Sorbonne Université

Master Informatique

Algorithmique Avancée
MU4IN500

Rapport de projet

Rectangle et cercle minimum d'un ensemble de points

Étudiant :
Léo CHECK

Superviseurs :
M. Danisch
BM. Bui-Xuan

12 janvier 2020

Les collisions entre entités est un problème courant lors de la conception d'un jeu vidéo, mais leur détection peut devenir très gourmande en ressource dès lors que la complexité et le nombre des formes géométriques augmentent. Afin de conserver des performances correctes pour l'expérience de jeu, les développeurs doivent optimiser au maximum ces opérations, notamment en utilisant des conteneurs, simplifiant grandement les calculs.

En effet, réduire les objets à des formes géométriques simples telles que les cercles et les rectangles, permet de réduire les temps de calcul étant donné que détecter leur collision se réalisent en temps constant. Cependant, ces conteneurs sont des approximations, agrandissant la réelle hitbox [1] des entités. Afin de réduire cette perte de précision, il est donc essentiel de réduire leur taille au maximum.
Une autre approche serait d'utiliser les enveloppes convexes. Bien plus précises, elle demande néanmoins plus de temps pour détecter la collision. En effet, étant donné un premier polygone de m côtés et un second de n côtés, il suffit de déterminer si un des côtés de chaque polygone se touchent. D'où une complexité en $\mathcal{O}(m \times n)$.

L'algorithme de *Toussaint* et l'algorithme de *Ritter*, permettant respectivement d'obtenir un rectangle d'aire minimum et le cercle d'aire minimum d'un ensemble de points P, seront les cas d'études de ce rapport. Nous analyserons notamment leur efficacité et leur qualité en tant que conteneur, sur une base de test de 1664 jeux de données à 256 points, dans un plan en 2 dimensions (Varoumas). Par efficacité, nous entendons le temps de calcul nécessaire à la création du conteneur, et par qualité, l'aire du conteneur divisé par l'aire de l'enveloppe convexe : plus le rapport est petit, plus l'aire du conteneur et de l'enveloppe convexe sont proches, et donc précis.

1. masque de collision

Chapitre 1

Algorithme Toussaint

1.1 Description et analyse de l'algorithme

L'algorithme de *Toussaint* permet d'obtenir, à partir d'un ensemble P de n points, son rectangle d'aire minimum couvrant R. Il se compose en 8 étapes majeures :

1. Calculer l'enveloppe convexe de P, noté $HULL$.
2. Calculer les indexes des points cardinaux, notés N, P, W et E, de l'enveloppe convexe $HULL$.
3. Construire le rectangle R avec les droites $Line_N$, $Line_S$, $Line_W$ et $Line_E$ passant respectivement par N, P, W et E.
4. Calculer l'angle minimum θ formé par une droite $Line_X$ et un côté $HULL[X]HULL[X+1]$. De plus, $X = HULL[X+1]$ (où $X \in N, S, W, E$).
5. Effectuer une rotation de θ des 4 droites.
6. Calculer le rectangle $R2$ avec ces dernières.
7. Si l'aire de $R2$ est inférieure à celle de R, $R = R2$ (mise à jour).
8. Tant que tous les points de $HULL$ n'ont pas été traité, on effectue les étapes 4 à 6. Sinon on retourne R.

La complexité de cet algorithme dépend de plusieurs facteurs, notamment de l'implémentation du calcul de l'enveloppe convexe en étape 1. Nous choisissons ici l'algorithme de Graham. Il nécessite un tri par pixel au préalable, qui se réalise en $\mathcal{O}(n)$. Pour chaque triplet successif de points qui "tournent à droite", on retire le points du milieu. On a alors $\mathcal{O}(n+n) => \mathcal{O}(n)$. Puis, de

l'étape 2 à 7, on réalise des calculs en temps constants, répétés sur les $n-1$ côtés de l'enveloppe convexe. D'où $\mathcal{O}(n)$. Par conséquent, la complexité globale de l'algorithme de Toussaint est en $\mathcal{O}(n)$.

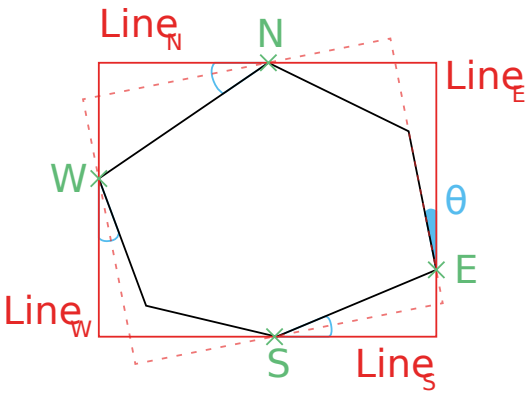

FIGURE 1.1 – Principe de l'algorithme de *Toussaint*

1.2 Résultats

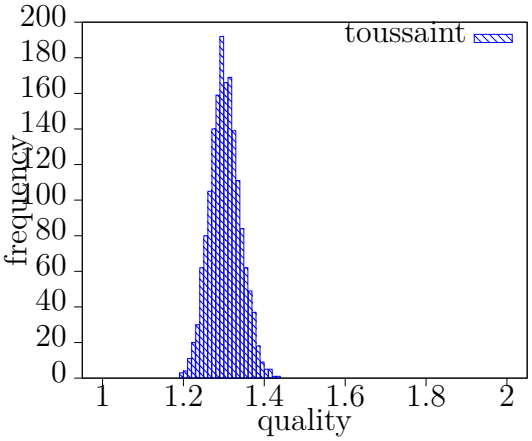

FIGURE 1.2 – Qualité du conteneur rectangle d'aire minimum par instance de *Varoumas*

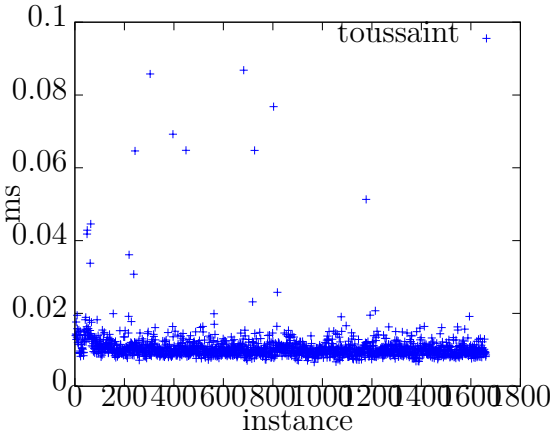

FIGURE 1.3 – Temps de calcul (ms) par instance de *Varoumas*

Nous pouvons dès à présent constater sur la figure 1.1 que la qualité du rectangle minimal sur différentes données varie peu. Avec un écart type d'environ 0.03, elle est en moyenne de 1.30. En d'autre terme, l'aire du rectangle d'aire minimal est 30% plus grand que l'aire de l'enveloppe convexe en moyenne.

De plus, le temps de calcul sur les instances de données de *Varoumas* est globalement constant. Le rectangle minimum d'un ensemble P de 256 points est obtenu très rapidement, en 0.013 ms en moyenne.

Afin d'évaluer comment le temps de calcul croît avec la taille d'une instance de donnée, nous créons un nouvel ensemble P_1 de 500 000 points aléatoires. Puis, nous tirerons au hasard 256 points de P_1, et les stockerons dans un ensemble P_2. Enfin, nous mesurerons l'exécution de l'algorithme sur P_2 en ajoutant au hasard un autre point de P_1. Nous réitérerons ces opérations jusqu'à ce que P_2 soit de taille 100 000.

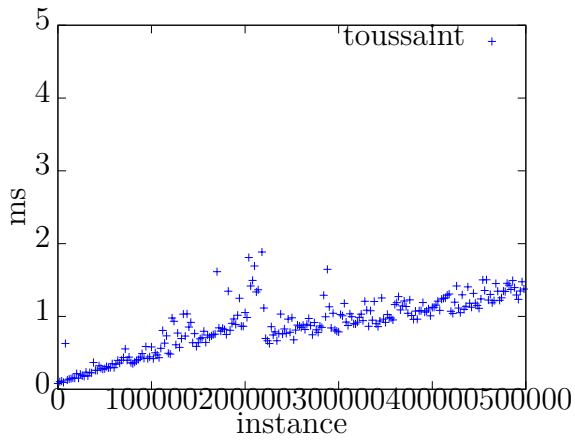

FIGURE 1.4 – Évolution du temps de calcul de *Ritter* avec le nombre de points (nombre de l'instance = nombre de points

Le temps d'exécution de l'algorithme de *Toussaint* augmente linéairement avec le nombre de points contenu dans l'instance de test, vérifiant notre analyse de complexité en $\mathcal{O}(n)$.

Chapitre 2

Algorithme Ritter

2.1 Description et analyse de l'algorithme

L'algorithme de *Ritter* permet d'obtenir, à partir d'un ensemble P de n points, son cercle d'aire minimum couvrant C. Il se compose en 9 étapes majeures :

1. Considérer un point arbitraire *dummy* de l'ensemble P.
2. Calculer un point P de distance maximale à *dummy*.
3. Calculer un pojnt Q de distance maximale à P.
4. Soit le cercle C de centre X, de rayon $|XP|$, passant par P et Q.
5. Retirer de P tous les points compris dans le cercle C
6. S'il reste des points dans P, C n'a pas encore couvert P. Soit S un de ces points. Sinon, on retourne C.
7. Tracer la droite passant par S et X, coupant C en deux points. Soit T, le point le plus éloigné de S
8. Calculer le nouveau cercle C ayant pour diamètre le segment ST, centré en X le centre de ST.
9. Boucler à l'étape 5.

Dans cet algorithme, nous réalisons aux étapes 1, 4, 6, 7 et 8 des opérations en temps constant, et aux étapes 2, 3 et 5 en $\mathcal{O}(n)$ (parcours de tous les points dans P). Nous bouclons les étapes 5 à 8 un nombre de fois limité a, dépendant des points S tiré à l'étape 7. En effet, en fonction de la distance $|SX|$, le nombre de points retirés à l'étape 5 à la prochaine itération sera plus ou moins grand, affectant directement la vitesse d'exécution de l'algorithme.

Nous obtenons donc un algorithme $\mathcal{O}(a \times n)$.
Plusieurs approches nous permettraient de réduire le facteur a. Nous pourrions par exemple effectuer un tri par pixel de l'ensemble P pour réduire sa taille, ou calculer l'enveloppe convexe de P au préalable.

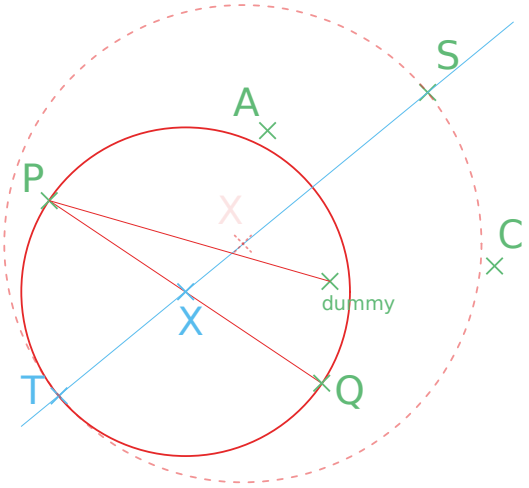

FIGURE 2.1 – Principe de l'algorithme de *Ritter*

2.2 Résultats

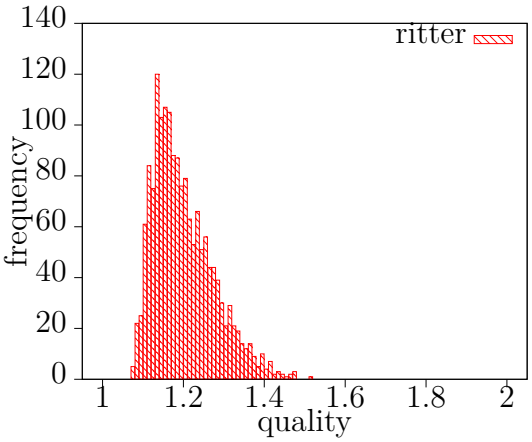

FIGURE 2.2 – Qualité du conteneur cercle d'aire minimum

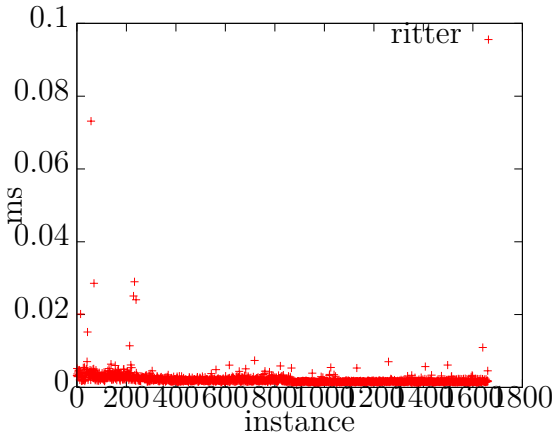

FIGURE 2.3 – Temps de calcul (ms)

L'air du cercle minimum est en moyenne 19% plus grande que celle de l'enveloppe convexe, bien meilleur que celle du rectangle minimum. Mais il

est important de noter la variation importante de la qualité sur la figure 2.1. En effet, l'écart type est de 0.07, soit plus de 2 fois plus important que celui du premier algorithme.

Le temps de calcul du conteneur est constant sur les jeux de données de Varoumas. Nous obtenons ainsi le cercle minimum d'un ensemble de 256 points en 0.002 ms en moyenne.

De la même manière que pour l'algorithme de *Toussaint*, nous allons évaluer comment le temps de calcul de l'algorithme de *Ritter* évolue avec le nombre de point. Notons que l'évaluation à été réalisée avec le même ensemble de point aléatoire.

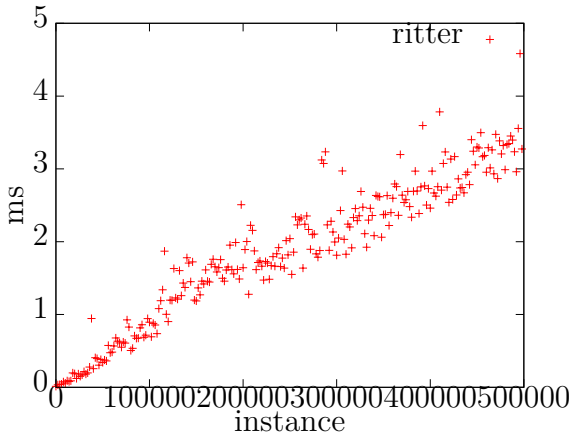

FIGURE 2.4 – Évolution du temps de calcul de *Toussaint* avec le nombre de points (nombre de l'instance = nombre de points

Le temps d'exécution de l'algorithme de *Ritter* augmente linéairement avec le nombre de points contenu dans l'instance de test, vérifiant notre analyse de complexité en $\mathcal{O}(a \times n)$.

Chapitre 3

Discussion

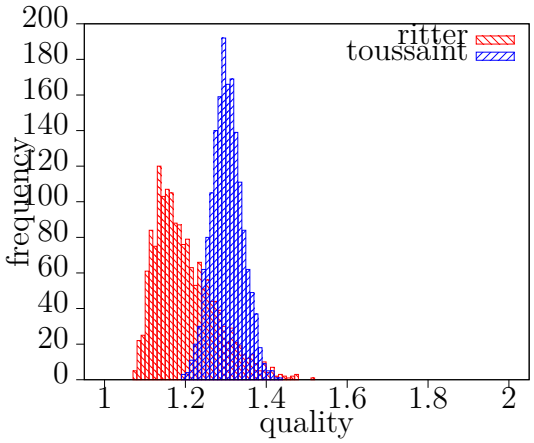

FIGURE 3.1 – Qualité des conteneurs par instance de *Varoumas*

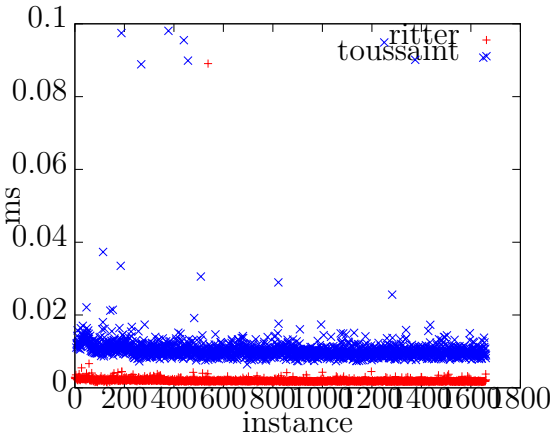

FIGURE 3.2 – Temps de calcul (ms) des conteneurs par instance de *Varoumas*

En comparant la qualité des conteneurs produits par les algorithmes de *Toussaint* et de *Ritter* sur les jeux de données de *Varoumas*, nous pouvons relever deux points.
Tout d'abord, le rectangle minimum est, dans l'ensemble, le plus mauvais conteneur en terme de qualité. Nous avons, en effet, une aire couvrante 30% plus importante en moyenne que celle de l'enveloppe convexe, contre 19% en moyenne pour le cercle minimum. De plus, le rectangle demande 6 fois plus de temps à être calculé.
Pourtant, la disparité de la qualité du rectangle minimum est 2 fois plus petite que celle du cercle minimum. Dans nos expériences, la qualité des rectangles minimums est comprise entre 1.19 et 1.43, tandis que les cercles

minimums entre 1.06 et 1.57.

Cependant, la qualité des conteneurs dépend directement de la "forme" du polygone à couvrir (voir figure 3.3).

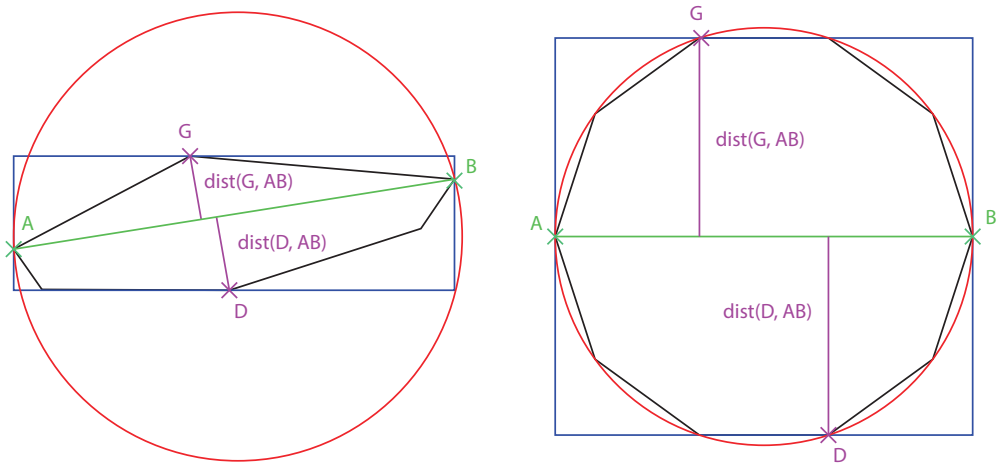

FIGURE 3.3 – Comparaison des conteneurs sur deux formes de polygones différentes

Il est donc important de choisir l'algorithme adéquat pour couvrir un ensemble de point P.
Une première solution serait alors de déterminer en premier temps le diamètre AB du polygone (c'est-à-dire, le segment reliant les deux points les éloignés dans P), et G (respectivement D) le point de P le plus éloigné de AB avec $\vec{AB} \wedge \vec{GB}$ maximum (respectivement $\vec{AB} \wedge \vec{AG}$ minimum). Ensuite, calculer :

$$r = |1 - \frac{|AB|}{dist(G, AB) + dist(D, AB)}|$$

Ainsi, plus le rapport tend vers 1, plus la largeur et la longueur du polygone sont proches et r petit. Ainsi, en fixant un ϵ jouant le rôle de délimiteur, nous pouvons soit appliquer l'algorithme de *Toussaint* si $r > \epsilon$ ou de *Ritter* si $r \leq \epsilon$.

Regardons à présent comment les temps de calcul des deux algorithmes évoluent avec le nombre de point sur la figure 3.4. Sur un ensemble P de 20 000 points ou moins, l'algorithme de *Ritter* est le plus efficace. En augmentant d'avantage le nombre de point, celui de *Toussaint* devient un meilleur choix en terme de performance. Évoluant tous les deux de façon linéaire comme démontré précédemment, nous pouvons clairement confirmer depuis le graphique nos analyses de complexité en $\mathcal{O}(a \times n)$ pour le premier et en $\mathcal{O}(n)$ pour le second.

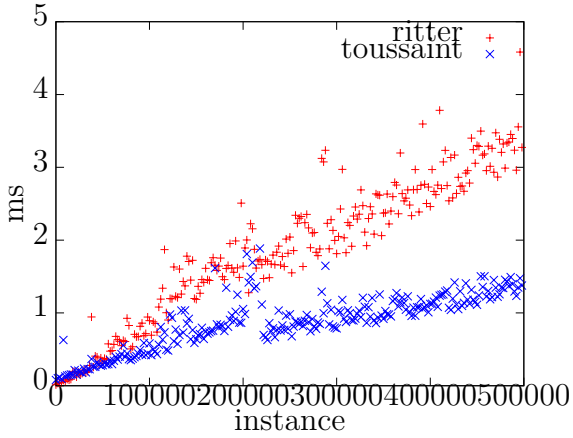

FIGURE 3.4 – Comparasion de l'évolution des temps de calcul de *Toussaint* et de*Ritter* avec le nombre de points (nombre de l'instance = nombre de points

Chapitre 4

Conclusion

Les algorithmes de *Toussaint* et de *Ritter* permettent d'obtenir respectivement des rectangles et des cercles d'aire minimum de qualité relativement similaires d'un ensemble de point donné. De complexité $\mathcal{O}(n)$ pour le premier et $\mathcal{O}(a \times n)$ pour le second, ils évoluent de manière linéaire avec le nombre de points à traiter. Au dessus de 20 000 points, il est préférable d'utiliser l'algorithme de *Toussaint* si nous cherchons la performance.
Nous avons vu par ailleurs que la qualité dépend directement de la "forme" de l'enveloppe convexe de l'ensemble de point. Si nous visons la précision du conteneur, il faut savoir adapter l'algorithme utilisé. Nous avons proposé une solution qui consistait à calculer le rapport entre le diamètre de l'enveloppe convexe, et la somme des distances entre les points "les plus à gauche et à droite" au diamètre.

Mais le problème de détection de collision ne s'arrête pas là pour les développeurs. En effet, ces algorithmes permettent seulement d'optimiser le traitement local d'une collision entre deux polygones. Dans une application réelle, nous traitons bien plus d'objets. Ainsi, dans un contexte en 2D, détecter si une collision a eu lieu entre N polygones se calcule en $\mathcal{O}(N^2)$. Différentes solutions existent : un premier exemple serait les *Quad-tree*, utilisant la méthode "diviser pour régner", qui permettent de réduire la complexité à $\mathcal{O}(N \log N)$. Mais ceci reste un autre sujet d'étude.