# FENETRE GLISSANTE

## Introduction : Déroulement de l'algorithme

On dispose de deux images : un pattern issu d'une base de données et une image dans laquelle on veut savoir si l'on peut trouver ce pattern, et si oui à quel endroit. Pour faire cela, on crée une fenêtre que l'on va déplacer dans l'image. Après chaque déplacement, on mesure un score de corrélation entre la région de l'image à l'intérieur de la fenêtre, et le pattern que l'on veut identifier. Une fois que l'on a calculé tous les scores, on les trie et on ne garde que ceux dépassant un certain seuil. Les coordonnées des fenêtres associées à ces scores sont les endroits où se trouvent le pattern.

Plusieurs questions se posent. Concernant la fenêtre, comment choisir sa taille et est-ce pertinant de parcourir toute l'image ? Comment se calcule le score de corrélation et quels critères permettent de déterminer la seuil permettant distinguer les bons et les mauvais scores ?

## Définition de la fenêtre glissante

Pour trouver le pattern dans l'image, on veut aussi tirer partie du fait que, dans la plupart des jeux 2D, les éléments sont agencés sur une grille (*tilemap*) dont chaque case (tuile) est susceptible de contenir le pattern. Cette information nous permet de chercher le pattern à des emplacements prédéfinis, plutôt que de parcourir toute l'image pixel par pixel. En revanche, il nous faut connaître les coordonnées de la grille par rapport à l'image (*offsets*) et les dimensions d'une tuile, comme indiqué sur l'image ci-dessous. Evidemment, le gain de performances est énorme.


Même si c'est souvent la même en pratique, en théorie les dimensions des éléments du jeu sont différentes de celles d'une tuile (on peut imaginer qu'un objet soit de dimension deux fois plus petite que la tuile et qu'il

puisse se trouver aux quatre coins de celle-ci; en le cherchant au milieu on ne le trouvera jamais). De plus, les patterns de la base de données ne conservent pas leurs dimensions une fois rendus dans le jeu. Pour ces deux raisons, il nous faut l'information de *scaling* entre la base de données et l'image afin de redimensionner notre pattern aux dimensions de ce même pattern dans l'image. Ceci amène quelques erreurs d'interpolation qui restent négligeables.

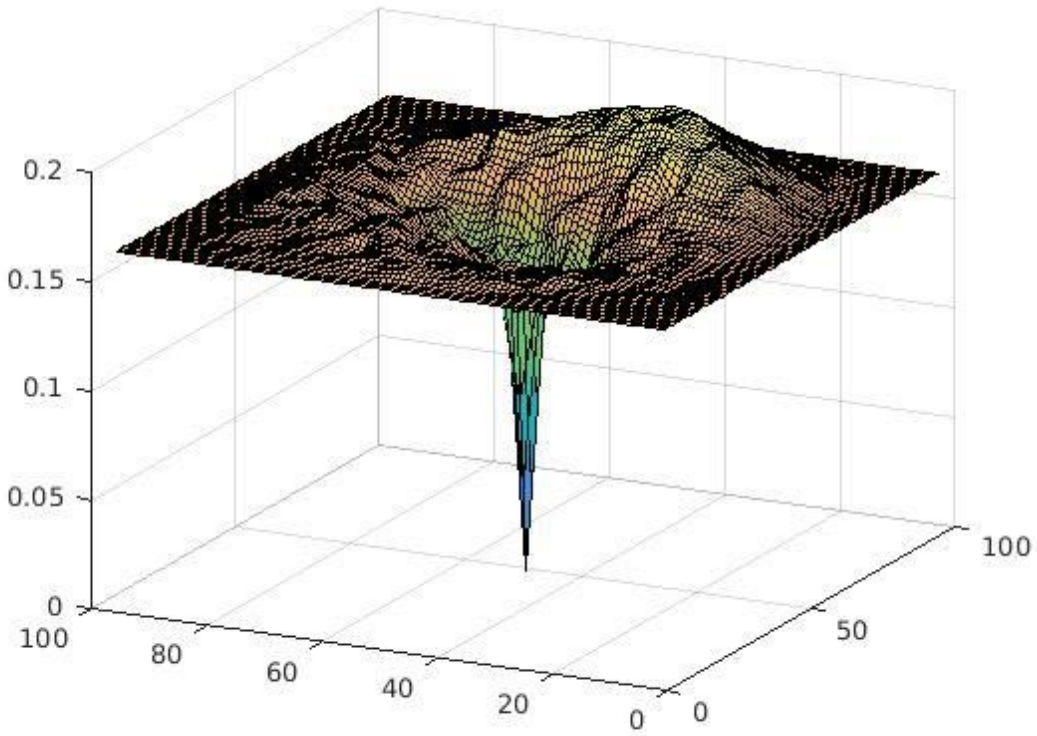
Sur des cas simples où le pattern occupe toute la tuile, il nous faut donc 2 informations a priori : l'*offset* de la grille par rapport aux bords de l'image ainsi que le ratio entre le pattern de notre base de données et le pattern dans l'image. Dans des cas plus compliqués, il nous faut également les dimensions d'une tuile dans l'image. Pour l'instant, je n'ai traité que des cas simples.

# Mesure de la corrélation

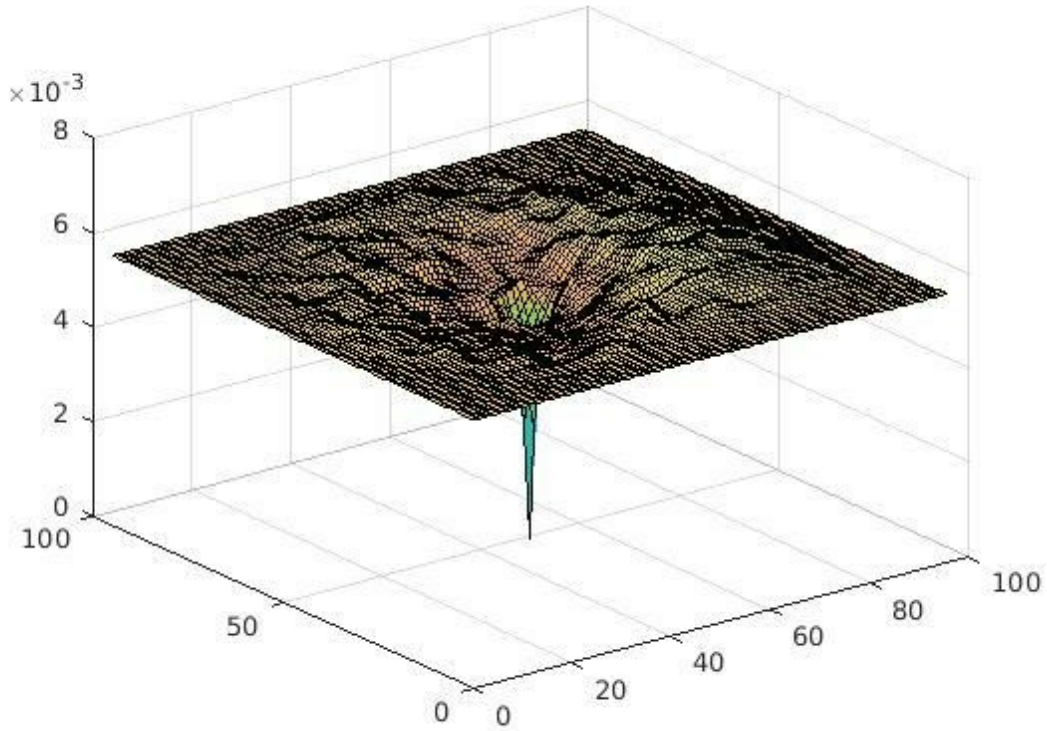
Une fois que l'on a positionné notre fenêtre sur une case de la grille, on peut comparer la sous-image correspondant à cette case avec notre pattern aux bonnes dimensions.

On commence donc pas créer un masque à partir du canal alpha du pattern. Ce masque nous permet de mesurer la corrélation uniquement entre les pixels utiles de notre pattern (ceux où il y a de la couleur). On peut ensuite mesurer la corrélation entre ces pixels utiles à l'aide de différentes mesures : L1, L2 ou ZNCC par exemple.

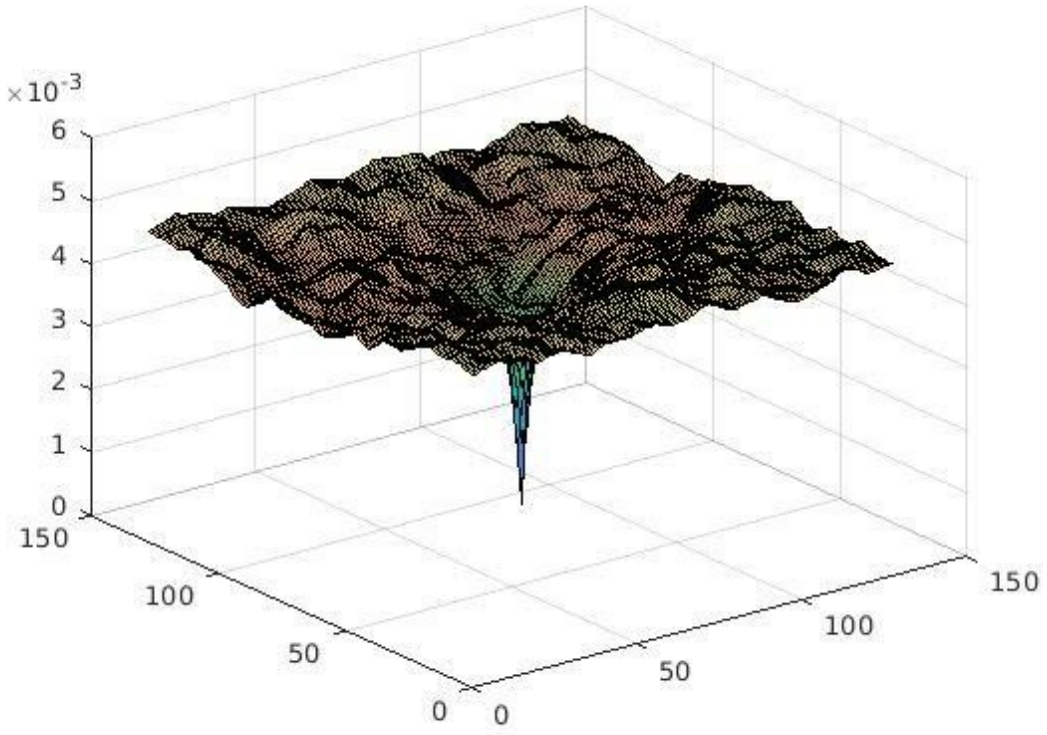
Les figures ci-dessous illustrent l'influence, sur ces mesures, du décalage entre la fenêtre et le pattern. On voit que si l'on applique la fenêtre précisémment où se trouve le pattern, les mesures arrivent à se distinguer. En revanche, le moindre pixel de décalage entraine de grosses erreurs de mesure. Comme on peut le voir sur la dernière figure, c'est la mesure L2 qui est la moins sensible aux décalages puisque entre 2 et 4 pixels de décalage, on peut toujours considérer le score comme correct.



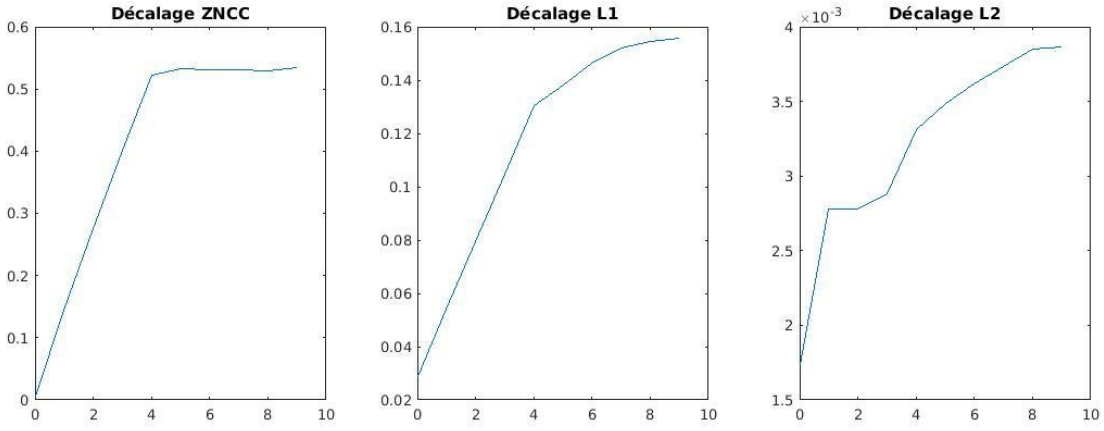

Mesure L1




Mesure L2




> Mesure ZNCC




> Mesures en fonction du nombre de pixels de décalage

# Résultats et pistes d'amélioration

Les résultats obtenus sur des exemples simples où les éléments sont de même dimension que la tuile sont très satisfaisants.

Cependant, la qualité des résultats est très sensible à l'*offset* de la grille. J'ai remarqué que les éléments à trouver ne sont pas toujours au centre de la tuile, mais parfois décalés de quelques pixels. Or pour le moment je ne cherche qu'au centre de la tuile. Un moyen simple de palier à ce problème est de modifier ma base de

données de patterns de sorte qu'ils soient décalés de la même manière que dans le jeu.


> Détection des baies épicées avec un offset optimal, puis avec un offset décalé de 2 pixels par rapport à l'offset optimal.

On peut aussi filtrer simplement les cases qui n'ont rien à voir avec le pattern. En comparant les histogrammes du pattern et de chaque case de la grille, on peut seuiller afin de traiter uniquement les cases contenant plus ou moins les bonnes couleurs. En plus, cette méthode n'est quasiment pas impactée par le décalage.