2020 – 2021

CFA INSTA – Analyste Développeur

L’algorithme de Welzl résolvant le problème du cercle minimum

Projet de Conception et Pratique Algorithmique

Amine Kheddar

I. Introduction

Le problème du plus petit cercle est un problème mathématique consistant à calculer le plus petit cercle qui contient la totalité d'un ensemble donné de points dans le plan euclidien. Ce problème a été proposé pour la première fois par le mathématicien anglais James Joseph Sylvester en 1857 et de nombreux algorithmes ont depuis été mis au point pour calculer la solution du problème.

Il provient de divers domaines d'application tels que l'analyse de localisation, les opérations militaires, l'ingénierie mécanique (optimisation des contraintes) et médicale (plus petit cercle contenant l'iris dans l'image d'un œil).

Les pixels dont l'intensité est supérieure au seuil sont marqués comme disque optique candidat avec une couleur verte (0xff00ff00) comme indiqué dans l'image 1.

Image 1 : *Image de la rétine avec un disque optique de couleurs vertes*

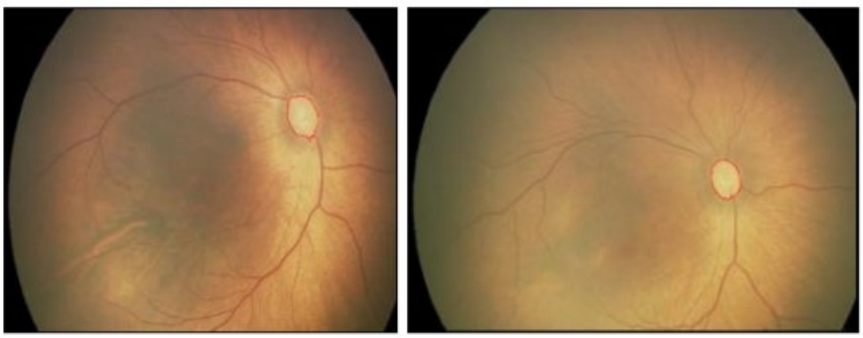


Image 2 : *Bord du disque optique résulte*

En se basant sur la taille normale du disque optique, centrée sur P, il est possible de déterminer un cercle qui peut entourer totalement la région du disque optique. Il faudra tout d'abord analyser la distribution de l'intensité des pixels dans le cercle, et définir les caractéristiques d'intensité des pixels sur le bord selon différente caractéristique. En fonction de ces caractéristiques, certains pixels sont sélectionnés pour calculer le rayon du cercle périphérique minimum (image 2) à d, où d est la constante incrémentielle utilisée pour que le cercle inclue tous les pixels du disque optique et exclue la fausse optique.

Dans l'armée, le cercle d'encerclement minimal est connu sous le nom de "problème des bombes". Si les points de l'ensemble sont considérés comme des cibles sur une carte, le centre du cercle minimal qui les entoure est un bon endroit pour larguer une bombe afin de détruire les cibles, et le rayon du cercle peut être utilisé pour calculer la quantité d'explosif nécessaire.

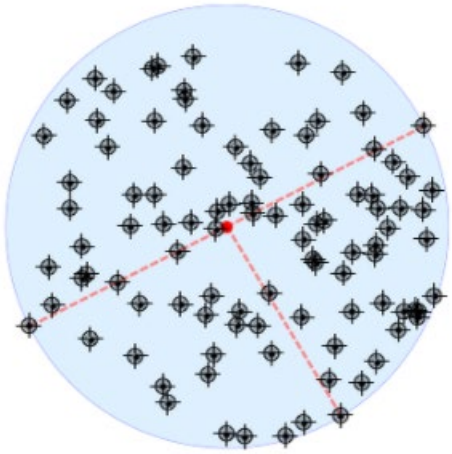


Image 3 : *Exemple d'une simulation possible de tir de missile*

Aujourd'hui, ils existent de nombreux algorithmes efficaces capables de résoudre ce problème :

- ***Chrystal*** : Publié en 1885 par Pr. Chrystal, cet algorithme utilise le fait que les points définissant le cercle minimum se trouveront à la limite de la coque convexe du point fixé.
- ***Eliosoff et Unger*** : Publié en 1998, Eliosoff et Unger ont observer que le cercle minium se base sur la détermination de la coque convexe de l'ensemble de points. Cela est dû au fait que les points de l'ensemble touchés par le cercle minimum se trouvent toujours sur la coque convexe de l'ensemble.
- ***Elzinga and Hearn*** : L'algorithme maintient un cercle de couverture pour un sous-ensemble de points. À chaque étape, un point non couvert par la sphère actuelle est utilisé pour trouver une sphère plus grande qui couvre un nouveau sous-ensemble de points, y compris le point trouvé.

Pour une complexité de O(n^X), la solution à ce problème est triviale pour X <= 3. En effet, plus X sera petit, plus l'algorithme sera efficace et nous permettra d'avoir des résultats rapides même pour une grande quantité de points. Ainsi, l'utilisation d'un algorithme performant sera très importante pour la résolution de ce problème.

L'idée pour ce projet sera d'utiliser l'algorithme récursif de Welzl qui, grâce à une complexité linéaire et plus abordable, nous permettra d'avoir des résultats plus rapides et optimaux.

Prérequis:

Durant l'utilisation de ces algorithmes, plusieurs notions géométriques et fonctions intermédiaires vont être utilisées.

- **Cercle selon 2 points :**

Une équation du cercle de centre C (h ; k) et de rayon r dans le plan muni d'un repère orthonormé est :

$$(x-h)^2+(y-k)^2=r^2$$

La formule de la distance entre deux points A et B :

$$AB=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

Soit M un point quelconque du cercle, de coordonnées (x ; y). Le carré de sa distance à C est donc $(x-h)^2+(y-k)^2$ et c'est, par définition, le carré du rayon.

- **Colinéarité :**

Géométriquement (et même intuitivement), trois points sont alignés s'ils se situent sur une même droite. Les points A, B et C sont alignés si les vecteurs $\overrightarrow{AB}$ et $\overrightarrow{AC}$ nt colinéaires. En effet, deux vecteurs colinéaires ayant un point commun sont les vecteurs directeurs d'une même droite. De façon analytique, on utilise les coordonnées des vecteurs ayant un point commun. Soit les vecteurs :

$\overrightarrow{AB}$ (x ; y) et $\overrightarrow{AC}$ (x ' ; y ') . A, B et C sont alignés si et seulement si ***x y ' – y x ' = 0***

```
public boolean arecolinear(Point p, Point q, Point r) {

    if ((q.x-p.x)*(r.y-p.y)-(q.y-p.y)*(r.x-p.x)==0) {
        return true;
    }
    return false;

}
```

- **Cercle circonscrit :**

Le cercle circonscrit d'un triangle est le cercle qui passe par les trois sommets de ce triangle. Les trois médiatrices d'un triangle sont concourantes en un point qui est le centre du cercle circonscrit à ce triangle.

- **Implémentation du cercle circonscrit pour 3 points :**

```
double cX,cY;
int radius;

double alpha1=(q.x-p.x)/(double)(p.y-q.y);
double beta1=(0.5 * (p.y+q.y))-alpha1*(0.5 * (p.x+q.x));
double alpha2=(r.x-p.x)/(double)(p.y-r.y);
double beta2=(0.5 * (p.y+r.y))-alpha2*(0.5 * (p.x+r.x));


cX=(beta2-beta1)/(double)(alpha1-alpha2);
cY=alpha1*cX+beta1;
radius=(int) ((p.x-cX)*(p.x-cX)+(p.y-cY)*(p.y-cY));
Point center = new Point();
center.setLocation(cX, cY);
```

II. Les Algorithmes

Afin de résoudre ce problème, nous allons exploiter une approche naïve et optimisée afin de voir l'efficacité de chacun. Bien que les solutions aient tous les deux la même fonctionnalité, nous allons voir que la complexité de chacun va jouer un rôle très important dans le calcul du cercle minimum.

1) Algorithme naïf

Pour chaque paire de points de l'ensemble (p,q), le cercle sera créé puis on regarde si ce cercle contient tous les points de l'ensemble . Si c'est le cas on peut choisir ce cercle comme étant le cercle minimum. Il est cependant possible qu'après avoir testé toutes les possibilités qu'on ne trouve pas de cercle minimum.
Une 2ème idée simple peut être formée pour résoudre ce problème. L'idée est d'utiliser tous les triples de points pour obtenir le cercle circonscrit défini par ces points. Après avoir obtenu le cercle, il faut vérifier si les autres points sont entourés par ce cercle et renvoyer le plus petit cercle valide trouvé (celui dont le rayon est le plus petit et contenant tous les points).

A partir de cette rapide mise en forme, on peut en déduire les 2 lemmes suivants :

- **Lemme 1** : Soit P un ensemble de points dans un plan donné. Soit C le cercle de diamètre égale à la distance de deux points couvrant tous les autres points du plan, ce cercle est appelé le cercle de couverture minimal.
- **Lemme 2** : Soit P un ensemble de points dans un plan donné. Soit C, le cercle passant par 3 points du plan non colinéaire et couvrant tous les autres points du plan, ce cercle est appelé le cercle de couverture minimal.

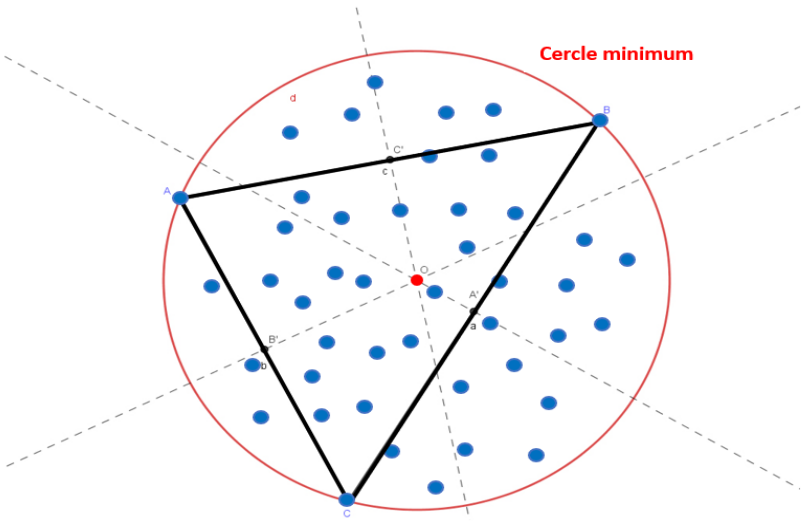


Schéma 1 : *Modélisation du principe de l'algorithme naïve*

- Implémentation :

Lemme 1

```
for (Point p: points){
    for (Point q: points){

        cX = .5*(p.x+q.x);
        cY = .5*(p.y+q.y);
        radius = 0.25*((p.x-q.x)*(p.x-q.x)+(p.y-q.y)*(p.y-q.y));

        boolean toucher = true;

        for (Point s: points) {

            if ((s.x-cX)*(s.x-cX)+(s.y-cY)*(s.y-cY)>radius){
                toucher = false;
                break;
            }
        }
        if (toucher == true) {
            return new Circle(new Point((int)cX,(int)cY),(int)Math.sqrt(radius));
        }
    }
}
```

Lemme 2

```
        double cercleminX=0;
        double cercleminY=0;
        double cercleminR=Double.MAX_VALUE;
        Point cerclemincenter = new Point();
        cerclemincenter.setLocation(cercleminX, cercleminY);
        double a1,a2,b1,b2;

        for (int i=0 ; i<points.size() ; i++){
            for (int j=i+1 ; j<points.size() ; j++){
                for (int k=j+1; k<points.size() ; k++){
                    Point p=points.get(i);Point q=points.get(j);Point r=points.get(k);

                    if (arecolinear(p,q,r)) continue;

                    if ((p.y==q.y)||(p.y==r.y)) {
                        if (p.y==q.y){
                           p=points.get(k); r=points.get(i);
                        }else {
                            p=points.get(j); q=points.get(i);
                        }}

                    a1=(q.x-p.x)/(double)(p.y-q.y);
                    b1=(0.5 * (p.y+q.y))-((q.x-p.x)/(double)(p.y-q.y))*(0.5 * (p.x+q.x));
                    a2=(r.x-p.x)/(double)(p.y-r.y);
                    b2=(0.5 * (p.y+r.y))-a2*(0.5 * (p.x+r.x));

                    cX=(b2-b1)/(double)(a1-a2);
                    cY=a1*cX+b1;
                    radius=(p.x-cX)*(p.x-cX)+(p.y-cY)*(p.y-cY);

                    if (radius>=cercleminR) continue;
                    boolean toucher = true;

                    for (Point s: points) {
                        if ((s.x-cX)*(s.x-cX)+(s.y-cY)*(s.y-cY)>radius){
                            toucher = false;
                            break;}}

                    if (toucher == true){
                        cercleminX=cX;cercleminY=cY;cercleminR=radius;
                        cerclemincenter = new Point();
                        cerclemincenter.setLocation(cercleminX, cercleminY);
                    }}}}
        return new Circle(cerclemincenter,(int)Math.sqrt(cercleminR));
    }
```

- Complexité :

Pour le lemme 1, nous aurons une complexité de O(n2) (boucles for imbriquées). A cela nous ajoutons O(n3) pour le lemme 2 (trois boucles for imbriquées) ainsi que le test de couverture du cercle en O(n). La complexité de l'algorithme naïf au pire cas est O(n4) .

2) Algorithme de Welzl

Emo Welzl a proposé en 1991 un algorithme randomisé simple pour le problème du cercle de couverture minimum qui s'exécute en temps attendu O(n), basé sur un algorithme de programmation linéaire de Raimund Seidel.

Le but de l'algorithme est de retirer aléatoirement un point de l'ensemble des entrées données pour former une équation circulaire. Une fois l'équation formée, il faut vérifier si le point qui a été supprimé est inclus dans l'équation ou non. Si ce n'est pas le cas, alors le point doit se trouver à la limite du cercle minimum. Par conséquent, ce point est considéré comme un point limite et la fonction est appelée récursivement.

Le fonctionnement détaillé de l'algorithme est le suivant :

L'algorithme prend comme entrée un ensemble de points P et un ensemble R initialement vide et utilisé pour représenter les points situés sur la limite du cercle minimum.

Le cas initial de l'algorithme est le suivant :

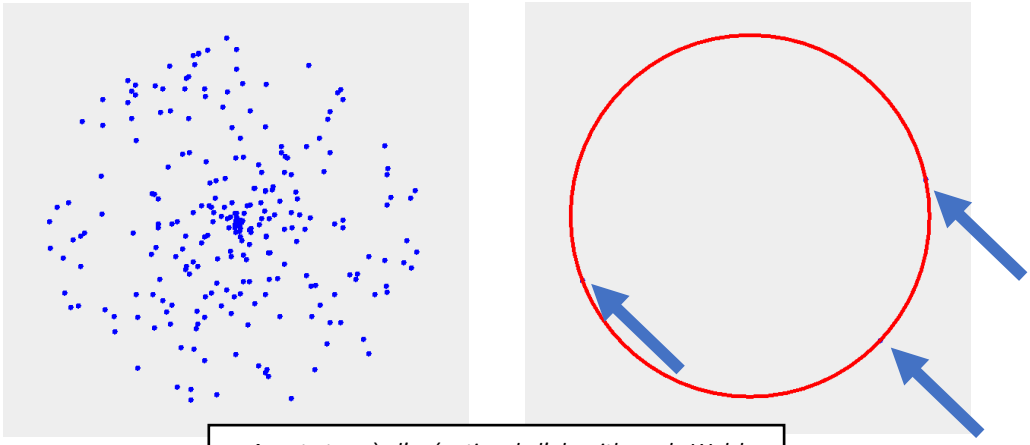
- *P devient vide ou la taille de l'ensemble R est égale à 3.*
- *Si P est vide, alors tous les points ont été traités.*
- *Si |R| = 3, alors on a déjà trouvé 3 points qui se trouvent sur la limite du cercle, et comme un cercle peut être déterminé de façon unique en utilisant seulement 3 points, la récursion peut être arrêtée.*

Lorsque l'algorithme atteint le cas de base ci-dessus, il renvoie la solution triviale pour R, étant:

- *Si |R| = 1, nous retournons un cercle centré sur R[0] avec un rayon = 0*
- *Si |R| = 2, on retourne le Cercle minimum pour R[0] et R[2]*
- *Si |R| = 3, on retourne le Cercle minimum en essayant les 3 paires (R[0], R[1]), (R[0], R[2]), (R[1], R[2])*
- *Si aucune de ces paires n'est valable, on retourne le cercle défini par les 3 points de R*

Si le scénario de base n'est pas encore atteint, nous procédons comme cela :

- *On choisit un point aléatoire p dans P et on le retire de P*
- *Appelez l'algorithme sur P et R pour obtenir le cercle résultat*
- *Si p est entouré de résultat, alors nous renvoyons résultat sinon, p doit se trouver à la limite du cercle minimum*
- *Ajouter p à R*
- *Renvoyer la sortie de l'algorithme sur P et R*



Avant et après l'exécution de l'algorithme de Welzl

- Voici la mise en œuvre de l'approche décrite ci-dessus :

```java
public Circle calculCercleMin(ArrayList<Point> points) {return MinCercle(points, new ArrayList<Point>());}

public Circle MinCercle(ArrayList<Point> liste1, ArrayList<Point> liste2) {
    Circle resultat = null;

    if (liste1.isEmpty()) {
        if (liste2.isEmpty())
            return new Circle(new Point(0, 0), 10);
        if (liste2.size() == 1) {
            resultat = new Circle(liste2.get(0), 0);}
        if (liste2.size() == 2) {
            double cx = 0.5*(liste2.get(0).x + liste2.get(1).x);
            double cy = 0.5*(liste2.get(0).y + liste2.get(1).y);
            double d = 0.5*(liste2.get(0).distance(liste2.get(1)));
            Point p = new Point((int) cx, (int) cy);
            resultat = new Circle(p, (int) Math.ceil(d));
        }
        if (liste2.size() == 3) {

            Point a = liste2.get(0);
            Point b = liste2.get(1);
            Point c =liste2.get(2);
            double tmp = (a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y - b.y)) * 2;
            double alpha = (a.x * a.x) + (a.y * a.y);
            double beta = (b.x * b.x) + (b.y * b.y);
            double zigma = (c.x * c.x) + (c.y * c.y);
            double x = ((alpha * (b.y - c.y)) + (beta * (c.y - a.y)) + (zigma * (a.y - b.y)))/tmp;
            double y = ((alpha * (c.x - b.x)) + (beta * (a.x - c.x)) + (zigma * (b.x - a.x)))/tmp;
            Point p = new Point(((int) x), ((int) y));
            resultat = new Circle(p, (int) Math.ceil(p.distance(a)));
        }}else {

        Point p = liste1.remove(r.nextInt(liste1.size()));
        resultat = MinCercle(liste1, liste2);
        if (resultat != null) {
            double dx = (p.x-resultat.getCenter().x);
            double dy = (p.y-resultat.getCenter().y);

            if(dx*dx+dy*dy <= resultat.getRadius()*resultat.getRadius() == false) {
                liste2.add(p);
                resultat = MinCercle(liste1, liste2);
                liste2.remove(p);
                liste1.add(p);
            }}}
    return resultat;
```

***Cas initial** : Complexité en O(1)*

***Pire cas**: Complexité en O(n)*

III. Test de performance - Résultat

Pour cette partie, les tests seront tous réalisés sur la base de test « Varoumas_benchmark ». Cette base contient 1664 fichiers de 256 points (une ligne va contenir les coordonnées x et y séparer par un espace).

L'objectif est de comparer les temps d'exécutions (en nanoseconde puis convertit en milliseconde) des deux Algorithmes précédemment énoncé précédemment. Les données seront récupérées dans des fichiers .csv et nous les visualiseront à l'aide de **matplotlib** sous python. Pour les figures suivantes, le fichier 1 de la base utilisé contenant 50000 points a été retirer pour pouvoir analyser l'oscillation des valeurs seulement de 256 points par fichier.

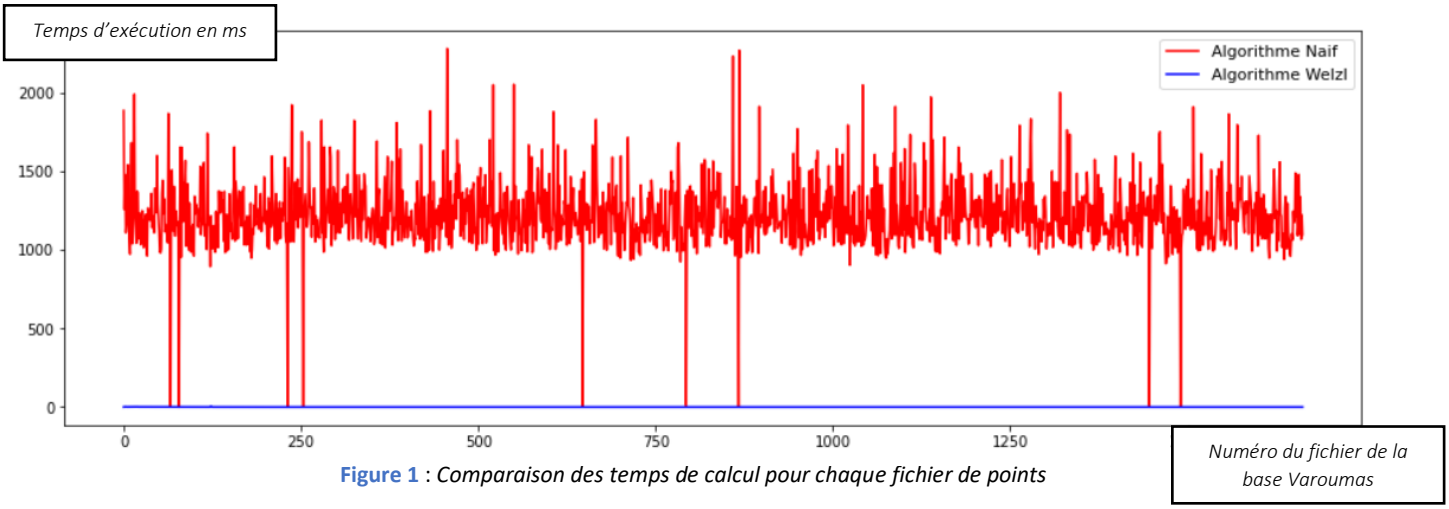


Figure 1 : *Comparaison des temps de calcul pour chaque fichier de points*

Ce graphique représentant le temps de calcul en milliseconde des différents fichiers test nous montre bien la différence d'efficacité entre les deux algorithmes. En effet, on peut remarquer lorsque les deux graphiques sont confrontés, les temps de calculs pour l'algorithme de Welzl semblent représenter une droite linéaire proche de y = 0.

On peut également remarquer que certains temps de calcul sont très rapides et peuvent se résoudre aussi rapidement que l'algorithme de Welzl. Ceci s'explique par le cas où le cercle est déterminé par le lemme 1 de l'Algorithme naïf, qui représente notre résolution de l'algorithme aux meilleurs cas (de valeurs comprises entre 0 et 1 ms). Au pire cas, nous pouvons une valeur qui peut atteindre environ 2500 ms soit environ 2500 fois plus long que l'algorithme de Welzl.

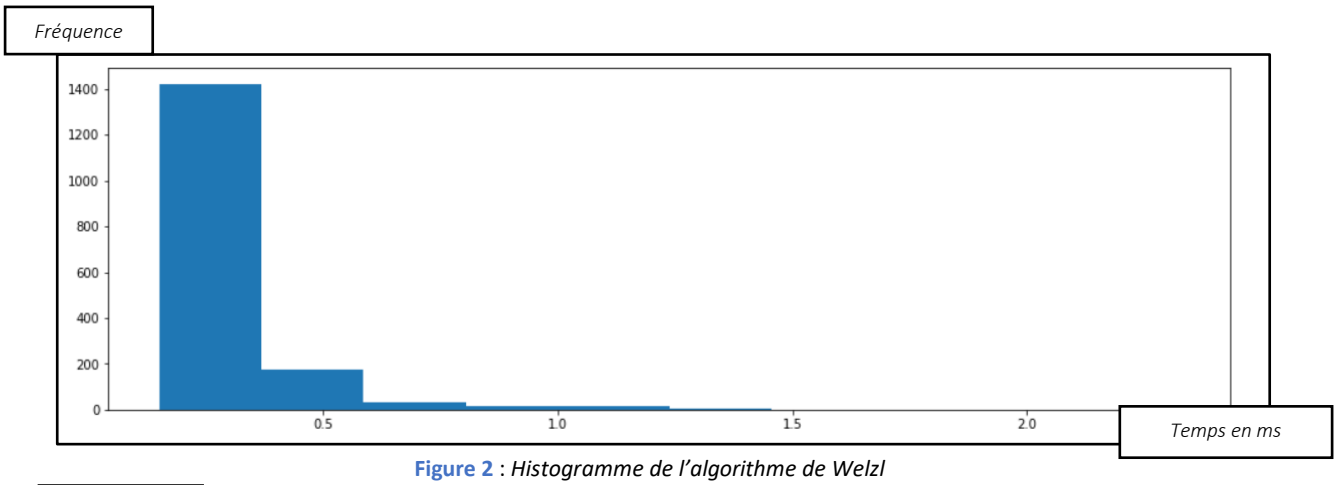


Figure 2 : *Histogramme de l'algorithme de Welzl*

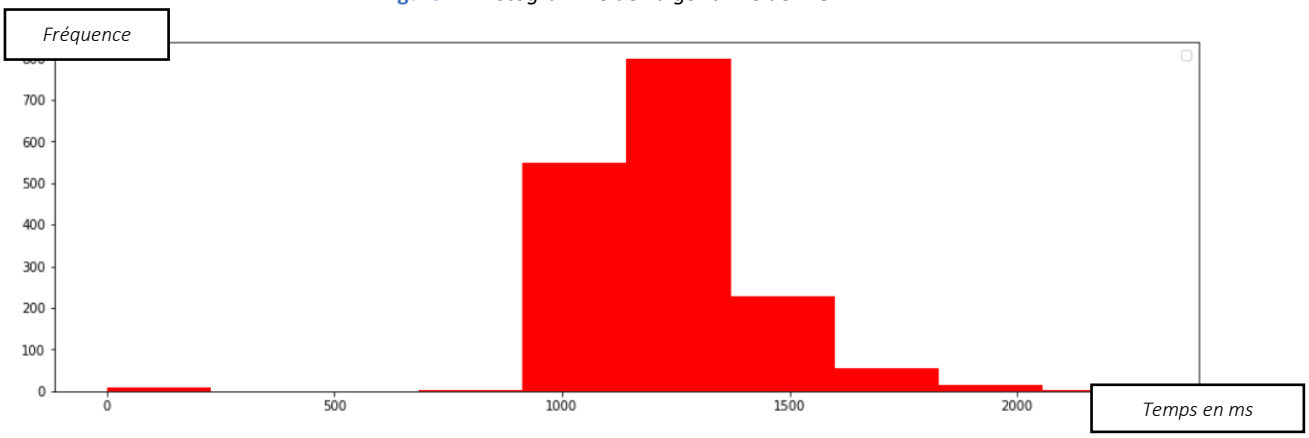


Figure 3 : *Histogramme de l'algorithme Naïf*

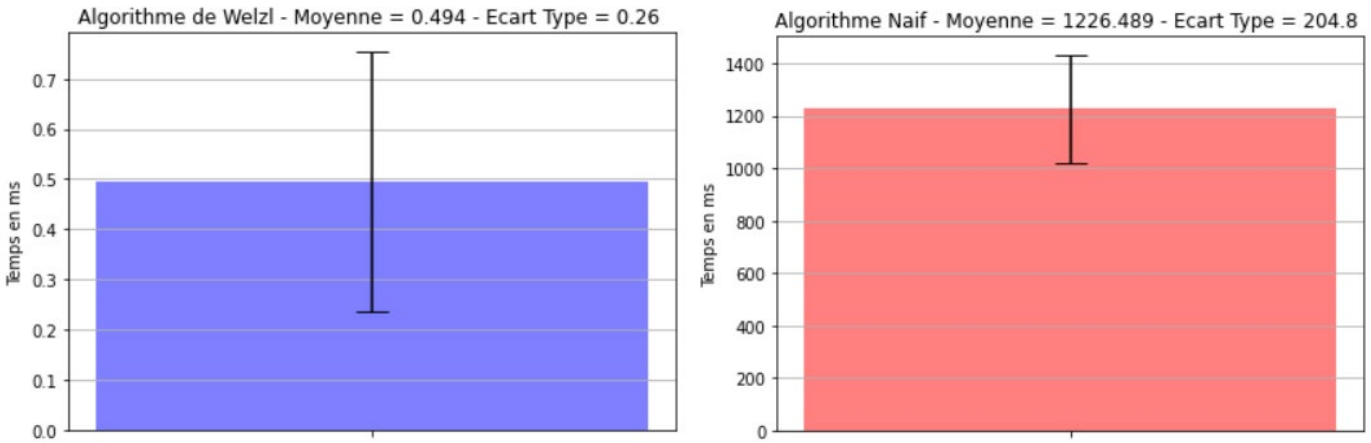


Figure 4 et 5 : *Moyennes et déviations standards*

Les graphiques ci-dessus vont nous permettre de comparer les deux algorithmes plus en détail.

La figure 3, nous confirme les résultats obtenus précédemment avec des valeurs majoritairement comprises entre 1000 et 1600 ms. On peut tout de même remarquer que la majorité des cas se situe entre 1100 et 1300 ms.

Grâce à la figure 2, nous pouvons voir plus en détail comment oscillent les valeurs pour l'algorithme de Welzl. On peut rapidement en conclure que la majorité des temps de calcul pour ce dernier est inférieur à 0.5 ms.

En effet, on peut voir qu'en moyenne (sur les figures 4 et 5), l'algorithme de Welzl est 2500 fois plus rapide (un gain de temps très conséquent pour une très grosse quantité de point).

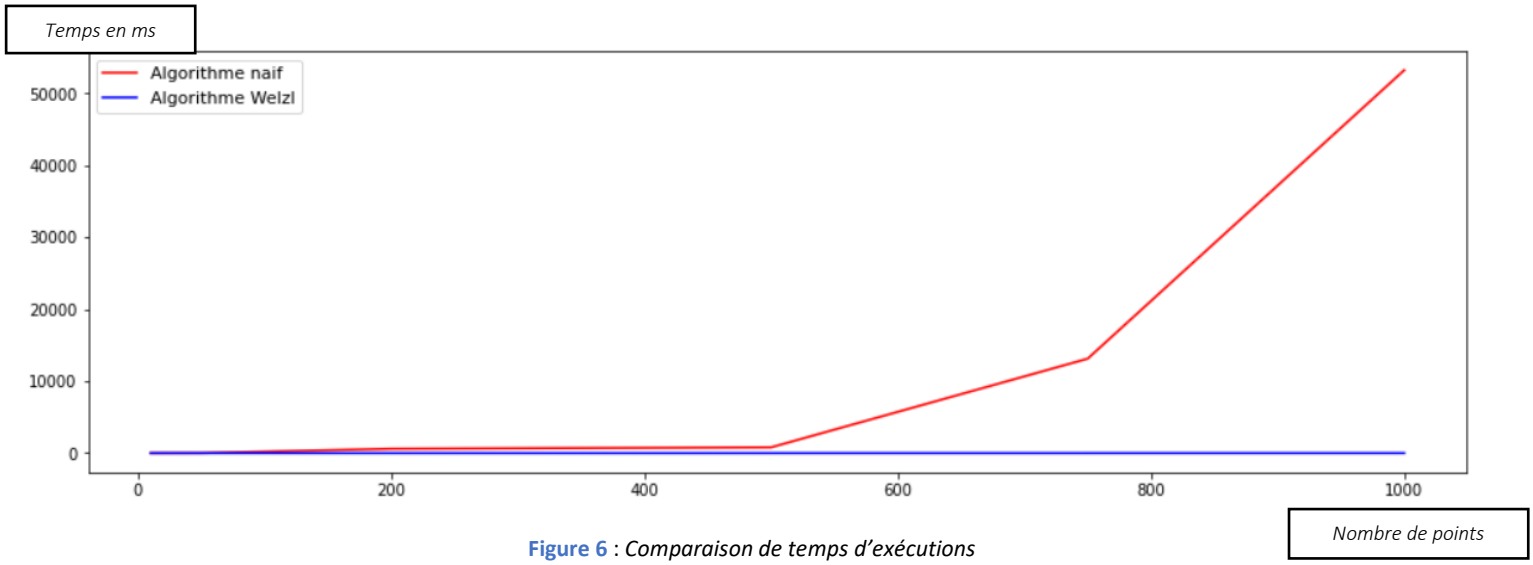


Figure 6 : *Comparaison de temps d'exécutions*

Ces résultats sont démonstratifs de l'efficacité de l'algorithme de Welzl par rapport à l'algorithme naïf. Cette efficacité peut être traduite par le fait que chaque point n'est traité qu'une seule fois puisque celui-ci est supprimé de la liste des points (complexité O(n)), contrairement à l'algorithme naïf, et qu'il n'y a que 256 points par fichier.

Malgré un nombre de points plus important, l'efficacité ne faiblit pas. En effet, on peut voir d'après la figure 6 que le temps d'exécution de l'algorithme naïf croit de manière polynomiale en fonction du nombre de points (preuve de sa complexité en O(n4)) par rapport à celui de l'algorithme de Welzl qui se comporte de manière linéaire avec une droite qui tend vers y=0 (le temps d'exécutions peut croitre en minutes pour un nombre de points trop élevé pour l'algorithme naïf). On peut également noter que les deux algorithmes sont de même qualité en comparant les rayons de chaque cercle minimum (quasiment de même valeur).

Cependant, l'algorithme de Welzl possède tout de même ses défauts.

En effet, l'algorithme a une très grande profondeur de récursion, à savoir le nombre de points d'entrées. Nous savons que chaque appel récursif sera empilé dans la pile qui possède une taille limitée.

Ainsi, on peut en déduire que malheureusement, cet algorithme ne s'applique que sur un nombre limité de point. Pour résoudre ce problème, il faudra changer l'implémentation en version itérative ou utiliser d'autres algorithmes itératifs, cependant ils fonctionnent de manière totalement différente et n'ont pas la durée d'exécution linéaire attendue de l'algorithme de Welzl.

Question Bonus :

Comme une sphère en 3D est uniquement définie par quatre points (deux en 2D), une approche simple naïve pourrait tenter de trouver la sphère minimale en essayant toutes les combinaisons de sphères composées de quatre points, trois points et deux points, en gardant la sphère de taille minimale qui contient tous les points. Cette approche est la pire avec une complexité de O(n5) et ne sera pas envisageable. La meilleure façon dans ce cas sera d'utiliser l'algorithme de Welzl (en utilisant des vecteurs pour vérifier la colinéarité des points).

IV. Conclusion

Les sphères peuvent constituer un grand volume limite dans la détection des collisions et sont généralement de grandes entités géométriques. L'algorithme décrit ici (Welzl) est une approche efficace pour calculer la sphère enveloppante minimale d'un ensemble de points, en temps réel.

La plupart des algorithmes peuvent être faciles à mettre en œuvre et à comprendre, mais ne parviennent pas à fournir des sphères optimales dans la plupart des cas.

Grace aux résultats obtenus, nous pouvons en conclure que, pour une quantité importante, la nécessité d'un bon algorithme est indispensable (cela permet également d'accélérer le prétraitement, ainsi l'algorithme fonctionne lors du chargement d'une scène ou lorsqu'un outil externe crée le volume limite). Malgré le fait que l'algorithme de Welzl soit un algorithme performant, sa limite d'utilisation représente un vrai défaut pour une implémentation récursive (l'implémentation de Gärtner est probablement la plus rapide). Aujourd'hui, il est nécessaire d'avoir un algorithme qui sera capable de calculer le cercle minimum pour des millions de points en un minimum de points.