

Fondements et Algorithmes de l'Imagerie Numérique

—

Projet : détection de fissures par l'algorithme du
germe

Année 2024 – 2025

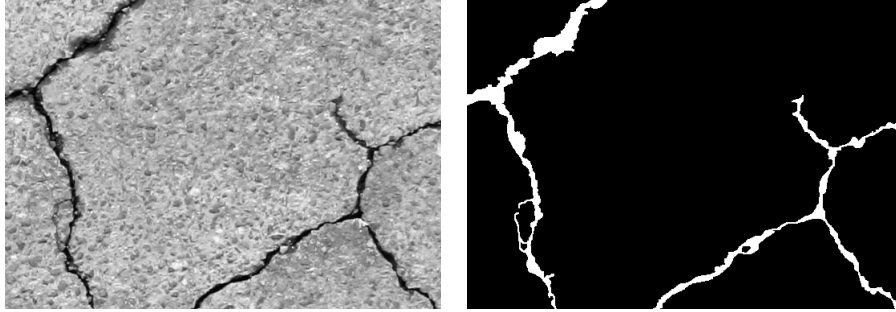


FIGURE 1 – Exemple de fissures avec la carte de détection associée. Un pixel blanc matérialise la présence d’une fissure quand un pixel noir indique son absence.

1 Objectif

On s’intéresse, dans ce projet, à la mise en œuvre d’une variante de l’algorithme du germe pour détecter des fissures dans des images en niveaux de gris (cf. figure 1). Ce sujet est librement inspiré de [1].

Dans sa forme la plus générale, l’algorithme du germe s’exprime sous la forme suivante :

Algorithme 1 : Algorithme du germe

```

Data : Un germe initial dont la position est  $p_0$ 
Result : La liste  $L$  des pixels visités
function germe( $p$ ):
    if condition( $p$ ) then
        visiter( $p$ );
        foreach 8-voisin  $v$  de  $p$  do
            | germe( $v$ );
        end
    end
end
germe( $p_0$ );

```

2 Travail à faire

Dans les sections suivantes, on présente un algorithme qui est amélioré au fil des questions. Pour chaque nouvelle tâche à faire, il faut repartir du résultat de la question précédente. L’énoncé est cependant suffisamment modulaire pour pouvoir utiliser un résultat antérieur si une question n’a pas été réussie. Seules exceptions à ce schéma de dépendance, les questions 7 et 10 ne servent de base à aucune autre question.

Deux types de productions sont attendues : des réponses directes aux ques-

tions de réflexions générales (« **Partie théorique :** ») et des implémentations en langage C (« **Partie pratique :** »).

2.1 Seuil fixe

Question 1 (*env. 1 point(s)*). En première approximation, on peut remarquer qu'une fissure est généralement plus sombre qu'un revêtement sain. Ainsi, n'ajouter que les voisins dont l'intensité est inférieure à celle du germe initial peut déjà donner des résultats probants.

Partie pratique : Implémenter l'algorithme 1 avec

$$\text{condition}(p) := i(p) \leq T$$

(où $i(x)$ désigne l'intensité du pixel x et $T = i(p_0)$) et où $\text{visiter}(p)$ ajoute p à la liste des pixels visités. À l'issue de l'algorithme, cette liste donne l'emplacement des fissures. Utiliser une pile explicite pour le parcours des pixels.

Partie théorique : Donner une limite de cet algorithme. Argumenter.

Question 2 (*env. 2 point(s)*). On définit la circularité d'un ensemble 8-connexe de points discrets comme la quantité

$$\frac{2R}{D}$$

où R est le « rayon » de l'ensemble, c'est-à-dire la distance minimale entre le bord et un de ses centres et D son « diamètre », le plus long des plus courts 8-chemins entre deux points appartenant à la région (le chemin peut éventuellement sortir de la région si cette dernière n'est pas convexe). Cette distance correspond à la distance d_∞ appliquée entre les deux extrémités du chemin¹. La figure 2 illustre ces deux notions sur un exemple.

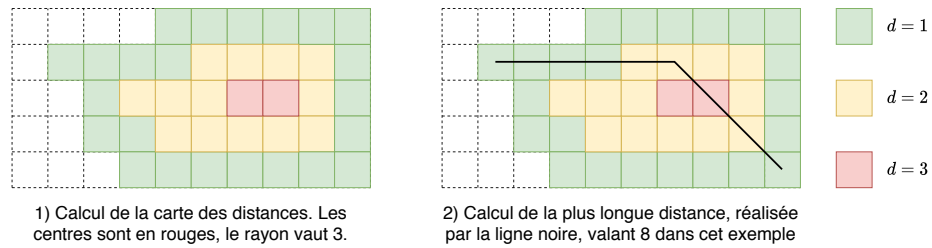


FIGURE 2 – Calculs du rayon (à gauche) et du diamètre (à droite)

Partie théorique : Exprimer la circularité d'un carré en fonction de la longueur $c \in \mathbb{N}^*$ de son côté. Idem pour le rectangle de dimension $1 \times c$. Ces deux exemples correspondent aux deux cas extrêmes (l'un est typique d'une

1. Voir https://en.wikipedia.org/wiki/Chebyshev_distance, qui contient une formule pouvant être utile.

circularité maximale, l'autre d'une circularité minimale). Faire tendre c vers $+\infty$ pour déterminer les valeurs minimale et maximale que peut prendre la circularité. On ne demande pas de détailler le calcul mais il faut donner les bornes trouvées. De quelle borne se rapproche la circularité d'une fissure ?

Partie pratique : Écrire un algorithme calculant la circularité d'une forme germée.

Question 3 (*env. 0,5 point(s)*). On considère le critère de rejet suivant :

Algorithme 2 : Critère de rejet

Data : La liste L des pixels obtenus par algorithme du germe
Seuil θ fourni par l'utilisateur
Result : Un booléen reject indiquant si la liste est rejetée ou non
 $C := \text{circularité}(L)$;
if $C > \theta$ **then**
 reject := True;
else
 reject := False;
end

Partie pratique : Après avoir implémenté le calcul de la circularité (cf. question 2), ajouter ce critère de rejet. Il doit être maintenu pour l'ensemble des questions suivantes.

2.2 Restriction spatiale

Question 4 (*env. 1 point(s)*). L'implémentation naïve de l'algorithme du germe privilégie généralement une direction. Pour les questions suivantes, il est impératif que le germe se propage à la même vitesse dans toutes les directions. La figure 3 présente une version de cet algorithme corrigeant le problème.

Partie pratique : Implémenter cette version de la propagation du germe.

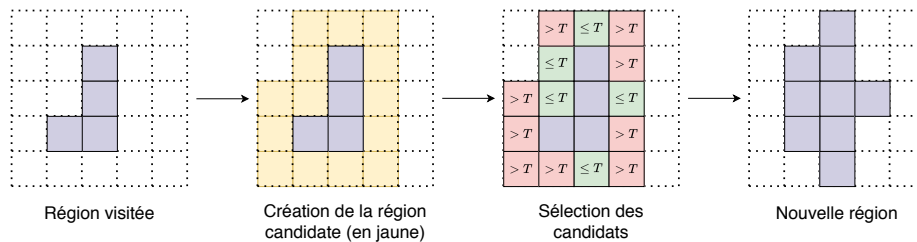


FIGURE 3 – Illustration de la propagation du germe par « couches »

Question 5 (*env. 1,5 point(s)*). Une autre possibilité pour éviter un arrêt prématuré de l'algorithme consiste à forcer la propagation à continuer tant que

la région visitée n'a pas atteint une certaine taille (en paramètre du programme). À cette fin, on modifie la condition d'acceptation des candidats par la suivante : si aucun candidat ne valide la condition initiale, on retient le candidat qui a l'intensité la plus basse (s'il y a plusieurs candidats vérifiant cette propriété, on en choisit un arbitrairement).

Partie pratique : Réaliser cette version. Pour ce faire, procéder en deux temps :

1. Une première propagation partant du germe initial avec la condition modifiée et qui se termine lorsque la taille minimale est atteinte.
2. Une seconde propagation partant de la région visitée à l'étape précédente et utilisant la condition initiale.

Question 6 (*env. 0,5 point(s)*). Quand la taille des régions augmente, le risque de rejet augmente également. Il est donc parfois préférable d'obtenir une détection partielle que de n'avoir aucune détection du tout.

Partie pratique : Faire en sorte que la propagation s'arrête dès que la taille est supérieure à un seuil donné en paramètre.

2.3 Seuil automatique

Question 7 (*env. 4,5 point(s)*). L'intensité des pixels composant une fissure n'est que rarement constante. Si le germe est situé sur un pixel trop sombre, le relevé de la fissure sera incomplet. Ce problème sera traité plus en profondeur à partir de la question 8. Un début de solution peut être de prendre une valeur de départ légèrement supérieure à l'intensité du pixel germé. Pour trouver une valeur raisonnable, on propose la méthode suivante (résumée en figure 4) :

1. Exécution de l'algorithme du germe dont le seuil initial est fixé au double de l'intensité du pixel initial. On regarde alors le résultat à la deuxième itération.
2. Sélection du point le plus éloigné du pixel de départ.
3. Détermination des pixels composant le segment de droite de Bresenham reliant ce point au point à l'origine.
4. Prolongation du segment de droite à toute l'image. La droite obtenue indique alors la direction « privilégiée » de la fissure.
5. Récupération des intensités le long de la droite.
6. Construction de l'histogramme. On fixe T au plus petit minimum local : dans une situation idéale, les fissures doivent être représentées par une vague dans l'histogramme. Si on fixe T à la valeur maximale de cette vague, on extrait tous les pixels de valeur inférieure, soit précisément la fissure.
7. Propagation d'un nouveau germe avec le seuil ainsi calculé.

Partie pratique : Réaliser ces différentes étapes. Enregistrer l'histogramme calculé à l'étape 6 dans un fichier CSV².

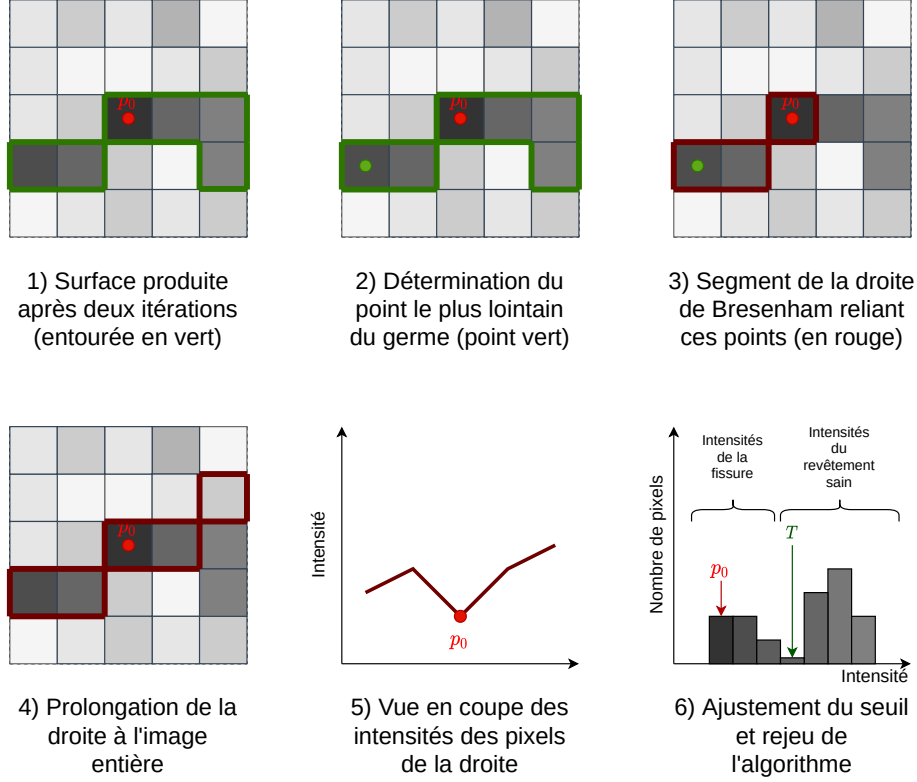


FIGURE 4 – Ajustement du seuil par histogramme

2.4 Seuil adaptatif

Question 8 (*env. 1 point(s)*). Pour prendre en compte la variabilité d'intensité des pixels composant les fissures, un seuil adaptatif est proposé :

$$\begin{cases} T_0 &= i(p_0) \\ T_{n+1} &= \max(\max_{p \in C_n}(i(p)), T_n) + w \end{cases}$$

où C_n est l'ensemble des candidats à l'étape n et $w \in \mathbb{R}_+$ est un paramètre à donner en entrée de la méthode et qui reste constant tout au long de la propagation. Noter que w n'est pas nécessairement un nombre entier.

Partie théorique : Comment évolue le seuil T_n lorsque n augmente ?

2. https://fr.wikipedia.org/wiki/Comma-separated_values

Partie pratique : Modifier le code pour prendre en compte ce seuil adaptatif.

Question 9 (*env. 2,5 point(s)*). Pour améliorer le seuil adaptatif, on peut pondérer le facteur w par la circularité de la région visitée (vue à la question 2). Les équations deviennent

$$\begin{cases} T_0 &= i(p_0) \\ T_{n+1} &= \max_{p \in C_n} (i(p)), T_n) + \mathbf{w} \times \mathbf{circularité}(\mathbf{V}_n) \end{cases}$$

où V_n est la région visitée à l'étape n .

Partie théorique : Quel est l'impact de la circularité sur l'incrément du seuil si on est au creux d'une fissure ? Et pour un revêtement sain ? En déduire, pour chacun des deux cas, la tendance qui va se dessiner pour la circularité de V_n lorsque la propagation se poursuit (c'est-à-dire lorsque n augmente). Expliquer en quoi ce phénomène est pertinent dans notre cas.

Partie pratique : Intégrer ces équations à la méthode précédente.

2.5 Germes multiples

Question 10 (*env. 0,5 point(s)*). Les méthodes vues jusqu'à présent ont un certain nombre de limites. Tout d'abord, elles ne peuvent produire qu'une seule composante connexe. Si deux fissures disjointes sont représentées dans l'image, il sera impossible de les détecter toutes les deux. Ensuite, elles demandent l'intervention d'un opérateur, ce qui les rend difficilement utilisables si le volume de données à traiter est important. Une solution est de lancer plusieurs fois l'algorithme du germe à différentes positions, selon un quadrillage.

Partie pratique : Lancer l'algorithme de la question précédente (ou, à défaut, l'algorithme de la question la plus avancée qui ait été traitée) aux points $\{(ks_x, ls_y) \mid k, l \in \mathbb{N}\}$ de l'image où $s_x, s_y \in \mathbb{N}$ sont les paramètres de décalage renseignés par l'utilisateur. Le résultat final est la réunion des détections.

2.6 Graphe des composantes connexes

On souhaite à présent travailler à plus gros grain et affiner le résultat par composantes « quasi-connexes », qui seront formellement définies à la question 12.

Question 11 (*env. 1,5 point(s)*). Pour la suite, on a besoin d'identifier les points non 8-simples.

Partie pratique : Programmer un algorithme pour détecter les points non 8-simples dans une image binaire et créer une carte de ces points. Concrètement, il faut produire une image de même dimension que l'image d'entrée et mettre les points non 8-simples en blanc et tous les autres points (points simples et pixels du fond) en noir.

Question 12 (env. 1,5 point(s)).

En retirant les points non 8-simples de la figure, on obtient un certain nombre de composantes connexes, appelées ici composantes « quasi-connexes » de la figure d'origine. On peut alors construire un graphe, simple et non orienté, défini comme suit :

- Les sommets de ce graphe sont les composantes quasi-connexes et les points non 8-simples
- Il existe une arête entre deux sommets si les deux éléments qu'ils représentent sont connexes

La figure 5 indique comment est construit ce graphe.

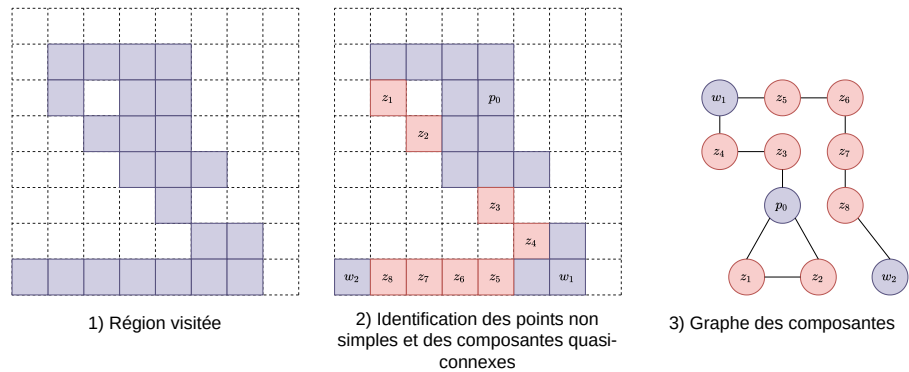


FIGURE 5 – Construction du graphe des composantes quasi-connexes

Partie pratique :

1. Définir une structure de données adaptée pour représenter un tel graphe.
Caractéristiques :
 - chaque noeud doit stocker une liste contenant les pixels qui constituent la composante
 - chaque noeud doit comporter un booléen indiquant s'il s'agit d'un point non 8-simple
2. Écrire l'algorithme permettant de construire ce graphe
3. Enregistrer ce graphe au format dot³ dans le fichier passé en paramètre.
Il n'est pas nécessaire de faire figurer la liste des pixels dans ce fichier.

Question 13 (env. 1,5 point(s)). On introduit alors une nouvelle méthode, opérant cette fois-ci à l'échelle des composantes quasi-connexes.

Partie pratique :

1. Identification des composantes quasi-connexes
2. Pour chaque composante, trouver un centre (selon la définition donnée en question 2)

3. [https://fr.wikipedia.org/wiki/DOT_\(langage\)](https://fr.wikipedia.org/wiki/DOT_(langage))

3. Propagation d'un nouveau germe en chacun de ces centres
4. Rejet éventuel de certaines régions visitées
5. Production du résultat par réunion des régions visitées

Question 14 (*hors-barème, jusqu'à 3 point(s) bonus*). Proposer une autre méthode de détection de fissures. Il est possible de repartir d'une des questions de l'énoncé ou de tenter quelque chose de totalement différent. Tout apport concret, même minime, sera valorisé.

La méthode proposée ne doit cependant pas recourir à l'apprentissage par l'exemple et la démarche mise en œuvre doit pouvoir être expliquée le jour de la soutenance. En conséquence, l'apprentissage profond et l'algorithmie évolutionnaire sont interdits.

3 Interface

Aucune interface graphique n'est demandée, on transmet les paramètres au programme via la ligne de commande et les résultats sont enregistrés dans des fichiers dédiés. Le code pour parser les arguments de la fonction main est fourni.

Bien que le sujet proposé soit incrémental, il est impératif de conserver le code de chaque question pour pouvoir l'exécuter dès que l'utilisateur le demandera via l'option `--question`. La table 1 liste les arguments pris en charge. Un exemple d'utilisation est disponible dans le fichier `main.c` présent dans l'archive.

Paramètre	Questions concernées
<code>--question <int></code>	Toutes
<code>--input_filename <char*></code>	Toutes
<code>--output_filename <char*></code>	1,3–13
<code>--x_0 <uint></code>	Toutes
<code>--y_0 <uint></code>	Toutes
<code>--reject_criterion <float></code>	3–13
<code>--min_size <uint></code>	5–13
<code>--max_size <uint></code>	6–13
<code>--w <float></code>	8–13
<code>--s_x <uint></code>	10
<code>--s_y <uint></code>	10
<code>--ns_map <char*></code>	11–13
<code>--graph <char*></code>	12–13

TABLE 1 – Arguments à renseigner au programme selon la question traitée.

4 Consignes

Le projet est à réaliser seul(e) et à déposer sur Gitlab : <http://git.unistra.fr>. Il donnera lieu à une soutenance (date sur moodle).

Pour tester votre programme, une banque d'images au format PGM (non compressé) est mise à disposition. Comme certaines images sont plus difficiles que d'autres, n'hésitez pas à faire varier vos images de test, en particulier si vous constatez des résultats décevants. Le résultat « idéal » pour chaque image est dans le dossier `images/gt`. Toutefois, ne soyez pas surpris, les méthodes vues dans ce projet produiront des résultats largement imparfaits. C'est pourquoi vous ne serez pas jugés sur ce critère.

Au niveau du code, vous devez repartir de l'archive disponible sur Moodle. Le fichier `main.c` ne doit contenir que la fonction `main()` et les réponses aux parties théoriques. Il n'est pas autorisé d'utiliser de dépendances externes mais vous pouvez reprendre vos TP de FAIN. Le jour de la soutenance, il faudra recompiler devant moi votre code avec le Makefile d'origine. S'il reste des warnings, vous devrez être en mesure d'expliquer en quoi ils n'impactent pas le projet. Dans le cas contraire (absence de justification ou impact négatif), un malus sera appliqué. Également, pensez à tester régulièrement vos programmes avec `valgrind` :

```
valgrind ./project [arguments] ou make check
```

Ce test fera partie des opérations à réaliser lors de la soutenance. Si je serais (relativement) souple sur les oublis de désallocation mémoire, je tiendrais compte dans la note des erreurs de type :

- `Invalid read/write`
- `Conditional jump or move depends on uninitialised value(s)`
- `Invalid free()`

Chacune de ces erreurs peut provoquer un segfault et indique donc qu'il y a un problème dans votre code⁴. Il s'agit essentiellement de variables que vous oubliez d'initialiser ou de dépassement de capacité dans les tableaux.

Pour toute question sur le projet, vous pouvez m'écrire à

baudrier@unistra.fr

5 Barème

Chaque question est composée d'une partie pratique et, éventuellement, d'une partie théorique. Dans ce second cas, 2/3 des points de la question sont consacrés à la partie pratique et 1/3 sont donc attribués à la théorie. Pour éviter de surcharger le sujet, seule la somme pondérée de ces deux éléments est indiquée après le numéro de question.

4. Attention, la ligne indiquée par les messages d'erreur correspond à la ligne de survenue du problème et non à l'origine de ce dernier. Par exemple, si vous passez une variable non initialisée à une fonction appartenant à une bibliothèque tierce, le message pointerait vers cette fonction alors que l'erreur est bien de votre fait.

Les parties pratiques sont évaluées selon 5 grades (A, B, C, D et E), octroyant chacun une certaine proportion des points. La description des différents grades est disponibles en table 2. Vous noterez qu’une part importante de la notation repose sur votre capacité à comprendre et à expliquer ce que vous avez fait, en particulier lorsque vous n’êtes pas parvenus à réaliser précisément ce qui était demandé.

Les parties théoriques sont notées de façon binaire (100% ou 0%). Lorsqu’elles sont subdivisées en sous-questions, la binarité porte sur chacune des sous-questions et non sur l’ensemble. Vous pouvez donc quand même avoir des points si vous ne répondez que partiellement aux sous-questions d’un même bloc.

Le barème des pénalités est disponible en table 3.

Grade	Valeur	Cas d’usage
A	100%	Le cahier des charges est intégralement rempli.
B	75%	Il existe des cas non triviaux qui sont parfaitement traités. La frontière entre cas fonctionnels et cas dysfonctionnels est connue.
C	50%	Mêmes conditions que B mais lorsque la frontière est inconnue.
D	25%	Aucun cas non trivial n’est parfaitement traité. Il faut alors remplir l’ensemble des conditions suivantes : <ul style="list-style-type: none"> — le code produit des motifs non dégénérés — l’étudiant(e) est en mesure d’expliquer le comportement du code — l’étudiant(e) peut justifier que sa méthode remplit un cahier des charges plus modeste
E	0%	Insuffisant

TABLE 2 – Barème pour les algorithmes à implémenter

Malus	Quantité
Warning non justifié ou délétère	2 points en moins sur la note finale s'il y a un ou plusieurs warnings de ce type.
Erreur valgrind	Puisque synonyme de comportements indéterminés, la présence de ces erreurs ne permet pas de considérer le cahier des charges comme pleinement rempli. De plus, la frontière entre cas fonctionnels et cas dysfonctionnels dépend de l'implémentation et n'est, a priori, pas connue. En conséquence, les parties pratiques des questions concernées par ces erreurs seront, au mieux, notées au grade C.
Code qui ne compile pas	Le grade E est attribué à toutes les parties pratiques. Les parties théoriques restent prises en compte dans l'évaluation.

TABLE 3 – Barème pour les pénalités

Références

- [1] T. Yamaguchi, S. Nakamura, and S. Hashimoto, “An efficient crack detection method using percolation-based image processing,” in *Conference on Industrial Electronics and Applications*, (Singapore, Singapore), pp. 1875–1880, IEEE, 2008.