MAP401

«Rapport Final»

Shaghayegh HAJMOHAMMADKASHI , Kimia KHADEMLOU– MIN3

Tâche 1 - Image Bitmap

Solutions Envisagées

Fonctions à implémenter : Pour compléter cette tâche, il a fallu implémenter deux fonctions principales dans le fichier image.c: ecrire_image et negatif_image.

`ecrire_image`

La fonction affiche l'image à l'écran en parcourant chaque pixel et en utilisant des caractères pour représenter le noir et le blanc :

- **Boucle sur les pixels** : Itère sur chaque ligne et chaque colonne de l'image.
- **Affichage des pixels** : Utilise '1' pour les pixels noirs et '0' pour les pixels blancs.
- **Nouvelle ligne après chaque ligne d'image** : Pour passer à la ligne suivante, elle utilise un saut de ligne (`\n`).

`negatif_image`

La fonction crée une image négative où les pixels noirs deviennent blancs et vice versa :

- **Créer une nouvelle image** : Elle a les mêmes dimensions que l'originale.
- **Inverser les pixels** : Parcourt chaque pixel et utilise un conditionnel pour changer le blanc en noir et le noir en blanc.
- **Retourner la nouvelle image** : La fonction retourne cette image négative.

Structures de Données

Les structures de données utilisées pour la tâche 1 sont :

- **Structure** `Image` : Représente une image PBM. Elle comprend la largeur, la hauteur, et un tableau de pixels.
- **Type** `Pixel` : Détermine si un pixel est blanc ou noir. Les valeurs possibles sont `BLANC` (0) et `NOIR` (1).

Structure des Programmes

Pour réaliser la tâche 1, deux modules principaux ont été utilisés : image.h et types_macros.h. Ces modules ont été fournis comme base pour le travail à accomplir.

`image.h`

- Définit les types et les fonctions pour gérer des images PBM (Portable Bitmap).
- Contient des fonctions comme `creer_image`, `supprimer_image`, `get_pixel_image`, `set_pixel_image`, et `ecrire_image`.

`types_macros.h`

- Fournit des macros et des types utiles pour le projet.
- Contient des définitions communes pour faciliter le développement.

Problèmes Rencontrés :

Aucun problème majeur n'a été rencontré lors de la réalisation de cette tâche.

Tâche 2 - Géométrie 2D

Solutions Envisagées

- **Création de types de données** : Le module `geometrie2D.c` définit les types `Point` et `Vecteur`, chacun avec des coordonnées x et y (de type `double`).
- **Opérations géométriques de base** : Les opérations implémentées incluent la somme de points/vecteurs, la soustraction de points, la multiplication par un scalaire, le produit scalaire entre vecteurs, la norme euclidienne, la distance entre points, et d'autres opérations liées à la géométrie 2D.

Structures de Données

- **Type** `Point` : Représente un point en 2D, avec des coordonnées `x` et `y`.
- **Type** `Vecteur` : Représente un vecteur en 2D, également avec des coordonnées `x` et `y`.

Structure des Programmes

- **Un module unique** : Le module géométrique a été implémenté dans `geometrie2D.c`, regroupant les types et les fonctions nécessaires.
- **Programme de test** : Un programme de test a été utilisé pour valider le bon fonctionnement des opérations géométriques. Le fichier Makefile a été mis à jour pour inclure le nouveau programme.

Problèmes Rencontrés

Aucun problème majeur n'a été rencontré lors de la réalisation de cette tâche.

Tâche 3 - Extraction d'un Contour Externe

Solutions Envisagées

Les solutions envisagées pour extraire le contour externe d'une image PBM (noir et blanc) se concentrent sur le suivi du contour, le stockage des points constituant le contour, et la validation des résultats. Voici une explication plus détaillée :

Suivi du Contour

- **Orientation et déplacement** : Le suivi du contour se fait en suivant une orientation spécifique (Nord, Est, Sud, Ouest). Le code utilise des fonctions comme `avancer` pour déplacer la position d'une unité dans la direction actuelle. La fonction `nouvelle_orientation` ajuste l'orientation en fonction des voisins (pixels autour), ce qui permet au suivi du contour de s'adapter aux changements de direction.
- **Boucle de suivi** : Le suivi du contour se fait en boucle, c'est-à-dire que le code continue à avancer tant que le point actuel n'est pas revenu au point de départ avec la même orientation. Cette logique permet de s'assurer que le contour externe est suivi jusqu'à son point de départ.

Utilisation d'une Liste de Points

- **Stockage du contour** : Pour stocker le contour, le code utilise une liste de points (`Liste_Point`). Chaque point représente une position sur le contour. La fonction `memoriser_position` est utilisée pour ajouter un point à la liste chaque fois que la position est mémorisée. Cela permet de garder une trace de tout le contour au fur et à mesure que le code le suit.
- **Création et manipulation de la liste** : Le code inclut des fonctions pour créer une liste de points vide, ajouter des éléments, et supprimer la liste. Ces fonctions permettent une gestion flexible de la structure des données utilisée pour stocker le contour.

Récupération du Contour dans une Séquence de Points

- Le code utilise une liste chaînée pour stocker le contour externe. La fonction `memoriser_position` ajoute des points à cette liste, qui représente la séquence du contour.
- Des fonctions comme `creer_liste_Point_vide` et `ajouter_element_liste_Point` sont utilisées pour créer et manipuler la liste de points.

Calcul du Nombre de Segments

- Le nombre de segments dans le contour est généralement égal au nombre de points moins un. Le code peut déterminer ce nombre en comptant les éléments de la liste de points.
- Le code peut utiliser `printf` pour afficher le nombre de segments après avoir suivi le contour.

Écriture du Contour dans un Fichier Texte

- La fonction `ecrire_contour_fichier` écrit le contour dans un fichier texte avec :
    - La première ligne indiquant le nombre de contours (généralement 1).
    - Le nombre de points dans le contour et leurs coordonnées au format décimal.
- Le code parcourt la liste de points pour écrire le contour dans le fichier, assurant le stockage correct des données.

Validation avec un Programme de Test

- **Création d'images PBM simples** : Pour valider que le contour est correctement extrait, le code utilise des images PBM de petites dimensions. Ces images-tests permettent de vérifier manuellement que le contour extrait correspond au contour attendu.
- **Programme de test** : Le programme de test utilise ces images PBM pour valider le module de contour. En comparant le contour calculé avec les attentes, le programme de test aide à s'assurer que le code fonctionne correctement.

Structures de Données

Le module de contour, défini dans `contour.h`, utilise les structures de données suivantes :

- **Point** : Une structure avec des coordonnées `x` et `y` (de type `double`). Elle représente un point dans un espace 2D.
- **Cellule_Liste_Point** : Une cellule d'une liste de points, contenant un point (`data`) et un pointeur vers la cellule suivante (`suiv`).
- **Liste_Point** : Une liste de points. Elle comprend la taille de la liste, le premier élément (`first`), et le dernier élément (`last`).

Structure des Programmes

Fonctionnement général du code

- **Trouver le point de départ** : Le code utilise la fonction `trouver_pixel_depart` pour identifier le premier pixel noir, qui sert de point de départ du contour.
- **Suivi du contour** : Le code maintient une orientation (Nord, Est, Sud, Ouest) pour déterminer la direction à suivre. Il utilise des fonctions comme `avancer` pour se déplacer d'une unité dans la direction actuelle, et `nouvelle_orientation` pour ajuster l'orientation en fonction des voisins (pixels autour).
- **Liste de points** : Le contour est stocké dans une liste de points (`Liste_Point`). Chaque point représente une position sur le contour. La fonction `memoriser_position` ajoute un point à cette liste.
- **Boucle de suivi** : Une boucle continue jusqu'à ce que le contour soit complet, c'est-à-dire lorsque le point actuel revient au point de départ avec la même orientation.
- **Écriture du contour** : Le code écrit le contour dans un fichier texte avec la fonction `ecrire_contour_fichier`. Cela permet de sauvegarder le contour pour une utilisation ultérieure ou pour des tests.

Le code `contour.c` utilise les modules `image.h`, `types_macros.h`, et `contour.h` pour effectuer le calcul du contour externe. Voici un résumé de la manière dont ces modules sont utilisés :

Utilisation de `image.h`

- **Manipulation d'images** : Le code utilise des fonctions de `image.h` pour interagir avec les images PBM. Cela inclut des opérations comme `get_pixel_image` pour obtenir la valeur d'un pixel et `hauteur_image`, `largeur_image` pour obtenir les dimensions de l'image.

Utilisation de `types_macros.h`

- **Macros communes** : Ce module fournit des macros génériques et des types utiles pour le projet. Ces éléments facilitent des tâches comme le travail avec des données structurées ou la gestion d'erreurs.

Utilisation de `contour.h`

- **Structures de données** : `contour.h` définit les structures nécessaires pour travailler avec des listes de points. Ces structures sont cruciales pour stocker et manipuler le contour.
- **Prototypes des fonctions** : Ce module inclut des prototypes pour des fonctions qui créent et manipulent des listes de points, mémorisent des positions, et écrivent des contours dans un fichier.

Ces modules ensemble permettent au code `contour.c` de :

- Trouver le point de départ du contour dans une image PBM.
- Suivre le contour en tenant compte de l'orientation et des voisins.
- Mémoriser le contour dans une liste de points.
- Écrire le contour dans un fichier texte pour analyse ou utilisation ultérieure.

Problèmes Rencontrés :

Aucun problème majeur n'a été rencontré lors de la réalisation de cette tâche.

Tâche 4 - Sortie fichier au format EPS

Solutions Envisagées

Pour résoudre le problème de la tâche 4, le code vise à extraire le contour d'une image PBM et à créer un fichier EPS qui représente ce contour, avec des options pour le tracer (stroke) ou le remplir (fill). Voici les solutions envisagées pour atteindre ces objectifs :

Étape 1 : Extraction du Contour d'une Image PBM

- **Trouver le point de départ** : La fonction `trouver_pixel_depart` recherche le premier pixel noir, généralement en scannant les pixels de l'image, pour déterminer le point de départ du contour.
- **Suivre le contour** : La fonction `calculer_contour` utilise des opérations comme `avancer` et `nouvelle_orientation` pour tracer le chemin du contour. Elle mémorise les positions au fur et à mesure qu'elle avance.
- **Stockage du contour** : Le contour est stocké dans une liste chaînée de points (`Liste_Point`), ce qui facilite la manipulation et l'écriture ultérieure dans un format EPS.

Étape 2 : Création du Fichier EPS

- **Définir l'en-tête EPS** : Pour créer un fichier EPS, le code ouvre un fichier en mode écriture et écrit l'en-tête EPS, qui inclut des informations comme la version EPS et la boîte englobante (BoundingBox).
- **Tracer le contour** : En parcourant la liste de points obtenue à partir de `calculer_contour`, le code utilise des commandes EPS pour tracer le contour. Il commence par déplacer le curseur au premier point (`moveto`), puis utilise des lignes pour connecter les points suivants (`lineto`).
- **Choix du mode de tracé** : La fonction `sauvegarder_contour_eps` permet de choisir entre le mode stroke (tracé) et le mode fill (remplissage). Pour le mode stroke, elle définit l'épaisseur de trait à zéro et utilise `stroke` pour tracer. Pour le mode fill, elle utilise `fill` pour remplir l'intérieur du contour.
- **Clôture du fichier EPS** : Après avoir tracé ou rempli le contour, le code utilise `showpage` pour indiquer la fin du document EPS. Il ferme ensuite le fichier EPS pour éviter les fuites de mémoire.

Validation du Résultat

- **Test avec des images PBM** : Pour s'assurer que le contour est correctement extrait et que le fichier EPS est valide, le code peut être testé

avec différentes images PBM. Cela permet de vérifier que le tracé ou le remplissage est correct et que le fichier EPS est bien formé.

Structures de Données

- **Liste de points (Liste_Point)** : Le contour est stocké dans une liste chaînée de points. Chaque point a des coordonnées x et y (représentées par la structure `Point`).
- **Orientation** : Pour suivre le contour, le code utilise une orientation (Nord, Est, Sud, Ouest) qui détermine la direction du déplacement.
- **Image PBM** : Le code utilise des images PBM pour extraire le contour. Les pixels sont soit noirs, soit blancs, ce qui facilite l'identification des contours.

Structure des Programmes

- **Module Principal (`contour.c`)** : C'est le module central où se trouvent les fonctions pour extraire le contour d'une image PBM et le stocker dans un fichier EPS. Les principales fonctions comprennent :
  - `calculer_contour`: Cette fonction calcule le contour externe d'une image PBM, en utilisant des listes chaînées pour stocker les points du contour. Elle utilise des fonctions comme `avancer` et `nouvelle_orientation` pour suivre le contour.
  - `sauvegarder_contour_eps`: Cette fonction crée un fichier EPS à partir du contour extrait. Elle définit l'en-tête EPS, la boîte englobante, et trace le contour selon le mode choisi (stroke ou fill). Elle utilise des commandes EPS comme `moveto`, `lineto`, `stroke`, et `fill`.

- **Modules auxiliaires** :
  - `image.h`: Ce module est utilisé pour lire des images PBM et accéder aux pixels. Des fonctions comme `get_pixel_image`, `largeur_image`, et `hauteur_image` sont utilisées pour obtenir des informations sur l'image.
  - `types_macros.h`: Fournit des types communs comme `UINT` et des macros utiles. Ces définitions assurent la cohérence des données dans le programme.
  - `contour.h`: Ce module contient les structures de données et les prototypes de fonctions liés au contour. Il définit `Liste_Point`,

`Cellule_Liste_Point`, et `Point`, ainsi que des fonctions pour créer et manipuler des listes de points, comme `creer_liste_Point_vide` et `ajouter_element_liste_Point`.

Problèmes Rencontrés

d’abord, on avait l’image à l’envere, on a changé les ordonnées, et on a les images dans la bonne orientation

Tâche 5 - Extraction des contours d'une image

Solutions Envisagées

Extraction de Tous les Contours d'une Image PBM

- **Détecter les Contours** : La fonction `detecter_contours` crée une image masque à partir de l'image PBM pour identifier les points de départ des contours. Elle utilise une boucle pour trouver le premier pixel noir, marquant le début d'un contour, puis suit le contour en ajoutant les positions à une liste de points (`Liste_Point`).
- **Stocker les Contours dans un Tableau** : Chaque contour extrait est stocké dans une liste de points. Ces listes sont ensuite stockées dans un tableau dynamique pour un traitement ultérieur.
- **Compter les Contours et les Segments** : La fonction maintient des compteurs pour le nombre total de contours et de segments extraits. Cela permet de suivre le nombre de contours détectés et le nombre de segments au fur et à mesure du suivi.
- **Marquage de l'Image Masque** : Pour éviter de redétecter les mêmes contours, la fonction met à jour l'image masque pour marquer les pixels déjà visités.

Sauvegarde des Contours dans un Fichier Texte

- **Écrire les Contours dans un Fichier** : La fonction `ecrire_contours_fichier` écrit les contours extraits dans un fichier texte. Elle commence par écrire le nombre total de contours, puis parcourt chaque contour pour écrire le nombre de points et leurs coordonnées au format décimal.
- **Fermeture du Fichier** : Après avoir écrit les contours, le fichier est fermé pour garantir qu'il n'y a pas de fuite de mémoire.

Création d'un Fichier EPS avec Plusieurs Contours

- **Détecter les Contours** : La fonction `sauvegarder_contour_eps` commence par détecter tous les contours de l'image PBM en utilisant `detecter_contours`.
- **Écriture des Contours dans le Fichier EPS** : La fonction parcourt chaque contour et écrit les points dans le format EPS, utilisant des commandes comme `moveto` et `lineto` pour tracer les contours.

- **Choix du Mode (Stroke ou Fill)** : Selon le mode choisi, la fonction utilise `stroke` pour tracer le contour ou `fill` pour le remplir. Cela permet de choisir le mode de présentation des contours.

Structure des Données

`Image`

- **Description** : La structure `Image` représente une image bitmap (PBM). Elle contient des attributs comme la largeur, la hauteur, et un pointeur vers le tableau de pixels.
- **Utilisation** : Cette structure est utilisée pour lire des images PBM, accéder aux pixels individuels, et déterminer les transitions entre le noir et le blanc qui indiquent le début d'un contour.

`Point`

- **Description** : La structure `Point` représente un point dans un espace bidimensionnel, avec des coordonnées `x` et `y`.
- **Utilisation** : Les points sont utilisés pour définir des positions dans le contour extrait. Lors du suivi des contours, chaque position est stockée dans une liste de points pour représenter le contour.

`Cellule_Liste_Point`

- **Description** : Cette structure représente une cellule dans une liste chaînée de points. Elle contient un point (`data`) et un pointeur vers la cellule suivante (`suiv`).
- **Utilisation** : Chaque contour est stocké comme une liste chaînée de points. Ces cellules permettent de créer des listes où chaque élément pointe vers le suivant, facilitant l'ajout de points au fur et à mesure que le contour est suivi.

`Liste_Point`

- **Description** : La structure `Liste_Point` représente une liste chaînée de points. Elle inclut des attributs pour le premier élément (`first`), le dernier élément (`last`), et la taille de la liste.
- **Utilisation** : Cette structure permet de stocker les contours extraits. Les points sont ajoutés à la liste au fur et à mesure que le contour est suivi. Une fois le contour complet, cette liste peut être utilisée pour des opérations comme l'écriture dans des fichiers texte ou EPS.

`Orientation`

- **Description** : Cette énumération représente l'orientation lors du suivi des contours. Elle comprend quatre valeurs : Nord, Est, Sud, Ouest.
- **Utilisation** : L'orientation est utilisée pour déterminer la direction dans laquelle avancer lors du suivi du contour. Elle permet de contrôler la direction du mouvement et de définir le point suivant à visiter.

Structure des Programmes

- **Fonctions Principales** : Le code contient des fonctions pour extraire les contours, les stocker, les écrire dans des fichiers texte, et les sauvegarder dans des fichiers EPS. Les fonctions clés incluent :
  - `detecter_contours` : Extrait tous les contours d'une image PBM et les stocke dans des listes chaînées.
  - `ecrire_contours_fichier` : Écrit les contours extraits dans un fichier texte.
  - `sauvegarder_contour_eps` : Sauvegarde les contours dans un fichier EPS.
- **Boucles pour Suivre les Contours** : Le code utilise des boucles pour suivre le contour à partir d'un point de départ, en utilisant des fonctions comme `avancer` et `nouvelle_orientation` pour déterminer la direction à suivre.
- **Gestion des Structures de Données** : Le code utilise des structures comme `Liste_Point` pour stocker les contours et des tableaux pour conserver plusieurs contours. La gestion de ces structures de données est cruciale pour éviter les fuites de mémoire.
- **Validation des Résultats** : Le code comprend des étapes pour afficher le nombre total de contours et de segments, ce qui permet de valider les résultats de l'extraction des contours.

Utilisation des Modules

- `image.h` : Ce module fournit les fonctions nécessaires pour manipuler des images PBM. Il inclut des opérations pour lire des images, obtenir des pixels, et déterminer les dimensions des images.
- `types_macros.h` : Ce module contient des macros et des types communs utilisés dans tout le code. Cela inclut des types comme `UINT` et des macros pour la gestion des erreurs.
- `contour.h` : Ce module définit les structures de données et les prototypes de fonctions pour le suivi des contours. Il inclut des définitions pour des listes chaînées de points, des orientations, et des opérations de manipulation de listes.

Problèmes Rencontrés

Il semble y avoir eu des problèmes dans la fonction "Image Masque" en raison d'orientations incorrectes. De plus, la fonction "detecter_contours" avait une boucle infinie. Avec l'aide du prof et en ajoutant des traces de contour, on a pu identifier et corriger ces erreurs. À présent, le problème est résolu et le code fonctionne correctement.

Manuel

Fichiers Sources :

Pour la tâche 5, les fichiers sources principaux comprennent :

- **Fichiers C** :
  - `contour.c` : Contient les fonctions pour détecter et extraire les contours d'une image PBM, ainsi que des fonctions pour sauvegarder les contours dans des fichiers texte et EPS.
- **Fichiers d'En-tête (Header)** :
  - `image.h` : Fournit des définitions et des fonctions pour manipuler des images PBM.
  - `contour.h` : Contient les définitions de structures de données et des prototypes de fonctions pour travailler avec les contours.
  - `types_macros.h` : Comprend des macros et des types communs utilisés dans le projet.

Compilation :

Comme toutes les autres tâches, vous pouvez compiler le programme avec le commande « make ».

Exécution du Programme :

Comme indiqué dans le fichier Makefile, vous pouvez exécuter le programme avec la commande suivante : ./contourIM <nom_fichier_image.pbm> <nom_fichier_contour.eps> <mode>

Données d'Entrée :

Pour le programme, les données d'entrée comprennent :

- **Image PBM** : Un fichier bitmap en noir et blanc qui sert de base pour l'extraction des contours. Vous pouvez utiliser des images PBM fournies pour tester le programme.
- Entier mode : un entier pour choisir le mode de tracé (stroke avec 1) ou de remplissage (fill avec 2).

Résultats en Sortie :

Les résultats en sortie du programme comprennent :

- **Affichage en terminal** : le nombre total de contours extraits, le nombre de segments pour chaque contour .
- **Fichier EPS du Contour** : Un fichier EPS qui représente graphiquement les contours extraits se produit quand vous exécutez le programme.

Tâche 6 - Partie 1 : Distance Point-Segment

Solutions Envisagées

Nous avons envisagé plusieurs approches pour calculer la distance entre un point et un segment, notamment en utilisant des algorithmes géométriques et des opérations vectorielles.

Fichiers de Code Source

Code source C de la fonction de calcul de la distance point-segment :

a. **geometrie2D.c :** Contient les définitions des fonctions pour le calcul de la distance entre un point et un segment, ainsi que d'autres fonctions géométriques nécessaires.
b. **test_geometrie2D.c :** Contient le programme de test pour la fonction de calcul de la distance point-segment.
c. **geometrie2D.h :** Fichier d'en-tête contenant les déclarations des fonctions et les définitions des structures utilisées dans geometrie2D.c.

Structures de Données

La structure **Point** est utilisée pour représenter un point dans un espace 2D, avec des coordonnées x et y.
La structure **Vecteur** est utilisée pour représenter un vecteur 2D, également avec des composantes x et y.

Structure des Programmes

Fonctions Utilisées :

a. **double distance_point_segment(Point P, Point A, Point B):** Calcule la distance entre un point P et un segment défini par les points A et B.
   i. **set_point(double x, double y) :** Cette fonction crée et retourne un point à partir de ses coordonnées (x, y).
   ii. **addPoint(Point p1, Point p2) :**Cette fonction effectue la somme de deux points et retourne le résultat.
   iii. **minusPoint(Point P1, Point P2) :**Cette fonction effectue la soustraction de deux points et retourne le résultat.
   iv. **vect_bipoint(Point A, Point B) :**Cette fonction calcule et retourne le vecteur correspondant au bipoint AB (AB = B - A).
   v. **add_vect(Vecteur v1, Vecteur v2) :**Cette fonction effectue l'addition de deux vecteurs et retourne le résultat.
   vi. **reel_point(double scalaire, Point p) :**Cette fonction multiplie un point par un scalaire et retourne le résultat.
   vii. **reel_vecteur(double scalaire, Vecteur v) :**Cette fonction multiplie un vecteur par un scalaire et retourne le résultat.
   viii. **produit_scalaire(Vecteur v1, Vecteur v2) :**Cette fonction calcule le produit scalaire entre deux vecteurs et retourne le résultat.

ix. **norme_euclidienne(Vecteur v) :**Cette fonction calcule la norme euclidienne d'un vecteur et retourne le résultat.
x. **distance_points(Point A, Point B) :**Cette fonction calcule la distance euclidienne entre deux points et retourne le résultat.

Cette fonctionnalité a été implémentée dans un module unique nommé **geometrie2D.c** et **geometrie2D.h.**

Problèmes Rencontrés

Aucun problème majeur n'a été rencontré lors de la réalisation de cette tâche.

Manuel

Compilation :

La compilation est gérée par le **Makefile** en utilisant les règles définies.

Exécution du Programme :

Une fois compilé, le programme peut être exécuté en utilisant la commande ./geometrie2d .

Données en Entrée :

Les données d'entrée sont les coordonnées des **points P, A et B**, fournies par l'utilisateur via le clavier.

Résultats en Sortie :

Le résultat affiché à l'écran est la distance calculée entre **le point P et le segment [A, B].**

Tâche 6 - Partie 2 : Simplification de Contour

Solutions Envisagées :

Pour la simplification de contours, nous avons envisagé l'utilisation de l'algorithme de Douglas-Peucker, une méthode couramment utilisée pour simplifier les contours polygonaux.

Fichiers de Code Source :

Code source C pour la simplification des contours :

a. **simplification_contours_segments.c :** Contient les définitions des fonctions pour la simplification des contours en utilisant l'algorithme de Douglas-Peucker.
b. **test_simplification_contours_segments.c :** Programme de test pour évaluer la simplification des contours.
c. **image.c** : Contient les définitions des fonctions pour la manipulation des images PBM.
d. **contour.c :** Contient les définitions des fonctions pour la détection et la manipulation des contours.
e. **geometrie2D.c :** Fichier source contenant les définitions des fonctions géométriques nécessaires à la simplification des contours.
f. **simplification_contours_segments.h :** Fichier d'en-tête contenant les déclarations des fonctions utilisées pour la simplification des contours.

Structures de Données :

Les structures de données utilisées incluent des points, des tableaux de points, des listes de points et des segments.

Structure des Programmes :

Le programme est divisé en deux parties :

1. **simplification_contours_segments.c :** Contient l'algorithme de Douglas-Peucker pour simplifier les contours.
2. **test_simplification_contours_segments.c :** Programme de test pour évaluer la simplification des contours en utilisant l'algorithme de Douglas-Peucker.

Fonctions Utilisées :

a. **Liste_Point simplification_douglas_peucker(Tableau_Point CONT, UINT j1, UINT j2, double d) :** Cette fonction implémente l'algorithme de Douglas-Peucker pour simplifier un contour polygonal avec une distance-seuil donnée.
b. **detecter_contours(Image image, Liste_Point **contours) :** Cette fonction détecte les contours dans une image et les stocke dans une liste de points.
c. **sequence_points_liste_vers_tableau(Liste_Point L) :** Cette fonction convertit une liste de points en un tableau de points pour faciliter le traitement.
d. **distance_point_segment(Point P, Point A, Point B) :** Cette fonction calcule la distance entre un point et un segment, utilisée dans l'algorithme de Douglas-Peucker.

Les fonctionnalités sont réparties dans plusieurs fichiers source, chacun contenant des fonctions spécifiques. Un fichier d'en-tête correspondant est également fourni pour chaque fichier source.

Problèmes Rencontrés :

Aucun problème majeur n'a été rencontré lors de la réalisation de cette tâche.

Manuel

Compilation :

La compilation est gérée par le **Makefile** en utilisant les règles définies pour chaque fichier source.

Exécution du Programme :

Une fois compilé, le programme peut être exécuté en utilisant la commande **./simplification_contours_segments <fichier_image_pbm> <seuil>.**

Données en Entrée :

Les données d'entrée comprennent **le nom du fichier image PBM à traiter** et **la distance-seuil** pour la simplification des contours, fournies par l'utilisateur via la ligne de commande.

Résultats en Sortie :

Le programme génère **des fichiers EPS contenant les contours simplifiés pour différentes valeurs de seuil**.

Tâche 7 - Partie 1 : Simplification par courbes de Bézier de degré 2

Solutions Envisagées :

Pour simplifier les contours d'une image en utilisant des courbes de Bézier, nous avons envisagé une approche basée sur l'algorithme de Douglas-Peucker appliqué aux segments de contour, puis sur l'approximation de ces segments simplifiés par des courbes de Bézier.

Fichiers de Code Source :

a. **simplification_contours_segments.c :** Contient les définitions des fonctions pour la simplification des contours par segments de droite à l'aide de l'algorithme de Douglas-Peucker.
b. **simplification_contours_segments.h :** Fichier d'en-tête contenant les déclarations des fonctions utilisées pour la simplification des contours par courbe de bezier de degré 2.
c. **bezier.c :** Contient les définitions des fonctions pour le calcul des courbes de Bézier, y compris l'approximation de segments de droite par des courbes de Bézier.
d. **bezier.h :** Fichier d'en-tête contenant les déclarations des fonctions utilisées pour le calcul des courbes de Bézier.
e. **test_simplification_contours_segments.c :** Contient les tests pour la fonction de simplification des contours par segments de droite.
f. **test_bezier.c :** Contient les tests pour les fonctions de calcul des courbes de Bézier.

Structures de Données :

Segment : Structure représentant un segment de droite par deux points.

Bezier2 : Structure représentant une courbe de Bézier de degré 2 par trois points de contrôle.

Liste_Bezier2 : Structure de liste chainée pour stocker des courbes de Bézier de degré 2.

Structure des Programmes :

Le programme est divisé en deux parties :

1. **bezier.c :** Contient les définitions des fonctions pour le calcul des courbes de Bézier, y compris l'approximation de segments de droite par des courbes de Bézier.(**test_bezier.c)**
2. **simplification_contours_segments.c :** Contient les définitions des fonctions pour la simplification des contours par segments de droite à l'aide de l'algorithme de Douglas-Peucker.
   (test_simplification_contours_segments.c)

1. **Fonctions Utilisées :**

   a. **simplification_douglas_peucker_bezier2 :** Fonction récursive pour simplifier un segment de contour par l'algorithme de Douglas-Peucker et approximer les segments simplifiés par des courbes de Bézier de degré 2.

      Pour la fonction simplification_douglas_peucker_bezier2, voici une explication des fonctions utilisées :

      i. **creer_liste_Bezier2_vide() :** Cette fonction crée une liste vide pour stocker les courbes de Bézier simplifiées.
      ii. **approx_bezier2(CONT, j1, j2) :** Cette fonction approxime la séquence de points CONT(j1..j2) par la courbe de Bézier de degré 2 en utilisant les points de contrôle appropriés.
      iii. **distance_bezier2(CONT.tab[j], B, t) :** Cette fonction calcule la distance entre un point de la séquence et la courbe de Bézier B à un paramètre t donné.
      iv. **ajouter_element_liste_Bezier2(&L, B) :** Cette fonction ajoute la courbe de Bézier B à la liste L.
      v. **simplification_douglas_peucker_bezier2(CONT, j1, k, d) :** Cette fonction réalise la simplification récursive d'une séquence de points CONT entre j1 et k en utilisant l'algorithme de Douglas-Peucker, avec une distance-seuil d. Elle retourne une liste de courbes de Bézier simplifiées.
      vi. **concatener_liste_Bezier2(L1, L2) :** Cette fonction concatène deux listes de courbes de Bézier L1 et L2.

   b. **approx_bezier2 :** Fonction pour approximer un segment de contour par une courbe de Bézier de degré 2.
   c. **bezier2_to_bezier3 :** Fonction pour convertir une courbe de Bézier de degré 2 en une courbe de Bézier de degré 3.
   d. **lire_fichier_image(fichier_image_pbm) :** Une fonction qui charge l'image à partir du fichier PBM spécifié et retourne une structure Image contenant les données de l'image.
   e. **largeur_image(image)** et **hauteur_image(image) :** Ces fonctions retournent respectivement la largeur et la hauteur de l'image.

f. **detecter_contours(image, &contours) :** Cette fonction détecte les contours de l'image et retourne une liste de contours.
g. **sequence_points_liste_vers_tableau(contours[i]) :** Cette fonction convertit un contour représenté sous forme de liste de points en un tableau de points.

Les fonctionnalités sont réparties dans plusieurs fichiers source, chacun contenant des fonctions spécifiques. Un fichier d'en-tête correspondant est également fourni pour chaque fichier source.

Problèmes Rencontrés :

Aucun problème majeur n'a été rencontré lors de la réalisation de cette tâche.

Manuel

Compilation :

La compilation est gérée par le **Makefile** en utilisant les règles définies.

Exécution du Programme :

Une fois compilé,

- Pour le programme test_bezier :

  ./ test_bezier

- Pour le programme test_contour_bezier2 :

  ./ test_contour_bezier2 <fichier_image_pbm> <seuil>

Données en Entrée :

- Les données d'entrée pour le programme test_bezier sont les coordonnées des points fournies par l'utilisateur via la console.
- Pour le programme test_contour_bezier2, les données d'entrée attendues sont le nom du fichier image PBM à traiter et le seuil de simplification, fournis par l'utilisateur via la ligne de commande.

Résultats en Sortie :

- Pour le programme **test_bezier**, les résultats en sortie sont les coordonnées des points de contrôle de la courbe de Bézier de degré 2 qui approxime le contour polygonal.

- Le programme **test_contour_bezier2** affiche des informations sur le nombre total de contours détectés dans l'image, le nombre total de segments après la simplification par l'algorithme de Douglas-Peucker, ainsi que le nombre total de courbes de Bézier de degré 2 utilisées pour approximer ces segments simplifiés.

- Le programme **test_contour_bezier2** affiche des informations sur le nombre total de contours détectés dans l'image, le nombre total de segments après la simplification par l'algorithme de Douglas-Peucker, ainsi que le nombre total de courbes de Bézier de degré 2 utilisées pour approximer ces segments simplifiés.

Tâche 7 - Partie 2 : Simplification par courbes de Bézier de degré 3

Solutions Envisagées

Pour continuer à simplifier les contours d'une image en utilisant des courbes de Bézier, nous avons envisagé une approche basée sur l'extension de l'algorithme de Douglas-Peucker pour les courbes de Bézier de degré 3. Cela implique d'adapter l'algorithme de Douglas-Peucker pour travailler avec des courbes de degré supérieur, ainsi que de fournir une fonction de conversion des courbes de Bézier de degré 2 en courbes de degré 3 pour une meilleure approximation.

Fichiers de Code Source

a. **simplification_contours_segments.c :** Contient les définitions la fonction de simplification des contours en utilisant des courbes de Bézier de degré 3 à l'aide de l'algorithme de Douglas-Peucker.
b. **simplification_contours_segments.h :** Fichier d'en-tête contenant les déclarations des fonctions utilisées pour la simplification des contours par courbes de Bézier de degré 3.
c. **bezier.c :** Ce fichier contient les fonctions nécessaires pour manipuler les courbes de Bézier de degré 2 et 3, telles que le calcul des points de la courbe, la conversion entre les degrés, et l'approximation des courbes.
d. **bezier.h :** Fichier d'en-tête contenant les déclarations des fonctions utilisées pour le calcul et la conversion des courbes de Bézier.
e. **test_simplification_contours_ Bezier3.c :** Ce fichier est un programme de test pour vérifier la simplification des contours d'une image PBM en utilisant des courbes de Bézier de degré 3. Il charge une image PBM, détecte les contours, simplifie chaque contour, puis écrit les contours simplifiés dans un fichier EPS.
f. **test_bezier.c :** Ce fichier est un programme de test pour vérifier le bon fonctionnement des fonctions liées aux courbes de Bézier de degré 3. Il demande à l'utilisateur de saisir les coordonnées de plusieurs points, calcule une courbe de Bézier de degré 3 à partir de ces points, puis affiche les coordonnées des points de contrôle de la courbe.

Structures de Données

Segment : Structure représentant un segment de droite par deux points.

Bezier3 : Structure représentant une courbe de Bézier de degré 3 par quatre points de contrôle.

Liste_Bezier3 : Structure de liste chaînée pour stocker des courbes de Bézier de degré 3.

Structure des Programmes

Le programme est divisé en deux parties :

1. **bezier.c :** Contient les définitions des fonctions pour le calcul des courbes de Bézier, y compris l'approximation de segments de droite par des courbes de Bézier.(**test_bezier.c)**
2. **simplification_contours_ segments.c :** Contient les définitions des fonctions pour la simplification des contours en utilisant des courbes de Bézier de degré 3 à l'aide de l'algorithme de Douglas-Peucker. **(test_simplification_contours_Bezier3.c)**

1. **Fonctions Utilisées :**

    a. **simplification_douglas_peucker_bezier3 :** Fonction récursive pour simplifier un segment de contour par l'algorithme de Douglas-Peucker et approximer les segments simplifiés par des courbes de Bézier de degré 3.

    Pour la fonction simplification_douglas_peucker_bezier3, voici une explication des fonctions utilisées :

    i. **creer_liste_Bezier3_vide() :** Cette fonction crée une liste vide pour stocker les courbes de Bézier simplifiées.
    ii. **approx_bezier3(CONT, j1, j2) :** Cette fonction approxime la séquence de points CONT(j1..j2) par la courbe de Bézier de degré 3 en utilisant les points de contrôle appropriés.
    iii. **distance_bezier3(CONT.tab[j], B, t) :** Cette fonction calcule la distance entre un point de la séquence et la courbe de Bézier B à un paramètre t donné.
    iv. **ajouter_element_liste_Bezier3(&L, B) :** Cette fonction ajoute la courbe de Bézier B à la liste L.
    v. **simplification_douglas_peucker_bezier3(CONT, j1, k, d) :** Cette fonction réalise la simplification récursive d'une séquence de points CONT entre j1 et k en utilisant l'algorithme de Douglas-Peucker, avec une distance-seuil d. Elle retourne une liste de courbes de Bézier simplifiées.
    vi. **concatener_liste_Bezier3(L1, L2) :** Cette fonction concatène deux listes de courbes de Bézier L1 et L2.

b. **approx_bezier3 :** Fonction pour approximer un segment de contour par une courbe de Bézier de degré 3.
c. **bezier2_to_bezier3 :** Fonction pour convertir une courbe de Bézier de degré 2 en une courbe de Bézier de degré 3.
d. **lire_fichier_image(fichier_image_pbm) :** Une fonction qui charge l'image à partir du fichier PBM spécifié et retourne une structure Image contenant les données de l'image.
e. **largeur_image(image)** et **hauteur_image(image) :** Ces fonctions retournent respectivement la largeur et la hauteur de l'image.
f. **detecter_contours(image, &contours) :** Cette fonction détecte les contours de l'image et retourne une liste de contours.
g. **sequence_points_liste_vers_tableau(contours[i]) :** Cette fonction convertit un contour représenté sous forme de liste de points en un tableau de points.

Les fonctionnalités sont réparties dans plusieurs fichiers source, chacun contenant des fonctions spécifiques. Un fichier d'en-tête correspondant est également fourni pour chaque fichier source.

Problèmes Rencontrés

Aucun problème n'a été rencontré.

Manuel

Compilation :

La compilation est gérée par le **Makefile** en utilisant les règles définies.

Exécution du Programme :

Une fois compilé,

- Pour le programme test_bezier :

  ./ test_bezier

- Pour le programme test_contour_bezier3 :

  ./ test_contour_bezier3 <fichier_image_pbm> <seuil>

Données en Entrée :

- Les données d'entrée pour le programme test_bezier sont les coordonnées des points fournies par l'utilisateur via la console.
- Pour le programme test_contour_bezier3, les données d'entrée attendues sont le nom du fichier image PBM à traiter et le seuil de simplification, fournis par l'utilisateur via la ligne de commande.

Résultats en Sortie :

- Pour le programme **test_bezier**, les résultats en sortie sont les coordonnées des points de contrôle de la courbe de Bézier de degré 3 qui approxime le contour polygonal.

- Le programme **test_contour_bezier3** affiche des informations similaires à celles de la partie précédente, mais utilisant des courbes de Bézier de degré 3 pour l'approximation des segments de contour.

Tâche 8 - Tests de robustesse et performance, Comparatif des simplifications

Introduction

La Tâche 8 de notre projet consiste à évaluer la robustesse et les performances de nos programmes de simplification d'images, ainsi qu'à comparer différentes méthodes de simplification pour évaluer leurs résultats visuels et leurs effets sur la taille des contours.

Partie 1 - Tests de robustesse et performance :

Dans cette partie, nous avons utilisé nos programmes de simplification d'images pour tester des images complexes afin de vérifier leur bon fonctionnement et de mesurer le temps nécessaire pour chaque opération. Les étapes suivantes ont été suivies :

Extraction des contours : Pour chaque image, nous avons extrait les contours en utilisant nos programmes.

Simplification par segments : Nous avons simplifié les contours en utilisant une distance-seuil de zéro.

Observations : Nous avons enregistré le nombre total de contours, le nombre total de segments de l'image initiale et le nombre total de segments du contour simplifié pour chaque image. De plus, nous avons mesuré le temps d'exécution réel de chaque opération à l'aide de la commande système time.

Partie 2 - Comparatif des simplifications :

Dans cette partie, nous avons comparé différentes méthodes de simplification d'images pour évaluer leurs résultats visuels et leurs effets sur la taille des contours. Les étapes suivantes ont été suivies :

Extraction des contours : Pour chaque image, nous avons extrait les contours en utilisant nos programmes.

Simplification : Nous avons effectué la simplification pour chaque méthode de simplification (segment, Bézier de degré 2 et Bézier de degré 3) avec différentes distances-seuils.

Observations : Nous avons enregistré le nombre total d'éléments après simplification pour chaque méthode et distance-seuil. Nous avons également noté les caractéristiques spécifiques des résultats obtenus pour chaque méthode et distance-seuil.

Conclusion

En conclusion, nous avons observé que le choix de la méthode de simplification dépend des besoins spécifiques de l'application. Les courbes de Bézier offrent une meilleure précision, tandis que la méthode Segment par Segment peut être plus efficace en termes de simplicité et de performances. Il est important de considérer le compromis entre fidélité visuelle et complexité de représentation lors du choix de la méthode de simplification.

Recommandations :

Sur la base de nos observations, nous recommandons d'utiliser les courbes de Bézier pour les applications nécessitant une précision élevée, tandis que la méthode Segment par Segment peut être préférable pour les applications où la simplicité et les performances sont prioritaires.