to generate L joint realizations $\{z^{(l)}(\mathbf{u}_\alpha), \alpha = 1, \ldots, N\}$ with $l = 1, \ldots, L$ of the N RVs, conditional to n available data and reproducing the properties of a given multivariate distribution. To achieve this goal, the N-point multivariate distribution is decomposed into a set of N univariate conditional distributions (ccdfs):

$$\begin{aligned} F\left(\mathbf{u}_1, \ldots, \mathbf{u}_N; z_1, \ldots, z_N | (n)\right) &= \\ F\left(\mathbf{u}_N; z_N | (n+N-1)\right) &\times \\ F\left(\mathbf{u}_{N-1}; z_{N-1} | (n+N-2)\right) &\times \ \ldots \ \times \\ F\left(\mathbf{u}_2; z_2 | (n+1)\right) &\times F\left(\mathbf{u}_1; z_1 | (n)\right) \end{aligned} \tag{2.1}$$

where $F\left(\mathbf{u}_N; z_N | (n+N-1)\right) = Prob\left\{Z(\mathbf{u}_N) \leq z_N | (n+N-1)\right\}$ is the conditional cdf of $Z(\mathbf{u}_N)$ given the set of n original data values and the $(N-1)$ realizations $z^{(l)}(\mathbf{u}_\alpha), \alpha = 1, \ldots, N-1$ of the previously simulated values.

This decomposition allows to generate a realization by sequentially visiting each node on the simulation grid. In theory, the approach requires a full analytical expression for the ccdf at each step.

2.2 Variogram-based Geostatistics

In "variogram-based geostatistics", the multivariate distribution of the RF $Z(\mathbf{u})$ is considered to be fully determined by its first and second moment (histogram and variogram; i.e., two-points statistics). The other, higher order moments are frozen by the particular multi-variate distribution used (typically Gaussian) or by the particular algorithm selected (e.g. `DSSIM`; Journel (1994)). In this section, a review of the two main variogram-based algorithms, sequential Gaussian simulation (`SGSIM`) and sequential indicator simulation (`SISIM`), is given.

2.2.1 Sequential Gaussian Simulation

The only known model for which the decomposition 2.1 is analytically available is the multivariate Gaussian distribution. The governing spatial law (multivariate distribution) of the Gaussian model is fully characterized by the two-point variogram model. All marginal and conditional probability distributions are Gaussian, the latter determined using simple kriging of each unknown value at any node $\mathbf{u}_j$ given the original data and previously simulated values $(n + j - 1)$ (Deutsch and Journel, 1998).

This unique theoretical convenience is exploited in `SGSIM` and made that algorithm one of the most popular for reservoir modeling. However, the underlying model of `SGSIM`, the multivariate Gaussian distribution, is known to have several practical limitations (Journel and Alabert, 1990; Journel and Deutsch, 1993; Strebelle, 2000). The most critical of these limitations is the maximum entropy property of any multivariate Gaussian model, which states that the Gaussian model is the least structured among all models sharing the same covariance or variogram (Figure 2.1). Thus, although globally more accurate than kriging, sequential Gaussian simulation still fails to address the need for modeling specific patterns of structured geological connectivity.

2.2.2 Generalized Sequential Simulation Framework

Because of the limitation of the maximum entropy property of Gaussian models, the quest for simulation algorithms that could reproduce specific geological structures

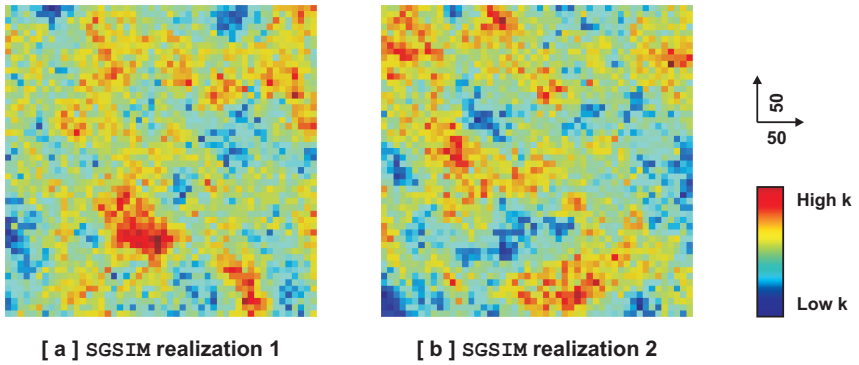


Figure 2.1: The maximum entropy property of the multivariate Gaussian model spatially scatters the extreme values of the reservoir on the SGSIM realizations.

remains open. To generate a wider class of simulation algorithms, research had to abandon pure analytical models, instead accepted a more ad-hoc approach where the random function is defined by the simulation algorithm actually implemented (Journel, 1992). Such an approach relies on the fact that any product of conditional distributions of the type equation 2.1 results in a specific multivariate distribution (not necessarily Gaussian) represented by its realizations. Rather than aiming to reproduce the properties of a known multivariate distribution, one aims to import certain statistical properties (histogram, variogram and possibly higher order statistics) through the sequential simulation of properties at each grid location. Accepting this paradigm, various sequential simulation methods were developed in the recent years, all following the same generic flowchart:

1. Define a random path visiting all uninformed nodes.
2. For each node $\mathbf{u}_\alpha, \alpha = 1, \ldots, N$ do,
   (a) Model or infer the conditional distribution $F\left(\mathbf{u}_\alpha; z_\alpha | (n + \alpha - 1)\right)$ of $Z(\mathbf{u}_\alpha)$, given the n original data values and all $\alpha - 1$ previously drawn values $z^{(l)}(\mathbf{u}_\beta)$, $\beta = 1, \ldots, \alpha - 1$;
   (b) Draw the simulated value $z^{(l)}(\mathbf{u}_\alpha)$ from $F\left(\mathbf{u}_\alpha; z_\alpha | (n + \alpha - 1)\right)$;
3. Continue until all nodes on the simulation grid have been simulated.

The only difference between the various sequential simulation algorithms is the way one derives and draws from the conditional distributions. The type and shape of this ccdf depends on the available conditioning data and on the statistical properties one wants the ultimate simulated realization to exhibit. One such geologically 'better' algorithm calculating the ccdf different from `SGSIM` is `SISIM`.

2.2.3 Sequential Indicator Simulation and Beyond

Sequential indicator simulation (`SISIM`) uses an indicator-based approach (Journel, 1982; Deutsch and Journel, 1998). In this approach, the ccdf $F(\mathbf{u}; z|(n))$ is modeled by a series of K threshold values z_k, discretizing the range of variation of z:

$$F(\mathbf{u}; z_k|(n)) = \text{Prob}\{Z(\mathbf{u}) \leq z_k|(n)\} \quad k = 1, \ldots, K \tag{2.2}$$

The indicator approach is based on the interpretation of the conditional probability equation (2.2) as the conditional expectation of an indicator random variable $I(\mathbf{u}; z_k)$ given the information (n):

$$F(\mathbf{u}; z_k|(n)) = \text{E}\{I(\mathbf{u}; z_k)|(n)\} \tag{2.3}$$

with $I(\mathbf{u}; z_k) = 1$ if $Z(\mathbf{u}) \leq z_k$ and zero otherwise. The algorithm was initially developed for the simulation of K categorical variables $I(\mathbf{u}; k)$ with $I(\mathbf{u}; k) = 1$ if facies k prevails at $\mathbf{u}$, zero otherwise.

Unlike `SGSIM`, `SISIM` permits the use of different indicator variograms to model the relations of each of the K ranges, i.e. one can account for category-specific patterns of spatial continuity. Thus, correlating and connecting the extreme values of a reservoir, as different from median values, becomes a possibility (Figure 2.2). Once the K ccdfs $F(\mathbf{u}; z_K|(n))$ are calculated, the simulation follows the generic sequential flowchart.

`SISIM` added more variogram flexibility to reservoir modeling but it was quickly recognized that the limitation in reproducing curvilinear geological shapes such as meandering channels, is in the variogram itself, not in the use of indicators. As a two-point statistics, the variogram (no matter how many) cannot capture complex curvilinear shapes. Some solutions have been proposed to address the curvilinearity

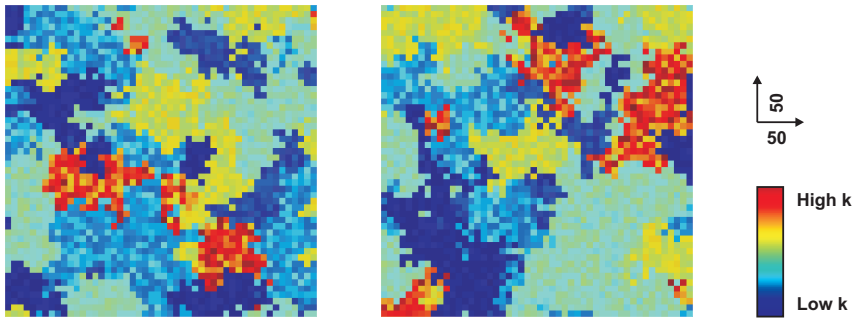


Figure 2.2: SISIM achieves connectivity of extreme values of the reservoir through using different variograms for different categories.

problem. For example, Xu (1996) proposes to use a distribution of local angles to adapt the local indicator kriging systems to varying anisotropy resulting in curvilinear structures. However, such adapted structures tend to be discontinuous. Clearly, the solution to generating curvilinear features is to abandon the variograms altogether.

2.3 Multiple-point Geostatistics

The need to go beyond the two-point statistics provided by a variogram calls for methods that explicitly utilize multiple-point statistics. Such methods would account for correlations between three or more locations at a time. Hence, in theory, they would be able to reproduce the connectivity of many locations and thus reproduce complex, curvilinear geological structures.

2.3.1 Training Images

The multiple-point correlation models required by multiple-point geostatistics methods are typically provided by a training image. Training images need not be conditioned to any location-specific subsurface data. They need only reflect a prior geological concept.

In theory, training images can come from several different sources such as interpreted outcrop photographs or a geologist's sketch, properly digitized and extended to 3D. In current practice, training images are almost always generated using unconditional object-based (Haldorsen and Damsleth, 1990; Deutsch and Wang, 1996; Holden et al., 1998; Viseur, 1999) or process-based (Wen et al., 1998) simulations. Some training image examples are shown in Figure 2.3.

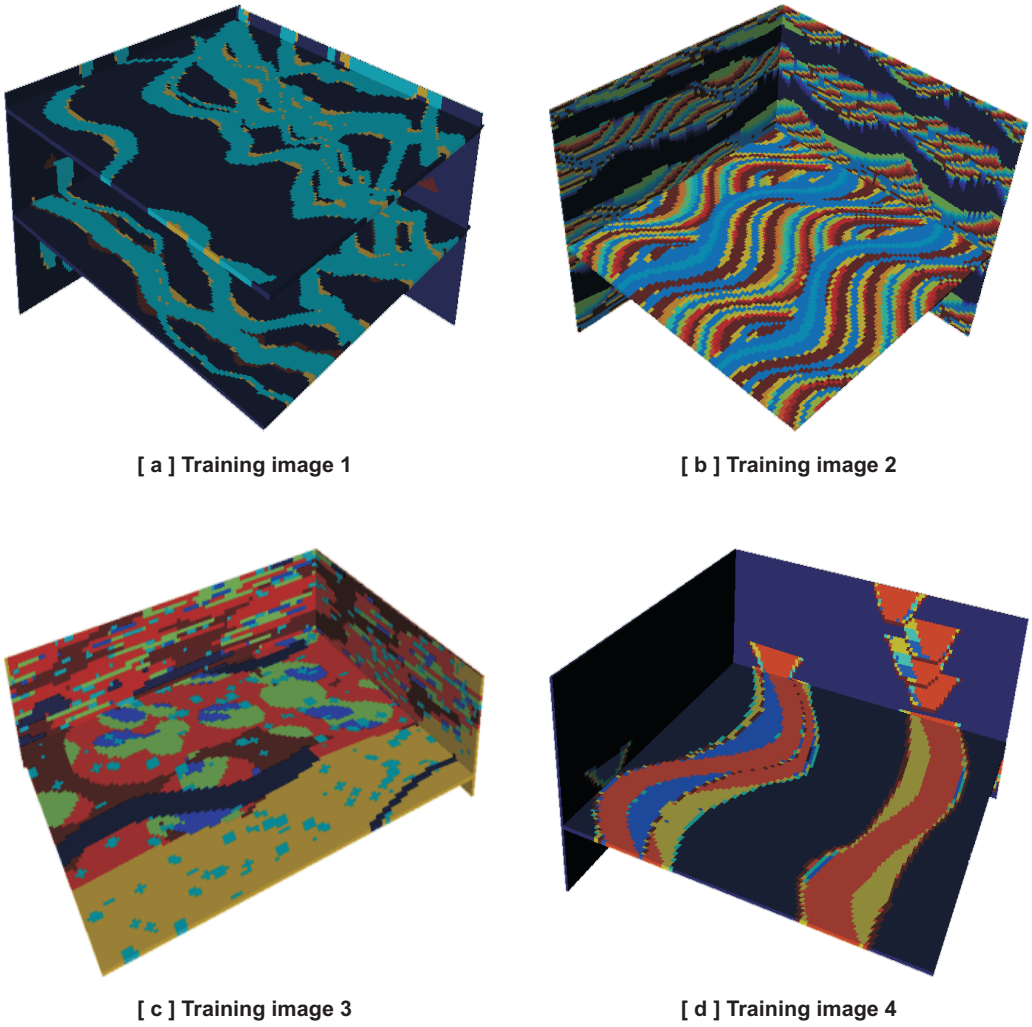


Figure 2.3: Examples of training images. All images generated using unconditional object-based or processed-based modeling tools.

2.3.2 Methods Utilizing Multiple-point Information

Methods utilizing multiple-point information can be examined in four main classes:

1. Simulated annealing has been used to reproduce a few specific multiple-point statistics previously modeled from a training image (Farmer, 1990; Deutsch, 1992). In this approach, higher order, multiple-point statistics are used as explicit constraints that each realization should honor through an objective function. Besides the fact that only very few statistics can be simultaneously considered as such constraints, simulated annealing suffers from problems of convergence related to the difficulty in choosing the correct set of annealing parameters, i.e. an efficient cooling schedule.

2. A more statistical approach is presented in Tjelmeland (1996) using a Markov random field. While the technique has definite theoretical appeal (any set of consistent multiple-point statistics can be considered), the method is iterative, extremely CPU demanding, and may not converge satisfactorily, and, thus far has not been proved to be practical for 3D applications.

3. A recent method proposed in Caers and Journel (1998) and Caers et al. (1999) uses artificial neural networks (ANN) to model multiple-point statistics inferred from a training image and utilizes the ANN for drawing simulated values. This algorithm produces reasonably good results. Yet, it remains iterative in nature; hence, is CPU-demanding and prone to convergence issues. Furthermore, issues related to the neural network architecture make it difficult to tune.

4. Another approach to multiple-point geostatistics was introduced by Srivastava (1992) and Guardiano and Srivastava (1993). The approach follows the extended sequential simulation framework and has a remarkably simple underlying idea: at each unsampled node, infer the local conditional probability by scanning the training image for replicates of the data event. The node is then simulated using this ccdf and considered as conditioning data from thereon. Since the conditioning data configuration is allowed to vary, the simulation is direct and avoids the convergence issues of iterative algorithms.

However, Srivastava's original implementation was extremely CPU-demanding because the full training image had to be rescanned at each unsampled node. Strebelle (2000) proposed to use a special data structure to overcome this drawback of an otherwise potentially powerful idea.

2.3.3 The SNESIM algorithm

Strebelle's algorithm is called SNESIM (Strebelle, 2000), which stands for "Single Normal Equation Simulation". The algorithm is given this name to insist on the fact that it utilizes only a single normal equation when modeling the probability of a facies at a particular grid node. That single normal equation is actually nothing but the Bayes relation defining a conditional probability. Previously, Journel (1992) had shown the connection between multiple-point geostatistics and the extended evaluation of probabilities via an extended systems of normal (kriging) equations. Instead of modeling the multiple-point statistics from some lower order statistics, the multiple-point probability is identified to corresponding experimental proportions read from the training image. Hence, the method eliminates the need to solve a full kriging system; instead, it derives the probability directly from a single normal equation equivalent to the identification of a proportion.

SNESIM scans the training image using a pre-defined data template to extract training image events. For every data event, SNESIM searches for replicates of that event, then retrieves the corresponding histogram of the central value. For example, in a binary (sand/non-sand) training image, if a data event is found 10 times with 3 out of these 10 replicates yielding a sand central node, SNESIM evaluates the sand conditional probability as 0.3. Once data events and their associated central values are retrieved from the training image, SNESIM stores them in a dynamic data structure called a "search tree" (Knuth, 1997). This is as opposed to Srivastava's original proposal which called for rescanning the training image for each new data event.

The algorithm, then, follows the flowchart of a typical sequential simulation algorithm, visiting unsampled nodes using a random path and simulating these nodes conditional on available original data and previously simulated values. Drawing from

the experimental training image proportions, SNESIM simulates the value at the current node and tags this nodal value as hard datum, which will condition the simulation of the nodes visited later in the sequence. SNESIM reproduces the structure of the training image (Figure 2.4) in as much as it reproduces its conditional proportions and, at the same time, it can honor any well or seismic data. For further details on the SNESIM algorithm, the reader is referred to Strebelle (2000, 2002).

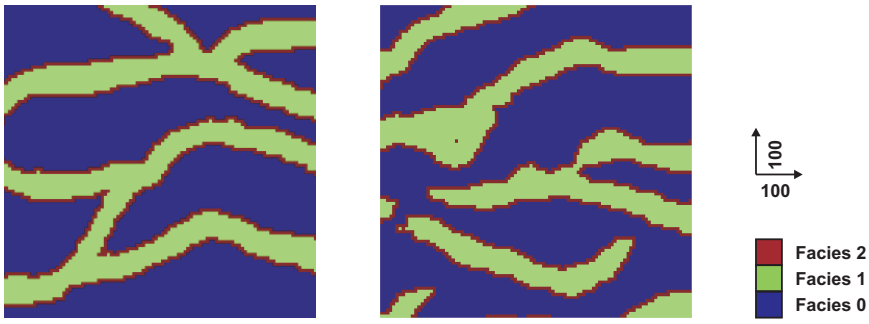


Figure 2.4: An unconditional SNESIM realization [b] and the training image used for the simulation [a]. The training image structures (as well as global training image proportions) are successfully reproduced in the final realization due to the use of multiple-point information.

The strength of SNESIM lies in the simple yet rigorous methodology of combining the traditional sequential simulation paradigm with the search tree concept for storing ccdfs derived from a training image and an effective use of the multiple-grid approach (Tran (1994); explained in the next section). However, the SNESIM algorithm has shortcomings when used with complex training images:

The search tree only stores data events that actually occur in the training image. When a data event is not repeated often enough in the training image, the SNESIM algorithm drops one datum from that event, typically the most distant from the center, and repeats the lookup. For complex training images and large templates, this might lead to removal of many data nodes from the neighborhood until a match is found in the training image. Dropping a data node implies a reduction in large scale

information which may result in poor reproduction of the training image patterns. One solution to this problem is to use larger and pattern-rich training images. But, the memory required by the corresponding search tree would increase correspondingly since it is determined by the number of unique events within the training image. Thus, for large and rich training images, the algorithm might require several GBs of memory, depending on the size of the template.

One might choose a relatively small template and rely on the multiple-grid approach (see the next section) to reproduce large-scale connectivities. But, even with a small template, the memory requirement might still be unacceptable, especially for multiple-facies training images with a number of facies greater than 5. It can be argued that the joint simulation of more than 2 or 3 facies can be bypassed through a hierarchical simulation of facies combinations as explained by Maharaja (2004). Alternatively, a variant multiple-grid simulation method utilizing sub-grids (where the templates are allowed to be even smaller) can be used (Strebelle, 2002). Regardless, the multiple-grid approach utilized by `SNESIM` was originally devised for variogram-based geostatistics and thus exhibits several shortcomings when applied to reproduction of multiple-point patterns as explained in the following section.

2.4 Multiple-grid Simulation

According to theory, each ccdf should be derived conditional to the original data and all previously simulated values (Equation 2.1). On large simulation grids, this poses a problem because of the increasing amount of previously simulated nodes. The ccdf of the last simulated node would have to be conditional to all $N-1$ previously simulated values; a very large number. As a result, calculation of such conditional cdfs become very CPU and possibly memory demanding. All sequential simulation methods suffer from this problem.

To reduce the CPU and memory demand, practical implementations of sequential simulation algorithms rely on the following two solutions: First, any conditioning data that does not fall within a given search template (neighborhood) centered at the current node to be simulated is discarded. Such reduction of the data makes inference

of the ccdfs easier, but as recognized by Tran (1994), may degrade the reproduction of long range correlations, especially when the search template is small. Therefore, to compensate for dropping the furthest data values, Tran (1994) proposes to use a multiple-grid approach.

On a Cartesian grid, the idea is to use a set of cascading grids and sparse templates (neighborhoods) instead of a single grid and one large dense template. The aim is to approximate the single large and dense template by means of increasingly sparser templates (so that CPU and memory demand remain reasonable for each sparse template). The approximation is carried out according to the following coarsening criterion: Consider a number n_g of increasingly finer grids. The g-th ($1 \leq g \leq n_g$) grid is constituted by each $2^{(g-1)}$-th node of the final (finest; $g = 1$) simulation grid. Accordingly, the nodes of the template $\mathbf{T}$ get 'expanded', i.e. the coarse template $\mathbf{T}^g$ has the same structural configuration of nodes as template $\mathbf{T}$ but with spacing 2^{g-1} times larger. See Figure 2.5 for an illustration of multiple-grid concepts.

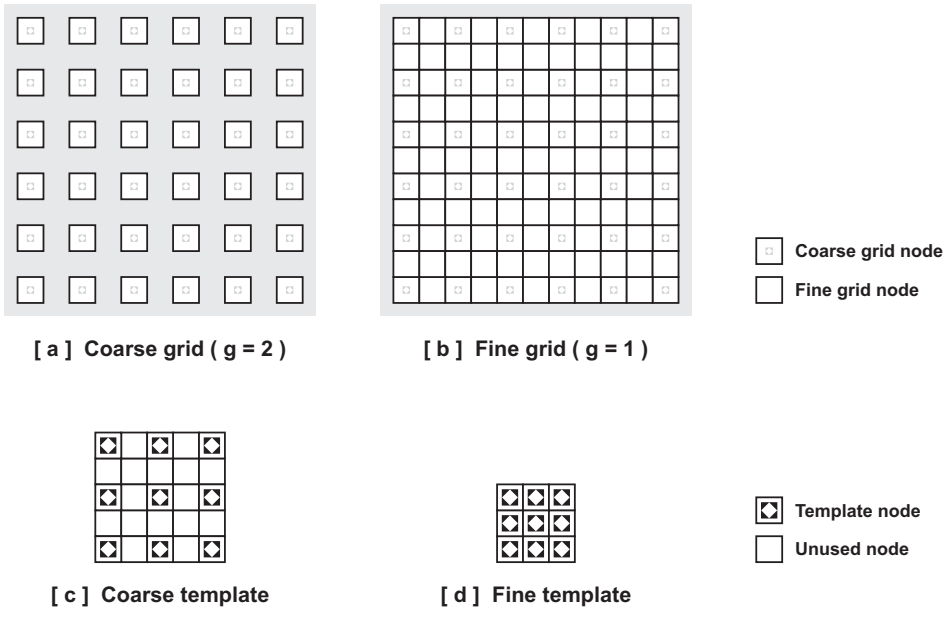


Figure 2.5: Illustration of the multiple-grid concepts for $n_g = 2$.

Algorithms using the multiple-grid approach proceed first by performing sequential simulation on the coarsest grid $g = n_g$. The values simulated on the current grid are frozen and transferred to the next finer grid as conditioning data, then g is set to $g - 1$. This succession of nested-grid simulations continues until the simulation of the finest grid $g = 1$ is completed.

While the multiple-grid method works reasonably well for reasonably 'homogeneous heterogeneities' with high entropy as considered by sequential Gaussian and indicator simulation, it has its shortcomings when attempting to reproduce lower entropy, structurally heterogeneous, multiple-scale geological patterns.

The first problem is related to the level of approximation introduced by the use of coarse templates. A coarse data event scanned by a coarse template is essentially an approximation of the actual fine scale data event, in that the coarse template selectively samples from the fine scale nodes of the complete, fine data event (according to the coarsening criterion as explained above). The key of the traditional multiple-grid approach is the assumption that this coarse sampling is representative of the actual, fine scale data event if applied successively. However, when attempting to reproduce complex, multiple-scale geological patterns (typical of realistic training images), such an assumption may not hold. The same coarse data event may correspond to several radically different finer data events in the training image. In such a case, assuming the coarse data event to be representative of the underlying finer information is simply inadequate. The finer grid simulations cannot reintroduce the correct fine information as they have no way to access the entire large scale structure as determined by the coarse grid. The fine grid can only model smaller parts of the geological complexity, limited by the increasingly smaller and finer template sizes. As a result, the traditional multiple-grid works well only if the geological continuity model used during the simulation has a clear hierarchy between coarser and finer scales. This is true for realizations generated using variogram-based methods such as `SGSIM` and `SISIM` since a variogram model jointly quantifies continuity at all scales. However, a complex, realistic training image does not exhibit such a 'homogeneous' hierarchy between its scales and thus the scale relations of such a training image cannot be fully reproduced using the traditional multiple-grid approach.

Another limitation is that, the traditional multiple-grid approach does not allow the values determined by coarser grids to be modified during the finer grid simulations. In other words, a decision made during a coarse grid simulation with few conditioning data is never revised. Consider a two multiple-grid, two facies (sand/non-sand) system associated to an indicator-based sequential simulation (Figure 2.6). Assume the training image used for the simulation consists only of continuous horizontal lines. Once a decision (possibly poor) is made on the coarse grid, that decision is frozen. In Figure 2.6, the discontinuity is introduced early on during the simulation of the coarse grid which propagates during the fine grid simulation.

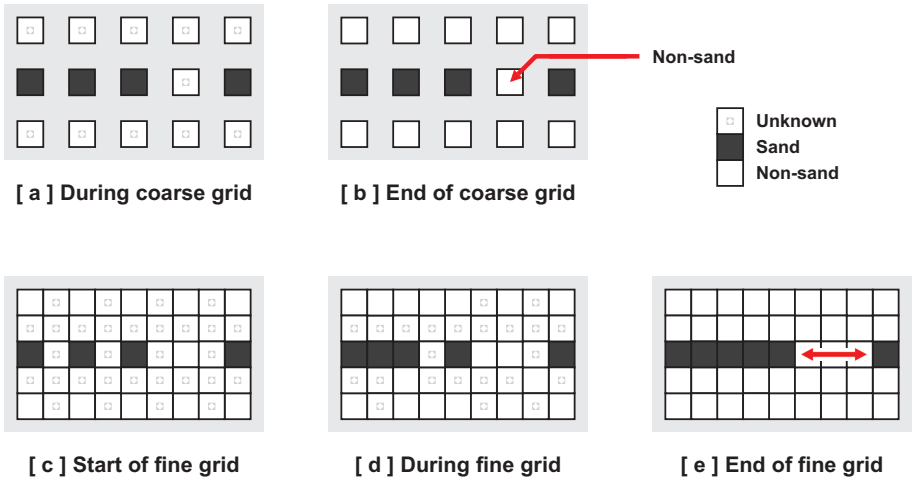


Figure 2.6: Illustration of the traditional multiple-grid simulation. Once a decision is made on the coarse grid [b], there is no way to 'correct' this decision later [c - e].

Last, conditioning to dense hard data may become problematic when performed in a traditional multiple-grid setting. This problem is discussed in the next section.

2.5 Data Conditioning

The term "hard data" is used to emphasize the fact that the modeling method should exactly reproduce this data value (typically point data obtained from wells) at its

location. Conversely, "soft data" (typically obtained from seismic surveys) are to be used only as a 'guide' to modeling and are not expected to be reproduced exactly.

2.5.1 Hard Data Conditioning

Hard data conditioning not only refers to exact reproduction of the original data values but also requires generating appropriate continuity around those locations, as dictated by the geological continuity model. For example, if the geology of a reservoir is known to contain thin, continuous and horizontal sheets of shale, a well data indicating a shale value at a particular location should be honored by generating such a shale sheet at and near that data location. Hard data conditioning is therefore more than simply injecting the data into the model and freezing it.

Variogram-based simulation algorithms always exactly honor hard data values through the use of kriging, which is known to be an exact interpolation method (Deutsch, 2002). MPS methods, on the other hand, achieve data conditioning by selecting whole training image events that are consistent with the available data.

Although the conditioning approach employed by MPS methods gives satisfactory results in practice, there are still remaining issues to be solved. For example, the `SNESIM` algorithm looks for an exact match in the search tree, dropping data nodes if necessary until such a match is found. But, dropping data nodes amounts to a loss of the large scale information which may result in generation of artificial discontinuities in the final realization. Consider a template containing only two hard data values: One datum near the center and the other near the edge of the template. In such a case, if no training event exactly matching these two data can be found in the search tree, `SNESIM` drops the furthest data and repeats the lookup; ignoring the large scale information provided by the second datum. Thus, there exists a risk that `SNESIM` might not constrain fully to the information carried by the two data combined.

Another problem occurs due to the use of multiple-grid simulation. In real applications, the spacing between coarse grid nodes may be several hundreds feet in the horizontal and several tens of feet in the vertical. Such a coarse grid cannot accommodate dense well data (dense in the horizontal or vertical). To overcome this problem,

MPS methods utilizing the traditional multiple-grid approach either (a) relocate the data itself (Strebelle, 2000) or (b) interpolate the data value to the nearest coarse grid node. However, neither of these solutions fully honor the data: either (a) the data location or (b) the data value is not honored anymore. In real reservoirs, the data relocation problem becomes more critical as wells might be relocated several hundreds of feet away from their original location. Fig. 2.7 illustrates this problem.

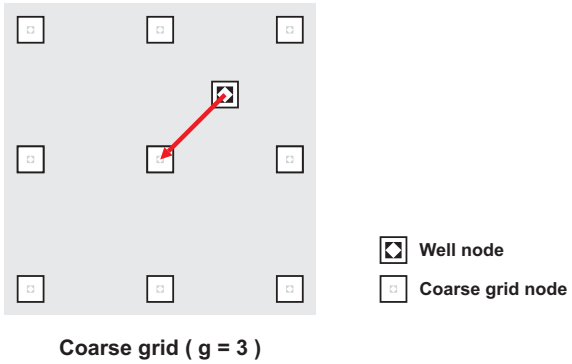


Figure 2.7: Illustration of data relocation in a multiple-grid setting. To account for the data on the coarse grid, the data is relocated to the nearest coarse grid node.

2.5.2 Soft Data Conditioning

In reservoir modeling, soft data typically refers to extensive data obtained from indirect measurements acquired using some form of remote sensing (e.g. geophysical methods such as seismic). Soft data often represents a 'filtered' view of the subsurface heterogeneity where filtering can be represented by a function $\mathbf{F}$ mapping the true Earth's heterogeneity into a filtered view (soft data) as provided by the remote sensing device. In general, $\mathbf{F}$, the exact physical response, is not known and is approximated by a mathematical model $\mathbf{F}^*$, sometimes termed the "forward model".

Variogram-based methods integrate soft data through some form of co-kriging (Deutsch and Journel, 1998). They rely on a pure statistical model and ignore any physical relationship between the data and the unknown. Co-kriging requires the joint modeling of the variogram of the property being simulated (primary variogram), the

variogram of the soft data (secondary variogram) and the cross-variogram (the two-point spatial correlation between the hard and soft variables). Such a modeling task is rarely practical in 3D and is typically approximated using a variety of methods such as Markov models (Goovaerts, 1997; Deutsch, 2002). Furthermore, even when all the variograms required by the full co-kriging are available, the correlations imposed by these variograms remain limited to two-point statistics.

MPS methods take a different approach to integrating soft data. The soft data is coded into prior probabilities using well-calibration or clustering methods (Payrazyan, 1998; Avseth, 2000; Caers et al., 2003). Once such soft-data-based prior probabilities are calculated, they are combined with the hard-data-based probabilities as read from the training image (Strebelle, 2000; Journel, 2002).

In `SNESIM`, the above idea is implemented as follows: Define $P(A|\mathbf{B})$ as the conditional cdf read from the training image where A represents the facies indicator value to be simulated on the current node $\mathbf{u}$ and $\mathbf{B}$ is the data event consisting of the hard data and previously simulated values. Similarly, the probability derived from soft data can be written as $P(A|\mathbf{C})$ where $\mathbf{C}$ is a vector of seismic values centered around $\mathbf{u}$. To integrate jointly both $\mathbf{B}$ and $\mathbf{C}$ data, a model for recombining the pre-posterior probabilities $P(A|\mathbf{B})$ and $P(A|\mathbf{C})$ into a single probability $P(A|\mathbf{B},\mathbf{C})$ is required. `SNESIM` calculates this combined probability as suggested by Journel (2002):

$$\frac{x}{b} = \frac{c}{a} \tag{2.4}$$

with

$$x = \frac{1 - P(A|\mathbf{B},\mathbf{C})}{P(A|\mathbf{B},\mathbf{C})},\ b = \frac{1 - P(A|\mathbf{B})}{P(A|\mathbf{B})},\ c = \frac{1 - P(A|\mathbf{C})}{P(A|\mathbf{C})},\ a = \frac{1 - P(A)}{P(A)} \tag{2.5}$$

where $P(A)$ is the prior probability (global proportion) of the facies. Using this relation, the probability $P(A|\mathbf{B},\mathbf{C})$ can be calculated from:

$$P(A|\mathbf{B},\mathbf{C}) = \frac{a}{a + b \cdot c} \tag{2.6}$$

Equation 2.4 amounts to a standardized conditional independence assumption between the $\mathbf{B}$ and $\mathbf{C}$ data, i.e. the seismic information $\mathbf{C}$ contributes the same to A regardless of how informative the geological information $\mathbf{B}$ is. However, one would

normally expect the contribution of $\mathbf{C}$ to decrease as $\mathbf{B}$ becomes more informative. To account for this dependence, Journel (2002) proposes a generalization (known as the "tau model"):

$$\frac{x}{b} = \left(\frac{c}{a}\right)^{\tau} \tag{2.7}$$

where setting the tuning parameter τ different from 1 accounts for different models of dependency between $\mathbf{B}$ and $\mathbf{C}$. See Caers et al. (2001) for an application of the tau model to a real reservoir with a varying τ and Krishnan et al. (2004) for a detailed theoretical account of the tau model including suggestions on how to estimate τ.

The above soft data conditioning methodology has been successfully applied to real reservoirs (Caers et al., 2001; Strebelle et al., 2002; Caers et al., 2003; Liu et al., 2004). However, it has one shortcoming that arguably limits its application: The probabilities $P(A|\mathbf{B})$ and $P(A|\mathbf{C})$ are calculated independently and combined later using the tau model. In other words, the method never jointly considers $\mathbf{B}$ and $\mathbf{C}$. Instead, the information provided by both $\mathbf{B}$ and $\mathbf{C}$ are essentially 'summarized' into single-point probabilities and then combined. Although it is shown in Krishnan et al. (2004) that the tau model provides closure, i.e. that there exists always a τ such that $P(A|\mathbf{B})$ and $P(A|\mathbf{C})$ can be combined exactly into $P(A|\mathbf{B},\mathbf{C})$, such a closure expression remains only of theoretical interest; no practical methods for determining an estimate of τ have been established yet.

Strebelle (2000) suggests an alternative soft data conditioning method that directly models $P(A|\mathbf{B},\mathbf{C})$. This method requires pairs of training images, namely, the original training image and a soft training image, typically obtained by applying the approximate forward model $\mathbf{F}^*$ to the original training image. The method has a definite appeal since the use of the tau model and the estimation of the τ parameter are no longer required; $P(A|\mathbf{B},\mathbf{C})$ is directly calculated from the training image pair. However, practical implementations using this approach have been lacking, mainly due to its CPU and memory limitations (Strebelle, 2000).

2.6 Reservoir Modeling as Image Construction

An alternative approach to the sampling strategy of Srivastava (1992) and Strebelle (2000) is to redefine the problem as a direct image construction problem instead of construction of higher order statistics (or any conditional probabilities derived thereof). The aim is not to explicitly reproduce multiple-point statistics of a training image but to directly reproduce multiple-scale training image patterns in a stochastic manner (which implicitly reproduces multiple-point statistics). Such an image construction task is commonly encountered in computer vision and image processing; especially in the field of texture synthesis (Bonet, 1997; Paget and Longstaff, 1998; Wang et al., 1998; Palmer, 1999; Bar-Joseph, 1999; Xu et al., 2000; Portilla and Simoncelli, 2000; Efros and Freeman, 2001; Wei, 2001; Liang et al., 2001).

Adopting the image construction strategy has one potential advantage. Algorithms of the sort are typically less limited by the stationarity requirement that is common to all probabilistic methods discussed in the previous sections. This is achieved by completely abandoning the explicit use of probabilities. Instead, image construction algorithms typically rely on the concept of similarity, i.e. such algorithms construct the final image (realization) based on the similarity of individual data events to training image patterns rather than building from a probability calculated conditional to the data event.

Another advantage of the image construction approach is the ability to capture pattern-to-pattern relations of a training image. Variogram-based algorithms capture only point-to-point correlations. MPS algorithms (such as `SNESIM`) use pattern-to-point correlations, i.e. an entire data event (vector of values) is considered to simulate a single node. However, using pattern-to-pattern relations (simulating a whole vector of values at a time instead of a single value) is especially important when conditioning to high quality seismic information. Such high quality soft data typically relate best to patterns in a realization (for example, small pieces of a meandering channel) and less to individual facies values. Thus, using point-to-point or pattern-to-point relations essentially amounts to not fully utilizing this information. In theory, probabilistic MPS algorithms can be modified to account for pattern-to-pattern relations of a

training image. This requires calculation of the conditional cdf of an entire pattern instead of the conditional cdf of a single node. Such a probabilistic modeling would not be an easy task, simply because the training image would need to contain enough repetitions (statistics) of all patterns. On the other hand, as will be shown, the image construction strategy and the similarity concept allows approximations, thus easier inference of pattern-to-pattern relationships.

This thesis investigates the applicability of the image construction approach to reservoir modeling. The subsequent chapters of the thesis build step-by-step an image construction algorithm (`SIMPAT`) that uses the similarity concept, utilizes pattern-to-pattern relations, and, in general, heavily relies on many commonly found concepts in the field of computer vision and image processing. However, since reservoir modeling is more than a mere image construction task, it should be stressed that, `SIMPAT` is more than a simple application of traditional and known image processing tools. Such traditional tools typically do not deal with 3D images, do not distinguish between hard and soft data, have limitations when dealing with categorical variables, uncertainty and strong non-stationarity. Images and textures as found in computer vision and image processing are not geology. Hence, many of these traditional tools had to redesigned to fit into the challenge of 3D reservoir modeling.

Chapter 3

Single-grid, Unconditional Algorithm

Consider the 11×11, binary (sand/non-sand) training image **ti** shown in Figure 3.1. Using this simple training image, this chapter explains the steps of a single-grid, unconditional simulation algorithm. First, the training image is preprocessed to extract its constituent patterns. Then, these patterns are used as building blocks within a sequential simulation algorithm. Last, several examples using this single-grid algorithm are given.

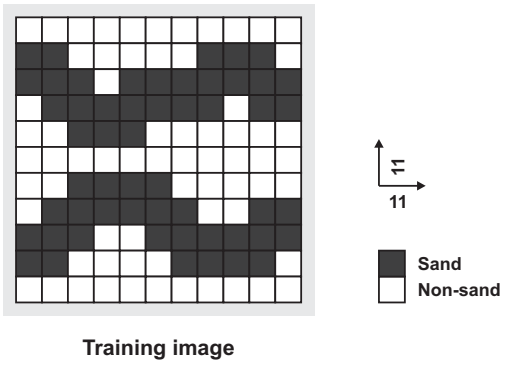


Figure 3.1: A 11×11, binary (sand/non-sand) training image. This simple training image is used throughout the thesis to explain concepts and algorithms.

3.1 Preprocessing of the Training Image

Define $ti(\mathbf{u})$ as a value of the training image $\mathbf{ti}$ where $\mathbf{u} \in \mathbf{G}_{ti}$ and $\mathbf{G}_{ti}$ is the regular Cartesian grid discretizing the training image. In general, $ti(\mathbf{u})$ can be a continuous, categorical or vectorial variable. In this chapter, for illustration purposes, the training image of interest (Figure 3.1) is a binary (sand/non-sand) system and thus an indicator notation is used for $ti(\mathbf{u})$:

$$ti(\mathbf{u}) = \begin{cases} 0 & \text{if at } \mathbf{u} \text{ } \mathbf{ti} \text{ contains non-sand} \\ 1 & \text{if at } \mathbf{u} \text{ } \mathbf{ti} \text{ contains sand} \end{cases} \tag{3.1}$$

$\mathbf{ti}_{\mathbf{T}}(\mathbf{u})$ indicates a specific multiple-point vector of $ti(\mathbf{u})$ values within a template $\mathbf{T}$ centered at node $\mathbf{u}$; i.e., $\mathbf{ti}_{\mathbf{T}}(\mathbf{u})$ is the vector:

$$\mathbf{ti}_{\mathbf{T}}(\mathbf{u}) = \left\{ ti(\mathbf{u}+\mathbf{h}_1), ti(\mathbf{u}+\mathbf{h}_2), \ldots, ti(\mathbf{u}+\mathbf{h}_\alpha), \ldots, ti(\mathbf{u}+\mathbf{h}_{n_{\mathbf{T}}}) \right\} \tag{3.2}$$

where the $\mathbf{h}_\alpha$ vectors are the vectors defining the geometry of the $n_{\mathbf{T}}$ nodes of template $\mathbf{T}$ and $\alpha = 1, \ldots, n_{\mathbf{T}}$. The vector $\mathbf{h}_1 = 0$ identifies the central location $\mathbf{u}$. Figure 3.2 illustrates these concepts for a 3×3 2D template, i.e. $n_{\mathbf{T}} = 9$.

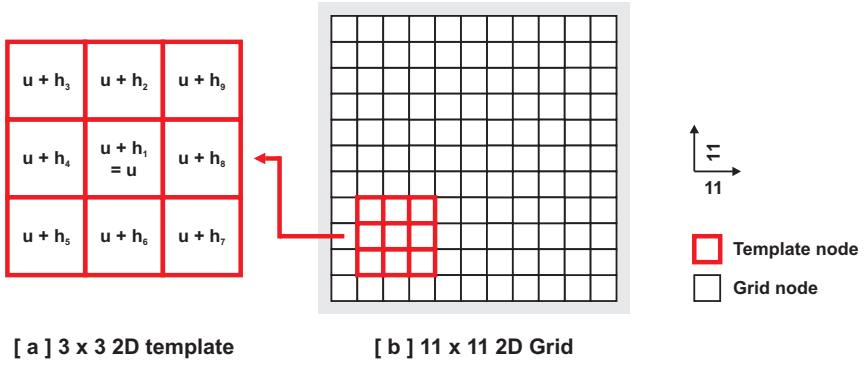


Figure 3.2: Illustration of concepts related to templates on a 3×3 2D template. The vector $\mathbf{h}_1 = 0$ identifies the central location $\mathbf{u}$.

The preprocessing of the training image **ti** is performed by scanning the image using template **T** and storing the corresponding multiple-point $\mathbf{ti}_{\mathbf{T}}(\mathbf{u})$ vectors in a database. Each such $\mathbf{ti}_{\mathbf{T}}(\mathbf{u})$ vector is called a "pattern" of the training image and the database is called the "pattern database" and is denoted by $\mathbf{patdb}_{\mathbf{T}}$. Figure 3.3 illustrates this process for a 3×3 2D template.

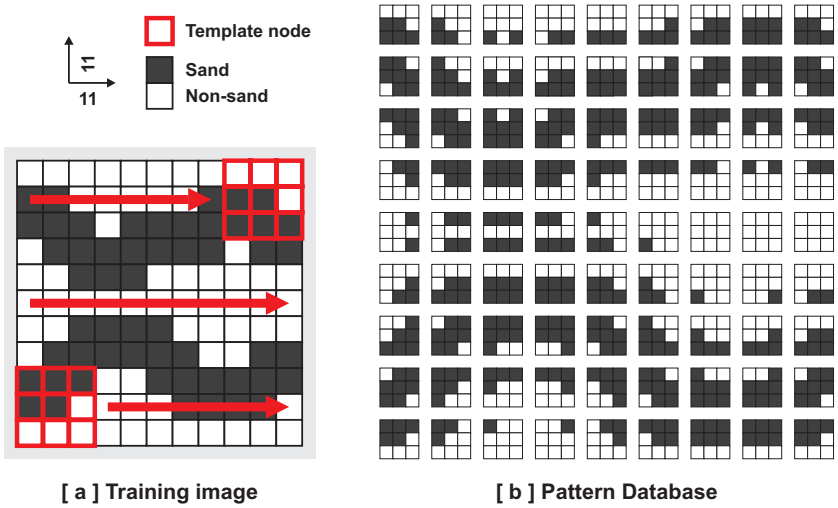


Figure 3.3: Preprocessing of the training image (of Figure 3.1) to obtain the pattern database using a 3×3 2D template.

By definition, training image patterns are fully informed, i.e. each $ti(\mathbf{u} + \mathbf{h}_\alpha)$ node of a pattern is informed and cannot fall 'outside' of the training image **ti**. For this reason, the scanning of the training image can only be performed on the eroded image including only nodes $\mathbf{u}$ such that $\mathbf{u} + \mathbf{h}_\alpha \in \mathbf{G}_{ti}$ with $\alpha = 1, \ldots, n_{\mathbf{T}}$.

Once the patterns are stored in the pattern database $\mathbf{patdb}_{\mathbf{T}}$, they are considered to be location independent, i.e. the database does not store the location $\mathbf{u} \in \mathbf{G}_{ti}$ of a pattern; only the content of the pattern is stored. Such patterns are referred to as $\mathbf{pat}_{\mathbf{T}}^{k}$ where k denotes the particular k-th configuration of the previous vector of values $\mathbf{ti}_{\mathbf{T}}(\mathbf{u})$ of the training image **ti**, with each value now denoted by $pat_{\mathbf{T}}^{k}(\mathbf{h}_\alpha)$, where $k = 1, \ldots, n_{\mathbf{pat}_{\mathbf{T}}}$ and $n_{\mathbf{pat}_{\mathbf{T}}}$ is the number of total available patterns in the pattern database $\mathbf{patdb}_{\mathbf{T}}$. In other words, k is an index used to refer to a specific

pattern in the database. Then:

$$\mathbf{pat}_{\mathbf{T}}^{k} = \left\{ pat_{\mathbf{T}}^{k}(\mathbf{h}_1), pat_{\mathbf{T}}^{k}(\mathbf{h}_2), \ldots, pat_{\mathbf{T}}^{k}(\mathbf{h}_\alpha), \ldots, pat_{\mathbf{T}}^{k}(\mathbf{h}_{n_{\mathbf{T}}}) \right\} \tag{3.3}$$

where all $n_{\mathbf{pat}_{\mathbf{T}}}$ patterns are defined on the same template $\mathbf{T}$. This definition of a pattern does not require uniqueness, i.e. a pattern $\mathbf{pat}_{\mathbf{T}}^{k_1}$ can have the same configuration and values as $\mathbf{pat}_{\mathbf{T}}^{k_2}$ (i.e., $\mathbf{pat}_{\mathbf{T}}^{k_1} \equiv \mathbf{pat}_{\mathbf{T}}^{k_2}$) even when $k_1 \neq k_2$.

In any given training image $\mathbf{ti}$, there is a finite maximum number of $n_{\mathbf{pat}_{\mathbf{T}}}^{max}$ of possible patterns (i.e. possible pixel configurations) defined over a template $\mathbf{T}$. For example, for a binary (sand/non-sand) training image defined over a regular Cartesian grid $\mathbf{G}_{ti}$, a 2D template $\mathbf{T}$ of size 3×3 yields $n_{\mathbf{pat}_{\mathbf{T}}}^{max} = 2^{3\times 3} = 2^9$. In practice, the number of patterns that can be extracted from a finite training image is much less because of the absence of fully random structures within the training image, i.e. not all possible pixel configurations exist in the training image. The actual maximum number of possible patterns that can be extracted from a training image is denoted by $n_{\mathbf{pat}_{\mathbf{T}}}^{ti} << n_{\mathbf{pat}_{\mathbf{T}}}^{max}$. In Figure 3.3, $n_{\mathbf{pat}_{\mathbf{T}}}^{ti} = 81$. Notice that, $n_{\mathbf{pat}_{\mathbf{T}}}^{ti}$ is different from the total number of nodes in $\mathbf{G}_{ti}$ ($= 121$) due to the erosion of the image with the template $\mathbf{T}$. In theory, a filter can be applied to discard the infrequent patterns of the training image $\mathbf{ti}$ or patterns with undesirable characteristics to reach the final smaller $n_{\mathbf{pat}_{\mathbf{T}}}$ count. Therefore, the 'active' number ($n_{\mathbf{pat}_{\mathbf{T}}}$) of patterns retained in the pattern database might be even smaller than $n_{\mathbf{pat}_{\mathbf{T}}}^{ti}$ and typically, $n_{\mathbf{pat}_{\mathbf{T}}} \leq n_{\mathbf{pat}_{\mathbf{T}}}^{ti} << n_{\mathbf{pat}_{\mathbf{T}}}^{max}$. In this chapter, no such filter is applied to the pattern database $\mathbf{patdb}_{\mathbf{T}}$ and thus $n_{\mathbf{pat}_{\mathbf{T}}} = n_{\mathbf{pat}_{\mathbf{T}}}^{ti} = 81$.

The final algorithm for preprocessing of the training image is given below:

Algorithm 3.1: Preprocessing of the training image

---

1. Scan the training image $\mathbf{ti}$ using the template $\mathbf{T}$ over the grid $\mathbf{G}_{ti}$ to obtain all existing patterns $\mathbf{pat}_{\mathbf{T}}^{k}$, $k = 1, \ldots, n_{\mathbf{pat}_{\mathbf{T}}}^{ti}$ that occur over the training image.

2. If desired, reduce the number of patterns to $n_{\mathbf{pat}_{\mathbf{T}}}$ by applying filters to construct the final pattern database $\mathbf{patdb}_{\mathbf{T}}$.

3.2 Sequential Simulation with Patterns

Once the pattern database $\mathbf{patdb_T}$ is constructed, the algorithm proceeds with the simulation of these patterns on a realization **re**. Initially, **re** is completely uninformed, i.e. nodes $re(\mathbf{u}) \in \mathbf{G}_{re}$ of the realization are unknown. A flag notation $re(\mathbf{u}) = \chi$ is used for such 'unknown' or 'uninformed' nodes. Thus, the definition of $re(\mathbf{u})$ used in this chapter becomes:

$$re(\mathbf{u}) = \begin{cases} \chi & \text{if at } \mathbf{u} \text{ } \mathbf{re} \text{ is uninformed (can be sand or non-sand)} \\ 0 & \text{if at } \mathbf{u} \text{ } \mathbf{re} \text{ contains non-sand} \\ 1 & \text{if at } \mathbf{u} \text{ } \mathbf{re} \text{ contains sand} \end{cases} \tag{3.4}$$

In general and similar to $ti(\mathbf{u})$, $re(\mathbf{u})$ can refer to a continuous, categorical or vectorial value. The only requirement is that $re(\mathbf{u})$ refers to the same type of variable as $ti(\mathbf{u})$, a binary indicator variable for this chapter.

Simulation on the realization **re** follows the footsteps of the extended sequential simulation framework previously described in Section 2.2.2. Before describing the detailed steps of this algorithm, two important concepts associated with the simulation of patterns, namely data events and similarity, are explained.

3.2.1 Data Events

In sequential simulation, a data event $\mathbf{dev_T}(\mathbf{u})$ is defined as the set of hard data and previously simulated values found in the template $\mathbf{T}$ centered on the visited location $\mathbf{u}$ where $\mathbf{T}$ is the same template used to scan the training image. Then:

$$\mathbf{dev_T}(\mathbf{u}) = \left\{ dev_{\mathbf{T}}(\mathbf{u}+\mathbf{h}_1), \ldots, dev_{\mathbf{T}}(\mathbf{u}+\mathbf{h}_\alpha), \ldots, dev_{\mathbf{T}}(\mathbf{u}+\mathbf{h}_{n_{\mathbf{T}}}) \right\} \tag{3.5}$$

where $dev_{\mathbf{T}}(\mathbf{u}+\mathbf{h}_\alpha) = re(\mathbf{u}+\mathbf{h}_\alpha)$ and **re** is the realization as previously defined. In other words, $\mathbf{dev_T}(\mathbf{u}) = \mathbf{re_T}(\mathbf{u})$.

Different from a pattern $\mathbf{pat}_{\mathbf{T}}^{k}$, a data event $\mathbf{dev_T}(\mathbf{u})$ is allowed to contain unknown values denoted by χ. Such unknown nodes correspond to the uninformed nodes of the realization **re** and these nodes will be informed by the algorithm as the simulation progresses.

3.2.2 Similarity

$s\langle \mathbf{x}, \mathbf{y} \rangle$ is used to denote a generic similarity function returning a scalar value; the vector entries $\mathbf{x}$ and $\mathbf{y}$, in a simulation context, are data events and/or patterns. The scalar returned by $s\langle \mathbf{x}, \mathbf{y} \rangle$ indicates how 'similar' $\mathbf{x}$ is to $\mathbf{y}$. In other words, if $s\langle \mathbf{x}, \mathbf{y} \rangle > s\langle \mathbf{x}, \mathbf{z} \rangle$, $\mathbf{y}$ is said to be 'more similar' to $\mathbf{x}$ than $\mathbf{z}$. When using this notation, by convention, the data event is always written first, i.e. $s\left\langle \mathbf{dev_T}(\mathbf{u}), \mathbf{pat}_\mathbf{T}^k \right\rangle$, meaning the "similarity of data event $\mathbf{dev_T}(\mathbf{u})$ to pattern $\mathbf{pat}_\mathbf{T}^k$".

In practice, defining dissimilarity is easier than defining similarity. Thus, practical algorithms using the similarity concept typically describe similarity through a dissimilarity or distance function $d\langle \mathbf{x}, \mathbf{y} \rangle$ (Duda et al., 2001) with $d\langle \mathbf{x}, \mathbf{y} \rangle \geq 0$ and $d\langle \mathbf{x}, \mathbf{y} \rangle = 0$ indicating complete similarity, i.e. $\mathbf{x} = \mathbf{y}$. The relation between similarity and dissimilarity is then given by $s\langle \mathbf{x}, \mathbf{y} \rangle = -d\langle \mathbf{x}, \mathbf{y} \rangle$.

Ideally, the dissimilarity function $d\langle \mathbf{x}, \mathbf{y} \rangle$ should satisfy the following conditions:

1. $d\langle \mathbf{x}, \mathbf{y} \rangle \geq 0$;
2. $d\langle \mathbf{x}, \mathbf{y} \rangle = 0$ only if $\mathbf{x} = \mathbf{y}$ (identity);
3. $d\langle \mathbf{x}, \mathbf{y} \rangle = d\langle \mathbf{y}, \mathbf{x} \rangle$ (symmetry);
4. $d\langle \mathbf{x}, \mathbf{z} \rangle \leq d\langle \mathbf{x}, \mathbf{y} \rangle + d\langle \mathbf{x}, \mathbf{z} \rangle$ (triangle inequality).

In practice, the last two conditions, which are required to make $d\langle \mathbf{x}, \mathbf{y} \rangle$ a metric (Duda et al., 2001), are sometimes relaxed for certain dissimilarity functions.

Chapter 5 further focuses on the similarity concept and compares several different (dis)similarity measures. In this chapter, for clarity, a simple, single-point dissimilarity function, namely the Manhattan distance, is used. Single-point dissimilarity functions calculate the dissimilarity simply by comparing each collocated component (node) of vectors $\mathbf{x}$ and $\mathbf{y}$ taken one at a time. The Manhattan distance between a data event ($\mathbf{x}$) and a pattern ($\mathbf{y}$) is defined as:

$$d\left\langle \mathbf{dev_T}(\mathbf{u}), \mathbf{pat}_\mathbf{T}^k \right\rangle = \sum_{\alpha=0}^{n_\mathbf{T}} \left| dev_\mathbf{T}(\mathbf{u} + \mathbf{h}_\alpha) - pat_\mathbf{T}^k(\mathbf{h}_\alpha) \right| \tag{3.6}$$

By definition, the Manhattan distance requires each vector entry to be fully defined, i.e. no missing values are allowed. However, a data event $\mathbf{dev_T}(\mathbf{u})$ might contain unknown nodes, i.e. $dev_{\mathbf{T}}(\mathbf{u}+\mathbf{h}_\alpha)=\chi$ for some $\mathbf{h}_\alpha$. The computer vision literature defines several approaches to deal with such missing values (Duda et al., 2001). One common method within the context of the Manhattan distance is to simply ignore the missing values, i.e. the missing nodes are not used in the distance calculation. In other words, for the distance function given in Equation 3.6, if $\mathbf{u}+\mathbf{h}_\alpha=\chi$, the node $\mathbf{h}_\alpha$ is simply skipped during the summation. This is essentially a modification to the original Equation 3.6 such that,

$$d\left\langle \mathbf{dev_T}(\mathbf{u}), \mathbf{pat}_{\mathbf{T}}^k \right\rangle = \sum_{\alpha=0}^{n_{\mathbf{T}}} d_\alpha \left\langle dev_{\mathbf{T}}(\mathbf{u}+\mathbf{h}_\alpha), pat_{\mathbf{T}}^k(\mathbf{h}_a) \right\rangle \tag{3.7}$$

and,

$$d_\alpha \langle x_\alpha, y_\alpha \rangle = \begin{cases} |x_\alpha - y_\alpha| & \text{if } x_\alpha \text{ and } y_\alpha \text{ are both known, i.e. } \neq \chi \\ 0 & \text{if either } x_\alpha \text{ or } y_\alpha \text{ is unknown, i.e. } = \chi \end{cases} \tag{3.8}$$

Figure 3.4 shows some results for the Manhattan distance when applied to some sample patterns previously shown in Figure 3.3. Since the patterns are extracted from a binary (sand/non-sand) training image (Figure 3.3a), the values of pattern nodes are represented by the indicator equation 3.1. Recall that, similarity $s\langle \mathbf{x}, \mathbf{y} \rangle$ is simply the negative of the dissimilarity values in the table.

It is important to note that, due to the way the Manhattan distance is defined, the similarity function of Equation 3.7 could be applied to any type of variable, i.e. the same similarity function can be used for both continuous and categorical values. Furthermore, extension to vectorial values is also trivial as explained in Chapter 5.

3.2.3 Simulation Algorithm (Single-grid)

Equipped with the similarity concept, the single-grid, unconditional algorithm can now be explained in detail. During simulation, the nodes $\mathbf{u} \in \mathbf{G}_{re}$ of the realization $\mathbf{re}$ are randomly visited and at every node $\mathbf{u}$, the data event $\mathbf{dev_T}(\mathbf{u})$ is extracted. This data event is compared to all available patterns $\mathbf{pat}_{\mathbf{T}}^k$ in the pattern database $\mathbf{patdb_T}$

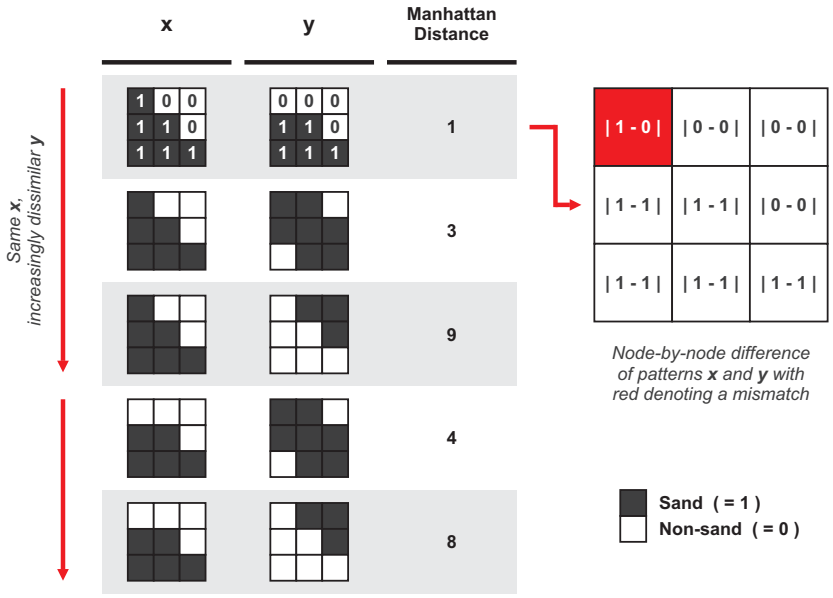


Figure 3.4: Application of the Manhattan distance when applied to sample binary (sand/non-sand) patterns previously shown in Figure 3.3b.

using a predefined similarity criterion (In this chapter, the Manhattan distance). The aim is to find the pattern $\mathbf{pat}_{\mathbf{T}}^{*}$ 'most similar' to the data event $\mathbf{dev}_{\mathbf{T}}(\mathbf{u})$, i.e. $\mathbf{pat}_{\mathbf{T}}^{*}$ is the pattern that maximizes $s\left\langle \mathbf{dev}_{\mathbf{T}}(\mathbf{u}), \mathbf{pat}_{\mathbf{T}}^{k} \right\rangle$ for $k = 1, \ldots, n_{\mathbf{pat}_{\mathbf{T}}}$. If several patterns $\mathbf{pat}_{\mathbf{T}}^{k}$ are equally similar to the data event $\mathbf{dev}_{\mathbf{T}}(\mathbf{u})$, one of them is randomly selected to be the most similar pattern. Once this most similar pattern $\mathbf{pat}_{\mathbf{T}}^{*}$ is found, the data event $\mathbf{dev}_{\mathbf{T}}(\mathbf{u})$ is replaced by this pattern, i.e. the values of the most similar pattern $\mathbf{pat}_{\mathbf{T}}^{*}$ are pasted on to the realization $\mathbf{re}$ at the current node $\mathbf{u}$, replacing all values of the data event $\mathbf{dev}_{\mathbf{T}}(\mathbf{u})$. Recall that the simulation considered here is unconditional, that is there is no prior original hard data values.

Algorithm 3.2 describes the above process. This algorithm is different from traditional sequential simulation algorithms such as `SGSIM` and `SNESIM` in two ways:

1. Patterns (set of nodes) are simulated instead of one single node at a time;

2. Previously simulated values can be 'updated', i.e. modified if the most similar pattern $\mathbf{pat}_{\mathbf{T}}^{*}$ dictates so.

Algorithm 3.2: Single-grid, unconditional simulation

1. Define a random path on the grid $\mathbf{G}_{re}$ of the realization **re** to visit each node $\mathbf{u} \in \mathbf{G}_{re}$ only once.
2. At every node $\mathbf{u}$, extract the data event $\mathbf{dev_T}(\mathbf{u})$ from the realization **re** and find the $\mathbf{pat}^*_{\mathbf{T}}$ that maximizes $s\left\langle \mathbf{dev_T}(\mathbf{u}), \mathbf{pat}^k_{\mathbf{T}} \right\rangle$ for $k = 1, \ldots, n_{\mathbf{pat_T}}$ in the pattern database $\mathbf{patdb_T}$, i.e. $\mathbf{pat}^*_{\mathbf{T}}$ is the 'most similar' pattern to $\mathbf{dev_T}(\mathbf{u})$.
3. If several $\mathbf{pat}^k_{\mathbf{T}}$ are equally similar to $\mathbf{dev_T}(\mathbf{u})$, randomly select one of such patterns to be the most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$.
4. Once the most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$ is found, assign $\mathbf{pat}^*_{\mathbf{T}}$ to $\mathbf{dev_T}(\mathbf{u})$, i.e. for all the $n_{\mathbf{T}}$ nodes $\mathbf{u} + \mathbf{h}_\alpha$ within the template $\mathbf{T}$, $dev_{\mathbf{T}}(\mathbf{u} + \mathbf{h}_\alpha) = pat^*_{\mathbf{T}}(\mathbf{h}_\alpha)$.
5. Move to the next node of the random path and repeat the above steps until all grid nodes along the random path are exhausted.

During the simulation, if any data event $\mathbf{dev_T}(\mathbf{u})$ is completely uninformed (typically during the initial steps of the random path), all patterns $\mathbf{pat}^k_{\mathbf{T}}$ of the pattern database $\mathbf{patdb_T}$ are considered equally similar to the data event $\mathbf{dev_T}(\mathbf{u})$. In such a case, the algorithm simply selects randomly a pattern from the pattern database $\mathbf{patdb_T}$ as the most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$ and proceeds with the next node. Since the same pattern can occur several times in the pattern database (assuming no filtering has been applied to remove non-unique patterns from the database), this random selection of patterns accounts for the frequency of occurrence of a pattern in the training image.

Figure 3.5 and 3.6 illustrates the first few steps of Algorithm 3.2 when applied to a 11×11 realization using the training image of Figure 3.1, the $\mathbf{patdb_T}$ of Figure 3.3 and a 3×3 template. In Figure 3.5, the first four steps of the random path lead to completely uninformed data events and thus the algorithm randomly selects a pattern for each location. In the fifth step (Figure 3.6h), the data event $\mathbf{dev_T}(\mathbf{u}_5)$ is partially informed with previously simulated nodes. Thus, the most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$ is found through a similarity calculation. This calculation reveals three patterns that

are equally similar to the data event $\mathbf{dev_T}(\mathbf{u}_5)$. The algorithm randomly selects one of these patterns and proceeds with the next node along the random path. Visiting all remaining nodes in this fashion, the final realization is obtained (Figure 3.6k).

3.3 2D Examples

This section shows application of the single-grid, unconditional algorithm to simple, 2D training images. The training images are chosen to demonstrate the use of the algorithm when applied to binary, multiple-category, and continuous variables. It is also shown that when a sufficiently large template is used, the large-scale patterns of a training image are captured despite using only a single grid.

Figure 3.7 shows the results of the algorithm when applied to a 250×250 binary training image using templates of size 9×9 and 35×35. When using the smaller 9×9 template, the algorithm successfully reproduces all the small-scale details of the training image but large-scale patterns, i.e. the overall structure of the channel network, is not reproduced. The larger template of 35×35 'sees' this overall structure better and thus produces better results (See Figure 3.8). However, using this larger template increases the total run-time of the single-grid algorithm approximately 15 fold over the 9×9 template simulation.

Figure 3.7c, e shows the E-types (averages) over 25 realizations of Figure 3.7b, d. Each realization is different, uniquely defined by a random seed that determines the random path and the random selection of patterns (the latter occurs whenever a data event has equal similarity to several patterns). Every new random seed thus provides stochasticity. Since no data conditioning is done, the E-types do not show any spatial structure as expected from averages of unconditional realizations.

Figure 3.9 demonstrates the use of a multiple-category training image; in this case, 3 facies with shale, sand and levee. Since the Manhattan distance (Equation 3.7) is indifferent to the nature of the simulation variable, the simulation algorithm remains the same. However, the values $ti(\mathbf{u})$ of the multiple-category training image $\mathbf{ti}$ can no longer be represented using the binary indicator equation 3.1. Instead, the equation

below is used (where the values assigned to each facies is determined arbitrarily):

$$ti(\mathbf{u}) = \begin{cases} 2 & \text{if at } \mathbf{u}\ \mathbf{ti} \text{ contains levee} \\ 1 & \text{if at } \mathbf{u}\ \mathbf{ti} \text{ contains channel} \\ 0 & \text{if at } \mathbf{u}\ \mathbf{ti} \text{ contains shale} \end{cases} \tag{3.9}$$

Figure 3.9 gives similar results to Figure 3.7 in that, the small 5×5 template fails to capture the large-scale patterns of the training image whereas the large 15×15 template produces better results but runs very slowly (approximately 10 times slower).

Figure 3.10 applies the single-grid algorithm to a continuous training image. The training image of the example is obtained through an unconditional `SGSIM` realization. Similar to the multiple-category case, the simulation algorithm remains the same but the values $\mathbf{ti}(\mathbf{u})$ of the training image are now continuous. The larger 15×15 template successfully captures the large-scale patterns (in this case, the long range of the variogram) but runs slow (approximately 10 times slower).

The above examples all show that the single-grid algorithm works well when a sufficiently large template is used to capture the large-scale patterns of a training image. However, using such a large template may not be practical due to excessive CPU-demand, particularly if applied to real 3D reservoirs.

The next chapter discusses a multiple-grid approach that aims to remain efficient both in terms of CPU and memory demand via approximating a large and dense template by several cascading sparse templates and coarse grids.

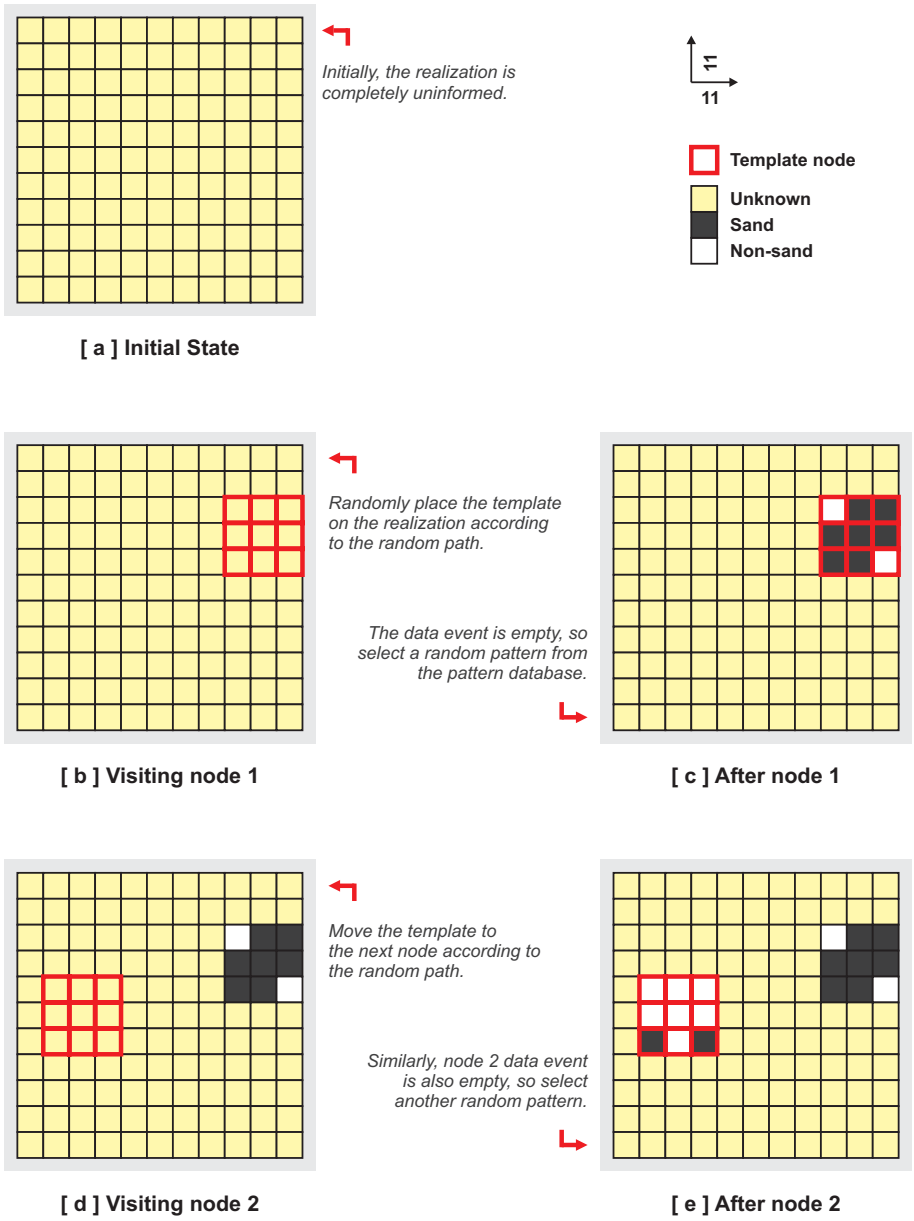


Figure 3.5: Internal steps of Algorithm 3.2 when applied to a 11×11 realization using the training image of Figure 3.1, the pattern database of Figure 3.3 and a 3×3 template. The figure continues on the next page as Figure 3.6.

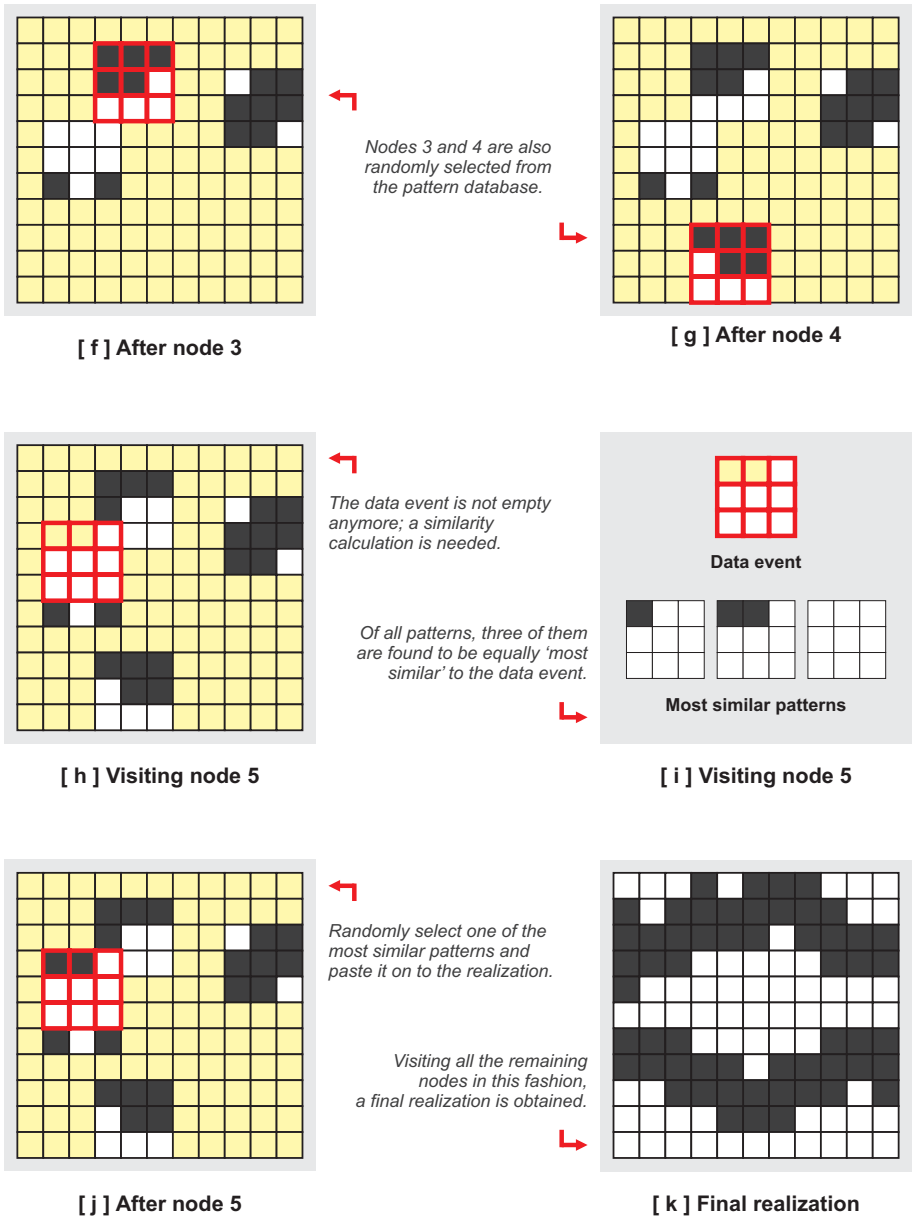


Figure 3.6: Continuation of Figure 3.5 from the previous page. Internal steps of Algorithm 3.2 when applied to a 11×11 realization using the training image of Figure 3.1, the pattern database of Figure 3.3 and and a 3×3 template.

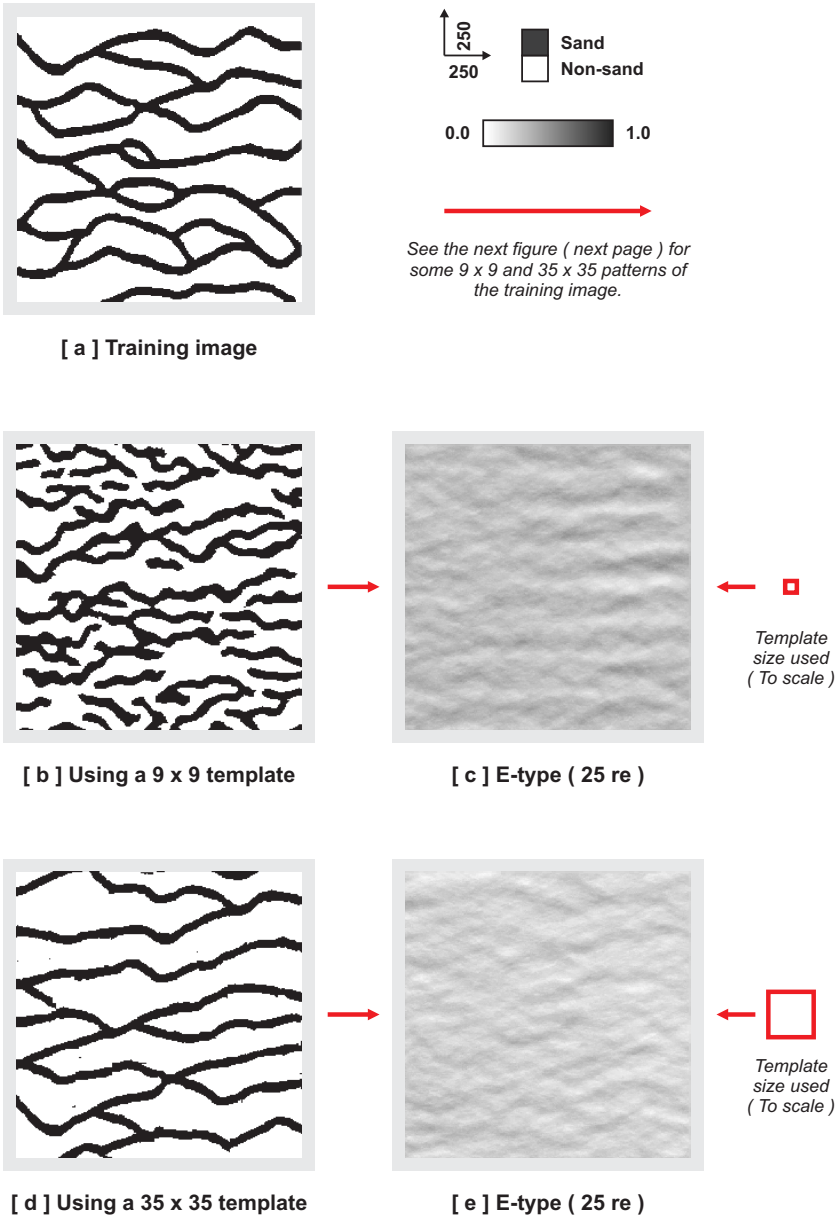


Figure 3.7: Application of the single-grid, unconditional algorithm to a 250×250, binary training image. [b] and [d] show realizations for different template sizes $n_{\mathbf{T}}$. [c] and [e] show E-types (averages) over 25 realizations.

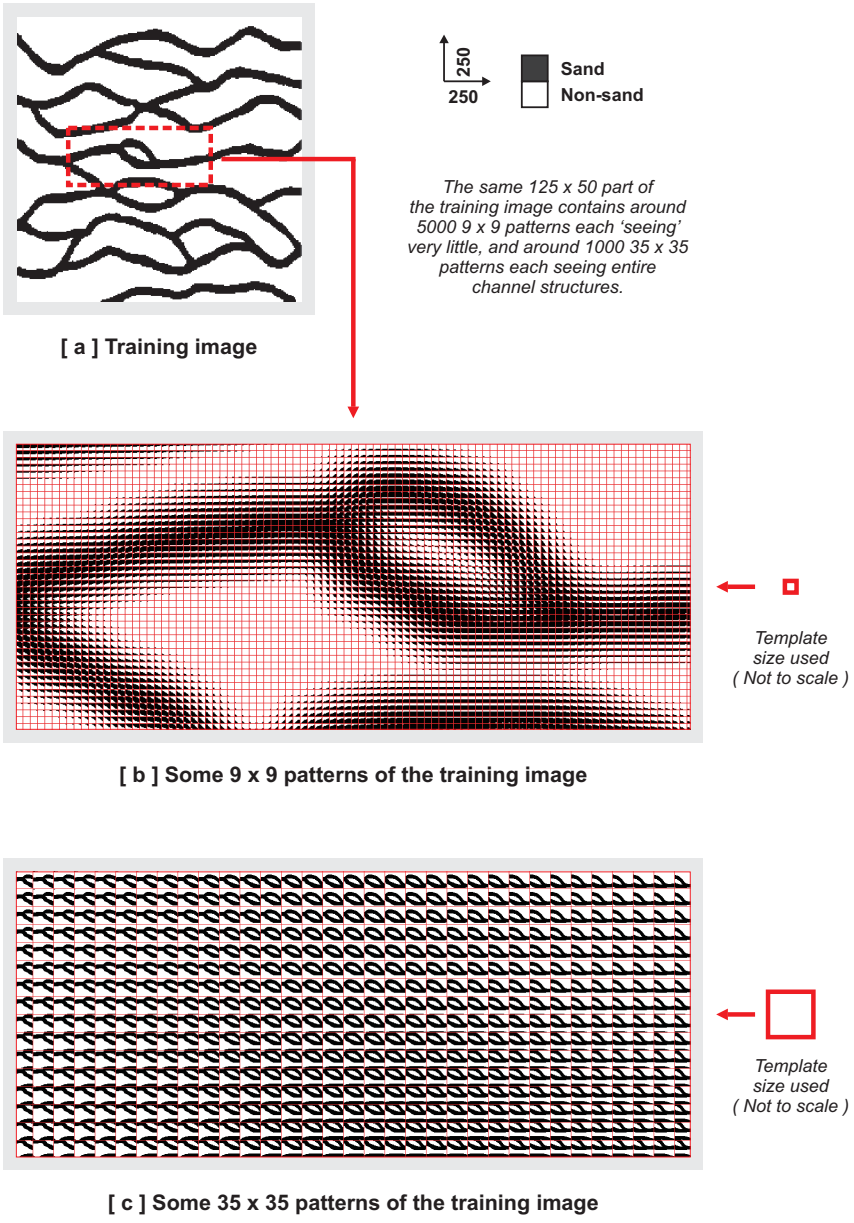


Figure 3.8: Some 9×9 and 35×35 patterns of the training image used for the realizations of Figure 3.7.

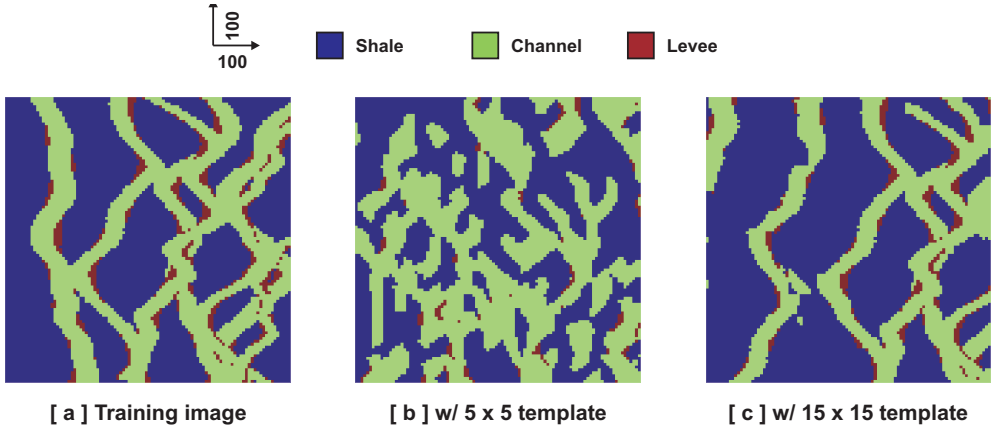


Figure 3.9: The single-grid algorithm can be applied to multiple-category training images without any change. For this example, similar to Figure 3.7, the large 15×15 template gives better results but runs very slowly.

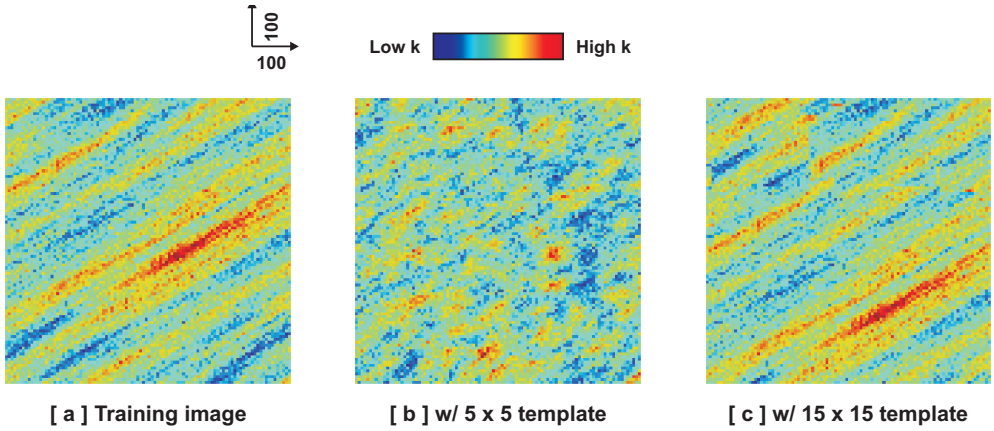


Figure 3.10: The single-grid algorithm can also be applied to continuous training images. In this case, the training image is obtained through an unconditional `SGSIM` realization. Similar to previous figures of the section, the large 15×15 template successfully captures the large-scale patterns of the training image (the long range of the variogram); however, runs much slower than the 5×5 template run.

Chapter 4

Multiple-grid, Unconditional Algorithm

The single-grid algorithm presented in the previous chapter works well if one can afford to use a very large template (See Section 3.3), i.e. if the CPU time spent to run the algorithm is not a factor. However, speed is one of the most critical aspects of any reservoir modeling endeavor. For this reason, one needs to approximate the required large template using less CPU-bound techniques. The most popular and well-established method to achieve such an approximation is the multiple-grid approach (Tran, 1994) as reviewed in Section 2.4. This chapter describes a multiple-grid algorithm inspired by the work of Tran, yet introduces several new ideas to improve multiple-grid simulation.

4.1 Simple Multiple-grid Simulation

The multiple-grid view of a grid $\mathbf{G}$ is defined by n_g cascading grids $\mathbf{G}^g$ and corresponding templates $\mathbf{T}^g$. Figure 2.5, repeated here as Figure 4.1, illustrates this. The coarse template $\mathbf{T}^g$ used for the grid $\mathbf{G}^g$ is defined by the vectors $\mathbf{h}_\alpha^g = 2^{g-1}.\mathbf{h}_\alpha$, i.e. the number of nodes in the template $\mathbf{T}^g$ is same as in the original $\mathbf{T}$ ($n_{\mathbf{T}} = n_{\mathbf{T}^g}$) but the nodes are 'expanded' with 2^{g-1} spacing. Due to the expansion logic, $\mathbf{u} \in \mathbf{G}^{g-1}$ if $\mathbf{u} \in \mathbf{G}^g$, i.e. coarser nodes are always included in all finer grids.

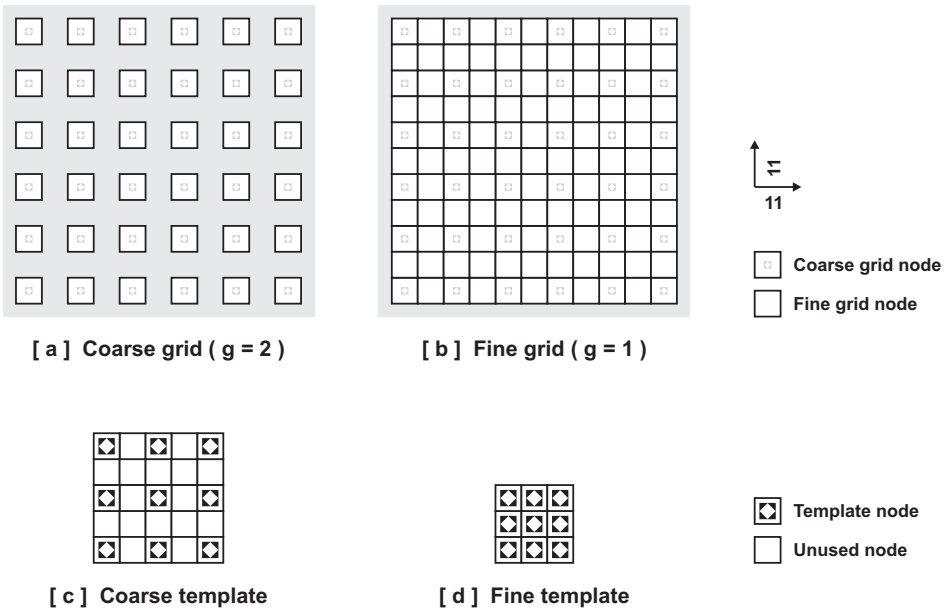


Figure 4.1: Illustration of the multiple-grid concepts for two multiple grids ($n_g = 2$).

When referring to coarse patterns extracted from a training image **ti** using the coarse template $\mathbf{T}^g$, the notation $\mathbf{pat}^k_{\mathbf{T}^g}$ is used. The preprocessing of the training image to extract such coarse patterns and to store them in a pattern database is described by Algorithm 3.1 of Section 3.1 with the only difference being the use of template $\mathbf{T}^g$ instead of $\mathbf{T}$. Note that, the coarse template $\mathbf{T}^g$ scans the fine grid $\mathbf{G}_{ti}$ to capture all coarse patterns of the training image, not only those patterns occurring on the coarse grid $\mathbf{G}^g_{ti}$. Figure 4.2 shows such coarse patterns extracted from the training image shown in Figure 3.1 for two multiple-grids.

Similar to the $\mathbf{pat}^k_{\mathbf{T}^g}$ notation, a coarse data event is denoted by $\mathbf{dev}_{\mathbf{T}^g}(\mathbf{u})$. The similarity measure remains the same (Section 3.2.2) since calculating similarity is a task independent from the underlying grid.

Application of the simple multiple-grid method starts with construction of the coarsest pattern database using the template $\mathbf{T}^{g=n_g}$. Then, the single-grid algorithm (Algorithm 3.2) is applied to the coarsest realization grid $\mathbf{G}^{g=n_g}_{re}$ using the coarsest template $\mathbf{T}^{g=n_g}$. Once that coarsest realization is fully informed, g is set to $g-1$

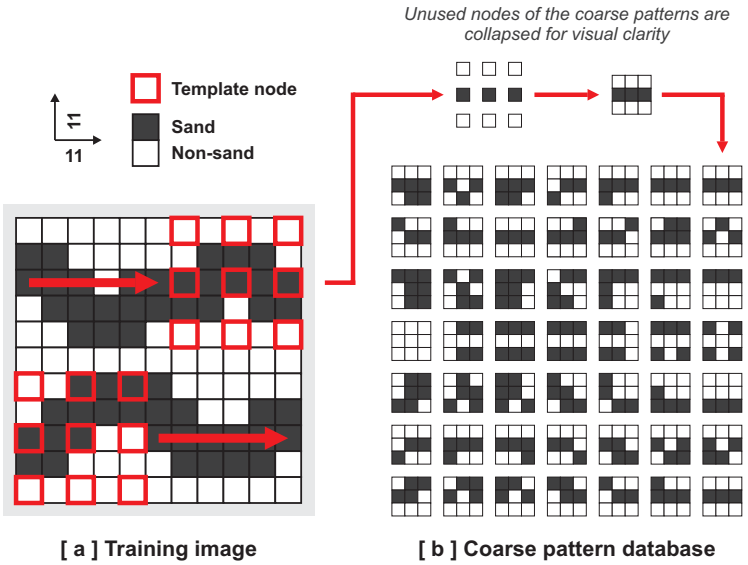


Figure 4.2: Coarse patterns of the training image shown in Figure 3.1. Patterns are extracted using template $\mathbf{T}^2$ of an original 3×3 template $\mathbf{T}$, i.e. the coarse template for the second multiple-grid, $g = 2$ (but scanning is performed on the fine grid $\mathbf{G}_{ti}$).

and the process is repeated until the finest grid $\mathbf{G}_{re}^{g=1}$ simulation is complete. In essence, the simple multiple-grid method is nothing but the successive application of the single-grid algorithm to increasingly finer grids such that, with the exception of the coarsest grid, all other simulations start with an already partially populated realization. Algorithm 4.1 describes this process.

The premise behind the simple multiple-grid method is that, the values simulated on a coarse grid $\mathbf{G}_{re}^{g}$ are to contribute to the similarity calculations of the next finer grid $\mathbf{G}_{re}^{g-1}$ resulting in a finer realization with large scale connectivity. In other words, when attempting to find the most similar pattern $\mathbf{pat}_{\mathbf{T}^g}^{*}$ for the grid $\mathbf{G}_{re}^{g}$, the calculations include the previously simulated values $re(\mathbf{u})$ of $\mathbf{re}$ with $\mathbf{u} \in \mathbf{G}_{re}^{g+1}$. Such values are assumed to 'contain' the large scale information and are to act as soft constraints on the finer grid simulation.

Algorithm 4.1 is essentially a modified form of the traditional multiple-grid approach (Tran, 1994). In the traditional approach, the simulation algorithm that runs

Algorithm 4.1: Simple, multiple-grid, unconditional simulation

1. Set $g = n_g$, i.e. the first simulation is to be performed on the coarsest grid $\mathbf{G}_{re}^{g=n_g}$ (which is initially completely uninformed).
2. Using the coarse template $\mathbf{T}^g$ on the finest grid $\mathbf{G}_{ti}$ of the training image **ti**, construct the coarse pattern database (Algorithm 3.1).
3. Using the coarse template $\mathbf{T}^g$, run a single-grid simulation on the coarse grid $\mathbf{G}_{re}^g$ of the realization **re** (Algorithm 3.2).
4. Set $g = g - 1$ and repeat from Step 2 until $g = 0$, i.e. until the finest grid $\mathbf{G}_{re}^1 = \mathbf{G}_{re}$ simulation is complete.

on grid $\mathbf{G}_{re}^g$ is not allowed to modify the values $re(\mathbf{u})$ of the realization **re** where $\mathbf{u} \in \mathbf{G}_{re}^{g+1}$. In other words, values simulated on coarser grids are frozen during the finer grid simulations. The method described in this section removes this restriction, i.e. any value $re(\mathbf{u})$ of **re** can be 'updated' at any time during the entire course of the multiple-grid simulation. Another difference between Algorithm 4.1 and the traditional approach is how the nodes of **re** are visited. During the simulation of grid $\mathbf{G}_{re}^g$, the traditional approach does not visit the nodes $\mathbf{u} \in \mathbf{G}_{re}^g$ if $\mathbf{u}$ is also in $\mathbf{G}_{re}^{g+1}$, i.e. if $\mathbf{u} \in \mathbf{G}_{re}^{g+1}$. In other words, the traditional approach does not revisit the nodes simulated on the coarser grid during a finer grid simulation. Algorithm 4.1, on the other hand, visits all nodes of $\mathbf{G}_{re}^g$ including those nodes $\mathbf{u} \in \mathbf{G}_{re}^{g+1}$ as such nodes are allowed to be modified during the finer grid simulation.

Figure 4.3 illustrates the application of Algorithm 4.1 when applied to a 11×11 realization using the training image of Figure 3.1 in a two multiple-grid setting (with a fine template of size 3×3). The coarse pattern database used during the simulation is given in Figure 4.2 and the fine pattern database in Figure 3.3. In the figure, the coarse realization (Figure 4.3c) is obtained using the template $\mathbf{T}^2$ (previously shown in Figure 4.1c). Then, using the fine template $\mathbf{T}^1 = \mathbf{T}$, the fine simulation is performed to reach the final realization of Figure 4.3e.

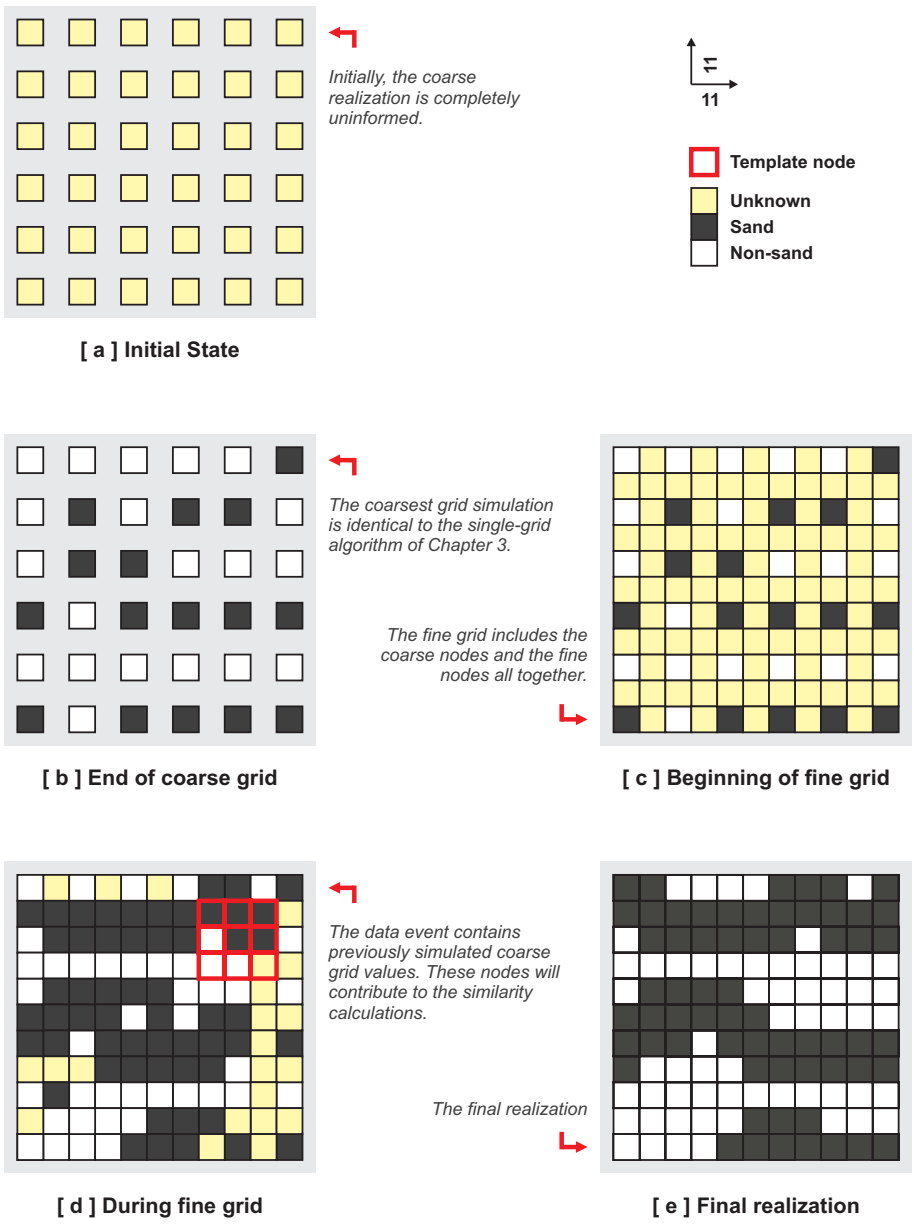


Figure 4.3: Application of Algorithm 4.1 to a 11×11 realization using the training image of Figure 3.1 and a 3×3 fine template in a two multiple-grid setting.

4.2 Dual Template Simulation

The restriction of freezing coarse grid values on finer grids, as dictated by the traditional multiple-grid approach, is removed by Algorithm 4.1. However, the level of approximation used in the algorithm is still the same as the traditional method (see Section 2.4) since a coarse grid $\mathbf{G}_{re}^{g}$ simulation still does not inform the nodes $\mathbf{u}$ of the finer grid $\mathbf{G}_{re}^{g-1}$ other than for $\mathbf{u} \in \mathbf{G}_{re}^{g}$. Hence, this modified multiple-grid approach still imposes a strong hierarchy from coarser grids to finer grids. In this section, Algorithm 4.1 is further modified to alleviate this problem.

4.2.1 Dual Templates

To further improve multiple-grid simulation, the concept of "dual templates" is introduced. A dual template is defined as the template $\widetilde{\mathbf{T}}^{g}$ with the same convex (bounding) volume as $\mathbf{T}^{g}$ on $\mathbf{G}^{g}$ but with all the finest nodes of the finest grid $\mathbf{G}$. $\widetilde{\mathbf{T}}^{g}$ is considered "dual" with respect to the "primal" template $\mathbf{T}^{g}$. In essence, the dual template $\widetilde{\mathbf{T}}^{g}$ is actually the full and dense template that the multiple-grid simulation approximates via the primal template $\mathbf{T}^{g}$. As a result, the total number of nodes within a dual template $\widetilde{\mathbf{T}}^{g}$ (denoted by $n_{\widetilde{\mathbf{T}}^{g}}$) is always greater than the number of nodes $n_{\mathbf{T}^{g}}$ within the primal template $\mathbf{T}^{g}$, i.e. $n_{\mathbf{T}^{g}} << n_{\widetilde{\mathbf{T}}^{g}}$.

By definition, a dual template $\widetilde{\mathbf{T}}^{g}$ always shares the center node $\mathbf{h}_1^{g} = \mathbf{h}_1 = 0$ with its corresponding primal template $\mathbf{T}^{g}$. The dual of a pattern $\mathbf{pat}_{\mathbf{T}^{g}}^{k}$, denoted as $\mathbf{pat}_{\widetilde{\mathbf{T}}^{g}}^{k}$ is defined as the pattern extracted at the same location $\mathbf{u}$ in the training image $\mathbf{ti}$ as $\mathbf{pat}_{\mathbf{T}^{g}}^{k}$ but using the template $\widetilde{\mathbf{T}}^{g}$ instead of $\mathbf{T}^{g}$, i.e. $\mathbf{pat}_{\widetilde{\mathbf{T}}^{g}}^{k} = \mathbf{ti}_{\widetilde{\mathbf{T}}_{g}}(\mathbf{u})$ whenever $\mathbf{pat}_{\mathbf{T}^{g}}^{k} = \mathbf{ti}_{\mathbf{T}_{g}}(\mathbf{u})$ for the same location $\mathbf{u}$. Figure 4.4 shows two primal and the corresponding dual patterns extracted from the training image shown in Figure 3.1 for the coarse grid $g = 2$.

To extract the primal and dual patterns jointly, the training image preprocessing algorithm is modified such that the pattern database $\mathbf{patdb}_{\mathbf{T}}$ now stores two patterns per index k, i.e. the primal pattern $\mathbf{pat}_{\mathbf{T}^{g}}^{k}$ and the dual pattern $\mathbf{pat}_{\widetilde{\mathbf{T}}^{g}}^{k}$ are stored together as a pair for the same k index. The database still does not store the location $\mathbf{u} \in \mathbf{G}_{ti}$ of a primal-dual pattern pair.

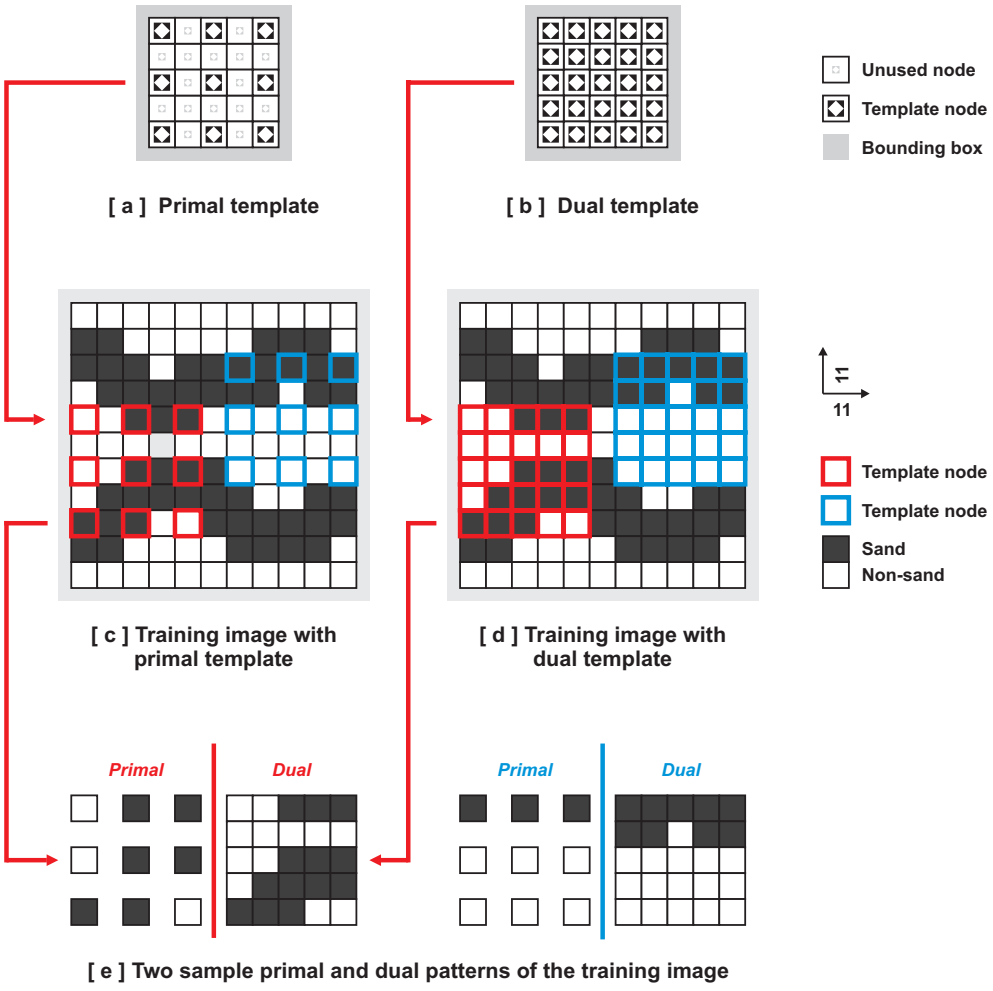


Figure 4.4: Sample primal and dual patterns extracted from the training image of Figure 3.1 using the primal template $\mathbf{T}^2$ and the dual template $\widetilde{\mathbf{T}}^2$.

Last, similar to the $\mathbf{pat}^k_{\mathbf{T}^g}$ notation, a dual data event captured using a dual template $\widetilde{\mathbf{T}}^g$ is denoted by $\mathbf{dev}_{\widetilde{\mathbf{T}}^g}(\mathbf{u})$.

4.2.2 Simulation Algorithm (Using Dual Templates)

Using dual templates, Step 4 of Algorithm 3.2 is now modified. At node $\mathbf{u}$, first, the most similar pattern $\mathbf{pat}^*_{\mathbf{T}^g}$ is found based on maximization of the similarity measure $s\left\langle \mathbf{dev}_{\mathbf{T}^g}(\mathbf{u}), \mathbf{pat}^k_{\mathbf{T}^g} \right\rangle$. Then, the corresponding dual pattern $\mathbf{pat}^*_{\widetilde{\mathbf{T}}^g}$ is retrieved from the pattern database. Finally, instead of populating the nodes of $\mathbf{dev}_{\mathbf{T}^g}(\mathbf{u})$, the algorithm populates all the nodes of the dual data event $\mathbf{dev}_{\widetilde{\mathbf{T}}^g}(\mathbf{u})$, i.e. not only the coarse nodes but also the finest nodes of $\mathbf{re}$ are updated. Since the finest grid $\mathbf{G}_{re}$ includes all nodes of the realization $\mathbf{re}$, updating $\mathbf{dev}_{\widetilde{\mathbf{T}}^g}(\mathbf{u})$ amounts to updating $\mathbf{re}$ at all grids finer than the current coarse grid. In essence, the multiple-grid simulation algorithm described in the previous section remains the same with the exception of updating of $\mathbf{dev}_{\widetilde{\mathbf{T}}^g}(\mathbf{u})$ using $\mathbf{pat}^*_{\widetilde{\mathbf{T}}^g}$ instead of updating of $\mathbf{dev}_{\mathbf{T}^g}(\mathbf{u})$ using $\mathbf{pat}^*_{\mathbf{T}^g}$.

In the above process, dual templates are used only for populating the nodes of the dual data event; they are not used for similarity calculations, i.e. for selecting the most similar pattern $\mathbf{pat}^*_{\mathbf{T}^g}$. The similarity calculations are still performed on the original, primal template $\mathbf{T}^g$ using the primal data event $\mathbf{dev}_{\mathbf{T}^g}(\mathbf{u})$. Thus, use of dual templates has no effect on the simulation of the current coarse grid since the finer nodes simulated by the dual pattern $\mathbf{pat}^*_{\widetilde{\mathbf{T}}^g}$ are never taken into account during the similarity calculations of the current grid $\mathbf{G}^g_{re}$. As a result, the CPU demand of the algorithm remains the same when using dual templates since that CPU demand is determined primarily by the size of the template used and that size ($= n_{\mathbf{T}^g}$) is the same for both the simple multiple-grid simulation of the previous section and the new dual template simulation.

The effects of using dual templates will only be seen when simulating the next finer grid $\mathbf{G}^{g-1}_{re}$ since that simulation now starts with a fully informed realization $\mathbf{re}$ including the nodes $\mathbf{u} \in \mathbf{G}^{g-1}_{re}$ instead of a partially informed realization where only nodes $\mathbf{u} \in \mathbf{G}^g_{re}$ are informed. In other words, once the coarsest grid realization $\mathbf{G}^{n_g}_{re}$ is complete, further multiple-grid simulations are performed only to 'correct' the finer

details of the realization. After the coarsest grid simulation, the realization **re** no longer contains any uninformed nodes $re(\mathbf{u}) = \chi$ and further, finer simulations only update the values of already informed nodes so that finer details of **ti** are honored.

The difference between the traditional multiple-grid approach (Tran, 1994) and the dual template simulation is how the information is passed from a coarse grid $\mathbf{G}_{re}^{g}$ to a finer grid $\mathbf{G}_{re}^{g-1}$. In the traditional method, the finer grid $\mathbf{G}_{re}^{g-1}$ simulation starts with a partially informed realization and simulates the missing values. In dual template simulation, the algorithm guesses the entire, finest realization **re** based on the coarse information provided by the coarsest grid $\mathbf{G}_{re}^{n_g}$ simulation and then lets finer grid simulations progressively correct this guess. Figure 4.5 illustrates this process.

4.3 2D Examples and Sensitivity Study

In Section 3.3, it was shown that using a 9×9 template on the binary training image of Figure 3.7 is not sufficient to capture the large-scale patterns of the training image; one needs a larger template. Figure 4.7 shows the application of the dual template simulation to the same example. In that figure, using 4 multiple-grids, the algorithm successfully reproduces the large-scale patterns of the training image. Furthermore, the total run-time is only 25% longer than the single-grid 9×9 template simulation.

Given the result of Figure 4.7, one might argue that, it may be possible to decrease the template size further since the dual template simulation seems to adequately capture the desired large-scale patterns. Figures 4.8 and 4.9 test this argument by applying the dual template simulation to the same training image using 3×3, 5×5 and 7×7 templates in addition to the 9×9 template used previously. The figure shows that the dual template simulation fails to compensate for very small templates (Figure 4.8b). It appears, for a given training image, there is a minimum template size required to reproduce the multiple-scale patterns of the image. For the training image shown in Figure 4.7, this minimum size seems to be approximately 7×7 (Figure 4.9b). Currently, there is no established method for analyzing a training image ahead of time to find the minimum template size. Chapter 8 discusses this issue and proposes a method to determine a template size suitable for a given training image.

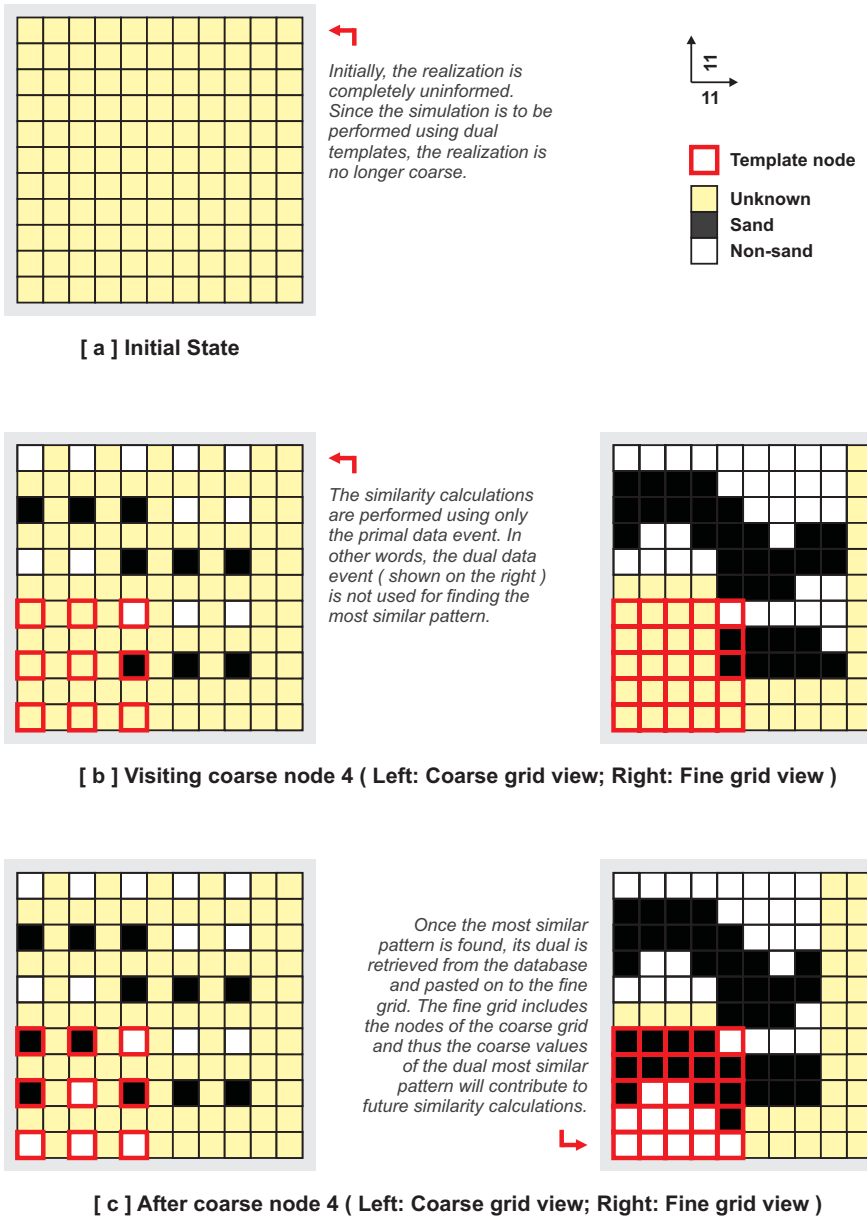
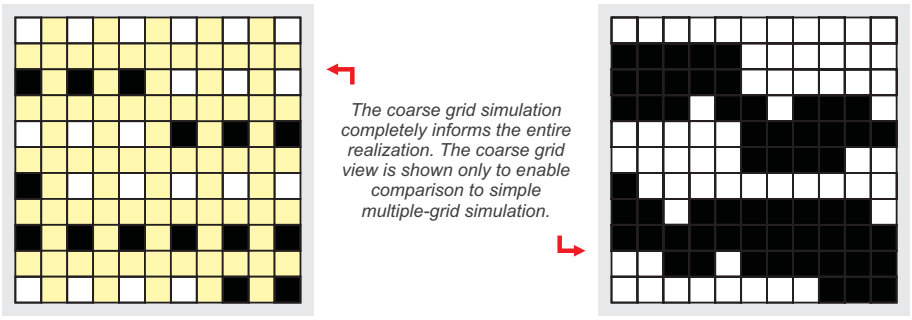
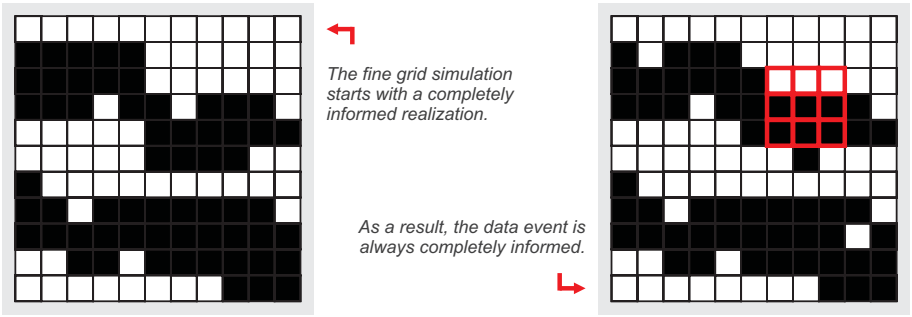


Figure 4.5: Application of dual template simulation to a 11×11 realization using the training image of Figure 3.1 and a 3×3 fine template in a two multiple-grid setting. The figure continues on the next page as Figure 4.6

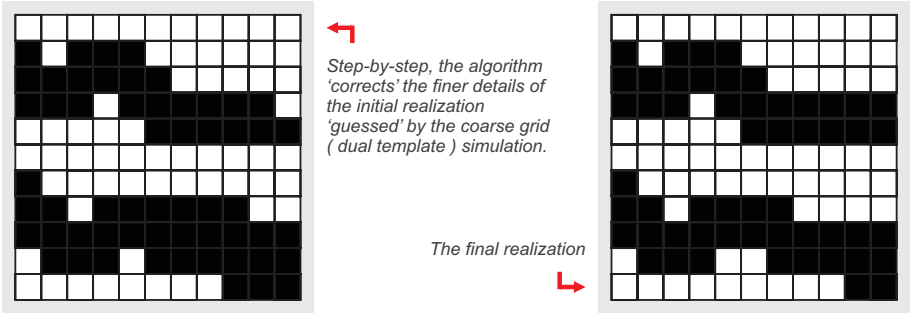


[d] End of coarse grid (Left: Coarse grid view; Right: Fine grid view)



[e] Beginning of fine grid

[f] During fine grid



[g] During fine grid

[h] Realization

Figure 4.6: Continuation of Figure 4.5 from the previous page. Application of dual template simulation to a 11×11 realization using the training image of Figure 3.1 and a 3×3 fine template in a two multiple-grid setting.

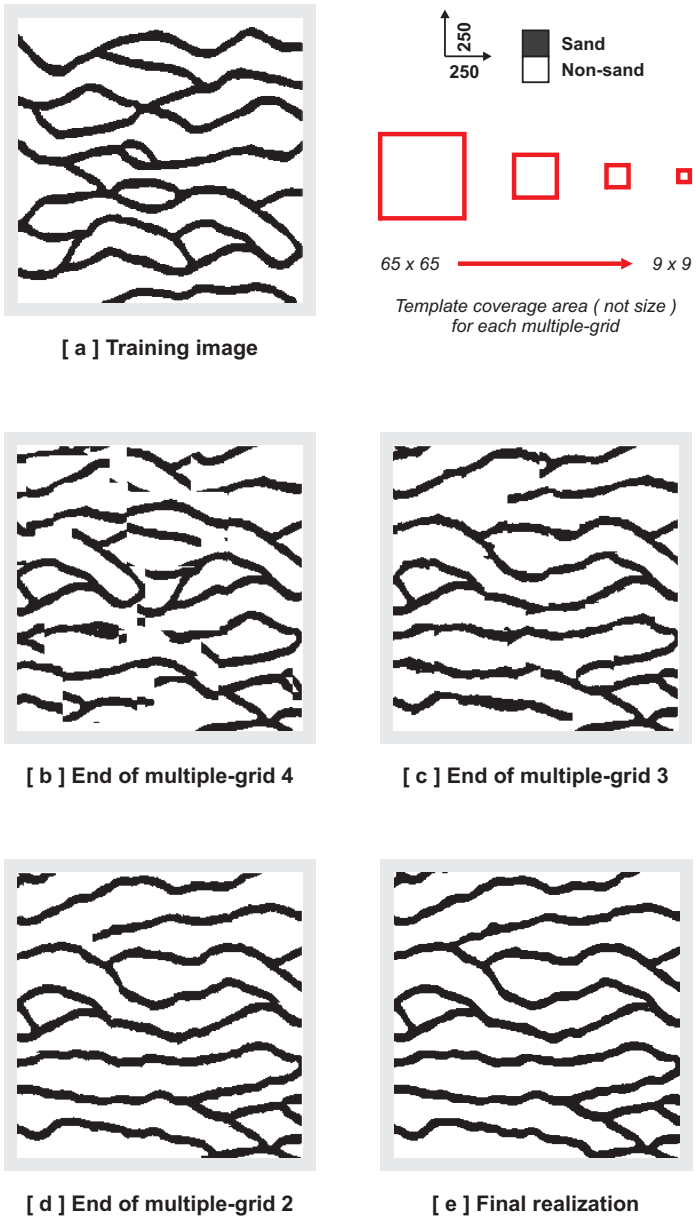


Figure 4.7: Application of the dual template simulation algorithm using a 9×9 base (finest) template. The progress is shown on the finest (final) grid only since dual template simulation informs all grids after the coarsest grid simulation, including the finest grid.

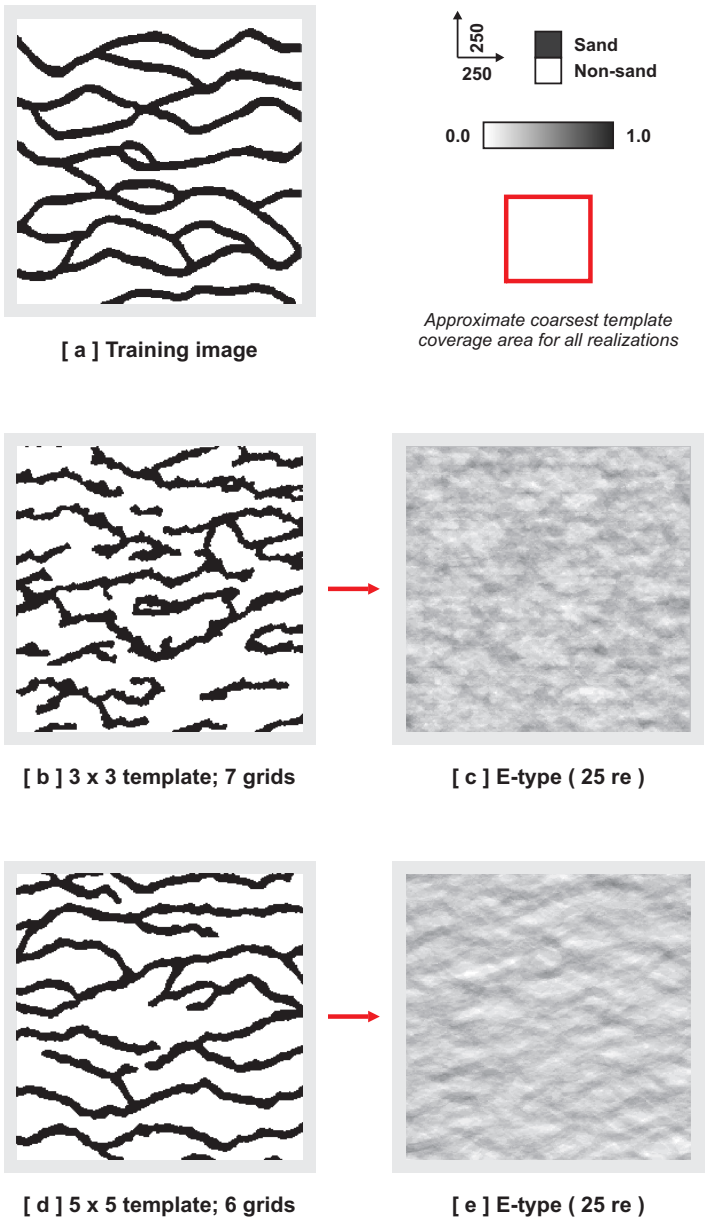


Figure 4.8: Dual template simulation cannot compensate for very small templates [b]. However, relatively small templates can still give reasonable results [d].

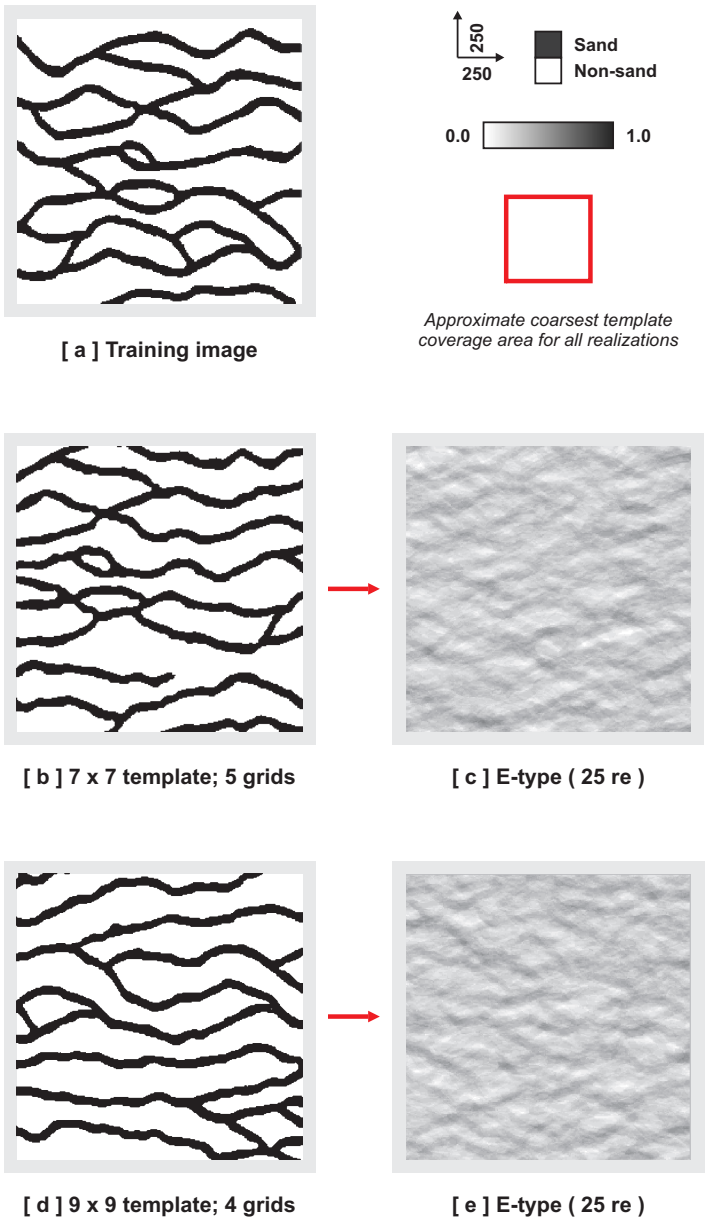


Figure 4.9: For the training image shown, the 'optimum' template size seems to be approximately 7×7 (Compare with Figure 4.8).

4.4 A Note on Uncertainty

The issue of template size discussed in the previous section also has an effect on the definition of the space of uncertainty, i.e. on the variability between different realizations. Figures 4.8, 4.9 and 4.10 show E-type estimates of sets of 25 realizations, each obtained with an increasing template size. It appears that the E-type estimate becomes increasingly more structured (less uncertainty) as the template size increases. Indeed, in the limit when the template size is equal to the training image size, all stochasticity disappears (since there is only a single pattern to choose from).

That the template size is an important parameter in defining the uncertainty space is not a surprise. Any algorithm-dependent random function, including the proposed algorithm, is defined by its simulation parameters. Therefore, it is important to understand the effects of these parameters for a given training image, such as that of the template size on the space of uncertainty. Since the proposed algorithm aims to exactly reproduce training image patterns in the final realization and since these patterns are determined by the template sized used, the template size is arguably the most influential parameter of the algorithm and is the primary factor that determines the overall characteristics of the generated realizations, including the uncertainty space defined by those realizations.

However, it should be remembered that, the choice of the training image itself (and possible alternatives) is much more consequential to the uncertainty space than the choice of the template size (or, for that matter, any other parameter of the algorithm). The uncertainty generated from multiple realizations with one fixed training image often does not provide a realistic uncertainty regardless of the simulation parameters used by the algorithm. Indeed, the main aim of the `SIMPAT` algorithm is to reproduce realistic geologic patterns constrained to data, not to assess uncertainty. Therefore, multiple realizations generated by `SIMPAT` should never be taken as the sole measure of uncertainty. Instead, one should always generate several alternative training images, possibly based on different sedimentological and structural scenarios (Caumon and Journel, 2004) and work with realizations generated from these training images.

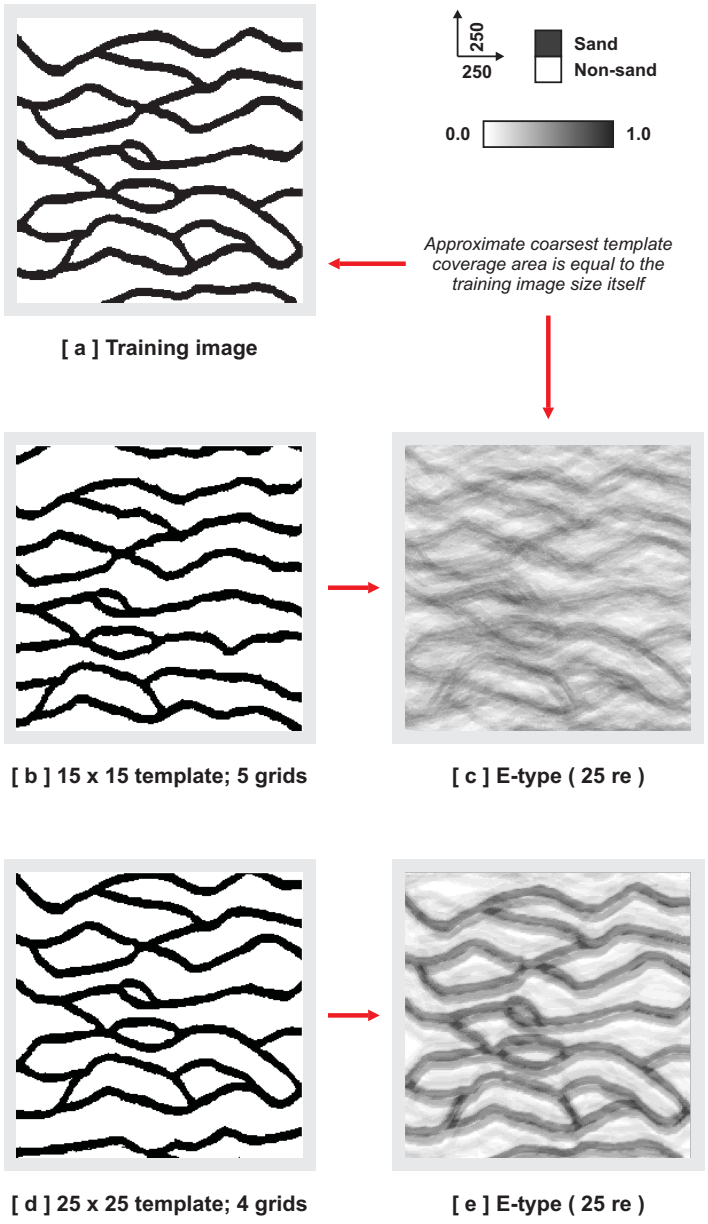
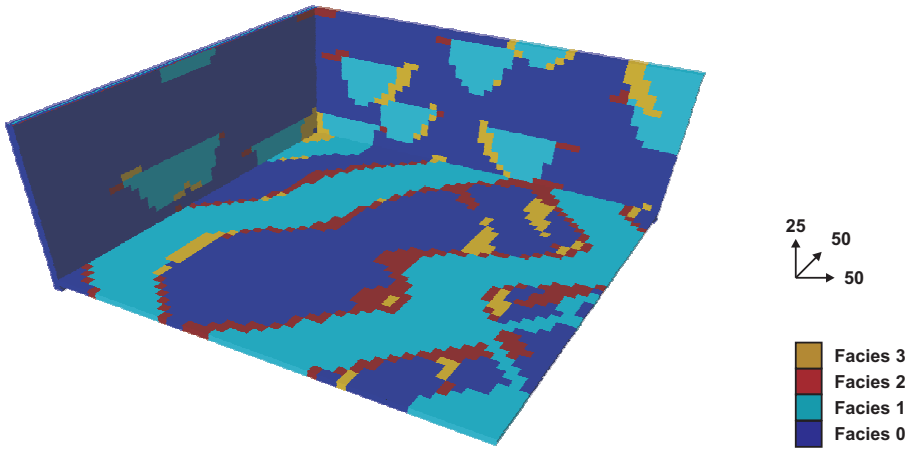


Figure 4.10: With increasing template size and coarse template coverage (i.e. increasing number of multiple-grids), the E-types start to show structure which signals a decrease in the stochasticity.

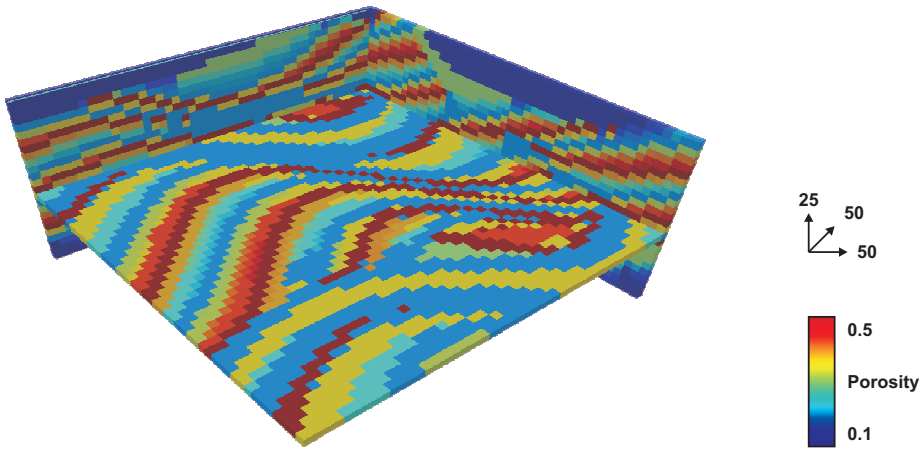
4.5 3D Examples

Since the dual template simulation allows use of smaller templates while still successfully reproducing large-scale patterns, it now becomes possible to use large, 3D training images for simulation. Figure 4.11 shows a 3D multiple-category and a 3D continuous training images. The multiple-category training image is obtained via an unconditional object-based simulation and the continuous training image is obtained via an unconditional processed-based simulation (Wen et al., 1998).

Figures 4.12 and 4.13 show application of the dual template simulation to the training images of Figure 4.11. In each figure, two realizations are shown; one obtained using a $7 \times 7 \times 3$ template and 4 multiple-grids and the other obtained using a $9 \times 9 \times 3$ template and 3 multiple-grids. The figures demonstrate that the dual template simulation can adequately model complex, 3D geological scenarios. In all simulations the Manhattan similarity measure of Equation 3.6 is used.



[a] Multiple-category training image



[b] Continuous training image

Figure 4.11: A 3D multiple-category training image and a 3D continuous training image. The following figures show application of dual template simulation on these training images.

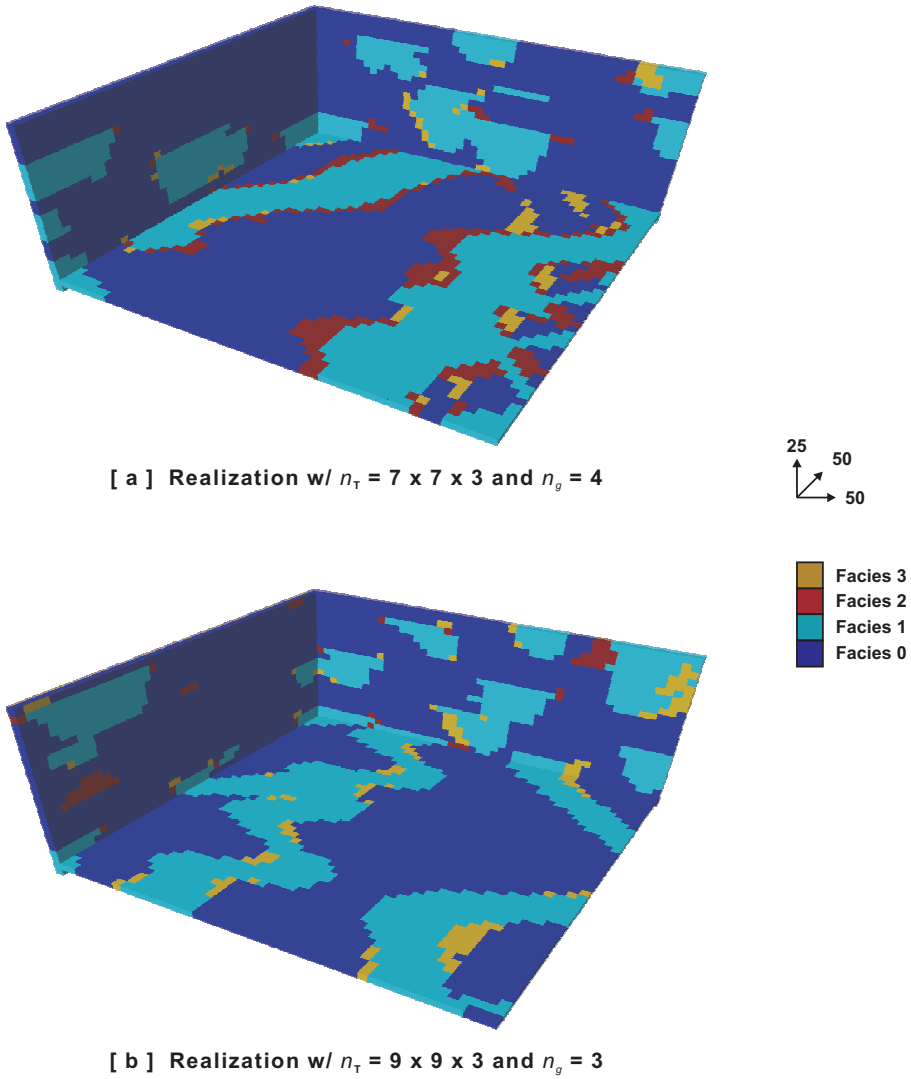


Figure 4.12: Application of the dual template simulation algorithm to the multiple-category training image of Figure 4.11a.

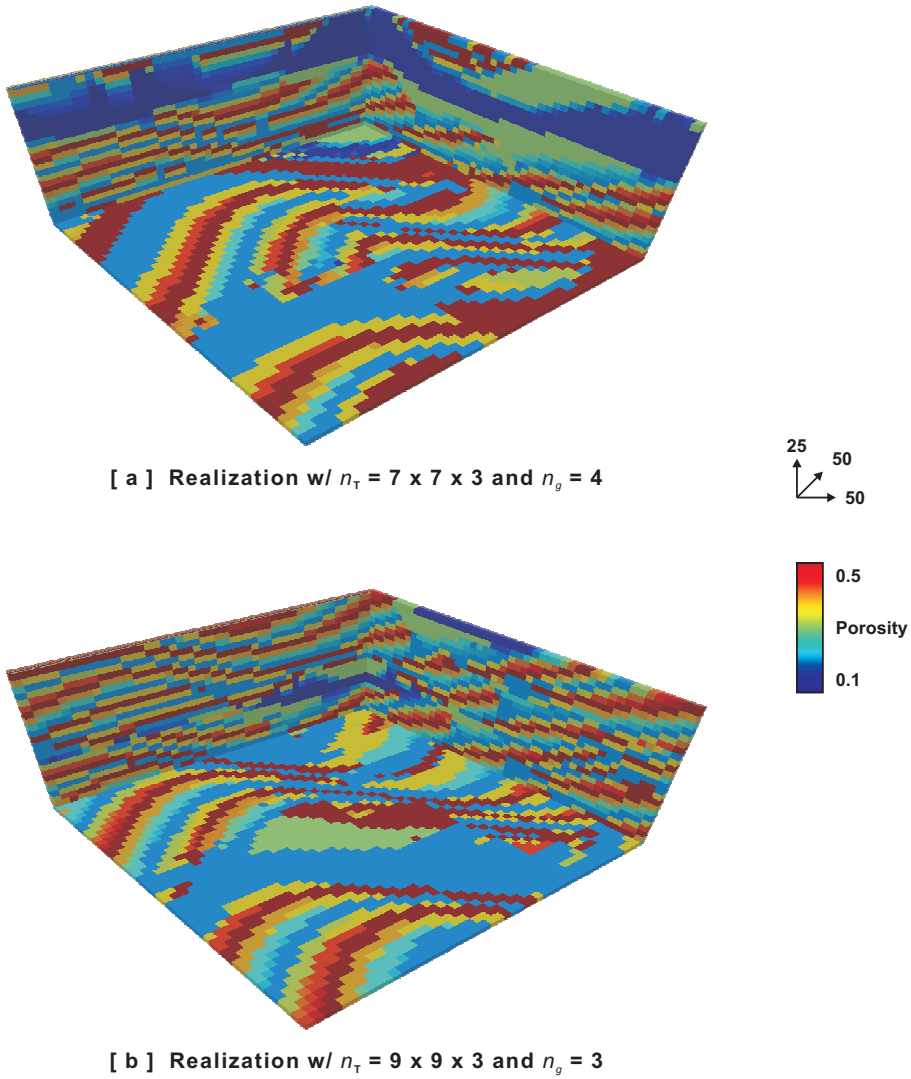


Figure 4.13: Application of the dual template simulation algorithm to the continuous training image of Figure 4.11b.

Chapter 5

Similarity Measures

Consider the 5×5 binary data event and the nine patterns shown in Figure 5.1. Assume that these patterns were extracted from a training image depicting slanted fractures (denoted by black pixels). Which of the nine candidate patterns shown on the right is most similar to the data event shown on the left?

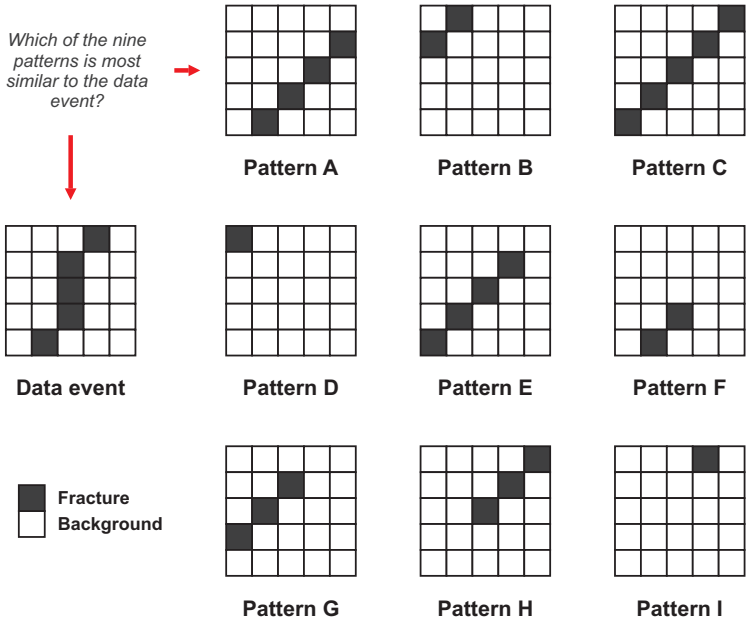


Figure 5.1: A 5×5 binary data event and nine candidate similar patterns.

5.1 Limitations of the Manhattan Distance

When asked the question in Figure 5.1, human experts (geologists, geophysicists, geostatisticians and petroleum engineers) typically select patterns A, C or E. However, in this case, the Manhattan distance is radically different from that used implicitly by a human expert. When applied to the data event and the patterns of Figure 5.1, Manhattan distance gives the results shown in Figure 5.2 with smaller numbers indicate greater similarity. The two most similar patterns selected by the Manhattan distance (marked in red) are counter-intuitive to human experts' choice (marked in blue) and furthermore geologically unrealistic. Whereas the original data event can be assumed to dictate a fracture with a slightly different slant, the most similar pattern selected by the Manhattan distance depicts an artificially disconnected fracture piece.

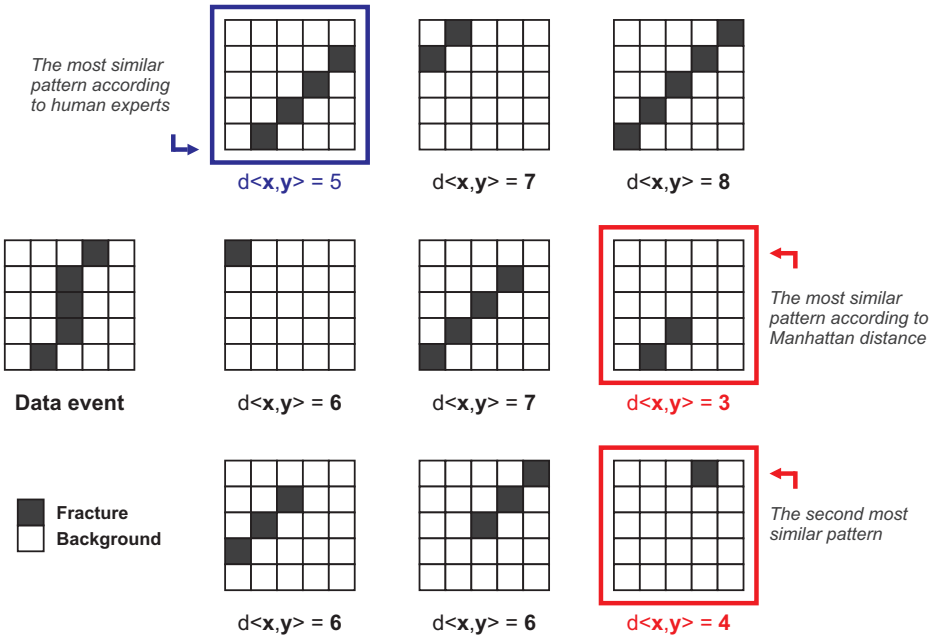


Figure 5.2: Results for the Manhattan distance when applied to the data event and candidate patterns of Figure 5.1. $d\langle \mathbf{x}, \mathbf{y} \rangle$ denotes the Manhattan distance of each candidate pattern to the data event when an indicator representation is used.

Figure 5.3 demonstrates the problem when simulating, using the Manhattan distance, a 250×250 binary fracture training image using a 9×9 template and 5 multiple-grids. In the figure, the length of fractures as depicted in the training image is not reproduced in the realization. During simulation, at every node $\mathbf{u}$, the Manhattan distance consistently selects patterns with less fracture content as it 'should', resulting in a severe reduction in the global fracture proportion (from 0.03 to 0.01) and in disconnected fracture pieces.

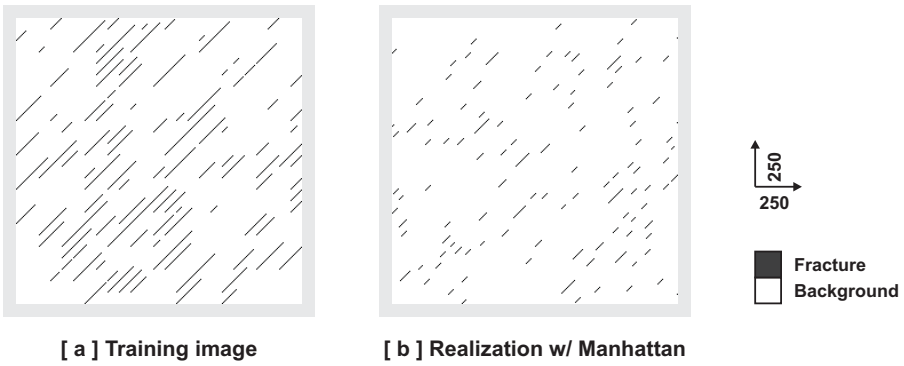


Figure 5.3: Manhattan distance consistently selects patterns with less fracture content as most similar patterns resulting in a severe reduction in the global fracture proportion and disconnected fracture pieces [$n_{\mathbf{T}} = 9 \times 9$; $n_g = 5$].

The choice of fractures to demonstrate the limitations of the Manhattan distance is not incidental. A fracture training image, such as the one shown in Figure 5.3 exhibits a critical property that forces the Manhattan distance to evaluate similarities differently from a human expert. In general, whenever the global proportion of a particular category is significantly less than that of any other category in a training image, the Manhattan distance is likely to 'prefer' patterns containing this low proportion category. In Figure 5.3, the global fracture proportion is only 3% and thus the problem becomes critical. During the simulation, when a data event $\mathbf{dev_T}(\mathbf{u})$ matches poorly to the patterns $\mathbf{pat}^k_{\mathbf{T}}$ available in the pattern database $\mathbf{patdb_T}$, the Manhattan distance settles for a most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$ that has the highest number of matching nodes to the data event $\mathbf{dev_T}(\mathbf{u})$. The problem is that this $\mathbf{pat}^*_{\mathbf{T}}$ is

likely not to contain the low proportion fracture category on the mismatching nodes, resulting in an artificial bias toward the category with higher proportion (in this case, the background category).

This problem is not specific to categorical variables and may also occur for continuous training images if the values of the underlying continuous variable exhibit a behavior similar to the above scenario. For example, in the case of a training image depicting permeabilities of a channel system with thin channels, the range of the background permeabilities (low) is typically different than the range of the channel permeabilities (high) and such low permeability values are dominant, resulting in the same problem.

5.2 The Need for a Better Similarity Measure

One possible solution to the problem described in the previous section is to utilize a specific mechanism to enforce the global proportions of the training image categories on to the realization. Indeed, probability-based MPS algorithms such as `SNESIM` typically use a "servo-system" (Strebelle, 2002) to achieve this. During a simulation, the servo-system tracks the current, simulated global proportion of each category, i.e. the proportions of all previously simulated values up until the currently visited node $\mathbf{u}$; then, adjusts the conditional probability at $\mathbf{u}$ depending on the difference between the current proportion and the target proportion.

Global control mechanisms such as a servo-system work well in practice (Hoffman and Caers, 2004). However, within the context of similarity, using such a global control mechanism only indirectly addresses the fundamental problem. Global proportion reproduction should be a consequence of correctly reproducing all higher order statistics of a training image. In other words, since first order statistics are intrinsically linked to the patterns of a training image at all scales, correctly reproducing such patterns will automatically reproduce all statistics of the training image, first and higher orders. For this reason, poor reproduction of global proportions should not be considered as the main problem but instead as a signal to the more fundamental

problem of not reproducing the training image patterns. Indeed, the problem of Section 5.1 occurs because, for many nodes $\mathbf{u}$ of a realization $\mathbf{re}$, the algorithm fails to find a sufficiently similar pattern $\mathbf{pat}^*_{\mathbf{T}}$ to the data event $\mathbf{dev}_{\mathbf{T}}(\mathbf{u})$ and settles for a pattern that is actually quite dissimilar.

One might argue that the problem is only due to limited number of available patterns. This essentially amounts to using a 'poor' training image with too few patterns (In Figure 5.3, fractures with only a single slant). Using a 'richer' training image where the data event itself would occur ought to solve the problem. While the premise of this argument is correct, in practice, it is unreasonable to require rich training images due to two main reasons:

1. Generating such a rich training image, where all possible patterns are present, may not be possible or would be extremely costly;

2. Even were a rich training image available, processing such a large image is likely to be very CPU demanding.

Thus, it appears that the only practical solution is to consider a different similarity measure; one that better mimics the human experts' similarity decision.

5.3 Single-point Dissimilarity Measures

The Manhattan distance actually belongs to a family of single-point distance functions known as the "Minkowski metric" (Duda et al., 2001). For a data event and a pattern, the most general form of this metric is defined as:

$$d\left\langle \mathbf{dev}_{\mathbf{T}}(\mathbf{u}), \mathbf{pat}^k_{\mathbf{T}} \right\rangle = \left(\sum_{\alpha=0}^{n_{\mathbf{T}}} \left| dev_{\mathbf{T}}(\mathbf{u}+\mathbf{h}_\alpha) - pat^k_{\mathbf{T}}(\mathbf{h}_\alpha) \right|^q \right)^{1/q} \tag{5.1}$$

where $q \geq 1$ is an input parameter. Setting $q = 2$ gives the familiar Euclidean distance (L2-norm), $q = 1$ gives the Manhattan distance and $q = \infty$ gives the Chessboard distance (L-∞ norm). Other q parameters are rarely used in practice (Duda et al., 2001). Unknown nodes of a data event are handled similar to Equation 3.7, i.e. whenever $dev_{\mathbf{T}}(\mathbf{u}+\mathbf{h}_\alpha) = \chi$, the node $\mathbf{h}_a$ is skipped during summation.

When $q = \infty$ (the Chessboard distance), Equation 5.1 reduces to,

$$d\left\langle \mathbf{x}, \mathbf{y} \right\rangle = max\left(d_\alpha\left\langle x_\alpha, y_\alpha \right\rangle\right) \tag{5.2}$$

i.e. to the difference $d_\alpha\left\langle x_\alpha, y_\alpha \right\rangle$ with the largest magnitude for $\alpha = 1, \ldots, n_{\mathbf{T}}$. For this reason, the Chessboard distance is ineffective for most applications.

On the other hand, the Euclidean distance ($q = 2$) is commonly used in the fields of computer vision and image processing (Gonzalez and Woods, 2002) and sometimes preferred over Manhattan distance despite the fact that it is computationally more expensive to calculate. However, when used in the context of reservoir modeling, the Euclidean distance brings little benefit over the Manhattan distance. Figure 5.4 shows the results of using the Euclidean distance when applied to the binary fracture training image of Figure 5.3; the same random seed is used to generate the random path. The Euclidean distance gives results identical to those obtained with the Manhattan distance since both distance functions are used in a comparative context, i.e. they simply measure whether a node of a data event $dev_{\mathbf{T}}(\mathbf{u} + \mathbf{h}_\alpha)$ matches the corresponding node of a pattern $pat^k_{\mathbf{T}}(\mathbf{h}_a)$ or not. Thus, Euclidean distance is not a solution to the issues of Section 5.1.

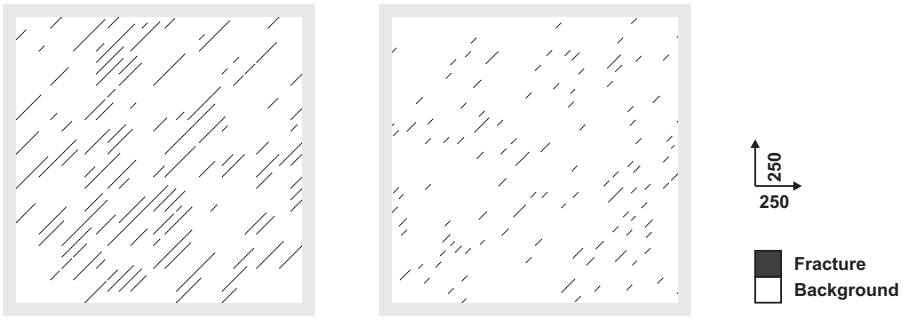


Figure 5.4: The Euclidean distance gives the same result as the Manhattan distance (Figure 5.3) since both distance functions are used only in a comparative context.

The computer vision and image processing literature list alternative single-point distance measures (Duda et al., 2001) different from the Minkowski metric. Yet, none

of these methods have enjoyed the popularity of Euclidean and Manhattan distances simply because these alternatives typically offer little improvement. The reason is that all single-point measures are bounded by the limited information provided by the co-located nodes comparison. For this reason, image construction algorithms that operate on complex images generally work with similarity measures utilizing multiple-point information.

5.4 Proximity Transforms

In all rigor, a multiple-point similarity calculation would involve comparing all higher order statistics between any two patterns. Clearly, such an approach is too CPU demanding especially for large $n_{\mathbf{T}}$ and/or $n_{\mathbf{pat}_{\mathbf{T}}}$ when performed in a brute-force manner. To avoid the problem, similarity measures that compare only selected and/or approximate multiple-point statistics have been proposed (Rubner et al., 1998; Hagedoorn, 2000; Duda et al., 2001). However, such measures are still several orders of magnitude more CPU demanding than the Manhattan distance.

An alternative approach to using a multiple-point similarity function is to calculate several multiple-point properties of an image ahead of time (sometimes called the "features" of an image; Duda et al. (2001)) and then to use the Manhattan distance on these properties, i.e. instead of calculating the direct multiple-point similarity, an indirect approach is taken. One method to calculate such multiple-point properties is through proximity transforms.

A proximity transform is simply a transformation applied to a training image **ti** such that, after the transformation, each node $\mathbf{u}$ of the image holds additional information about the values of the neighboring nodes $\mathbf{u} + \mathbf{h}_\alpha$. In other words, individual node values $ti(\mathbf{u})$ now contain multiple-point information. Once the transformation is complete, the simulation proceeds as usual but uses the transformed training image rather than the original. Apart from the use of a transformed training image, the simulation algorithm remains the same.

Since proximity transforms are closely related to a well-known image processing tool called "distance transforms", this section first describes the distance transform

concept. Then, the use of proximity transforms within the context of the simulation algorithm described in the previous chapters is explained.

5.4.1 Distance Transforms

By definition, a distance transform applies only to binary images. Within the context of a distance transform, the 'black' nodes ($ti(\mathbf{u}) = 1$ when using the binary indicator equation 3.1) of the binary image are said to belong to an 'object'. For example, in Figure 5.5, the channel is the object of the training image. Since the image is binary, the method does not differentiate between multiple objects (for example, multiple channels) and considers them a single, disjoint object.

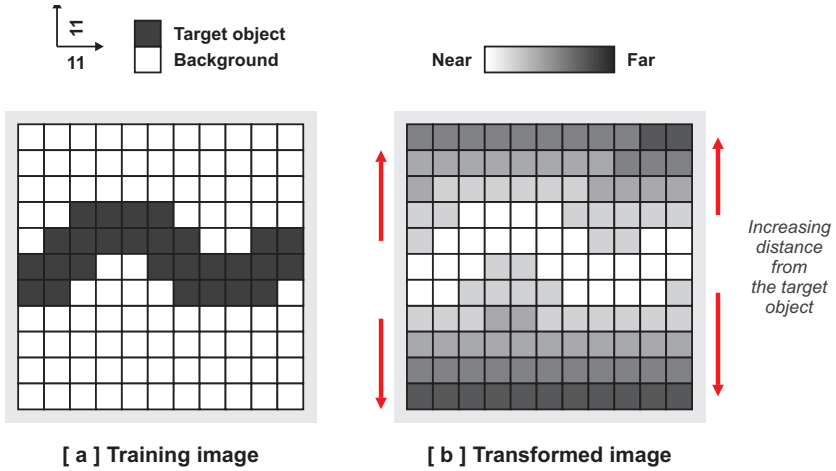


Figure 5.5: Illustration of distance transforms.

When applying a distance transform to a training image **ti**, one calculates the proximity of each node of the image **ti** to the target object. In other words, after the transformation, each non-black node ($ti(\mathbf{u}) = 0$) of the image **ti** is mapped into a continuous, scalar value denoting how 'distant' the node is to the target object. The result of this transformation, sometimes called a "distance map", is another image $\mathbf{DT}(\mathbf{ti})$ such that the values in this transformed image are equal to zero for the nodes defining the target object of the original image ($ti(\mathbf{u}) = 1$) and increasingly greater (more distant) for further away nodes. Figure 5.5 illustrates this concept. In the

figure, gray scale values denote increasingly distant nodes with white indicating zero, i.e. an object (channel) node.

Distance maps are commonly used for image processing tasks such as shape description, feature detection, skeletonization, segmentation, and multiple-scale morphological filtering (Danielsson, 1980; Borgefors, 1986; Gonzalez and Woods, 2002). Due to this popularity, the image processing literature offers several different methods for calculating such maps. One of these methods, the chamfer transformation, is well-established, easy to implement and fast (Borgefors, 1986). For this reason, it is chosen as the primary method to implement distance transforms in this study.

A chamfer transformation is essentially a two-pass algorithm scanning the original image from two opposite corners (Generally, bottom left corner and top right corner). In each pass, the algorithm starts from one corner, and propagates the 'distance' of the underlying node to the target object to other nodes within a template $\mathbf{T}$. This propagation can be envisioned as a type of wave propagation and requires defining the 'distance' between nodes. Incidentally, this distance is typically defined through a Minkowski metric where, this time, the vector entries $\mathbf{x}$ and $\mathbf{y}$ of Equation 5.1 are location vectors $\mathbf{u} = (i, j, k)$. When the Euclidean distance is used, the transformation is generally known as "Euclidean Distance Transform" or $\mathbf{EDT}(\cdot)$.

To be able to apply the chamfer transform, the original binary image $\mathbf{ti}$ has to be first modified such that all locations $\mathbf{u}$ belonging to the target object (denoted by $ti(\mathbf{u}) = 1$) are set to zero reflecting the fact that the distance of these nodes to the target object is exactly zero (nodes belong to the object). The background nodes (denoted by $ti(\mathbf{u}) = 0$) are set to ∞ (infinity), i.e. initially, each background node is assumed to be infinitely distant from the target object. Once this modification is done, the image $\mathbf{ti}$ is ready for the transformation.

Algorithm 5.1 details the first pass of the chamfer transform of a 3D training image $\mathbf{ti}$ (defined on a Cartesian grid $\mathbf{G}_{ti}$) for a generic distance function $d\langle \mathbf{u}_\alpha, \mathbf{u}_\beta \rangle$. The second pass uses the same algorithm and is performed on the resulting intermediate image of the first pass. In the second pass, the algorithm starts from the diagonal corner of the image ($\mathbf{u}_\alpha = (n_i - 1, n_j - 1, n_k - 1)$ where n_i, n_j and n_k denote the size of the image) and decrements $\mathbf{u}_\alpha$ toward the origin. During any pass, if a point $\mathbf{u}_\beta$

falls outside the image (when the template $\mathbf{T}$ is placed near one of the edges of the image), it is value is assumed to be ∞ for the purpose of distance calculations.

Algorithm 5.1: Chamfer transform

1. Start from origin, i.e. $\mathbf{u}_\alpha = (0, 0, 0)$.
2. Place the template $\mathbf{T}$ on the image $\mathbf{ti}$ centered on node $\mathbf{u}_\alpha$.
3. Set $ti(\mathbf{u}_\alpha) = \underset{\mathbf{u}_\beta}{min} \{ti(\mathbf{u}_\beta) + d\langle \mathbf{u}_\alpha, \mathbf{u}_\beta \rangle\}$ where nodes $\mathbf{u}_\beta$ are the nodes within the template $\mathbf{T}$ and $\beta = 1, \ldots, n_{\mathbf{T}}$, i.e. calculate the distance of $\mathbf{u}_\alpha$ to each neighbor node $\mathbf{u}_\beta$ and add to this distance the value of $ti(\mathbf{u}_\beta)$ (which itself is a distance). Set the value of $ti(\mathbf{u}_\alpha)$ to the minimum of these distances.
4. Increment $\mathbf{u}_\alpha$ to the next node such that i-coordinate cycles the fastest and k-coordinate cycles the slowest.
5. Repeat the steps 2 - 4 until all nodes are exhausted.

The accuracy of a chamfer transform, i.e. whether the distance value of a node $\mathbf{u}$ is the actual distance to the nearest object node, depends on the size of the template $\mathbf{T}$. In practice, $\mathbf{T}$ is typically chosen to be $3 \times 3 \times 3$ or $5 \times 5 \times 5$ (Borgefors, 1986). Figure 5.6 shows the application of the Euclidean distance transform $\mathbf{EDT}(\cdot)$ to the fracture training image of Figure 5.3 using a 3×3 template.

5.4.2 Transformation and Simulation of Binary TIs

The result of a distance transformation is a distance map where the values of the map are scalar distances varying depending on the underlying distance function $d\langle \cdot, \cdot \rangle$ and the target object morphology. A proximity transform $\mathbf{PT}(\mathbf{ti})$ is essentially a distance transform $\mathbf{DT}(\mathbf{ti})$ followed by an inverse normalization of the resulting distances such that all nodes of the transformed image range between $[0, 1]$ with 0 indicating the furthest node and 1 indicating a target object node. In other words, after a proximity transformation, the final, transformed image contains nodes $\mathbf{u}$ such that the original object nodes retain their $ti(\mathbf{u}) = 1$ value and the background nodes (originally = 0)

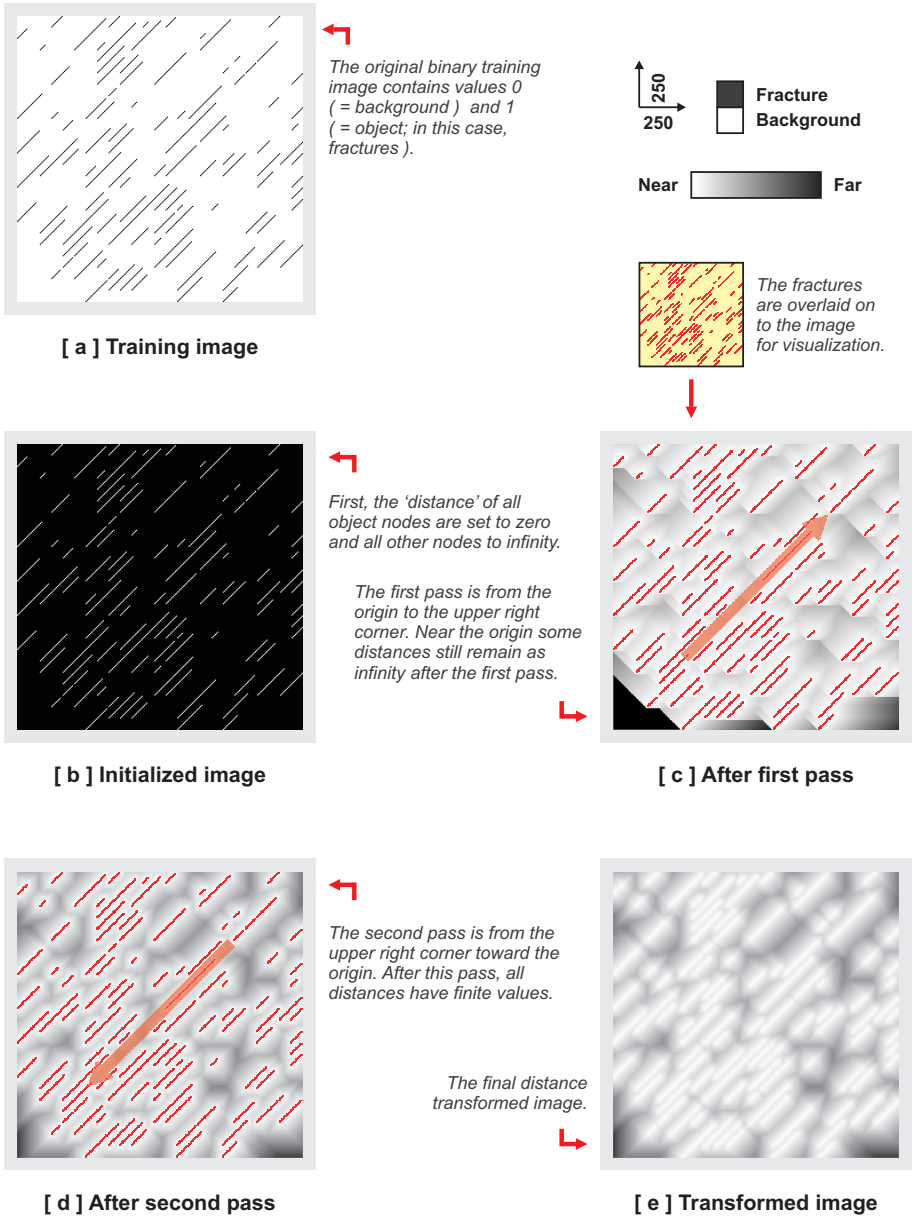


Figure 5.6: Application of the Euclidean distance transform $\mathbf{EDT}(\cdot)$ using a 3×3 template when applied to the training image of Figure 5.3.

now range between $[0, 1)$ with higher numbers indicating adjacency or 'proximity' to the target object. Figure 5.7 shows the Euclidean proximity transform $\mathbf{EPT}(\cdot)$ of the training image of Figure 5.3 using a 3×3 template.

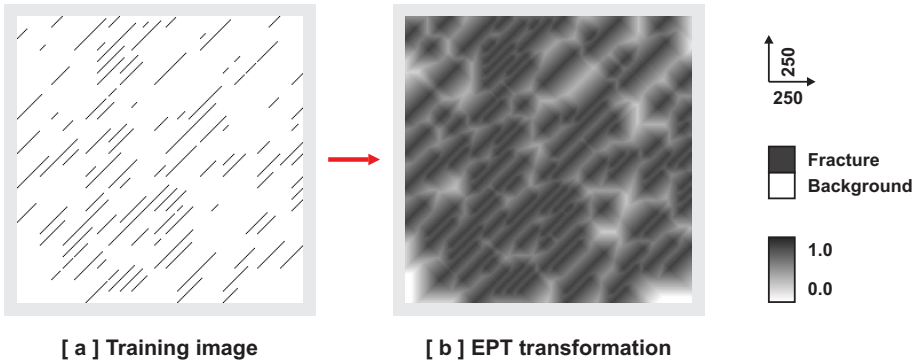


Figure 5.7: Application of proximity transforms to the binary training image previously shown in Figure 5.3. For binary images, the proximity transform $\mathbf{EPT}(\cdot)$ looks like the photographic negative of the distance transform $\mathbf{EDT}(\cdot)$ [$n_{\mathbf{T}} = 3 \times 3$].

Once the transformed training image is obtained, the dual template simulation of Chapter 4 is applied but using $\mathbf{DT}(\mathbf{ti})$ as the training image instead of $\mathbf{ti}$. In other words, similarity calculations are still performed using the Manhattan distance but with the patterns of the transformed training image instead of the original binary training image. The rest of the algorithms remains the same.

Since the transformed training image $\mathbf{DT}(\mathbf{ti})$ is continuous (i.e. its values range between $[0, 1]$), the resulting realization $\mathbf{re}$ is also continuous and thus requires a back-transformation to the original binary space. This back-transformation is trivial: Simply, set nodes $\mathbf{u}$ of the realization $\mathbf{re}$ equal to zero for all $re(\mathbf{u}) \neq 1$, i.e. preserve only the nodes with proximity $= 1$ to the target object since such nodes actually constitute the object itself. Figure 5.8 shows this process for the training image of Figure 5.7. In the figure, all simulation parameters except for using the transformed training image are identical to those of Figure 5.3.

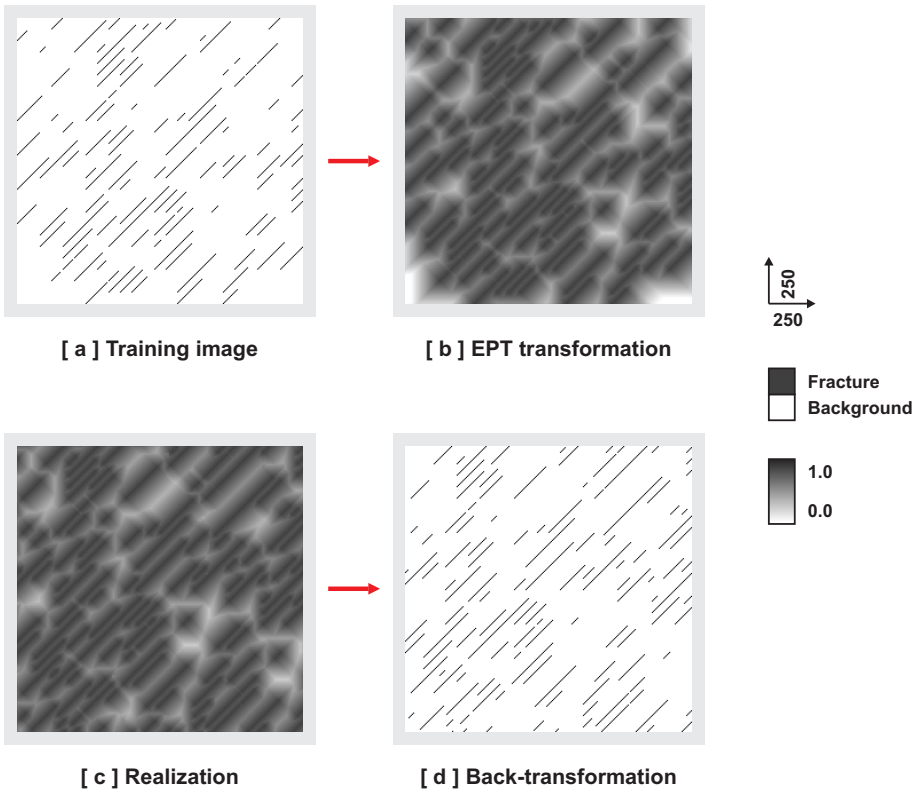


Figure 5.8: The simulation parameters of the above realization are identical to that of Figure 5.3. However, due to the use proximity transforms, the final realization reproduces the desired training image patterns better.

5.4.3 Transformation of Multiple-category TIs

The above proximity transforms are defined only for binary images. Applying proximity transforms to multiple-category training images requires a more generic approach. Before describing the details of this approach, the concept of "training image bands" requires introduction.

Define n_f as the number of categories (facies) in a multiple-category training image **ti** with each category identified by an index f, i.e. $ti(\mathbf{u}) = f$ and $f \in [1, n_f]$. The categorical values $ti(\mathbf{u})$ of the training image **ti** can then be represented by a binary vectorial variable $\overrightarrow{ti}(\mathbf{u})$ such that for a specific category index f, this binary vectorial

variable has entries $ti(\mathbf{u}, f') = 0$ if $f' \neq f$ and 1 otherwise. For example, for a 3 facies training image ($n_f = 3$), if $ti(\mathbf{u}) = 2$, then $\vec{ti}(\mathbf{u}) = \{0, 1, 0\}$, i.e. $ti(\mathbf{u}, 1) = 0$, $ti(\mathbf{u}, 2) = 1$ and $ti(\mathbf{u}, 3) = 0$.

A "training image band" is then defined as the vector $\vec{\mathbf{ti}}(f)$ of all vectorial values $\vec{ti}(\mathbf{u}, f), \forall \mathbf{u} \in \mathbf{G}_{ti}$. In other words, a training image band is simply obtained by separating the image into its categories and taking each category one at a time as a separate binary image. Figure 5.9 illustrates this.

Then, the proximity transform $\mathbf{PT}(\mathbf{ti})$ of a multiple-category training image $\mathbf{ti}$ is simply obtained by applying a proximity transformation separately to each band of the vectorial representation $\vec{\mathbf{ti}}$ of the original training image $\mathbf{ti}$. Figure 5.10 shows this for the training image of Figure 5.9.

5.4.4 Simulation of Multiple-category TIs

Simulation based on a transformed multiple-category training image $\vec{\mathbf{ti}}$ is identical to the regular dual template simulation algorithm of Chapter 4 with the exception of the similarity function used.

A data event $\vec{\mathbf{dev}}_{\mathbf{T}}(\mathbf{u})$ is now a composition of several single data events each belonging to a particular band or category, i.e.,

$$\vec{\mathbf{dev}}_{\mathbf{T}}(\mathbf{u}) = \Big\{\mathbf{dev}_{\mathbf{T}}(\mathbf{u}, 1), \mathbf{dev}_{\mathbf{T}}(\mathbf{u}, 2), \ldots, \mathbf{dev}_{\mathbf{T}}(\mathbf{u}, f), \ldots, \mathbf{dev}_{\mathbf{T}}(\mathbf{u}, n_f)\Big\} \tag{5.3}$$

with,

$$\mathbf{dev}_{\mathbf{T}}(\mathbf{u}, f) = \Big\{dev_{\mathbf{T}}(\mathbf{u} + \mathbf{h}_1, f), \ldots, dev_{\mathbf{T}}(\mathbf{u} + \mathbf{h}_\alpha, f), \ldots, dev_{\mathbf{T}}(\mathbf{u} + \mathbf{h}_{n_{\mathbf{T}}}, f)\Big\} \tag{5.4}$$

A pattern $\vec{\mathbf{pat}}^{\,k}_{\mathbf{T}}$ (scanned from $\mathbf{PT}(\mathbf{ti})$) is defined similarly. Figure 5.11 illustrates this for a 3×3 multiple-band pattern scanned from the training image of Figure 5.10.

The similarity between data events $\vec{\mathbf{dev}}_{\mathbf{T}}(\mathbf{u})$ and patterns $\vec{\mathbf{pat}}^{\,k}_{\mathbf{T}}$ is still defined through the Manhattan distance (3.6), however, since the nodes $\vec{ti}(\mathbf{u})$ of the transformed training image $\vec{\mathbf{ti}}$ are now vectorial values, the Manhattan distance is defined as the sum of all absolute differences of all bands:

$$d\left\langle \mathbf{dev}_{\mathbf{T}}(\mathbf{u}), \mathbf{pat}^k_{\mathbf{T}} \right\rangle = \sum_{\alpha=0}^{n_{\mathbf{T}}} d_\alpha \left\langle \vec{dev}_{\mathbf{T}}(\mathbf{u} + \mathbf{h}_\alpha), \vec{pat}^{\,k}_{\mathbf{T}}(\mathbf{h}_a) \right\rangle \tag{5.5}$$

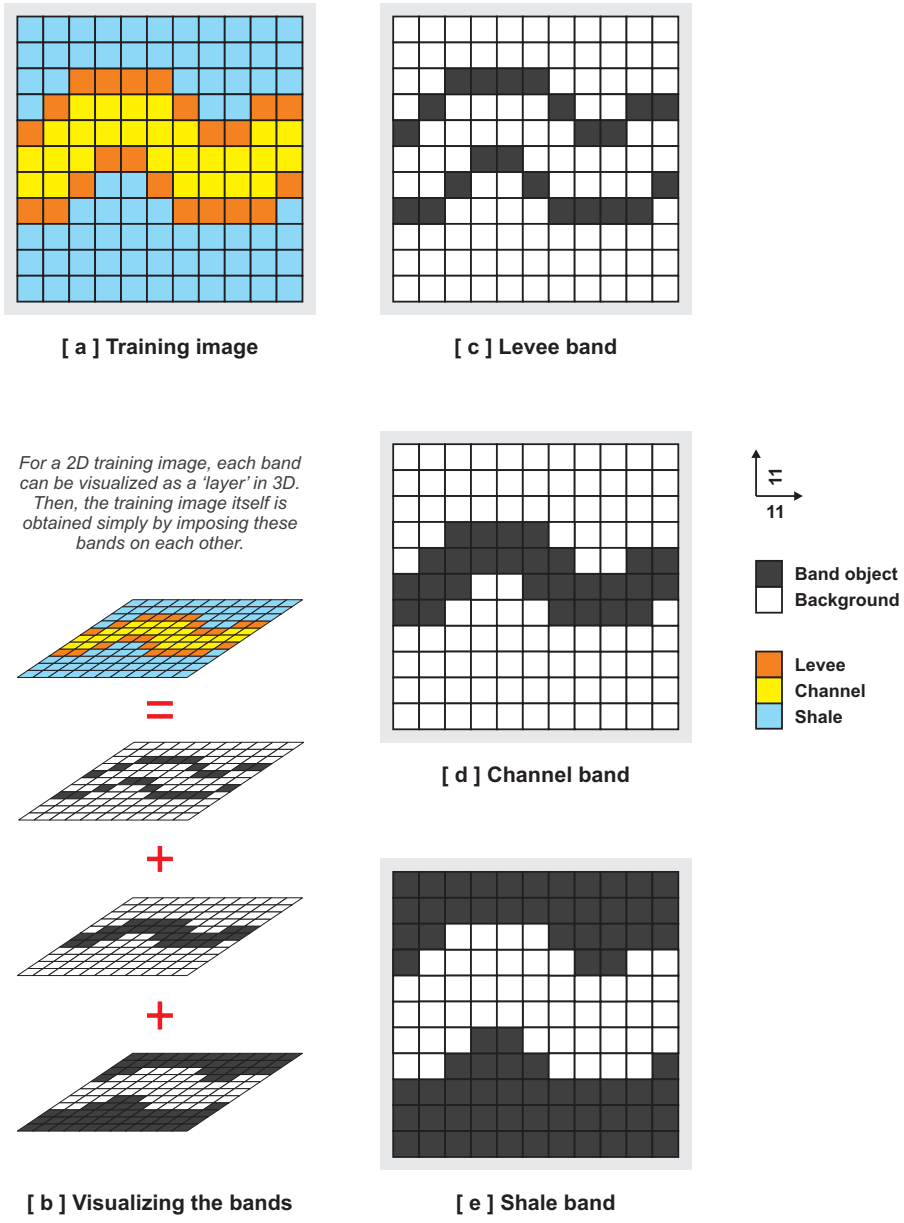


Figure 5.9: A multiple-category training image ($n_f = 3$) and its bands. For 2D images, bands can be visualized as 'layers' in 3D.

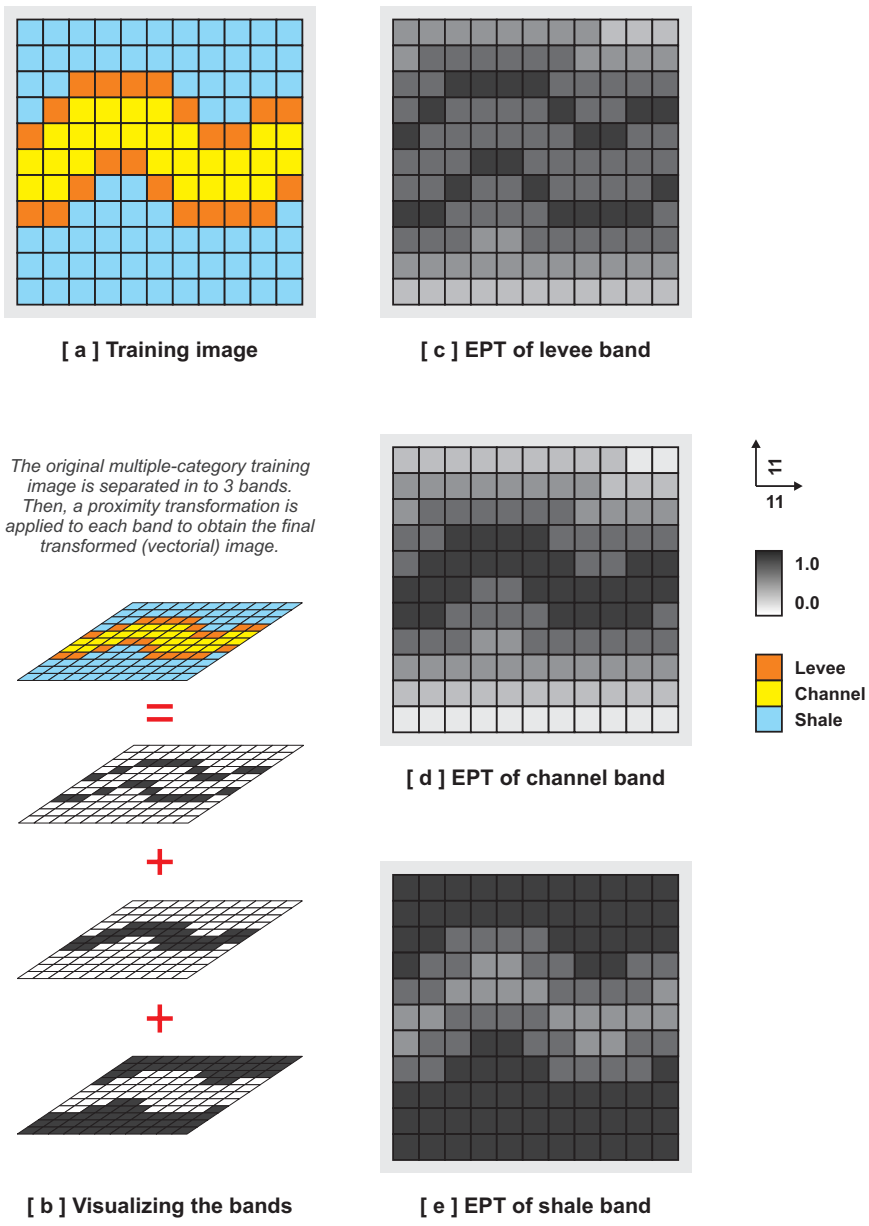


Figure 5.10: Application of the Euclidean proximity transformation to the multiple-category training image of Figure 5.9.

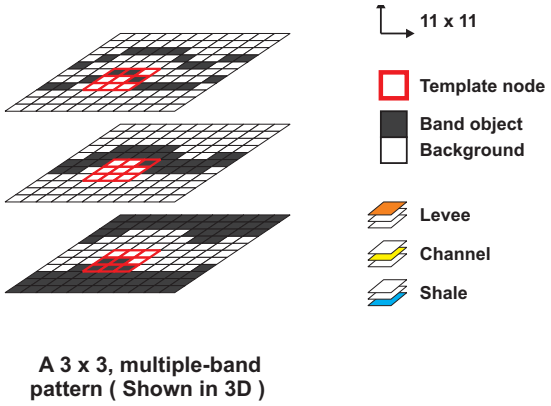


Figure 5.11: A 3×3 pattern scanned from the training image of Figure 5.10.

and,

$$d_\alpha \langle x_\alpha, y_\alpha \rangle = \sum_{i=1}^{n_f} \left| dev_{\mathbf{T}}(\mathbf{u} + \mathbf{h}_\alpha, f_i) - pat^k_{\mathbf{T}}(\mathbf{h}_a, f_i) \right| \tag{5.6}$$

Uninformed values of the data events ($= \chi$) are handled, again, by skipping these unknown nodes during the summation.

Equipped with this new similarity measure, the simulation algorithm proceeds as before. Since the patterns $\overrightarrow{\mathbf{pat}}^k_{\mathbf{T}}$ are now vectorial patterns, the final result of the simulation is also vectorial. Transformation of this result back to the original categorical variable is trivial; namely, every vectorial node $\vec{re}(\mathbf{u})$ has one entry (and only one entry) $re(\mathbf{u}, f)$ such that $re(\mathbf{u}, f) = 1$. Then, the simulated value of the original categorical variable for node $re(\mathbf{u})$ is simply f.

5.5 Examples

This section gives examples of the application of proximity transforms to highly complex, 3D training images. Each example shows first a realization obtained using only the Manhattan distance and then a realization obtained using proximity transforms. For all runs, all other simulation parameters are the same.

Figure 5.12 shows the results for a 3 facies training image. In the figure, the realization obtained using only the Manhattan distance shows a clear bias toward

the background facies. Using proximity transforms, this bias is removed. Figure 5.13 shows the multiple-band training image used to obtain the second realization of Figure 5.12c.

Figure 5.14 shows the results for a 7 facies training image. In the figure, when using Manhattan distance only, the patterns with high yellow facies content gets preferentially selected early on during the coarse grid simulations and thus the final realization contains artificially high proportions of this facies. Using proximity transforms improves the results.

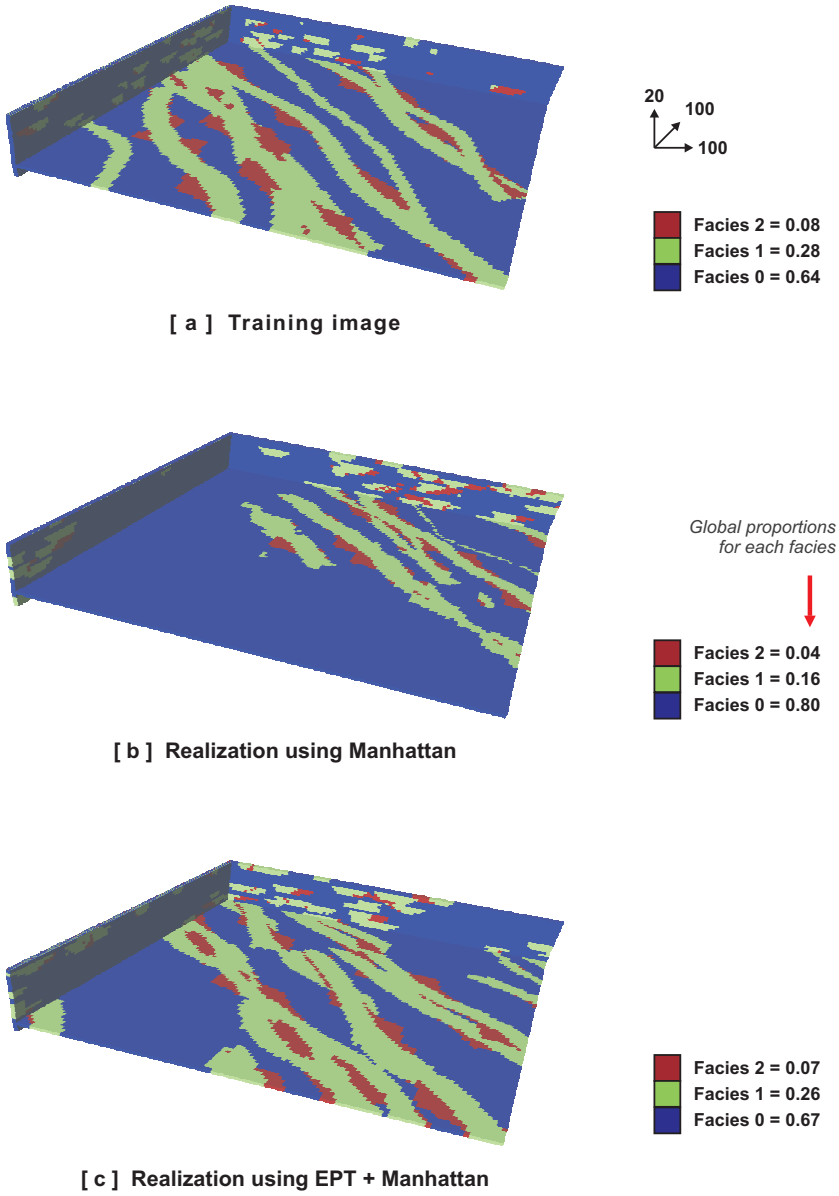


Figure 5.12: Application of unconditional, dual template simulation algorithm to a multiple-category training image using only Manhattan distance and Euclidean proximity transforms $\mathbf{EPT}(\cdot)$ + Manhattan distance.

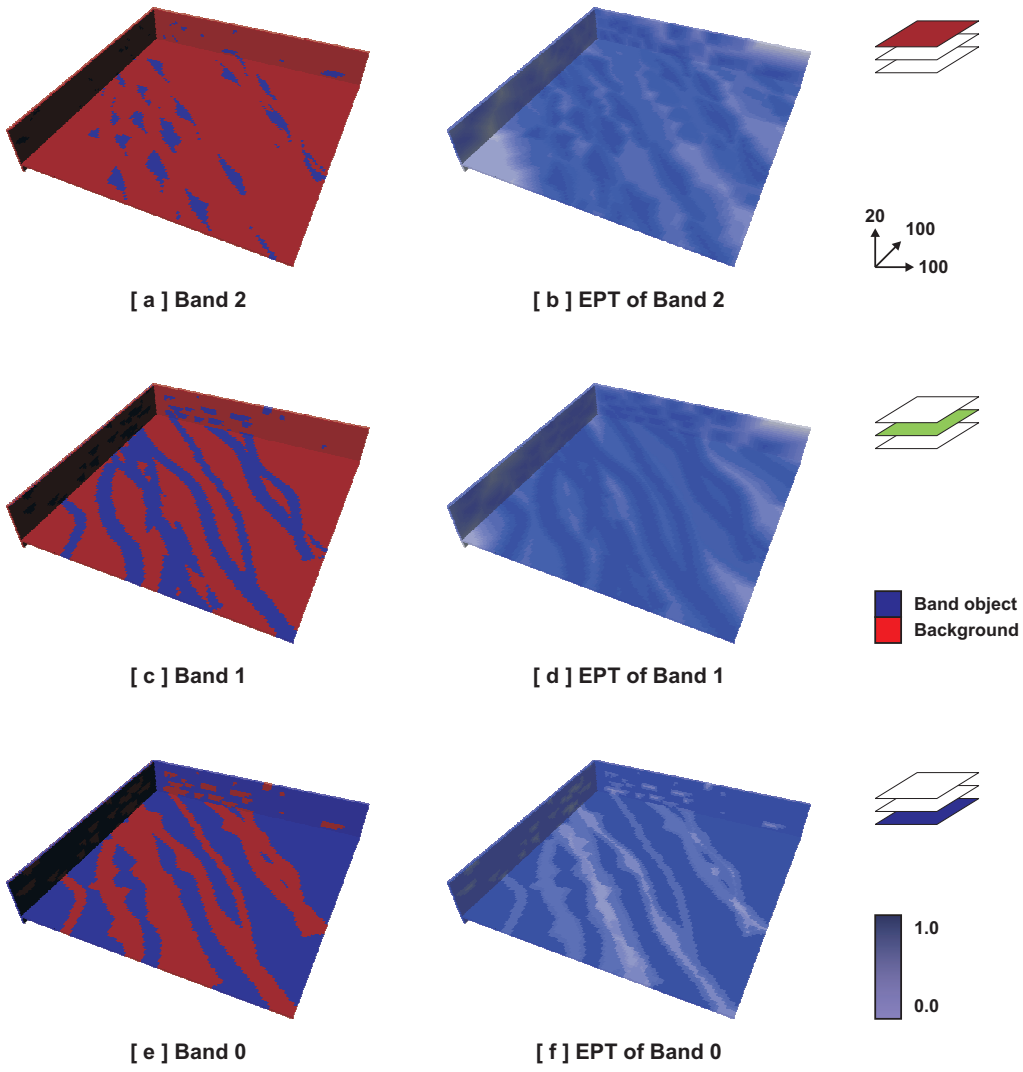


Figure 5.13: Bands of the training image shown in Figure 5.12 and Euclidean proximity transformation of these bands.

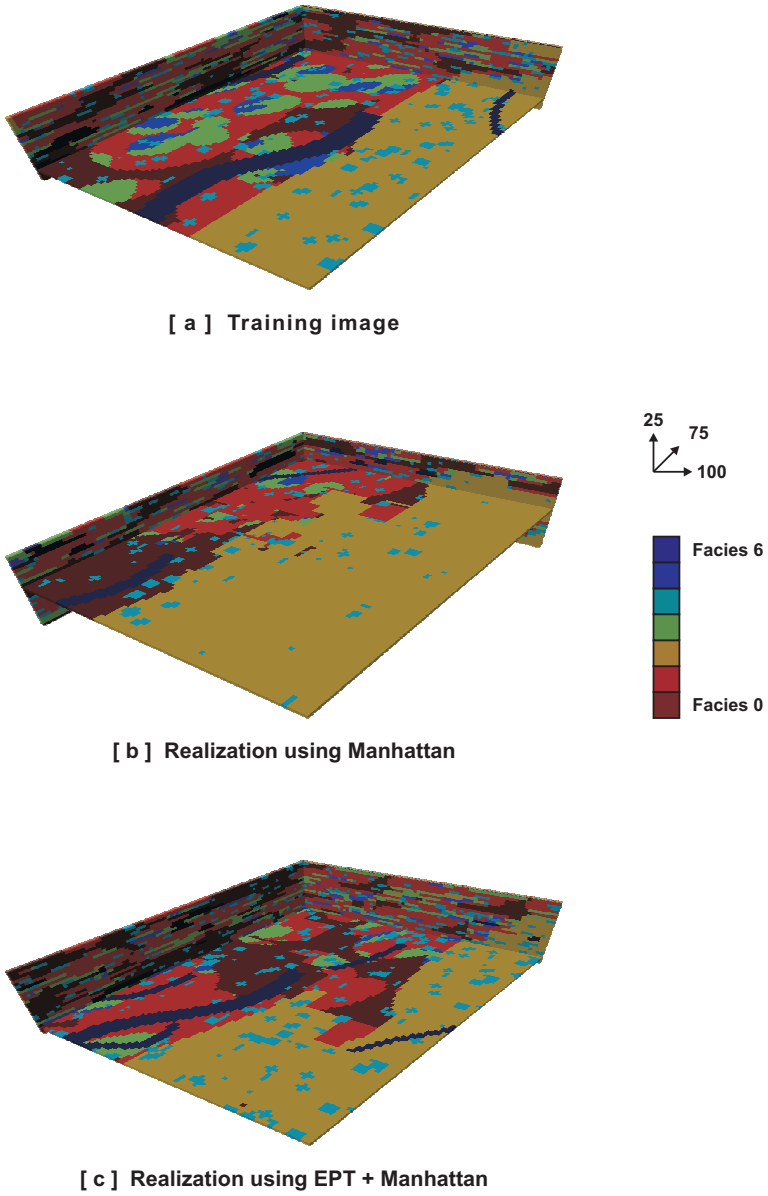


Figure 5.14: Application of unconditional, dual template simulation algorithm to a multiple-category training image using only Manhattan distance and Euclidean proximity transforms $\mathbf{EPT}(\cdot)$ + Manhattan distance.

Chapter 6

Data Conditioning

This chapter explains the details of how data conditioning to both hard data (such as well data) and soft data (such as seismic) is obtained.

6.1 Hard Data Conditioning

Consider the hard datum shown in Figure 6.1. The datum is assumed to belong to a hard data image **hd** and **hd** is discretized by the same Cartesian grid $\mathbf{G}_{re}$ used for the realization **re**. A hard data image **hd** contains only hard data and is uninformed elsewhere, i.e. $hd(\mathbf{u}) = \chi$ if at node $\mathbf{u}$ there is no hard datum.

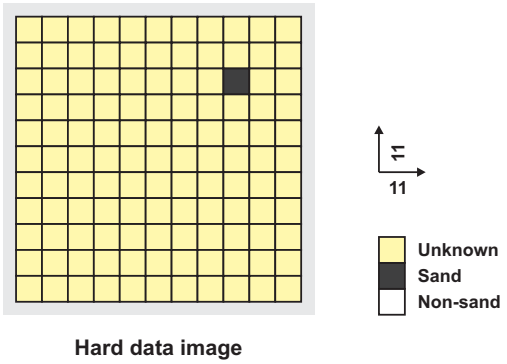


Figure 6.1: A hard data image contains only data and is uninformed elsewhere.

Similar to a data event $\mathbf{dev_T(u)}$, a hard data event $\mathbf{hdev_T(u)}$ is defined as the vector of values $hd(\mathbf{u}+\mathbf{h}_\alpha)$ that falls within a template $\mathbf{T}$ on the hard data image $\mathbf{hd}$, i.e. $\mathbf{hdev_T(u)} = \mathbf{hd_T(u)}$. Since hard data never change at any time, different from a data event $\mathbf{dev_T(u)}$, a hard data event $\mathbf{hdev_T(u)}$ remains the same throughout a simulation.

To generate conditional realizations, the hard data image $\mathbf{hd}$ is first copied on to the realization $\mathbf{re}$, i.e. the realization $\mathbf{re}$ is no longer fully uninformed at the beginning of a simulation but contains the hard data instead. Since the hard data image $\mathbf{hd}$ is copied on to the realization $\mathbf{re}$, a data event $\mathbf{dev_T(u)}$ automatically contains the hard data event $\mathbf{hdev_T(u)}$. The unconditional algorithm described in the previous chapters is then modified such that it accepts the hard data as hard constraints, i.e. during the simulation, for each data event $\mathbf{dev_T(u)}$ with $\mathbf{u}$ in the vicinity of a hard data point (defined by template $\mathbf{T}$), the conditional algorithm attempts to find a most similar pattern $\mathbf{pat^*_T}$ such that the pattern $\mathbf{pat^*_T}$ reproduces the hard data event $\mathbf{hdev_T(u)}$ on the realization $\mathbf{re}$.

In the following sections, this modification is first described for single-grid simulation; then, for dual template (multiple-grid) simulation.

6.1.1 Conditional Single-grid Simulation

Conditioning to hard data in a single-grid setting requires a two-stage lookup of the pattern database $\mathbf{patdb_T}$ using first the hard data event $\mathbf{hdev_T(u)}$ and then the data event $\mathbf{dev_T(u)}$.

The algorithm starts with the first lookup where the actual conditioning to hard data occurs. In this stage, called "preconditioning", the algorithm finds all patterns $\mathbf{pat^k_T}$ such that $s\left\langle \mathbf{hdev_T(u)}, \mathbf{pat^k_T} \right\rangle \geq -\epsilon_{hd}$ where $\epsilon_{hd} \geq 0$ is a threshold typically taken close to zero. In terms of dissimilarity (distance), the same condition is written as: $d\left\langle \mathbf{hdev_T(u)}, \mathbf{pat^k_T} \right\rangle \leq \epsilon_{hd}$. In other words, during the first lookup, all patterns $\mathbf{pat^k_T}$ of the pattern database $\mathbf{patdb_T}$ that are similar to the hard data event $\mathbf{hdev_T(u)}$ within a given threshold ϵ_{hd} are found. Setting $\epsilon_{hd} = 0$ amounts to searching only for patterns $\mathbf{pat^k_T}$ that exactly match the hard data event $\mathbf{hdev_T(u)}$.

There are three possible outcomes of this first stage:

1. The hard data event $\mathbf{hdev_T(u)}$ is completely uninformed, i.e. there is no hard data near the vicinity of the node $\mathbf{u}$ (defined by template $\mathbf{T}$). Then, all patterns $\mathbf{pat}_\mathbf{T}^k$ fulfill the condition and are considered for the next stage;

2. The hard data event $\mathbf{hdev_T(u)}$ is at least partially informed and at least one $\mathbf{pat}_\mathbf{T}^k$ fulfills the condition $s\left\langle \mathbf{hdev_T(u)}, \mathbf{pat}_\mathbf{T}^k \right\rangle \geq -\epsilon_{hd}$. Note that, when $\epsilon_{hd} = 0$ and the hard data event $\mathbf{hdev_T(u)}$ contains several hard data nodes, it is typically unlikely to find many if any one $\mathbf{pat}_\mathbf{T}^k$ which fulfill the condition.

3. None of the patterns $\mathbf{pat}_\mathbf{T}^k$ fulfill the condition $s\left\langle \mathbf{hdev_T(u)}, \mathbf{pat}_\mathbf{T}^k \right\rangle \geq -\epsilon_{hd}$. In such a case, another search is performed to find a most similar pattern $\mathbf{pat}_\mathbf{T}^*$ to the hard data event $\mathbf{hdev_T(u)}$ by maximizing $s\left\langle \mathbf{hdev_T(u)}, \mathbf{pat}_\mathbf{T}^k \right\rangle$. This most similar pattern $\mathbf{pat}_\mathbf{T}^*$ is then assumed to fulfill the condition. In other words, the algorithm settles for a pattern that is most similar to the hard data event $\mathbf{hdev_T(u)}$ but that does not necessarily match the hard data up to the desired level (indicated by ϵ_{hd}).

The last case of the above list occurs when the training image **ti** is not representative of the hard data. In other words, the geological scenario described by the training image **ti** (and the pattern database $\mathbf{patdb_T}$) conflicts with the available data, i.e. the **ti** is not rich enough in that it does not contain patterns that can fully explain the hard data patterns of the hard data image **hd**. Section 6.1.3 of this chapter further discusses this issue and gives examples.

The patterns found in the first lookup form another pattern database $\mathbf{hpatdb_T}$ called the "preconditioned pattern database". The second lookup is performed on this preconditioned pattern database using the data event $\mathbf{dev_T(u)}$. The algorithm finds the most similar pattern $\mathbf{pat}_\mathbf{T}^* \in \mathbf{hpatdb_T}$ that maximizes $s\left\langle \mathbf{dev_T(u)}, \mathbf{pat}_\mathbf{T}^{hk} \right\rangle$ for all patterns $\mathbf{pat}_\mathbf{T}^{hk} \in \mathbf{hpatdb_T}$ and hk indicates an index of the preconditioned pattern database $\mathbf{hpatdb_T}$. In other words, conditioning to hard data does not change how the most similar pattern $\mathbf{pat}_\mathbf{T}^*$ is found but only changes the pattern database on which the similarity search is performed.

In the above process, if the preconditioned pattern database $\mathbf{hpatdb_T}$ contains only a single pattern (either because there was only a single pattern $\mathbf{pat}^k_{\mathbf{T}}$ that fulfilled the condition $s\left\langle \mathbf{hdev_T}(\mathbf{u}), \mathbf{pat}^k_{\mathbf{T}} \right\rangle \geq -\epsilon_{hd}$ during the first lookup or because none of the patterns fulfilled the condition and a most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$ to the hard data event $\mathbf{hdev_T}(\mathbf{u})$ was found as an approximation), the second lookup is not performed and this pattern is immediately taken as the most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$.

Once the most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$ is found, it is still pasted on to the data event $\mathbf{dev_T}(\mathbf{u})$ but now only for nodes $\mathbf{u}+\mathbf{h}_\alpha$ that does not contain a hard datum, i.e. $dev_{\mathbf{T}}(\mathbf{u}+\mathbf{h}_\alpha) = pat^*_{\mathbf{T}}(\mathbf{h}_\alpha)$ only if $hdev_{\mathbf{T}}(\mathbf{u}+\mathbf{h}_\alpha) = \chi$.

Algorithm 6.1 describes the above two-stage lookup and the conditional pasting of the most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$. Figure 6.2 illustrates the application of Algorithm 6.1 using the hard data image of Figure 6.1.

6.1.2 Conditional Dual Template Simulation

Section 2.5.1 indicate that conditioning to hard data might create problems when used in a multiple-grid setting. A particular multiple-grid $\mathbf{G}^g_{re}$ might not include the locations containing the hard data, i.e. the data might be located on one of the finer grids $\mathbf{G}^{g'}_{re}$ with $g > g'$. Figure 2.7 of Section 2.5.1 illustrates this. In such a case, the hard data will not be included within the similarity calculations, resulting in poorly-conditioned realizations.

The cause of the above problem lies in the multiple-grid approach, namely the approximation of the large and full data event using a coarse data event $\mathbf{dev_{T^g}}(\mathbf{u})$. Due to this approximation, it is virtually impossible to honor the data exactly at their locations $\mathbf{u}$ if $\mathbf{u}$ does not fall on the current coarse grid $\mathbf{G}^g_{re}$. From this point of view, solutions such as data relocation (Section 2.5.1) simply amount to approximating the actual hard data event with another (different) hard data event.

An alternative (and exact) approach to data relocation is to utilize the dual template $\widetilde{\mathbf{T}}$ for hard data conditioning. In Step 2 of Algorithm 6.1, instead of using the primal hard data event $\mathbf{hdev_T}(\mathbf{u})$, one can use the dual hard data event $\mathbf{hdev}_{\widetilde{\mathbf{T}}^g}(\mathbf{u})$

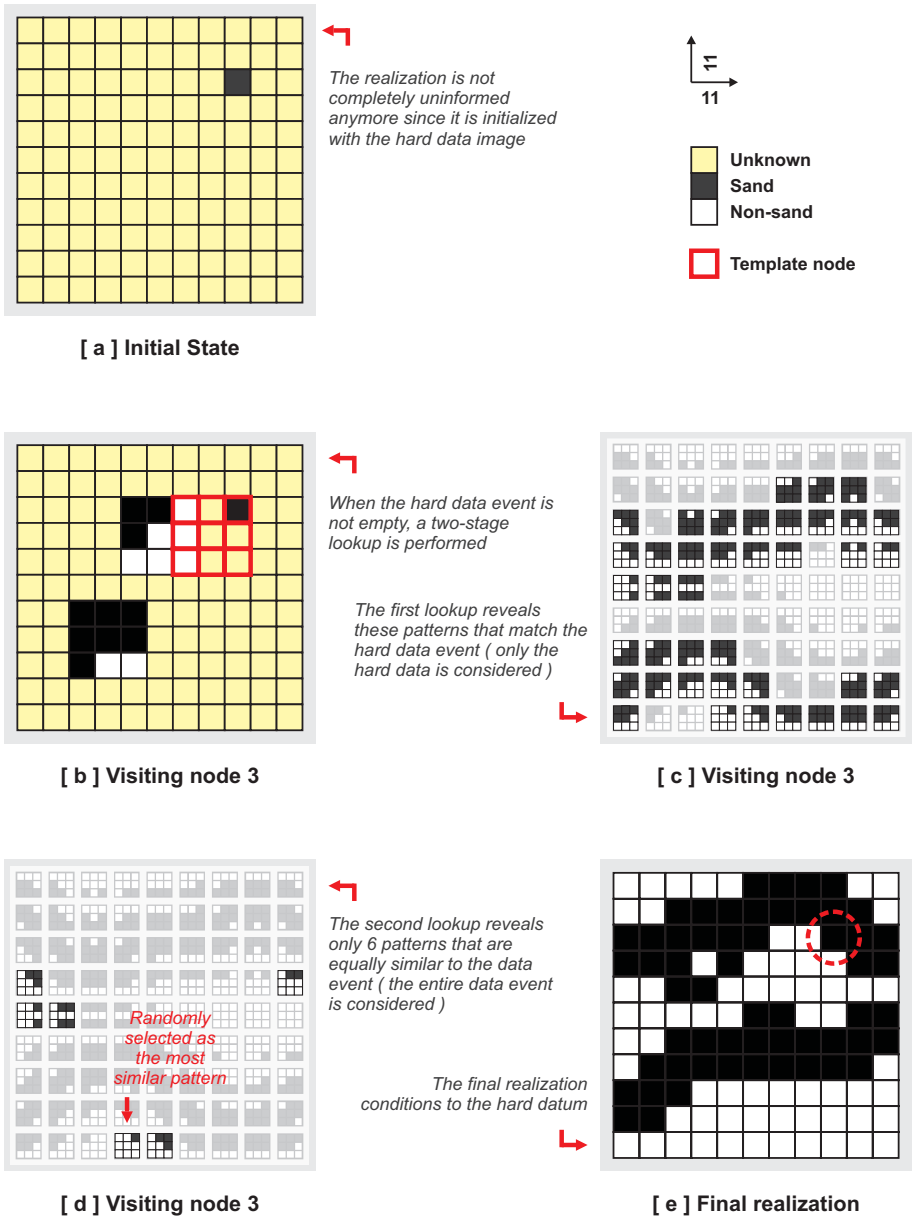


Figure 6.2: Internal steps of Algorithm 6.1 when applied to a 11×11 realization using the hard data image of Figure 6.1, the training image of Figure 3.1, the pattern database of Figure 3.3 and a 3×3 template.

Algorithm 6.1: Hard data conditioning

1. During the simulation of node $\mathbf{u}$, retrieve the hard data event $\mathbf{hdev_T}(\mathbf{u})$ and the data event $\mathbf{dev_T}(\mathbf{u})$.
2. Find all patterns $\mathbf{pat}_\mathbf{T}^k \in \mathbf{patdb_T}$ such that $s\left\langle \mathbf{hdev_T}(\mathbf{u}), \mathbf{pat}_\mathbf{T}^k \right\rangle \geq -\epsilon_{hd}$ where $\epsilon_{hd} \geq 0$ is a threshold typically taken as zero. Such patterns $\mathbf{pat}_\mathbf{T}^k$ are considered to form a preconditioned pattern database $\mathbf{hpatdb_T}$ [First stage].
3. If none of the patterns $\mathbf{pat}_\mathbf{T}^k$ fulfill the condition $s\left\langle \mathbf{hdev_T}(\mathbf{u}), \mathbf{pat}_\mathbf{T}^k \right\rangle \geq \epsilon_{hd}$ (i.e. the preconditioned pattern database $\mathbf{hpatdb_T}$ is empty), find the most similar pattern $\mathbf{pat}_\mathbf{T}^*$ that maximizes $s\left\langle \mathbf{hdev_T}(\mathbf{u}), \mathbf{pat}_\mathbf{T}^k \right\rangle$ instead. This pattern $\mathbf{pat}_\mathbf{T}^*$ is then taken as the only pattern of $\mathbf{hpatdb_T}$ [First stage].
4. If $\mathbf{hpatdb_T}$ contains only a single pattern, accept this pattern as the most similar pattern $\mathbf{pat}_\mathbf{T}^*$ [Second stage].
5. If $\mathbf{hpatdb_T}$ contains more than one pattern, find the most similar pattern $\mathbf{pat}_\mathbf{T}^*$ by maximizing $s\left\langle \mathbf{dev_T}(\mathbf{u}), \mathbf{pat}_\mathbf{T}^{hk} \right\rangle$ where $\mathbf{pat}_\mathbf{T}^{hk} \in \mathbf{hpatdb_T}$ with hk indicating an index of the previously constructed preconditioned pattern database $\mathbf{hpatdb_T}$ [Second stage].
6. Paste all non-hard-data nodes of the most similar pattern $\mathbf{pat}_\mathbf{T}^*$ on to the data event $\mathbf{dev_T}(\mathbf{u})$, i.e. $dev_\mathbf{T}(\mathbf{u}+\mathbf{h}_\alpha) = pat_\mathbf{T}^*(\mathbf{h}_\alpha)$ only if $hdev_\mathbf{T}(\mathbf{u}+\mathbf{h}_\alpha) = \chi$.

to construct the preconditioned pattern database $\mathbf{hpatdb_T}$. Since the dual template $\widetilde{\mathbf{T}}$ contains all the nodes of the hard data image $\mathbf{hd}$, there is no need for data relocation. Then, finding all patterns $\mathbf{pat}_{\widetilde{\mathbf{T}}^g}^k$ that fulfills the condition,

$$s\left\langle \mathbf{hdev}_{\widetilde{\mathbf{T}}^g}(\mathbf{u}), \mathbf{pat}_{\widetilde{\mathbf{T}}^g}^k \right\rangle \geq -\epsilon_{hd} \tag{6.1}$$

during the first lookup guarantees that all hard data are considered during any coarse grid $\mathbf{G}_{re}^g$ simulation even if these data are located on the finest grid $\mathbf{G}_{re}^0$. Figure 6.3 illustrates this. The preconditioned pattern database $\mathbf{hpatdb_T}$ is still constructed

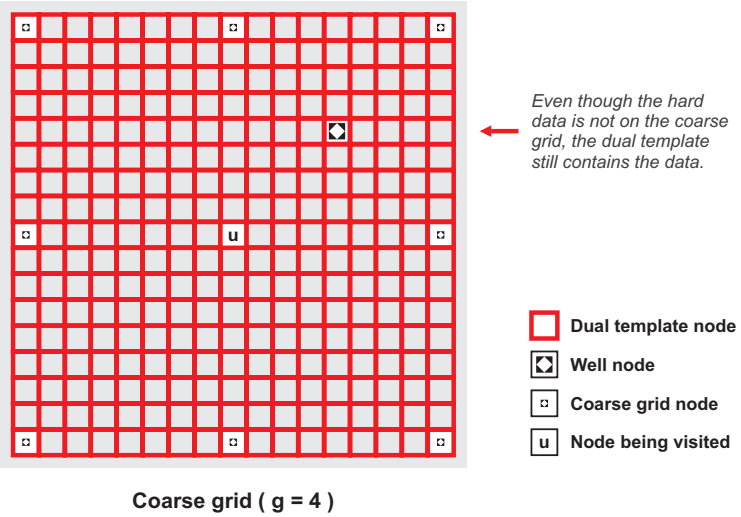


Figure 6.3: Unlike Figure 2.7, the hard datum does not need to be relocated when dual templates are used for preconditioning.

from primal patterns $\mathbf{pat}^k_{\mathbf{T}^g}$. One retains only the primal $\mathbf{pat}^k_{\mathbf{T}^g}$ for which the corresponding dual pattern $\mathbf{pat}^k_{\widetilde{\mathbf{T}}^g}$ fulfills Equation 6.1. If no such dual pattern(s) $\mathbf{pat}^k_{\widetilde{\mathbf{T}}^g}$ can be found, then the algorithm maximizes $s\langle\cdot,\cdot\rangle$ of Equation 6.1 to find a most similar dual pattern $\mathbf{pat}^*_{\widetilde{\mathbf{T}}^g}$ to the dual hard data event $\mathbf{hdev}_{\widetilde{\mathbf{T}}^g}(\mathbf{u})$ and pastes this pattern on to $\mathbf{re}$ (on the finest grid $\mathbf{G}^1_{re}$).

Since $n_{\widetilde{\mathbf{T}}^g} >> n_{\mathbf{T}^g}$ (i.e. the number of nodes in a $\widetilde{\mathbf{T}}^g$ is greater than the number of nodes in the corresponding $\mathbf{T}^g$), using dual templates when searching for the most similar pattern using Equation 6.1 may impose an additional CPU load increasing with the density of hard data. For partially informed dual hard data event $\mathbf{hdev}_{\widetilde{\mathbf{T}}^g}(\mathbf{u})$, the 'active' number of nodes used for the similarity calculations is equal to $n_{hd} \leq n_{\widetilde{\mathbf{T}}^g}$, i.e. the number of hard data nodes that are located within the dual template including the ones on the finest grid. Since n_{hd} only refers to original hard data and not to any previously determined nodes on any grid, for sparse hard data, n_{hd} remains small (Figure 6.4) and using dual templates for conditioning would not require significantly more CPU time. However, for dense hard data, with n_{hd} large (possibly $>> n_{\mathbf{T}^g}$), the search for a most similar dual pattern $\mathbf{pat}^*_{\widetilde{\mathbf{T}}^g}$ can be expensive.

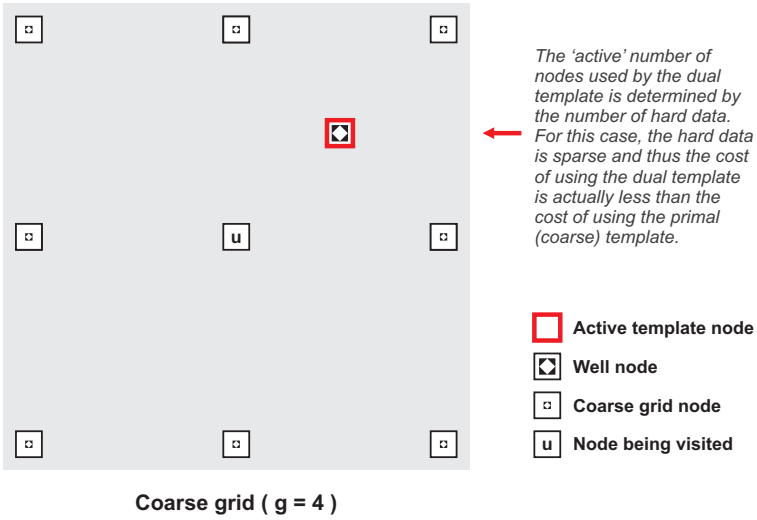


Figure 6.4: The 'active' nodes of the dual template $\widetilde{\mathbf{T}}^g$ (Previously shown in Figure 6.3). In the figure, the dual template contains only a single hard data node of the hard data image **hd**, i.e. $n_{hd} < n_{\mathbf{T}^g} << n_{\widetilde{\mathbf{T}}^g}$.

The expensive search for $\mathbf{pat}^*_{\widetilde{\mathbf{T}}^g}$ is required for exactness in hard data conditioning and cannot be avoided. However, since the dual hard data event $\mathbf{hdev}_{\widetilde{\mathbf{T}}^g}(\mathbf{u})$ remains the same for a given template $\widetilde{\mathbf{T}}^g$ for all realizations regardless of the random path, it is possible to perform the necessary calculations only once prior to simulation and to reuse the results for all subsequent realizations. This essentially amounts to constructing a location specific preconditioned pattern database $\mathbf{hpatdb}_{\mathbf{T}}(\mathbf{u})$ before performing the first simulation and using this pattern database for all realizations, i.e. the first lookup of Algorithm 6.1 is performed ahead of time and the actual simulations only perform the second lookup. Chapter 8 describes in detail how this can be efficiently implemented.

6.1.3 2D Examples and Sensitivity Study

Consider the reference case of Figure 6.5 and the training images of Figure 6.6. Using this reference and the training images, several runs are performed to investigate the

sensitivity of hard data conditioning to different scenarios of available hard data.

Figure 6.7 shows sample realizations and the E-types (averages) of 25 realizations generated using the first training image of Figure 6.6 with different hard data sets. In the figure, the E-types are used to check the quality of conditioning. If the algorithm conditions properly, an E-type should give less uncertainty near the hard data points. Furthermore, the overall uncertainty of the final reservoir model should decrease with increasing number of hard data. Figures 6.7b, d and f demonstrate this. Since the training image used for this example provides a geological scenario consistent with the reference, the results do not conflict with the data and, as expected, with more data, the realizations converge to the reference itself.

Figure 6.8 uses the same hard data sets as Figure 6.7 but with the second, incorrect training image of Figure 6.6. This training image poorly explains the patterns of the reference. Despite the conflict between the training image and the hard data, the conditional algorithm successfully fits the data to the training image patterns for up to 100 hard data nodes. However, for the dense data set with 1000 hard data nodes (Figure 6.8e/f), both the realization and the e-type show patterns and trends that do not exist in the training image. This occurs since the realization honors the hard data no matter how wrong is the training image and thus the training image patterns are no longer fully copied on to the realization **re**.

Figure 6.9 shows an example of geobody conditioning. In reservoir modeling, such contiguous geobodies are often obtained from 3D seismic and considered hard. As apparent from the E-type estimates, the presence of the geobody effects an area much larger than the actual size of the geobody itself (6.9c and e). Since both training images depict a channel system with a certain, average distance between individual channels, a channel piece (geobody) taken as hard data heavily constrains regions above and below it in Figure 6.9c, and regions left and right of the geobody in Figure 6.9e. Since the training image used for Figure 6.9e conflicts with the data, the geobody does not reduce uncertainty as much as the geobody of Figure 6.9c; compare the corresponding two E-types.

In Figure 6.10, the conditional algorithm is applied to multiple geobodies of varying sizes. Once again, when the correct training image is used, the results approach

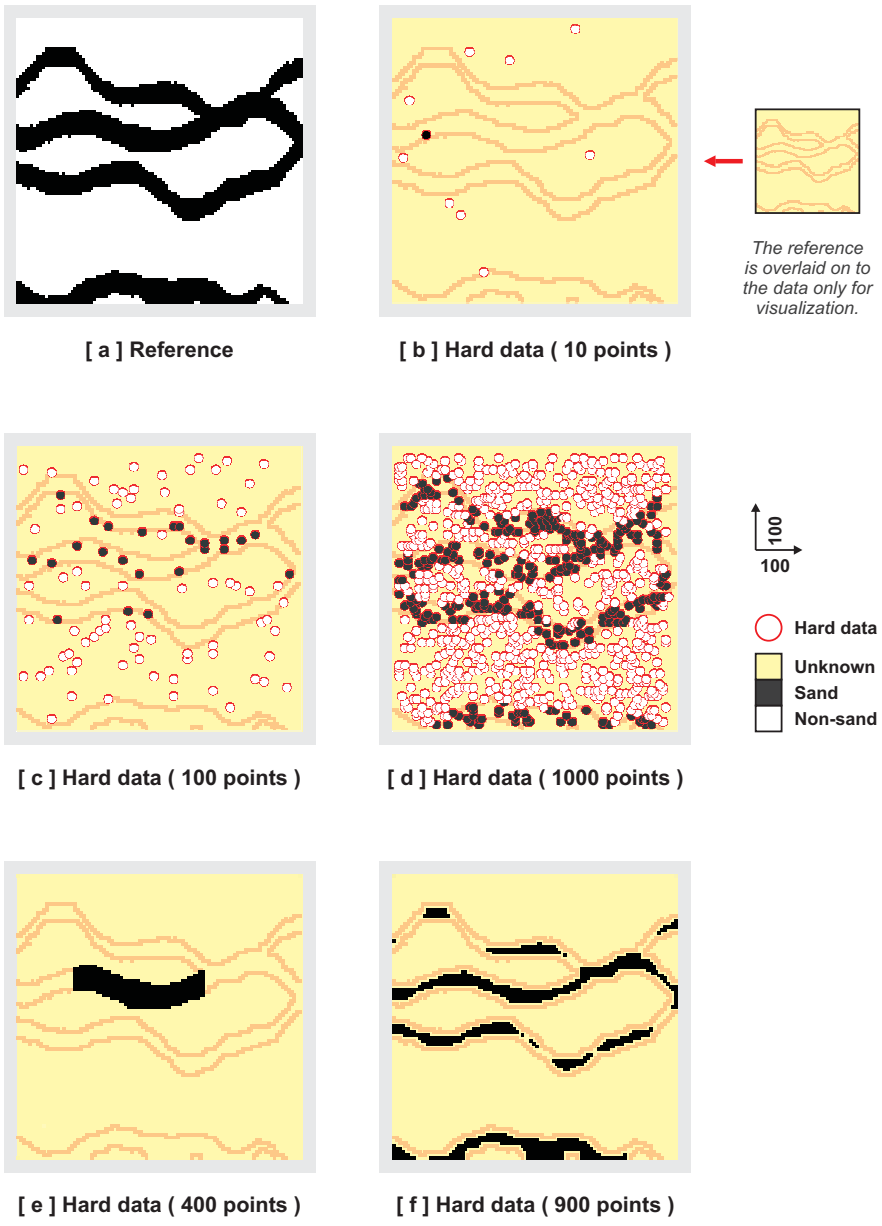
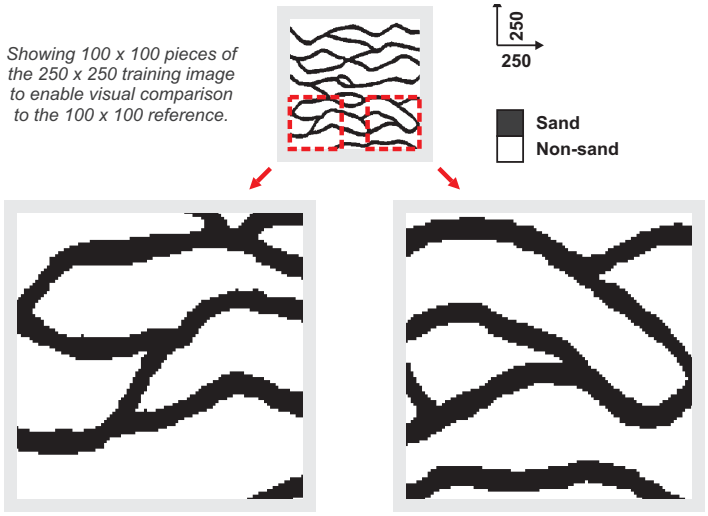
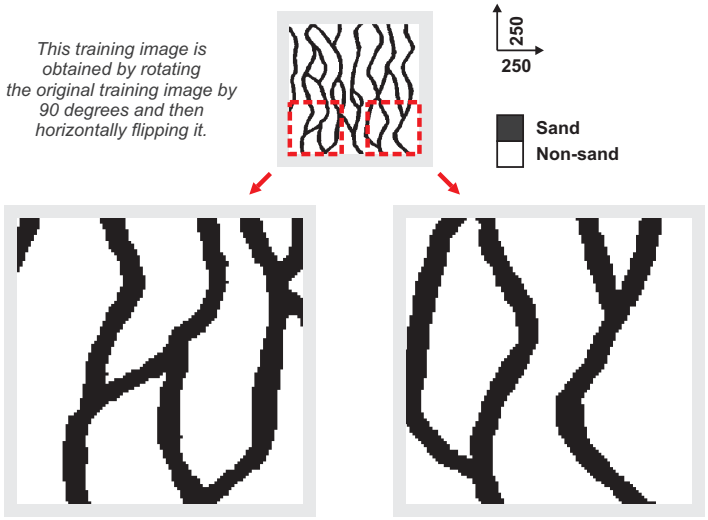


Figure 6.5: The 100×100 binary (sand/non-sand) reference [a] and several different sets of hard data sampled from this reference [b - f].



[a] Geologically consistent training image



[b] Geologically inconsistent training image

Figure 6.6: The two training images used for the sensitivity analysis. The first training image is highly representative of the actual geology of the reference (Figure 6.5a) whereas the second training image represents a conflicting geological scenario.

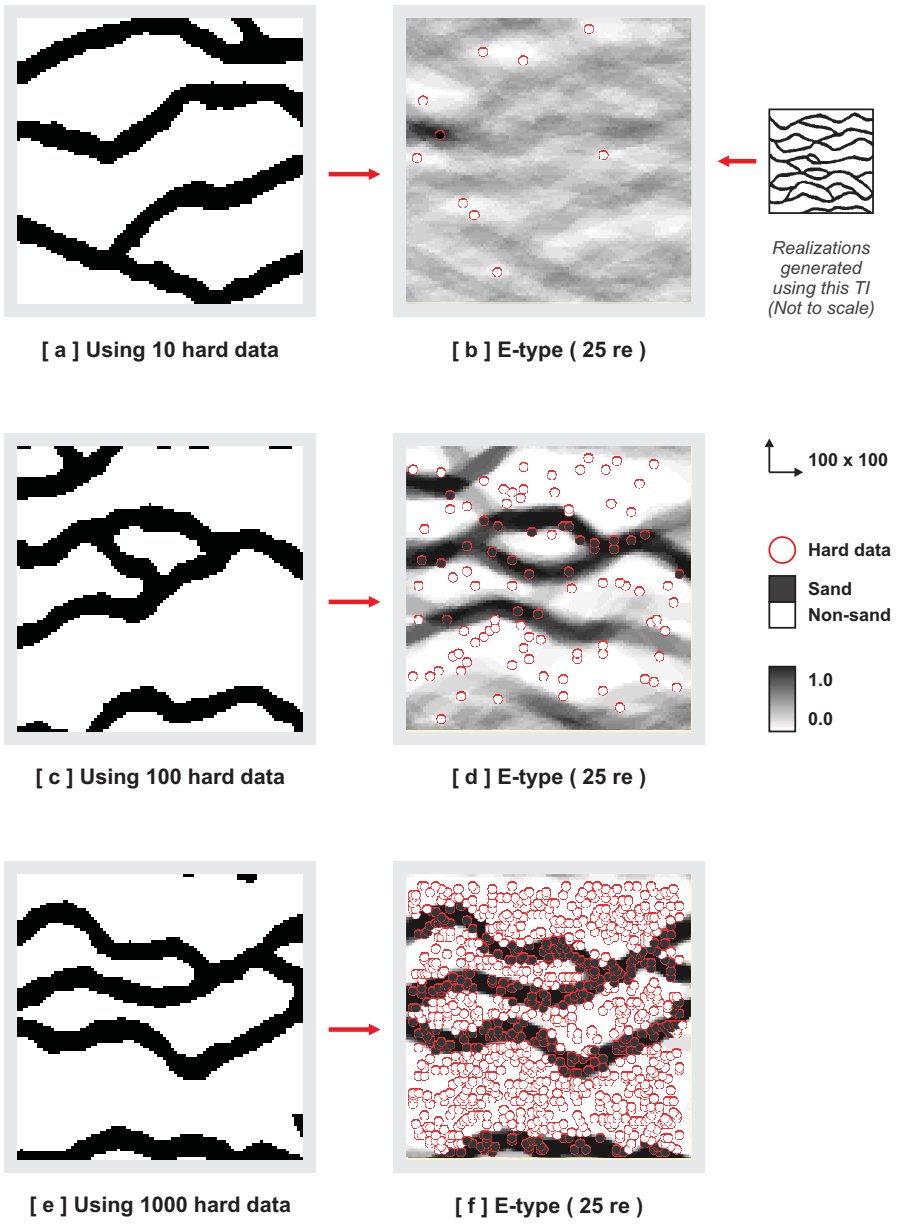


Figure 6.7: Conditional realizations and E-types obtained using the first training image of Figure 6.6 [$n_{\mathbf{T}} = 7 \times 7$; $n_g = 3$].

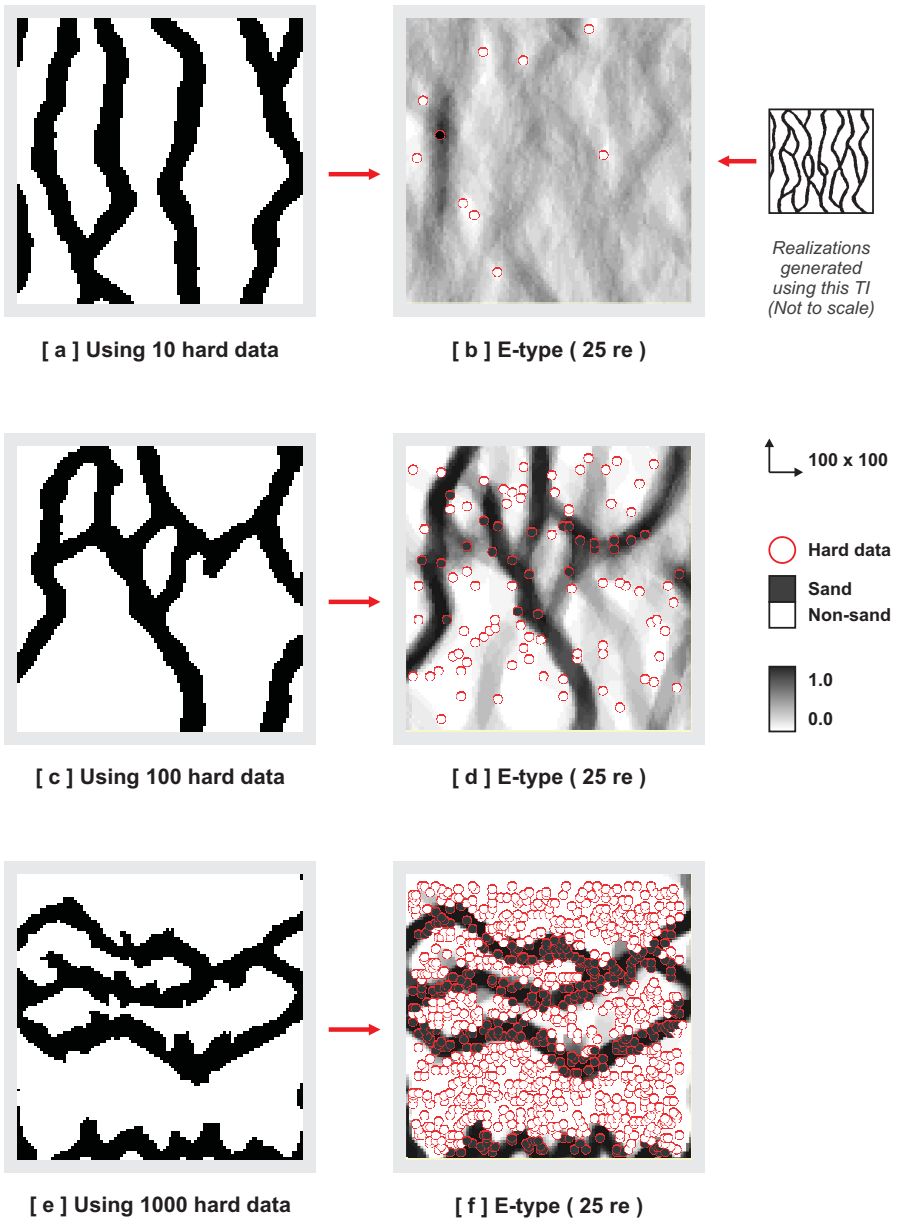


Figure 6.8: Conditional realizations and E-types obtained using the second training image of Figure 6.6 [$n_{\mathbf{T}} = 7 \times 7$; $n_g = 3$].

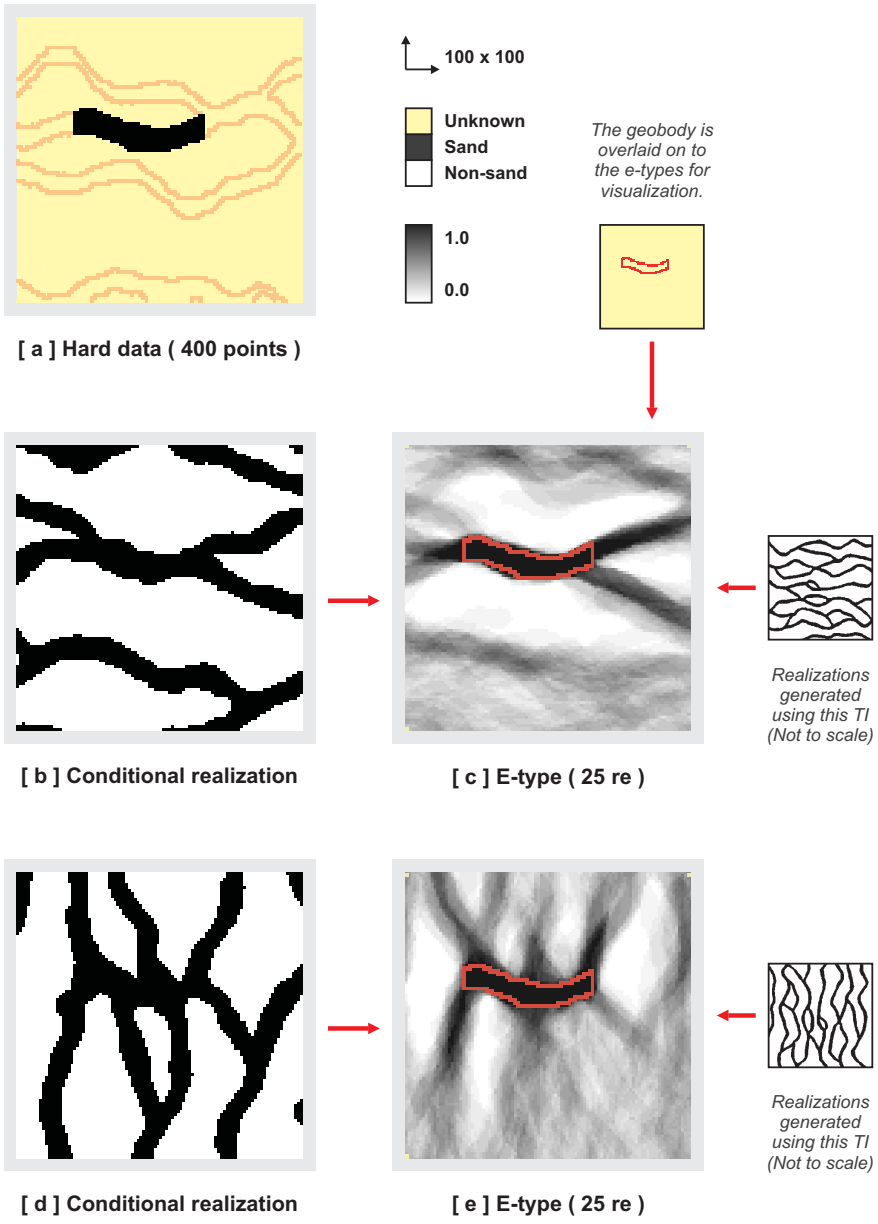


Figure 6.9: Conditioning to a geobody (sand only; 400 points) [a] using a representative and a conflicting training image [$n_{\mathbf{T}} = 7 \times 7$; $n_g = 3$].

the reference. Using the wrong training image introduces artifact patterns into the realizations. Since the geobodies clearly indicate channels undulating in east - west direction, different from Figure 6.8e and f, this time, the algorithm cannot avoid introducing large-scale artifacts during the simulations. As a result, the realizations are poorly conditioned to the hard data; yet, the hard data nodes themselves are still exactly reproduced.

The pattern reproduction problem of Figure 6.10d is apparent via a simple visual check. However, quality assurance via visualization might not always be reliable. Such assessment of quality is highly subjective (especially for complex, 3D training images), hard to quantify, heavily dependent on the experience of the expert performing the check. For this reason, a quantitative quality check is often required; especially when several realizations are to be checked.

Since the conditional algorithm calculates the similarity $s\langle\cdot,\cdot\rangle$ for every node of a realization, one way to quantify and visualize the quality of pattern reproduction in a single realization is to utilize the similarity error (i.e., the distance $d\langle\cdot,\cdot\rangle$) as a measure of mismatch. Every time a lookup of the pattern database provides a most similar pattern that does not exactly match the current data event, a "pattern mismatch" occurs, i.e. $d\langle\cdot,\cdot\rangle \neq 0$. Figure 6.11 demonstrates the use of location-specific similarity errors as such a measure. In the figure, a so-called "simulation fitness map" is calculated through cumulative summation of individual node errors $d\langle\cdot,\cdot\rangle$. In this scheme, since the similarity to a hard data event $\mathbf{hdev}_{\widetilde{\mathbf{T}}^g}(\mathbf{u})$ is calculated using a dual template $\widetilde{\mathbf{T}}^g$ with $n_{\widetilde{\mathbf{T}}} >> n_{\mathbf{T}}$, when a hard data mismatch occurs, it imposes a larger error $d\langle\cdot,\cdot\rangle$ compared to a data event without any hard data points (calculated using $\mathbf{T}^g$ instead of $\widetilde{\mathbf{T}}^g$) and results in poorer fitness. Due to this property, simulation fitness maps can be used as indicators of hard data conditioning quality (Note that, there is always a simulation error, even in unconditional simulations; though, that error is typically low). In Figure 6.11, the 'good' realization has a very low overall error (i.e. high fitness) on par with an unconditional simulation, whereas the 'bad' realization clearly has conditioning problems (the red to block spots). Furthermore, since the error is comparative and always positive (or zero), one can simply take the mean of the simulation fitness map as a single, scalar indication of the overall fitness

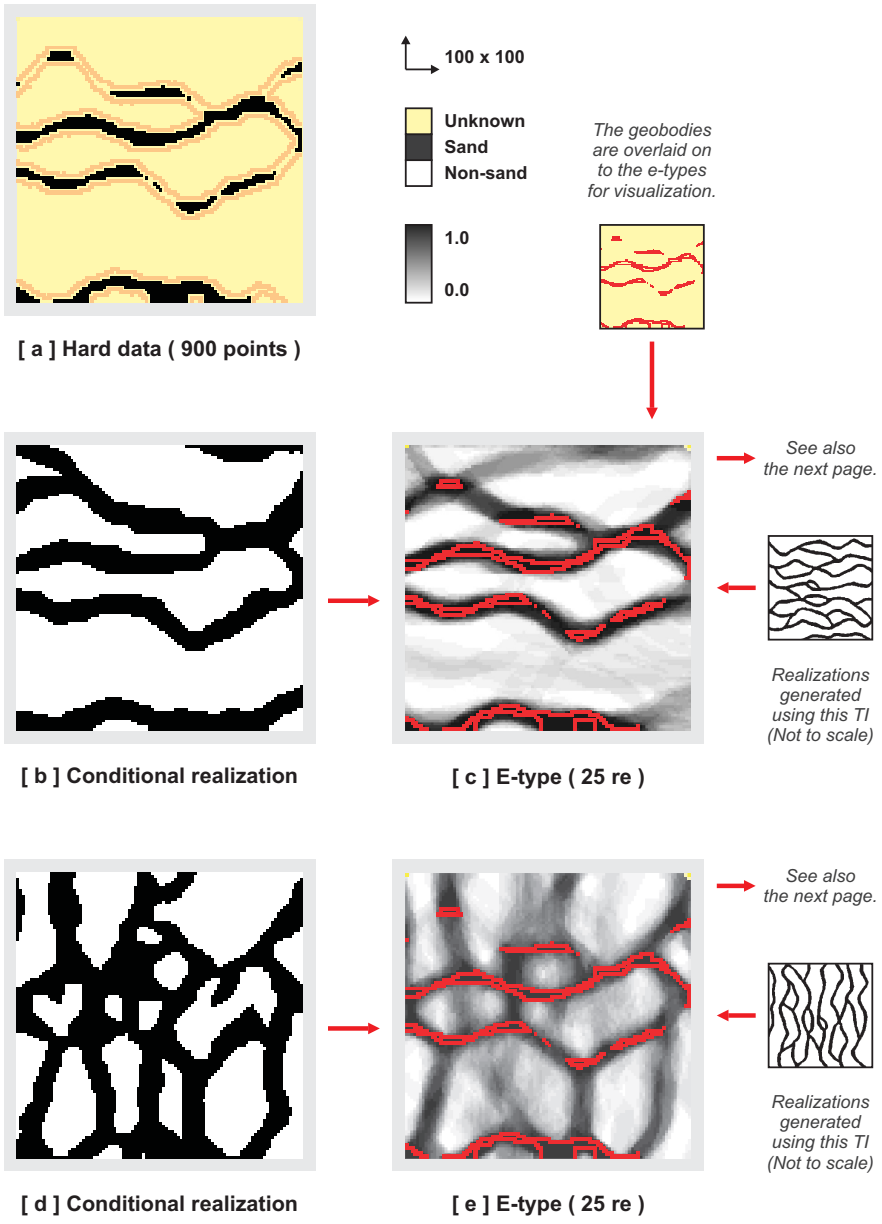


Figure 6.10: Conditioning to different sizes of geobodies (sand only; 900 points) [a] using a good and a poor training image [$n_{\mathbf{T}} = 7 \times 7$; $n_g = 3$].

of a realization, allowing quantitative comparisons between different realizations, or better between different alternative training images.

6.1.4 A Synthetic, 3D Example

Figure 6.12 shows the application of the conditional algorithm to a 3D, $100 \times 100 \times 50$ synthetic reference case with 6 facies generated using the SBED software (Wen et al., 1998). Two different data sets are sampled from this reference case to test the hard data conditioning capabilities of the algorithm in 3D (Figure 6.12b and 6.12c).

The first, dense data set is used with a training imagine that is highly representative of the reference (Figure 6.13a). Using such a good training image guarantees that, during the simulation, the number of conflicting patterns are kept to a minimum. However, in any real reservoir, one would expect a reasonable amount of conflict between the available data and the selected training image. The final realization (Figure 6.13c) obtained for this case should be viewed as a check of the conditioning capabilities of the algorithm in 3D rather than a representative example of hard data conditioning in a real reservoir.

The second, less dense data set is used with a training image that contains patterns likely to conflict with the available data (Figure 6.14a). In this case, the data dictates stacked channels whereas the training image depicts isolated channels. Figure 6.14c is the final conditional realization obtained, which presents lots of discontinuities.

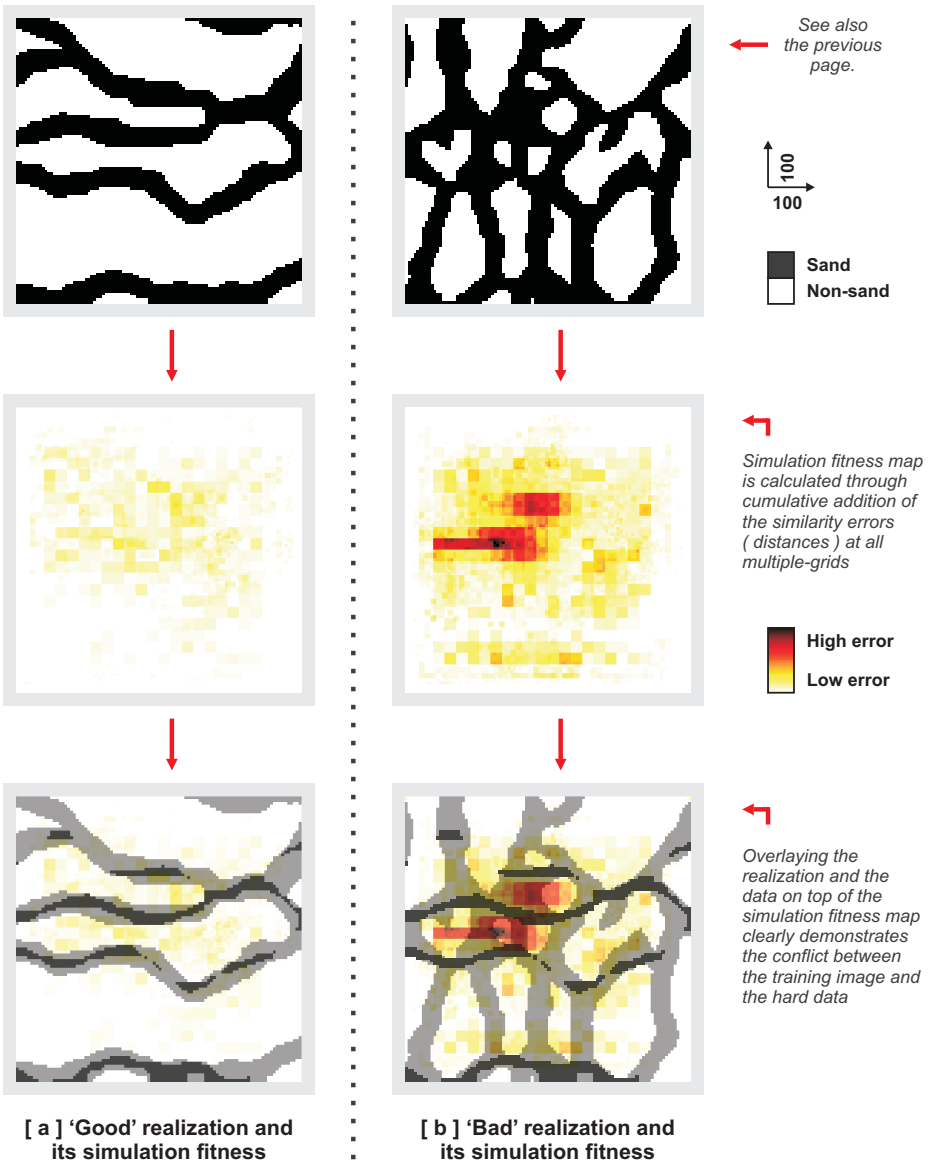
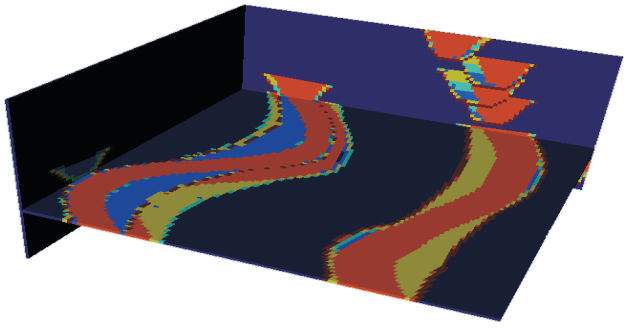
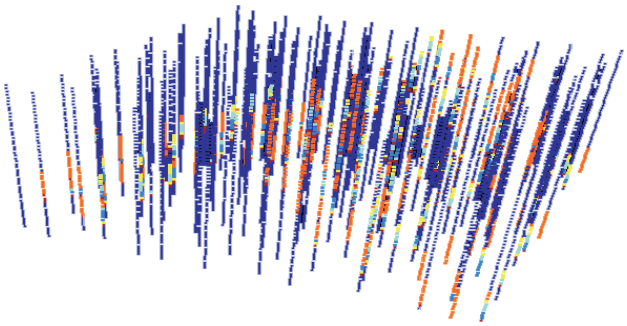


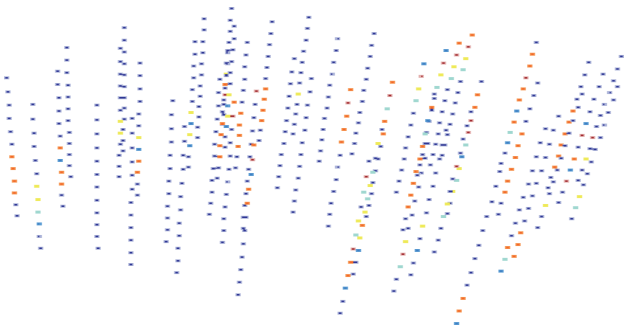
Figure 6.11: Simulation fitness maps can be used as an alternative quality assurance method when visual quality check of the realizations is not conclusive (Conditional realizations are taken from Figure 6.10).



[a] Reference

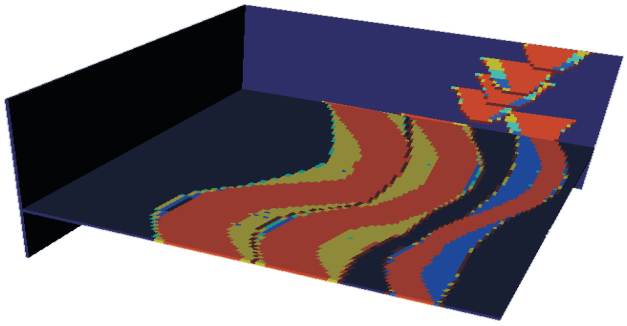


[b] Dense well data (150 wells; 2% of all nodes)

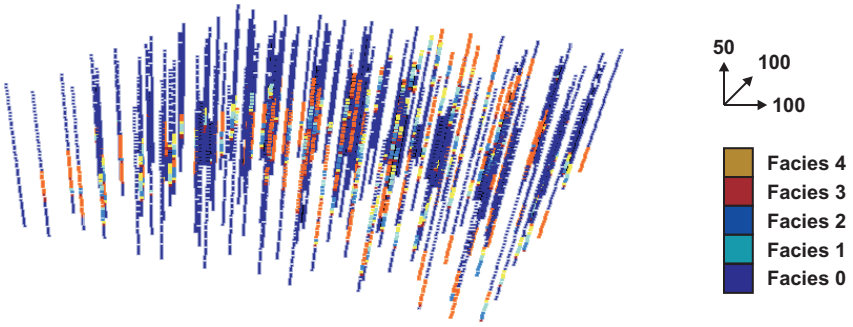


[c] Sparse well data (50 wells; 0.25% of all nodes)

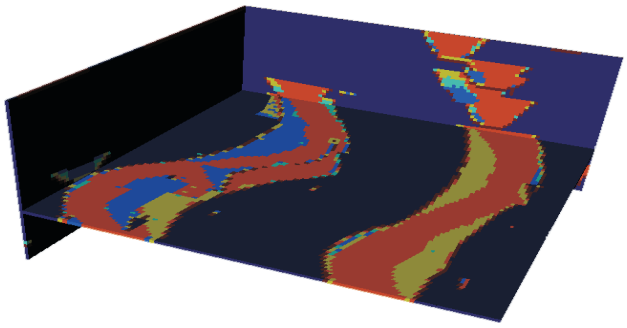
Figure 6.12: Reference for 3D hard data conditioning and two data sets randomly sampled from this reference.



[a] Training image

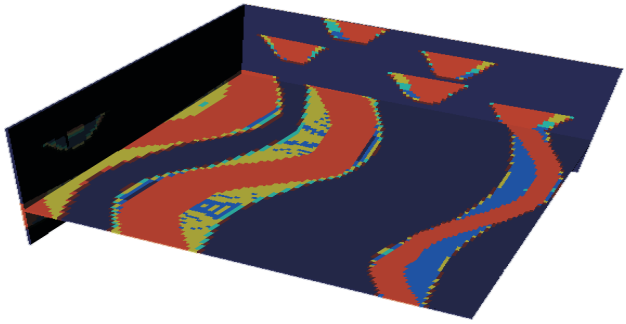


[b] Dense well data (150 wells; 2% of all nodes)

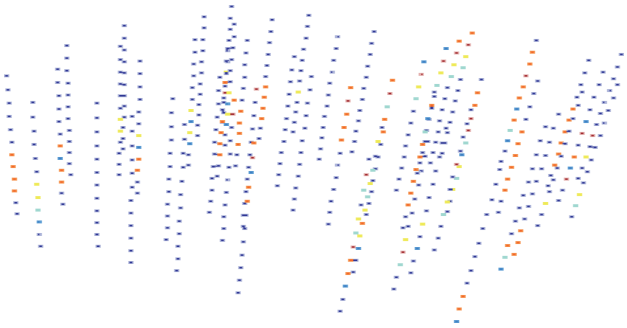


[c] Conditional realization

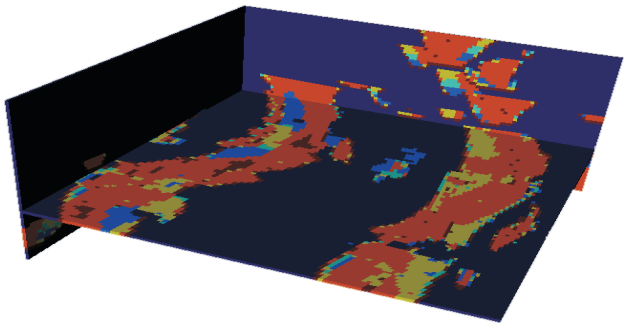
Figure 6.13: Hard data conditioning result obtained using a highly representative training image (of the reference given in Figure 6.12) [$n_{\mathbf{T}} = 11 \times 11 \times 5$; $n_g = 3$].



[a] Training image



[b] Sparse well data (50 wells; 0.25% of all nodes)



[c] Conditional realization

Figure 6.14: Hard data conditioning result obtained using a training image that has conflicting patterns with the given sparser data set [$n_{\mathbf{T}} = 11 \times 11 \times 5$; $n_g = 3$].

6.2 Soft Data Conditioning

Consider the soft data image **sd** shown in Figure 6.15b, discretized by the same Cartesian grid $\mathbf{G}_{re}$ used for the realization **re**. In general, soft data represents a 'filtered' view of the subsurface heterogeneity with a filter **F** representing the mapping between the true Earth's heterogeneity and the corresponding filtered view (soft data) as provided by the remote sensing device. Since **F** is dependent on the exact physics of the Earth, for real cases, it is never fully known. For this particular exercise (Figure 6.15b), the soft data was obtained by a simple 3×3 moving average ($= \mathbf{F}$) of the reference shown in Figure 6.15a.

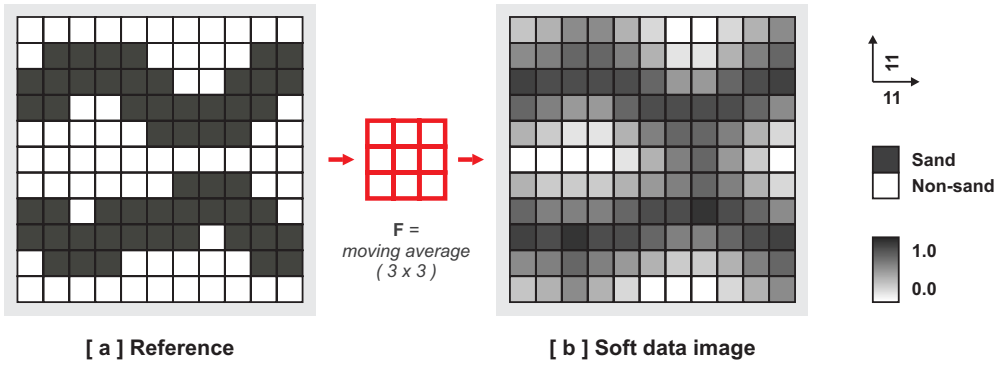


Figure 6.15: Soft data represents a 'filtered' view of the subsurface heterogeneity.

Similar to a data event $\mathbf{dev_T}(\mathbf{u})$, a soft data event $\mathbf{sdev_T}(\mathbf{u})$ is defined as the vector of values $sd(\mathbf{u}+\mathbf{h}_\alpha)$ that falls within a template **T** on the soft data image **sd**, i.e. $\mathbf{sdev_T}(\mathbf{u}) = \mathbf{sd_T}(\mathbf{u})$. Since the soft data never change at any time, as different from a data event $\mathbf{dev_T}(\mathbf{u})$, and as similar to a hard data event $\mathbf{hdev_T}(\mathbf{u})$, a soft data event $\mathbf{sdev_T}(\mathbf{u})$ remains the same throughout a simulation.

To condition to soft data, a modified version of Type 1 soft conditioning method of Strebelle (2000) is used (Section 2.5.2). This method was originally devised for the probability-based `SNESIM` algorithm. The following sections explain the details of how Strebelle's method is modified.

6.2.1 Soft Training Images and their Preprocessing

The proposed method requires pairs of training images: one for the variable that is to be simulated and another for the soft data variable, i.e. instead of a single training image **ti**, there are now two training images **ti** and **sti** (called the "soft training image"). The two training images are used to establish a relation between the (hard) simulation variable and the soft data variable. Theoretically, the soft training image **sti** should be obtained by applying **F** to the training image **ti**, i.e. $\mathbf{sti} = \mathbf{F}(\mathbf{ti})$. In other words, the soft training image represents the filtered view of the (hard) training image when the filter **F** is used. Since, in real cases, **F** is never fully known, it is approximated by $\mathbf{F}^*$ (the forward model) and, in practice, the soft training image is always obtained via $\mathbf{sti} = \mathbf{F}^*(\mathbf{ti})$. Figure 6.16 shows this for the training image previously shown in Figure 3.1. In the figure, the approximate model $\mathbf{F}^*$ is taken as equal to **F** of Figure 6.15 for illustrative purposes.

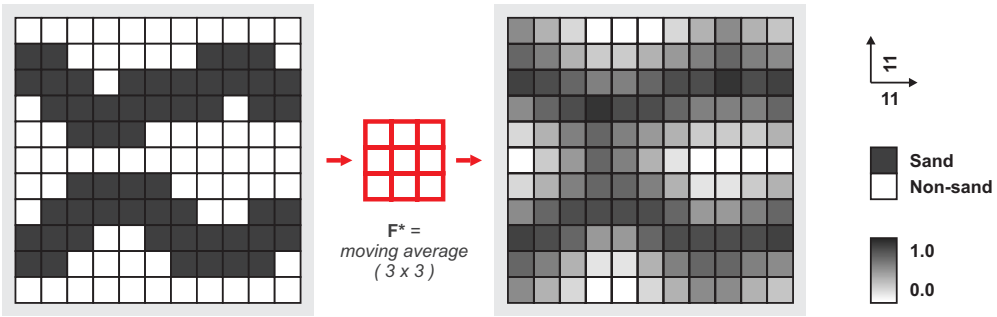


Figure 6.16: The soft training image **sti** is obtained via $\mathbf{sti} = \mathbf{F}^*(\mathbf{ti})$.

The relation between the hard and soft variables is obtained by preprocessing both training images simultaneously and by linking the hard patterns $\mathbf{pat}_\mathbf{T}^k$ of the training image **ti** to the soft patterns $\mathbf{spat}_\mathbf{T}^k$ $(= \mathbf{sti}_\mathbf{T}(\mathbf{u}))$ of the soft training image **sti** in the pattern database $\mathbf{patdb}_\mathbf{T}$. The scanning of each training image is still performed by Algorithm 3.1 but now both training images (hard and soft) are scanned jointly and the patterns from both images are paired together in $\mathbf{patdb}_\mathbf{T}$. Figure 6.17 illustrates this process for the grid $\mathbf{G}_{re}$ using a 3×3 template and the images of Figure 6.16.

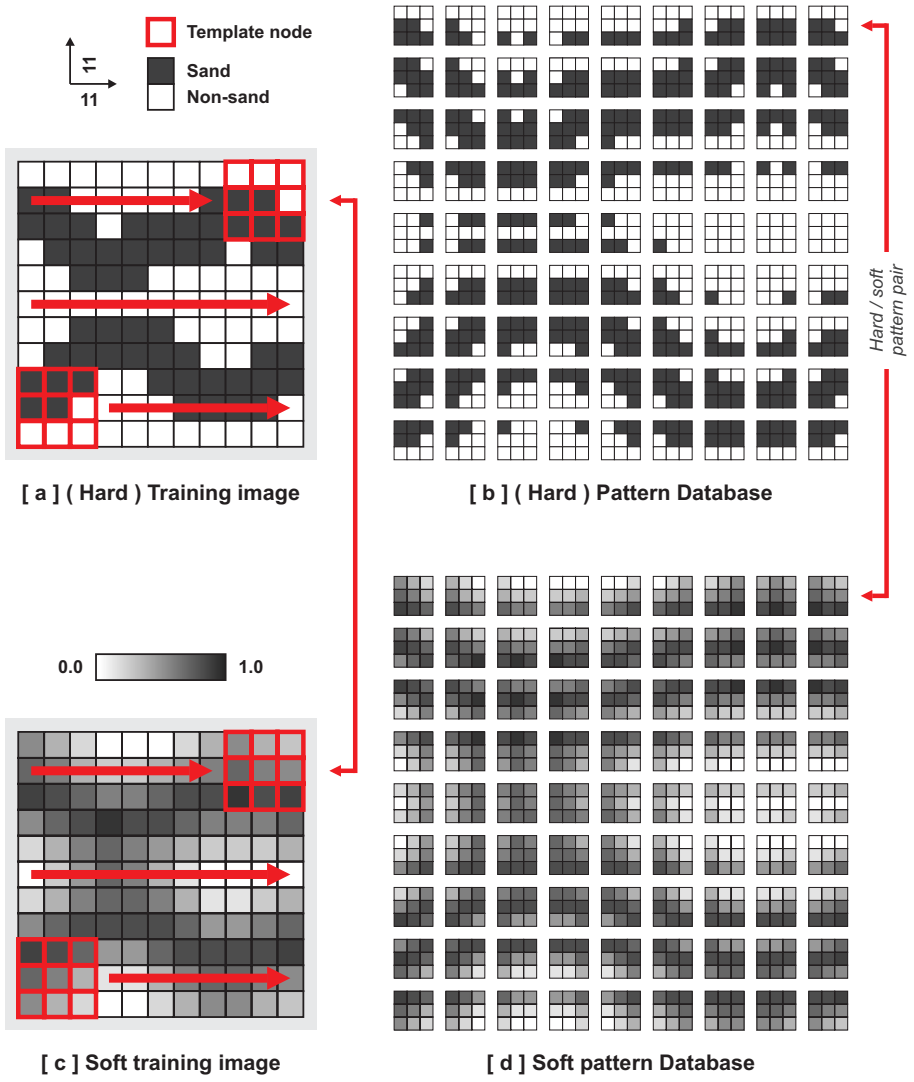


Figure 6.17: The pattern database $\mathbf{patdb_T}$ now consists of both hard patterns $\mathbf{pat}^k_\mathbf{T}$ and soft patterns $\mathbf{spat}^k_\mathbf{T}$, linked together as pairs.

When performed in a multiple-grid setting, for any grid $\mathbf{G}_{re}^{g}$, the preprocessing algorithm scans for dual hard patterns $\mathbf{pat}_{\widetilde{\mathbf{T}}^g}^{k}$ but not for dual soft patterns. In other words, for multiple-grid simulation, the preprocessing algorithm extracts a total of three patterns: the hard primal pattern $\mathbf{pat}_{\mathbf{T}^g}^{k}$, the hard dual pattern $\mathbf{pat}_{\widetilde{\mathbf{T}}^g}^{k}$ and the soft primal pattern $\mathbf{spat}_{\mathbf{T}^g}^{k}$, all extracted from the same location $\mathbf{u}$ and associated with the same database index k.

6.2.2 Conditional Simulation (for Single and Multiple-grid)

Once the combined pattern database $\mathbf{patdb_T}$ is obtained, the conditional, single-grid simulation proceeds as follows: At node $\mathbf{u}$, both the data event $\mathbf{dev_T}(\mathbf{u})$ and the soft data event $\mathbf{sdev_T}(\mathbf{u})$ are obtained. Then, the algorithm maximizes,

$$s\left\langle \cdot,\cdot \right\rangle = s_h\left\langle \mathbf{dev_T}(\mathbf{u}), \mathbf{pat}_{\mathbf{T}}^{k} \right\rangle + s_s\left\langle \mathbf{sdev_T}(\mathbf{u}), \mathbf{spat}_{\mathbf{T}}^{k} \right\rangle \tag{6.2}$$

where $s_h\left\langle \cdot,\cdot \right\rangle$ is the similarity of the data event $\mathbf{dev_T}(\mathbf{u})$ to a hard pattern $\mathbf{pat}_{\mathbf{T}}^{k}$ and $s_s\left\langle \cdot,\cdot \right\rangle$ is the similarity of the soft data event $\mathbf{sdev_T}(\mathbf{u})$ to the soft pattern $\mathbf{spat}_{\mathbf{T}}^{k}$, which shares the same database index k as the hard pattern $\mathbf{pat}_{\mathbf{T}}^{k}$. In other words, the algorithm now considers both the data event $\mathbf{dev_T}(\mathbf{u})$ and the soft data event $\mathbf{sdev_T}(\mathbf{u})$ jointly for finding the most similar pattern $\mathbf{pat}_{\mathbf{T}}^{*}$.

After finding a most similar pattern $\mathbf{pat}_{\mathbf{T}}^{*}$, the simulation proceeds as before, i.e. $\mathbf{pat}_{\mathbf{T}}^{*}$ is pasted on to the realization $\mathbf{re}$ and the next node $\mathbf{u}$ along the random path is visited. Figure 6.18 illustrates this process using the pattern database of Figure 6.17. In the figure, whenever a data event $\mathbf{dev_T}(\mathbf{u})$ is completely uninformed, only the soft data event $\mathbf{sdev_T}(\mathbf{u})$ is considered for similarity calculations, i.e. the hard similarity $s_h\left\langle \cdot,\cdot \right\rangle$ of Equation 6.2 is skipped.

Conditional dual template (multiple-grid) simulation is performed similar to the above conditional single-grid simulation with the exception of pasting of the dual most similar pattern $\mathbf{pat}_{\widetilde{\mathbf{T}}^g}^{*}$ instead of the primal most similar pattern $\mathbf{pat}_{\mathbf{T}^g}^{*}$. In other words, conditioning to soft data only changes the similarity criterion of dual template simulation of Section 4.2.2 to Equation 6.2, i.e. instead of maximizing $s_h\left\langle \cdot,\cdot \right\rangle$, it is $s_h\left\langle \cdot,\cdot \right\rangle + s_s\left\langle \cdot,\cdot \right\rangle$ which is maximized. The rest of the algorithm remains the same.

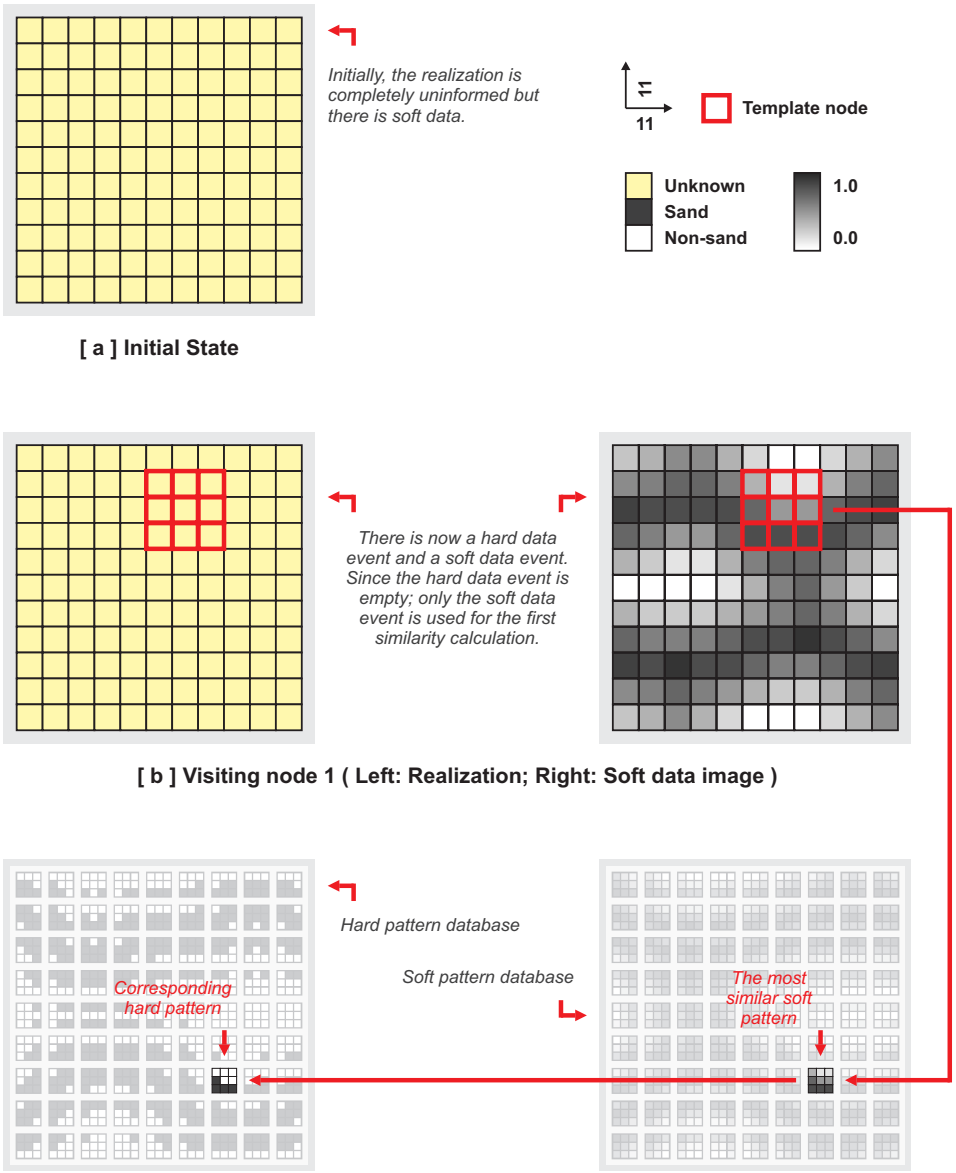


Figure 6.18: Application of soft data conditioning to a 11×11 realization using the soft data of Figure 6.15, the training image of Figure 6.16 and a 3×3 template in a single-grid setting. The figure continues on the next page as Figure 6.19.

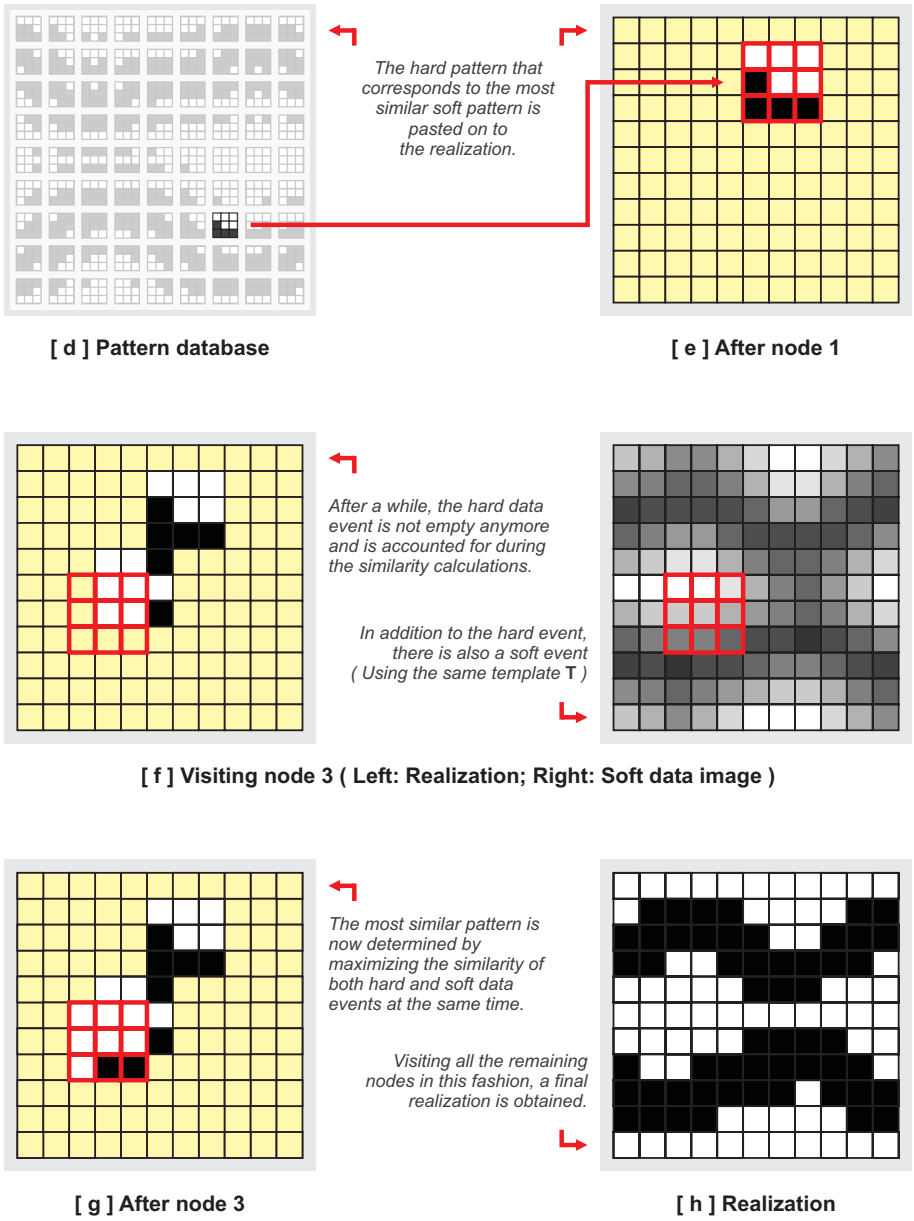


Figure 6.19: Continuation of Figure 6.18 from the previous page. Application of soft data conditioning to a 11×11 realization using the soft data of Figure 6.15, the training image of Figure 6.16 and a 3×3 template in a single-grid setting.

When there is hard data in addition to soft data, the first database lookup of Section 6.1 (preconditioning) remains the same and the combined similarity criterion of Equation 6.2 is used only for the second lookup. In other words, the algorithm first preconditions to hard data ignoring the soft data and only if several $\mathbf{pat}_{\mathbf{T}}^{k}$ fulfills the hard conditioning criterion $s\left\langle \mathbf{hdev}_{\mathbf{T}}(\mathbf{u}), \mathbf{pat}_{\mathbf{T}}^{k} \right\rangle \geq -\epsilon_{hd}$, does it consider the soft data event $\mathbf{sdev}_{\mathbf{T}}(\mathbf{u})$ for finding the most similar pattern $\mathbf{pat}_{\mathbf{T}}^{*}$, i.e. hard data always have priority over soft data.

The values $sti(\mathbf{u})$ of the soft training image **sti** and the soft patterns $\mathbf{spat}_{\mathbf{T}}^{k}$ need not be in the same unit as the values $ti(\mathbf{u})$ of the hard training image **ti** and the hard patterns $\mathbf{pat}_{\mathbf{T}}^{k}$. This is typically the case when seismic information is used as soft data and when $\mathbf{F}^{*}$ is a geophysical forward model. In such a case, the combined similarity of Equation 6.2 might result in biased similarities, since for a node $\mathbf{u}$, the measure $s_s\langle\cdot,\cdot\rangle$ might be several orders of magnitude greater than $s_h\langle\cdot,\cdot\rangle$, effectively dominating the similarity calculation and shadowing the contribution of the previously calculated **re** nodes to the combined similarity. To prevent this, the similarities $s_h\langle\cdot,\cdot\rangle$ and $s_s\langle\cdot,\cdot\rangle$ are normalized such that both $s_h\langle\cdot,\cdot\rangle$ and $s_s\langle\cdot,\cdot\rangle$ range between the same values; typically, $[-1,0]$, i.e. $d_h\langle\cdot,\cdot\rangle$ and $d_s\langle\cdot,\cdot\rangle$ are normalized to $[0,1]$.

A common situation, especially with seismic information, is to have spatially varying degrees of confidence to soft data. For example, for a particular seismic survey, certain subsurface structures (such as salt domes) might decrease the quality of soft data obtained for certain regions of a reservoir. In such cases, it may be desirable to reflect this varying confidence on to the soft data conditioning process. To account for this, Equation 6.2 is modified as follows:

$$s\langle\cdot,\cdot\rangle = s_h\langle\cdot,\cdot\rangle + \omega_{sd}(\mathbf{u}) \times s_s\langle\cdot,\cdot\rangle \tag{6.3}$$

where $\omega_{sd} \in [0,1]$ is a location-specific weight factor used to reflect the 'confidence' to soft data. Setting $\omega_{sd}(\mathbf{u}) = 1$ amounts to full confidence whereas setting $\omega_{sd}(\mathbf{u}) = 0$ amounts to ignoring the soft information.

6.2.3 2D Examples and Sensitivity Study

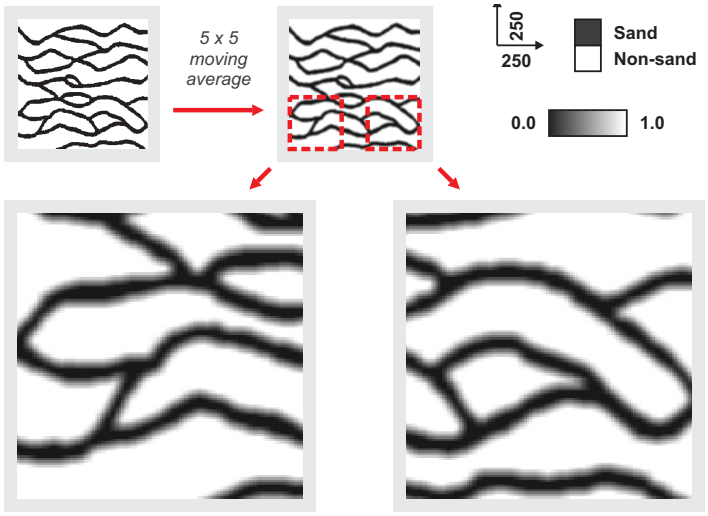
Consider the reference case of Figure 6.20. In the figure, the soft data image is obtained using a 15×15 moving average ($= \mathbf{F}$). Using this reference and the training images of Figure 6.6, several runs are performed to investigate the sensitivity of soft data conditioning to using different forward models ($\mathbf{F}^*$).

Figure 6.20: The reference and the soft data obtained from this reference using a 15×15 moving average ($= \mathbf{F}$).

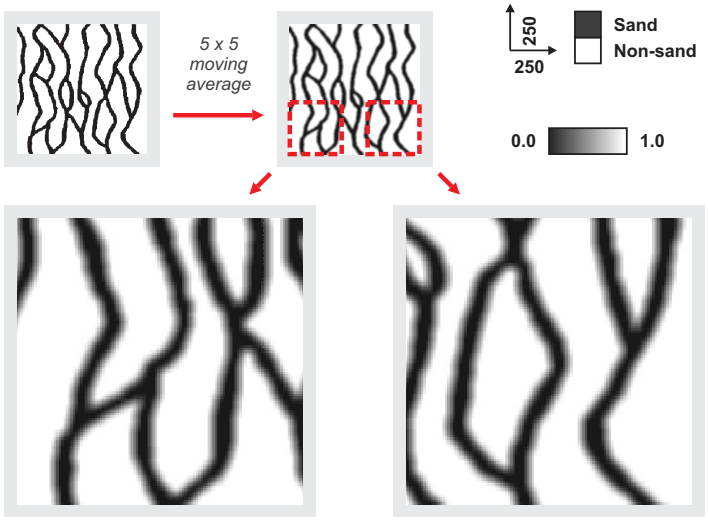
Since conditioning to soft data requires a soft training image in addition to a hard training image, three different forward models are applied to the training images of Figure 6.6 to obtain the corresponding soft training images. The first $\mathbf{F}^*$ is taken as equal to $\mathbf{F}$, i.e. 15×15 moving average. Since for real cases $\mathbf{F}$ is never fully known, 5×5 and 25×25 moving averages are used as additional $\mathbf{F}^*$s to reflect this uncertainty. Figures 6.21, 6.22 and 6.23 show these soft training images.

Figure 6.24 shows the realizations and the E-types obtained using the geologically consistent training image of Figure 6.6a. In Figure 6.24c,d, for $\mathbf{F}^* = \mathbf{F}$, the algorithm successfully integrates the geological information and the soft data resulting in a significant decrease of uncertainty as made evident by the E-type (Figure 6.24d). However, when $\mathbf{F}^* \neq \mathbf{F}$ (Figures 6.24b,f), the uncertainty does not decrease as much.

Since the soft data is exhaustive, i.e. informs the realization everywhere, using a geologically inconsistent training image (and thus a geologically inconsistent soft training image) results in poor conditioning regardless of $\mathbf{F}^*$ (Figure 6.25). Notice

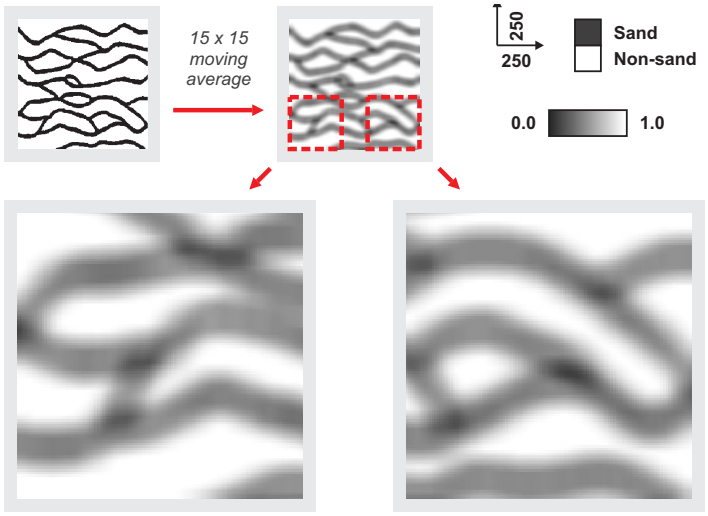


[a] Soft TI from a geologically consistent TI (5 x 5 average)

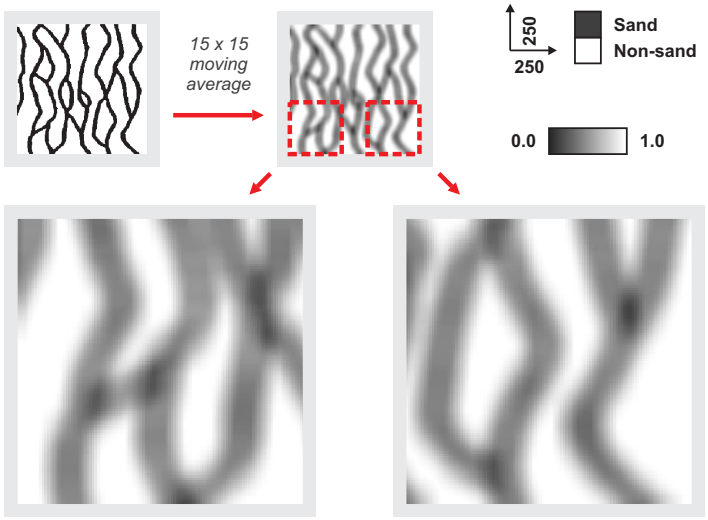


[b] Soft TI from a geologically inconsistent TI (5 x 5 average)

Figure 6.21: The soft training images (obtained using 5×5 moving average $= \mathbf{F}^*$) corresponding to the hard training images of Figure 6.6.

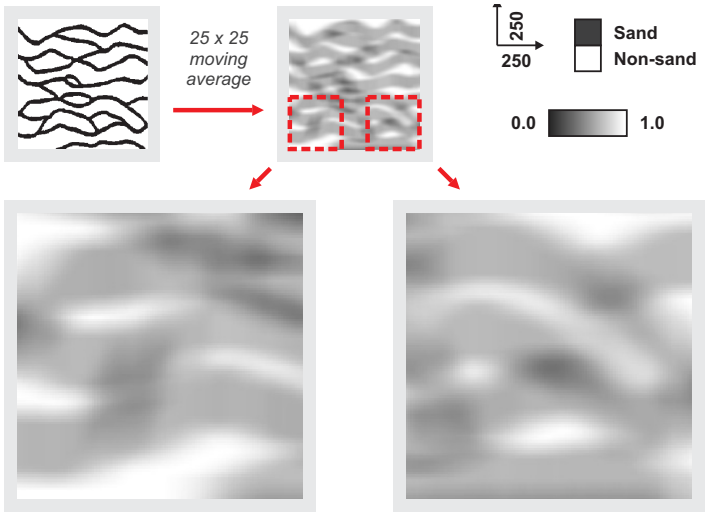


[a] Soft TI from a geologically consistent TI (15 x 15 average)

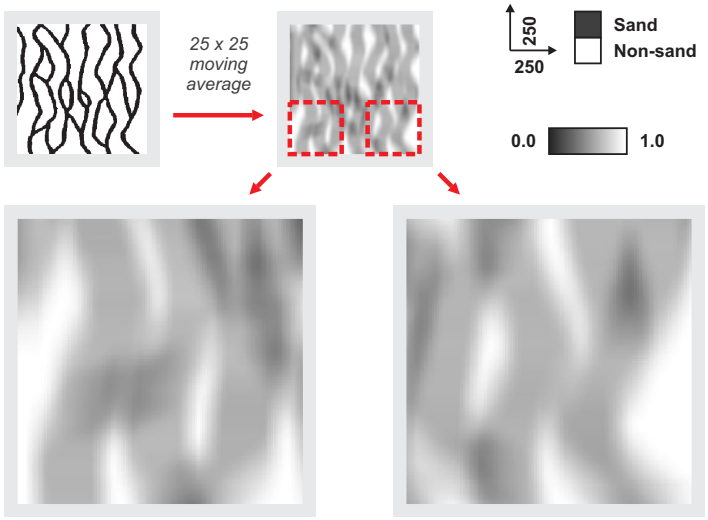


[b] Soft TI from a geologically inconsistent TI (15 x 15 average)

Figure 6.22: The soft training images (obtained using 15×15 moving average $= \mathbf{F}^*$) corresponding to the hard training images of Figure 6.6.



[a] Soft TI from a geologically consistent TI (25 x 25 average)



[b] Soft TI from a geologically inconsistent TI (25 x 25 average)

Figure 6.23: The soft training images (obtained using 25×25 moving average $= \mathbf{F}^*$) corresponding to the hard training images of Figure 6.6.

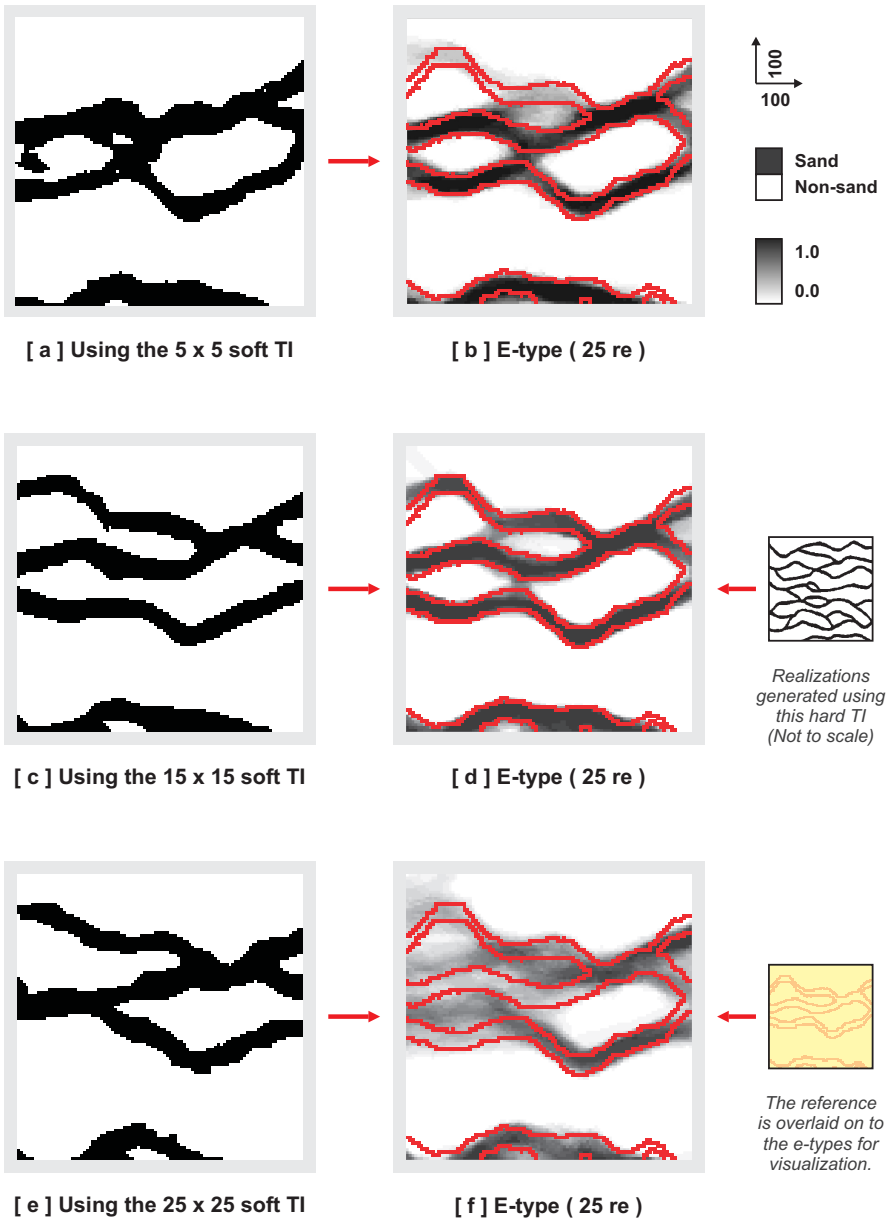


Figure 6.24: Conditioning to soft data using a geologically consistent training image and three different corresponding soft training images [$n_{\mathbf{T}} = 7 \times 7$; $n_g = 3$].

that, as different from Figure 6.10 (hard data conditioning to geobodies using a geologically inconsistent training image), the realizations of Figure 6.25 reproduce the small-scale training image patterns better. In the presence of a geologically inconsistent training image, hard data conditioning typically causes small-scale pattern reproduction problems whereas soft data conditioning causes large-scale pattern reproduction problems. This difference is due to the different methods used to condition to hard and soft data. With hard data, the algorithm always tries to exactly reproduce the data and thus might allow patterns on the realization that do not exist in the training image (pattern database) to achieve that aim. However, with soft data, the algorithm always copies existing training image patterns on to the realization, making the overall pattern reproduction of the realization more consistent with the training image rather than the soft data.

Last, Figures 6.26 and 6.27 show conditioning to both hard and soft data using the 10 and 100 hard data points of Figure 6.5b,c. In the figures, as expected, using hard data decreases uncertainty even when $\mathbf{F}^* \neq \mathbf{F}$.

6.2.4 A Synthetic, 3D Example

Figures 6.28, 6.29 and 6.30 show the application of soft data conditioning to a 3D soft data and a training image. For this example, the soft data (Figure 6.28b) is obtained by forward simulation of seismic amplitude traces using a normal incidence 1D convolution model with Fresnel zone lateral averaging ($= \mathbf{F}^*$) on the binary (sand/non-sand) reference case. See Wu and Journel (2004) for details. The same forward model is applied to the training image to obtain the soft training image (Figure 6.29).

Figure 6.30a shows the conditional realization generated using a $11 \times 11 \times 3$ template, 3 multiple-grids and the Manhattan distance. The realization conditions well to soft data but pattern reproduction is somewhat degraded as made evident by the disconnected channel pieces. Figure 6.30b shows another realization obtained using the same settings but also proximity transforms. Since using proximity transforms entails enhancing pattern reproducing, the disconnected channel pieces of Figure 6.30a no longer appear in the realization.

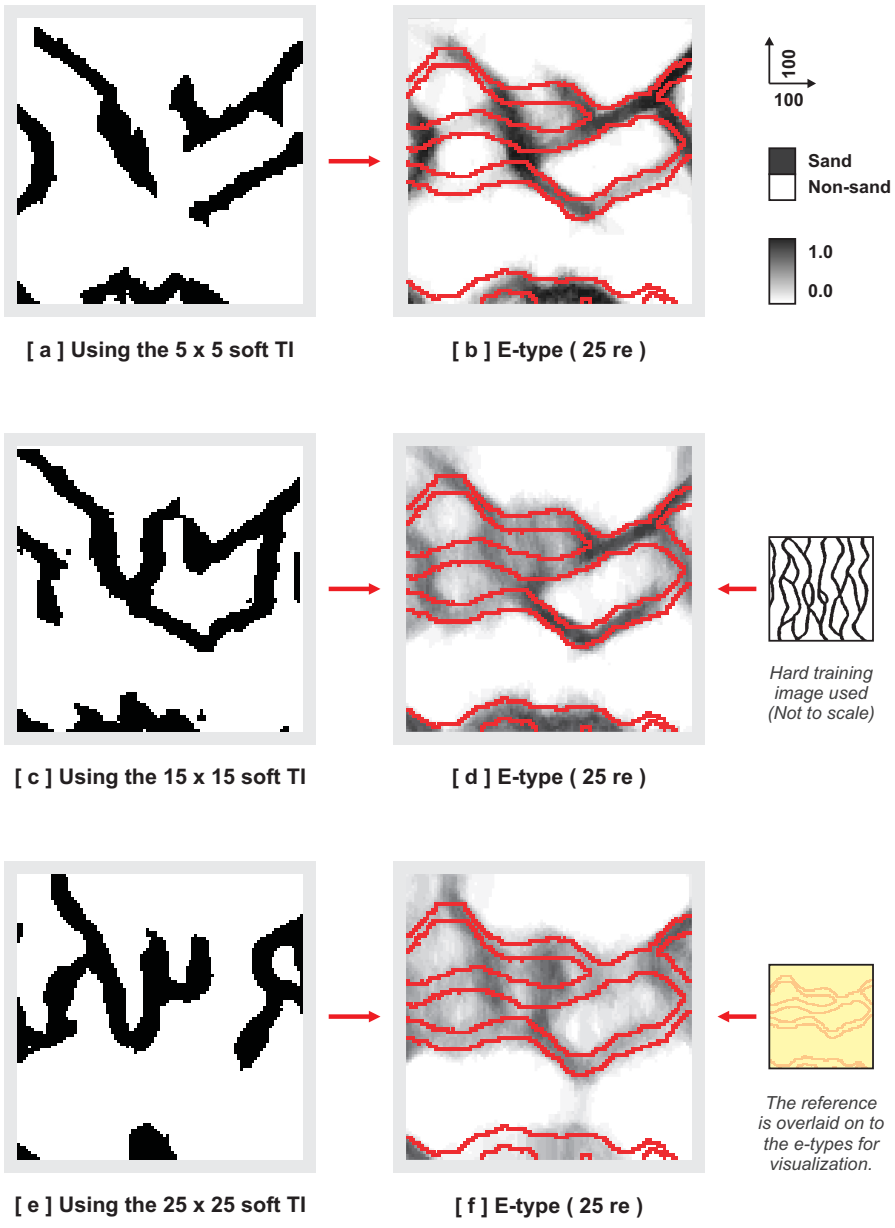


Figure 6.25: Conditioning to soft data using a geologically inconsistent training image and three different corresponding soft training images [$n_{\mathbf{T}} = 7 \times 7$; $n_g = 3$].

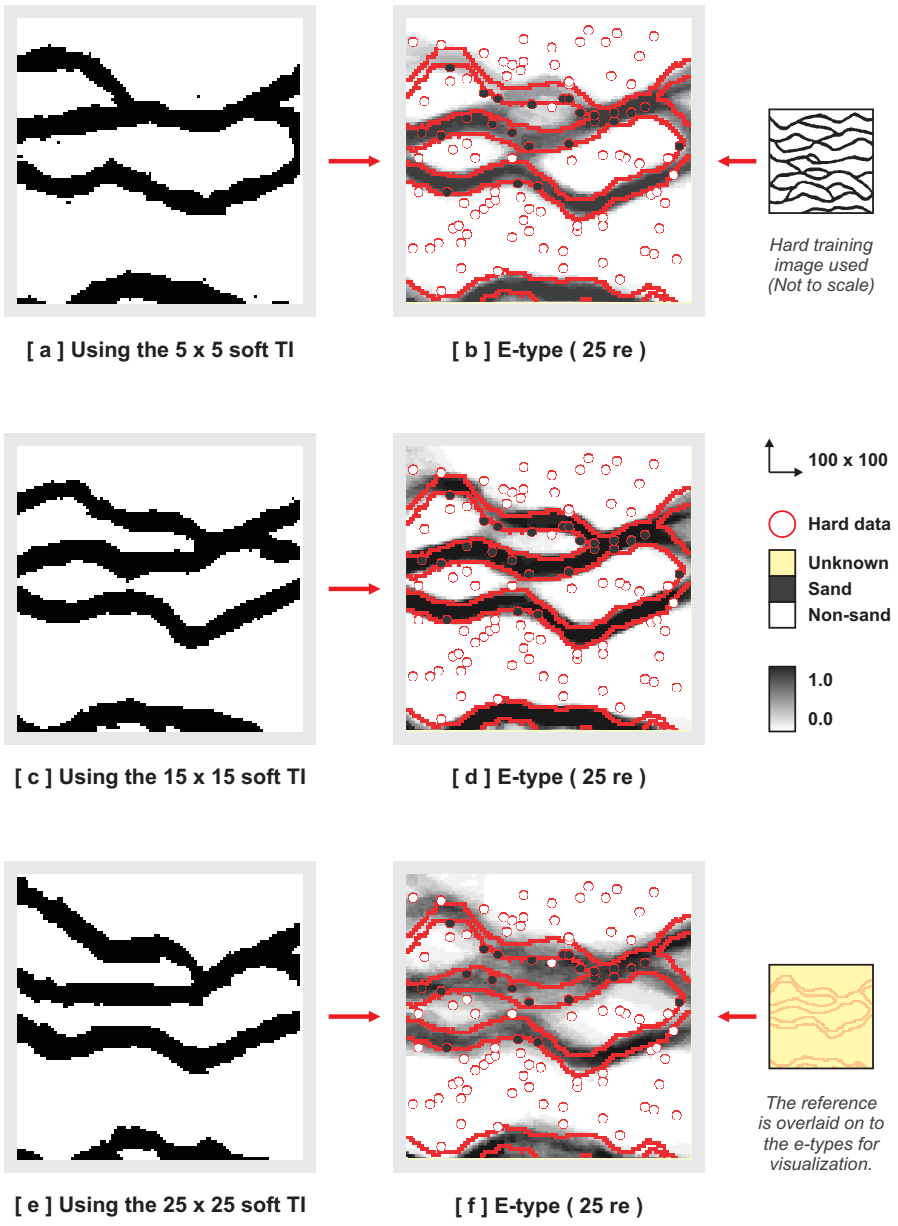


Figure 6.26: Conditioning to both hard and soft data using the 10 hard data points of Figure 6.5b [$n_{\mathbf{T}} = 7 \times 7$; $n_g = 3$].

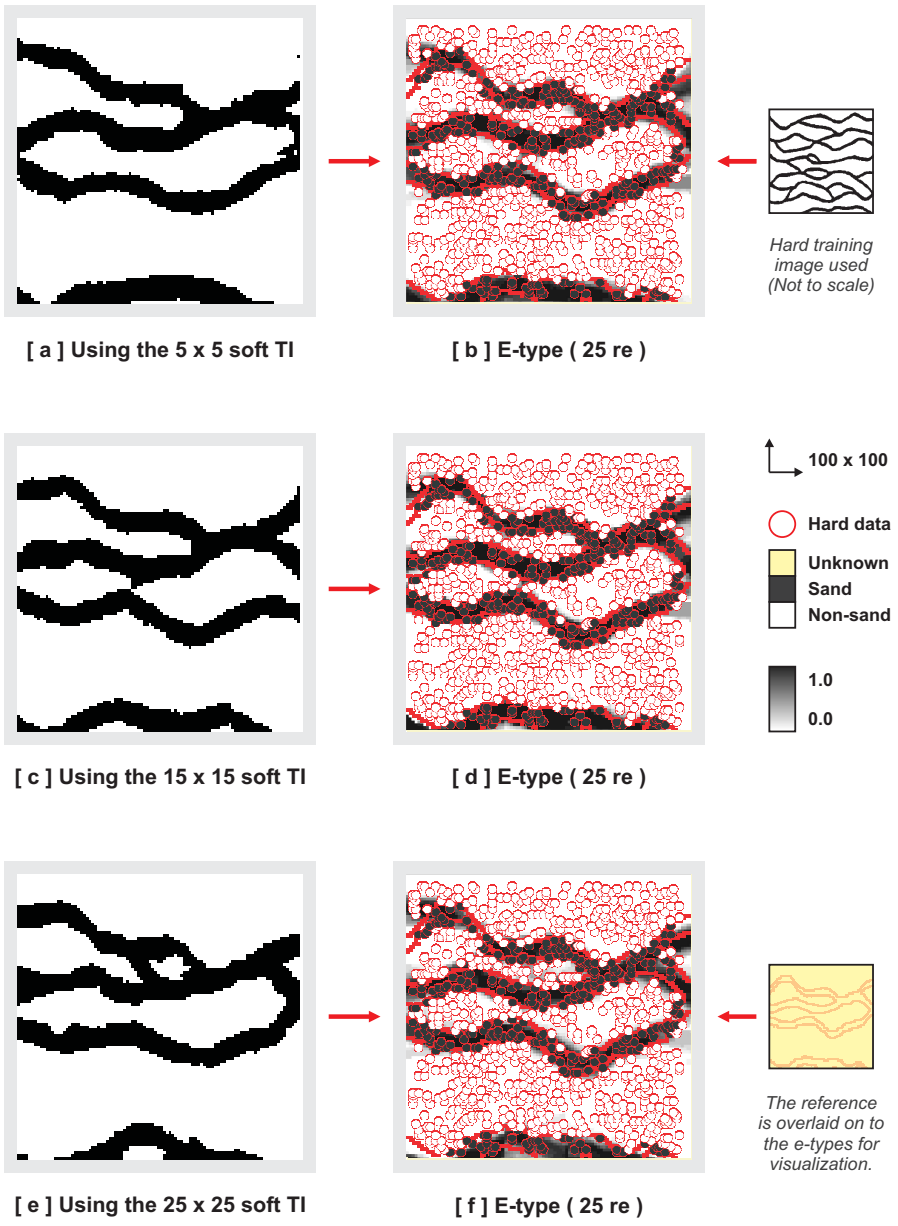
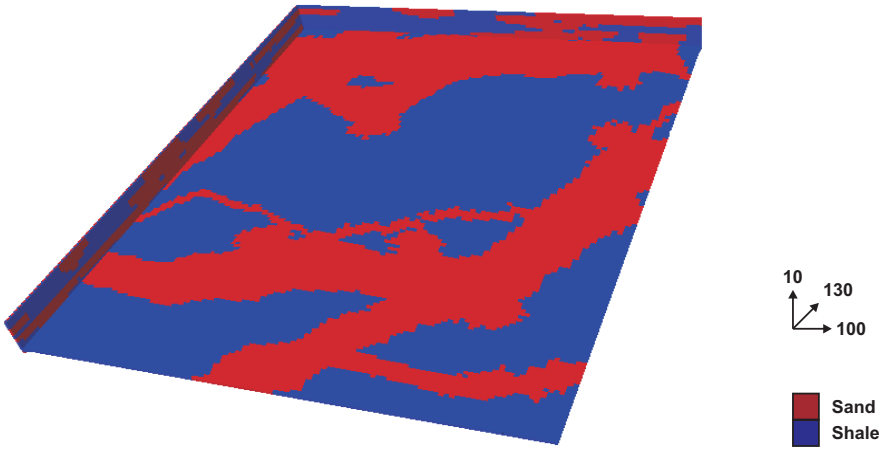
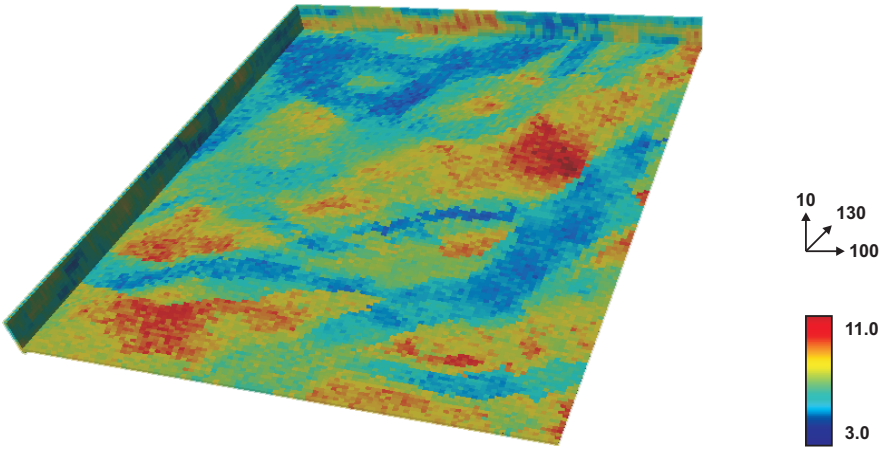


Figure 6.27: Conditioning to both hard and soft data using the 100 hard data points of Figure 6.5c [$n_{\mathbf{T}} = 7 \times 7$; $n_g = 3$].

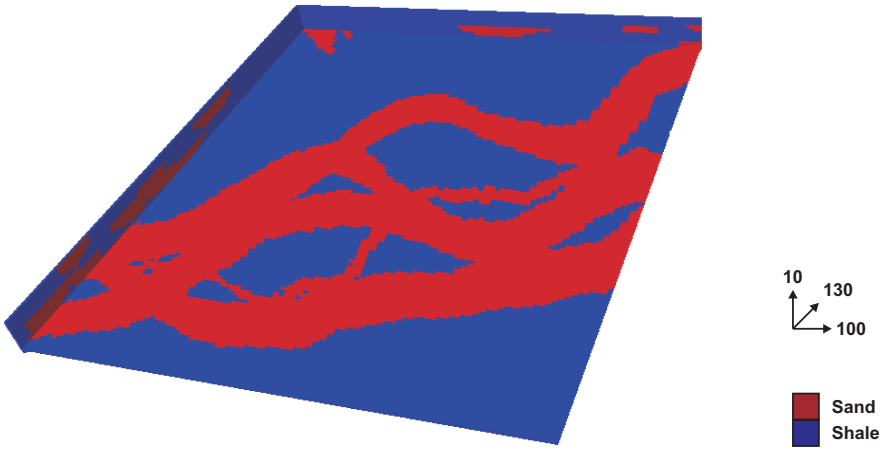


[a] Reference

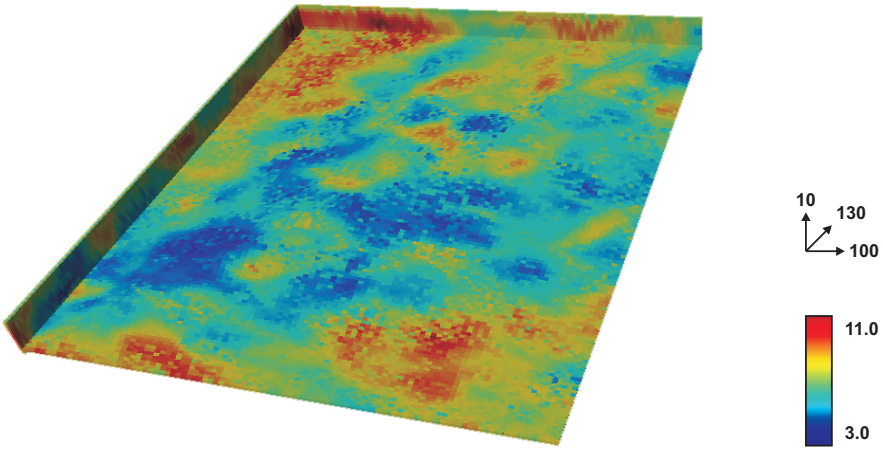


[b] Soft data (Synthetic seismic)

Figure 6.28: Reference for 3D soft data conditioning and the soft data (synthetic seismic) corresponding to this reference.

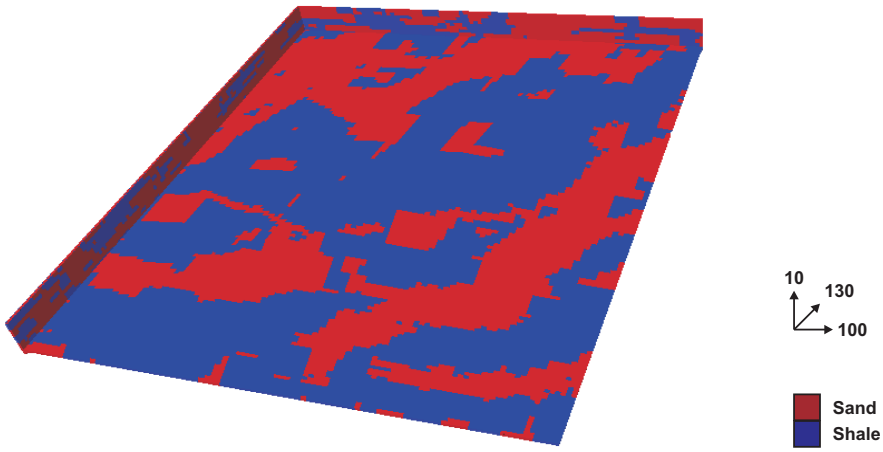


[a] Training image

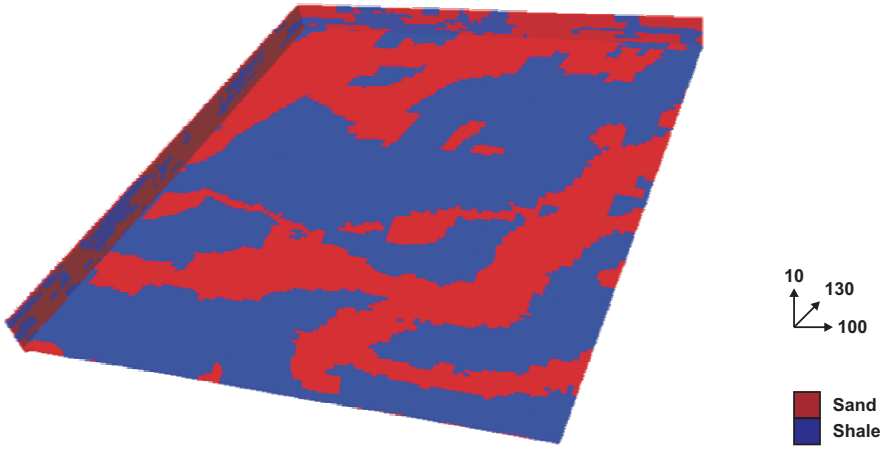


[b] Soft training image

Figure 6.29: The training image and the soft training image obtained using a forward model $\mathbf{F}^*$ similar to that of the soft data (Figure 6.28).



[a] Conditional realization using Manhattan similarity



[b] Conditional realization using proximity transforms

Figure 6.30: Two conditional realizations obtained using the soft data of Figure 6.28 and the training image pair of Figure 6.29. The first realization has disconnect channel pieces and uses Manhattan similarity. The second realization uses proximity transforms to improve the similarity calculations resulting in better conditioning.

Chapter 7

Regions and Training Image Transformations

Consider the reference image shown in Figure 7.1. The figure shows a deltaic channel reservoir with significant non-stationarity: the orientation and the width of the channels vary from one area of the reservoir to another. Modeling such a reservoir requires reproducing this pattern trend in the simulation. This chapter explains how the simulation algorithm is modified to account for such local 'morphology data'.

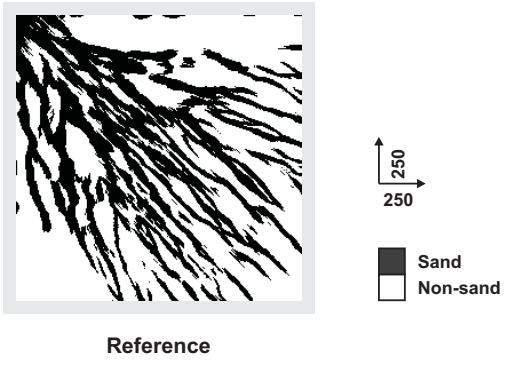


Figure 7.1: A 250×250, binary (sand/non-sand) reference.

7.1 Region Simulation

Simulation constrained by locally varying pattern geometry requires using different training images for different areas of the reservoir.

To achieve this, the reservoir is first divided into several subregions $\mathbf{R}_r$. The notation $\mathbf{R}_r$ denotes a subset of the simulation grid $\mathbf{G}_{re}$, i.e. $\mathbf{R}_r \subset \mathbf{G}_{re}$ with r as the region index, $r = 1, \ldots, n_{\mathbf{R}}$ and $n_{\mathbf{R}}$ is the total number of regions. Regions are not allowed to overlap, i.e. $\mathbf{u}_\alpha \neq \mathbf{u}_\beta, \forall \mathbf{u}_\alpha \in \mathbf{R}_\alpha, \forall \mathbf{u}_\beta \in \mathbf{R}_\beta$ and the union of all regions must be equal to the grid itself, i.e. $\mathbf{R}_1 \cap \mathbf{R}_2 \cap \ldots \cap \mathbf{R}_r \cap \ldots \cap \mathbf{R}_{n_{\mathbf{R}}} = \mathbf{G}_{re}$. Nodes $\mathbf{u}$ of a region $\mathbf{R}_r$ need not be within the vicinity of each other, i.e. a region can consist of disjoint neighborhoods.

The decision of how the reservoir should be divided into regions is based on the morphological parameters associated with each region. For example, for the reservoir shown in Figure 7.1, this information might consist of the orientation and the width of the sand (channel) facies. Figure 7.2 shows this. In the figure, the reservoir is first divided into 10 regions, i.e. $n_{\mathbf{R}} = 10$. Then, each region is associated with a different channel width (Figure 7.2c) and a different channel orientation (Figure 7.2d). In the case of Figure 7.2, all the data is derived from the reference. In real cases, this information is typically derived from seismic and geological interpretation.

Once the regional morphology data is determined, each region $\mathbf{R}_r$ is associated with a different training image $\mathbf{ti}_r$ describing the different patterns of each region. Ideally, each such training image is obtained through a different unconditional object-based or process-based simulation. Alternatively, a single training image $\mathbf{ti}$ can be used as a "master" geological concept. Then, this master training image is transformed for each region such that each transformed training image $\mathbf{ti}_r$ reflects the desired morphological characteristics of that region (Caers and Zhang, 2004). Section 7.2 of this chapter explains how such a master training image can be transformed. Figures 7.3, 7.4, 7.5 show these different regional training images $\mathbf{ti}_r$ as transformed from a single master training image.

After associating each region $\mathbf{R}_r$ with a specific training image $\mathbf{ti}_r$, the simulation proceeds with the algorithm described in the previous chapters. First, each training

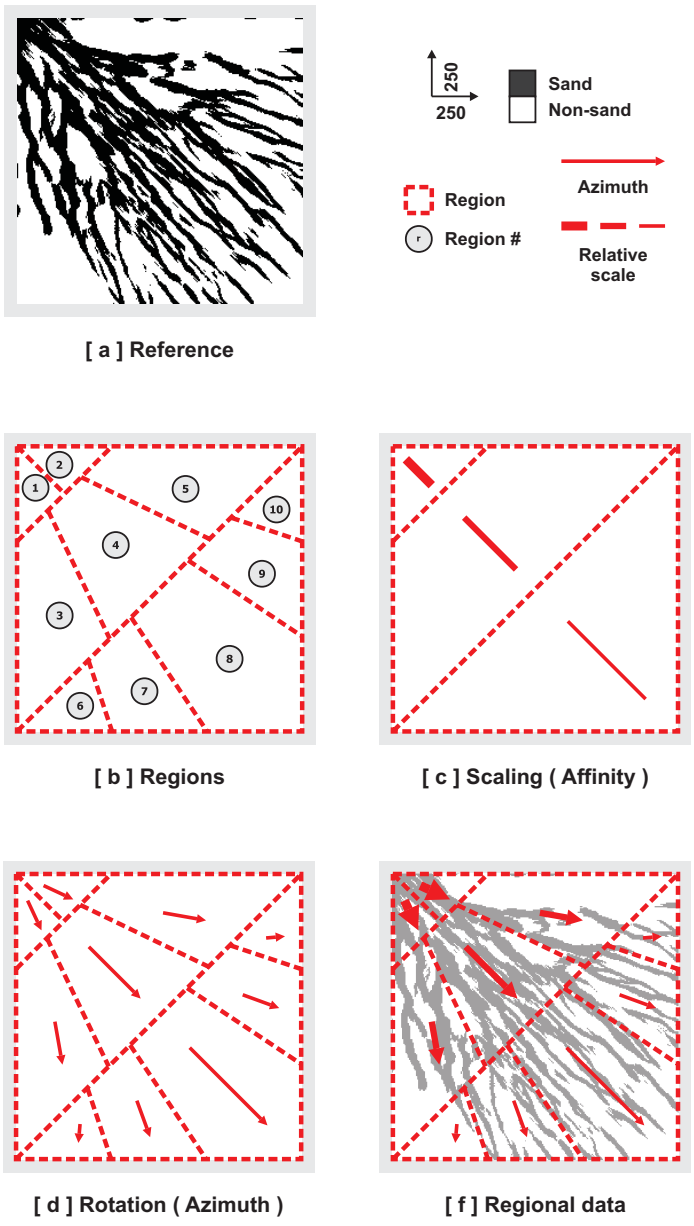


Figure 7.2: The morphological information is described through 10 regions with varying affinities (scaling factors) and azimuths (angles).

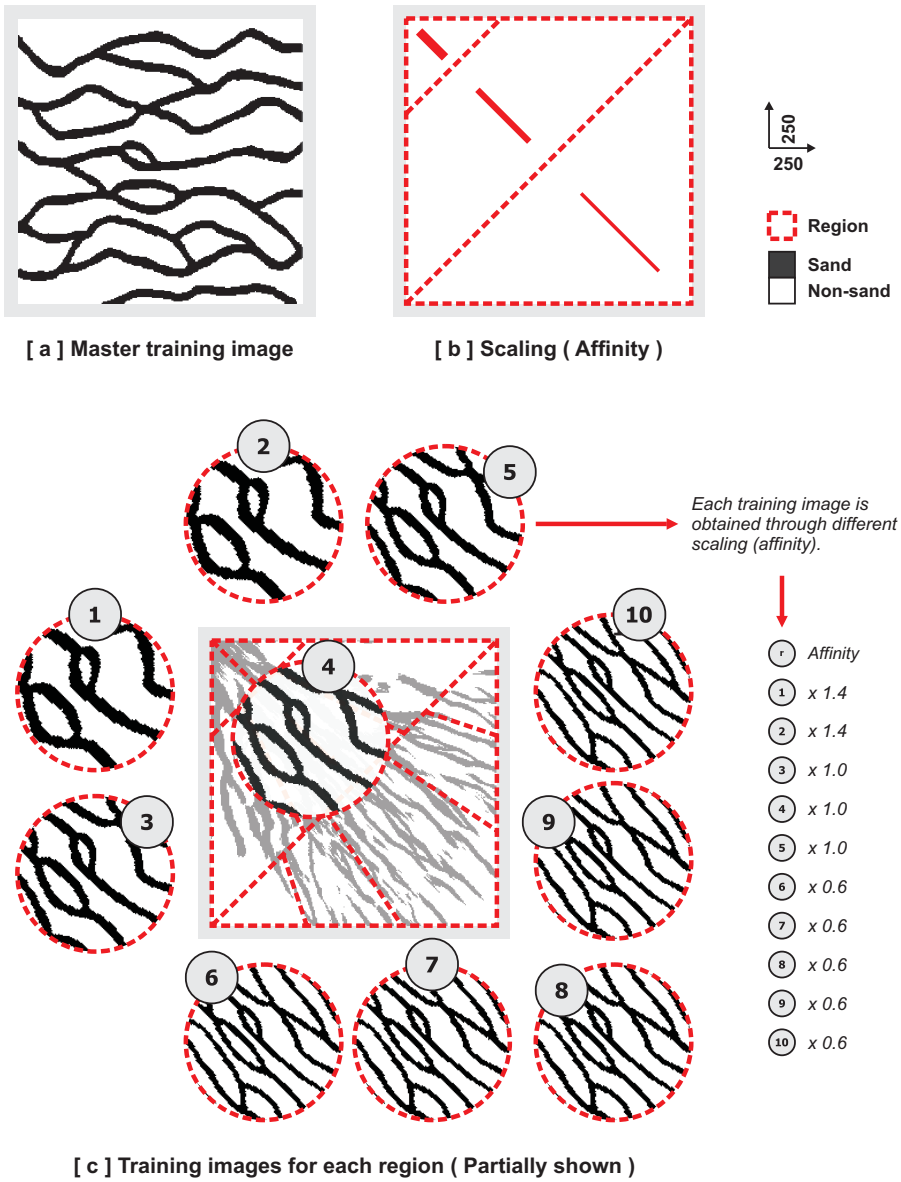


Figure 7.3: Regional training images for scaling. Each training image is obtained using the master training image [a] by applying the scaling transformation described in Section 7.2.

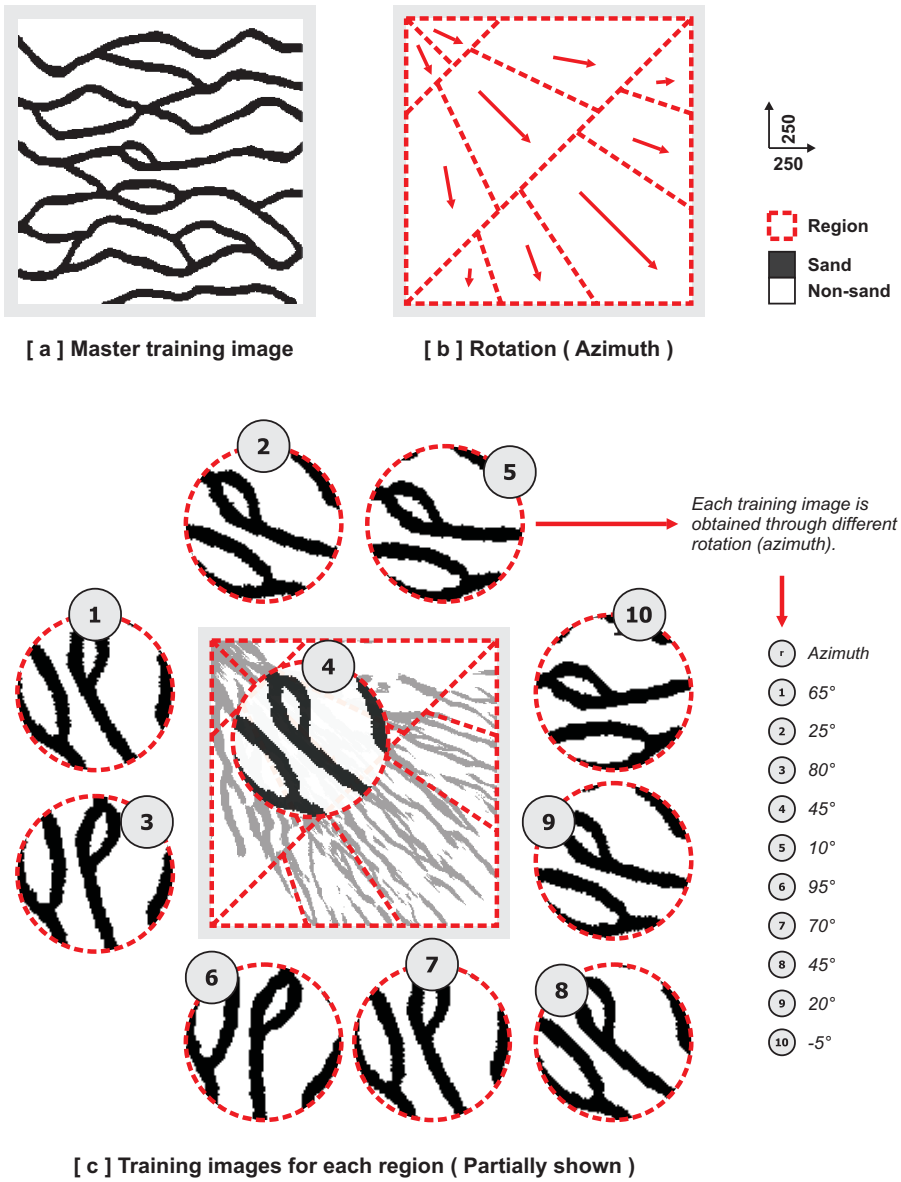


Figure 7.4: Regional training images for rotation. Each training image is obtained using the master training image [a] by applying the rotation transformation described in Section 7.2.

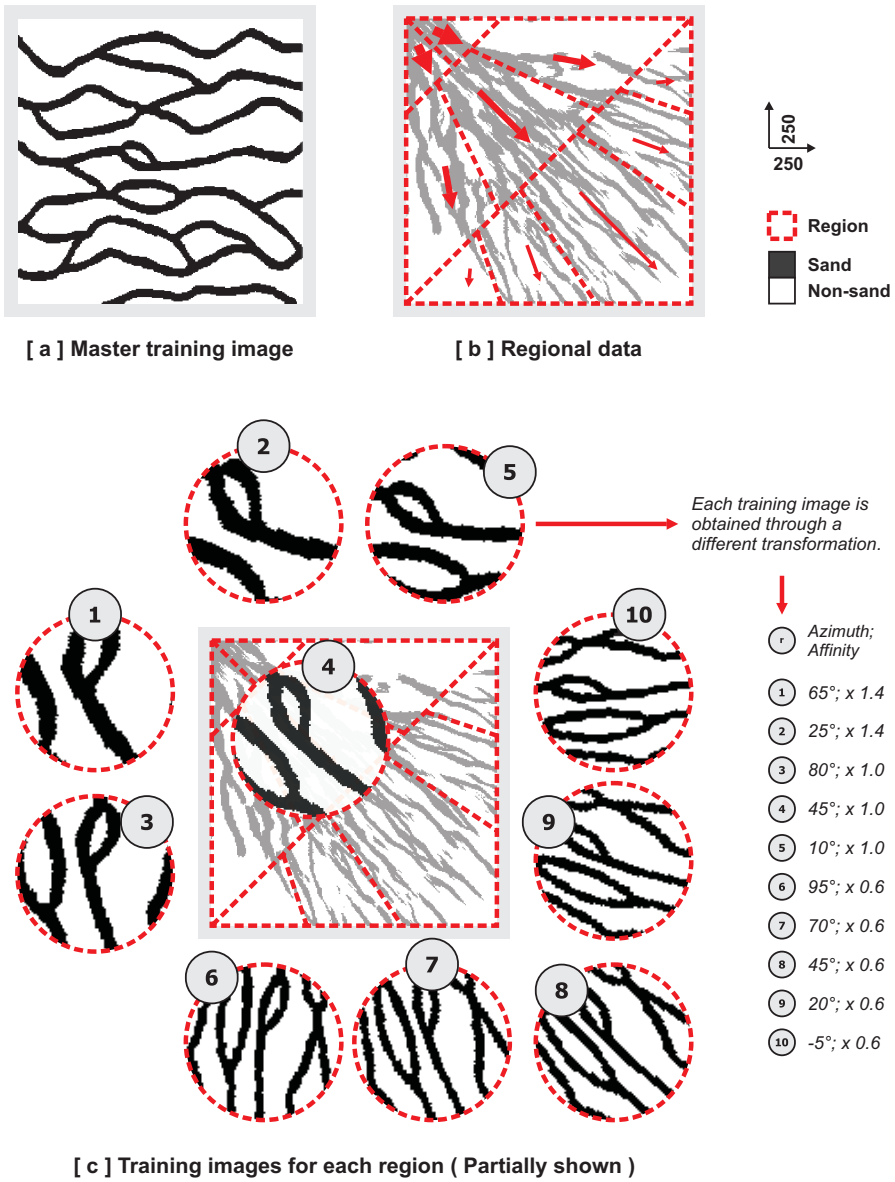


Figure 7.5: Regional training images for combined transformation. Each training image is obtained using the master training image [a] by applying the transformations described in Section 7.2.

image $\mathbf{ti}_r$ is preprocessed and the patterns $\mathbf{pat}_{\mathbf{T}}^{r,k}$ of that training image $\mathbf{ti}_r$ are stored in a pattern database $\mathbf{patdb}_{\mathbf{T},r}$, i.e. each training image has a separate pattern database associated with it. Then, when visiting node $\mathbf{u}$ with $\mathbf{u} \in \mathbf{R}_r$, the algorithm simply uses the corresponding pattern database $\mathbf{patdb}_{\mathbf{T},r}$ for the similarity calculations. The rest of the algorithm remains the same. In other words, the most similar pattern $\mathbf{pat}_{\mathbf{T}}^{*}$ for node $\mathbf{u}$ is found via maximizing $s\left\langle \mathbf{dev}_{\mathbf{T}}(\mathbf{u}), \mathbf{pat}_{\mathbf{T}}^{r,k} \right\rangle$ where patterns $\mathbf{pat}_{\mathbf{T}}^{r,k}$ belong to the pattern database $\mathbf{patdb}_{\mathbf{T},r}$. Since $\mathbf{patdb}_{\mathbf{T},r}$ only describes the patterns of region $\mathbf{R}_r$, when pasting the nodes $\mathbf{u} + \mathbf{h}_\alpha$ of the most similar pattern $\mathbf{pat}_{\mathbf{T}}^{*}$ on to the data event $\mathbf{dev}_{\mathbf{T}}(\mathbf{u})$, only the nodes $\mathbf{u}+\mathbf{h}_\alpha$ that are within the region $\mathbf{R}_r$ are updated, i.e. the update occurs only for $\mathbf{u}+\mathbf{h}_\alpha \in \mathbf{R}_r$ and the remaining nodes $\mathbf{u} + \mathbf{h}_\alpha \notin \mathbf{R}_r$ of the $\mathbf{dev}_{\mathbf{T}}(\mathbf{u})$ are not modified.

Figure 7.6 shows the application of region simulation using the regional data of Figure 7.2 and the regional training images shown in Figures 7.3, 7.4 and 7.5.

7.2 Training Image Transformations

The example given in the previous section exhibits different pattern morphology for each region of the reservoir. But, these patterns are all described through a single master training image. This section explains how a master training image can be transformed into another training image such that the new training image reflects the desired morphological characteristics of the region it represents.

The image processing literature lists several morphological image transformations (Gonzalez and Woods, 2002) that may be useful in a variety of cases for reservoir modeling. In this section, only the most fundamental transformations, namely, scaling and rotation, are described.

Rotation and scaling of images are special cases for a general, geometric transformation known as the "affine transformation". An affine transformation is essentially a class of linear 3D geometric transformations which maps the values $ti(\mathbf{u})$ of the input (master) training image $\mathbf{ti}$ on the output (regional) training image $\mathbf{ti}_r$ by applying a sequence of translation, rotation, scaling and/or shearing (i.e. non-uniform scaling in some directions) operations. Since it is always possible to define a 3D affine

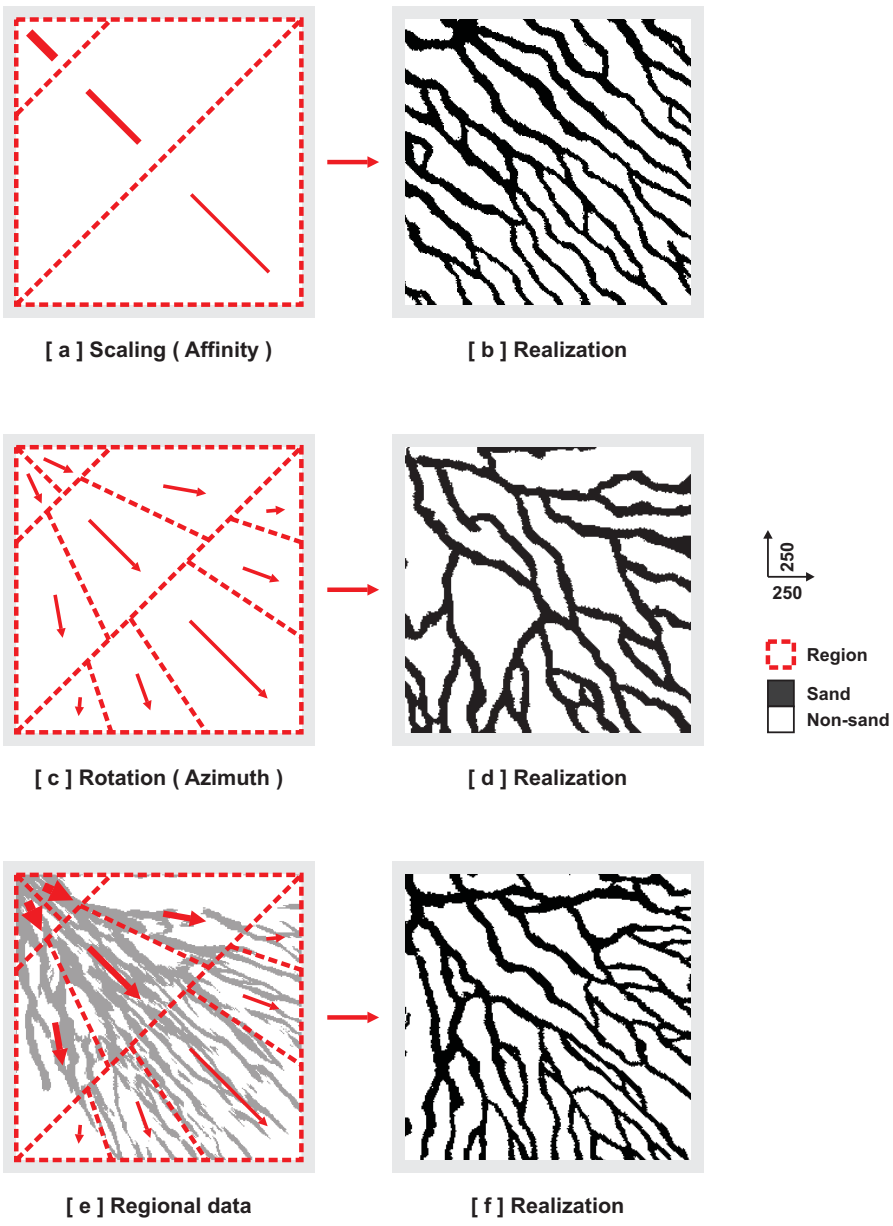


Figure 7.6: Realizations constrained by regional information. The final result is obtained using the regional training images of Figures 7.3, 7.4 and 7.5.

transformation about a line passing through the origin in terms of a series of 2D affine transformations (Gonzalez and Woods, 2002), this section describes only 2D rotation and scaling.

A 2D rotation of node $\mathbf{u} \in \mathbf{G}_{ti}$ about the z-axis can be written as:

$$\begin{aligned} i' &= \cos(\alpha) \times i - \sin(\alpha) \times j \\ j' &= \sin(\alpha) \times i + \cos(\alpha) \times j \\ k' &= k \end{aligned} \tag{7.1}$$

where the original node is $\mathbf{u} = \{i, j, k\}$, the transformed node is $\mathbf{u}' = \{i', j', k'\}$ and α is the rotation angle measured in an anti-clockwise direction around z-axis. Then, the rotation of a training image $\mathbf{ti}$ to a transformed training image $\mathbf{ti}_r$ is performed by setting $ti_r(\mathbf{u}') = ti_r(\mathbf{u})$ for all $\mathbf{u} \in \mathbf{G}_{ti}$. Figure 7.7 shows this.

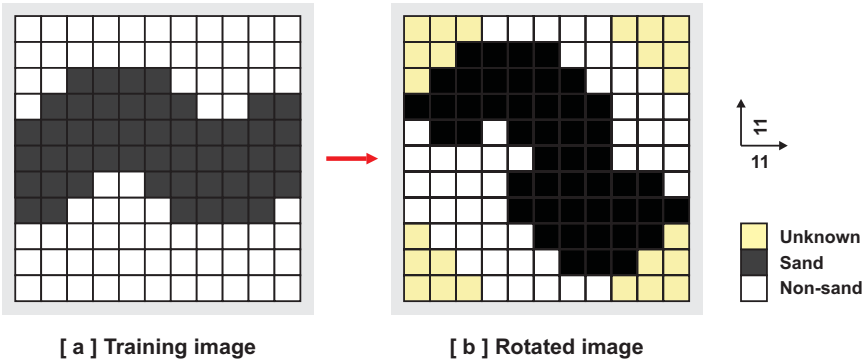


Figure 7.7: Rotation by $\alpha = 45°$ clockwise. Rotation of a training image may introduce local artifacts and unknown nodes.

During a rotation, some locations $\mathbf{u}' = \{i', j', k'\}$ may not fall on the Cartesian grid $\mathbf{G}_{ti,r}$. In such a case, the calculated location $\mathbf{u}'$ is relocated to the nearest grid node. Since this relocation amounts to modifying the image, the rotation is typically performed in reverse to minimize the modification, i.e. instead of finding the new node $\mathbf{u}'$ given an original node $\mathbf{u}$, one finds the original node $\mathbf{u}$ for all nodes $\mathbf{u}' \in \mathbf{G}_{ti,r}$. In other words, Equation 7.1 is used to find the corresponding $\mathbf{u}$ for a given $\mathbf{u}'$ with angle α now denoting the rotation in the reverse direction (clockwise).

When performing the above reverse rotation, if the calculated original node falls 'outside' of the grid $\mathbf{G}_{ti}$, $ti_r(\mathbf{u}')$ is set as unknown, i.e. $ti_r(\mathbf{u}') = \chi$ for $\mathbf{u} \notin \mathbf{G}_{ti}$. Chapter 8 explains how such training images containing unknown values are preprocessed.

Scaling of a training image **ti** is performed similar to rotation using the transformation equation:

$$\begin{aligned} i' &= \beta_i \times i \\ j' &= \beta_j \times j \\ k' &= \beta_k \times k \end{aligned} \tag{7.2}$$

where β denotes the scaling factor for each axis, i.e. the transformation is in 3D. Setting $\beta < 1$ for all axes amounts to shrinking the image and setting $\beta > 1$ amounts to expanding. Figure 7.8 illustrates this for a scaling factor β of 0.75 for all three axes.

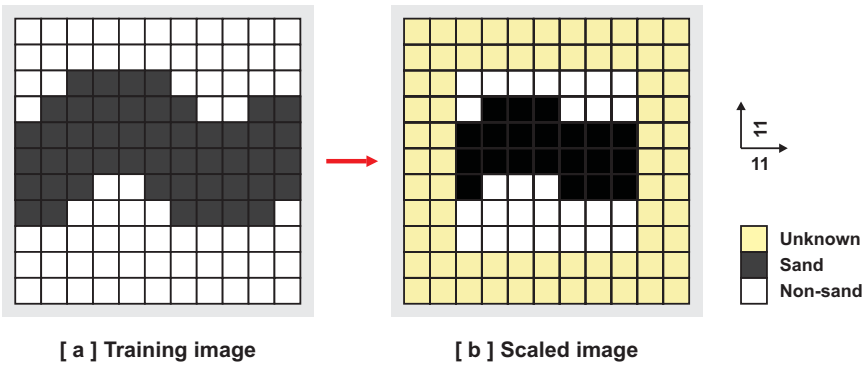


Figure 7.8: Scaling by $\beta = 0.75$ (shrinking) for all axes.

The order in which rotation and scaling are applied to an image is significant, i.e. rotating an image with angle α and then scaling the result (i.e. the rotated image) with factor β is not equivalent to first scaling using β and then rotating using α. In practice, scaling should be performed first; followed by rotation.

Last, it is important to note that, due to the inherent destructive nature of any image transformation when applied on a discrete Cartesian grid $\mathbf{G}$, the result of a transformation should be checked by a human expert prior to simulation. For

example, for the fracture training image of Figure 5.3, rotating the image by $\alpha = 45°$ clockwise to obtain a new training image with a different fracture slant produces unintended results in the form of highly disconnected fracture pieces (Figure 7.9). In such cases, instead of transforming the master training image **ti** to generate regional training images $\mathbf{ti}_r$, one should use an object-based or process-based tool to generate these regional images.

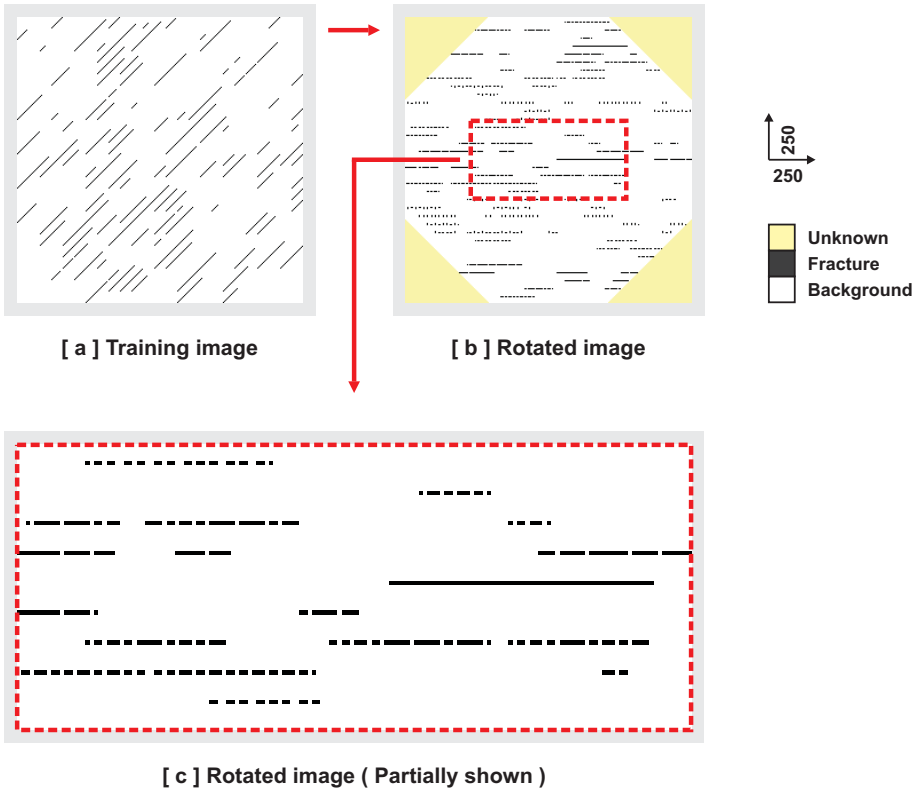


Figure 7.9: Rotation by $\alpha = 45°$ clockwise. In this particular case, simple rotation of the image [a] produces unintended results [b, c] in the form of highly disconnect fracture pieces.

Chapter 8

A Practical Implementation: SIMPAT

This chapter provides some details of SIMPAT, a C++ program that implements the algorithm described in the previous chapters. The chapter is divided into two sections: (1) sub-algorithms aimed to improve the overall pattern reproduction of realizations and (2) methods to improve the performance of the most similar pattern search.

8.1 Utility Algorithms

This section discusses several independent sub-algorithms used in SIMPAT. These sub-algorithms aim to improve the overall pattern reproduction of realizations.

8.1.1 Finding a Reasonable Template Size

Possibly the most influential parameter of SIMPAT is the size $n_{\mathbf{T}}$ of the base template $\mathbf{T}$. Chapters 3 and 4 demonstrate the impact of using different template sizes on the pattern reproduction. The template geometry (as defined by vectors $\mathbf{h}_\alpha$) also effect the result but preliminary experience indicates that the size $n_{\mathbf{T}}$ of a template is much more consequential than its geometry.

Thus, a method to automatically calculate a reasonable template size is desired.

Furthermore, such an automatic method allows considering different template sizes for different multiple-grids, i.e. instead of simply coarsening the fine template $\mathbf{T}$ to a coarse template $\mathbf{T}^g$ while retaining $n_{\mathbf{T}^g} = n_{\mathbf{T}}$, it becomes possible to use a different $n_{\mathbf{T}^g}$ for each multiple grid $\mathbf{G}^g_{re}$. Similarly, the method can be used for different regions $\mathbf{R}_r$ and for each multiple-grid $\mathbf{R}^g_r$ within a region.

This section describes a method to automatically calculate $n_{\mathbf{T}^g}$ for a given training image **ti** and a multiple-grid level g. The basic idea is to start with a small template $\mathbf{T}^g$ and successively expand this template until the pattern database $\mathbf{patdb}_{\mathbf{T}^g}$ obtained using the template contains only unique patterns $\mathbf{pat}^k_{\mathbf{T}^g}$, i.e. no pattern $\mathbf{pat}^k_{\mathbf{T}^g}$ in the pattern database is equal to any another pattern $\mathbf{pat}^{k'}_{\mathbf{T}^g}$ and each pattern has frequency $1/n_{\mathbf{pat}_{\mathbf{T}}}$. This idea is elaborated below:

Algorithm 8.1: Finding a resonable template size

---

1. Start with a coarse template $\mathbf{T}_{t=1}$ of size $n_{\mathbf{T}^g_{t=1}} = 3 \times 3$ on the desired multiple-grid level $\mathbf{G}^g_{ti}$ where t is an iteration index, i.e. $t = 1$ denotes the first template size tested and $n_{\mathbf{T}^g_{t'}} > n_{\mathbf{T}^g_t}$ if $t' > t$.

2. Using the coarse template $\mathbf{T}^g_t$, scan the training image **ti** to obtain the pattern database $\mathbf{patdb}_{\mathbf{T}^g_t}$ (Algorithm 3.1).

3. Search for replicates in the pattern database, i.e. for every pattern $\mathbf{pat}^k_{\mathbf{T}^g_t} \in \mathbf{patdb}_{\mathbf{T}^g_t}$, search for another pattern $\mathbf{pat}^{k'}_{\mathbf{T}^g_t} \in \mathbf{patdb}_{\mathbf{T}^g_t}$ such that $k \neq k'$ but $\mathbf{pat}^k_{\mathbf{T}^g_t} = \mathbf{pat}^{k'}_{\mathbf{T}^g_t}$.

4. If at least one such pattern pair exists, stop searching for more pairs, increase the size $n_{\mathbf{T}^g_t}$ of the template $\mathbf{T}^g_t$, set $t = t + 1$ and repeat the search. $n_{\mathbf{T}^g_{t+1}}$ is obtained by simply adding 2×2 to the previous size. For example, a template of size 3×3 is expanded into a template of size 5×5.

5. If the pattern database $\mathbf{patdb}_{\mathbf{T}^g_t}$ has no replicates, then the final template size $n_{\mathbf{T}^g_*}$ is taken as $n_{\mathbf{T}^g_{t-1}}$, i.e. the template size that was tested for the previous iteration $t - 1$.

In other words, the algorithm keeps increasing the size $n_{\mathbf{T}^g}$ of the template $\mathbf{T}^g$ on the current multiple-grid $\mathbf{G}^g_{ti}$ until all patterns $\mathbf{pat}^k_{\mathbf{T}^g}$ in the pattern database $\mathbf{patdb}_{\mathbf{T}^g}$ are unique. Once this particular size $n_{\mathbf{T}^g}$ is found, the algorithm takes the previously tested size as the final template size for the current multiple-grid $\mathbf{G}^g_{ti}$ and the training image **ti**.

The method finds a 'reasonable' template size in the sense that the calculated size $n_{\mathbf{T}}$ is not necessarily the minimum required size to reproduce desired training image patterns but, in general, it is possibly a good choice, especially for complex training images. The main premise is that, if for $n_{\mathbf{T}}$, the pattern database $\mathbf{patdb}_{\mathbf{T}}$ contains only unique patterns $\mathbf{pat}^k_{\mathbf{T}}$, then it may be argued that $n_{\mathbf{T}}$ captures the 'essence' of the training image and using a larger template will only result in blind copying of large chunks of the image and thus artificially decrease uncertainty.

Figure 8.1 illustrates the application of Algorithm 8.1 to the 11×11 training image of Figure 3.1 for a single-grid. Figure 8.2 shows the results of the algorithm when applied to the 250×250 training image of Figure 4.7, using 4 multiple-grids. In the figure, the algorithm finds smaller templates for coarser grids (Figure 4.7b, c) and larger templates for the finer grids (Figure 4.7d, e).

8.1.2 Calculating a Semi-random (Structured) Path when Conditioning to Hard Data

When conditioning to hard data, it may be desirable to use a structured simulation path instead of a completely random path to honor better the information provided by the hard data, i.e. use a path that visits each node **u** of the realization **re** such that the hard data vicinity is visited first.

`SNESIM` achieves this by constructing a simulation path that spirals away from hard data points (Strebelle, 2002), i.e. the path visits first the hard data points; then, nodes that are increasingly further away. Additionally, using the method described in Liu (2004), `SNESIM` can also construct a structured path visiting first more informative soft data based on an information content measure defined by the user.

Another approach is to construct a simulation path based on the density of the

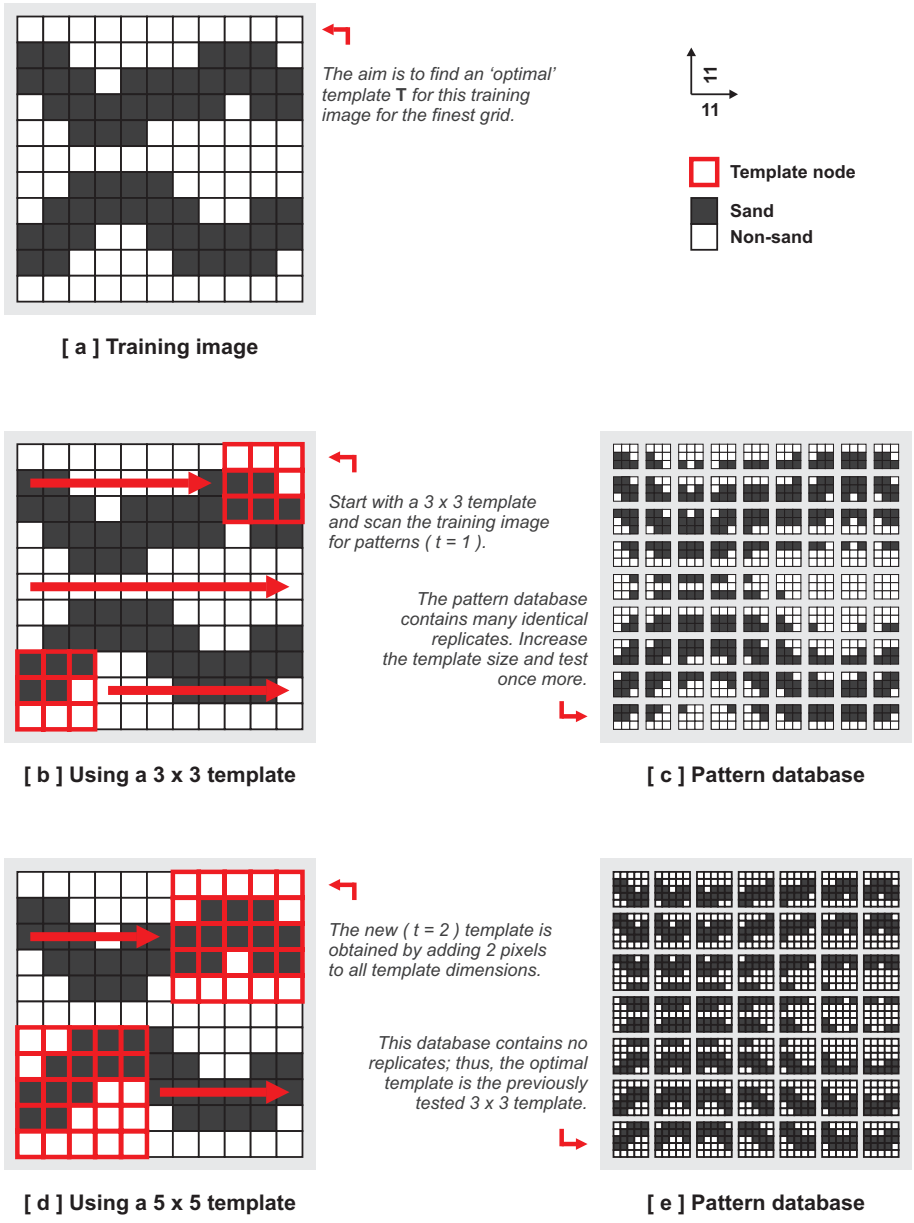


Figure 8.1: Application of Algorithm 8.1 to the training image previously shown in Figure 3.1. The final optimal template size is found as 3×3.

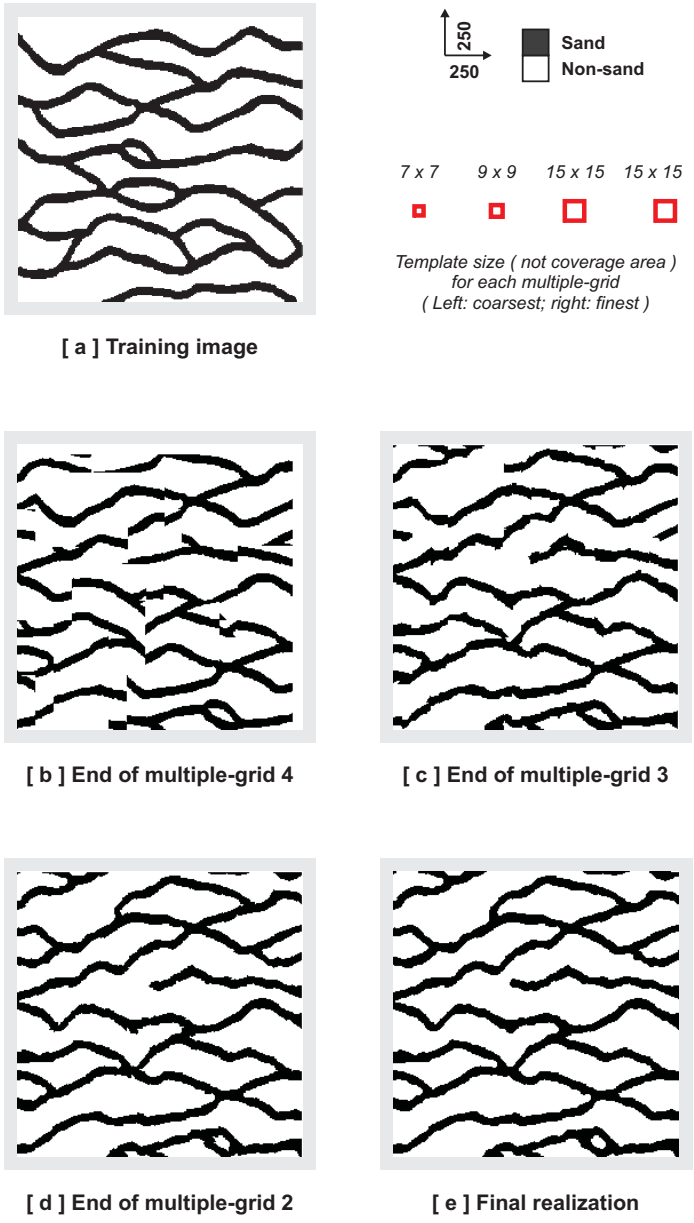


Figure 8.2: Application of Algorithm 8.1 to the training image shown in Figure 4.7. The template sizes found by the algorithm are shown on the top right corner.

hard data in different areas of the hard data image **hd**. This idea is elaborated in Algorithm 8.2 which results in a fully random path when there is no hard data since $n_{hd}(\mathbf{u}) = 0$ in such a case and becomes increasingly more structured with increasing number of hard data and/or larger $n_{\mathbf{T}}$.

Algorithm 8.2: Calculating a structured path based on hard data

---

1. Given a hard data image **hd** discretized by grid $\mathbf{G}_{re}$, scan the $\mathbf{G}_{re}$ using raster scan order.
2. At a node $\mathbf{u}$, place the template $\widetilde{\mathbf{T}}$ (the dual template that will be used by the simulation algorithm) at $\mathbf{u}$ and count the number of nodes $\mathbf{u} + \mathbf{h}_\alpha$ such that $hd(\mathbf{u} + \mathbf{h}_\alpha) \neq \chi$, i.e. count the number of nodes within the template that contains hard data. Store this count (denoted by n_{hd}) for node $\mathbf{u}$.
3. Repeat the above step for all nodes of the hard data image **hd** to obtain the hard data counts $n_{hd}(\mathbf{u})$ for every node $\mathbf{u}$.
4. Generate a semi-random path such that a node $\mathbf{u}_\alpha$ is visited before another node $\mathbf{u}_\beta$ if $n_{hd}(\mathbf{u}_\alpha) > n_{hd}(\mathbf{u}_\beta)$, nodes with equal n_{hd} are visited randomly.

It is important to note that, Algorithm 8.2 does not necessarily start visiting the nodes $\mathbf{u} \in \mathbf{G}_{re}$ for $hd(\mathbf{u}) \neq \chi$. In other words, the structured path might not contain the hard data nodes themselves as its starting nodes. Instead, the first node $\mathbf{u}_1$ to be visited is the node with the highest number n_{hd} of hard data nodes within its neighborhood (but $hd(\mathbf{u}_1)$ is not necessarily informed). Figure 8.3 illustrates this.

Figure 8.4 shows the application of Algorithm 8.2 using the hard data of Figure 6.5e. In the figure, two different sets of simulations are performed; one using a fully random path and another using the structured path. When a fully random path is used, the realizations still replicate the geobody but the overall uncertainty is higher (Compare the E-types of Figure 8.4d to Figure 8.4f).

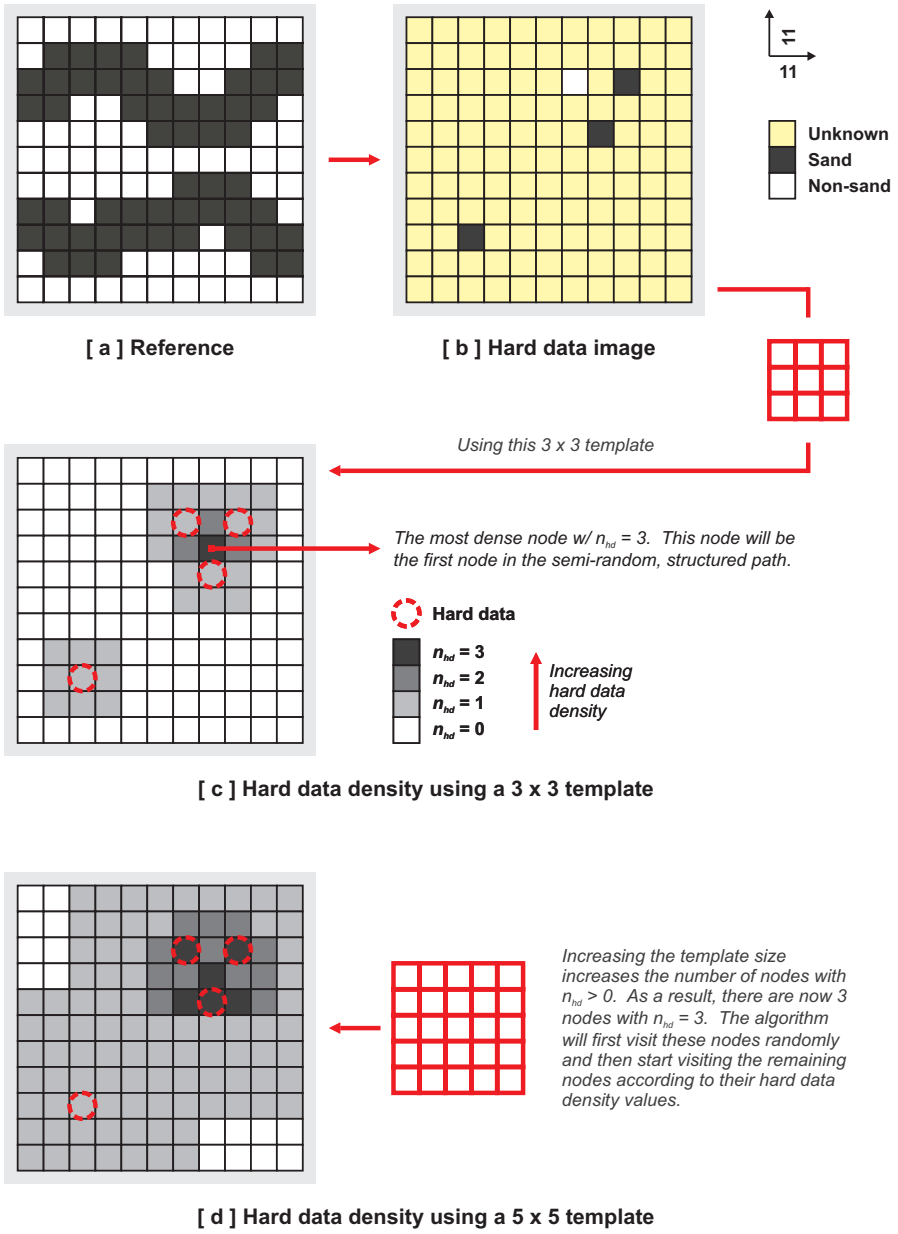


Figure 8.3: The proposed structured path does not necessarily start from hard data nodes. Instead, the starting point is the most data-dense area of the hard data image.

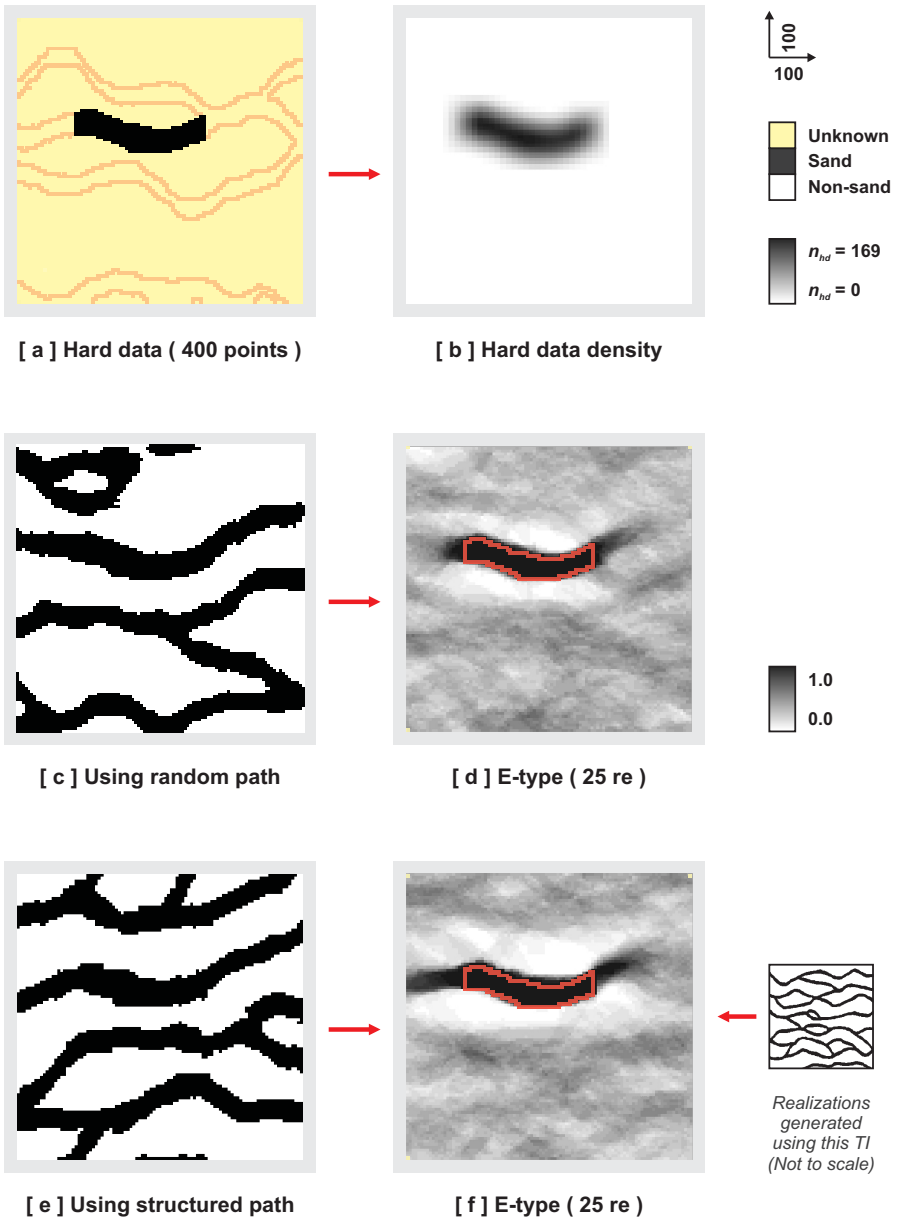


Figure 8.4: Application of Algorithm 8.2 to the hard data of Figure 6.5e of Chapter 6. Using a structured (semi-random) path, the uncertainty of the realizations are decreased when compared to the fully random path [$n_{\mathbf{T}} = 7 \times 7$; $n_g = 2$].

8.1.3 Pre-processing Soft Training Images (TRANSPAT)

The method for soft data conditioning described in Chapter 5 requires a so-called soft training image **sti**. This image is obtained by applying a forward model $\mathbf{F}^*$ (approximating the actual Earth filter $\mathbf{F}$) to the training image **ti**, i.e. $\mathbf{sti} = \mathbf{F}^*(\mathbf{ti})$. In real cases, $\mathbf{F}$ and $\mathbf{F}^*$ are always different. For example, it is extremely difficult to model the full complexity of an actual geophysical survey, including all the measurement and processing errors. As a result, the statistics of the actual soft data **sd** may be considerably different from the statistics of the soft training image **sti**.

One possible correction to the above problem is to identify the histogram of the soft training image **sti** with the histogram of the soft data **sd** (For example, using the GSLIB program **trans**; Deutsch and Journel (1998)). However, this would only identify the first order statistics of the two images and leave unchanged the differences in higher order statistics, resulting possibly in little improvement in terms of pattern reproduction when conditioning to soft data.

This section describes a method to perform higher order statistics identification, akin to the histogram identification of GSLIB's **trans** but using patterns instead. For this reason the method is called "pattern identification" and the algorithm "TRANSPAT". What does TRANSPAT achieve? The method replaces each pattern of the original soft training image with the corresponding most similar pattern of the original soft data. Hence, it aims to make the soft data and soft training image patterns consistent. Algorithm 8.3 describes TRANSPAT.

In Algorithm 8.3, pattern identification is achieved by running the SIMPAT algorithm but using a data input different from a normal SIMPAT run. To stress this difference, the name TRANSPAT is used. A normal, conditional SIMPAT run requires as input a training image $\mathbf{ti}^i$, a soft training image $\mathbf{sti}^i$ and a soft data $\mathbf{sd}^i$ where i identifies these images as original input data. Then, for a SIMPAT run, one has:

$$\begin{aligned} \mathbf{ti} &= \mathbf{ti}^i \\ \mathbf{sti} &= \mathbf{sti}^i \\ \mathbf{sd} &= \mathbf{sd}^i \end{aligned} \tag{8.1}$$

In other words, the simulation is performed using for training image $\mathbf{ti} = \mathbf{ti}^i$, the soft

Algorithm 8.3: Pattern identification (TRANSPAT)

1. Call the original soft data image $\mathbf{sd}^i$ and copy it to the pair of images $\mathbf{ti}^t$ and $\mathbf{sti}^t$, i.e. $\mathbf{ti}^t = \mathbf{sd}^i$ and $\mathbf{sti}^t = \mathbf{sd}^i$.
2. Call the original soft training image $\mathbf{sti}^i$ and copy it to $\mathbf{sd}^t$, i.e. $\mathbf{sd}^t = \mathbf{sti}^i$.
3. Run a single-grid, conditional simulation using $\mathbf{ti}^t$ as the training image, $\mathbf{sti}^t$ as the soft training image and $\mathbf{sd}^t$ as the soft data. The run is typically performed using a template $\mathbf{T}$ with size $n_{\mathbf{T}}$ that reflects the scale of the $\mathbf{F}^*$ used to obtained the original soft training image $\mathbf{sti}^i$.
4. Replace the original soft training image $\mathbf{sti}^i$ with the result of this single-grid simulation $\mathbf{re}^t$, i.e. instead of pairing the original $\mathbf{ti}^i$ and $\mathbf{sti}^i$, the actual SIMPAT is to be performed using $\mathbf{ti}^i$ and $\mathbf{re}^t$ (as the soft training image corresponding to $\mathbf{ti}^i$).

training image $\mathbf{sti} = \mathbf{sti}^i$ and the soft data $\mathbf{sd} = \mathbf{sd}^i$. And, for a TRANSPAT run, one has:

$$\begin{aligned} \mathbf{ti} &= \mathbf{sd}^i \\ \mathbf{sti} &= \mathbf{sd}^i \\ \mathbf{sd} &= \mathbf{sti}^i \end{aligned} \tag{8.2}$$

In other words, the simulation is performed using the training image $\mathbf{ti} = \mathbf{sd}^i$, the soft training image $\mathbf{sti} = \mathbf{sd}^i$ and the soft data $\mathbf{sd} = \mathbf{sti}^i$. Thus, in essence, a TRANSPAT run is a 'reverse' SIMPAT run and is performed by taking the soft data as the training image and vice-versa.

Figure 8.5 shows the application of the TRANSPAT algorithm when used for the soft training image of Figure 6.23. For this case, the soft training image was obtained using a 25×25 moving average ($= \mathbf{F}^*$) and the soft data was obtained using a 15×15 moving average ($= \mathbf{F}$). See Chapter 6 for details. Since the soft training image contains 'smoother' patterns than the soft data, running TRANSPAT on the soft training image essentially removes this smoothness and identifies the channels in the

soft training image better, i.e. crispier as dictated by the 15×15 soft data. Figure 8.6 shows a conditional realization obtained using this pre-processed soft training image (Figure 8.5).

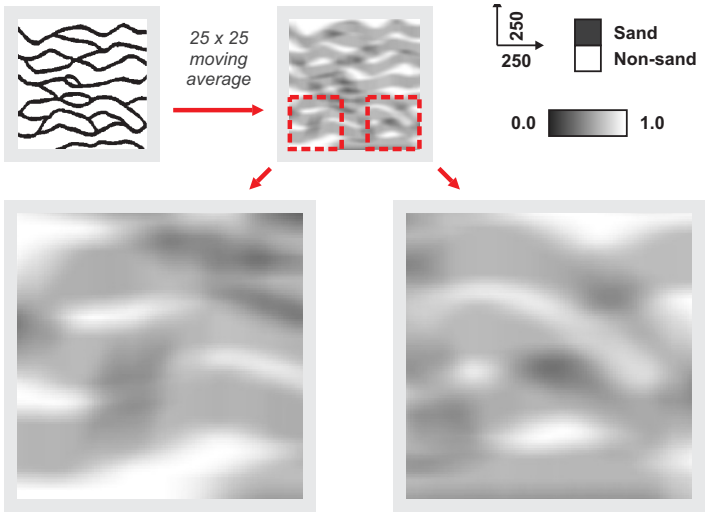
8.2 Most Similar Pattern Search

The most CPU demanding sub-algorithm of SIMPAT is related to the search for the most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$, i.e. the maximization of $s\left\langle \mathbf{dev}_{\mathbf{T}}(\mathbf{u}), \mathbf{pat}^k_{\mathbf{T}} \right\rangle$ for $k = 1, \ldots, n_{\mathbf{pat}_{\mathbf{T}}}$. A variation of this maximization is finding all patterns $\mathbf{pat}^k_{\mathbf{T}}$ that fulfill a certain condition, i.e. the first lookup of hard data preconditioning.

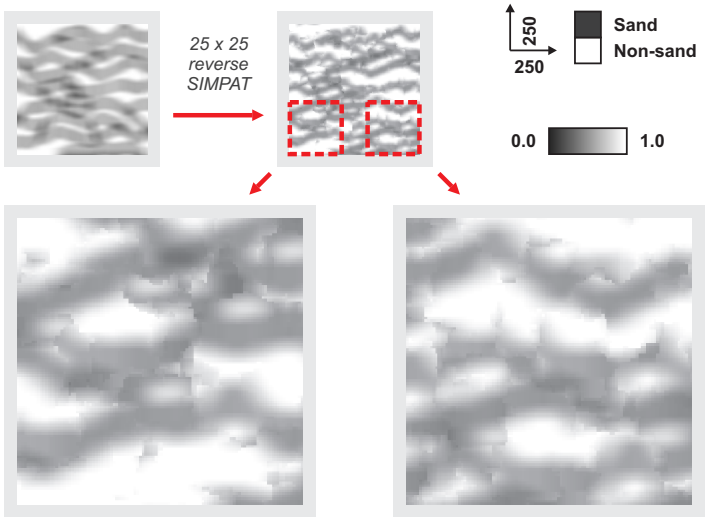
This search is a well understood problem in computational geometry known as the "nearest-neighbor search" (Sedgewick, 1988) or sometimes the "post office problem" (Knuth, 1997). The problem is given the name "nearest neighbor search" since it can be stated as finding a hyper-point that is 'nearest' to another hyper-point in a D dimensional space. In the case of SIMPAT, the hyper-points are data events $\mathbf{dev}_{\mathbf{T}}(\mathbf{u})$ and patterns $\mathbf{pat}^k_{\mathbf{T}}$; the distance is defined through the similarity measure $s\langle \cdot, \cdot \rangle$ and $D = n_{\mathbf{T}}$.

This problem can be trivially solved in $O(n_{\mathbf{pat}_{\mathbf{T}}})$ time by calculating all possible $s\left\langle \mathbf{dev}_{\mathbf{T}}(\mathbf{u}), \mathbf{pat}^k_{\mathbf{T}} \right\rangle$ and taking the $\mathbf{pat}^k_{\mathbf{T}}$ with maximum similarity as the most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$. Yet, this trivial and brute-force solution (sometimes called "linear search") can be extremely CPU demanding, especially when both $n_{\mathbf{pat}_{\mathbf{T}}}$ and $n_{\mathbf{T}}$ are large; typically when associated to a large and complex 3D training image calling for the use of a large template. Figure 8.7 shows the complexity behavior of the algorithm with increasing $n_{\mathbf{pat}_{\mathbf{T}}}$ and $n_{\mathbf{T}}$.

The computational geometry literature gives many algorithms that solve the exact nearest-neighbor problem in $O(\log n_{\mathbf{pat}_{\mathbf{T}}})$ time for dimensions smaller than 50, i.e. $n_{\mathbf{T}} < 50$. Approximate solutions that perform better than this logarithmic time are also known. The generally accepted, ad-hoc limit of 50 dimensions is due to the typical high memory demand of the most popular nearest neighbor search algorithms. These solutions generally utilize some form of a tree structure (similar to the search tree found in SNESIM) and tree structures are known to be highly memory demanding for



[a] Original soft training image (25 x 25 average)



[b] Pre-processed soft training image

Figure 8.5: Application of Algorithm 8.3 (TRANSPAT) to the soft training image of Figure 6.23.

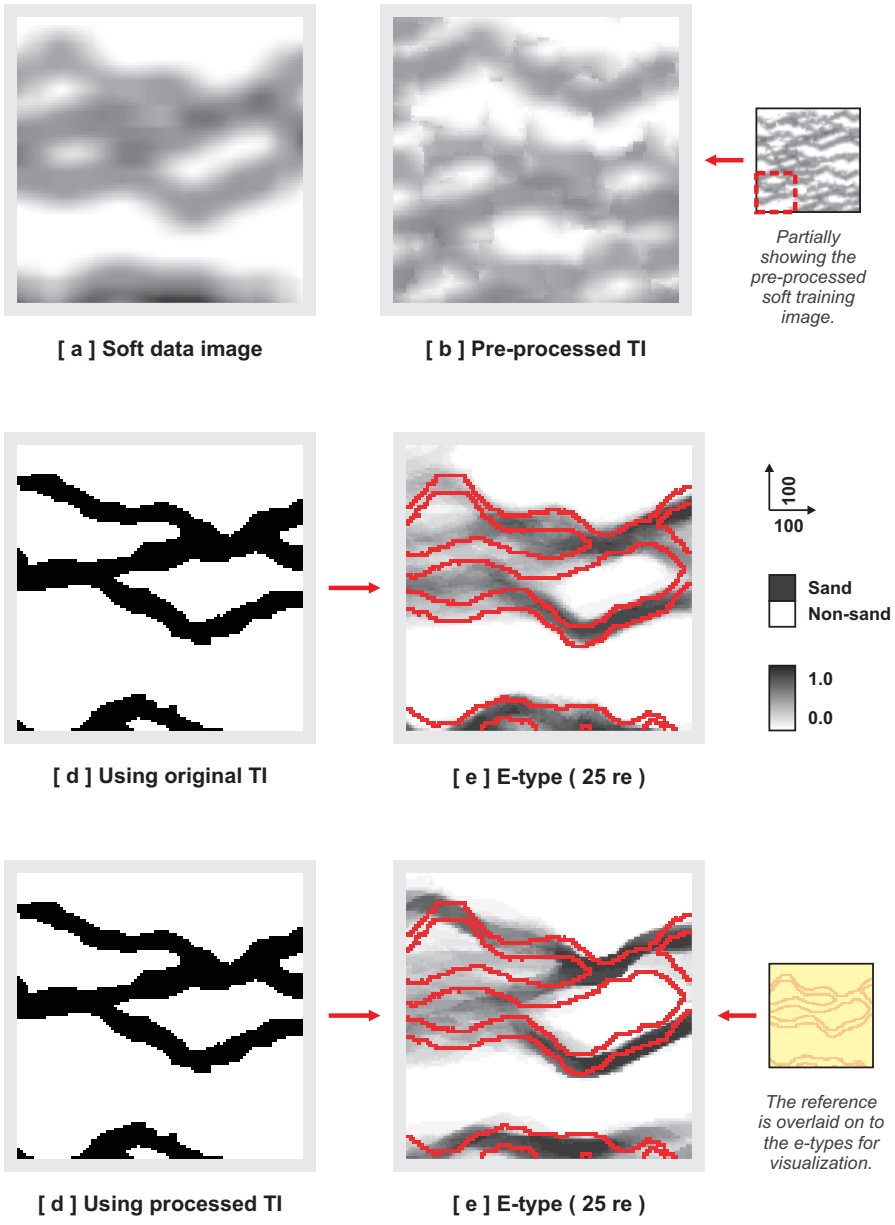


Figure 8.6: Conditional simulation using the soft training image of Figure 6.23.

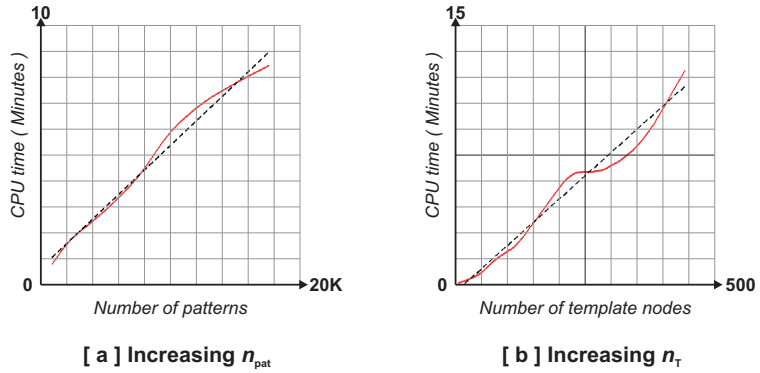


Figure 8.7: The complexity behavior of SIMPAT with increasing $n_{\mathbf{pat_T}}$ (and a fixed template size) and $n_{\mathbf{T}}$ (and a fixed pattern database size). The CPU used is an Intel Pentium-M 1.7 GHz. mobile processor (2004 model).

higher dimensions. For a comprehensive review of exact and approximate solutions, see Smid (1999).

Since, in SIMPAT, $n_{\mathbf{T}}$ is typically > 250 for realistic 3D training images, it is not straightforward to adapt a computational geometry algorithm to the most similar pattern search without imposing extreme memory demands on the algorithm. Furthermore, SIMPAT needs to operate on data events with missing (unknown) nodes ($= \chi$), an issue rarely addressed by the computational geometry literature. Due to this reason, SIMPAT does not employ any single computational geometry method, instead it relies on a combination of several different techniques to improve the overall performance. It should be noted that, although these 'tricks' do improve the performance, they do not change the linear complexity behavior of the algorithm as shown in Figure 8.7.

8.2.1 Skipping Grid Nodes

During simulation, whenever a $\mathbf{pat}^*_{\mathbf{T}}$ defined over a template $\mathbf{T}$ is pasted on to the realization, the algorithm actually simulates the values of several nodes of $\mathbf{re}$, not only the visited central value. One might consider not visiting all or some of these nodes

later during the simulation, i.e. they might be skipped while visiting the remaining nodes of the random path. This results in visiting less number of nodes within **re** and thus improves the CPU efficiency of the algorithm since visiting less number of nodes means performing less number of similarity calculations. In SIMPAT, skipping already calculated nodes is achieved by visiting all grid nodes separated by a minimum distance (called the "skip size").

Consider the 2D 9×9 Cartesian grid given in Figure 8.8. Assume a 5×5 template is used and is currently located on $\mathbf{u} = \mathbf{u}_{20}$. If the template $\mathbf{T}$ is defined by $\mathbf{h}_\alpha$ vectors with $\alpha = 1, \ldots, n_\mathbf{T}$ and $n_\mathbf{T} = 25$, then, the data event $\mathbf{dev_T}(\mathbf{u})$ is defined by the following vectors $\mathbf{u} + \mathbf{h}_\alpha$:

$$\mathbf{u} + \mathbf{h}_\alpha = \{\mathbf{u}_0, \ldots, \mathbf{u}_4, \mathbf{u}_9, \ldots, \mathbf{u}_{13}, \mathbf{u}_{18}, \ldots, \mathbf{u}_{22}, \mathbf{u}_{27}, \ldots, \mathbf{u}_{31}, \mathbf{u}_{36}, \ldots, \mathbf{u}_{40}\} \qquad (8.3)$$

Given the above data event, assume the skip size is set to 3. During the simulation, when node $\mathbf{u} = \mathbf{u}_{20}$ is visited, the values of the most similar pattern $\mathbf{pat}^*_\mathbf{T}$ are used to populate all the values of the data event $\mathbf{dev_T}(\mathbf{u})$. With a skip size of 3, all the nodes within the 3×3 neighborhood of the node $\mathbf{u}$ defined by the $\mathbf{h}_\beta$ vectors, $\beta = 0, \ldots, 8$ such that $\mathbf{u} + \mathbf{h}_\beta = \{\mathbf{u}_{10}, \mathbf{u}_{11}, \mathbf{u}_{12}, \mathbf{u}_{19}, \mathbf{u}_{20}, \mathbf{u}_{21}, \mathbf{u}_{28}, \mathbf{u}_{29}, \mathbf{u}_{30}\}$ are marked 'visited' and removed from the random path. This removal from the random path does not mean the values of these nodes remain frozen. It simply means the algorithm will not perform an explicit most similar pattern search on these nodes. The node values are still allowed to change depending on the similarity calculations that will be performed within their vicinity.

In the above example, a single most similar pattern search is performed and a total of $5 \times 5 = 25$ values are calculated for node $\mathbf{u}_{20}$, i.e the data event $\mathbf{dev_T}(\mathbf{u}_{20})$. Then, the central 3×3 area (9 nodes) of this data event are removed from the random path, i.e. they will not be visited by the algorithm. The 'nearest' nodes to $\mathbf{u}_{20}$ that the algorithm will visit are $\mathbf{u}_{23}$ and $\mathbf{u}_{38}$. When visiting one of these nodes (or other further away nodes that are still withing the vicinity of $\mathbf{u}_{20}$), the algorithm uses the nodes within the 3×3 central area of $\mathbf{u}_{20}$ for similarity calculations and may modify them depending on the outcome, i.e. these nodes still effect the simulation results although they are never explicitly visited. As a result, the number of nodes that needs

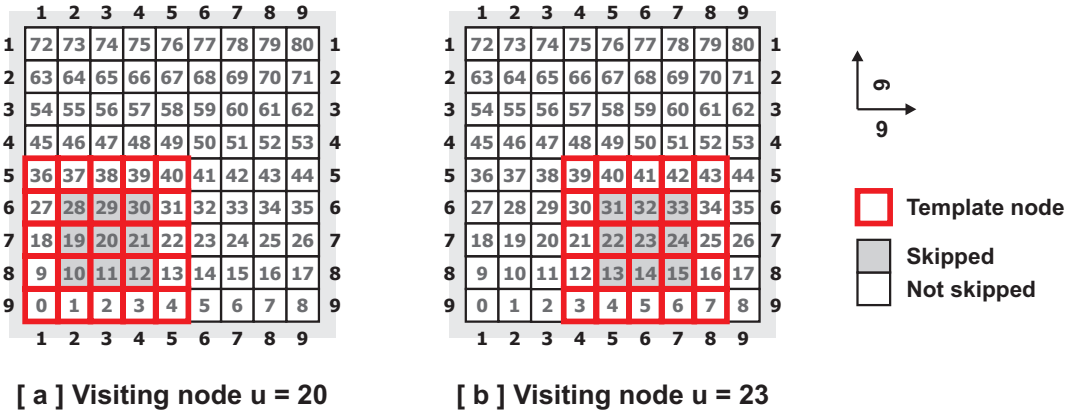


Figure 8.8: (a) Visiting node $\mathbf{u} = \mathbf{u}_{20}$. The 5×5 template is placed on the node. Due to the skip size of 3, the nodes within the 3×3 shaded area marked visited and removed from the random path. (b) Visiting node $\mathbf{u} = \mathbf{u}_{23}$ later during the simulation.

to be visited is decreased by a factor of 9, which results in 9 times less number of most similar pattern searches. Thus, depending on the skip size, considerable speed gains might be achieved.

8.2.2 Fast Calculation of Manhattan Distances

Consider the Manhattan distance of Equation 3.6, repeated here:

$$d\left\langle \mathbf{dev_T}(\mathbf{u}), \mathbf{pat}_{\mathbf{T}}^{k} \right\rangle = \sum_{\alpha=0}^{n_{\mathbf{T}}} \left| dev_{\mathbf{T}}(\mathbf{u}+\mathbf{h}_\alpha) - pat_{\mathbf{T}}^{k}(\mathbf{h}_\alpha) \right| \tag{8.4}$$

Using this definition, define a partial distance $d^{\beta}\left\langle \mathbf{dev_T}(\mathbf{u}), \mathbf{pat}_{\mathbf{T}}^{k} \right\rangle$ as:

$$d^{\beta}\left\langle \mathbf{dev_T}(\mathbf{u}), \mathbf{pat}_{\mathbf{T}}^{k} \right\rangle = \sum_{\alpha=0}^{\beta} \left| dev_{\mathbf{T}}(\mathbf{u}+\mathbf{h}_\alpha) - pat_{\mathbf{T}}^{k}(\mathbf{h}_\alpha) \right| \tag{8.5}$$

where $\beta \in [1, n_{\mathbf{T}}]$. In other words, $d^{\beta}\langle \cdot, \cdot \rangle$ is the partial distance between the data event $\mathbf{dev_T}(\mathbf{u})$ and the pattern $\mathbf{pat}_{\mathbf{T}}^{k}$, calculated using only the nodes $\mathbf{u}+\mathbf{h}_\alpha$ with $\alpha = 1, \ldots, \beta$.

When searching for the most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$, for $k > 1$, it is unnecessary to evaluate the full summation over all $n_{\mathbf{T}}$ nodes of the template $\mathbf{T}$ to determine whether the current pattern $\mathbf{pat}^k_{\mathbf{T}}$ is a possible most similar pattern candidate or not. Instead, the summation can be terminated whenever the partial sum $d^\beta \langle \cdot, \cdot \rangle$ is greater than the distance between the current most similar pattern candidate and the data event. In other words, as soon as the distance of the current pattern $\mathbf{pat}^k_{\mathbf{T}}$ is 'worse' then the current known 'best' distance, the summation is stopped.

8.2.3 Parallel SIMPAT (Using Multiple CPUs)

For complex training images and large simulation grids, the performance level of linear search is unacceptable; especially when a large template is used. One solution is to parallelize the algorithm and exploit multiple CPUs to improve the search performance.

From an algorithmic point of view, parallelization of SIMPAT is trivial: One can divide the pattern database into n_{cpu} parts where n_{cpu} is the number of CPUs available. Then, instead of minimizing $d\left\langle \mathbf{dev}_{\mathbf{T}}(\mathbf{u}), \mathbf{pat}^k_{\mathbf{T}} \right\rangle$ for $k = 1, \ldots, n_{\mathbf{pat}_{\mathbf{T}}}$, each CPU minimizes $d\left\langle \mathbf{dev}_{\mathbf{T}}(\mathbf{u}), \mathbf{pat}^k_{\mathbf{T}} \right\rangle$ for $k = \alpha, \ldots, \beta$ where α and β denote the beginning and ending indices of ranges of size $n_{\mathbf{pat}_{\mathbf{T}}}/n_{cpu}$. In essence, each CPU searches for a similar pattern $\mathbf{pat}^{*,c}_{\mathbf{T}}$ where c denotes the CPU index and $c = 1, \ldots, n_{cpu}$. Once the results for these minimizations are obtained, a final minimization of $d\left\langle \mathbf{dev}_{\mathbf{T}}(\mathbf{u}), \mathbf{pat}^{*,c}_{\mathbf{T}} \right\rangle$ is performed to find the final most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$.

Chapter 9

Conclusions

Assume that a box set of jigsaw puzzle pieces and partial (and possibly contradicting) information about the final image to be reconstituted from these pieces are given. The puzzle pieces are allowed to be used more than once or none at all and they can overlap with each other. The pieces should not only relate/interlock with their neighbors but also with pieces further away. The puzzle is 3D. How does one approach solving that puzzle?

This thesis sees reservoir modeling to be analogous to the above puzzle solving problem and proposes an image construction approach as a solution. In this approach, the 'pieces' of the final image to be constructed (the reservoir) are obtained from a training image that defines a conceptual geological scenario for the reservoir by depicting relevant geological patterns expected to be found in the subsurface under the assumed scenario. The aim is, then, to reproduce these training patterns within the final image while anchoring them to the available subsurface data.

To achieve this aim, the proposed approach utilizes a non-iterative sequential simulation. Sequential simulation, in its simplest form, can be seen as a divide-and-conquer technique where a large problem is divided into several small problems and the final solution is obtained through the sequential solution of these smaller problems (called the "local problem" from hereon). Traditional reservoir modeling tools such as variogram-based and multiple-point geostatistics approach this local problem through the estimation of a local, conditional probability distribution (ccdf) for node $\mathbf{u}$ of a

realization **re** given the available neighborhood information (data event) $\mathbf{dev_T(u)}$. Once the conditional probability distribution is estimated, a value is drawn from it and one proceeds to the next local problem. The proposed method, however, utilizes a different approach:

- Instead of aiming to inform only the central node **u**, the method aims to inform all the nodes within a given template at **u**..
- More importantly, instead of estimating a conditional cdf, the method aims to find a training image pattern $\mathbf{pat^*_T}$ 'most similar' to the data event $\mathbf{dev_T(u)}$ located at **u**.

This new approach has the advantage that it focuses directly on the core purpose of reservoir modeling: reproduction of desired geological patterns in the final realization (be it two-point or multiple-point, stationary or non-stationary). The similarity-based image construction approach avoids many of the restrictions of probabilistic approaches such as assumption of stationarity for the inference and modeling of patterns via probabilities.

This study demonstrates the potential of a similarity-based image construction approach to reservoir modeling. Section 10.1 summarizes the main contributions of the study. However, it must be stressed that this purely algorithmic formulation of the approach to stochastic pattern reproduction and data conditioning is not yet fully understood. Future research should therefore focus on understanding better the advantages and limitations of the various new concepts introduced in this study, possibly borrowing more from the fields of computer vision and image processing and linking them to the traditional probabilistic vision. Section 10.2 discusses possible avenues for such future research.

9.1 Summary of the Study

The main contributions of the thesis can be summarized as follows:

- A general framework for reservoir modeling that does not utilize probabilities and instead relies on the concept of similarity. This framework borrows concepts

from the fields of computer vision and image processing and adapts them to the task of reservoir modeling.

- A new multiple-grid approach to better handle the scale interactions of complex geological scenarios. To overcome the CPU demand of using large neighborhoods (templates), sequential simulation algorithms often use a multiple-grid approach to approximate the large template with many smaller templates. In complex geological scenarios (training images) where all geological scales interact with each other, the approximation introduced by using a multiple-grid approach affects the final results. The proposed multiple-grid approach attempts to improve this approximation by propagating more information from coarser grids to finer grids using dual templates.

- A new way to handle dense hard data when using a multiple-grid approach. The traditional multiple-grid approach forces hard data to be relocated (moved) to coarse grid nodes. Such relocation is problematic, since several hard data can be relocated to the same coarse grid node. The proposed conditioning method, using dual templates, solves this problem and attempts to honor the hard data directly at their original location.

- An alternative approach to soft data conditioning. This method is capable of conditioning to any type of soft data as long as the process governing the relation between the property to be simulated (hard variable) and the soft data can be implicitly expressed through pairs of training images. Using this approach one can better account for the pattern information provided by seismic data. Moreover, if the soft training image has been generated using a physical model, this method allows to account for the physical relationship between seismic and the simulated property.

- A software implementation (`SIMPAT`) of the proposed algorithm implemented using a C++ library following the generic programming approach (Austern, 1998) that allows future modifications to be performed with ease.

9.2 Future Work

This section summarizes possible avenues for future research related to the similarity-based reservoir modeling framework proposed in this thesis.

Better understanding of training image transformations

Chapter 7 describes two fundamental image transformations, namely rotation and scaling. The image processing literature lists several other image transformations that may be useful in a variety of cases for reservoir modeling. Since training images currently constitute the most important input to the algorithm, allowing the users of the algorithm to easily morph existing training images to fit any specific problem at hand is of utmost importance to ensure that the method remains practical and versatile. For example, transformations such as erosion and dilation (Gonzalez and Woods, 2002) can be used to modify the patterns and multiple-point statistics of a training image to match a set of target statistics and expected patterns.

Improved methods for quantifying pattern reproduction

Chapter 6 describes the generation of a so-called "simulation fitness map" as a way to quantify the overall pattern reproduction of a realization and a method to check hard (and soft) data conditioning. When attempting to find a most similar pattern $\mathbf{pat}^*_{\mathbf{T}}$ at each node $\mathbf{u}$, a similarity error is calculated as a side product of the similarity maximization. This concept can be further developed to perform several other checks that may be useful for reservoir modeling. For example,

1. Calculating the similarity of realizations one to each other, and a way to rank them;

2. Calculating the similarity of two training images, which can be used to asses the geological uncertainty space, i.e. when there are several different candidate training images, what are the training images that are most dissimilar to each other and more likely to span uncertainty;

3. Calculating the similarity of hard/soft data images to hard/soft training images to decide ahead of time whether the actual data conflicts with the training image.

Better similarity measures, different representations of patterns and different ways to search for the most similar pattern

Chapter 5 describes the proximity transforms as a way to provide similarity measures better than the traditional (single-point) Manhattan distance (Better in the sense that the result of a similarity comparison is more tuned to the intuition of a human expert). The computer vision literature lists several other similarity measures that can be used within the proposed framework. Such similarity measures often require representing the patterns in a different way, for example, by extracting the common features of patterns and performing the similarity calculation on such features. In addition to possibility improving the similarity calculation itself, such an approach often has the advantage of decreasing the CPU demand of the algorithm. The ongoing work of Zhang et al. (2004) using filters applied to the training images, follows this avenue and yields promising results.

A related approach is to group the available patterns into clusters and then to perform a staged most similar pattern search, first locating the cluster that the most similar pattern falls into and then finding the most similar pattern within this cluster. Indeed, a similar approach was initially used by `SIMPAT` (Arpat and Caers, 2003) but was later abandoned in favor of using the more direct, albeit slower, search.

Finally, methods that exploit the spatial nature of the patterns (i.e. the fact that individual patterns are actually extracted from a training image) may also prove to be beneficial.

Better handling of complex multiple-scale nature of training images

Handling complex training images that exhibit complex relations between heterogeneity at different scales requires use of advanced methods that can handle such

relations. Chapter 4 describes a new multiple-grid that aims to improve the traditional approach in this avenue. However, that method remains hierarchical in nature, i.e. the algorithm first simulates the coarsest grid (scale) and successively refines the realization to impose finer scale details. Furthermore, to simulate the coarsest grid, sub-sampled patterns from a fine-scale training image are used, i.e. the coarse patterns $\mathbf{pat}^k_{\mathbf{T}_g}$ approximate large scale patterns by taking increasingly distant nodes of the training image instead of averaging (upscaling) all the finer nodes.

A different approach for modeling the multiple-scale heterogeneity of reservoirs is to use different training images for each scale. In such an approach, a coarse scale training image can be obtained by averaging (upscaling) a fine scale training image or can be provided independently of other scale training images. The method of averaging should be defined according to the problem at hand. Then, the simulation could be performed simultaneously at all scales, not hierarchically from coarse to fine. In order to simulate in parallel, a multiple-scale template would need to be used. This template would consists of various sub-templates, each covering a different scale. Similarity calculations could then be performed using all scales jointly. Evidently, this will increase the CPU demand when searching for the most similar 'multiple-scale' patterns. Hence new search strategies would need to be developed, if the method is to be practical.

The possible advantages of the above approach can be summarized as follows:

1. Different training images or different upscaling methods can be used to describe the different characteristics of different geological scales within a reservoir. For example, in fracture reservoirs, a fine scale training image might describe many details of individual fractures whereas coarser scale training images might describe fracture networks.

2. Since the simulation is no longer hierarchical with respect to scale relations, it becomes possible for fine scale details to affect coarse scale features of a realization during simulation, i.e. a feedback mechanism from fine scales to coarse scales is established.

3. It becomes possible to integrate information from different sources at their correct scales. For example, one could relate seismic data to an petrophysical property or a facies that are upscaled to the seismic scale instead of attempting to relate the fine-scale petrophysical properties to (coarse-scale) seismic data.

Abandoning training images in favor of object piece generators

The most important challenge of multiple-point geostatistics is how to obtain a training image. One possible alternative is to forgo the use of explicit training images altogether and instead rely on generators that can output small object pieces (instead of complete objects as done in object-based modeling tools). These object pieces can then be internally converted to their corresponding pixel representations to be used in similarity calculations. In other words, instead of a pattern database $\mathbf{patdb_T}$, one would have a pattern generator that outputs candidate patterns on demand; the rest of the algorithm remains the same. The advantage of such an approach is twofold:

1. Since a pattern generator can output practically an infinite number of patterns, the realizations would no longer be limited to the patterns of a finite training image; and,

2. Describing object properties (such as channel width, thickness and sinuosity) is often more intuitive to human experts. Once these descriptions are available, it may be possible to derive the internal parameters of the algorithm (such as the template size $n_{\mathbf{T}}$ or the number of multiple-grids n_g), relieving the user of the algorithm from supplying these possibly less intuitive parameters (Less intuitive in the sense that parameters such as the template size does not directly relate to any geological concept).

However, it should be noted that, using object piece generators might also introduce new challenges, mainly variations of those found in fully object-based methods such as convergence problems when conditioning to dense hard data. Furthermore, the additional CPU power required by the approach might prove to be impractical.

Nomenclature

Grids and Regions

n_g	.....	Number of multiple-grids
g	.....	Coarseness level; $g = 1, \ldots, n_g$
r	.....	Region index; $r = 1, \ldots, n_{\mathbf{R}}$
$\mathbf{R}_r$, $\mathbf{R}_r^g$	.....	Region r [at coarseness level g]
$\mathbf{G}, \mathbf{G}^g$	.....	Generic grid [at coarseness level g]
$\mathbf{G}_{re}$	.....	Realization grid
$\mathbf{G}_{ti}$, $\mathbf{G}_{ti,r}$	.....	Training image grid [of Region r]

Templates

$n_{\mathbf{T}}$	.....	Number of nodes in template $\mathbf{T}$
$\mathbf{T}$, $\mathbf{T}^g$	.....	Template [at coarseness level g]
$\widetilde{\mathbf{T}}^g$	.....	Dual of template $\mathbf{T}^g$

Location

$\mathbf{u}$	.....	Absolute location vector
$\mathbf{h}$, $\mathbf{h}^g$	.....	Relative location vector [$\in \mathbf{G}^g$]

Images

$\mathbf{img}$, $\vec{\mathbf{img}}$	.....	Generic image [with multiple-bands]
$\mathbf{re}$, $\mathbf{hd}$, $\mathbf{sd}$	.....	Realization, hard data and soft data
$\mathbf{ti}$, $\mathbf{ti}_r$	.....	Training image [of Region r]
$\mathbf{sti}$, $\mathbf{sti}_r$	.....	Soft training image [of Region r]

Patterns

$n_{\mathbf{pat_T}}$	.....	Number of patterns using template $\mathbf{T}$
k	.....	Pattern index; $k = 1, \ldots, n_{\mathbf{pat_T}}$
$\mathbf{pat}_{\mathbf{T}}^{k}$	.....	Pattern k using template $\mathbf{T}$
$\vec{\mathbf{pat}}_{\mathbf{T}}^{k}$	.....	Multiple-band pattern (of $\vec{\mathbf{ti}}$)
$\mathbf{spat}_{\mathbf{T}}^{k}$	.....	Soft pattern (of **sti**)

Data Events

$\mathbf{dev_T}(\mathbf{u})$	.....	Data event at $\mathbf{u}$ using template $\mathbf{T}$
$\vec{\mathbf{dev}}_{\mathbf{T}}(\mathbf{u})$	.....	Multiple-band data event (of $\vec{\mathbf{re}}$)
$\mathbf{hdev_T}(\mathbf{u})$	.....	Hard data event (of **hd**)
$\mathbf{sdev_T}(\mathbf{u})$	.....	Soft data event (of **sd**)

Similarity

$s\langle \mathbf{x}, \mathbf{y} \rangle$	.....	Similarity of $\mathbf{x}$ to $\mathbf{y}$
$d\langle \cdot, \cdot \rangle$	.....	Dissimilarity (Distance)
$s_h\langle \cdot, \cdot \rangle$	.....	Hard similarity (of $\mathbf{dev_T}(\mathbf{u})$ and $\mathbf{pat}_{\mathbf{T}}^{k}$)
$s_s\langle \cdot, \cdot \rangle$	.....	Soft Similarity (of $\mathbf{sdev_T}(\mathbf{u})$ and $\mathbf{spat}_{\mathbf{T}}^{k}$)

Bibliography

G. Arpat and J. Caers. A multi-scale, pattern-based approach to sequential simulation. SCRF Annual Meeting Report 16, Stanford Center for Reservoir Forecasting, Stanford, CA 94305-2220, 2003.

G. Arpat and J. Caers. A multiple-scale, pattern-based approach to sequential simulation. In *GEOSTAT 2004 Prooceedings*, Banff, Canada, October 2004. 7th International Geostatistics Congress.

M. H. Austern. *Generic Programming and the STL: Using and Extending the C++ Standard Template Library*. Addison-Wesley Pub. Co., New York, 1st edition, 1998.

P. Avseth. *Combining rock physics and sedimentology for reservoir characterization in North Sea turbidite systems*. PhD thesis, Stanford University, Stanford, CA, USA, 2000.

Z. Bar-Joseph. Statistical learning of multi-dimensional textures. Master's thesis, The Hebrew University of Jurusalem, Jurusalem, Israel, June 1999.

J. S. D. Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Computer Graphics*, pages 361–368. ACM SIGGRAPH, 1997.

G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34(3):344–371, June 1986.

J. Caers, P. Avseth, and T. Mukerji. Geostatistical integration of rock physics, seismic amplitudes and geological models in north-sea turbidite systems. In *SPE ATCE Prooceedings*, number SPE 71321. Society of Petroleum Engineers, October 2001.

J. Caers and A. Journel. Stochastic reservoir simulation using neural networks trained on outcrop data. In *SPE ATCE Prooceedings*, number SPE 49026. Society of Petroleum Engineers, September 1998.

J. Caers, S. Srinivasan, and A. Journel. Geostatistical quantification of geological information for a fluvial-type north sea reservoir. In *SPE ATCE Prooceedings*, number SPE 56655. Society of Petroleum Engineers, October 1999.

J. Caers, S. Strebelle, and K. Payrazyan. Stochastic integration of seismic and geological scenarios: A submarine channel saga. *The Leading Edge*, pages 192–196, March 2003.

J. Caers and T. Zhang. Multiple-point geostatistics: a quantitative vehicle for intergration geologic analogs into multiple reservoir model. In *Integration of outcrop and modern analog data in reservoir models*, pages 383–394. AAPG memoir 80, 2004.

G. Caumon and A. Journel. Early uncertainty assessment: Application to a hydrocarbon reservoir development. In *GEOSTAT 2004 Prooceedings*, Banff, Canada, October 2004. 7th International Geostatistics Congress.

P. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980.

C. Deutsch. *Annealing Techniques Applied to Reservoir Modeling and Integration of Geological and Engineering (Well Test) Data*. PhD thesis, Stanford University, Stanford, CA, 1992.

C. Deutsch. *Geostatistical Reservoir Modeling*. Oxford University Press, Oxford, 2002.

C. Deutsch and A. Journel. *GSLIB: Geostatistical Software Library*. Oxford University Press, Oxford, 2nd edition, 1998.

C. Deutsch and L. Wang. Hierarchical object-based stochastic modeling of fluvial reservoirs. *Mathematical Geology*, 28(7):857–880, 1996.

O. Duda, P. Hart, and D. Stork. *Pattern Clasification*. John Wiley & Sons, Inc., New York, 2nd edition, 2001.

A. Efros and W. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH 2001 Prooceedings*, Los Angeles, CA, Aug. 2001.

C. Farmer. Numerical rocks. In P. King, editor, *The mathematical generation of reservoir geology*, pages 22–33. Clarendon Press, Oxford, 1990.

R. C. Gonzalez and R. Woods. *Digital Image Processing*. Prentice Hall, 2nd edition, 2002.

P. Goovaerts. *Geostatistics for natural resources evaluation*. Oxford University Press, Paris, France, 1997.

F. Guardiano and M. Srivastava. Multivariate geostatistics: Beyond bivariate moments. In A. Soares, editor, *Geostatistics-Troia*, pages 133–144. Kluwer Academic Publications, Dordrecht, 1993.

M. Hagedoorn. *Pattern matching using similarity measures*. PhD thesis, Universiteit Utrecht, Utrecht, Holland, 2000.

H. Haldorsen and E. Damsleth. Stochastic modeling. *JPT*, (SPE 20321):404–412, April 1990.

A. Harding, S. Strebelle, and M. Levy. Reservoir facies modeling: New advances in mps. In *GEOSTAT 2004 Prooceedings*, Banff, Canada, October 2004. 7th International Geostatistics Congress.

T. Hoffman and J. Caers. History matching under geological control: application to a north sea reservoir. In *GEOSTAT 2004 Prooceedings*, Banff, Canada, October 2004. 7th International Geostatistics Congress.

L. Holden, R. Hauge, O. Skare, and A. Skorstad. Modeling of fluvial reservoirs with object models. *Mathematical Geology*, 24(5):473–496, July 1998.

A. Journel. Geostatistics for conditional simulation of ore bodies. *Economic Geology*, 69(5):673–687, 1974.

A. Journel. The indicator approach to estimation of spatial distributions. In *17th APCOM Symposium Prooceedings*. Society of Mining Engineers, April 1982.

A. Journel. Geostatistics: Roadblocks and challenges. In A. Soares, editor, *Geostatistics-Troia*, pages 213–224. Kluwer Academic Publications, Dordrecht, 1992.

A. Journel. Modeling uncertainty: Some conceptual thoughts. In R. Dimitrakopoulos, editor, *Geostatistics for The Next Century*, pages 30–43. Kluwer Academic Publications, Dordrecht, Holland, 1994.

A. Journel. Combining knowledge from diverse sources: An alternative to traditional data independence hypotheses. *Mathematical Geology*, 34(5):573–596, 2002.

A. Journel and F. Alabert. New method for reservoir mapping. *JPT*, (SPE 18324): 212, Feb. 1990.

A. Journel and C. Deutsch. Entropy and spatial disorder. *Mathematical Geology*, 25 (3):329–355, 1993.

D. Knuth. *Art of Computer Programming*. Addison-Wesley Pub. Co., 3rd edition, 1997.

S. Krishnan, A. Boucher, and A. Journel. Evaluating information redundancy through the tau model. In *GEOSTAT 2004 Prooceedings*, Banff, Canada, October 2004. 7th International Geostatistics Congress.

L. Liang, C. Liu, T. Xu, B. Guo, and G. Shum. Real-time texture synthesis by patch-based sampling. Technical Report MSR-TR-2001-40, Microsoft Corporation, 2001.

Y. Liu. An information content measure using multiple-point statistics. In *GEOSTAT 2004 Prooceedings*, Banff, Canada, October 2004. 7th International Geostatistics Congress.

Y. Liu, A. Harding, and R. Gilbert. Multiple-point geostatistical simulation. In *GEOSTAT 2004 Prooceedings*, Banff, Canada, October 2004. 7th International Geostatistics Congress.

A. Maharaja. 3D stochastic modeling of the rhine-meuse delta using multiple-point geostatistics. In *GEOSTAT 2004 Prooceedings*, Banff, Canada, October 2004. 7th International Geostatistics Congress.

G. Matheron. *Traite de geostatistique applique*. Edition Technip, Paris, France, 1962.

R. Paget and I. D. Longstaff. Texture synthesis via a noncausal nonparametric multiscale markov random field. *IEEE Transactions on Image Processing*, 7(6):925–931, June 1998.

S. Palmer. *Vision Science: Photons to Phenomenology*. MIT Press, Massachusetts, 1999.

K. Payrazyan. *Modelization 3D des reservoirs petroliers par l'integration des donnees sismiques et geologiques: Approches quantitatives multivariables*. PhD thesis, Ecole Nationale Superieur de Geologie: Institut National Polytechnique de Loraine, Nancy, France, 1998.

J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *Int'l Journal of Computer Vision*, 40(1):49–71, October 2000.

Y. Rubner, C. Tomasi, and L. Guibas. The earth mover's distance as a metric for image retrieval. Technical Report STAN-CS-TN-98-86, Computer Science Department, Stanford University, Stanford, CA 94305-2220, September 1998.

R. Sedgewick. *Algorithms*. Addison-Wesley Pub. Co., New York, 2nd edition, 1988.

M. Smid. Closes-point problems in computational geometry. In J. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier Science, Amsterdam, Holland, 1999.

M. Srivastava. Iterative methods for spatial simulation. SCRF Annual Meeting Report 5, Stanford Center for Reservoir Forecasting, Stanford, CA 94305-2220, May 1992.

S. Strebelle. *Sequential Simulation Drawing Structures from Training Images*. PhD thesis, Stanford University, 2000.

S. Strebelle. Conditional simulation of complex geological structures using multiple-point statistics. *Mathematical Geology*, Jan. 2002.

S. Strebelle, K. Payrazyan, and J. Caers. Modeling of a deepwater turbidite reservoir conditional to seismic data using multiple-point geostatistics. In *SPE ATCE Proocedings*, number SPE 77425. Society of Petroleum Engineers, Oct. 2002.

H. Tjelmeland. *Stochastic Models in Reservoir Characterization and Markov Random Fields for Compact Objects*. PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway, 1996.

T. Tran. Improving variogram reproduction on dense simulation grids. *Computers and Geosciences*, 20(7):1161–1168, 1994.

I. Tureyen and J. Caers. A parallel scheme for multi-scale data integration. In *GEOSTAT 2004 Prooceedings*, Banff, Canada, October 2004. 7th International Geostatistics Congress.

S. Viseur. Stochastic boolean simulation of fluvial deposits: A new approach combining accuracy with efficiency. In *SPE ATCE Proceedings*, volume October, Society of Petroleum Engineers 1999.

Z. Wang, Y. Yu, and D. Zhang. Best neighborhood matching: An information loss restoration technique for block-based image coding systems. *IEEE Transactions on Image Processing*, 7(7):1056–1061, July 1998.

L. Wei. *Texture synthesis by fixed neighborhood searching*. PhD thesis, Stanford University, Stanford, CA, USA, 2001.

R. Wen, A. Martinius, A. Nass, and P. Ringrose. Three-dimensional simulation of small-scale heterogeneity in tidal deposits - a process-based stochastic simulation method. In *IAMG 1998 Proocedings*. International Association for Mathematical Geology, 1998.

J. Wu and A. Journel. Water saturation prediction using 4D seismic data. SCRF Annual Meeting Report 17, Stanford Center for Reservoir Forecasting, Stanford, CA 94305-2220, May 2004.

W. Xu. Conditional curvilinear stochastic simulation using pixel-based algorithms. *Mathematical Geology*, 28(7):937–949, 1996.

Y. Q. Xu, B. N. Guo, and H. Shum. Chaos mosaic: Fast and memory efficient texture synthesis. Technical Report MSR-TR-2000-32, Microsoft Corporation, 2000.

T. Zhang, P. Switzer, and A. Journel. Sequential conditional simulation using classification of local training patterns. In *GEOSTAT 2004 Prooceedings*, Banff, Canada, October 2004. 7th International Geostatistics Congress.