

Implémentation de l'algorithme du fast-marching pour la résolution d'un labyrinthe

Jean Prost Lucas Potin Edouard Gouteux

28 mai 2019

1 L'algorithme du fast marching

La méthode du fast marching a été introduite par James Setian. Cette méthode possède notamment des applications en mécanique des fluides et en traitement d'image. Cette méthode permet de résoudre l'équation d'Eikonal, de la forme :

$$|\nabla T| = \mathcal{F} \quad (1)$$

Ici, \mathcal{F} et T sont des fonctions $\mathbb{R}^n \rightarrow \mathbb{R}$, ou n peut prendre la valeur 1, 2, où 3. \mathcal{F} est la métrique donnée du problème, et T est la fonction à déterminer, avec comme condition initiale $T(x_0) = 0$. Ici nous nous concentrerons sur des fonctions \mathcal{F} et T de $\mathbb{R}^2 \rightarrow \mathbb{R}$:

$$|\nabla T(x, y)| = \mathcal{F}(x, y) \quad (2)$$

L'algorithme du fast-marching nous permet d'obtenir la solution de viscosité de l'équation d'Eikonal. D'un point de vue géométrique, la fonction $T(x, y)$ obtenue correspond alors à la distance au point initial x_0 sur la surface $x, y, \mathcal{F}(x, y)$, la distance étant alors donnée par $ds^2 = \mathcal{F}^2(x, y)(dx^2 + dy^2)$.

On suppose que la fonction F nous est donnée sur une grille $(i\Delta x, j\Delta y)_{i=\{1, \dots, N\}, j=\{1 \dots M\}}$. On cherche alors à connaître la valeur de T pour tous sommet de la grille.

1.1 Schéma numérique

L'algorithme du fast-marching repose sur approximation "ascendante" du gradient donnée par :

$$|\nabla T(x, y)|^2 \approx \left(\left(\max(D_{ij}^{-x}T, -D_{ij}^{+x}T, 0) \right)^2 + \left(\max(D_{ij}^{-y}T, -D_{ij}^{+y}T, 0) \right)^2 \right)^{\frac{1}{2}} \quad (3)$$

Où D_{ij} est l'opérateur approchant la dérivée en $T_{ij} \equiv T(i\Delta x, j\Delta y)$:

$$D_{ij}^{-x}T = \frac{(T_{ij} - T_{i-1,j})}{h} \quad (4)$$

L'algorithme consiste donc à résoudre :

$$\begin{cases} (max(D_{ij}^{-x}T, -D_{ij}^{+x}T, 0))^2 + (max(D_{ij}^{-y}T, -D_{ij}^{+y}T, 0))^2 & = \mathcal{F}_{ij}^2 \\ T(x_0, y_0) & = 0 \end{cases} \quad (5)$$

On suppose $\Delta x = \Delta y = h = 1$. On a alors :

$$max(D_{ij}^{-x}T, -D_{ij}^{+x}T, 0) = max(T_{i,j} - min(T_{i-1,j}, T_{i+1,j}), 0)$$

L'approximation 5 devient :

$$(max\{T_{i,j} - min\{T_{i-1,j}, T_{i+1,j}\}, 0\})^2 + (max\{T_{i,j} - min\{T_{i,j-1}, T_{i,j+1}\}, 0\})^2 = \mathcal{F}_{ij}^2 \quad (6)$$

2 Application à la recherche du plus court chemin dans un labyrinthe

2.1 Première approche

On cherche à trouver le plus court chemin pour aller d'un point a à un point b dans un labyrinthe. Le labyrinthe nous est donné comme une image en noir blanc. Tout d'abord on normalise l'image : aux pixels blanc (où l'on peut passer) on affecte la valeur 1, aux pixels noir (que l'on ne peut pas franchir), on affecte la valeur 0. On construit ensuite notre métrique W à partir de l'imager normalisé I , tel que pour tout pixel ij

$$W_{ij} = \frac{1}{\epsilon + I_{ij}} \quad (7)$$

Cette métrique pénalise donc les passages par les sommets infranchissables. La valeur ϵ au dénominateur permet d'éviter les divisions par zéro. On applique ensuite l'algorithme du fast marching pour cette métrique, en choisissant comme point initial l'entrée du labyrinthe. On obtient alors la fonction T représentant la distance à notre sommet initial par rapport à la métrique W (voir figure 1)

L'obtention du chemin nous est alors donné en appliquant la descente du gradient sur la fonction T obtenue, en partant de l'autre extrémité du labyrinthe (celle que l'on n'a pas utilisé comme sommet initial). Le chemin correspond alors à la suite des points obtenus au cours des itérations successives de la descente (voir figure 2)

2.2 2 fois plus de fast marching !

Le chemin obtenu est correcte, néanmoins il a tendance à "coller" les bords. Cela n'est pas très beau visuellement, et, pour certaines applications cela peut être problématique. Par exemple, dans le cas de la planification du chemin d'un robot, on peut souhaiter que notre robot ne passe pas trop près des bords. Pour remédier à ce problème, une solution est d'inclure dans la métrique W une pénalité pour les passages près des bords. Pour cela il faut re en mesure de mesurer

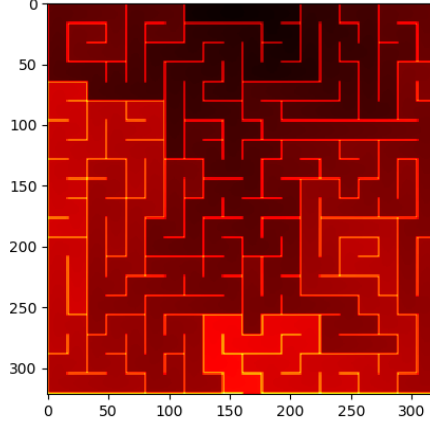


FIGURE 1 – distance au sommet initial (en haut au centre)

la distance d'un point du labyrinthe à son bord le plus proche. L'algorithme du fast-marching nous permet justement de réaliser cette opération. Pour cela on effectue l'algorithme en choisissant comme sommets initiaux l'ensemble des bords du labyrinthe. On appellera la solution obtenue S (speed), dans le sens où, plus la distance au bord sera importante, plus l'on pourra se déplacer au rapidement (figure 3)

Une fois la fonction S calculé, nous réappliquons l'algorithme du fast-marching, en ajoutant à notre métrique W un facteur prenant en compte la distance au bord :

$$W_{ij} = \frac{1}{\epsilon + I_{ij}} + \alpha * \frac{1}{\epsilon + S_{ij}} \quad (8)$$

Le paramètre α nous permet de choisir l'importance de la pénalisation des passages près des murs. Plus le paramètre α sera important, plus le chemin trouvé sera éloigné des murs. Si on prend $\alpha = 0$, alors on retrouve le chemin obtenu avec la première méthode (figure 4).

2.3 Autres labyrinthes

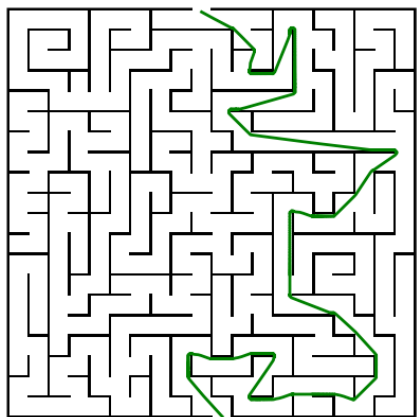


FIGURE 2 – Solution du labyrinthe obtenue avec la descente du gradient

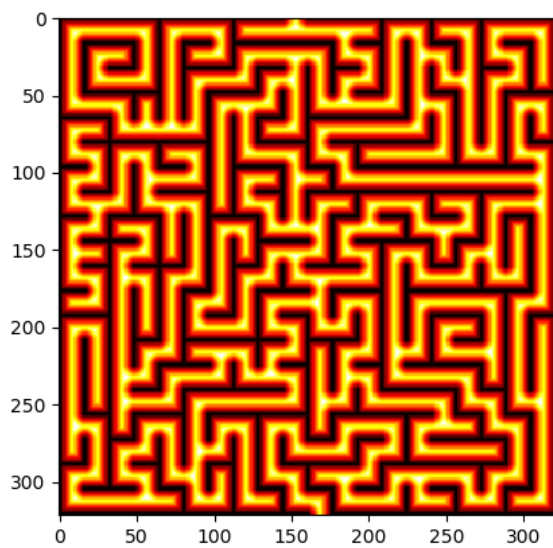


FIGURE 3 – Distance aux bords, en initialisant l'algorithme avec les sommets correspondants aux bords

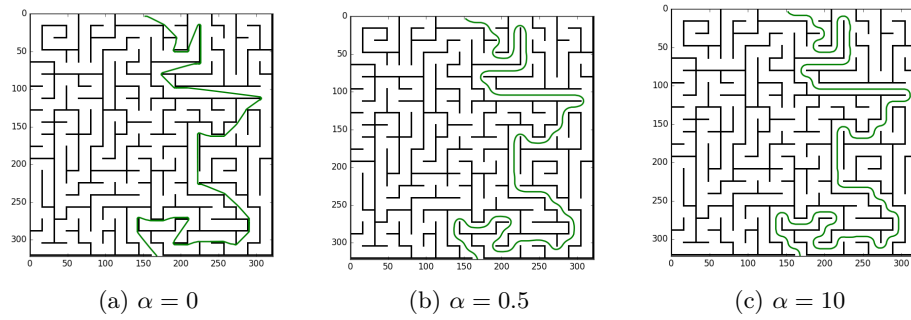


FIGURE 4 – Influence du choix du paramètre alpha sur le chemin obtenu

