SORBONNE UNIVERSITY

Faculté des Sciences et Ingénierie

Parcours Science et Technologie du Logiciel (STL)

Conception et Pratique de l'Algorithmique (CPA) : Rédaction d'un rapport de recherche sur l'algorithme Welzl résolvant le problème du cercle minimum

Étudiant :
Abdelkader Boumessaoud, N°21218306, abdelkader.boumessaoud@etu.sorbonne-universite.fr

Encadrant :
Bùi Xuân Bình Minh, buixuan@lip6.fr

19 mars 2024

Table des matières

Table des algorithmes

Table des figures

1 Introduction

1.1 Problème du cercle minimum

Le problème du cercle minimum, également connu sous le nom de problème du cercle englobant minimum, est un problème classique en algorithmique et en géométrie. Il consiste à trouver le cercle de rayon minimum qui englobe un ensemble donné de points dans le plan euclidien.

1.1.1 Historique

Ce problème a été initialement formulé par James Joseph Sylvester en 1857, où il est décrit comme la recherche du "plus petit cercle qui contient un ensemble donné de points dans le plan". Plus tard, en 1885, le professeur M. Chrystal a développé le premier algorithme pour déterminer ce cercle.

1.1.2 Applications

La résolution du problème du cercle minimum trouve des applications dans divers domaines, notamment :

- Facility Location : Il peut être utilisé pour déterminer l'emplacement optimal d'une installation ou d'une ressource qui dessert plusieurs clients. Par exemple, dans le "problème du bureau de poste", le centre du cercle minimum représente l'emplacement qui minimise la distance moyenne entre les clients et le bureau de poste, considérant que l'effort d'accès est proportionnel à la distance.

- Télécommunications : Il peut être utilisé pour déterminer l'emplacement optimal des relais de radiotéléphonie, minimisant ainsi la puissance d'émission nécessaire pour couvrir des stations d'émission-réception données.

- Planification de Frappes : Il peut être utilisé pour déterminer le point d'impact et le rayon d'action d'une bombe afin de toucher des objectifs spécifiques.

- Critère de Position : Dans le domaine de la statistique et de la recherche opérationnelle, le cercle minimum peut être utilisé comme critère de position pour estimer la dispersion des points expérimentaux ou pour définir le centre de l'hypersphère minimale contenant tous les points.

- Critères de Fatigue : Dans le domaine de la mécanique des matériaux, le cercle minimum peut être utilisé pour analyser le trajet parcouru par le vecteur de cisaillement au cours du temps, permettant ainsi de définir des critères de fatigue tels que la contrainte de cisaillement moyenne.

1.2 Objectifs de l'étude

L'étude entreprise vise à mener une analyse approfondie de l'algorithme Welzl dans la résolution du problème du cercle minimum, en le comparant à un algorithme naïf.

Tout d'abord, une implémentation d'un algorithme naïf sera réalisée. Ce choix méthodologique permettra de calculer le cercle minimum pour un ensemble donné de points dans le plan. Cette approche, bien que simple, servira de référence pour évaluer l'efficacité de l'algorithme de Welzl.

En parallèle, l'algorithme de Welzl sera mis en œuvre conformément à sa description dans la littérature scientifique. Ce dernier est reconnu pour son efficacité accrue dans la résolution du problème du cercle minimum, ce qui en fait un candidat idéal pour une comparaison rigoureuse avec l'algorithme naïf.

Une fois les implémentations terminées, une évaluation des performances sera effectuée. Cette évaluation consistera à confronter les résultats de l'algorithme de Welzl aux instances de la base de test, en mettant un accent particulier sur la précision de sa solution par rapport à l'algorithme naïf. De plus, les temps de calcul respectifs des deux algorithmes seront mesurés pour chaque instance de test.

Enfin, une analyse des résultats sera menée pour mettre en évidence les avantages de l'algorithme de Welzl par rapport à son homologue naïf. Au moins un diagramme à barres sera produit pour illustrer le gain en temps de calcul obtenu avec l'implémentation de l'algorithme de Welzl. Ces diagrammes faciliteront la comparaison des performances relatives des deux approches, fournissant ainsi des insights précieux sur l'efficacité et l'applicabilité pratique de l'algorithme de Welzl dans la résolution du problème du cercle minimum.

2 Méthodologie

2.1 Algorithme Diamètre

L'algorithme Diamètre sert à trouver le diamètre d'un ensemble de points dans un plan cartésien. Il commence par vérifier s'il y a au moins deux points, puis initialise deux points p et q avec les deux premiers points de la liste. Ensuite, il parcourt toutes les paires possibles de points pour trouver la paire avec la plus grande distance. Une fois trouvée, cette paire définit le diamètre, qui est retourné sous forme d'objet de type Line. Bien que cette méthode garantisse la découverte du diamètre, elle peut être inefficace pour de grands ensembles de points en raison de sa complexité quadratique.

Algorithm 1 Algorithme de recherche du diamètre

```
function DIAMETRE(points)
    if size(points) < 2 then
        return null
    end if
    p ← points[0]
    q ← points[1]
    for s in points do
        for t in points do
            if distance(s, t) > distance(p, q) then
                p ← s
                q ← t
            end if
        end for
    end for
    return Line(p, q)
end function
```

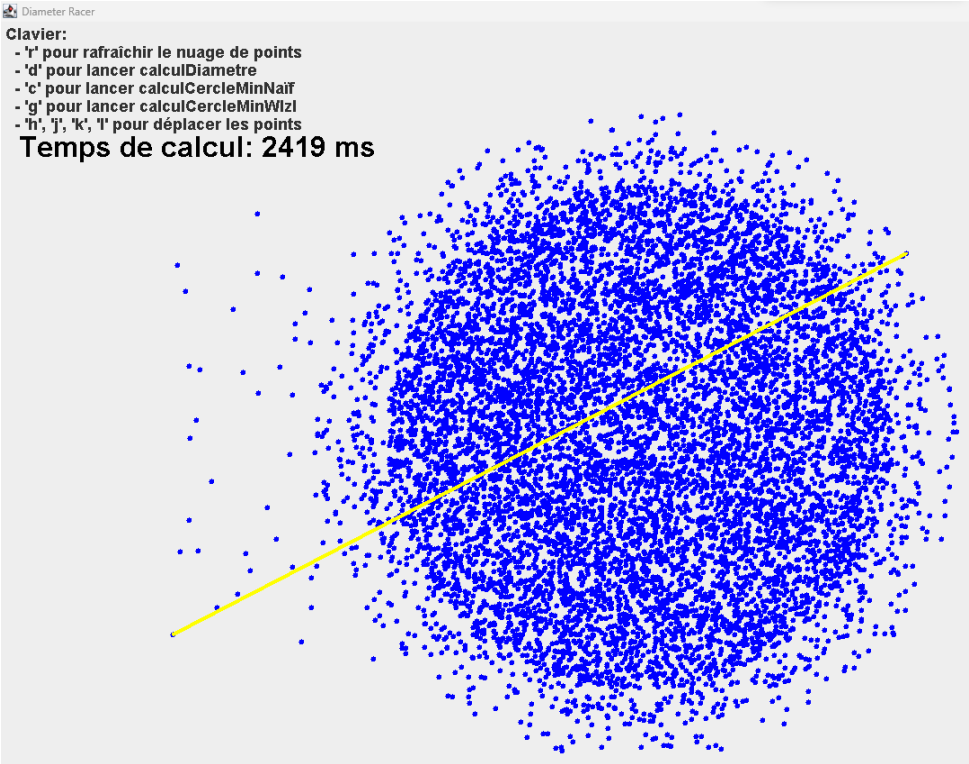

FIGURE 1 – Diamètre calculé à partir d'un ensemble de points

2.2 Algorithme QuickHull

L'algorithme Quick Hull est une méthode de calcul de l'enveloppe convexe d'un ensemble de points dans un espace bidimensionnel. Cette méthode exploite la récursivité pour diviser l'ensemble de points

en sous-ensembles jusqu'à ce que l'enveloppe convexe soit construite. Les points extrêmes sont identifiés comme les points les plus à l'ouest, au sud, à l'est et au nord, puis utilisés comme points de départ pour construire l'enveloppe convexe. La procédure `quickHull` commence par sélectionner les points extrêmes, puis construit récursivement l'enveloppe convexe en éliminant les points à l'intérieur du triangle formé par les points extrêmes. Cette approche garantit que l'enveloppe convexe résultante enveloppe tous les points de l'ensemble d'entrée.

[Algorithm 6 Algorithme QuickHull] en annexe.

2.3 Algorithme Naïf pour calculer le cercle minimum

L'approche naïve pour calculer le cercle minimum englobant (CME) d'un ensemble de points est une méthode itérative qui explore toutes les combinaisons possibles de trois points parmi l'ensemble donné. Cette approche, bien que moins efficace que d'autres algorithmes spécialisés, offre une solution simple et directe au problème.

[Algorithme de recherche du disque minimum (Naïf)] en annexe.

2.3.1 Processus de Calcul

`Sélection de Triplet de Points` : La première étape consiste à sélectionner tous les triplets de points non colinéaires parmi l'ensemble de points donné. Ces triplets représentent les points qui définissent les triangles formés par l'ensemble de points.

`Calcul du Cercle Circonscrit` : Pour chaque triplet de points sélectionné, le centre et le rayon du cercle circonscrit sont calculés. Ces calculs sont basés sur des principes géométriques fondamentaux, notamment la médiane des côtés d'un triangle et la distance entre un point et un cercle.

`Vérification de la Validité du Cercle` : Une fois le cercle circonscrit calculé pour chaque triplet de points, sa validité est vérifiée. Cette vérification implique de s'assurer que tous les points de l'ensemble initial sont contenus à l'intérieur ou sur le cercle circonscrit.

`Identification du Cercle Minimum` : Parmi tous les cercles circonscrits valides, le cercle avec le rayon le plus petit est identifié comme le cercle minimum englobant l'ensemble de points. Cela garantit que le cercle englobe tous les points de manière optimale.

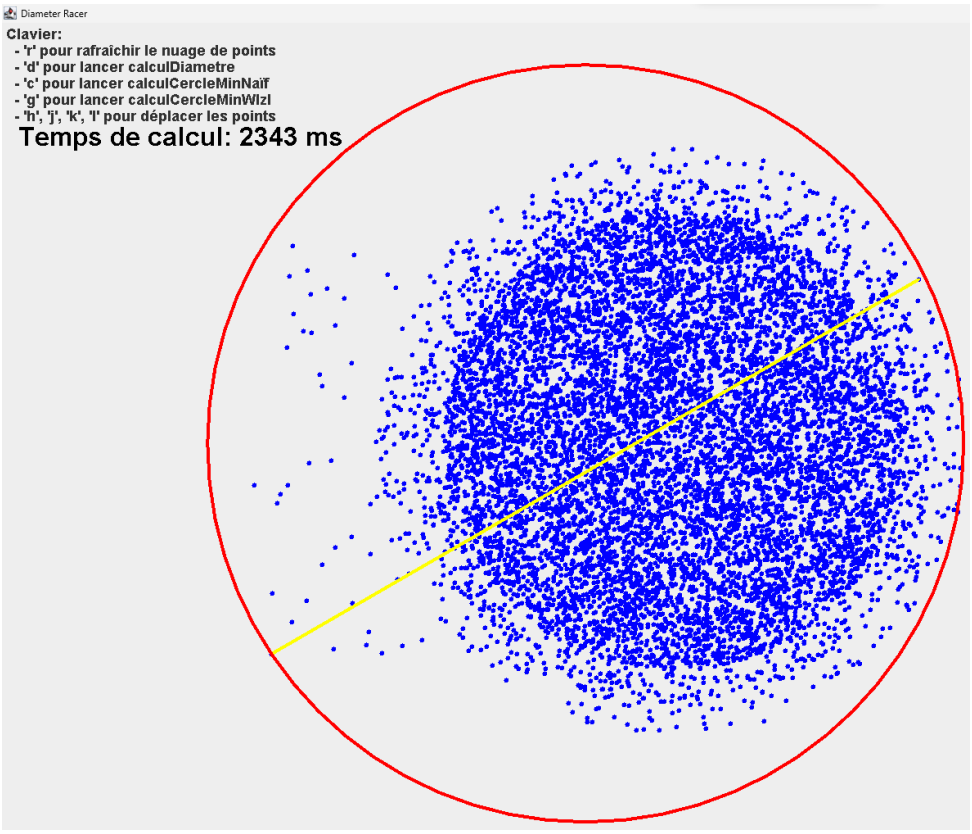

Figure 2 – Approche naïve pour le calcul du Cercle Minimum

2.3.2 Avantages et Limitations

`Simplicité` : L'approche naïve est relativement simple à implémenter et à comprendre, ce qui en fait une option attrayante pour les cas où la complexité n'est pas un problème majeur.

`Exactitude` : Bien que cette méthode puisse être moins efficace que d'autres algorithmes plus spécialisés, elle garantit une solution exacte au problème du CME.

`Coût Algorithmique` : Cependant, cette approche peut devenir coûteuse en termes de temps de calcul, surtout pour des ensembles de points volumineux, en raison de sa nature itérative et de sa complexité exponentielle.

2.4 Algorithme Welzl pour calculer le cercle minimum

L'algorithme de calcul du cercle minimum englobant (CME) par méthode incrémentale, basé sur l'algorithme de Welzl, offre une approche efficace et précise pour trouver le plus petit disque qui englobe un ensemble de points dans le plan. Ce processus est fondé sur des principes géométriques, récursifs et probabilistes, permettant une résolution rapide et optimale du problème. En exploitant ces techniques, il parvient à déterminer rapidement un cercle optimal qui englobe tous les points de l'ensemble, ce qui en fait un choix attrayant pour de nombreuses applications nécessitant une solution rapide et précise en géométrie informatique.

2.4.1 Vue d'Ensemble de l'Algorithme

L'algorithme commence par un ensemble de points P dans le plan. Son objectif est de déterminer le cercle de rayon minimum qui couvre tous les points de P.

`Initialisation` : L'algorithme débute avec un ensemble vide de points.

`Ajout de Points et Mise à Jour du Cercle Minimum` : - Les points de l'ensemble P sont ajoutés un par un. - À chaque ajout, le cercle minimum englobant est mis à jour de manière itérative pour inclure le nouveau point ajouté. - Si le nouveau point est déjà inclus dans le cercle minimum, aucun calcul supplémentaire n'est nécessaire. Sinon, le cercle minimum est recalculé pour inclure le nouveau point.

`Calcul du Cercle Minimum` : Le calcul du cercle minimum est réalisé de manière récursive en utilisant une méthode auxiliaire `bMinDisk`, qui prend en compte les points déjà inclus et le nouveau point ajouté.

`Stratégie de Sélection de Point` : - À chaque étape, un point est choisi aléatoirement parmi les points restants. - Ce point est retiré de l'ensemble des points, et le cercle minimum est recalculé pour inclure les points restants.

`Analyse de Complexité` : - L'algorithme est conçu pour minimiser le nombre d'opérations nécessaires pour calculer le cercle minimum englobant. - Une analyse probabiliste est utilisée pour estimer le nombre d'étapes moyen nécessaires pour calculer le cercle minimum en fonction de la taille de l'ensemble de points.

2.4.2 Avantages et Limitations

`Efficacité` : L'algorithme est conçu pour être très efficace, avec une complexité moyenne linéaire en fonction du nombre de points dans l'ensemble.

`Optimalité` : Le cercle minimum englobant calculé est garanti d'être le plus petit possible tout en couvrant tous les points de l'ensemble.

`Simplicité` : Malgré sa complexité algorithmique, l'algorithme est relativement simple à comprendre et à implémenter grâce à son approche récursive.

`Sensibilité à la Sélection Aléatoire` : Comme l'algorithme utilise une sélection aléatoire de points, sa performance peut varier en fonction des choix aléatoires effectués.

Algorithm 2 Algorithme de recherche du cercle minimum (Welzl)

```
function bMinDisk(Ps, R)
    P ← clone(Ps)
    r ← Random()
    D ← null
    if P is empty or size(R) = 3 then
        D ← bmd(new ArrayList<Point>(), R)
    else
        p ← P[r.nextInt(P.size())]
        P.remove(p)
        D ← bMinDisk(P, R)
        if D ≠ null and !pointInCircle(D, p) then
            R.add(p)
            D ← bMinDisk(P, R)
            R.remove(p)
        end if
    end if
    return D
end function
```

Algorithm 3 Algorithme de calcul du cercle minimum englobant

```
function bmd(P, R)
    if P is empty and size(R) = 0 then
        return Circle(Point(0, 0), 10)
    end if
    r ← Random()
    D ← null
    if size(R) = 1 then
        D ← Circle(R[0], 0)
    end if
    if size(R) = 2 then
        cx ← (R[0].x + R[1].x) / 2
        cy ← (R[0].y + R[1].y) / 2
        d ← distance(R[0], R[1]) / 2
        p ← Point(round(cx), round(cy))
        D ← Circle(p, ⌈d⌉)
    end if
    if size(R) = 3 then
        D ← calculateCircumcircle(R[0], R[1], R[2])
    end if
    return D
end function
```

Algorithm 4 Algorithme de vérification d'appartenance d'un point à un cercle

```
function pointInCircle(c, p)
    distance ← distance(p, c.getCenter())
    result ← distance − c.getRadius()
    if result < 0.00001 then
        return true
    else
        return false
    end if
end function
```

Algorithm 5 Algorithme de calcul du cercle circonscrit

function CALCULATECIRCUMCIRCLE(a, b, c)
  $d \leftarrow 2 \times ((a.x \times (b.y - c.y)) + (b.x \times (c.y - a.y)) + (c.x \times (a.y - b.y)))$
  if $d = 0$ **then**
    return null
  end if
  $x \leftarrow ((((a.x \times a.x) + (a.y \times a.y)) \times (b.y - c.y)) + (((b.x \times b.x) + (b.y \times b.y)) \times (c.y - a.y)) + (((c.x \times c.x) + (c.y \times c.y)) \times (a.y - b.y)))/d$
  $y \leftarrow ((((a.x \times a.x) + (a.y \times a.y)) \times (c.x - b.x)) + (((b.x \times b.x) + (b.y \times b.y)) \times (a.x - c.x)) + (((c.x \times c.x) + (c.y \times c.y)) \times (b.x - a.x)))/d$
  $p \leftarrow \text{Point}(\text{round}(x), \text{round}(y))$
  $radius \leftarrow \lceil \text{distance}(p, a) \rceil$
  return $\text{Circle}(p, radius)$
end function

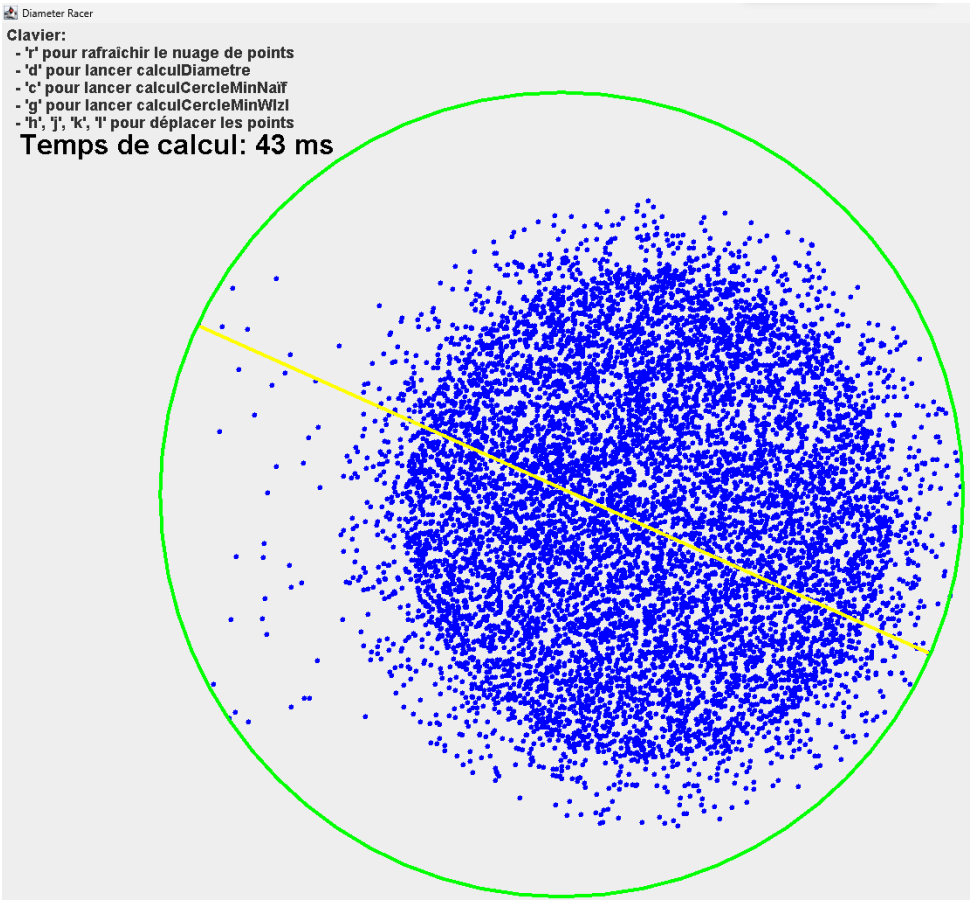

FIGURE 3 – Approche efficace (Welzl) pour le calcul du Cercle Minimum

2.5 Acquisition et utilisation de la base de test

Pour garantir des résultats fiables et une évaluation rigoureuse des performances des algorithmes implémentés, j'ai utilisé la base de test VAROUMAS. Cette base de test répond aux critères spécifiés dans le cadre du projet, notamment en termes de taille minimale d'instances et de nombre de points dans chaque instance.

Pour exploiter cette base de test, j'ai développé un module de benchmarking. Ce module se compose de plusieurs fonctionnalités essentielles pour l'acquisition des données de performance des algorithmes étudiés.

`Liste des fichiers dans le répertoire :` Le module commence par scanner le répertoire contenant les fichiers d'instances de test. À l'aide de la fonction *listeRepertoire*, il récupère la liste des fichiers disponibles dans ce répertoire.

`Lecture des fichiers d'instances` : Ensuite, le module lit chaque fichier d'instance en utilisant la fonction *readFile*. Cette fonction parcourt chaque fichier ligne par ligne, extrait les coordonnées des points et les stocke dans une structure de données appropriée. Dans ce cas, les coordonnées des points sont stockées dans une liste d'objets *Point*.

`Benchmarking des algorithmes` : Une fois les données d'instance obtenues, le module évalue les performances des algorithmes implémentés. Pour chaque fichier d'instance, il mesure le temps nécessaire à l'exécution des deux algorithmes : l'algorithme naïf (fonction *algoCalculCercleMinNaïf*) et l'algorithme de Welzl (fonction *algoCalculCercleMinWelzl*). Les temps d'exécution sont enregistrés et écrits dans un fichier CSV pour une analyse ultérieure.

`Analyse des performances` : Enfin, le module affiche les résultats des performances des algorithmes sur la sortie standard et les écrit également dans un fichier CSV nommé *"benchmark.csv"*. Ce fichier contient les temps d'exécution des deux algorithmes pour chaque instance de test, facilitant ainsi l'analyse comparative des performances.

3 Résultats et Discussion

3.1 Performances sur la base de test préparée

3.1.1 Algorithme Naïf pour calculer le cercle minimum

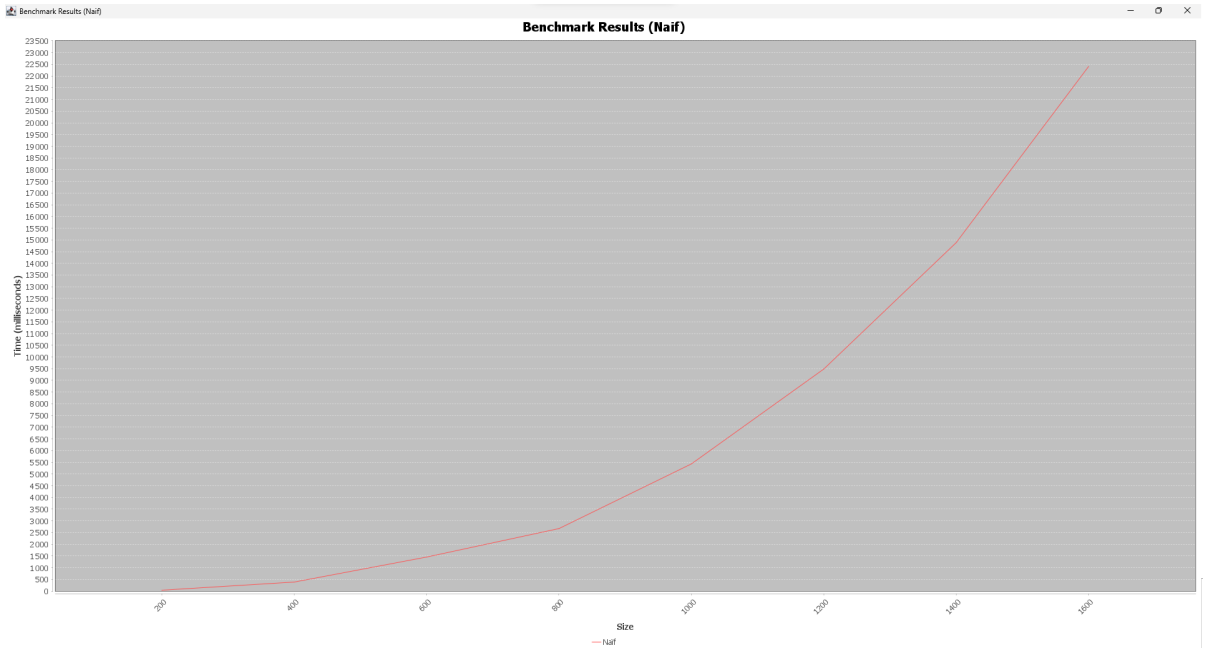

FIGURE 4 – Comparaison des temps d'exécution par rapport au nombre de points : algorithme Naïf

Dans le cadre de l'analyse de l'algorithme Naïf, il est essentiel d'évaluer sa complexité pour comprendre son efficacité et sa scalabilité. La complexité algorithmique est un aspect fondamental en informatique permettant de quantifier les ressources nécessaires à l'exécution d'un algorithme en fonction de la taille de l'entrée.

L'algorithme est constitué de deux parties principales. Dans la première partie, deux boucles imbriquées parcourent tous les points de l'entrée, avec une troisième boucle interne effectuant des opérations arithmétiques simples pour chaque paire de points. Cette structure engendre une complexité quadratique de l'ordre de $\mathcal{O}(n^2)$, où n représente le nombre de points dans l'entrée.

Quant à la seconde partie de l'algorithme, elle consiste en trois boucles imbriquées parcourant toutes les combinaisons de trois points parmi ceux de l'entrée. Cette structure génère une complexité cubique de l'ordre de $\mathcal{O}(n^3)$, en raison du nombre exponentiel de combinaisons à considérer.

En combinant ces deux parties, l'algorithme présente une complexité globale de $\mathcal{O}(n^3)$, ce qui signifie que le temps d'exécution augmente de manière significative avec l'augmentation du nombre de points. Cette complexité, bien que polynomiale, peut devenir prohibitivement coûteuse pour des ensembles de données volumineux.

3.1.2 Algorithme Welzl pour calculer le cercle minimum

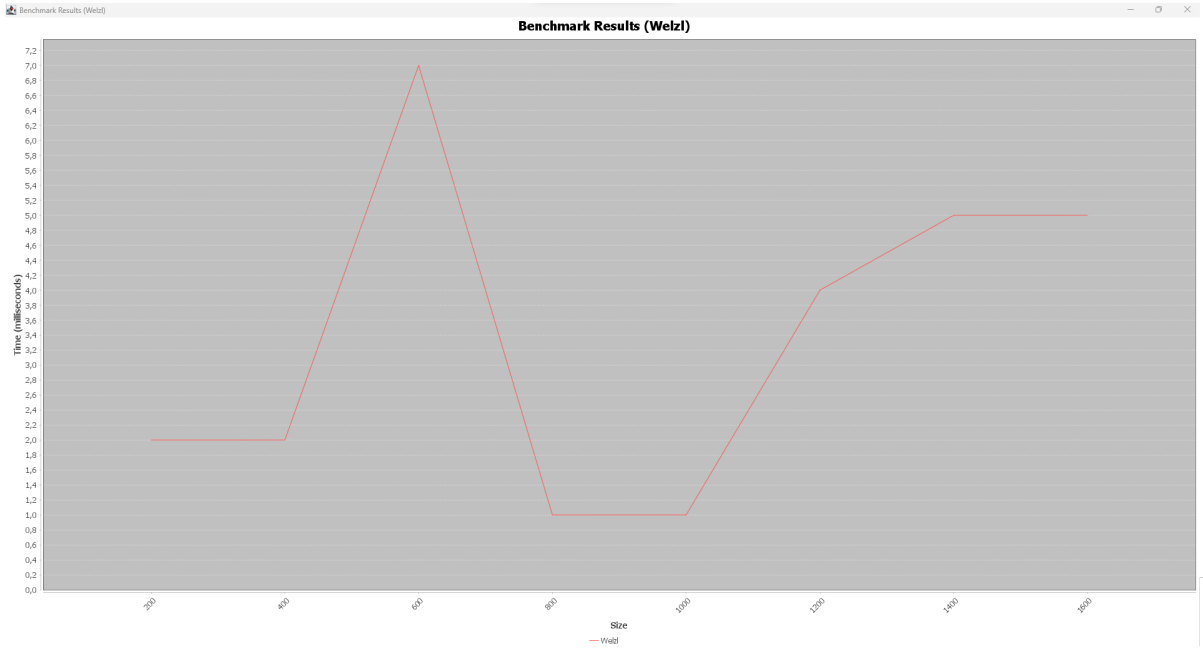

FIGURE 5 – Comparaison des temps d'exécution par rapport au nombre de points : algorithme Welzl

L'algorithme utilise une approche de type "diviser pour régner" pour construire le cercle minimum progressivement. Il commence par sélectionner de manière aléatoire un point de l'ensemble donné, puis le retire de l'ensemble initial. Ensuite, il récursivement tente de construire le cercle minimum englobant l'ensemble de points restants. Cette étape récursive est répétée jusqu'à ce que le cercle minimum soit obtenu.

Dans le pire des cas, l'algorithme effectue une série de sélections aléatoires et de vérifications de points pour chaque récursion, jusqu'à ce que le cercle minimum soit trouvé. Cette étape récursive peut potentiellement parcourir toutes les combinaisons possibles de points.

Le temps d'exécution dépend donc fortement du nombre de récursions nécessaires pour former le cercle minimum.

Malgré sa nature récursive, l'algorithme de Welzl a une complexité moyenne linéaire. Cela signifie que dans la plupart des cas, il effectue un nombre limité d'opérations pour chaque point de l'entrée, ce qui le rend plus efficace que l'algorithme naïf.

3.2 Comparaison des performances entre l'algorithme Naïf et Welzl

En résumé, comparé à l'algorithme naïf, l'algorithme de Welzl offre une meilleure efficacité en termes de temps d'exécution pour la construction du cercle minimum englobant un ensemble de points. Cette efficacité découle de sa complexité moyenne linéaire, qui le rend plus adapté pour le traitement d'ensembles de données volumineux, contrairement à l'algorithme naïf dont la complexité cubique peut devenir prohibitivement coûteuse.

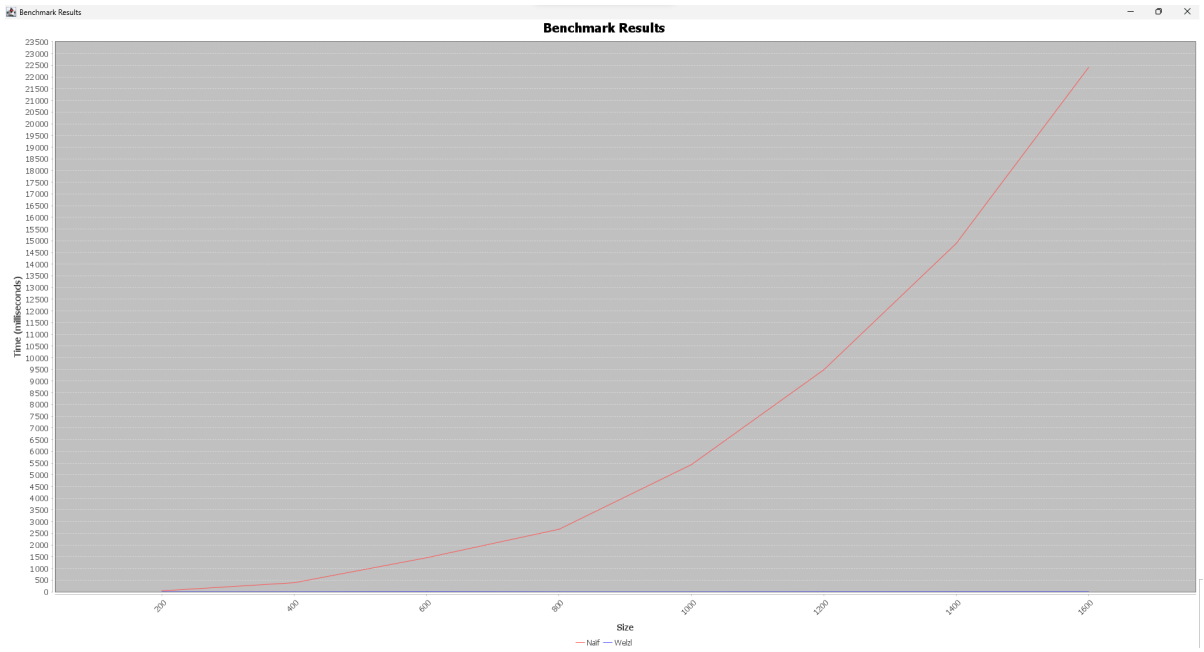

FIGURE 6 – Comparaison des temps d'exécution par rapport au nombre de points : algorithme Naïf (rouge) vs. algorithme de Welzl (bleu)

Après avoir effectués 1664, les résultat suivant on été obtenu :

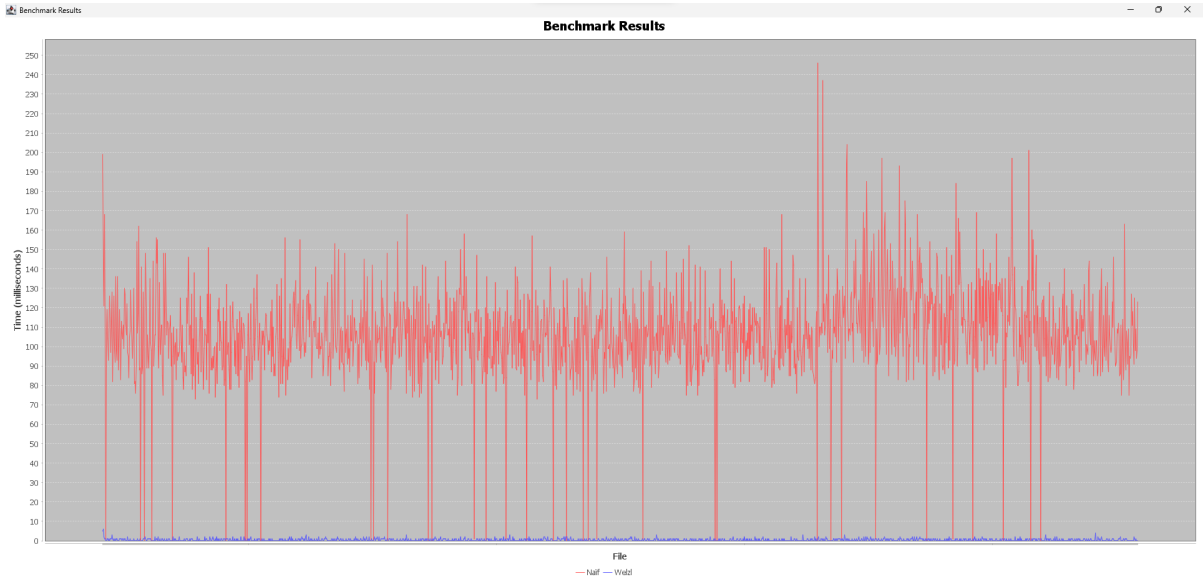

FIGURE 7 – Benchmark : algorithme Naïf (rouge) vs. algorithme de Welzl (bleu)

L'algorithme naïf affiche des performances nettement inférieures à celles de l'algorithme de Welzl. En moyenne, l'algorithme naïf nécessite environ 106.21 millisecondes pour s'exécuter, tandis que l'algorithme de Welzl ne nécessite que près de 0.45 milliseconde. Ces résultats mettent en évidence une différence significative de temps d'exécution entre les deux algorithmes.

Un gain de temps conséquent peut se voir sur ce diagramme :

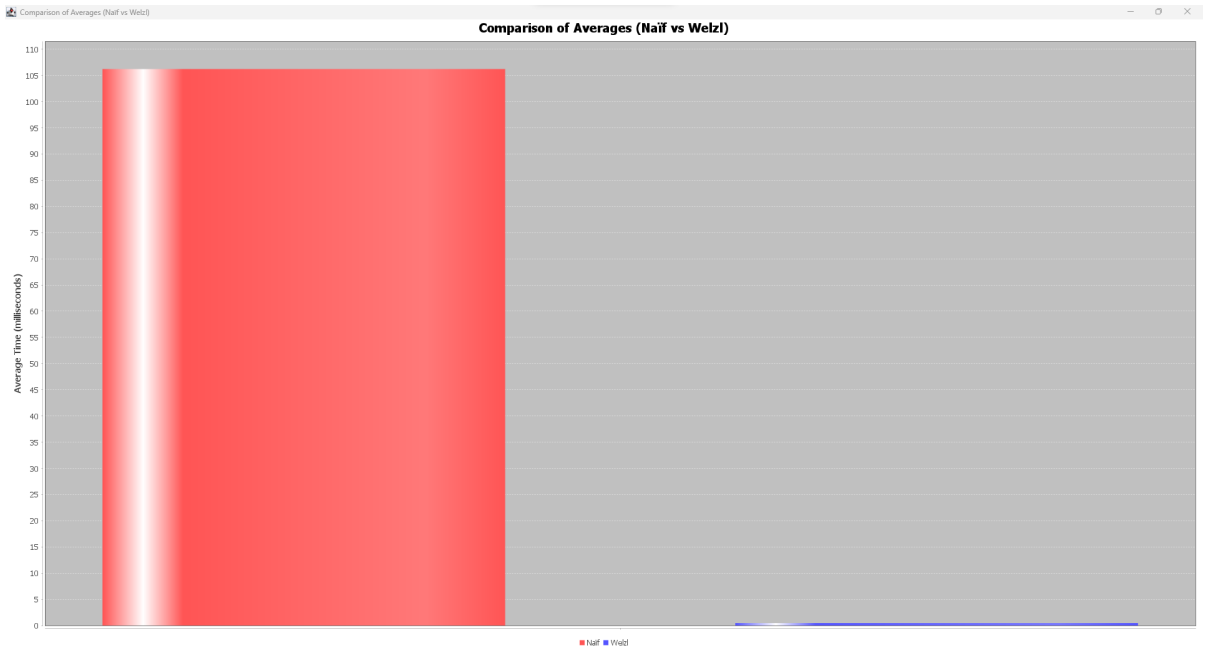

FIGURE 8 – Comparaison des temps moyens : algorithme Naïf (rouge) vs. algorithme de Welzl (bleu)

4 Conclusion

En somme, après approfondi ma compréhension de l'algorithme de Welzl pour résoudre le problème du cercle minimum, en démontrant sa supériorité par rapport à l'approche naïve à travers une série d'analyses méthodiques et d'expérimentations pratiques. Les résultats ont clairement souligné l'importance de développer des solutions efficaces pour ce problème, compte tenu de ses nombreuses applications dans des domaines variés.

L'algorithme de Welzl s'est distingué par sa capacité à offrir des performances nettement meilleures en termes de temps d'exécution, ce qui en fait une option privilégiée pour les ensembles de données de taille importante. De plus, sa simplicité relative de compréhension et d'implémentation renforce son attrait pour des applications pratiques.

5 Bonus : Notions similaires en 3D

Les concepts similaires en 3D, tels qu'un problème de sphère englobante minimale, présentent des défis supplémentaires en raison de la complexité spatiale accrue. Dans le cas du problème du cercle minimum en 2D, les points sont définis par leurs coordonnées (x, y), tandis qu'en 3D, les points sont définis par leurs coordonnées (x, y, z).

L'adaptation de l'algorithme de Welzl pour résoudre le problème de la sphère englobante minimale en 3D implique des ajustements et des considérations supplémentaires liés à la dimension supplémentaire. Les méthodes algorithmiques doivent prendre en compte la géométrie tridimensionnelle et la nature des données pour déterminer efficacement la sphère englobante minimale.

L'extension du test 2D pour déterminer si un point est à l'intérieur d'une sphère en 3D est assez directe. Si les coordonnées du centre de la sphère sont (cx, cy, cz) et que son rayon est r, alors le point (x, y, z) est à l'intérieur de la sphère si et seulement si la distance euclidienne entre ce point et le centre de la sphère est inférieure au rayon de la sphère, soit :

$$\sqrt{(x - cx)^2 + (y - cy)^2 + (z - cz)^2} < r$$

Cette expression permet de déterminer de manière précise si un point donné est contenu à l'intérieur de la sphère en considérant ses coordonnées en trois dimensions. Elle généralise efficacement le test 2D pour le cas tridimensionnel en incluant la troisième dimension, Z, dans le calcul de la distance euclidienne.

Références

[1] *«Projet individuel : rédaction d'un rapport de recherche sur l'algorithme Welzl résolvant le problème du cercle minimum.»*. BM Bui-Xuan. https://www-npa.lip6.fr/~buixuan/files/cpa2023/petitprojetCPA.pdf.

[2] *«Smallest enclosing disks (balls and ellipsoids).»*. Emo Welzl. https://www.stsci.edu/~RAB/Backup%20Oct%2022%202011/f_3_CalculationForWFIRSTML/Bob1.pdf.

[3] *«Computing the Diameter of a Point Set.»*. Grégoire Malandain, Jean-Daniel Boissonnat. https://inria.hal.science/inria-00072354/PDF/RR-4233.pdf.

[4] *«Collisions simples et probleme du cercle minimum.»*. BM Bui-Xuant. https://www-npa.lip6.fr/~buixuan/files/cpa2023/cpa2023_tme1.pdf.

[5] *«Documentation Java : Collisions simples et probleme du cercle minimum..»*. BM Bui-Xuant. https://www-npa.lip6.fr/~buixuan/files/cpa2023/corrigeTME1.java.

[6] *«Enveloppe convexe.»*. BM Bui-Xuant. https://www-npa.lip6.fr/~buixuan/files/cpa2023/cpa2023_tme2.pdf.

[7] *«Documentation Java : Enveloppe convexe.»*. BM Bui-Xuant. https://www-npa.lip6.fr/~buixuan/files/cpa2023/corrigeTME2.java.

[8] *«Problème du cercle minimum.»*. Wikipédia, 8 octobre 2023. https://fr.wikipedia.org/w/index.php?title=Probl%C3%A8me_du_cercle_minimum&oldid=208545755.

[9] *«Steinmetz Solid.»*. Wikipedia, 4 février 2024. https://en.wikipedia.org/w/index.php?title=Steinmetz_solid&oldid=1203116866.

Algorithm 6 Algorithme QuickHull

```
function QuickHull(points)
    if size(points) < 4 then
        return points
    end if
    west ← points[0]
    south ← points[0]
    east ← points[0]
    north ← points[0]
    for each point in points do
        if point.x < west.x then
            west ← point
        end if
        if point.y > south.y then
            south ← point
        end if
        if point.x > east.x then
            east ← point
        end if
        if point.y < north.y then
            north ← point
        end if
    end for
    result ← [west, south, east, north]
    rest ← points.clone()
    for i ← 0 to size(rest) do
        if triangleContainsPoint(west, south, east, rest[i]) or triangleContainsPoint(west, east, north, rest[i])
then
            rest.remove(i)
            i ← i − 1
        end if
    end for
    i ← 0
    while i < size(result) do
        a ← result[i]
        b ← result[(i + 1)  mod size(result)]
        ref ← result[(i + 2)  mod size(result)]
        signRef ← crossProduct(a, b, a, ref)
        maxValue ← 0
        maxPoint ← a
        for each point in points do
            piki ← crossProduct(a, b, a, point)
            if signRef × piki < 0 and |piki| > maxValue then
                maxValue ← |piki|
                maxPoint ← point
            end if
        end for
        if maxValue ≠ 0 then
            for j ← 0 to size(rest) do
                if triangleContainsPoint(a, b, maxPoint, rest[j]) then
                    rest.remove(j)
                    j ← j − 1
                end if
            end for
            result.insert(i + 1, maxPoint)
            i ← i − 1
        end if
        i ← i + 1
    end while
    return result
end function
```

Algorithm 7 Algorithme de recherche du disque minimum (Naïf)

function AlgoCalculCercleMinNaïf($inputPoints$)
  $points \leftarrow$ clone($inputPoints$)
  if size($points$) < 1 **then**
    return null
  end if
  $cX, cY, cRadius, cRadiusSquared \leftarrow 0$
  for each p in $points$ **do**
    for each q in $points$ **do**
      $cX \leftarrow 0.5 \times (p.x + q.x)$
      $cY \leftarrow 0.5 \times (p.y + q.y)$
      $cRadiusSquared \leftarrow 0.25 \times ((p.x - q.x)^2 + (p.y - q.y)^2)$
      $allHit \leftarrow$ true
      for each s in $points$ **do**
        if $((s.x - cX)^2 + (s.y - cY)^2) > cRadiusSquared$ **then**
          $allHit \leftarrow$ false
          break
        end if
      end for
      if allHit **then**
        return Circle(Point(int(cX), int(cY)), int($\sqrt{cRadiusSquared}$))
      end if
    end for
  end for
  $resX, resY, resRadiusSquared \leftarrow 0$
  for $i \leftarrow 0$ **to** size($points$) **do**
    for $j \leftarrow i + 1$ **to** size($points$) **do**
      for $k \leftarrow j + 1$ **to** size($points$) **do**
        $p \leftarrow points[i]$
        $q \leftarrow points[j]$
        $r \leftarrow points[k]$
        if $(q.x - p.x) \times (r.y - p.y) - (q.y - p.y) \times (r.x - p.x) == 0$ **then**
          continue ▷ les points sont colinéaires
        end if
        if $(p.y == q.y)$ or $(p.y == r.y)$ **then**
          if $p.y == q.y$ **then**
            $p \leftarrow points[k]$
            $r \leftarrow points[i]$
          else
            $p \leftarrow points[j]$
            $q \leftarrow points[i]$
          end if
        end if
        $mX \leftarrow 0.5 \times (p.x + q.x)$
        $mY \leftarrow 0.5 \times (p.y + q.y)$
        $nX \leftarrow 0.5 \times (p.x + r.x)$
        $nY \leftarrow 0.5 \times (p.y + r.y)$
        $\alpha 1 \leftarrow \frac{q.x - p.x}{p.y - q.y}$
        $\beta 1 \leftarrow mY - \alpha 1 \times mX$
        $\alpha 2 \leftarrow \frac{r.x - p.x}{p.y - r.y}$
        $\beta 2 \leftarrow nY - \alpha 2 \times nX$
        $cX \leftarrow \frac{\beta 2 - \beta 1}{\alpha 1 - \alpha 2}$
        $cY \leftarrow \alpha 1 \times cX + \beta 1$
        $cRadiusSquared \leftarrow (p.x - cX)^2 + (p.y - cY)^2$
        if $cRadiusSquared \geq resRadiusSquared$ **then**
          continue
        end if

$allHit \leftarrow$ true
for each s in $points$ **do**
    if $((s.x - cX)^2 + (s.y - cY)^2) > cRadiusSquared$ **then**
        $allHit \leftarrow$ false
        break
    end if
end for
if allHit **then**
    $resX \leftarrow cX$
    $resY \leftarrow cY$
    $resRadiusSquared \leftarrow cRadiusSquared$
end if

return Circle(Point(int($resX$), int($resY$)), int($\sqrt{resRadiusSquared}$))
=0