Sorbonne université

UE ALGAV

Rapport de projet

Maxence BRUNET

Enseignant : BM Bui-Xuan

27 Decembre 2019

Introduction

Dans les jeux vidéos et la géometrie algorithmique, la détéction de collision implique l'utilisation d'algorithmes. Les algorithmes de détection de collisions dans les jeux en 2 dimensions dépendent de la forme des objets à détecter. Pour celà, on favorisera l'utilisation de masque de collision pour couvrir l'entité (Hitbox) plûtot que la comparaison de chacuns des pixels entourant l'objet. Une comparaison de chaque pixels formant l'objet à détecter devient très couteux en termes de performance dès lors que le nombre de pixels croît ou que le nombre de collisions à détecter augmente. C'est pour celà que nous allons nous restreindre à l'utilisation de forme générique tel que le rectangle avec l'algorithme de Toussaint ainsi que le cercle avec l'algorithme de Ritter. Dans ce rapport, nous voulons analyser la qualitée en tant que conteneur d'un rectangle minimum couvrant un ensemble de points dans le plan. Dans un premier temps nous verrons en détails l'implémentation de l'algorithme de Toussaint, puis celui de Ritter. Enfin nous finirons avec la comparaison de ces deux algorithmes, avec l'implémentation de test de performances et de qualités.

1 Algorithme de Toussaint

Il est utile de trouver des conteneurs de délimitation de surface minimale qui impliquent uniquement l'évaluation de quelques expressions pour tester leur inclusion. Le rectangle de délimitation minimum est un tel conteneur. Il s'agit de la surface minimale d'un rectangle avec une orientation arbitraire, qui contient un ensemble de sommets S d'un objet géométrique. En 2D, il existe un algorithme bien connu pour trouver le rectangle minimum en utilisant une technique appelée *rotating calipers* de G.Toussaint. Cette technique s'applique à un polygone convexe et permet de trouver le rectangle minimal en une complexité en *O(n)*.

Pour un objet non convexe, il faut d'abord calculer son enveloppe convexe, ce qui peut se faire dans le meilleur cas en une complexité en *O(nlog(n)*. Dans notre cas, le calcul de notre enveloppe convexe s'effectue avec l'algorithme de Graham modifié qui nous donne une complexité en O(n) dans le pire cas.

L'algorithme de Toussaint consiste donc à obtenir un rectangle d'aire minimum contenant un ensemble de points S.

Notre algorithme se décompose en 7 étapes:

1. Trouver les points *Pi,Pj,Pk,Pl* de l'enveloppe convexe, respectivement d'abscisse minimum, d'ordonnée minimum, d'abscisse maximum, et d'ordonnée maximum.

2. Construire les 4 droites *line_i,line_j,line_k,line_l* passant respectivement par *Pi,Pj,Pk,Pl*.

3. Trouver l'angle minimum *angle_min* formé par une des droites et un côté de l'enveloppe convexe. Pour celà, on calcul le cosinus de l'angle à l'aide de 2

droites: La droite passant par son point associé (ref: étape 2) et un côté de l'enveloppe convexe.

```
//Cosinus de l'angle entre deux lignes L1 et L2
public static double cos(Line l1, Line l2) {

   //cosinus(u.v)=u*v/|u|*|v| avec u et v deux vecteurs
   double cos=l1.getVec().ProduitDeVecteur(l2.getVec())/
   (l1.getVec().norme()*(l2.getVec().norme()));

    return Math.abs(cos); //valeur absolue du cosinus
   }
```

4. Effectuer la rotation d'angle *angle_min* des 4 droites (ref: étape 2).

5. Créer le rectangle formé par l'intersection des droites.

6. Calculer l'aire du rectangle et mettre à jour le rectangle minimum si besoin.

7. Calculer les nouveaux angles après rotation et répéter les étapes 3-6 pour tous les côtés de l'enveloppe convexe.

1.1 Description des outils utilisées

Pour la réalisation de notre algorithme de Toussaint ont à besoin de différents outils. Le premier est l'enveloppe convexe, calculé à partir de l'algorithme de Graham modifier. Cet algorithme consiste à effectuer un Trie Pixel sur un ensemble de points. Ce trie se réalise en temps O(n). Puis on parcours le résultat de se trie pour retirer les points qui ne sont pas dans l'enveloppe convexe .On obtiens un algorithme avec une complexité en *O(n+n)* soit *O(n)*.

Trie par Pixel: "Si trois points d'un ensemble de points ont le même abscisse, alors au plus deux de ces points appartiennent à l'enveloppe convexe S".

Graham :"Si P, Q et R sont trois points de Points d'abscisse croissant, et R appartient au "mauvais côté" des demi-plans définis par PQ, alors Q n'est pas sur l'enveloppe convexe de Points"

```
//Extrait de l'Algorithme de Graham modifier
   public static ArrayList<Point> enveloppeConvexe(ArrayList<Point>
       points){ //-->O(n)
       if (points.size()<4) return points;

       ArrayList<Point> result = TriePixel(points); //O(n)
       for (int i=1;i<result.size()+2;i++) { //O(n)
           Point p = result.get((i-1)%result.size());
           Point q = result.get(i%result.size());
           Point r = result.get((i+1)%(result.size()));
           if (crossProduct(p,q,p,r)>0) {
               result.remove(i%result.size());
               if (i==2) i=1;
               if (i>2) i-=2;
           }
       }
       return result;
    }
```

On a également besoin d'une nouvelle classe *Line* permettant d'effectuer plus facilement des rotations d'angles. Elle prendra donc un point représentant l'un des points de l'enveloppe convexe et un vecteur prenant en paramètre 2 coordonnées (x et y).

```
//Extrait de la classe Line
public class Line {
   private Point p;
   private Vecteur vec;

   public Line(Point p, Vecteur vec) {
      this.p=p;
      this.vec=vec;
   }

   public Line(Point p1, Point p2) {
      if (p1.x == p2.x) {
         vec = new Vecteur(0, 1);
      } else {
         vec = new Vecteur((p2.x - p1.x), (p2.y - p1.y));
      }
      p = p1;
   }
```

```java
//Extrait de la classe Vecteur
public class Vecteur {

   private double x;
   private double y;

   public Vecteur(double x, double y) {
      this.x = x;
      this.y = y;
   }
```

La dernière classe utilisé est la classe *Rectangle* qui permet de tracer le rectangle minmum de notre algorithme. Il contient deux constructeurs. L'un pour ajouter les 4 points au Rectangle, et l'autre qui éffectue l'intersection de 4 droites pour obtenir les 4 points du rectangle.

```java
//Extrait de la classe Rectangle
public class Rectangle {
   private Point.Double a,b,c,d;

   public Rectangle(Point.Double a,Point.Double b,Point.Double
       c,Point.Double d) {
      this.a=a;
      this.b=b;
      this.c=c;
      this.d=d;
   }

   public Rectangle(Line l1, Line l2, Line l3, Line l4) {
      this(l1.intersection(l2), l2.intersection(l3),
          l3.intersection(l4), l4.intersection(l1));
   }
```

1.2 Compléxite de l'algorithme de Toussaint

La réalisation de l'algorithme a permis de déterminer une complexité en O(n):

Etape 1: On parcours l'ensemble des points de notre enveloppe convexe pour trouver le maximum et le minimum en abscisse et ordonnonées. On obtiens une complexite en O(n).

Etape 7: On parcourt l'ensemble de l'enveloppe convexe pour les étape 3 à 6 décrites précedemment. On obtiens ainsi une complexite en O(n).

Le reste des étapes de l'algorithme décrit précedemment étant des opéreations élémentaires.

On souhaite effectuer une comparaison en terme de performance et de qualité de cette algorithme avec celui de Ritter. On décrira en détails le fonctionnement de son algorithme.

2 Algorithme de Ritter

Le cercle minimum d'un objet géométrique est le plus petit cercle contenant un objet. Dans n'importe quelle dimension, le cercle minimum d'un objet géométrique linéaire, défini par l'ensemble S de son ensemble de sommets est unique et est spécifiée par un point central C et un rayon R. Il existe plusieurs algorithmes pour calculer exactement le cercle minimale pour un ensemble S de n points. Certains auteurs ont noté que le cercle minimale peut être dérivée directement du "Diagramme de Voronoi" ((1)) qui peut être calculé en une complexité en O(nlog(n)). Il n'en reste pas moins difficile à mettre en œuvre dans la pratique. De plus, il a été montré que le cercle minimum peut être calculée en utilisant un algorithme de programmation linéaire randomisé qui s'exécute avec une complexite en O(n). C'est le cas pour l'algorithme de Ritter qui permet d'avoir une très bonne approximation du cercle minimum.

L'algorithme utilisé est le suivant,il se décompose en 11 étapes:

1. Prendre le premier point dans la liste des points de départ.On le nommera *Dummy*

2. Parcourir l'ensemble des points pour trouver un autre point de distance maximum au point *Dummy*. On le nommera *P*.

3. Parcourir l'ensemble des points pour trouver un autre point de distance maximum à *P*. On le nommera *Q*.

4. Determiner le point C tel que C est le centre du segment [PQ]. On obtiens alors un cercle de centre C et de rayon [CP].

5. Parcourir la liste des points et retirer tous les points qui sont dans le cercle. Si suite à cette étape la liste des points est vide alors on renvoie le cercle courant, sinon on passe à l'étape 6

6. Prendre le premier point dans la liste des points restant. On le nommera S.

7. On détermine la droite passant par S et C. Elle coupe le cercle courant en deux points. Soit T le point le plus éloigné de S.

8. On consière le point C' le centre du segment [ST]. Le segment [ST] étant la longueur du segment [CP] plus la longueur du segment [SC]. On détermine les coordonnées de C' à l'aide des coordonnées barycentriques ((2)).

9. Répeter les étape 5 à 8 jusqu'a qu'il ne reste plus de points.

2.1 Description des outils utilisées

Pour la réalisation de notre algorithme de Ritter ont a besoin d'un outil en particulier. On redéfini une classe "Circle" dont le constructeur prend en paramètre

un centre representé par un point et un rayon.

```
//Extrait de la classe Circle
public class Circle {
    public Point centre;
   public int rayon;

   public Circle(Point centre, int rayon) {
      this.centre = centre;
      this.rayon = rayon;
   }
}
```

2.2 Complexité de l'algorithme de Ritter

On détermine une complexite de l'algorithme de Ritter en O(xn):

Etape 2,3,5 : On parcours la liste des points. On obtiens une complexite en O(n)

Etape 9: On répète les opérations élémentaire un nombre limité x de fois. On obtiens une complexité en O(x)

Maintenant que l'on connait les structures des algorithmes utilisés, nous allons comparer leur performances ainsi que leur qualité.

3 Test de qualité et de performance

Dans un premier temps, nous allons nous concentrer sur la comparaison des algorithmes en terme de temps d'execution. Les tests ont été effectué sur 1664 instances et chaques instances allant de 256 à 425984 points. Les testbeds ont été obtenus suite à la creation d'un algorithme *filesPointsGenerator* générant des points aléatoire alant de *1 à n* dans x fichiers différents.

D'après le graphique (Fig 1), on observe que le temps d'exécution des algorithmes de Toussaint et Ritter augmente de façon linéaire en fonction du nombre de points. On remarque que celui de Ritter augmente beaucoup plus rapidement que celui de Toussaint. Celà confirme bien leur complexité respective.

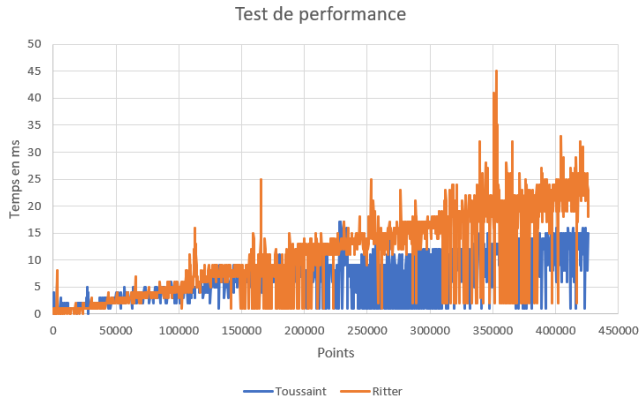

Figure 1: Test de performance pour l'Algorithme de Toussaint et Ritter

En ce qui concerne la qualité du conteneur, on sait que plus la qualité est petite et plus l'aire du conteneur est proche de celle de l'enveloppe convexe. Pour les test de la *figure 2*,on utilise une base de test de 1664 instances et chaques instances possèdent 256 points: C'est la base de test de *Varoumas*

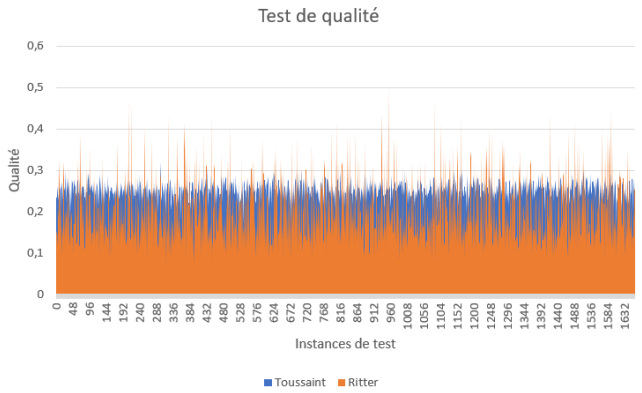

Figure 2: Test de qualité pour l'Algorithme de Toussaint et Ritter

D'après le graphique (fig 2), on observe que Ritter est très légèrement meilleur que Toussaint sur la majorité des instances. La qualité moyenne obtenue par l'algorithme Toussaint est de 0,25 ce qui indique un rectangle d'aire en moyenne 25% plus grande que celle de l'enveloppe convexe. L'algorithme Ritter quant à lui nous donne une moyenne de 0.20 ce qui indique un cerlce d'air en moyenne 20% plus grande que celle de l'enveloppe convexe.

Conclusion

Pour conclure, on est en droit de ce demander quel algorithme est le meilleur. Celà va dépendre de l'utilisation qu'on en fait. Si l'on cherche un algorithme rapide, on aura tendence à se tourner vers l'algorithme de Toussaint qui croit moins rapidement que Ritter lorsque l'on se trouve au delà des 10^4 points. Pour ce qui est de la qualité, celà va fortement dépendre de la forme du nuage de point. Un nuage de point de la forme d'un polygone régulier sera plus intérerssant pour l'algorithme de Ritter tandis qu'un polygone irrégulié le sera plus pour l'algorithme de Toussaint.

References

[1] https://fr.wikipedia.org/wiki/Diagramme_de_Voronoï

[2] https://fr.wikipedia.org/wiki/Coordonnées_barycentriques

Annexe

https://www.geometrictools.com/Documentation /MinimumAreaRectangle.pdf
https://fr.slideshare.net /MagStellonNadarajah /approximation-de-ritter
https://vincentcordobes.github.io/rittertoussaint