Devoir de Programmation : Algorithmes de Toussaint et de Ritter
Conception Pratique de l'Algorithmique 2015

Maxime Bonnet

Master 1 - Semestre 2

Contents

Chapter 1

Introduction

La gestion de collisions est un problème que l'on retrouve très souvent dans les simulations ou dans les jeux vidéos. Dans ces domaines, il est courant de devoir trouver des collisions entre des dizaines de milliers d'entités. Il faut donc que le calcul de collisions entre deux entités soit le plus rapide possible. Pour cela, on a recours à des approximations. Les entités étudiées peuvent avoir des formes complexes (par exemple, un personnage, un arbre), et il n'existe pas d'algorithme de complexité satisfaisante pour de tels polygones. On essaye donc de se ramener à des polygones plus simples, sur lesquels on est capable de calculer rapidement. La première solution est de trouver l'enveloppe convexe du polygone étudié. L'enveloppe convexe correspond à un contour grossier du polygone. Si la précision attendue par l'algorithme de collisions n'est pas extrêmement pointue, l'étude d'intersection entre les enveloppes convexe des entités suffit amplement. Mais, tout comme les polygones représentant les entités elles-mêmes, il n'existe pas non plus d'algorithme trivial pour calculer l'intersection de deux polygones convexes. C'est pour cela que l'on va essayer de trouver une forme géometrique encore plus basique englobant tous les points de l'entité. On décrira dans ce rapport deux algorithmes répondant à ce problème. L'algorithme de Toussaint permet de trouver le rectangle d'aire minimum contenant tous un nuage de points. L'algorithme de Ritter quant à lui, permet de trouver une approximation du cercle d'aire minimum contenant tous les points.

Chapter 2

Algorithme de Toussaint

2.1 Introduction

Dans son article de 1983, Godfried Toussaint présente un algorithme linéaire qui permet de trouver le rectangle d'aire minimum englobant tous les points d'un nuage de points. Il se base sur une idée de Shamos, qui est d'utiliser deux droites parallèles pour tourner autour de l'enveloppe convexe, à la manière d'un pied à coulisse. Ainsi, le polygone n'est parcouru qu'une seule fois, et l'algorithme se termine en temps linéaire.

2.2 Résultats

2.2.1 Implémentation

On décrit ici l'implémentation choisie, qui se base sur l'article de Toussaint dans lequel il décrit son algorithme. Le langage de programmation choisi est le JAVA.

L'algorithme : L'algorithme suppose que le nuage de points est réduit à son enveloppe convexe, celle-ci est passée en paramètre d'entrée de l'algorithme. On utilise le parcours de Graham pour la déterminer. On ne se concentrera pas sur l'implémentation du parcours de Graham dans ce rapport. L'algorithme de Toussaint se base sur un théorème :

Théorème 1. *Le rectangle d'aire minimum contenant un polygone convexe a un de ses cotés collinéaire avec l'un des cotés du polygone.*

Grâce à ce théorème, on obtient un premier algorithme naïf qui consiste à construire pour chaque coté de l'enveloppe convexe un rectangle contenant toute l'enveloppe. Cet algorithme se termine en un temps quadratique, car la construction du rectangle se calcule en temps linéaire, pour chaque coté de l'enveloppe. Grâce à l'utilisation des pieds à coulisse néanmoins, il est possible de réduire se temps de calcul à un temps linéaire. On utilise deux paires de pieds à coulisse, qui resteront orthogonales pendant toute l'execution de l'algorithme. Elles vont tourner autour du polygone, se calquant à chaque itération de l'algorithme sur un des cotés de l'enveloppe. Grâce à ces lignes, la construction du rectangle se fait en temps constant. Pour mettre en place ces pieds à coulisse, la première étape consiste à chercher les quatre points de coordonnées extrêmes du polygone, nommés i, j, k et l . On notera que l'algorithme ne marche pas si deux de ces quatre points sont confondus. On supposera donc par la suite que ces quatre points sont distincts. Après avoir obtenu ces quatre points, il nous faut créer des droites, on crée donc les *droites de support*, correspondant aux lignes des pieds à coulisse pour chaque point. Celles-ci sont créées parallèle à un des axes, suivant le point. On calcule ensuite les coordonnées des intersections des droites, pour obtenir le premier rectangle. Maintenant que nous avons un rectangle, la boucle principale de l'algorithme commence. On calcule tout d'abord pour chaque point i, j, k ou l, l'angle θ entre sa droite de support et le prochain coté de l'enveloppe convexe. On veut trouver l'angle minimum, pour cela on calcule le cosinus de chaque angle, car le plus grand cosinus correspond au plus petit angle. On effectue alors une rotation du rectangle de l'angle θ minimum. Ainsi, le coté du rectangle dont l'angle était minimum va devenir confondu avec le prochain coté de l'enveloppe. De plus, comme la rotation se fait de l'angle le plus petit, il est garanti qu'il n'y ait pas d'intersection entre le polygone et le rectangle. Maintenant que l'un des cotés du rectangle est confondu avec un des cotés de l'enveloppe convexe, on peut calculer l'aire du rectangle. On itère sur la boucle tant que toute l'enveloppe n'a pas été parcourue entièrement. En sortant de la boucle, on a donc considéré tous les rectangles possibles ayant un coté en commun avec l'enveloppe convexe. On renvoie alors le rectangle ayant l'aire la plus petite.

2.2.2 Tests

2.3 Discussion

2.4 Conclusion

Chapter 3

Algorithme de Ritter

3.1 Introduction

3.2 Résultats

3.3 Discussion

3.4 Conclusion

Chapter 4

Conclusion