Synthèse de texture Mono-Echelle

Comme expliqué précédemment, les méthodes géostatistiques basées sur l'utilisation d'un variogramme ne prennent en compte que les corrélations entre deux points du réservoir. Les objets complexes de forme curvilinéaire tels que les chenaux ne peuvent être modélisés par deux points. Les méthodes de simulation multipoints, qui dépendent de plus de deux points, ont été proposées pour pallier cette difficulté. L'idée principale consiste à utiliser une image d'entrainement pour y analyser les statistiques multipoints. Puis, ces statistiques sont reproduites pendant l'étape de simulation.

Dans ce chapitre on se propose d'adapter une méthodologie multipoints, appelée synthèse de texture, pour la modélisation de réservoir. Cette approche largement utilisée dans le domaine du logiciel depuis une dizaine d'années s'est développée, en parallèle/en même temps que les nouveaux algorithmes de simulation multipoints développés en géostatistique.

La section suivante fait l'état de l'art de ces méthodes selon leur appartenance à l'infographie ou aux géostatistiques pour le réservoir. Puis on décrit en détails les étapes de l'algorithme mis au point. On procède à une analyse de sensibilité des différents paramètres, ainsi qu'à une étude de la mesure de distance entre deux motifs. Finalement on explique et on teste deux techniques pour diminuer le temps de calcul.

4.1. Etat de l'art

4.1.1. Géostatistiques multipoints

L'idée première pour modéliser les chenaux a été de modéliser des objets au lieu de simuler une valeur à un point donné. Le développement des méthodes appelées méthodes de génération par objet débute à notre connaissance avec Bridge et Leeder en 1979, et se poursuit avec Haldorsen et Damsleth (1990), Omre (1991) ou encore Deutsch et Wang (1996). Elles consistent à définir la forme de l'objet sur la base de quelques paramètres géométriques. En raison des incertitudes, les paramètres sont caractérisés par les probabilités déduites des données ou des observations. Puis ces objets aux formes tirées aléatoirement, sont positionnés sur un ensemble de points obtenus par un processus de Poisson. Bien que séduisante cette méthode se heurte à deux grandes difficultés. Le calcul des paramètres géométriques est loin d'être simple, ainsi que le conditionnement aux données dures.

Vient ensuite l'idée d'utiliser une statistique multipoint déduite à partir d'une image d'entrainement pour la simulation géostatistique. L'image d'entrainement est une représentation conceptuelle des structures spatiales attendues dans le réservoir. Elle peut provenir d'une simulation précédente, d'une image satellite, d'une image élaborée par un géologue en fonction de la connaissance qu'il a du mileu, *etc*. Cette méthode a été initialement proposée par

Guardiano et Srivastava en 1993. La motivation de ces auteurs était d'étendre la méthode de simulation séquentielle par indicatrices pour rendre compte d'objets géologiques d'architecture plus complexe sans construire complètement et explicitement un modèle de fonction aléatoire non-Gaussienne (Journel, 2005). Les premiers algorithmes de simulation multipoints impliquaient de très long temps de calcul, car il fallait scanner l'image d'entrainement à chaque fois qu'une valeur devait être simulée pour une maille. Le premier algorithme efficace, appelé SNESIM, pour Single Normal Equation Simulation, a été développé par Strebelle (2000) et permet de simuler des caractéristiques complexes pour des variables discrètes. Le point clé de cet algorithme est que les diverses configurations ou événements de faciès extraits de l'image d'entrainement sont stockés dans une structure arborescente. Cette base de données ordonnée est ensuite utilisée pour calculer les fonctions de densité de probabilité conditionnelles lors de la simulation. Cet algorithme marque une étape importante dans l'utilisation des méthodes géostatistiques multipoints, car il les rend accessibles en termes de temps de calcul, même si l'utilisation de la RAM reste un de ces désavantages majeurs. De nombreux travaux récents cherchent à améliorer cet algorithme. Straubhaar *et al.* (2011,2013), dans IMPALA, ont proposé de remplacer la classification arborescente par une classification mixte en liste et en arbre, pour pallier le problème de stockage en mémoire d'un arbre pour des images 3D.

En 2007, Arpat et Caers abandonnent le cadre probabiliste (reproduction de statistiques) au profit de la reproduction de motifs utilisée en infographie. L'image d'entrainement est utilisée comme base de données de motifs, un motif étant un morceau d'image (comme un puzzle mais de forme fixe). Une propriété importante de ce motif est qu'il doit caractériser les variations spatiales de la géologie du réservoir. Pour simuler une réalisation, chaque pixel est visité de façon aléatoire, et pour attribuer une valeur à ce pixel on colle un des motifs de l'image d'entrainement directement dessus, au lieu de tirer une valeur suivant un modèle de probabilité. Toute la qualité de l'algorithme, et de la reproduction des caractéristiques de l'image d'entrainement, tient à la façon de choisir le motif à coller et de le coller en tenant compte des données dure. Selon Arpat et Caers, la méthode est cependant plus lente que SNESIM, et la méthode ne résout pas les conflits qu'il peut y avoir entre le motif collé et les données dures. Pour diminuer le temps de simulation, les auteurs proposent d'utiliser une solution proposée par Zhang *et al.* (2006) avec FILTERSIM. Le coût de calcul est réduit grâce à un filtrage : le motif qui peut faire jusqu'à une centaine de pixels est filtré et sa représentation est alors restreinte aux scores issus des différents filtres. Le gain en temps de calcul est important mais au détriment de la qualité de reproduction de l'image d'entrainement. Dans le même courant d'idée en 2009, Gloaguen et Dimitrakopoulos utilisent la transformée en ondelettes discrètes à la place des filtres pour réduire les dimensions des motifs.

En 2010, Honarkhah et Caers proposent de jouer non pas sur la dimension du motif mais sur le nombre de motifs que l'on va comparer lors de la simulation. Ils introduisent la classification par noyaux k-means et une mesure de distance entre deux motifs, qui leur permettent de regrouper les motifs selon leur ressemblance. Lors de la simulation d'un ensemble de pixels seuls les motifs du groupe le plus proches sont utilisés. En 2010, Mariethoz *et al.* proposent dans leur algorithme d'échantillonnage direct, de ne scanner qu'une partie de l'image d'entrainement et surtout de ne pas choisir le meilleur motif mais le premier qui est en-dessous d'une distance choisie au préalable. Cette méthode permet de diminuer la mémoire occupée car l'image d'entrainement n'est pas stockée sous forme de base de données et de plus elle permet d'augmenter la variabilité des motifs reproduits en ne prenant pas le motif le plus proche mais seulement un motif assez proche.

En 2012, Tahmasebi *et al*. Proposent l'algorithme CCSIM (cross-correlation simulation) où ils n'utilisent plus la mesure de distance entre deux motifs mais une fonction de corrélation-croisée plus rapide à calculer. De plus ils utilisent un chemin régulier pour visiter les pixels et lorsqu'ils

arrivent près d'une données dures, s'il n'y a pas de motifs qui s'ajustent à la donnée, ils réduisent la taille du motif jusqu'à en trouver un qui s'y ajuste, le cas limite étant un motif réduit à un pixel.

En parallèle, des concepts similaires sont apparus dans l'infographie et traitement de l'image, regroupés sous le nom d'algorithmes de synthèse de texture. La synthèse de texture est plus généralement utilisée dans des domaines tels que les jeux vidéo, le cinéma (film d'animation) ou encore le design d'objet.

4.1.2. La synthèse de texture

Les travaux avant-gardistes de Shannon en 1948 lancent les bases de la synthèse de texture. Ils visent à reproduire un texte en utilisant la méthode des *n*-gram. L'idée est que *n* lettres consécutives (un mot) déterminent complètement la distribution des lettres suivantes. Un problème majeur est l'obtention des tables de probabilités pour chaque n-gram. Cette approche requiert beaucoup d'échantillons, probablement autant que dans un livre par exemple.

En 1974 Catmull, et plus tard en 1976, Blinn et Newell, introduisent le placage de texture (texture mapping) qui sert à habiller un objet en 3D à l'aide d'échantillons de texture 2D. La texture est appliquée sur l'objet en déformant l'image de telle sorte qu'elle épouse les surfaces courbes de l'objet. Cette méthode tient aussi compte de la direction de la lumière pour rendre à l'objet son effet 3D, ainsi que des propriétés de réflexion de la surface. Cependant, elle génère des problèmes de paramétrisation en 2D comme des distorsions de l'image pour des surfaces complexes (par exemple, pour des surfaces bicubiques ou de genre élevé). Elle ne s'applique correctement que lorsque la surface à habiller est de même aire que l'image de la texture. Si la surface de l'objet est plus grande que celle de la texture, il faut paver l'objet afin de conserver un certain niveau de détail. En effet, on comprend aisément qu'on ne peut pas trop agrandir une image sans qu'elle se pixélise. Néanmoins, cette méthode n'est pas toujours une solution puisqu'elle tend à créer des contours artificiels sauf si la texture est répétitive. Même dans ces conditions, l'œil humain perçoit très facilement les mauvais alignements et les répétitions (Palmer, 1999) ce qui réduit la qualité de l'image créée.

Une solution (Heeger et Bergen, 1995) consiste à générer une texture semblable à l'échantillon de départ directement sur la surface de l'objet. La texture générée n'est pas une copie ou un pavage. Il suffit que l'image échantillon soit suffisamment grande pour capturer le motif principal de la texture. Toutes les méthodes dérivées de cette idée relèvent de la génération de textures.

Dans sa thèse, Duranleau (2008) regroupe les méthodes de génération de texture en fonction des catégories suivantes :

- Méthodes paramétriques : basées sur un modèle statistique donné, elles génèrent de nouvelles textures selon la distribution des pixels (équivalents aux mailles d'une grille) observés en réponse à divers filtres simulant la perception humaine. Une description assez complète de ces méthodes a été rapportée par Portilla et Simoncelli (2000).

- Génération par pixel : basées sur l'échantillonnage de la texture et la réorganisation des pixels
    - méthodes multi-résolutions : construction d'une pyramide de la structure source et remplissage de la pyramide de la texture à générer niveau par niveau.

    - méthodes par recherche de voisinage : recherche un pixel dans la texture source dont le voisinage est similaire au voisinage courant du pixel à générer.
    - optimisation globale : l'échantillonnage de texture se fait par l'entremise d'une optimisation itérative sur l'ensemble de la texture générée.

- Génération par patch : même principe que la génération par pixel, mais au lieu de générer la réalisation pixel par pixel, on la génère patch par patch (*i.e.*, par groupe de pixels).

- Méthodes par analyse structurale : enrichissement des méthodes précédentes par analyse de la texture source pour en extraire la structure et guider ou modifier la génération en fonction de cette information.

- Génération adaptée aux surfaces : cette méthode améliore aussi la précédente en introduisant une déformation de l'image pour tenir compte de la courbure de la surface pour les objets 3D.

Nous nous concentrons ici sur les méthodes de génération par pixel ou par patch, qui sont les plus adaptées au contexte de modélisation de réservoir. Dans la suite nous détaillons les méthodes par recherche de voisinage. Les méthodes multi-résolution qui se prêtent bien au contexte multi-échelles seront détaillées dans le chapitre 5 section 5.1 .

Pour simuler un pixel dans les méthodes de recherche par voisinage, on regarde ses voisins et on lance la recherche dans l'image d'entrainement pour trouver un ensemble de pixels candidats dont le voisinage est similaire. La valeur d'un des pixels candidats est alors attribuée au pixel simulé. On passe ensuite au pixel suivant. Le type de voisinage, la méthode de recherche et l'ordre de génération varient selon la méthode employée.

En 1999, Efros et Leung proposent un algorithme de synthèse de texture qui donne de très bons résultats sur la réparation d'images (hole-filling). Leur voisinage est un carré centré sur le pixel en cours de simulation. Tous les pixels déjà simulés contenus dans ce voisinage servent pour la comparaison avec l'image d'entrainement. A chaque simulation de pixel l'image d'entrainement est scannée et les pixels dont le voisinage est proche de celui simulé sont extraits. A partir de ces informations, on construit une fonction de densité de probabilité pour la valeur du pixel simulé. On fait ensuite un tirage aléatoire, pour lui attribuer une valeur. Cet algorithme a un gros défaut, il est très lent et ne marche que pour les textures assez répétitives. Une méthode plus rapide a été proposée par Wei et Levoy (2000). Les pixels à simuler sont visités selon un chemin unilatéral et le voisinage à une forme en 'L', de sorte que seuls les pixels qui viennent d'être visités sont pris en compte. De plus, ils introduisent eux aussi la classification par arbre des motifs extraits de l'image d'entrainement.
De nombreux travaux portent sur l'accélération de l'algorithme de Wei et Levoy (2000). On citera Ashikmin (2001), qui propose de ne scanner que les voisinages des voisins du pixel à simuler. Zelinka et Garland (2004) construisent une carte de saut, c'est-à-dire que toutes les distances motifs à motifs sont calculées dans un premier temps, et sauvegardées. Lors de la simulation, ces auteurs vont s'en servir pour guider la recherche de motifs à scanner. Un état de l'art sur les algorithmes de synthèse de texture a été fait en 2009 par Wei *et al.*, pour plus de détails sur les autres méthodes de synthèse de texture.

En 2014, Mariethoz et Lefebvre passent en revue les liens qui existent entre la synthèse de texture et les géostatistiques multipoints. Ils soulignent le fait que les deux domaines partagent beaucoup de points communs mais ont été développé selon des axes différents. Les

algorithmes de synthèse de texture doivent être très efficaces en temps de calcul et très bons en reproduction de motifs, alors que les algorithmes de géostatistiques multipoints eux doivent intégrer les données dures et produire des réalisations stochastiques 3D.

C'est en 2007 que deux algorithmes de géostatistiques multipoints font état de leur lien avec l'infographie. Il s'agit de l'algorithmes SIMPAT de Arpat et Caers (2007) et d'un autre développés par Daly et Knudby (2007). Les différences sont dans le chemin de visite aléatoire pour l'intégration des données qui ne se fait que très peu en infographie. C'est Parra et Ortiz en 2011 qui proposent une adaptation d'un algorithme de synthèse de texture pour la simulation conditionnelle de réservoir. Ils utilisent un chemin de visite régulier comme pour Wei et Levoy (2000), mais leur voisinage est composé de deux 'L' emboités. Le premier 'L', appelé voisinage causal, contient les pixels que l'on vient de simuler. Le deuxième 'L', appelé voisinage non-causal, contient les pixels à venir. Lorsque le voisinage non-causal est vide, *ie* il ne contient aucune valeur, alors on ne compare que le voisinage causal à l'image d'entrainement. Lorsque le voisinage non-causal contient des données dures alors les deux voisinages sont utilisés pour la comparaison avec l'image d'entrainement. Cela permet d'avoir un temps de calcul raisonnable en ayant un plus petit voisinage dans la majorité des comparaisons, et de prendre en compte les données dures en amont pour éviter les problèmes de connectivité ou de continuité.

Un autre algorithme issu de l'infographie est étudié maintenant en géosciences, c'est la méthode d'image quilting pour la simulation conditionnelle (Mahmud *et al.*, 2014). Introduit en 2001 par Efros et Freeman, l'image quilting consiste à paver la grille avec des morceaux d'images. Sa particularité est que les patchs sont collés initialement plus grands que nécessaire et ensuite découpés de manière à minimiser l'erreur sur les bords et à assurer la continuité.

L'algorithme développé dans ce chapitre s'inspire de celui d'Arpat et Caers (2007). Il combine les techniques de simulation géostatistiques avec la synthèse de texture. Bien ancré dans la synthèse de texture, cet algorithme est conçu pour les applications de modélisation géologiques. Notre travail diffère de l'algorithme SIMPAT de par l'utilisation d'un chemin régulier de simulation et donc du challenge pour intégrer les données dures. De plus nous nous concentrons sur diverses techniques permettant de le rendre plus flexible et plus rapide, et proposons quelques modifications pour permettre l'intégration des données dures avec un chemin régulier sans perdre la continuité dans l'image finale.
Dans la première section, on explique pas à pas l'algorithme pour la simulation non-conditionnelle, puis pour la simulation conditionnelle. Dans la deuxième section, nous examinons deux méthodes pour diminuer le temps de calcul et nous les testons pour voir leur effet sur la qualité de l'image finale.

4.2. Algorithme

4.2.1. Simulation non-conditionnelle

L'image d'entrainement doit être premièrement scannée à l'aide d'un template pour en extraire des motifs et créer ainsi une base de données de motifs. Notre choix par défaut pour le template s'est porté sur une fenêtre carrée centrée sur le pixel en train d'être simulé (Figure 24).

Comme il n'y a pas de données dures, la phase d'initialisation de la grille à simuler débute par le choix d'un patch aléatoirement dans l'image d'entrainement et par son collage au milieu de la grille. Ensuite la grille est remplie couronne par couronne autour du patch collé (Figure 24). Une couronne est définie à chaque étape à partir des pixels non simulés voisins directs de pixels déjà simulés. La première couronne de pixel est simulée. Ensuite, on prend les pixels non simulés à coté et on génère de nouvelles valeurs pour ces pixels. Ce processus est répété jusqu'à ce que la grille soit entièrement simulée. Cet ordre de visite est appelé « régulier » par contraste avec l'ordre de visite « aléatoire » qui est généralement utilisé pour les simulations géostatistiques. Pour chaque pixel dans une couronne, on compte le nombre de pixels voisins déjà simulés. Puis on visite les pixels de la couronne de celui qui a le plus de voisins à celui qui en a le moins. Avant d'aller plus loin, il serait utile de mentionner le fait que la simulation se fait patch par patch au lieu de pixel par pixel, et que la taille du patch est plus petite que la taille du template. Cela permet de réduire le temps de calcul et de mieux préserver la continuité spatiale des structures géologiques dans la nouvelle image.
I est l'image simulée à partir de l'image d'entrainement *TI*. *p* est le patch de pixels en train d'être simulés sur *I* à l'itération courante, il est centré sur le pixel *i*. Le cas limite apparaît lorsque *p* ne contient que *i*. *w* désigne le template. Lorsqu'il est centré sur *i*, on appelle *w(i)* les pixels voisins de *i* appartenant à *w*. Les étapes successives de l'algorithme sont listées ci-dessous.

Algorithme :

- Initialisation: sélectionner un patch aléatoirement dans la *TI* et le coller au milieu de la grille à simuler *I*. Le patch pourrait être collé n'importe où ailleurs sur la grille.
- Jusqu'à ce que tous les pixels soient visités une fois
    - A partir des pixels déjà simulés, identifier une couronne de pixels voisins non simulés
    - Trier ces pixels selon leur nombre de voisins déjà simulés, de celui qui en a le plus à celui qui en a le moins
    - Les simuler un à un selon cet ordre
        - Identifier les voisins *w(i)* du pixel courant selon le template choisi
        - Comparer *w(i)* à la base de données des motifs extraits de la *TI* et sélectionner le plus proche de *w(i)*. La distance *d* entre les motifs et *w(i)* est définie par une mesure donnée. Ici par défaut le choix est la norme L2 où chaque terme a un poids qui dépend de son éloignement au pixel simulé. Ce point sera développé dans la section 4.3.2
        - Sur le motif choisi, le patch central est extrait et collé sur le groupe de pixel autour de *i* sur *I*. Précision : les pixels déjà simulés ne sont pas effacés, c'est-à-dire que l'on ne colle de nouvelles valeurs que sur les pixels qui n'en contiennent pas.

Jusqu'à maintenant toute l'image d'entrainement devait être scannée pour chaque patch simulé pour identifier le motif le plus proche du template. D'autres stratégies sont envisagées pour améliorer le temps de calcul et exposées dans la section 4.4.

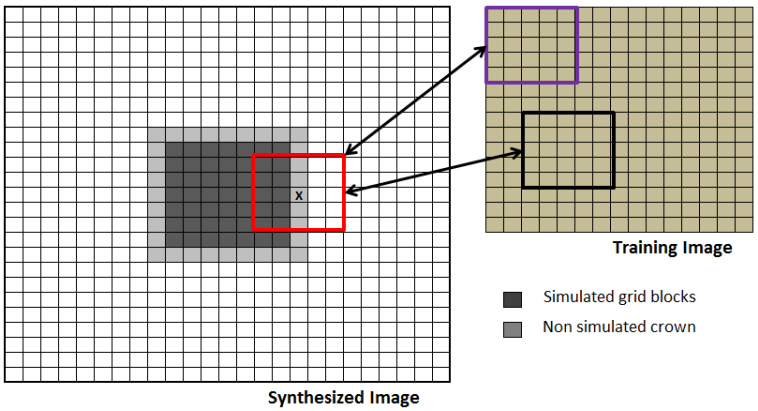


Figure 24 Schéma de synthèse pour attribuer une valeur à un patch. Sur la gauche, l'image synthétisée *I*. Sur la droite l'image d'entrainement *TI*. On veut attribuer une valeur au patch *p* (*i* est marqué par un x) en comparant *w(i)* (ici en rouge à gauche) aux motifs extraits (ici en exemple violet et noir) de l'image d'entrainement (à droite).

A ce stade on peut préciser que chaque fois que l'on change le germe de la fonction de génération de nombres pseudo-aléatoires, on change à l'initialisation le motif extrait de l'image d'entrainement. Cela se traduit par la génération de différentes réalisations. Cependant, une fois le motif initial choisi, le processus de simulation défini ci-dessus est déterministe : en effet lorsque l'on remplit la grille on choisit les meilleurs patchs, ceux qui ont la plus petite distance au voisinage. Une variante de l'algorithme qui permet d'utiliser les divers algorithmes de perturbations tels que la déformation graduelle (Hu, 2000) ou la méthode de perturbation des probabilités (Caers, 2003), est présenté section 4.2.3.

4.2.2. Simulation conditionnelle

L'utilisation d'un chemin de visite aléatoire est habituellement recommandée pour permettre l'intégration des données dures. Dans cette section, nous montrons comment prendre en compte les données dures tout en gardant un chemin régulier et un processus de génération par couronnes.

Intuitivement, on voit que la taille du template est directement liée au temps de calcul. Par conséquent, on va préférer de petits templates pour réduire le temps de simulation. Cela veut dire d'un autre coté que si on commence l'initialisation au centre de la grille et que l'on grandit par couronnes concentriques, les données dures ne seront détectées que lorsqu'elles seront proches du pixel en cours de simulation. Donc à un moment donné autour du pixel simulé, on aura tous les pixels qui ont déjà été simulés sans tenir compte de la donnée dure qu'ils n'ont pas vus, et la donnée dure qui peut être très différente. Il est fort à parier que cela va créer de grosses discontinuités dans l'image.

C'est la raison pour laquelle les algorithmes de géostatistiques multipoints utilisent généralement un chemin aléatoire qui atténue cet effet. Toutefois, avec un chemin aléatoire, il faut de plus grands templates pour capturer correctement les structures de l'image d'entrainement, ce qui signifie un temps de calcul plus important. Une autre option est proposée par Parra et Ortiz (2011) comme on l'a évoqué dans l'état de l'art (section 4.1.2). Ils ont proposé de considérer des voisinages carrées divisés en deux domaines : l'un contenant les pixels déjà simulés, et l'autre ceux à venir. Seul le premier est utilisé si le deuxième ne contient pas de données. Cela réduit partiellement les discontinuités. Cependant, il y a de nombreux cas où il faut réconcilier dans un même template les données dures et les pixels qui ont été simulés sans

tenir compte des données. De tels motifs ont peu de chance d'exister dans la base de données, créant ainsi de nouveau des discontinuités dans la réalisation finale.

Nous proposons une approche différente. L'étape d'initialisation change. Elle doit maintenant tenir compte de la présence de données dures. On va donc créer une figure géométrique fermée en reliant les données deux à deux par des segments. Ensuite, nous simulons des valeurs uniquement pour les pixels traversés par ces segments. Ceci est effectué en utilisant le même procédé que pour les couronnes, à savoir les pixels avec les plus de voisins non-vides sont simulés en premier. Une fois les pixels des segments simulés, on obtient une figure fermée déjà simulée et on revient au processus de construction par couronnes présenté dans la section 4.2.1. Dans ce cas, les couronnes comprennent les pixels à l'intérieur et à l'extérieur de la figure fermée.

Un exemple avec trois données dures est représenté sur la Figure 25. Les pixels gris sont les pixels non simulés, les noirs représentent les chenaux et les blancs le milieu encaissant. Dans la Figure 25 a), on voit la grille non simulée et l'emplacement de 3 données dures qui signalent l'emplacement de chenaux. Dans la Figure 25 b) on voit le résultat de l'étape de création de la figure fermée et de la simulation des pixels traversés par chaque segment de la figure. Dans la Figure 25 c) et d) on n'a illustré que la simulation des couronnes intérieures pour plus de clarté. Dans la Figure 25 e) on voit la simulation de la première couronne extérieure. Enfin dans la Figure 25 f) on voit la réalisation finale, les données dures sont signalées par des points rouges. On voit que les données sont bien respectées.

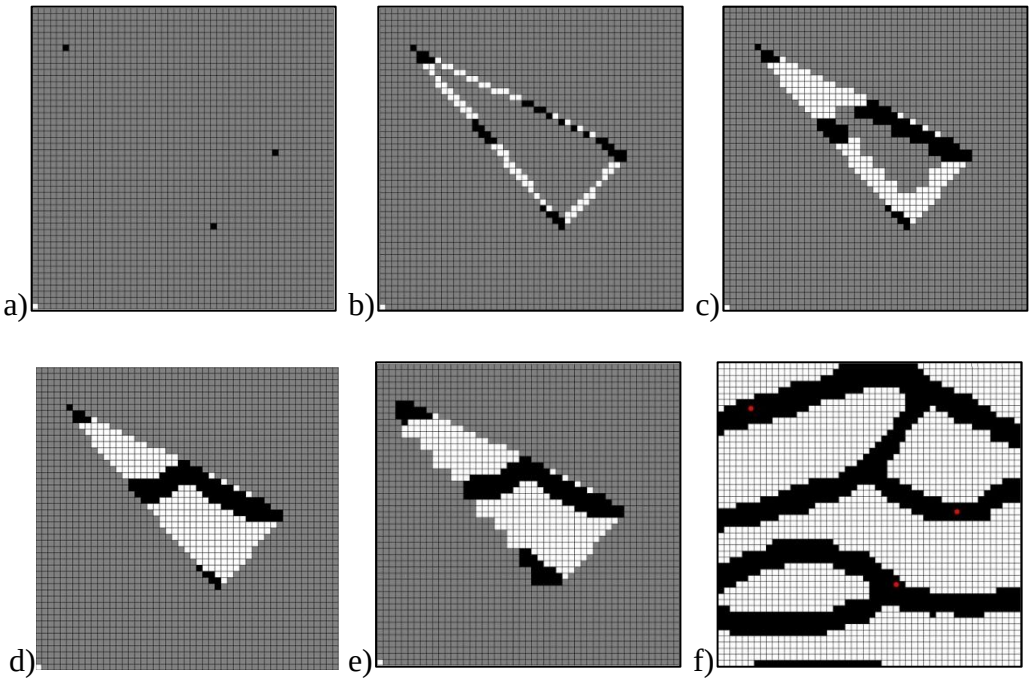


Figure 25 Schéma illustrant les étapes successives suivies pour la simulation conditionnelle. a) Emplacement des 3 données dures sur une grille de 50 × 50. b) Construction du triangle et simulation des pixels le long des segments avec un template de taille 19 × 19 pixels. c à d) Simulation de l'intérieur du triangle en propageant par couronnes. e) Simulation de la première couronne extérieure f) La réalisation finale (points rouges : emplacements de données).

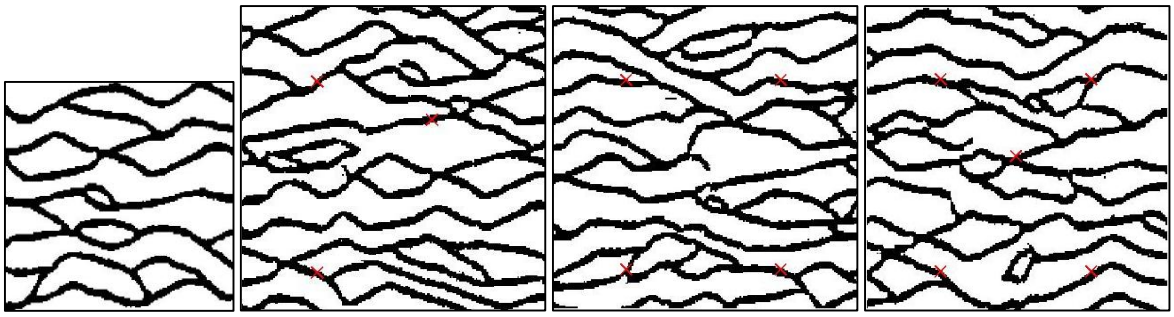


Figure 26 Simulation Conditionnelle. Première image à gauche : image d'entrainement. Autres images : réalisations simulées avec contraintes sur respectivement 3, 4 ou 5 données dures. Les croix rouges représentant les emplacements des données. Taille du template : 19×19 pixels; Taille du patch : 5×5 pixels.

D'autres exemples de réalisations conditionnelles sont présentés dans la Figure 26 avec trois, quatre et cinq données dures indiquant la présence de chenaux. Les réalisations finales obtenues avec l'algorithme proposé sont cohérentes avec les structures représentées dans l'image d'entrainement et respectent les données.

4.2.3. Déformation Graduelle

Comme dit précédemment l'algorithme présenté ci-dessus est déterministe car on choisit à chaque simulation de patch, le motif le plus proche du voisinage du patch. Pour rendre cet algorithme compatible avec la méthode de déformation graduelle, on modifie l'étape de sélection du meilleur motif.

Lors de cette étape de sélection, au lieu de garder la meilleure adéquation entre le voisinage et les motifs extraits de l'image d'entrainement, nous gardons les *n* plus proches motifs et en choisissons un au hasard parmi ceux-ci. Le choix n'est pas totalement aléatoire. On calcule un poids pour chacun des *n* motifs extraits qui dépend de la distance entre le motif et le voisinage du patch simulé. Plus le motif est loin du voisinage plus son poids est faible, moins il a de chance d'être sélectionné.

Pour illustrer l'effet du choix parmi les 10 meilleurs motifs, sur les réalisations, nous prenons quatre données dures, signalant quatre emplacements pour les chenaux, et simulons quatre réalisations avec quatre jeux de nombres aléatoires différents. Les résultats sont montrés Figure 27. On voit que l'on obtient quatre réalisations différentes, qui respectent les données et qui restent semblables à l'image d'entrainement. On observe quelques discontinuités de chenaux, dues au fait que l'on ne prend pas le motif le plus proche du voisinage.

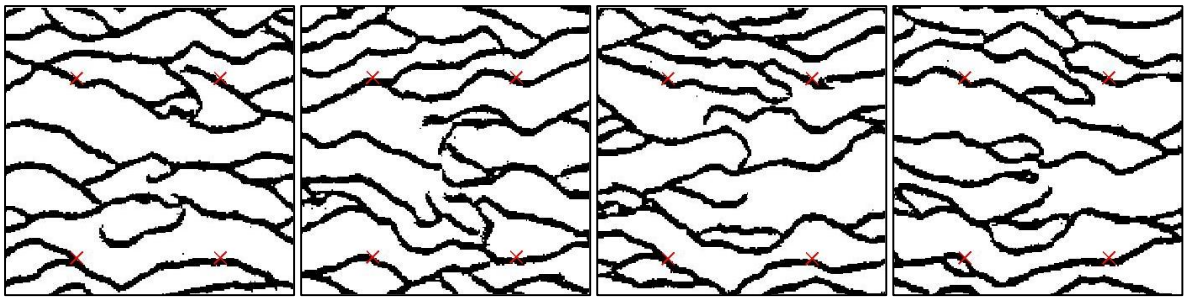


Figure 27 Quatre réalisations conditionnées sur quatre données dures en prenant quatre bruits blancs Gaussiens différents. Template : 19x19. Taille de Patch: 5x5

On rappelle brièvement que la méthode de déformation graduelle est une technique utilisée pour perturber des réalisations en géostatistique avec un nombre réduit de paramètres. Elle est basée sur la combinaison linéaire de bruits blancs Gaussiens indépendants (rappel de l'Equation 8) :

$$z(\theta) = z_1 \cdot cos(\pi\theta) + z_2 \cdot sin(\pi\theta)$$

où z_1 et z_2 sont des bruits blancs Gaussien indépendants et θ est le paramètre de déformation généralement compris entre -1 et 1.
Le fait d'utiliser un jeu de nombres aléatoires tirés d'une distribution uniforme pour choisir parmi *n* motifs, n'implique que peu de modifications pour utiliser la déformation graduelle.

Introduisons quelques notations: *G* est la fonction de répartition de la loi normale standard et u_i un jeu d'éléments tiré de la loi uniforme.

$$z(\theta) = G^{-1}(u_1) \cdot cos(\pi\theta) + G^{-1}(u_2) \cdot sin(\pi\theta)$$
$$u(\theta) = G\big(z(\theta)\big) \qquad \textbf{Equation 18}$$

On transforme donc les u_i en bruits blancs Gaussien à l'aide de la fonction probit G^{-1}. On fait une déformation graduelle pour trouver le θ optimal et on peut revenir à un jeu d'éléments uniformes en utilisant la fonction de répartition G.

4.3. Analyses de sensibilité des paramètres du modèle

Dans cette section nous discutons des performances de l'algorithme décrit ci-dessus, et de sa sensibilité sur certains paramètres comme la taille du template ou encore la mesure de la distance *d* entre les motifs et le template *w(i)*. Nous montrons l'influence du chemin de visite pour la simulation des patchs sur la qualité de l'image obtenue.
Cette analyse est basée sur un ensemble de simulations effectuées à partir de la même image d'entrainement. Elle est connue et utilisée pour illustrer le potentiel des méthodes de géostatistiques multipoints (Figure 28). Elle représente des chenaux de haute perméabilité contenu dans une matrice très peu poreuse/perméable. L'image d'entrainement fait 150×160 pixels. Dans tous les cas considérés, la réalisation fait 200×200 pixels.

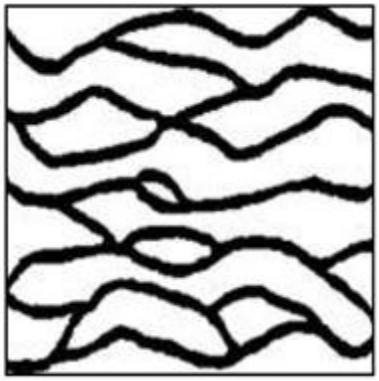


Figure 28 Image d'entrainement représentant des chenaux dans un milieu argileux

4.3.1. Quelle est l'influence de la taille du template et de la taille des patchs ?

L'algorithme présenté dans la section 4.2 implique la définition de paramètres tels que la taille du template ou encore la taille du patch collé à chaque simulation.
Les résultats de la Figure 29 montrent que la taille du voisinage est importante. Clairement un voisinage trop petit ne permet pas de capturer les hétérogénéités de grande taille présentes sur l'image d'entrainement. L'image synthétisée avec un template 5×5 (Figure 29) ne présente aucun chenal. Augmenter la taille du template de 9×9 à 19×19 (Figure 29 b et c) permet de faire apparaitre des chenaux et d'améliorer la continuité et la connectivité de la géométrie des chenaux. Cependant augmenter la taille du template diminue le nombre de motifs extraits de l'image d'entrainement. Par exemple considérons une image d'entrainement de taille 150×160, et une taille de template de 9×9, il en résulte 21 584 motifs extraits. Maintenant pour cette même image d'entrainement, augmentons la taille du template à 31×31, le nombre de motifs extraits est alors de 15 600, soit 30% de moins. On a alors un problème de représentativité. Si la base de données des motifs extraits ne contient pas assez de motifs, il est plus difficile de trouver le motif approprié lors de la simulation d'un patch. Il en résulte des discontinuités dans l'image finale (réalisation) comme dans la Figure 29 e). Une taille de template appropriée est un compromis entre les deux. Elle doit être suffisamment grande pour capturer l'hétérogénéité mais suffisamment petite pour qu'il y ait un nombre suffisant de motifs extraits.

Ici nous avons procédé par un test exhaustif de toutes les tailles de templates entre 5× 5 et 31×31, pour trouver la taille de fenêtre optimale. Il s'avère qu'un template de 19×19 donne les meilleurs résultats. Cependant quelques auteurs ont développé des méthodes pour trouver cette taille de template (Tan *et al.*, 2013). On voit aussi dans la Table 1 le temps de calcul augmente avec la taille du template.

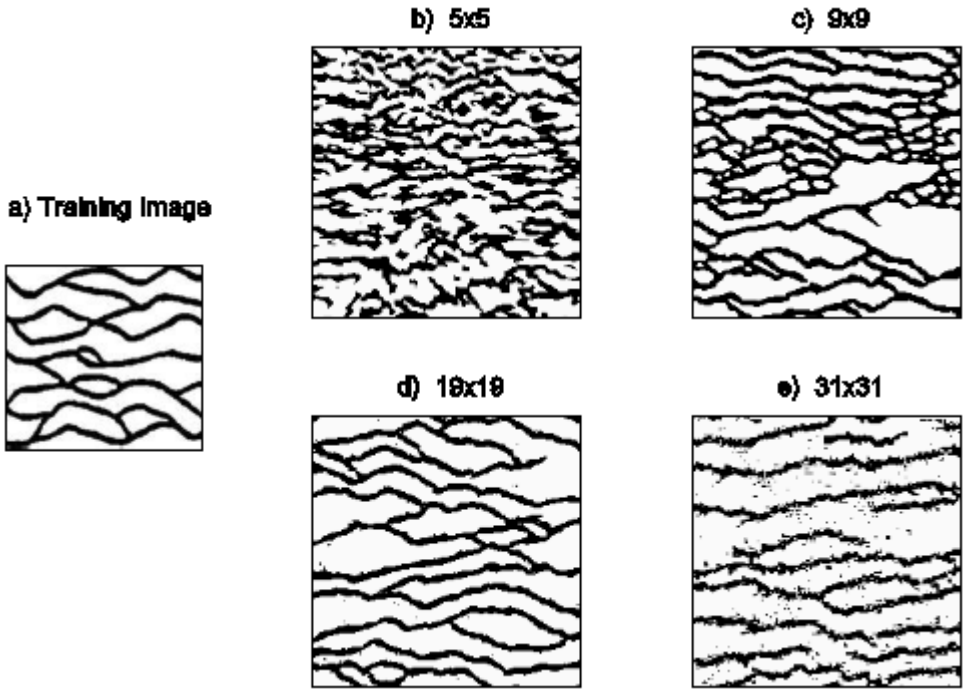


Figure 29 Influence de la taille du template. A gauche : image d'entrainement (a). Autres images : réalisations simulées avec une taille de template grandissante de b) 5×5 à e) 31×31. Taille de patch fixe : 5×5. Chemin de visite régulier.

Table 1 Impact de la taille du template sur le temps de calcul. Le temps de référence T_0 est relatif au temps de simulation avec le plus petit template 5×5 avec un patch 3×3

Taille du template	5×5	9×9	19×19	31×31
CPU-time	T_0 =42sec	$3 \cdot T_0$	$9 \cdot T_0$	$15 \cdot T_0$

On a estimé la complexité de l'algorithme à partir de deux tests. Dans le premier test, illustré Figure 30 à gauche, on trace le temps de simulation en fonction du nombre de pixels contenus dans le template. On voit clairement que la courbe n'est pas linéaire mais plutôt logarithmique. Or on sait que la complexité des algorithmes de synthèse de texture usuels est directement proportionnelle aux nombres de pixels dans le template. Dans un deuxième test, illustré Figure 30 à droite, on trace le temps de calcul en fonction du nombre de pixels dans le template mais à nombre de motifs dans la base de données fixe. On a montré ci-dessus que lorsque l'on augmente la taille du template on diminue le nombre de motifs extraits dans la base de données. Donc on peut expliquer la décélération du temps de calcul dans le test 1 pour les grands templates par le fait qu'il y a moins de templates à comparer même si ceux-ci contiennent plus de pixels. C'est pourquoi dans le test 2, on fixe le nombre de motifs dans la base de données, de façon artificielle, au nombre de motifs qu'il y a dans la base de données avec le plus grand template utilisé (*ie* avec le template de 31×31, on a 15 600 motifs) même lorsque l'on simule avec de plus petits templates. On voit clairement que le temps de simulation augmente linéairement avec le nombre de pixels dans le template (Figure 30 à droite).

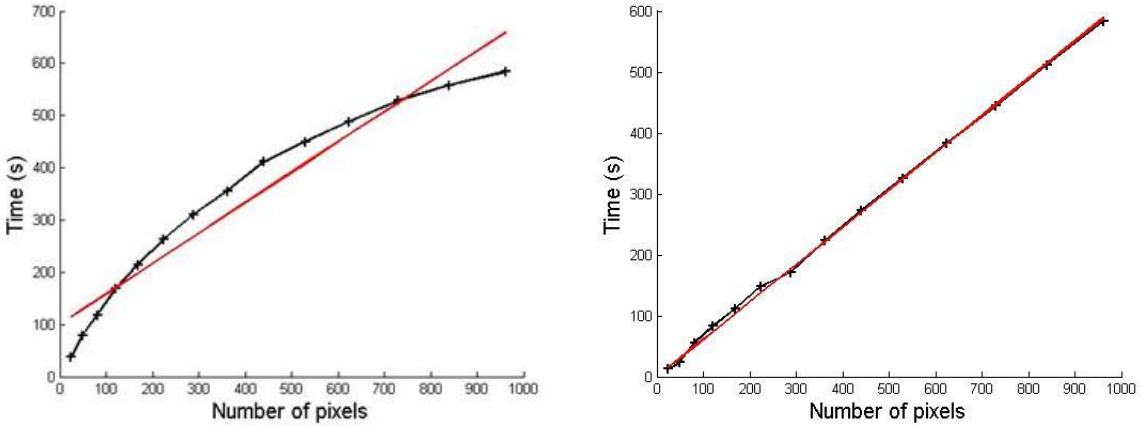


Figure 30 Complexité de l'algorithme. On trace le temps de simulation en fonction du nombre de pixels dans le template. Dans le test à gauche le nombre de motifs dans la base de données décroit. Dans le test à droite on fixe artificiellement le nombre de motifs dans la base de données. En noir les résultats numériques, en rouge la droite de régression.

La taille des patchs collés est aussi un paramètre qui a son importance. Les cas les plus extrêmes sont lorsque le patch est réduit à un pixel ou alors au contraire lorsque le patch fait la même taille que le template.
Quand le patch est trop petit, le temps de simulation est très long. Quand le patch est trop gros, l'image créée devient une copie conforme de l'image d'entrainement, on perd la notion de procédé aléatoire : la construction de l'image revient à assembler les pièces d'un puzzle et l'apparition de nouveaux motifs devient improbable. Encore une fois, la taille du patch est un compromis.

La Figure 31 montre l'influence de la taille du patch, lorsqu'on garde la taille du template fixe (ici 19×19) sur la réalisation. Lorsque l'on augmente la taille du patch de 1×1 à 5×5, la qualité du rendu final de l'image est améliorée et préservée. Dans un même temps le temps de calcul est divisé par 3 (Table 2). Les temps de calcul relatifs en fonction de la taille du patch sont reportés dans la Table 2. Pour une taille de patch de 19×19 (Figure 31 e), la réalisation ressemble à un arrangement de morceaux provenant de l'image d'entrainement sans continuité aux interfaces.

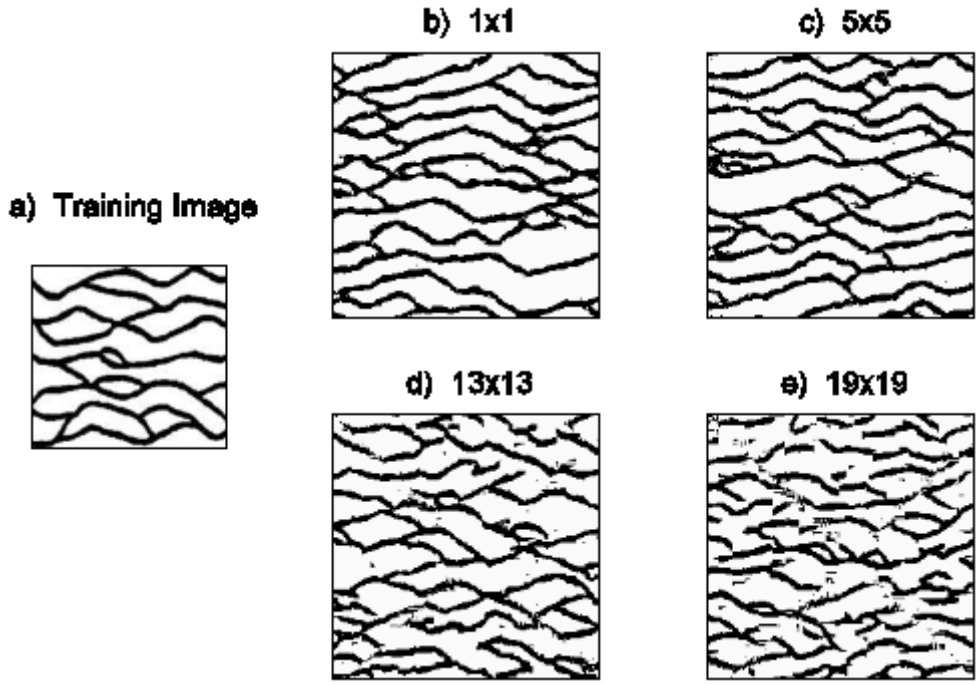


Figure 31 Influence de la taille du patch. A gauche : image d'entrainement (a). Autres images : réalisations simulées avec une taille de patch grandissante de b) 1×1, à e) 19×19. Taille du template fixe : 19×19. Chemin de visite régulier.

Table 2 Impact de la taille du patch sur le temps de calcul. Le temps de référence est celui de la simulation avec un template de taille 19×19 et un patch de taille 1×1

Taille du patch	1×1	5×5	7×7	11×11	13×13	19×19
CPU-time	T_1 =765sec	$T_1/3$	$T_1/4$	$T_1/6$	$T_1/7$	$T_1/10$

4.3.2. Comment gérer le voisinage?

Dans l'algorithme introduit dans la section 4.2, il faut comparer le template aux motifs extraits de l'image d'entrainement pour trouver le plus proche. Cette comparaison est établie sur une mesure de distance.
Plusieurs méthodes sont utilisées pour estimer cette distance, certaines étant plus appropriée que d'autres. Pour simplifier, on considère un template de 5×5 pixels et on calcule la distance entre un voisinage donné V et un motif donné T, qui contiennent tous les deux 5×5 pixels (Figure 32) .

Voisinage V

		V_3	V_4	
			V_9	V_{10}
		V_{18}	V_{19}	V_{20}
			V_{24}	V_{25}

Motif T

T_1	T_2	T_3	T_4	T_5
T_6	T_7	T_8	T_9	T_{10}
T_{11}	T_{12}	T_{13}	T_{14}	T_{15}
T_{16}	T_{17}	T_{18}	T_{19}	T_{20}
T_{21}	T_{22}	T_{23}	T_{24}	T_{25}

Figure 32 Voisinage V et motif extrait T, utilisant un template 5×5 pixels

Tous les pixels de T ont une valeur extraite de l'image d'entrainement. Ceux de V (sur la grille simulée) peuvent ne pas contenir de valeur, s'ils n'ont pas été déjà simulés ou s'ils ne contiennent pas de données.

La distance entre V et T est donnée par la somme des différences pondérées au carré entre chaque pixel contenant une information de V et ceux de T :

$$d = \sum_{i \in N} \omega_i (v_i - t_i)^2 \qquad \textbf{Equation 19}$$

avec n le nombre de pixels dans le template, ω_i un poids, v_i la valeur du pixel i dans le voisinage V, et t_i la valeur du pixel i dans le motif T. Pour calculer les poids, on commence par leur attribuer la valeur 0 si les pixels de V associés sont vides et 1 sinon (Figure 33 a). Puis on calcule un noyau Gaussien (Figure 33 b), et enfin on multiplie les deux jeux de poids et on les normalise (Figure 33 c). Dans ce cas-ci, les pixels les plus au centre du voisinage ont plus de poids que ceux qui sont aux bords.

a) Poids uniformes

0	0	1	1	0
0	0	0	1	1
0	0	0	0	0
0	0	1	1	1
0	0	0	1	1

b) Noyau Gaussien

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

c) Poids Gaussiens

$\frac{1}{82}$

0	0	7	4	0
0	0	0	16	4
0	0	0	0	0
0	0	26	16	4
0	0	0	4	1

Figure 33 Calcul des poids du voisinage V de la Figure 6. a) Poids Uniformes. b) Noyau Gaussien de variance 1. c) Poids Gaussiens calculés à partir du noyau Gaussien

Un exemple de l'impact des poids sur le rendu de l'image finale est présenté en Figure 34. La première réalisation (Figure 34 b) est générée à l'aide de poids uniformes alors que la deuxième (Figure 34 c) est générée avec les poids Gaussiens. On peut voir qu'attribuer des poids plus forts aux pixels du centre du voisinage améliore la qualité de l'image finale.

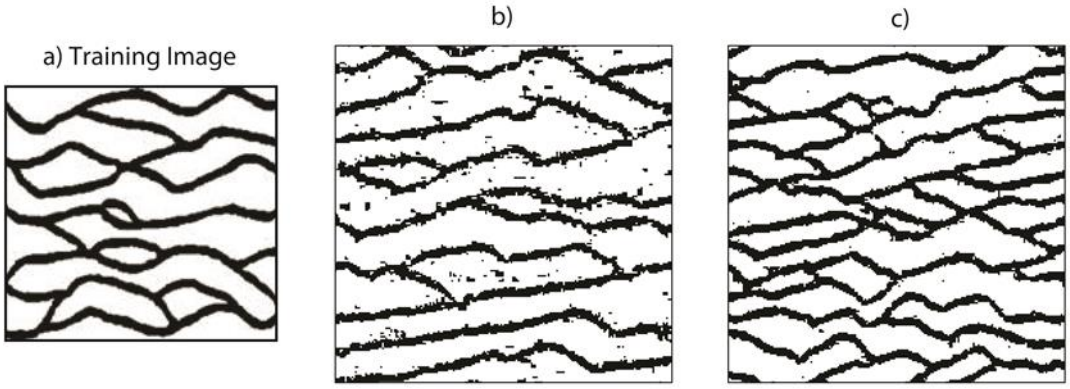


Figure 34 Impact des poids sur le rendu de l'image finale. a) L'image d'entrainement. b) La réalisation générée à l'aide des poids uniformes. c) La réalisation générée à l'aide des poids Gaussiens. Le template utilisé dans les deux réalisations est de 23×23 pixels et la taille du patch est de 5×5. L'ordre de visite est régulier.

4.3.3. Quel est l'impact du chemin de visite ?

Le procédé de simulation décrit ici est basé sur la construction successive de couronnes, ces couronnes étant définies comme un ensemble de pixels avec au moins un pixel voisin non vide. Les pixels d'une couronne sont simulés avant de déterminer la couronne suivante et ce jusqu'à ce que tous les pixels de l'image finale aient été visités une fois. Par ailleurs, le chemin de visite habituel en simulation géostatistique multipoints est aléatoire. Dans ce paragraphe, nous testons l'impact de l'ordre de visite lors de la simulation.

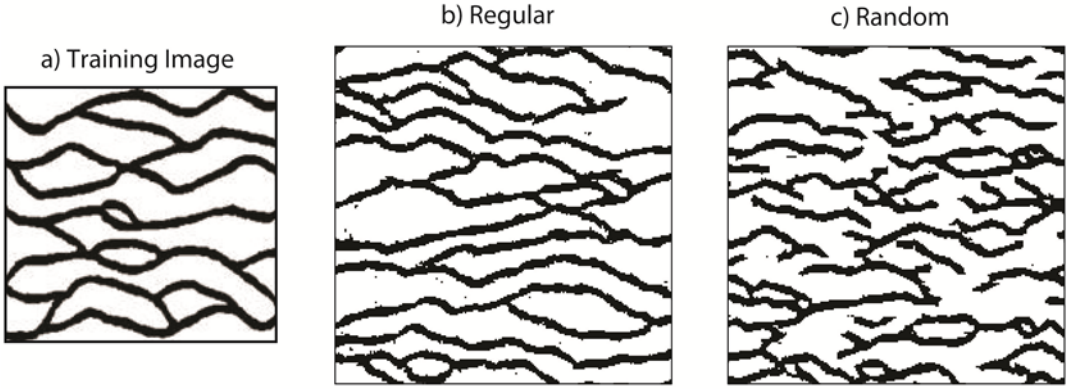


Figure 35 Impact de l'ordre de visite. a) L'image d'entrainement. b) Réalisation générée par un chemin régulier. c) Réalisation générée par un chemin aléatoire. Template : 19×19. Taille patch : 5×5.

Les deux réalisations sont montrées Figure 35, la première est obtenue avec le procédé de couronnes successives (Figure 35 b), la deuxième en visitant aléatoirement les pixels lors de la simulation (Figure 35 c). Dans les deux cas on a utilisé un template de 19×19 pixels. De façon très claire, le procédé utilisant un chemin régulier permet d'obtenir de meilleurs résultats. Il préserve la connectivité latérale des chenaux, alors que pour un chemin aléatoire les chenaux sont déconnectés. Une possibilité pour améliorer les résultats dans le cas d'un chemin aléatoire est d'augmenter la taille du template, tout en sachant que cela va augmenter les temps de calcul et la place mémoire. Dans la Figure 36 on se concentre sur le procédé avec chemin aléatoire et on augmente progressivement la taille du template. Comme on peut le constater le rendu de l'image finale s'améliore mais pas jusqu'à atteindre celui obtenu par un procédé avec chemin régulier.

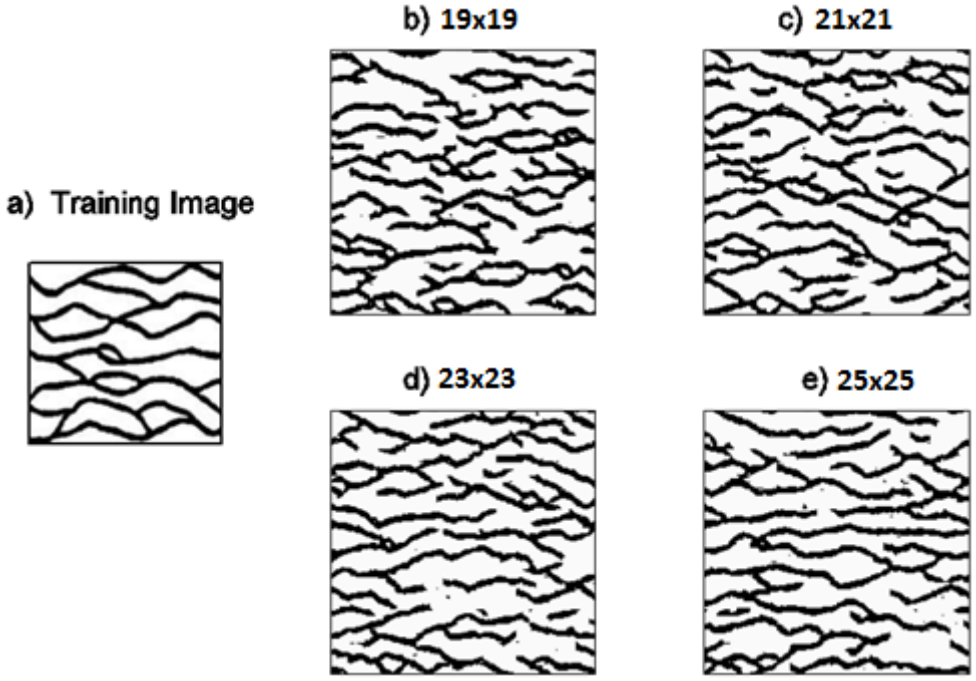


Figure 36 Influence de la taille du template sur un procédé avec chemin aléatoire. a) Image d'entrainement. b) Template 19×19. c) Template : 21×21. d) Template : 23×23. e) Template 25×25. Taille patch : 5×5. Ordre de visite : aléatoire.

L'utilisation du chemin de visite aléatoire est généralement recommandée pour permettre l'intégration des données dures. Cependant nous avons montré dans le paragraphe 4.2.2 la simulation conditionnelle reste possible pour un procédé avec couronne.

4.4. Techniques de diminution du temps de calcul

Dans cet algorithme, les temps de calcul sont très significatifs. En effet, il faut scanner toute l'image d'entrainement pour attribuer une valeur à un seul patch. L'idée générale pour diminuer le temps de calcul est de réduire le nombre de voisinages à scanner pour attribuer une valeur à un pixel. Deux solutions assez intuitives ont été testées ici. La première consiste à ne scanner qu'une partie de l'image au lieu de l'image entière (Mariethoz *et al.*, 2010). La deuxième est de classer les motifs extraits de l'image d'entrainement, soit à l'aide d'un arbre (Strebelle, 2000 ; Wei et Levoy, 2000) soit en classes (Zhang *et al.*, 2006 ; Honarkhah et Caers, 2010).

4.4.1. K-means clustering

Pour réduire le nombre de voisinages à scanner, nous suggérons de définir des classes à partir des motifs extraits de l'image d'entrainement, par exemple en utilisant l'algorithme k-means (Duda *et al.*, 2001). L'algorithme K-means est un algorithme itératif utilisé pour partitionner des objets en k classes fixées *a priori*. En quelques mots, l'algorithme sélectionne aléatoirement k points, un pour chaque groupe. Ces points sont considérés comme les barycentres initiaux des clusters. Par la suite, chaque objet est affecté au barycentre dont il est le plus proche. Cela implique la définition d'une distance. Enfin, les barycentres sont mis à jour en fonction des objets assignés à leur classe. Le processus d'attribution des objets aux barycentres et de modification des coordonnées des barycentres est répété jusqu'à convergence. Dans le cas général, les barycentres sont définis à partir de la moyenne des objets assignés à leur classe.

L'algorithme de simulation multipoints FILTERSIM (Wu *et al.*, 2008) fait également référence à l'algorithme k-means. Il se déroule comme suit. Tout d'abord, les filtres sont appliqués à l'image d'entrainement lorsqu'elle représente une variable continue ou aux cartes d'indicateurs correspondants lorsqu'elle représente une variable discrète (les cartes d'indicateurs indiquent l'absence/présence d'un faciès donné à n'importe quel endroit). Il en résulte un ensemble de scores, qui sont ensuite classés par k-means. Contrairement à Zhang *et al.* (2006) ou Wu *et al.* (2008), nous n'effectuons aucun filtrage, et la classification des motifs extraits de l'image d'entrainement n'est calculée qu'une fois. Deuxièmement, dans le cas de variables discrètes, nous n'utilisons pas de cartes d'indicateurs : les modèles de référence sont extraits de l'image d'entrainement tels quels. Enfin, pour une classe donnée, nous ne considérons pas comme représentant de la classe, le motif résultant de la moyenne point à point des motifs de la classe, mais le motif le plus proche du barycentre. Une autre différence est soulignée ci-dessous.

Nous regroupons les motifs extraits de l'image d'entrainement en k classes, chacune d'elles étant représentée par le motif le plus proche de la moyenne point à point des motifs dans la classe. Au cours du processus de simulation, chaque fois que nous affectons une valeur à un pixel donné, il faut examiner la base de données des motifs pour identifier le plus proche du voisinage du pixel cible. Si la base de données comprend 100.000 modèles, 100.000 comparaisons de distance sont nécessaires pour simuler une valeur unique, ce qui est très demandeur en temps de calcul. Lorsqu'au préalable la base de données a été classée en k classes, il ne faut que k comparaisons de distance pour trouver la classe la plus proche. Ensuite, la recherche du meilleur motif est limitée à cette classe, conduisant ainsi à une accélération significative du temps de calcul.

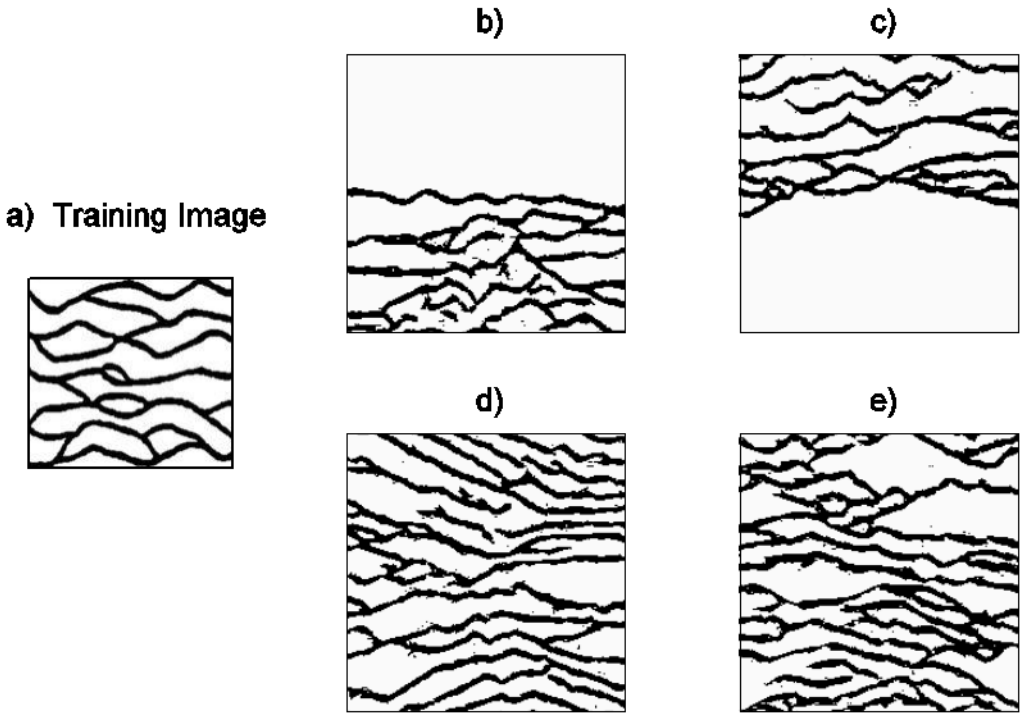


Figure 37 Simulations avec k-means clustering. a) Image d'entrainement. b) Classification avec 5 classes. c) Avec 10 classes. d) Avec 20 classes. e) Avec 30 classes. Taille du template 19×19. Taille du patch : 5×5. Chemin de visite régulier

Table 3 Temps de calcul selon le nombre de classes. Le temps de référence T_3 est donné par la réalisation générée avec 1 classe (*ie.* sans classification). Taille de template 19×19. Taille de patch : 5×5. Chemin de visite régulier.

Nombre de classes	1	5	10	20	30
Temps de calcul pour la classification et pour la simulation	0 – T_3 (255s)	0.08 T_3 - 0.86 T_3	0.18 T_3 - 0.75 T_3	0.33 T_3 - 0.33 T_3	0.66 T_3 - 0.20 T_3

Une première série de tests avec un nombre croissant de classes est représentée sur la Figure 37. Les temps relatifs de calcul sont rapportés dans le Table 3. Comme on s'y attendait, la classification contribue à rendre le processus de simulation plus rapide, même si elle nécessite une phase de classification préliminaire. Cela est d'autant plus avantageux qu'il n'est pas nécessaire de répéter cette phase préliminaire lors de la génération de plusieurs réalisations. La Figure 37 montre aussi une caractéristique particulière. Les réalisations obtenues avec un petit nombre de classes (5 et 10) montrent des chenaux, mais seulement dans une partie de la grille de simulation. La qualité des réalisations s'améliore avec le nombre de classes. Une explication possible est que le tri des motifs est moins discriminant quand il y a quelques classes seulement. Ainsi, des motifs similaires peuvent être répartis dans des classes distinctes. Cependant, même si les résultats sont meilleurs quand il y a plus de classes, la continuité latérale de chenaux n'est pas parfaitement reproduite.

En nous référant aux travaux de Ashikhmin (2001) nous introduisons le concept de cohérence. Le principe de base est très simple. Les valeurs affectées à des pixels voisins dans la réalisation ont tendance à être extraites de la même région dans l'image de la formation. Par conséquent, Ashikhmin (2001) a réduit la recherche de la meilleure configuration à une liste très courte contenant seulement les motifs associés aux voisins. De même, nous suggérons l'extension de la recherche du meilleur motif aux classes desquelles les voisins ont été extraits. En fait, nous pouvons restreindre notre attention sur les classes correspondant aux voisins les plus proches. Dans ces conditions, le choix du meilleur motif implique la recherche d'un maximum de quatre classes. Ceci dégrade légèrement les performances de l'algorithme, mais le temps de calcul reste raisonnable. Pour en revenir à l'exemple décrit ci-dessus avec 30 classes, le temps de calcul nécessaire pour simuler une réalisation augmente de 0,20 T_3 à 0,37 T_3. Ceci contribue cependant à améliorer de manière significative les caractéristiques des réalisations générées (Figure 38) : la continuité latérale et la connectivité des chenaux obtenus sont beaucoup plus cohérentes avec ceux de l'image d'entrainement.

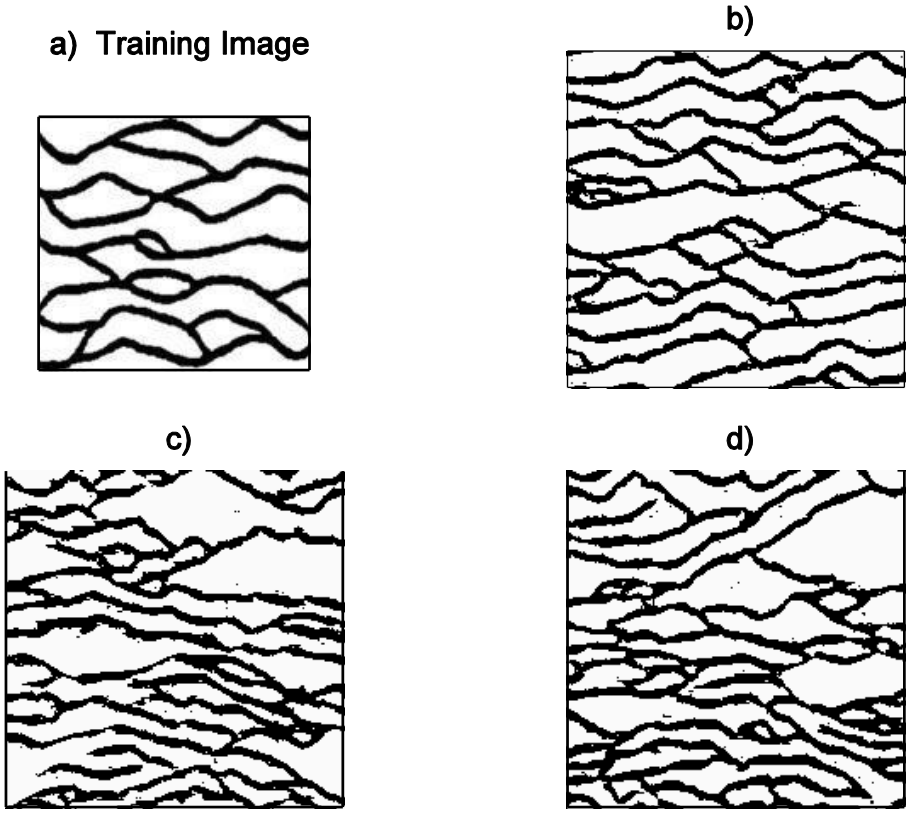


Figure 38 Impact de la cohérence et de la classification. a) Image d'entrainement. b) Simulation sans classification. C) Classification sans cohérence : 30 classes. D) Classification et cohérence : 30 classes. Taille du template : 19×19. Taille du patch : 5×5. Chemin de visite régulier.

4.4.2. Scan partiel de l'image d'entrainement

L'autre possibilité proposée pour réduire le temps de calcul consiste à ne balayer qu'une partie de l'image d'entrainement lors de la recherche du meilleur motif (Mariethoz *et al.*, 2010). Dans ce cas, il n'est pas nécessaire de créer une base de données pour stocker les motifs provenant de l'image d'apprentissage. Chaque fois qu'un patch doit être simulé, l'image d'entrainement est totalement ou partiellement scannée. Le résultat est obtenu d'autant plus rapidement que le sous-domaine de l'image d'entrainement parcouru est petit.

Table 4 Temps de calcul selon la fraction scannée de l'image d'entrainement. Le temps de référence T_0 correspond au temps de simulation lorsque toute l'image d'entrainement est scannée. Taille de template : 19×19. Taille du patch : 5×5. Chemin de visite régulier

Fraction	100%	80%	66%	50%
Temps de calcul	T_3	0.60 T_3	0.34 T_3	0.10 T_3

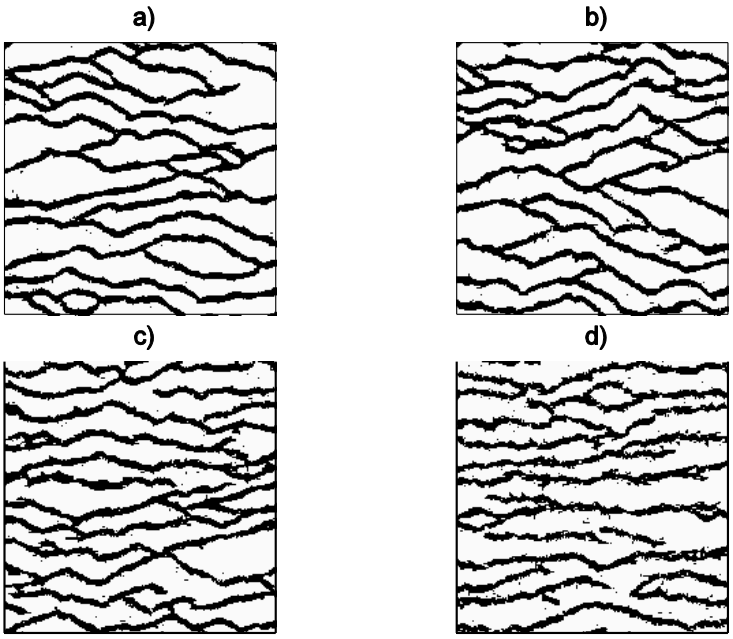


Figure 39 Réalisations simulées en ne scannant que : a) 100%, b) 80%, c) 66%, d) 50 %, de l'image d'entrainement. Taille du template : 19×19. Taille du patch : 5×5. Chemin de visite régulier.

La Figure 39 montre l'impact de la réduction de la zone balayée de l'image d'apprentissage. La continuité latérale et la connectivité des chenaux sont mieux restituées lorsque l'on parcourt une partie importante de l'image d'entrainement. Dans l'exemple ci-dessus, nous recommandons d'explorer au moins 66% de l'image d'entrainement. Les chenaux simulés sont alors relativement cohérents avec ceux figurant sur l'image d'entrainement. En outre, le temps de calcul est réduit à 34% de celui obtenu lors du balayage complet de l'image d'entrainement (Table 4).

En conclusion, pour l'exemple des chenaux étudié jusqu'à ce point, les deux stratégies décrites ci-dessus pour accélérer le processus de détermination du meilleur motif sont plus ou moins équivalentes en termes de temps de calcul et en termes de résultats. En effet, on obtient quasiment la même chose lorsque la simulation est réalisée avec 30 classes et prend en compte la cohérence, et lorsque 66% de l'image d'entrainement est parcourue. Dans ces deux cas, le temps de calcul est réduit à environ un tiers du temps de référence. Nous aurions pu essayer aussi de profiter des deux stratégies en les combinant.

4.5. Mesure de la distance entre le template et les motifs

Le temps de calcul nécessaire pour comparer le voisinage du pixel actuellement visité avec les motifs stockés dans la base de données est un point clé de l'algorithme. Il est répété autant de fois qu'il y a de motifs et chaque comparaison prend d'autant plus de temps que le template est grand. La différence est quantifiée à partir d'une distance telle que présentée dans la section 4.3.2.

Nous étudions maintenant le potentiel de différentes mesures. Premièrement, nous calculons les différences point-à-point en utilisant soit le carré de la norme L_2 soit la norme L_1. Deuxièmement, nous proposons l'application de la transformée en ondelettes dans une étape préliminaire avant d'estimer les différences point-à-point des coefficients de la transformée.

4.5.1. Mesure de distance définie par les normes L1 ou L2

Rappelons quelques notations. V est le voisinage du pixel actuellement visité et v_i la valeur du pixel i dans V. T est un motif extrait de l'image d'apprentissage et t_i est la valeur du pixel i dans T. Lorsqu'on se réfère à la norme L_2, la distance d entre V et T est donnée par l'EQUATION 20. Le carré de la différence pixel par pixel est pondéré par ω. Ce poids est plus grand lorsque le pixel d'intérêt est plus proche du pixel central du template (voir paragraphe 4.3.2). Il est égal à zéro lorsque le pixel ne contient pas de valeur (Figure 32 et Figure 33).

Pour calculer la distance d à partir de la norme L_2 :

$$d = \sum_{i \in V} \omega_i (v_i - t_i)^2 \qquad \textbf{Equation 20}$$

Pour calculer la distance d à partir de la norme L_1 :

$$d = \sum_{i \in V} \omega_i |v_i - t_i| \qquad \textbf{Equation 21}$$

Un exemple de réalisations simulées avec ces distances est montré sur la Figure 41. Les résultats sont très similaires quelle que soit la norme (L_1 ou L_2) utilisée. Les réalisations ont la même apparence et les temps de calcul sont équivalents.

4.5.2. Transformée en ondelettes

Pour réduire le temps nécessaire pour calculer une distance entre un template et un motif, nous avons testé la transformée en ondelettes. Une étape préliminaire consiste à faire une décomposition en ondelettes discrètes de premier niveau sur les motifs extraits de l'image d'apprentissage. Nous utilisons des ondelettes, car elles fournissent des approximations sur les dessins par sous-échantillonnage et ont la capacité de détecter les bords. Selon l'application, différentes ondelettes peuvent être choisies. Nous avons sélectionné les ondelettes de Haar parce qu'elles sont simples, efficaces en temps de calcul et localisées dans l'espace. Dans ce cas, le processus de décomposition est équivalent à la convolution des motifs avec des noyaux simples, comprenant les valeurs 1 et -1 (Figure 40).

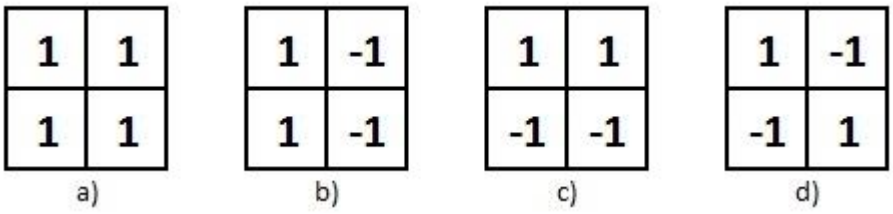


Figure 40 Noyaux de Haar. a) Basses fréquences. b) Hautes fréquences verticales. c) Hautes fréquences horizontales. d) Hautes fréquences horizontales et verticales.

La décomposition en ondelettes des motifs donne un ensemble de coefficients dont nous ne gardons que les principaux, qui sont $c = \{c_i\}_{i = 1 \ldots n}$, n étant inférieur au nombre de pixels dans le template. La réduction du nombre de valeurs à comparer (juste les coefficients) contribue à réduire de manière significative la place mémoire.

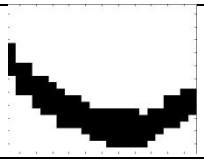
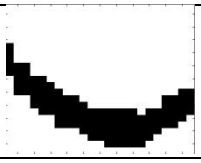
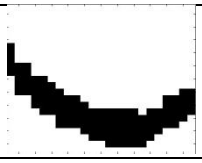
La première étape consiste à faire la décomposition en ondelettes de premier niveau de tous les motifs extraits de l'image d'entrainement. On a donc une bibliothèque de coefficients, chaque vecteur représentant un motif. Puis, de façon similaire à l'approche expliquée dans la section 4.4.1, ces vecteurs de coefficients sont classés en 20 catégories à l'aide de l'algorithme k-means.
Pendant la simulation, on visite les pixels un par un. Lors d'une itération donnée, nous déterminons le voisinage *V* du pixel courant et réalisons sa décomposition en ondelettes de premier niveau. Nous estimons alors la distance entre cette décomposition du voisinage (*i.e* les coefficients *c* dans *V*) et les 20 centres de gravité identifiés lors de la classification des décompositions des motifs extraits de l'image de la formation. Cela implique la définition de la distance suivante :

$$d = \sum_{i=1,n} \left(c_i^V - c_i^P \right)^2 \qquad \textbf{Equation 22}$$

Les coefficients *c* sont fournis grâce à une décomposition en ondelettes. Les exposants *V* et *P* indiquent si les coefficients correspondent au voisinage de *V* ou à la bibliothèque de coefficients de décomposition des motifs extraits *P*. À ce stade, une difficulté est de tenir compte du fait que *V* n'est pas entièrement défini : il peut inclure des pixels vides. Comme le montre la Figure 40, la décomposition en ondelettes de premier niveau implique que les coefficients c sont calculés à partir de 4 pixels. Par conséquent, nous ne tenons compte que des coefficients de décomposition calculés à partir d'au moins 3 pixels non vides. Si le calcul contient moins de 3 pixels alors le poids est mis à zéro. Il est à noter que si on ne tient compte que des coefficients calculés à partir de 4 pixels on enlève trop d'information et on dégrade alors l'image finale.

Afin d'illustrer le potentiel des ondelettes, on effectue un simple test. On sélectionne de façon aléatoire un motif dans l'image d'entrainement et on voit à quelle vitesse on peut retrouver ce motif en scannant toute l'image d'entrainement selon la mesure utilisée. Dans un premier test, on utilise la distance définie par le carré de la norme L_2 pondérée (Equation 20). Dans un deuxième test, on utilise la distance définie par la norme L_1 pondérée (Equation 21). Enfin dans un dernier test, on utilise la décomposition en ondelettes (Equation 22). Dans tous les tests on retrouve très bien le motif choisi aléatoirement. Les temps sont regroupés dans la Table 5. On voit qu'en utilisant la distance définie à partir de la transformée en ondelettes, le temps de calcul est 6 fois plus petit. La réduction de ce temps est un point clé, car l'étape de comparaison entre un voisinage et la bibliothèque de motifs est répétée des milliers de fois lors de la simulation.

Table 5 Temps de calcul pour trouver un motif dans l'image d'entrainement en fonction de la mesure de la distance. Taille de la fenêtre de voisinage: 19 x 19 pixels ; taille de l'image d'entrainement: 150 × 160.

Mesure	L2	L1	Ondelettes
Temps	0.046s	0.046s	0.007s
Motif sélectionné			

Sur la Figure 41, on compare 3 réalisations obtenues en utilisant les mesures décrites ci-dessus. La réalisation b) est simulée en utilisant le carré de la norme L_2 pondérée, la réalisation c) la norme L_1 pondérée et en enfin la réalisation d) en utilisant la transformée en ondelettes. Il n'y a pas beaucoup de différences entre les réalisations b et c, les chenaux sont bien reproduits ainsi que leur espacement. Sur la réalisation d) on voit qu'il y a plus de pixels noirs isolés et que les chenaux bien que correctement reproduits semblent plus rapprochés que dans l'image d'entrainement.

Concernant les temps de calcul pour la simulation avec transformée en ondelettes on est à 0.15 T_0 où T_0 est le temps de calcul de référence pour la simulation sans technique de clustering ou scanning partiel de l'image (Table 3, 1 classe). Un temps de calcul supplémentaire est nécessaire pour effectuer la transformée en ondelettes de la bibliothèque de motifs, qui est de 0.16 T_0.

En d'autres termes, si l'on utilise la norme L_2 sur les coefficients de la transformée en ondelettes comme mesure de distance et que l'on procède à une classification en 20 classes des motifs, on peut réduire le temps de calcul par deux par rapport au clustering 20 classes seul.

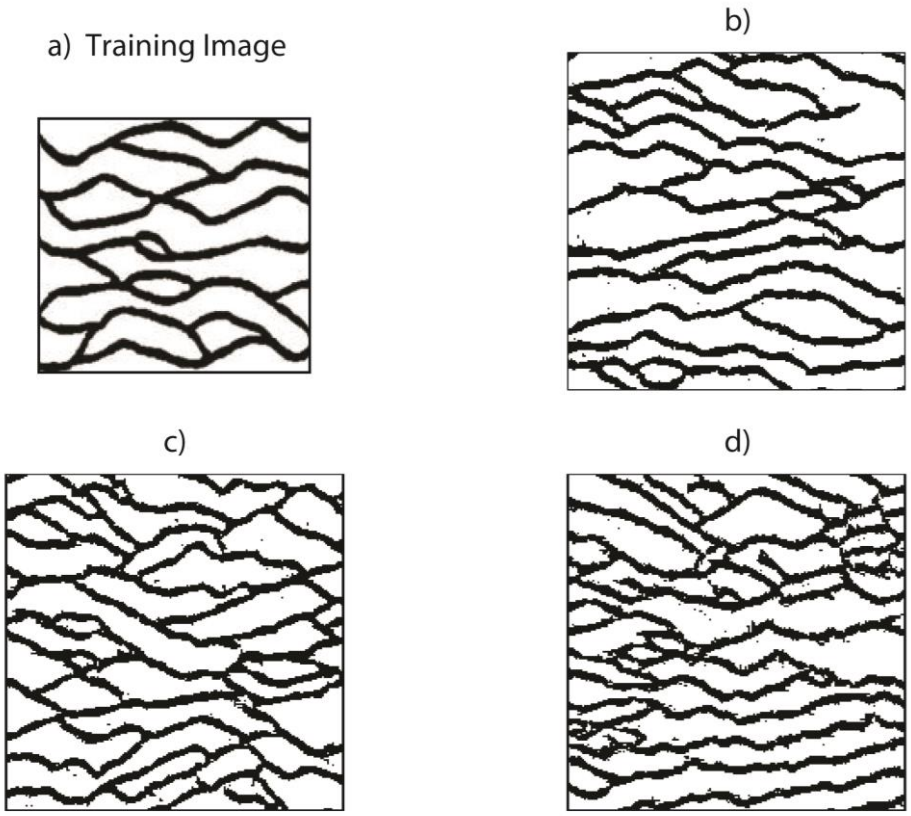


Figure 41 Réalisations simulées à l'aide des différentes définitions de distance. a): l'image d'entrainement ; b) la réalisation obtenue avec la norme L_2; c) la réalisation obtenue avec la norme L_1; d) La réalisation obtenue avec l'approche basée sur les ondelettes.

4.6. Quelques Résultats

Quelques essais sont présentés ci-après qui soulignent le potentiel et les limites de l'algorithme décrit dans le présent chapitre. Nous ne considérons que la version la plus simple de l'algorithme avec un scan complet de l'image d'entrainement et aucune classification préliminaire des motifs.

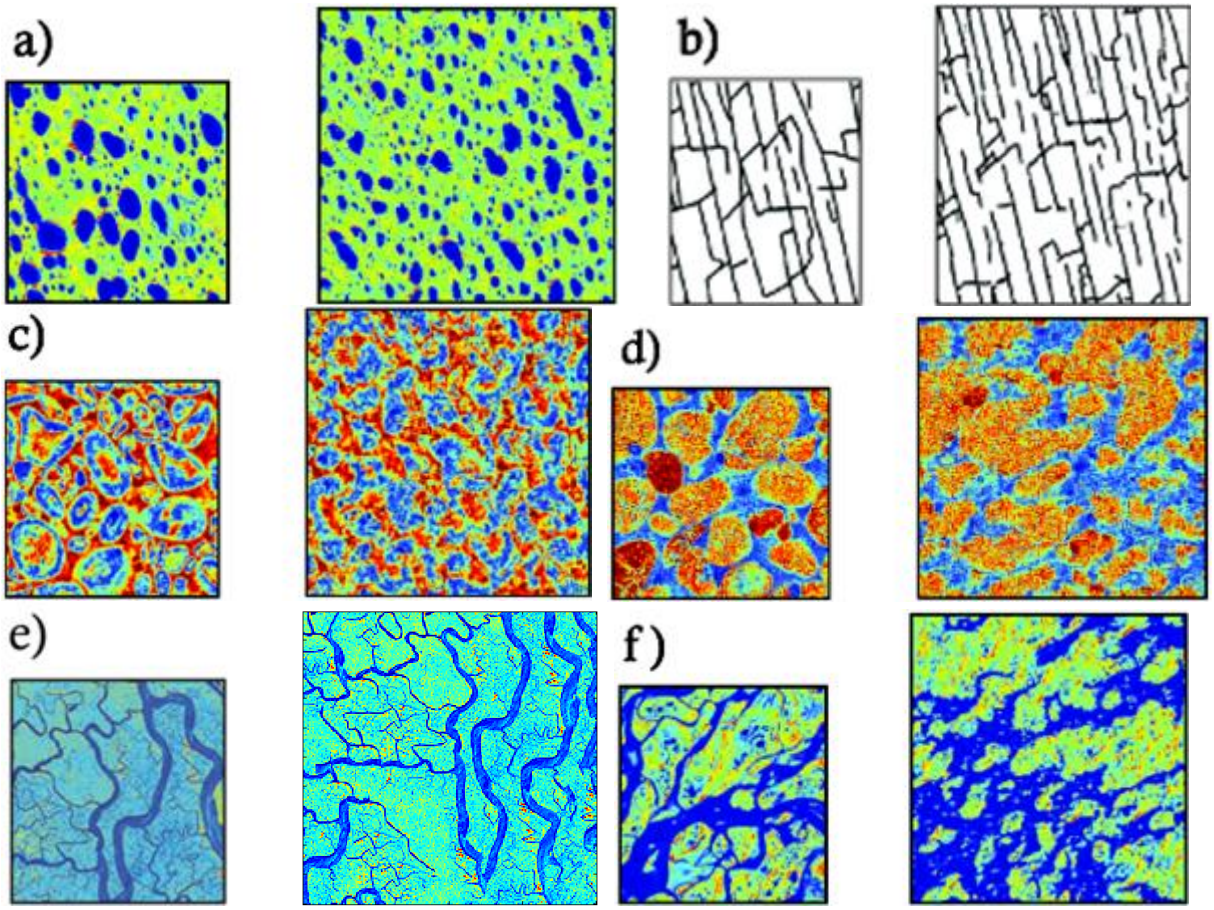


Figure 42 Les résultats obtenus par notre algorithme. Les petites images sont des images d'entraînement (150 × 160 pixels). Les plus grands sont les réalisations simulées (200 × 200 pixels).

Six exemples sont présentés dans la Figure 42. Certains d'entre eux représentent des objets à l'échelle microscopique comme les pores et grains (a, c et d) et des objets à l'échelles macroscopique comme des fractures ou des chenaux (b, e et f). Nous vérifions que le processus de simulation fonctionne bien pour les cas continus et discrets. Les réalisations générées sont semblables aux images d'apprentissage. Cependant, les résultats commencent à se dégrader lorsque les images d'entraînement comprennent de très grands objets tels que des gros grains ou des grandes chenaux. Une possibilité pour améliorer les résultats serait d'augmenter la taille du template. Cependant, il en résulterait une augmentation du temps de calcul et de la charge de mémoire.

4.7. Conclusion

Dans ce chapitre, nous avons développé un algorithme qui combine les deux concepts de simulation multipoints introduit en géosciences et de synthèse de texture utilisé dans l'infographie. En plus de sa simplicité et de son efficacité, l'algorithme proposé permet de simuler des objets géologiques avec des formes géométriques complexes. Le processus de simulation consiste à coller sur la grille de simulation un patch extrait d'une image d'apprentissage, selon un chemin en couronne jusqu'à ce que la grille soit entièrement simulée.

Des tests de sensibilité ont été effectués pour saisir les influences de 1) la taille de la fenêtre et du patch, de 2) la définition des poids, et de 3) le chemin de propagation lors de la visite des pixels. Nous avons montré que l'utilisation de grands templates permet de reproduire de grands objets. Cependant, cela induit une augmentation du temps de calcul et de la charge de mémoire. Nous avons également observé que l'application d'un noyau gaussien met l'accent sur l'importance des pixels centraux de la fenêtre de contexte et contribue à rendre les objets simulés plus semblables à ceux de l'image d'apprentissage. Enfin, nous avons montré que le respect d'un chemin de visite régulier au lieu d'un chemin aléatoire améliore la continuité et la connectivité de gros objets.

Nous avons montré que l'étape d'initialisation de l'algorithme pouvait être modifiée pour tenir compte des données dures tout en utilisant un chemin régulier de simulation par couronne. Le principe de base implique la construction d'une forme géométrique fermée reliant les données deux à deux. Par exemples, pour trois données on obtient un triangle, pour quatre données un carré, et pour cinq données et plus des polygones de forme un peu plus complexe. Les pixels traversés par les côtés de ces polygones sont simulés en premier, puis on crée des couronnes autour de ces formes en utilisant l'algorithme décrit ci-dessus.

Nous avons également étudié deux techniques de gestion de motifs extraits de l'image d'entrainement pour diminuer le temps de simulation. La première technique consiste à ne scanner qu'une partie de l'image d'entrainement lors de la simulation d'un patch. Plus on diminue la partie scannée, plus le temps de calcul diminue mais plus l'image finale est dégradée. La deuxième technique suit la même idée mais au lieu de ne scanner qu'une partie aléatoire de l'image d'entrainement on réduit intelligemment la partie scannée. On classe les motifs extraits par ressemblance et on leur attribue un représentant, *ie,* un motif proche représentatif de l'ensemble du groupe. On ne fait la recherche du meilleur motif que dans le groupe dont le représentant est le plus proche du voisinage du patch simulé. Plus le nombre de groupes est important plus le temps de calcul diminue, cependant comme pour la première technique on observe une dégradation des résultats de qualité de rendu. On introduit alors le concept de cohérence. Au lieu de ne scanner que le groupe du meilleur représentant, on scanne également les groupes à l'origine des pixels voisins. Cette approche permet d'augmenter la continuité et la connectivité des chenaux. L'image finale est alors bien plus semblable à l'image d'entrainement. En prenant les paramètres tels que l'image finale ne soit pas trop dégradée on arrive à diviser par 3 le temps de simulation.

De plus, nous avons testé différentes distances pour mesurer l'écart entre le voisinage du patch simulé et un motif extrait de l'image d'entrainement. Celle qui présente le plus d'intérêt pour la diminution du temps de calcul est la décomposition en ondelettes. La première étape consiste à appliquer une décomposition en ondelettes de premier niveau aux motifs de la base de données. Un motif est alors représenté par les coefficients de cette décomposition. Lors de la simulation, on procède à la même décomposition du voisinage, en ne gardant que les

coefficients de premier niveau et on les compare à ceux de la base de données. Comme il y a moins de coefficient à comparer, le temps de calcul est diminué. Considérant que les motifs ou les coefficients sont classés en 20 groupes, utiliser la décomposition en ondelettes permet de réduire encore le temps de calcul par 2.

Pour finir, six exemples sont testés pour montrer le potentiel et les limites de l'algorithme proposé. Nous avons vérifié que le processus de simulation fonctionne bien pour les deux cas continus et discrets, mais que la reproduction de très grands objets est toujours un défi.

Même si cette méthode montre des résultats encourageants, elle rencontre quelques difficultés à capturer de larges structures. De plus elle reste très coûteuse en temps de calcul. L'introduction d'une échelle intermédiaire devrait au moins en partie résoudre ces problèmes.