**Le Problème du Cercle Minimum**

Notre projet consiste à trouver le cercle le plus petit contenant un ensemble de points d’un plan. On a utilisé « Eclipse Neon » comme compilateur, afin de visualiser ce code, on a rajouté SDL 1.

Dans un premier temps, on a révisé le cours d’Algèbre (Théorème de Cramer) et intégré les propriétés géométrique dans le projet, après avoir terminé la théorie du code, on a commencé à écrire le code utilisant toutes ces informations.

Dans la partie du code, on a créé des structures pour déterminer les points qu’on saisit par clavier puisqu’on a utilisé des points et des cercles. Pour résoudre notre problème, on a décidé d’appliquer l’algorithme de ‘’recherche par essai-erreur’’ qui contient deux méthodes.

* les cercles dont les diamètres sont les segments formés par les points pris deux par deux — il y en a *n*(*n* - 1)/2
* les cercles circonscrits aux triangles formés par les points pris trois par trois — il y en a *n*(*n* - 1)(*n* - 2)/6

Pour être convaincu que le cercle qu’on a trouvé est le minimum, il faut donc, pour chaque cercle, vérifier si les *n* - 2 ou *n* - 3 points restants sont dans le cercle. L’algorithme a donc une complexité en [O(n4)](https://fr.wikipedia.org/wiki/Complexit%C3%A9_en_temps).

Sur le programme ‘Cercle Minimum’, on a 4 fichiers qui sont **main.c, primitive.h, primitives\_graphiques.c** et **primitives\_graphiques.h**. Les fichiers primitives\_graphiques.c et primitives\_graphiques.h c contiennent les définitions des fonctions de SDL. Le fichier primitive.h contient toutes les structures et les fonctions que l’on a créées pour trouver le cercle minimum.

*Fonction « cercle* ***methode\_2par2****(point \*tab, int n\_pts) »*

On cherche la distance maximum de deux points distincts en utilisant deux boucle nichée. Après avoir trouvé la distance la plus grande, on suppose que cette distance est le diamètre d’un cercle et puis une autre fonction *int verification(cercle c, point \*tab)* qui permet de vérifier si les n-2 points restants sont inclus dans le cercle qu’on tracera.

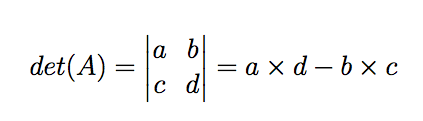
*Fonction « cercle* ***methode\_3par3****(point \*tab, int n\_pts) »*

Notre but est de trouver tous les 3 points circonscrits à un triangle afin de trouver les équations des médiatrices finalement et de considérer les intersections comme centre du cercle que l'on tracera.

On a 3 fonctions nichées qui font déplacer des points milieux des médiatrices. En utilisant le théorème de Cramer, nous trouvons le point des intersections qui est admis comme les coordonnées du centre .On a comparé ce point avec tous les autres points. On a pris la distance entre ce point de centre et tous les autres points, et testé si tous les points se trouvent dans ce cercle. Si on peut trouver plus de cercles, on prend le cercle qui contient le plus petit rayon entre eux. Si on n'a trouvé aucun cercle, cette fonction retourne le cercle de rayon -1 qui signifie qu'on n'a trouvé aucun cercle en utilisant cette méthode.

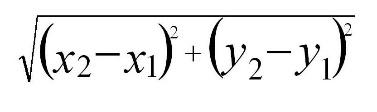
*Fonction « double* ***determinant****(double a, double b, double c, double d) »*

Cette fonction est de type double et ses 4 paramètres sont aussi de type double. Elle prend a, b, c, d et elle retourne leur déterminant. Cette fonction calcule le determinant des matrices 2x2.



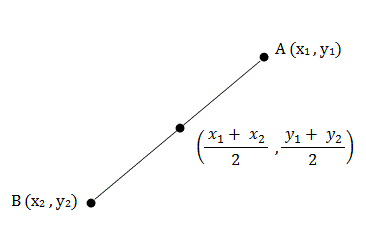
*Fonction « double* ***distance\_points****(point a, point b) »*

Cette fonction est de type double. Elle prend 2 points comme paramètres. Simplement, elle retourne une valeur de type double qui est égale la distance.



*Fonction « point* ***millieu****(point a, point b) »*

Cette fonction prend 2 paramètres qui sont de type point. Cette fonction sert à déterminer le point qui se trouve entre les 2 points que l'utilisateur a saisi.



*Fonction « int* ***verification****(cercle c, point \*tab, int n\_pts) »*

Cette fonction est de type int et elle prend 3 arguments, un cercle, l’ensemble de points et nombre de points. Pour chaque point, elle vérifie si la distance entre le centre du cercle et ce point est plus petite que rayon.

*|x-a|<r avec a : centre du cercle, x : un point de l’ensemble de points, r : rayon du cercle.*

Si une d’eux est hors du cercle, on retourne « -1 ». Si tous les points sont dans le cercle, on retourne « 1 ».

*Fonction « void* ***trace\_les\_axes****() »*

Cette fonction sert à tracer le système de coordonnées cartésiennes. Dans cette fonction, On appelle le fonction *« void trace\_segment\_surface(SDL\_Surface\* surface,int xa,int ya, int xb, int yb, Uint8 r, Uint8 v, Uint8 b) »* qui se trouve dans le fichier primitives\_graphiques.c. Ainsi, on dessine un segment vertical (l’axe y) au millieu de l’écran et un segment horizontale (l’axe x) au millieu de l’écran.

*Fonction « void* ***epaissir\_cercle****(cercle c, int n) »*

Le cercle peut être vu très mince. Pour cette raison, nous avons eu besoin d'une fonction comme celle-là. Cette fonction prend comme argument le cercle que nous avons trouvé à la fin et trace n cercles plus dont la rayonne est 1 pixel de plus.

*Fonction «****double******de\_pixel\_a\_coor\_x****(int x, int longueur\_daxe) » et*

*«****double******de\_pixel\_a\_coor\_y****(int y, int longueur\_daxe)*

Sur l’écran, les pixels commence en haut à gauche. Mais le point(0,0) du système des coordonnées cartésiennes est au milieu de l’écran. Aussi, le nombre de pixels et longueur des axes qu’on a choisi sont différentes. Donc, on a besoin d’une échelle. Par exemple, pour :

*HAUTEUR : 600 Longueur de l’axe x : 20 cm*

*LARGEUR : 600 Longueur de l’axe y : 20 cm*

notre échelle est « *HAUTEUR / Longueur de l’axe x »* (ou « *LARGEUR / Longueur de l’axe y »*).

Formule générale pour x : (x-(LARGEUR/2))/echelle

Formule générale pour y : -(y-(HAUTEUR/2))/echelle

Il y a un « - » au début car pour y (pour système de pixels), quand on va vers le haut, le coordonné y diminue.

*Fonction « cercle* ***de\_coor\_xy\_a\_pixel****(cercle c, int longueur\_daxe) »*

(Pour comprend cette fonction, il faut d’abord lire l’explication écrite dans la fonction précédente)

Pour trouver le cercle minimum, on fait tous les calculs en coordonnées cartésiennes. Mais pour tracer ce cercle, on a besoin de passer aux coordonnées pixels car la fonction qui va tracer le cercle *« void trace\_cercle\_surface(SDL\_Surface\* surface,int x0, int y0, int rayon,Uint8 r, Uint8 v, Uint8 b) »* prend ses arguments en pixels. Donc, pour ses transformations :

Formule générale pour x : echelle\*x + LARGEUR/2

Formule générale pour y : -echelle\*y + HAUTEUR/2 (Il y a un « - » au début car pour y (pour système de pixels), quand on va vers le haut, le coordonné y diminue.)

Formule pour rayon : double rayon=c.rayon\*(echelle);

c.rayon=(int)rayon;

On a besoin de passer de « double » à « int » car l’argument de la fonction qui va tracer cercle est de type int.

*Fonction « point\** ***choisir\_avec\_Mouse****(point \*tab, int \*n\_pts, int longueur\_daxes) »*

Cette fonction est de type point\* et prend 3 arguments, le nuage, nombre de points et longueur des axes. Il est utile d’utiliser un pointeur pour nombre de points(deuxième argument), de cette façon, on va modifier le variable « int **n\_pts**=0 ; » qui se trouve dans le fichier main.c et qui va être utiliser comme argument dans les fonctions *« cercle methode\_3par3(point \*tab, int n\_pts) », « cercle methode\_2par2(point \*tab, int n\_pts) » et « int verification(cercle c, point \*tab, int n\_pts) »*

Cette fonction, pour chaque clic droit, dessine un point-là et stocke ses coordonnées (en pixels) de ce point dans un tableau(le nuage) et pour chaque clic il augmente le compteur dont le nom est n\_pts (nombre de points). Quand on clique gauche, on sort de boucle.

*Fonction « int* ***main****(int argc, char \*\*argv) »*

Finalement, après avoir écrit ces fonctions nous pouvons donc terminer le programme. En utilisant des fonctions qu’on en a déjà parlé, on peut trouver le cercle minimum, soit en précisant des points avec la quantité de ces points ou/et en cliquant sur la fenêtre de SDL.

Dans un premier temps, on appelle la fenêtre de SDL, “Trouver le cercle minimum”. Si on va saisir les points par clavier, il nous faut donner le nombre de points puisque les méthodes 2 par 2 et 3 par 3 en ont besoin pour faire tourner les boucles.

Dans un deuxième temps, on crée un tableau de points pour stocker les points. Puis, on trace les axes x,y sur SDL. Après cet étape, la fonction choisir\_avec\_Mouse(nuage,&n\_pts,longueur\_daxes)commence à fonctionner, si on clique le bouton gauche pour préciser un points, jusqu'à on clique le bouton droite, cette fonction va continuer de prendre les points sur lesquels on clique. Pourtant, cette fonction fait stocker les points à partir d’indice “n\_pts”. Donc on peut dire que c’est possible de préciser des points en saisissant par clavier et aussi en cliquant les boutons de souris d’ordinateur.

On déclare un cercle c et on commence par fonction 2par2, si le cercle que la fonction 2par2 a retourné ne vérifie pas la condition: “Tous les points qui sont dans le nuage doivent se trouver dans le cercle.”, alors dans ce cas la méthode 3 par 3 marche. On préfère d’abord essayer de méthode 2 par 2 car elle est plus simple que la méthode 3 par 3. Puis, si la fonction nous renvoi un cercle (de rayon!=-1) c’est à dire qu’on a finalement trouvé le cercle minimum. Donc on va dessiner ce cercle sur SDL maintenant avec les aides des fonctions de SDL qu’on a déjà expliqué.

**Source:**

https://fr.wikipedia.org/wiki/Probl%C3%A8me\_du\_cercle\_minimum

Berkan Gamsız,

Enes Uysal,

Mustafa Halkalı.