- Rapport
  - Tache 1 - Image Bitmap
    - Structures de données
    - Structure du programme
    - Difficultés, problems et resolutions
  - Tache 2 - Géométrie 2D
    - Structure du programme
    - Structures de données
    - Difficultés, problems et resolutions
  - Tache 3 - Extraction d'un contour externe
    - Structure du programme
    - Structures de données
      - sequence_point.h
      - contour.h
    - Difficultés, problems et resolutions
  - Tache 4 - Sortie fichier au format EPS
    - Structure du programme
    - Structures de données
    - Difficultés, problems et resolutions
  - Tache 5 - Extraction des contours d'un image
    - Structure du programme
    - Structures de données
    - Difficultés, problems et resolutions
    - Extra
  - Tache 6 - simplification de contours par segments
    - Structure du programme
    - Structures de données
    - Difficultés, problems et resolutions
  - Tache 7 - Simplification de contours par Bézier
    - Structure du programme
    - Structures de données
    - Difficultés, problems et resolutions
- Manuel utilisateur
  - Tests
  - Tache 5
  - Tache 6
  - Tache 7
    - test_degree2
    - test_degree3
- Diagramme de Gantt
- Journal de bord
  - Copyright 2023 | All rights reserved

Gros Maxime, Vasilis Skarleas

Rapport

Tache 1 - Image Bitmap

Dans cette première tache le plus dur fut de compendre comment on allait s'y prendre pour extraire des données de chaque image bitmap (en évaluant chauqe pixel par noir ou blanc).

Lors de cette première on a dut construire les bases de notre codes qui sont extraire les données d'une image, on sauvegarde donc dans un premier temps la taille de l'image et on enregistre dans un tableau les coordonnées de chaque pixel noir.

Structures de données

Dans cette tache pour la realisation des fonctions ***ecrire_image*** et ***negatif_image*** on utilise la structure de base d'une image bitmap qui est compose par le largeur de l'image, l'hauteur de l'image et un pointeur vers le tableu de pixels.

Structure du programme

Pour le tache 1, le code principal des fonctions ***ecrire_image*** et ***negatif_image*** sont inclus sur le fichier ***image.c*** qui viens avec un fichier de liaisons ***image.h***

Difficultés, problems et resolutions

Dans cette partie du rapport final on decrit en bullets les differents problems avec la resolution finale pour resoudre ce probleme, ainsi que differents points qu'on trouvé difficile pendant la realisation du tache.

Pour le Tache 1:

- Utilisation du profil de l'image pour recuperer l'hauter et le largeur au lieu d'utiliser les fonctions déja definis au debut du projet (source code initial)
    - On a modifié le syntax sur nos fonctions ***ecrire_image*** et ***negatif_image*** pour faire une appelle aux fonctions hauteur_image et largeur_image.
- Retouner une image negative sans modification de l'image original
    - Initialisation d'une image en utilisant la fonction creer_image, tout en declarant son hauteur et largeur par rapport l'hauteur et le largeur de l'image initial.
- Probleme du bon lecture de l'image initial (soit pour effectuer un affichage sur le terminal en utilisant 0 et 1, soit pour creer le negatif de cette image) aux fonctions ***ecrire_image*** et ***negatif_image***.

    - Initialisation de variable de parcours largeur-hauteur (boucle imbroques) à 1

```
for (int i=1; i<=hauteur; i++)
        {
                for (int j=1;j<=largeur; j++)
                {
...
```

au lieu de 0

```
for (int i=0; i<hauteur; i++)
        {
                for (int j=0;j<largeur; j++)
                {
...
```

- Avoir une lecture des pixels le plus efficace pour les fonctions *ecrire_image* et *negatif_image*
    - Utilisation de la syntax switch(A), avec A le couleur du pixel en question

```
...
A = get_pixel_image(I, j, i);
                switch(A)
                        {
                                case(BLANC):
                                        ...
                                        break;
                                default:
                                        ...
                                        break;
                        }
...
```

Tache 2 - Géométrie 2D

Maintenant l'objectif de cette tache est de créer les différents objets/fonctions mathématiques que nous aurions besoin dans le reste du projets.

Tous ces objets peuvent être trouver dans le fichier *geom2d.h* ,il sera compléter au fur et à mesure mais la plus grosse partie des fonctions géométrique qu'on utlisera seront créer lors de cette tache. La création du type point qu'on utilisera durant tous le projet sera créer à ce moment là.

Structure du programme

Pour la meileur organisation des fichier de code, on a créé un nouveau fichier nome *geom2d.c* qui inclut tous les fonctions qui correspondent en calcul geometrique en dimension 2.

De plus, on a construit le fichier *geom2d.h* dans lequel on a declaré les structures de données utilisés par les modules sur *geom2d.c*. Aussi, ce fichier inclut le profil de chaque focntion ecrit sur *geom2d.c*

Structures de données

Dans cette partie du projet, on aura besoin de declarer les types Vecteur et Point. Leur structure est donnes ci-dessous (*geom2d.h*)

```
//Type Vecteur
typedef struct Vecteur_
{
   double x,y;
} Vecteur;

//Type Point
typedef struct Point_
{
   double x,y;
} Point;
```

Difficultés, problems et resolutions

- Devrait utiliser des fonctiosn mathematiques en C. Par exemple: sqrt() pour la racine d'un nombre reel
    - On a definit *<math.h>* sur *geom2d.c*
- Choicir le bon typage pour les fonctions geometrqiues en plan de dimension 2
    - Utiliser le type double pour les fonctions de la norme, de la distance et du produit scalaire

Tache 3 - Extraction d'un contour externe

Structure du programme

Dans cette partie du projet, on a cree un nouveau fichier qui s'appele ***contour.c***. À partir du cette tache, c'est cette fichier qu'on va mettre à jour majoritement. Pour le tache 3, on a declaré tous les fonctions necesaires pour l'extraction d'un contour externe via la methode d'un robot virtuel.

Ça veut dire qu'il s'agit des fonctions d'initialisation du robot, lecture de la position, mis à jour de ses coordones et orientation, declaration de l'agorithme pour l'extraction d'un seul contour, ainsi que la recherche du pixel de depart (pixel par lequel on va commencer de chercher pour un contour). Effectivement, le fichier contour.c viens avec le fichier contour.h qui inclut la declaration de la structure des donnes pour le robot virtuel et son orientation ainsi que les profils de tous les fonctions.

De plus, on a declaré un fichier qui s'appelle ***sequence_point.c***. Il s'agit d'un fichier essentiel qui inclut tous les aides de traitment des structures des donnes declaré sur ***sequence_point.h***. À partir de l'application de l'agorithme de contour (chercher pour un contour externe), on a besoin de stoquer tous les points du contour pour faire son extraction. C'est pourquoi, on a besoin de declarer des nouveaux types (structures des donnes).

Comme expliqué dans la partie ***structures de données - sequence_point.h***, l'utilisation des listes chainées est essentiel. Dans ce cadre la, les aides de traitment d'une liste des points s'agisent des fonctions

de concatenation des listes, de suppresions des elements, initialisation d'une telle liste, ajouter elements dans une liste, ainsi que transformation d'une telle liste vers un tableau (qu'on aura besoin aux prochaines taches).

Structures de données

sequence_point.h

Vu qu'on ne connait pas le nombre exact des points d'un contour externe, on a besoin de declarer une liste des points (contour) avec un taille dynamique. C'est pourquoi il est utilisé la syntax d'une liste chainé. Effectivement, une liste chaine est une chaine des cellules laquelle à son ensemble construit la liste chainé. Alors on a:

```
...
/*---- le type cellule de liste de point ----*/
typedef struct Cellule_Liste_Point_
{
        Point data;    /* donn@e de l'@l@ment de liste */
        struct Cellule_Liste_Point_* suiv; /* pointeur sur l'@l@ment suivant */
} Cellule_Liste_Point;

/*---- le type liste de point ----*/
typedef struct Liste_Point_
{
        unsigned int taille;        /* nombre d'@l@ments dans la liste */
        Cellule_Liste_Point *first; /* pointeur sur le premier @l@ment de la liste */
        Cellule_Liste_Point *last;  /* pointeur sur le dernier @l@ment de la liste */
                                /* first = last = NULL et taille = 0 <=> liste vide */
} Liste_Point;

typedef Liste_Point Contour; /* type Contour = type Liste_Point */
...
```

Pour la transformation d'une liste chaine vers un tableau, on a besoin de declarer le typage d'un tel tableau. Alors on a:

```
...
/*---- le type tableau de point ----*/
typedef struct Tableau_Point_
{
        unsigned int taille; /* nombre d'@l@ments dans le tableau */
        Point *tab;          /* (pointeur vers) le tableau des @l@ments */
} Tableau_Point;
...
```

---

contour.h

On declare comme type enumere l'orientation qui peut etre pris par le robot. Après, on commence avec la declaration du robot virtuel qui a besoin des coordonees (x,y) et l'orientation. Alors on a:

```
...
typedef enum {Nord, Est, Sud, Ouest} Orientation;

typedef struct {
  int x, y;
  Orientation o;
} Robot;
...
```

Difficultés, problems et resolutions

- Trouver la signature de l'algo de contour (*algo_contour*)
    - On a decidé de prendre une image et un nom du fichier. En ce qui concerne le fichier, il s'agot d'un fichier .txt qui inclus les points d'un contour (tres utile pour la verification que la fonction marce comme attendue)
- Posibilité d'ecrire sur le fichier les differents points du contour sans lire le contour en utilisant des cellules vu que cette notion n'etait pas trés claire à notre tete pendant la periode de realisation de cette tache.
    - Transformation du contour (liste des points [liste chainé]) à un tableau. Aprés on fait un parcours normal du tableau, tout en ecrivant sur le fichier (*fptr*) le coordonees du chaque point (*P.x* et *P.y*)

```
...
Tableau_Point TP = sequence_points_liste_vers_tableau(c);
        int k;
        int nP = TP.taille;
        fprintf(fptr, "\n");
        fprintf(fptr, "%d\n", nP);
        for (k = 0; k < nP; k++)
        {
            Point P = TP.tab[k];
            fprintf(fptr, "%.1f %.1f\n", P.x, P.y);
        }
        free(TP.tab);
...
```

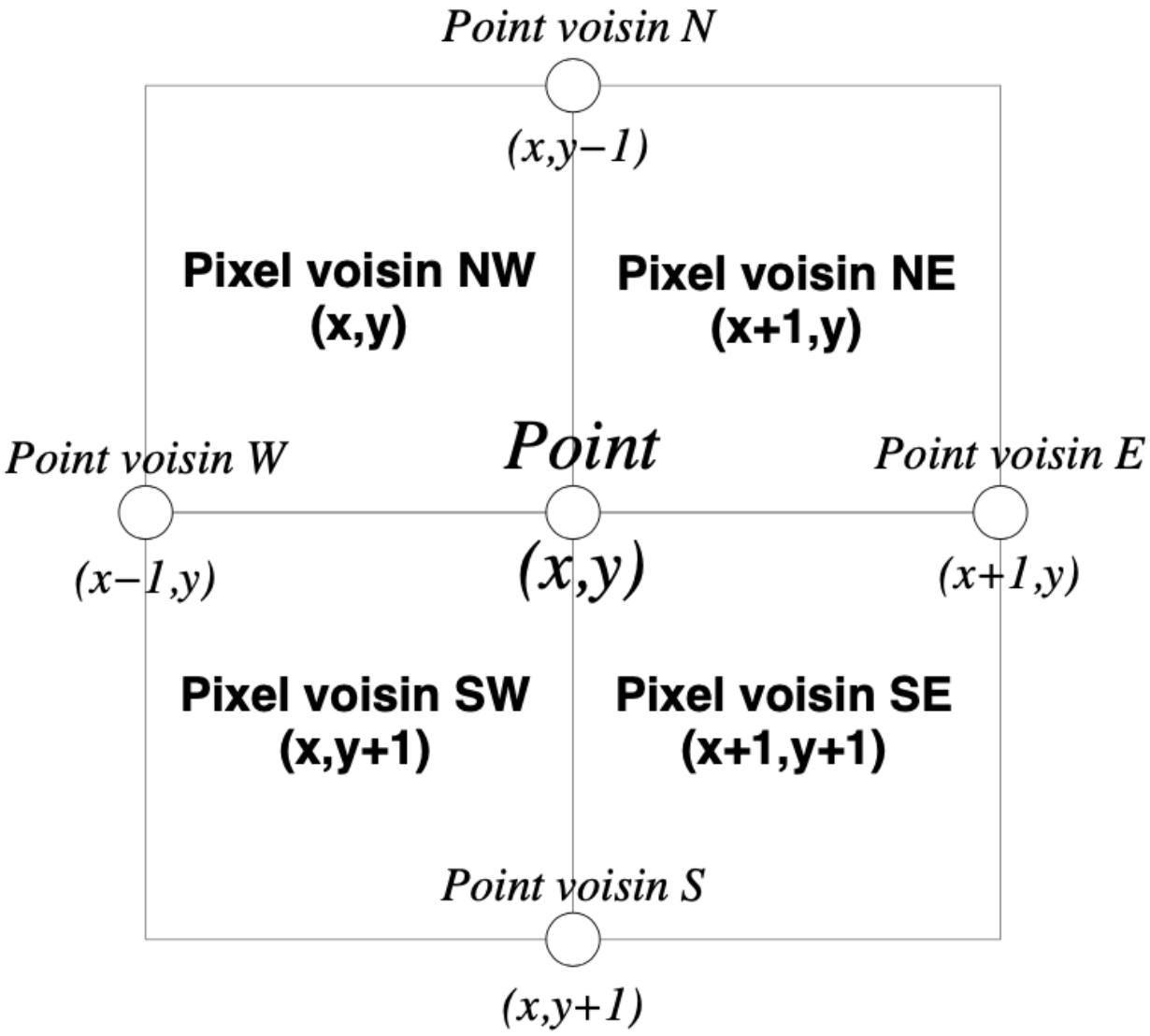
- Etablir une logique pour trouver le point de depart
    - Dans le cadre de la lecture de l'image original, si on trouve un pixel noir tel que son voisin en nord est blanch, on retourne cette pixel (ses coordones x et y) comme le point de depart. Pour le pixel voisin il suffit de regarder en meme largeur mais en hauteur-1 de notre pixel en question comme marqué ci-dessous:

```
...
case (NOIR):
                if (j != largeur)
                {
                    voisin = get_pixel_image(I, j,
```

- L'agorithme de contour demande de calculer la nouvelle orientation du robot pour qu'on peut bien suivre le contour externe. La detection des bonnes pixels-voisin autour de notre pixel en question par

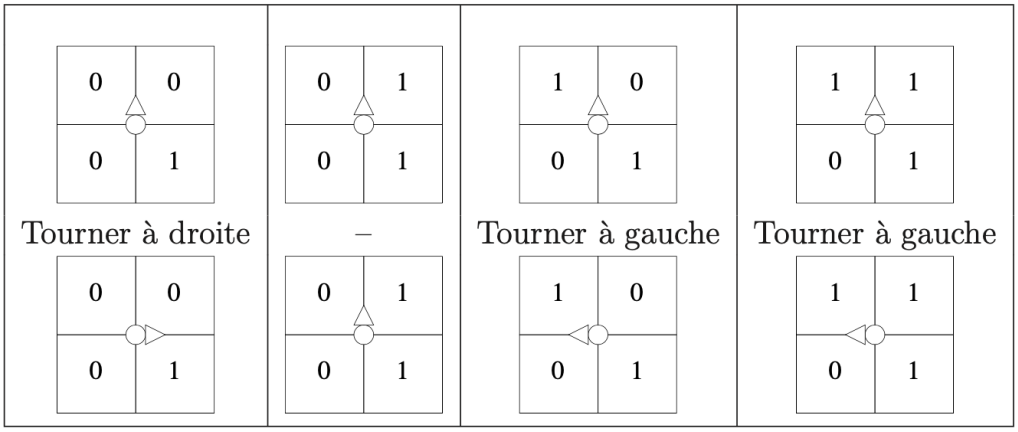
rapport l'orientation du robot etait essentiel:

- On a decidé d'avancer en divisant notre problem en 4 sous-cas qui dependent par l'orientation du notre robot. Après, en utilisant l'image "voisins d'un point (x,y)" on recupere les bonnes pixels voisins à gauche et à droit, et par rapport la logique expliqué sur l'image "configurations possibles", on decide la nouvelle orientation du robot.



Voisins d'un point (x, y)

Il y a quatre configurations possibles :



Alors on obtiens la loqique suivante:

```
...
switch (r->o)
    {
    case (Est):
        pG = get_pixel_image(I, x + 1, y);
        pD = get_pixel_image(I, x + 1, y + 1);
        if (pG == NOIR)
        {
            r->o = Nord;
            break;
        }
        else if (pD == BLANC)
        {
            r->o = Sud;
            break;
        }
        else
        {
            break;
        }
    case (Nord):
        pG = get_pixel_image(I, x, y);
        pD = get_pixel_image(I, x + 1, y);
        if (pG == NOIR)
        {
            r->o = Ouest;
            break;
        }
        else if (pD == BLANC)
        {
            r->o = Est;
            break;
        }
        else
        {
            break;
        }
    case (Sud):
        pG = get_pixel_image(I, x + 1, y + 1);
        pD = get_pixel_image(I, x, y + 1);
        if (pG == NOIR)
        {
            r->o = Est;
            break;
        }
        else if (pD == BLANC)
        {
```

```
            r->o = Ouest;
            break;
        }
        else
        {
            break;
        }
    case (Ouest):
        pG = get_pixel_image(I, x, y + 1);
        pD = get_pixel_image(I, x, y);
        if (pG == NOIR)
        {
            r->o = Sud;
            break;
        }
        else if (pD == BLANC)
        {
            r->o = Nord;
            break;
        }
        else
        {
            break;
        }
    }
...
```

- On avait un probleme par rapport le tournage à gauche ou à droit du robot
  - Effectivement, c'est l'orientation du robot qui change si on tourne à gauche ou à droit. Avec une ΣOBAPH reflection on a obtenu la logique suivante:

```
...
void tourner_a_gauche(Robot *r)
{
    switch (r->o)
    {
    case Nord:
        r->o = Ouest;
        break;
    case Ouest:
        r->o = Sud;
        break;
    case Sud:
        r->o = Est;
        break;
    case Est:
        r->o = Nord;
        break;
    }
}

/* faire tourner le robot à droite */
void tourner_a_droite(Robot *r)
{
    switch (r->o)
    {
    case Nord:
        r->o = Est;
        break;
    case Ouest:
        r->o = Nord;
        break;
    case Sud:
        r->o = Ouest;
        break;
```

```
    case Est:
        r->o = Sud;
        break;
    }
}
...
```

Tache 4 - Sortie fichier au format EPS

Structure du programme

Dans le fichier contour.c on ajouté les fonctions *create_postscript* et *create_postscript_fill* qui construit le fichier EPS. Effectivement, le fichier contour.h etait mis à jour avec les profils des nouveaux fonctions.

Structures de données

Pas des nouvelles structures etait crée. On utilise la structure de données liste_point pour la realisation de cette tache.

Difficultés, problems et resolutions

- Comment automatiser la processus de la creation et l'ouverture d'un fichier .EPS à partir d'un fichier .PBM passé en argument
    - Utilisation de la fonction pre-declare en C strtok pour recuperer le nom du fichier sans l'extension .PBM, tout en allocant la memoire necesaire pour l'ajoute de l'extension .EPS en utilisant le malloc et creation du nom final avec l'extension .eps

```
...
// Extension managment
    char *no_extension = strtok(file_name, ".");
    char *with_extension = malloc(strlen(no_extension) + 4);
    strcpy(with_extension, no_extension);
    strcat(with_extension, ".eps");
...
```

- Image sur le fichier EPS etait affichè à l'invers
    - Modification de la valeur *y* d'un point avec hauteur de l'image moins ca valeur initial comme marqué ci-dessous:

```
...
Cellule_Liste_Point *el;
    el = c.first;
    fprintf(fptr, "%.0f %.0f moveto ", el->data.x, hauteur-el->data.y);
    el = el->suiv;
    while (el != NULL)
    {
        fprintf(fptr, "%.0f %.0f lineto ", el->data.x, hauteur-el->data.y);
```

```
            el = el->suiv;
        }
    ...
```

- N'oubliez pas d'ajouter le *setlinewidth* sur le fichier EPS
- Faire atention en mode de remplisage (fill ou stroke)

Tache 5 - Extraction des contours d'un image

Structure du programme

Dans le fichier contour.c on ajoute la fonction algo_contours qui renvoie la liste des contours pour un image. Pour etre capable de detecter tous les contours d'une image on a besoin le mask de l'image originale. Il s'agit d'une image qui a que les pixels noirs de l'image originale avec un voisin en nord blanche.

Alors, au lieu de chercher le point de depart de chaque contour dans l'image originale, on le cherche dans l'image mask qui est mis à jour chaque fois qu'un point est mis sur le contour en question.

Effectivement, on a besoin une maniere de detcter qu'il n'y plus des contours pour l'image orginale. C'est pourquoi, on a construit la fonction *image_mask* qui verifie si il y a ou pas des pixels noirs ou si l'image a que des pixels blanches. Si l'image mask a que des pixels blanches, alors ça veut dire qu'il n'y a pas d'autres contours.

Dans le cadre des compte rendus pour le Tache 5, on avait besoin d'ajouter quelques d'autres fonctions sur le fichier contour.c. Il s'agit des fonctions:

1. *ecrire_fichier_contours*: Creer un .TXT avec tous les contours au lieu d'un seul contour comme on avait fait en Tache 4
2. *contours_data*: Il affiche sur le terminal le nombre des contours et le nombre des segments totales
3. *create_postscript_contours*: Ecriture du fichier .EPS pour un multiple des conours

Evidement, les profils des nouveaux fonctions etaient ajouté sur contour.h

Structures de données

Dans cette partie du projet, il y avait quelques additions des types sur le fichier sequence_point.h et des fonctions d'aide correspondante sue sequence_point.c. Pour etre capable de traiter le mupliple nombre des contours, on a construit un nouveau type nomé *Liste_Contours* qui est en fait une liste chainé des contours (liste_point).

Dans ce cadre là, on a construit le type Cellule_Liste_Contours qu'on a besoin de construire la liste chaine des contours. Vous pouvez trouver les structures de données consruit ci-dessous:

```
...
/*---- le type cellule de liste de point ----*/
typedef struct Cellule_Liste_Point_
{
```

```
        Point data;    /* donn@e de l'@l@ment de liste */
        struct Cellule_Liste_Point_* suiv; /* pointeur sur l'@l@ment suivant */
} Cellule_Liste_Point;
...
typedef struct Liste_Contours_
{
        unsigned int taille;        /* nombre d'@l@ments dans la liste */
        Cellule_Liste_Contours *first; /* pointeur sur le premier @l@ment de la liste */
        Cellule_Liste_Contours *last;  /* pointeur sur le dernier @l@ment de la liste */
                                /* first = last = NULL et taille = 0 <=> liste vide */
} Liste_Contours;
...
```

Difficultés, problems et resolutions

- Faire le parcour de la liste des contours, tout en faisant le parcour de chaque contour dans le cadre de trouver le numero des contours et le numero des segments totales pour la fonction *contours_data*
    - On a decidé d'utiliser la methode de lecture par cellules. Alors on a:

```
...
 Cellule_Liste_Contours *el;
    el = c.first;
    ..
    while (el != NULL)
    {
        ..
        Cellule_Liste_Point *e;
        e = (el->data).first;
        while (e != NULL)
        {
            ..
            e = e->suiv;
        }
        ..
        el = el->suiv;
    }
...
```

      .. corespond à d'autres parties du code qui n'ont pas une connection par le probleme mentioné

- Ecriture du fichier EPS pour un multiple des contours
    - On suivre la meme methode qu'au point d'avant (lecture en utilisant les cellules).
- Le pixel voisin pendant la lecture de l'image originale pour la construction de l'image doit etre le pixel du nord par rapport le pixel en question.
    - On fait toujours hauteur-1 pour trouver le pixel de voisin

```
...
voisin = get_pixel_image(I, j, i - 1);
...
```

- Ajouter chaque contour dans la liste des contours
    - On a ecrit la fonction *ajouter_element_liste_Contours* qui ajout une liste de point sur un eliste des contours (sequence_point.c)

Extra

On trouve assez interesant le fait que pour la boucle while de l'agorithme de contours (algo_contours), on utilise une variable bool initialisé à true et qui change a false si et seulement si l'orientation du robot est Est et que ses coordones sont les memes que les coordones du point de depart.

```
...
avancer(&robot);
rb = set_point(robot.x, robot.y);
nouvelle_orientation(&robot, rb.x, rb.y, I);

            if ((robot.o == Est) && (original_position.x == rb.x) && (original_position.y ==
rb.y))
            {
                repeat = false;
            }
...
```

Tache 6 - simplification de contours par segments

Structure du programme

...

Structures de données

...

Difficultés, problems et resolutions

- .
- .
- .

Tache 7 - Simplification de contours par Bézier

Structure du programme

...

Structures de données

...

Difficultés, problems et resolutions

- .
- .
- .

Manuel utilisateur

Chaque tache du ce projet viens avec plusieurs fichiers de test qu'on ecrit specificquement pour chaque tache dans le but de repondre aux question du compte rendu ainsi que de verifier que le programe marce comme prevu.

Ceux fichiers de test sont les prgrammes principaux que l'utilisateur doit utiliser pour chaque Tache 5, 6 ou 7. Ci-dessous vous pouvez trouver un recupelatif des tous les tests qu'on a effectué pendant la duree du projet, ainsi que comment faire la compilation, comment executer le programme, quelles sont les donnees en entree et quels sont les r´esultats en sortie pour les tests des taches 5, 6 et 7.

Tests

- Tache1: test_image
- Tache2: test_geom
- Tache3: test_contour
- Tache4: test_postscript
- Tache5: test_mask
- Tache6: test_simplification
- Tache7.1: test_approx & test_degree2
- Tache7.2: test_approx3 & test_degree3
- Tache8.1: test_simplification-courbe_hilbert_7, test_simplification-courbe_hilbert_8, test_simplification-courbe_hilbert_9, test_simplification-courbe_hilbert_10, test_simplification-zebres-1000x0750, test_simplification-zebres-2000x1500, test_simplification-zebres-3000x2250, test_simplification-zebres-4000x3750(Tache 6 folder)
- Tache8.2: test_degree2-tache8, test_degree3-tache8 & test_simplification-tache8 (Tache 7 folder)

On considere qu'on est placé sur le repertoire "Final Project" qui inclus que le source code du projet.

Tache 5

Le tache 5 depend des fichiers suivantes (compilation):

1. test_mask.c
2. contour.h
3. image.h

Pour compiler il suffit d'ouvrir un terminal et se placer sur le repertoir du source code. Taper make. L'etape suivant est de taper ./*test_mask*.

Donner le nom du fichier de l'image de la quelle vous souhaitez faire l'extraction des contours.

À la compleetion du programme, il y aura deux fichiers cree dans le repertoire. Un fichier .TXT avec tous les contours et ses points ainsi qu'un fichier .EPS que vous pouvez utiiser pour visiauliser l'image en utilisant un logiciel approprié.

De plus sur le terminal vous pouvez trouver les details de cette image en ce qui cocerne le nombre des contours totales et les segments totals. Sur le terminal il y a aussi l'affichage de l'image initial et du mask de l'image (il est possible qu'il serait impossible de bien visualiser l'image et son mask sur le terminal à cause du grand nombre de l'hauteur et du largeur)

Tache 6

Le tache 6 depend des fichiers suivantes (compilation):

1. test_simplification.c
2. contour.h
3. image

Pour compiler il suffit d'ouvrir un terminal et se placer sur le repertoir du source code. Taper make. L'etape suivant est de taper ./*test_simplification*.

Donner le nom du fichier de l'image de la quelle vous souhaitez faire la simplification par segments.

Vous allez etre demandé de donner la distance seuil D qui vous souhaitez pour cette simplification.

À la compleetion du programme, il y aura deux fichiers cree dans le repertoire. Un fichier .TXT avec tous les contours et ses points ainsi qu'un fichier .EPS que vous pouvez utiiser pour visiauliser l'image en utilisant un logiciel approprié. Le nom des fichiers crées est toujours de la forme NOM_IMAGE-simp-tailleD.(eps/txt)

Sur le terminal vous pouvez voir le nombre des contours et des segments totals avant et apres la simplification par segments.

Tache 7

Pour le tache 7, il y a plusieurs programmes principaux que vous pouvez utiliser par rapport le chose que vous souhaitez faire:

- Si vous sohaitez faire une simplification par courbe de bezier de degree 2, vous devrez utiliser le programme ./test_degree2
- Si vous sohaitez faire une simplification par courbe de bezier de degree 3, vous devrez utiliser le programme ./test_degree3

Effectivement, il y aussi les programmes test_approx et test_approx3 qui etait utilisé par le programmers pour la verification des fonctions approx_bezier2 et approx_bezier3 respectivement. Dans le cadre d'un utilisateur, il aura utiliser que les programmes test_degree2 et test_degree3 pour lesquels on decrit les etapes d'utilisation ci-desosus.

test_degree2

Le test_degree2 depend des fichiers suivantes (compilation):

1. test_degree2.c
2. contour.h
3. image

Pour compiler il suffit d'ouvrir un terminal et se placer sur le repertoir du source code. Taper make. L'etape suivant est de taper ./*test_degree2*.

Donner le nom du fichier de l'image de la quelle vous souhaitez faire la simplification par segments.

Vous allez etre demandé de donner la distance seuil D qui vous souhaitez pour cette simplification.

À la compleetion du programme, il y aura deux fichiers cree dans le repertoire. Un fichier .TXT avec tous les contours et ses points ainsi qu'un fichier .EPS que vous pouvez utiiser pour visiauliser l'image en utilisant un logiciel approprié. Le nom des fichiers crées est toujours de la forme NOM_IMAGE-deg2-tailleD.(eps/txt)

Sur le terminal vous pouvez voir le nombre des contours et des segments totals avant et apres la simplification par courbes de bezier de degree 2.

test_degree3

Le test_degree3 depend des fichiers suivantes (compilation):

1. test_degree3.c
2. contour.h
3. image

Pour compiler il suffit d'ouvrir un terminal et se placer sur le repertoir du source code. Taper make. L'etape suivant est de taper ./*test_degree3*.

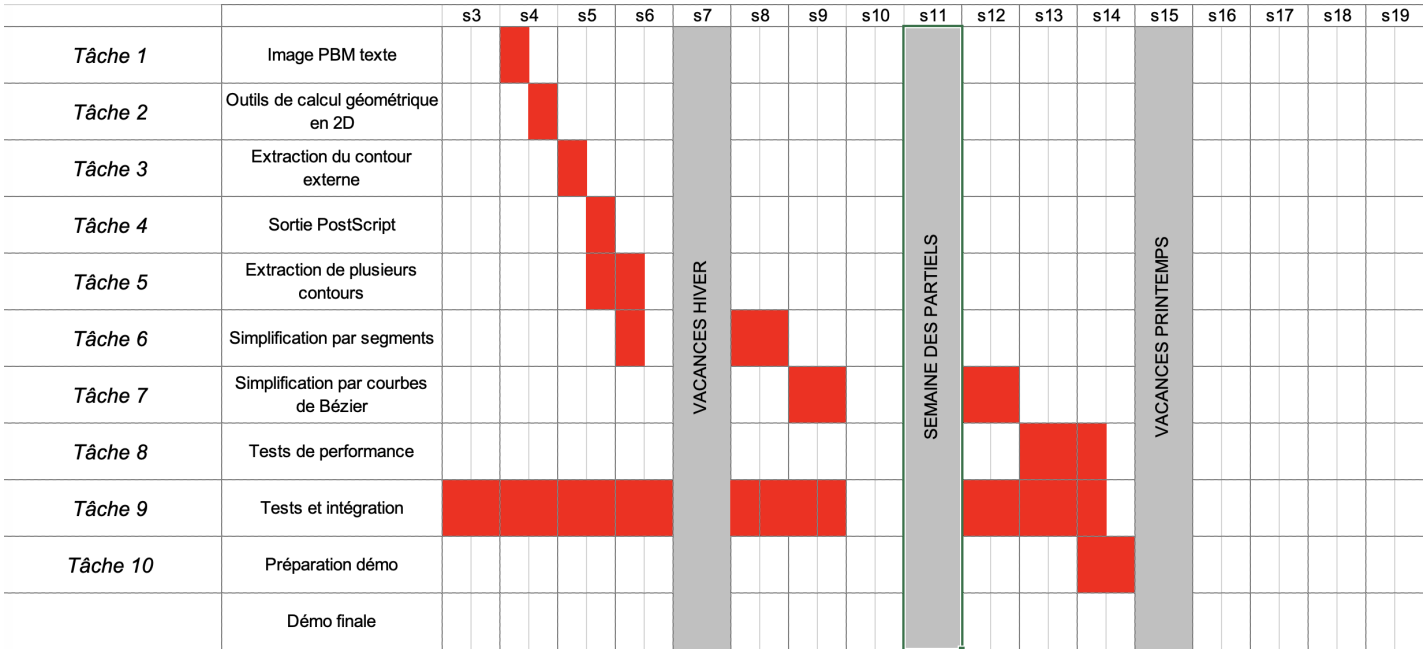
Donner le nom du fichier de l'image de la quelle vous souhaitez faire la simplification par segments.

Vous allez etre demandé de donner la distance seuil D qui vous souhaitez pour cette simplification.

À la compleetion du programme, il y aura deux fichiers cree dans le repertoire. Un fichier .TXT avec tous les contours et ses points ainsi qu'un fichier .EPS que vous pouvez utiiser pour visiauliser l'image en utilisant un logiciel approprié. Le nom des fichiers crées est toujours de la forme NOM_IMAGE-deg3-tailleD.(eps/txt)

Sur le terminal vous pouvez voir le nombre des contours et des segments totals avant et apres la simplification par courbes de bezier de degree 3.

Diagramme de Gantt



Journal de bord

Réf	Date	Problème/Information	Action/Décision	Date de réalisation prévue	Date de réalisation réelle	Etat
1	24/1/23	Utiliser des fonctions mathematiques en C	#define <math.h>	24/1/23	24/1/23	terminé
2	24/1/23	Choicir le bon typage pour les fonctions geometrqiues en plan 2	Bien comprendre qu'on doit retourner la distance, norme et produit scalaire avec le typage double	24/1/23	24/1/23	terminé
3	24/1/23	Comprendre comment recuperer les pixels d'un type image	On a utilise une variable independant "id" qui change a chaque changement ligne et collone	24/1/23	24/1/23	terminé

Réf	Date	Problème/Information	Action/Décision	Date de réalisation prévue	Date de réalisation réelle	Etat
4	24/1/23	Retouner une image negative sans modification de l'image original	Utilisation des fonctions ecrits déjà pour l'initialisation	24/1/23	24/1/23	terminé
5	24/1/23	Creation des liens entre le fichier test et le reste du programme dans le Makefile	Double verification des noms des fichiers	24/1/23	24/1/23	terminé
6	24/1/23	Initialisation de variable de parcours largeur-hauteur à 1 au lieu de 0	C'est comme ca que les fonctions sont intiliasés (par exemple: set_pixel_image)	24/1/23	24/1/23	terminé
7	27/1/23	Correction de l'image.c de la tache 1	Utilisation des fonctions ecrits tels que hauter_image etc.	27/1/23	27/1/23	terminé
8	30/1/23	Trouver la signature de l'algo de contour	Prendre une image et un nom du fichier	30/1/23	30/1/23	terminé
9	31/1/23	Correction de creation du fichier EPS par rapport l'hauteur (affichage à l'inverse)	Modification de la valeur y d'un point avec hauteur de l'image moins ca valeur initial	31/1/23	31/1/23	terminé
10	7/2/23	Correction de tache 4 comme deamndé	Set le linewidth dans le ficher EPS	6/2/23	6/2/23	terminé
11	7/2/23	Correction de l'image mask	Changer une logique pour bien deconstruire l'image mask	7/2/23	7/2/23	terminé
12	7/2/23	Correction du programme qui creer le fichier EPS pour plusieurs contours	fill devrait etre que à la fin et pas à chque contour	7/2/23	7/2/23	terminé
13	21/2/23	Ajout des tests pour le Tache6.1		21/2/23	21/2/23	terminé
14	21/2/23	Probeleme avec la concatenation des listes 1 et 2 dans l'agorythm de Douglass	On initiliase tous les lists à chaque appel recursif	21/2/23	21/2/23	terminé
15	28/2/23	Correction de l'affichage de nombre	Division par 2 de nombre des segments dans la fonction	28/2/23	28/2/23	terminé

Réf	Date	Problème/Information	Action/Décision	Date de réalisation prévue	Date de réalisation réelle	Etat
		des segments pour le Tache 6	contours_data_simplification d'affichage			
16	2/3/23	Fausse typage pour le calcul de CT	Changement de int à double	2/3/23	2/3/23	terminé
17	2/3/23	Wrong writing of courbes Béziers on the postscript file	Change the moveto to single writing in the beginning and handle the courbes of degree 2 to courbes of degree 3	2/3/23	2/3/23	terminé
18	21/2/23	Corrections on creating courbe bezier degree 3	On the last point, it was missing t*t instead of t*t	21/3/23	21/3/23	terminé
19	21/2/23	Wrong usage of int in the approx_degree2	changed typing from int to double	21/3/23	21/3/23	terminé
20	21/2/23	Error in creating .txt file for the test test_degree2	Fixed bugs on the test program	21/3/23	21/3/23	terminé
21	21/2/23	Wrong number on the of the bezier on small d	Verifying the algorythm and checking for non usage of double	21/3/23	21/3/23	terminé
22	21/2/23	There was an error in boucle logic for approx_bezier2	Fixed the boucle login for case n>1	21/3/23	21/3/23	terminé
23	21/2/23	Segmentation fault for Tache7.2	Verifying every single line of code that has been deployed on the github repository	21/3/23	26/3/23	terminé
24	28/3/23	Couldn't concatinate text with number in order to generate the final name of the exite_file (for instance when the postscript is generated for an image and it's saved on a separate file)	Usage of the command snprintf with a size of buffer strlen(name)+x	28/3/23	28/3/23	terminé
25	4/4/23	Programme faisait de "killed" dans le tache 8.1	On a changé le profil de la fonction simplification_dougal_peucker d'accepter directement un	4/4/23	4/4/23	terminé

Réf	Date	Problème/Information	Action/Décision	Date de réalisation prévue	Date de réalisation réelle	Etat
			tableau des points au lieu d'une liste chaine			