Génération de routes

1st Mikel-ange Barros
computer science department
University Claude Bernard Lyon 1
Lyon, France
mikel-ange.barros@etu.univ-lyon1.fr

2nd Zacharia Beddalia
computer science department
University Claude Bernard Lyon 1
Lyon, France
zacharia.beddalia@etu.univ-lyon1.fr

3rd Thomas Scouflaire
computer science department
University Claude Bernard Lyon 1
Lyon, France
thomas.scouflaire@etu.univ-lyon1.fr

***Abstract*—Dans ce papier nous proposons une méthode pour générer des routes respectant les contraintes d'un terrains et imposées par un utilisateur. La trajectoires prises par cette route vont minimiser une fonction de coût prenant en compte ces différentes contraintes.**

***Index Terms*—Road generation, terrain modeling, QT**

I. Introduction

La modélisation de terrains réalistes est un problème de plus en plus important de nos jours où l'informatique graphique se dirige vers une recherche d'un réalisme photographique. Ainsi, la modélisation de routes sur un terrain vierge tends à renforcer l'aspect réaliste d'un monde virtuel. Pour ce faire, la route doit respecter différentes contraintes afin de respecter ce qui peut être raisonnablement fait, en respectant l'environnement qui l'entoure. Dans ce papier, nous présentons une application et un algorithme permettant d'approximer une route respectant ces dites contraintes.

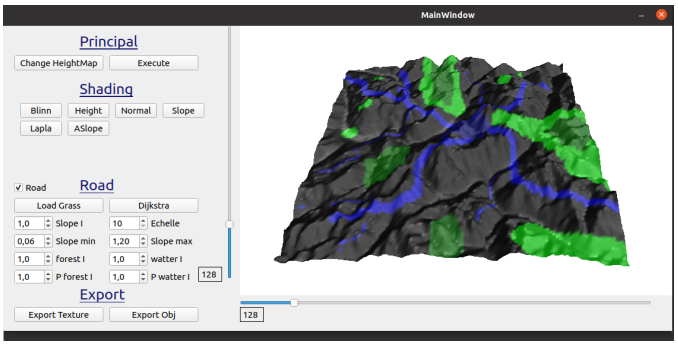
A. Liens

Repository : https://forge.univ-lyon1.fr/p1609419/mmv_tp2
Vidéo de présentation : https://youtu.be/6tvUThjpwDA

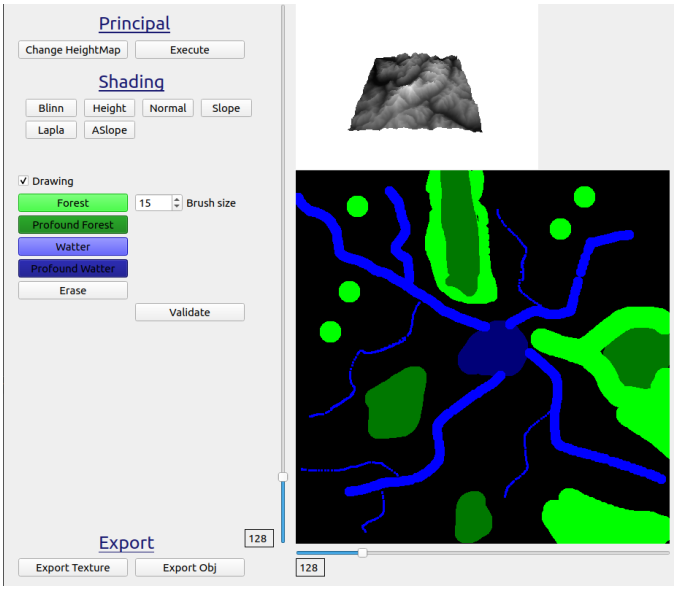
B. L'application

Pour la partie qui nous intéresse, l'application est séparée en deux :

- une partie de génération de routes permettant de lancer l'algorithme et d'en faire varier les paramêtres. (fig 1a)
- une partie de dessin pour positionner nos rivières, lacs et forets. (fig 1b)



(a) Partie route



(b) Partie dessin

Fig. 1. L'application

II. Implémentation

A. Principe

Afin de trouver le meilleur chemin entre deux points du terrain, l'approche la plus courante est de définir un graphe représentant tout les chemins empruntables par notre route et de déterminer lequel sera le meilleur selon une fonction de coût prennant en compte plusieurs paramètres. Ainsi à chaque point nous attribuons un tableau contenant l'ensemble de ces proches voisins ainsi que le coût nécessaire pour atteindre ces dit voisins.

B. Structure de données

Afin de pouvoir mettre en place notre algorithme nous avons du mettre en place deux structures de données :

- une heightmap contenant le type d'environnement auquel appartient une unité de terrain.
- un tableau à double entrée stockant les chemins (en connexité M3) pouvant être pris par notre route

La connexité M3 (figure 2) désigne l'ensemble des voisins directs à un point dans un rayon de trois points autour du point central.

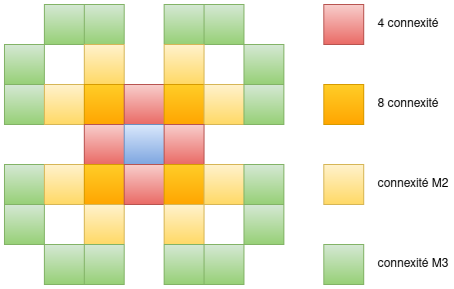


Fig. 2. Connectivité exemple

C. L'algorithme de parcours

Comme expliqué au dessus, nous allons devoir mettre en place un algorithme de parcours, pour cela nous avons utilisé un algorithme de Dijkstra [ROS20] qui nous servira de base pour le parcours.

L'algorithme de Dijkstra permet de trouver le chemin le plus court entre deux points. Pour cela, il calcul un coût dépendant du chemin faisant passer d'un point à un autre et va chercher à minimiser ce coût. Cet algorithme convient parfaitement car il permet de trouver un seul et unique chemin, ce que l'on veut dans le cadre de la génération d'une route. Cependant, si nous avions du construire un réseau routier nous aurions sans doute du changer d'algorithme.

D. La fonction de poids

Ainsi, comme dis au dessus, l'algorithme de Dikstra nécessite de connaitre le coût d'un chemin entre chaque points du graphe pour pouvoir fonctionner. Dans le cadre d'un terrain il y a plusieurs choses à prendre en compte :

- la pente
- la profondeur d'eau
- la densité de végétation

Afin de simplifier le calcul, nous avons fait le choix de définir seulement deux valeurs de poids pour la profondeur de l'eau et deux valeurs de poids pour la densité de végétation. Cela nous permet de visualiser assez bien les chemins pris par la route en fonction des éléments du terrain. Le poids de la pente quant à elle est définis selon deux seuils : si on est en dessous du premier seuils alors le poids est nul, si on est entre les deux seuils alors le poids suis une courbe linéaire, si on est au dessus du deuxième seuil alors le poids est infini. Nous pouvons résumé notre fonction de coût comme étant :

$$P(x) = L + Ps + Pe$$

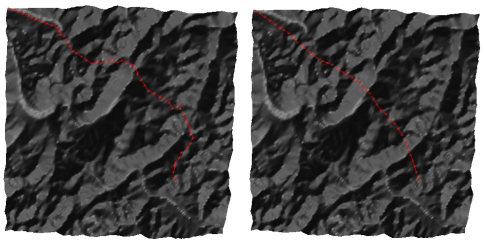
.

Avec P(x) le poids total du chemin x, L la distance entre les deux points, Ps le poids de la pente et Pe le poids de l'élément du noeud correspondant au milieu (eau, vegétation).

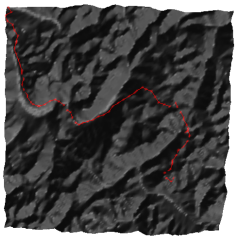
Afin de pouvoir simuler différents résultats, l'influence de chacun des paramêtres de la fonction est modifiable directement dans l'application. Cela permet de définir si l'utilisateur préfère raser la foret ou la contourner, construire un pont ou éviter la rivière, etc...

III. Exemple de résultats

Les tests se font sur une heightmap du mont blanc. On utilise une grille de 128x128 pour les résultats. Dans les calculs, la longueur est définie comme la distance entre deux points. Ici, elle sera de l'ordre de quelques centaines d'unités de longueur.



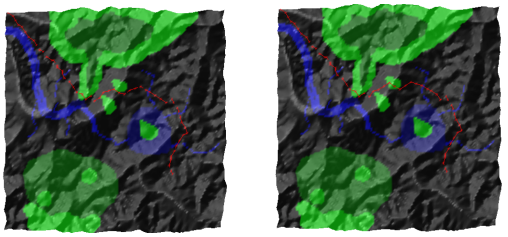
(a) Ps= 100 (b) Ps=10



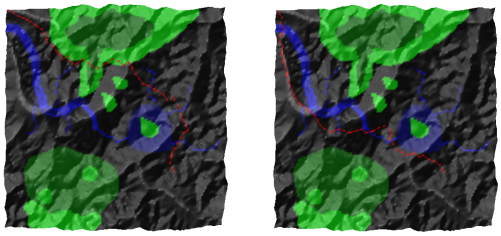
(c) Ps=500

Fig. 3. Terrain neutre, Pe=0

Dans la suite des exemples, F=forêt, FP=forêt profonde, E=eau et EP=eau Profonde.

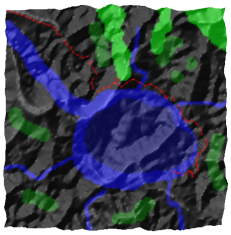


(a) Pe(F)=150, Pe(FP)=500 (b) Pe(F)=15, Pe(FP)=500

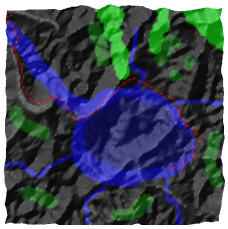


(c) Pe(F)=15, Pe(FP)=50 (d) Pe(F)=750, Pe(FP)=2500

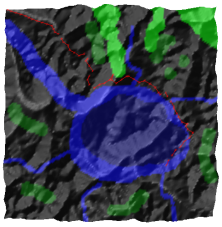
Fig. 4. Terrain colorié, Pe(E)=200, Pe(EP)=1000, Ps=100



(a) Pe(E)=200, Pe(EP)=1000



(b) Pe(E)=20, Pe(EP)=1000



(c) Pe(E)=1000, Pe(EP)=100

Fig. 5. Terrain colorié, Pe(F)=150, Pe(FP)=500, Ps=500

Sur l'ensemble des exemples, nous pouvons remarquer les différents choix fait par l'algorithme en fonction des paramètres. On peut remarquer que chacun des paramètres va influer sur la forme que la route va prendre.

IV. Temps de calcul

Dans cet algorithme, ce qui va le plus influencer le temps de calcul est le parcours de graphe selon Dijkstra (de complexité $O((a+n)\log(n))$ avec a le nombre de chemin et n le nombre de points). Ainsi augmenter ou diminuer la taille de la grille est ce qui posera le plus de problèmes à notre algorithme, nous allons donc comparer les temps en fonction de différentes grilles afin de voir pour qu'elle taille de grilles nous pouvont obtenir un résultat satisfaisant dans un temps acceptable.

Taille grille	128*128	256*256	512*512	1024*1024
Temps (s)	0,201	0,847	3,53	15,225

Fig. 6. Performances

Le temps d'execution de cet algorithme reste acceptable pour une grille de taille inférieure à 512x512, pour une grille plus grande et si l'on veut générer beaucoup de route, on commence à obtenir des temps de plus en plus long dûs à la multiplication du nombre de points et de chemin.

V. Limite de l'approche

Notre algorithme à une limite majeure qui est l'imprécision sur la profondeur de l'eau et la densité de végétation. En effet, cela nous empêche de faire une estimation très fine du chemin optimale. Pour palier à ce problème, nous pourions définir un ensemble de valeurs plus précises ([0,255] par exemple) et faire une courbe de poids comme ce qui a été fait pour la pente.

Reference

[GAL10] Procedural Generation of Roads
[ROS20] Rosettacode