KAOUANE MARWANE TD4/TP6
NEJI WADI TD2/TP3

Sommaire

Présentation du Projet :

Durant plus d'un mois nous avons participer à un projet qui avait pour but d'aborder les différentes étapes de conception d'une enveloppe convexe.

Nous avons du faire face à de nombreuses difficultés et challenges dans la réalisions de ce projet qui nous ont permis d'apprendre et d'évoluer pour de futurs.

Manuel Utilisateur :

Les explications pour gérer les lignes de commandes sont dans le readme, on peut le lancer dans le terminal avec la commande : ***more readme.md***

Résumé des étapes:

Liste doublement chainées bouclées:

Créer une liste doublement chaînée bouclées avec plusieurs fonctions bien utiles comme une fonction qui renvoie sa longueur, ses éléments à l'intérieur etc.

Initialiser le polygone :

Cette étape consiste à modéliser sous la forme d'un triangle l'enveloppe convexe correspondant à la

Traitement et Insertion de nouveau point dans l'enveloppe:

```c
double randDouble(){
    return ((double)rand()) / (double)RAND_MAX;
}

double randDoubleI(double n, double m){  //Renvoie un intervalle -> [n, m]
    return randDouble() * (m-n) + n;
}
```

Ici il fallait vérifier si le point était direct ou non pour cela on à utiliser une fonction qui va prendre en paramètre le point qui précède le point que l'on va insérer dans la liste (expliquer dans Fonction).

Nettoyage avant et arrière:

Plusieurs difficultés rencontrées lors de cette étape notamment gérer le parcours de la liste.

Quelques Fonctionnalités :

Fonctions :

- Nous avons créée une fonction **randDouble et randDoubleI** qui tire un nombre de **type double** dans un intervalle compris entre les deux double passer en paramètres, eux inclus. Cela nous a été utile notamment pour gérer le nuage de point sous forme de cercle et de carré.

```c
void affichePoint(int nb, Point *tab, ConvexHull env){
    for(int i = 0; i<nb; i++){
        if(tab[i].x != 0.000000 && tab[i].y != 0.000000)  //Si le point à été initialiser puisqu'on à utiliser un calloc
            MLV_draw_filled_circle(tab[i].x, tab[i].y, 2, MLV_COLOR_GOLD2);
    }
    Polygon tmp = env.pol;
    do{
        MLV_draw_filled_circle(tmp->s->x, tmp->s->y, 3, MLV_COLOR_RED);
        tmp = tmp->next;
    }
    while(tmp != env.pol);
}
```

```c
void afficheEnv(ConvexHull env){
    Polygon temp = env.pol;
    do{
        MLV_draw_line(temp->s->x, temp->s->y, temp->next->s->x, temp->next->s->y, MLV_COLOR_BLUE_VIOLET);
        temp = temp->next;
    }while(temp != env.pol);
}
```

- Pour optimiser l'affichage des points et de l'enveloppe nous avons mis au point deux fonctions, une qui affiche tout les points et une autre qui affiche l'enveloppe. Nous avons pour cela créer un tableau de type **Point** avec la fonction **calloc(n*sizeof(Point))**, de taille **n**, n étant le nombre de points saisit par l'utilisateur, et à chaque itération i, le tableau à l'indice i prend la valeur du point générer. Les points dans l'enveloppe sont d'une couleur différentes que les Vertex de l'enveloppe.

- La fonction insererEnv(), prend plusieurs paramètres mais en particuliers un de type Point qui n'est autre que le point qui précède le point que l'on va insérer. On va donc chercher ce point en parcourant le Polygone et une fois ce point trouver on va insérer le point juste après celui-ci.

```
Polygon insererEnv(Polygon l, Polygon *liste, Point p, Vertex v){
    Polygon tmp = NULL;
    tmp = allouerCellule(v);
    Polygon q = l;  //Variable qui va nous permettre de parcourir la liste et de placer le nouveau point insérer en tête de liste

    if (!l) //Si c'est une liste vide
        { *liste = tmp; return tmp; }

    do{  //On parcours la liste
        if(q->s->x == p.x && q->s->y == p.y){  //Si on trouve le Point qui précède le point qu'on va ajouter
            tmp->next = q->next;  //Le suivant de tmp est le suivant de l
            if (q->next) tmp->next->prev = tmp;
            tmp->prev = q;
            q->next = tmp;
            q = q->next;  //On place le point insérer en tête de liste
            return q;
        }
        q = q->next;  //On passe au suivant
    }while(q!=l);  //Tant qu'on a pas fini de parcourir la liste
    return q;
}
```

Difficultés rencontrées :

La première difficulté et sûrement la plus compliqué durant ce projet était de comprendre comment fonctionnais la liste doublement chainées bouclées avec des type Point* . Nous étions pas à l'aise avec les pointeurs, puis on a compris comment cela fonctionnais petit à petit.

Le fait d'allouer un nouvel espace pour les Point* nous avais échapper, on essayais tant bien que mal à initialiser la valeur des Point* sans succès, que des Segfault. Une fois compris , le fait de boucler la liste était aussi rude. Le reste du programme était plutôt intéressant , le nettoyage avant et arrière était compliqué à gérer en particulier le parcours de la liste .

Dernière difficulté principale rencontré était l'affichage du cercle et du carré. On a ensuite compris comment cela fonctionnais, et nous avons utiliser des fonctions prédéfinis comme **cos** et **sin.**

Pistes d’améliorations de notre travail :

- La fonctionnalité pour quitter l’application lorsque l’on a sélectionner l’insertion des points à la souris, ne fonctionne pas très bien. Il faut appuyer sur **echap + click gauche**.
- Essayer de centrer le carré pour qu’il grandisse aussi du côté gauche et en haut. Le problème est que le carré est centré en haut à gauche et donc limiter par sa taille ne va pas grandir de ces côtés.
- Faire la partie 2 du projet.