Rapport Projet De Mathématiques

Enveloppe Convexe

Clément BOISSARD
Imad BOUFENCHOUCHE

3e année ITC
Semestre 1, année 2017 *ESIREM*

Table des matières

Introduction

Durant ce premier semestre du cycle, de 3 ans, d'ingénieur informatique nous avons un projet de mathématiques à réaliser. Le projet a pour objectif d'afficher l'enveloppe convexe d'un regroupement de points dans un plan euclidien. Pour se faire il faudra programmer un algorithme, avec le langage C ou C++, affichant uniquement les points de l'enveloppe convexe finale graphiquement.

Le programme devra être géré depuis un terminal pour l'éxecution et la compilation, avec l'utilisation d'un Makefile pour simplifier et définir les opérations qu'il faut effectuer pour utiliser le programme.

Pour la réalisation de l'application il faudra utiliser la librairie graphique "OpenGL" afin d'afficher les points et l'enveloppe convexe en deux dimensions sur la fenêtre de l'application.

Quelques éléments du programme, réalisé en C, sont à notre disposition pour comprendre le fonctionnement de la librairie et comment l'utiliser dans notre cas, c'est-à-dire l'affichage d'un ensemble de points définis par leurs coordonnées `x` et `y` dans un plan en deux dimensions. Il y a aussi des fonctions pour la gestion des actions de l'utilisateur avec l'application.

Avant de commencer à développer notre application, il nous faut comprendre ce qu'est une enveloppe convexe et comment il est possible d'en obtenir une ayant un périmètre minimum depuis un ensemble fini de points.

De nombreux mathématiciens se sont déjà penchés sur la question, il faut donc trouvé et comprendre des solutions algorithmiques possible pour se problème d'enveloppe convexe. Pour cela nous nous contenterons de solutions au problème dans un plan.

Chapitre 1

L'enveloppe convexe

1.1 Description

Une enveloppe convexe d'un ensemble de points est le plus petit polygone formé avec des points de l'ensemble tel que tout point de l'ensemble appartient, soit au polygone soit à l'intérieur de ce polygone qui est l'enveloppe convexe.

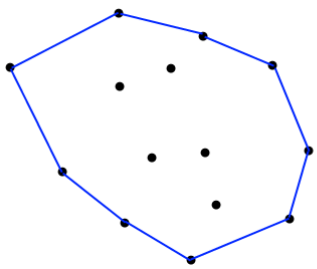

FIGURE 1.1 – Enveloppe convexe

1.2 Analogie

Dans un plan l'enveloppe complexe est souvent comparée au phénomène et à la forme que prend un élastique lorsqu'il se contracte en englobant l'ensemble des points représentés par des clous sur une planche dans cette vue de l'esprit du "fonctionnement" de l'enveloppe convexe.

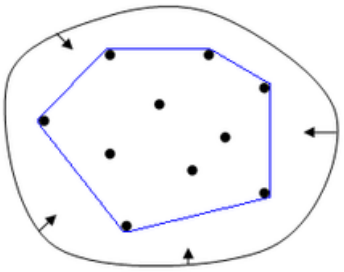

FIGURE 1.2 – Phénomène enveloppe convexe
Source : Wikipédia

1.3 Explication

L'explication plus mathématiques et théorique de l'enveloppe convexe.
Soit E un espace vectoriel de dimensions finies sur $\mathbb{R}$ et $x, y \in E$. On appele *segment* de E, un ensemble noté et défini comme suit :

$$[x, y] := (1 - t)x + ty : t \in [0, 1]$$

Lorsque $x \neq y$, on définit les segments $[x, y[$, $]x, y]$ et $]x, y[$, en remplaçant dans la formule ci-dessus, l'intervalle $[0, 1]$ respectivement par les intervalles $[0, 1[$, $]0, 1]$ et $]0, 1[$. Lorsque $x = y$, les segments $[x, y[$, $]x, y]$ et $]x, y[$ sont vides, par définition. On dit qu'une partie C de E est convexe si pour tout $x, y \in C$, le segment $[x, y]$ est contenu dans C. On dit aussi que C est "un convexe". La figure 1.3 illustre cette notion.

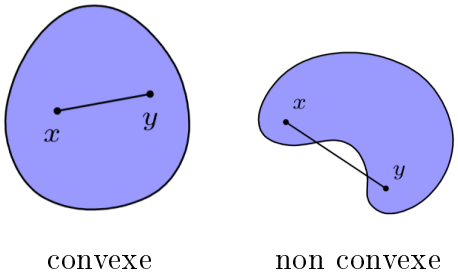

FIGURE 1.3 – Définition d'un ensemble convexe

Chapitre 2

Différents algorithmes

Il existe de nombreux algorithmes pour permettre de trouver l'enveloppe convexe d'un ensemble de points.

Nous allons voir les plus connus.

2.1 Marche de Jarvis

La marche de Jarvis est un algorithme dont le principe et de sélectionner un premier point appartenant à l'enveloppe, pour cela il suffit de prendre le point le plus à droite par exemple ou le plus en bas etc... Puis de ce premier point p_0 il suffit de parcourir l'ensemble des points en selectionnant le point p_1 avec lequel le point p_0 aura un angle maximal. Il suffit de repéter successivement depuis le point p_i pour trouver un point différent de p_{i-1} avec lequel l'angle est maximal. Il faut répéter l'opération jusqu'à retomber sur le point de départ p_0.

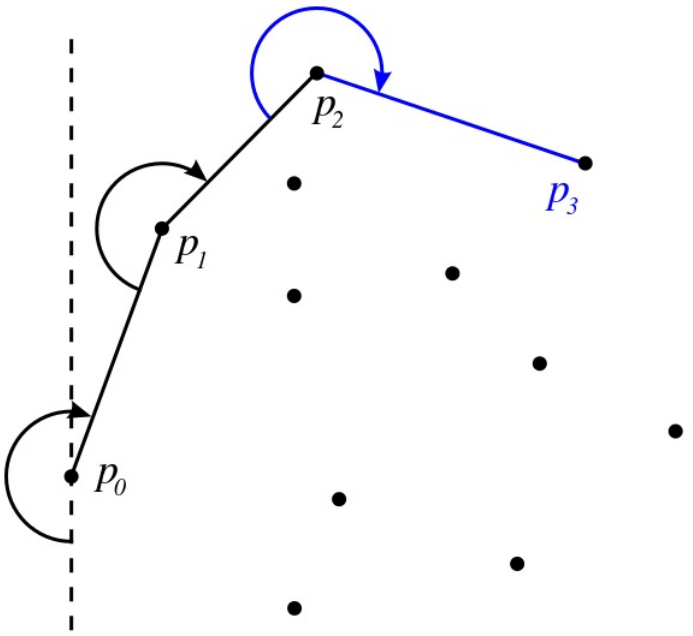

FIGURE 2.1 – Marche de Jarvis
Source : Wikipédia

Algorithme 2.1 – Marche de Jarvis

```
1  fonction marcheJarvis(E)
2     p := p
3     pour tout p' de E
4        Si p = p ou p' est à gauche de [pp) alors
5           p = p'
6     p := p
7     L := liste vide
8     répéter
9        insérer p dans L
10       p := point p' de E tel que [pp') est le plus incliné vers la
             gauche (1)
11    jusqu'à p = p0
12    retourner L
```

La boucle est répétée autant de fois qu'il y a de points dans l'ensemble E.
Soit n le nombre de points de E et m le nombre de point de l'enveloppe convexe, l'algorithme a donc une complexité égale à $n * m$: $O(nm)$

2.2 Algorithme de Graham

L'algorithme de Graham, ou parcours de Graham, est un autre algorithme pour trouver l'enveloppe convexe d'un ensemble E de points p_i distinct.

Tout d'abord il faut définir p_0, le point le plus bas parmis les points de l'ensemble E, c'est-à-dire le point ayant la valeur de y la plus petite, et ayant la valeur de x la plus petite si il y a plusieurs point ayant la valeur minimale y.
Puis il faut trier les points $E - p_0$, par taille de l'angles entre la droite passant par le point p_i et le point p_0 avec la droite des abscisses.
De plus p_0 fait forcement partie de l'enveloppe convexe, car est la plus extreme dans une direction.

Ensuite il suffit de construire l'enveloppe convexe en parcourant le tableau et en testant suivant 2 cas :

1. Si l'angle $p_{i-1}p_ip_{i+1}$ est supérieur ou égal à 180 $^\circ$, alors p_i appartient aux points de l'enveloppe convexe. On incrémente i de 1.
2. Si l'angle $p_{i-1}p_ip_{i+1}$ est inférieur strictement à 180 $^\circ$, alors p_i n'appartient pas à l'enveloppe convexe, et il faut le retiré du tableau trié, donc p_{i+1} est maintenant p_i. Dans ce cas il ne faut pas incrémenter i afin de réeffectuer le test.

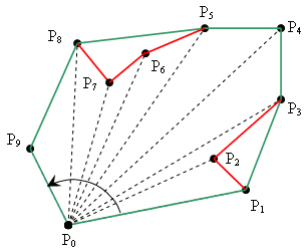

FIGURE 2.2 – Parcours de Graham
Source numéro 3.2.3

Il ne restera à la fin dans le tableau de départ seulement les points de l'enveloppe convexe.

A la différence de la "Marche de Jarvis" (section 2.1), l'"algorithme de Graham" a une complexité algorithmique bien moindre lorsque le nombre de points de l'ensemble devient très important. Sa complexité est en $O(n * log(n))$ où n est le nombre de point de l'ensemble.

2.3 Quickhull

L'algorithme du Quickhull est un algorithme fonctionnant avec le même principe que le "Quicksort" (ou "tri rapide" en français), c'est-à-dire que l'algorithme place un pivot afin de diviser l'ensemble en 2 sous-ensembles et pour chacun des sous-ensemble de trouver le point le plus éloigner du pivot pour en faire le pivot du sous-ensemble est le diviser en 2 sous-sous-ensembles, et répéter cette opération.

Plus précisement, l'algorithme commence par chercher les 2 points les plus éloignés, soient p_1 et p_2 ces points, la droite passant par p_1 et p_2 partage l'ensemble en 2 sous-ensembles.
Dans chacun de ces 2 sous-ensembles, il faut cherche le point le plus éloigné de la droite (p_1p_2), soit p_3 ce point et soit h_{1-2} le point d'intersection entre la droite (p_1p_2) et la droite perpenticulaire à la précédente passant par p_3.
Soient 2 ensembles E_1 et E_2, respectivement l'ensemble des points à l'extérieur du triangle p_1 p_3 et h_{1-2} et les points à l'extérieur du triangle p_2 p_3 et h_{1-2}.
Parmis ces 2 ensembles E_1 et E_2 il faut trouvé respectivement le point le plus éloigné de la droite (p_1h_{1-2}) et de la droite (p_2h_{1-2}).
Et recommencer tant que les ensembles à l'extérieur des nouveaux triangles sont non-vides.
Chacun des points p_1, p_2, p_3, p_n appartient à l'enveloppe convexe de l'ensemble.

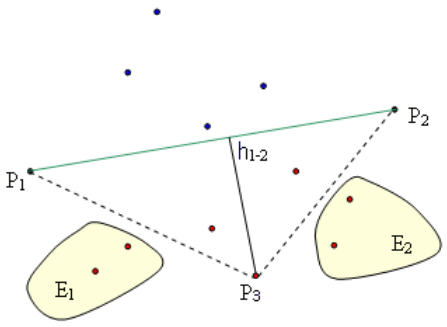

FIGURE 2.3 – Algorithme Quickhull
Source numéro 3.2.3

Voici le pseudo-code le plus intéressant et le plus compréhensible trouvée :

Algorithme 2.2 – Quick Hull Source numéro 3.2.3

```
1  Input = a set S of n points
2
3  Assume that there are at least 2 points in the input set S of points
4  QuickHull (S) {
5        // Trouver l'enveloppe convexe depuis l'ensemble de n points
6        Convex Hull := {}
7        Trouver le plus à gauche et le plus à droite, nommé A & B, et
             ajouter A & B à l'enveloppe
8        Le segment AB divises les n-2 points en 2 groupes S1 et S2
9              où S1 est l'ensemble des points de S à droite du segment
                   A vers B,
10             et S2 est l'ensemble des points de S à droite du segment
                   B vers A
11       FindHull (S1, A, B)
12       FindHull (S2, B, A)
13 }
14
15 FindHull (Sk, P, Q) {
16    // Trouver les points de l'enveloppe depuis l'ensemble de points
         Sk
17    // Ils sont sur le côté droit du segment P vers Q
18    Si Sk n'a pas de point
19    Alors return.
20
21    Depuis l'ensemble Sk, le point le plus loin, nommé C, du segment
         [PQ]
22    Ajouter le point C à l'enveloppe entre P et Q
23    3 points P, Q, et C divise les points de Sk en 3 sous-ensembles:
         S0, S1, et S2
24         où S0 sont les points dans le triangle PCQ,
25         S1 sont les points sur la droite du segement P vers C,
26         et S2 les points sur la droite du segement de C vers Q.
27     FindHull(S1, P, C)
28     FindHull(S2, C, Q)
29 }
30
31 Output = Convex Hull
```

Dans le meilleur des cas l'algorithme du "Quickhull" a une complexité de l'ordre de $O(n*log(n))$ tout comme pour le "Parcours de Graham" (section 2.2). Mais dans le pire des cas l'algorithme peut avoir une complexité quadratique en $O(n^2)$.

Chapitre 3

Le développement

Le développement de l'application a été réalisé avec le langage C++, le développement se divise en plusieurs parties :

1. Premièrement la création d'une liste de points qui constituera le nuage de points.
2. Deuxièment gérer l'affichage des points.
3. Troisièmement programmer les algorithmes de calcul de l'enveloppe convexe
4. Quatrièmement afficher l'enveloppe convexe fraichement calculée.

Pour la réalisation de la partie graphique, nous devions utiliser "OpenGL" qui est une interface de programmation, une bibliothèque de fonctions graphique. Pour utiliser OpenGL nous utilisons "GLUT" qui permet de gérer les fenêtres OpenGL et les interactions avec le système d'exploitation pour la gestion du clavier, de la souris, etc...

Quant à la compilation du code, il fallait que ça soit fait grâce à un makefile pour simplifier la compilation et autres commandes pratiques par l'utilisateur.

3.1 Liste des points

3.1.1 Création

La première étape fut la réalisation du code permettant de créer la liste de points qui vont constituer le nuage de points sur lequel réaliser son enveloppe convexe.
Pour cela j'ai mis en place 3 manières différentes.
Il est possible de créer la liste de points depuis une liste prédéfinie dans un fichier.
Il est aussi possible de créer une liste de n points aléatoirements compris dans une fourchette de 2 entiers.
La dernière possibilité est d'ajouter les points manuellement en entrant les coordonnées les uns à la suite des autres.

La liste de point est représenté par un vector, qui est un container représentant un tableau plus simple à utiliser qu'un tableau avec ses fonctions prédéfinies. Ce vector ou tableau, est composé d'objets, instances de la classe "Point". Un objet avec un membre pour sa coordonnée x et y.

3.1.2 Affichage

Une fois que nous avons une liste de points, il est nécessaire de l'afficher, sinon l'application n'a pas d'intérêt. Grâce à OpenGL il est possible d'afficher des éléments graphiques tels que des points, des segments ou même des polygônes, que ce soit en deux ou en trois dimensions.

Pour commencer on cherche à afficher des points sur un plan en deux dimensions.

Dans la partie du code qui nous a été fourni, il y a une suite d'instruction OpenGL qui permettent dafficher des points. Par exemple pour afficher un point de coordonnées $\{1; 2\}$ la suite d'instruction est :

Algorithme 3.1 – Affichage d'un point

```
1 glNewList(glGenLists(1),GL_COMPILE_AND_EXECUTE); // Nouvelle liste
    d'objets graphiques
2   glPointSize(10); // taille du point
3   glBegin(GL_POINTS); // on trace un point
4      glVertex2f(1,2); // coordonnees du point
5   glEnd();
6 glEndList();
```

Dans mon cas l'affichage de mon tableau de point se fait grâce à cette suite d'instruction :

Algorithme 3.2 – Affichage nuage de points

```
1 glNewList(3,GL_COMPILE_AND_EXECUTE);
2   // Parcours du tableau de points
3   for(vector<Point>::iterator it = points.begin(); it !=
       points.end(); ++it)
4      // Création du point graphique
5      creerPoint(*it,0.,0.,0.,8.);
6 glEndList();
```

Pour parcourir le tableau de points, `vector<Point>`, j'utilise un "iterator" qui est un objet que tous les conteneurs ont et qui facilite leur parcours.

Appelant la fonction "creerPoint", qui est une fonction légèrement modifiée de celle qui nous a été fournie, la fonction prend en paramètres le point en question, sa couleur et sa taille :

Algorithme 3.3 – Fonction creerPoint

```
1 void OpenGLManager::creerPoint(Point p, double rouge, double vert,
    double bleu, double taille){
2   glColor3f(rouge,vert,bleu); //initialisation de la couleur
3   glPointSize(taille); // initialisation de la taille
4   glBegin(GL_POINTS); // on trace un point
5      glVertex2f(p.getX(),p.getY()); // coordonnees du point
6   glEnd(); // fin de glBegin
7 }
```

3.2 L'enveloppe convexe

L'intérêt de l'application est le calcul de l'enveloppe convexe et de son affichage. La première étape est donc de calculer cette enveloppe convexe. L'affichage de cette dernière ce fera après.

3.2.1 Calcul

La partie la plus intéressante est donc de calculer l'enveloppe convexe à partir d'un nuage de points.

Pour cela, ont été décris précédement plusieurs algorithme, lidéal serait de mettre en place les trois mais pour commencer nous avons mis en place l'algorithme de Graham (cf. section 2.2) puis l'algorithme du QuickHull (cf. section 2.3).

Algorithme de Graham

La première étape est de trouver le point le plus bas, le point p_0.

Ensuite il faut les trier par taille de l'angle entre la droite passant par le point courant et le point p_0 avec la droite des abscisses.

Puis il faut remplir l'enveloppe convexe avec les points qui appartiennent.

Enfin il suffit de retourner le tableau de points correspondant à l'enveloppe.

Algorithme 3.4 – Fonction algoGraham

```
1  vector<Point> CalculEnveloppeConvexe::algoGraham(vector<Point>
       points){
2     vector<Point> enveloppe;
3
4     if(points.size() < 3)
5        return points;
6
7     // Trouver le point le plus bas
8     int yMinim = 0, i = 0;
9     Point pivot = points.front();
10    for(vector<Point>::iterator it = points.begin(); it !=
          points.end(); ++it){
11       if(*it < pivot){
12          pivot = *it;
13          yMinim = i;
14       }
15       i++;
16    }
17
18    // Le mettre en première position
19    swap(points[0],points.at(yMinim));
20
21    // Trier les points
22    sort(points.begin() + 1, points.end(),
          PointComparateur(pivot.getX(),pivot.getY()));
23
24    // Mettre les trois premiers points dans l'enveloppe
25    enveloppe.push_back(points[0]);
```

```
26    enveloppe.push_back(points[1]);
27    enveloppe.push_back(points[2]);
28
29    // Construire l'enveloppe
30    for(vector<Point>::iterator it = points.begin() + 3; it !=
          points.end(); ++it){
31       Point top = enveloppe.back();
32       enveloppe.pop_back();
33       while(Point::angleHoraire(enveloppe.back(), top, *it) != -1){
34          top = enveloppe.back();
35          enveloppe.pop_back();
36       }
37       enveloppe.push_back(top);
38       enveloppe.push_back(*it);
39    }
40
41    return enveloppe;
42 }
```

Algorithme du Quick-Hull

L'algorithme se divise en 2 fonctions différentes, la première est celle permettant l'initialisation des valeurs et les premiers calculs à effectuer ainsi que d'appeler la deuxième fonction, la deuxième fonction est une fonction dite "récursive", c'est-à-dire qu'elle s'appelle elle-même jusqu'à un cas d'arrêt.

La première étape est de trouver les points les plus éloignés, pour cela j'ai fais le choix de prendre ceux avec respectivement le plus petit `x` et le plus grand `x`.

La deuxième étape consiste en la division de l'ensemble de points de base en deux sous-ensembles dont chacun représente les points d'un coté de la droite passant par les deux points extrêmes.

Il faut ensuite appeler deux fois la fonction récursive avec chacune un sous-ensemble, afin que cette fonction récursive continue de fabriquer l'enveloppe convexe.

L'ajoute des points extrêmes à l'enveloppe doivent ce faire à des moments précis.

A la fin il ne reste plus qu'à retourner l'enveloppe fabriqué.

Algorithme 3.5 – Quick Hull initialisation

```
1  vector<Point> CalculEnveloppeConvexe::quickHull(vector<Point>
       points){
2     vector<Point> enveloppe, dessus, dessous;
3     Graphique graphique(points);
4
5     Point gauche = graphique.pointLePlusAGauche();
6     Point droit = graphique.pointLePlusADroite();
7     enveloppe.push_back(gauche);
8     float distance;
9
10    for(vector<Point>::iterator it = points.begin(); it !=
          points.end(); ++it){
```

```
11      if(*it != gauche && *it != droit){
12         distance = Point::distancePointSegment(*it,gauche,droit);
13         if(distance > 0)
14            dessous.push_back(*it);
15         else if(distance < 0) dessus.push_back(*it);
16      }
17   }
18
19   quickHullRecursive(dessous, gauche, droit, enveloppe);
20   enveloppe.push_back(droit);
21   quickHullRecursive(dessus, droit, gauche, enveloppe);
22
23   return enveloppe;
24 }
```

La fonction permettant réellement de construire l'enveloppe convexe est la fonction récursive, et c'est tout le principe de l'algorithme du QuickHull qui est inspiré du QuickSort et qui utilise une dichotomie de l'ensemble de base jusqu'à ce que les sous-ensembles soient vides.

Pour cela la fonction trouve le point le plus éloigné de manière orthogonal au segment de séparation.

Une fois ce point le plus éloigné trouvé, il faut trouvé deux sous-sous-ensembles composés des points à l'extérieur du triangle, ces points sont répartis dans un ensemble ou l'autre suivant s'ils sont du coté exterieur du segment allant d'un point de la base jusqu'au sommet ou s'ils sont du coté extérieur du segment allant de l'autre point de la base jusqu'au sommet.

De nouveau les points doivent être correctement ajoutés à la base afin de correctement construire l'enveloppe.

Algorithme 3.6 – Quick Hull récursivité

```
1  void CalculEnveloppeConvexe::quickHullRecursive(vector<Point>
       points, Point P, Point Q, vector<Point>& enveloppe){
2     if(points.empty()) return;
3
4     float plusLoin = 0, distance;
5     Point C = points.at(0);
6     vector<Point> s1, s2;
7
8     for(vector<Point>::iterator it = points.begin(); it !=
          points.end(); ++it){
9        distance = Point::distancePointSegment(*it,P,Q);
10       if(distance > plusLoin){
11          plusLoin = distance;
12          C = *it;
13       }
14    }
15
16    for(vector<Point>::iterator it = points.begin(); it !=
          points.end(); ++it){
17       if(*it != P && *it != Q){
```

```
18          if(Point::distancePointSegment(*it,P,C) > 0)
19             s1.push_back(*it);
20          else if(Point::distancePointSegment(*it,C,Q) > 0)
21             s2.push_back(*it);
22       }
23    }
24
25    quickHullRecursive(s1, P, C, enveloppe);
26    enveloppe.push_back(C);
27    quickHullRecursive(s2, C, Q, enveloppe);
28 }
```

Dans ces codes l'appel à d'ifférentes fonctions est effectué, il y a plus de détails dans les codes sources.

Notamment l'utilisation de la fonction permettant de calculer la plus courte distance d'un point à un segment donc orthogonale au segment.

Le point est nommé P et le segment est défini par les points [AB].

Algorithme 3.7 – Fonction calcul distance orthogonale

```
1 double Point::distancePointSegment(Point P, Point A, Point B){
2    return (P.getX() - A.getX())*(B.getY()-A.getY())-((P.getY() -
        A.getY())*(B.getX() - A.getX()));
3 }
```

L'algorithme du QuickHull est le plus efficace notamment dans le cas de point alignés.

Quant à l'algorithme de la marche de Jarvis, nous ne sommes pas parvenu à avoir une bonne efficacité, c'est notamment très long quand il y a beaucoup de point du à sa complexité quadratique mais aussi notre algorithme est inefficace lors de points alignés.

3.2.2 Affichage

Après que l'application ait fini de calculer l'enveloppe convexe du nuage de point, c'est-à-dire une fois que l'application a retourné un tableau de points contenant les points de l'enveloppe convexe, il faut pouvoir afficher cette enveloppe sur le graphe.

Pour cela le plus intéressant est, à mon avis, de relier entre eux les points de l'enveloppe convexe afin de la représenter sur le graphe est de pouvoir vérifier que tous les points sont dans l'ensemble représenté par l'enveloppe convexe.

Pour relier deux points, il suffit de tracer un segment dont l'une des extrémités se situe sur l'un des points et l'autre sur le second point, pour ce faire la suite d'instuction nécessaire est :

Algorithme 3.8 – Affichage d'un segment

```
1 glNewList(glGenLists(1),GL_COMPILE_AND_EXECUTE); // Nouvelle liste
     d'objets graphiques
2    glColor3f(rouge,vert,bleu);//initialisation de la couleur
```

```
3    glLineWidth(taille); // initialisation de la taille
4    glBegin(GL_LINES); // on trace un segment
5       glVertex2f(1,2); // coordonnees du premier point
6       glVertex2f(-1,-2); // coordonnees du dernier point
7    glEnd(); // fin de glBegin
8 glEndList();
```

Dans mon cas il faut afficher un segment entre les points de l'enveloppe convexe de tel manière à la former, il faut donc relier les points dans l'ordre, horaire ou non du moment que les points se suivent, donc depuis le tableau de points préalablement trié, il suffit de parcourir le tableau est de tracer le segment entre le point numéro i et le numéro $n+1$.
Pour autant un problème se pose, lorsqu'on arrive au dernier point, il n'y a pas de point $n+1$, mais il faut relier au point 0.

Le code qui permet de tracer l'enveloppe convexe est :

Algorithme 3.9 – Affichage enveloppe convexe

```
1  glNewList(4,GL_COMPILE_AND_EXECUTE);
2     for(vector<Point>::iterator it = enveloppe.begin(); it !=
          enveloppe.end();){
3        Point p1 = *it, p2 = *++it;
4        // Si on arrive au dernier point
5        if(it == enveloppe.end())
6           // Alors il faut relier au premier
7           p2 = *enveloppe.begin();
8
9        // Tracer le segment de l'enveloppe convexe
10       OpenGLManager::creerSegment(p1,p2,1.0,0.0,1.0,2.0);
11    }
12 glEndList();
```

La suite d'instruction affichant l'enveloppe convexe appelle la fonction "creerSegment" qui prend les deux points de l'extrémité, sa couleur et sa taille, de la même manière que pour un point :

Algorithme 3.10 – Fonction creerSegment

```
1  void OpenGLManager::creerSegment(Point p1, Point p2, double rouge,
       double vert, double bleu, double taille){
2     glColor3f(rouge,vert,bleu);//initialisation de la couleur
3     glLineWidth(taille); // initialisation de la taille
4     glBegin(GL_LINES); // on trace un segment
5        glVertex2f(p1.getX(),p1.getY()); // coordonnees du premier
            point
6        glVertex2f(p2.getX(),p2.getY()); // coordonnees du dernier
            point
7     glEnd(); // fin de glBegin
8  }
```

3.2.3 Compilation et execution

Pour pouvoir utiliser l'application il faut d'abord pouvoir la compiler. Pour ça un Makefile dans lequel les différentes commandes pour compiler, executer, mais aussi nettoyer le répertoire des fichiers objets ".o" pré-édition de liens, ainsi qu'une commande pour compresser le répertoire de travail.

Pour quelques commandes pratiques, un fichier "README" est à disposition.

Conclusion

Pour la réalisation du projet, et donc la programmation d'un algorithme permettant de trouver l'enveloppe convexe d'un ensemble de points distincts, l'idéal est de mettre en place l'algorithme de Graham qui est le plus rapide et le plus efficace des différents algorithmes décris précédement dans ce rapport. Toute fois il est intéressant d'implémenter plusieurs algorithmes.

Nous avons donc décidé de mettre en place la marche de Jarvis ainsi que le QuickHull.

Il se trouve que ce dernier est le plus fonctionnel dans notre application de calcul de l'enveloppe convexe.

Pour conclure, ce projet est très intéressant et va permettre d'appliquer et d'acquérir des connaissances en programmation, mais aussi de lier véritablement l'informatique aux mathématiques afin de résoudre un problème plus ou moins long et complexe.

De plus ce projet risque de nous donner du fil à retordre que ce soit pour l'écriture de l'algorithme choisi mais aussi pour la partie affichage graphiquement du résultat, ce qui va être très instructif et va nous demander de la rigueur quant à la réalisation de celui-ci.

Bibliographie/Webographie

Liens url des sites utilisés pour la compréhension du sujet et la réalisation de ce rapport.

Wikipedia :

OpenGL : `https://fr.wikipedia.org/wiki/OpenGL`

Enveloppe Convexe : `https://fr.wikipedia.org/wiki/Enveloppe_convexe`

Marche de Jarvis : `https://fr.wikipedia.org/wiki/Marche_de_Jarvis`

Parcours de Graham : `https://fr.wikipedia.org/wiki/Parcours_de_Graham`

Complexité algorithmique : `https://fr.wikipedia.org/wiki/Complexité_en_temps`

QuickHull : `https://en.wikipedia.org/wiki/Quickhull`

Autres :

`http://pauillac.inria.fr/~maranget/X/TC/TD-10/index.html`

`https://who.rocq.inria.fr/Jean-Charles.Gilbert/p6/1-rappels-ac.pdf`

`http://imss-www.upmf-grenoble.fr/prevert/Prog/Complexite/enveloppeConvexe.html`

Table des figures

Algorithmes