
Projet : extraction des contours d'une image – Version 1

Semestre pair 2022-2023

Avertissement

Ceci est une première version du sujet qui vous permet de commencer le travail dès maintenant. Elle sera complétée plus tard avec la description d'un troisième niveau.

1 Introduction

Le but de ce projet est de manipuler les contours des objets présents dans une image en mettant en œuvre les principes vus en CTD et TP.

1.1 Évaluation

Le projet qui est décrit ci-après doit être réalisé de manière individuelle.

Le travail que vous allez produire sera évalué :

- à partir d'un dépôt sur Moodle que vous devrez effectuer au plus tard le dimanche 21 mai 2023 à 23h59 (n'attendez pas le dernier moment...); quelle que soit la raison, si le dépôt est vide le 21 mai à minuit alors vous aurez 0; il est donc conseillé de déposer votre projet avant, même s'il n'est pas fini; mais la version déposée doit pouvoir être compilée sans erreur;
- à partir d'un oral qui se déroulera :
 - le mardi 23 mai 2023 à 10h pour les groupes B11, B12, B21, B22 et B31;
 - le jeudi 25 mai 2023 à 15h45 pour les groupes A11 et A12.

1.2 Barème

Le projet comporte trois niveaux de difficulté croissante, numérotés de 1 à 3. La réalisation complète et exacte de chaque niveau donne la possibilité d'obtenir une certaine note maximale, comme cela est indiqué dans le tableau suivant :

Niveau	Note maximale
1	10
2	15
3	20

Pour commencer un niveau, il faut avoir complètement réalisé le niveau précédent. Par exemple, si vous n'avez pas terminé le niveau 2 et si vous réalisez une partie du niveau 3, vous serez tout de même noté sur 15. Un niveau est considéré comme réalisé quand toutes ses fonctionnalités sont programmées et que le programme correspondant au niveau s'exécute sans erreur.

Attention : si le projet que vous avez déposé sur Moodle ne peut pas être compilé sans erreur en utilisant la commande `make`, vous aurez automatiquement la note 0.

1.3 Dépôt sur Moodle

Vous déposerez sur Moodle un seul fichier archive au format ZIP qui doit être produit en suivant les directives suivantes.

1. Créez un répertoire dont le nom doit suivre le modèle :

NOM-PRENOM-NUMETU

où :

- NOM, PRENOM et NUMETU doivent être remplacés, respectivement et dans cet ordre, par votre nom de famille en majuscules, votre prénom en majuscules et votre numéro d'étudiant (8 chiffres) ;
- n'apparaît aucune lettre accentuée ;
- n'apparaît aucun espace ni apostrophe, les éventuels espaces et apostrophes dans votre nom ou votre prénom devant être remplacés par des tirets - (signe moins) ;

2. Placez dans ce répertoire :

- les fichiers sources des modules de votre projet, c'est-à-dire les fichiers d'extensions `.h` et `.c`, y compris `svg.h` et `svg.c` ;
- les fichiers contenant la fonction principale (`main`) de chaque niveau (`testpolygone.c` pour le niveau 1, `contex.c` pour le niveau 2) ;
- les fichiers de description des dépendances de nom `Makefilen`, où n est le numéro du niveau, permettant, grâce à la commande `make -f Makefilen`, de produire l'exécutable correspondant au niveau n ;
- un fichier de texte (obtenu avec un simple éditeur de texte) de nom `niveau-n.txt` où vous remplacerez n par le numéro du niveau que vous avez réalisé ; ce fichier pourra être vide ou contenir d'éventuelles informations destinées à expliquer les choix que vous aurez faits.

Vos fichiers sources doivent être indentés et contenir des commentaires permettant de bien comprendre votre travail. Aucun autre fichier (objets, exécutable, données, résultats, etc.) ne doit être présent dans le répertoire.

3. Créez une archive au format ZIP de ce répertoire ; cette archive doit porter le même nom que le répertoire auquel est ajoutée l'extension `.zip` ; pour produire cette archive, dans le terminal, vous pouvez vous placer dans le répertoire parent du répertoire contenant vos fichiers sources et utiliser la commande :

`zip NOM-PRENOM-NUMETU.zip NOM-PRENOM-NUMETU/*`

Respectez bien toutes ces consignes car des tests et des vérifications automatiques seront effectués sur votre projet.

1.4 Oral

Au moment de l'oral, vous devrez être prêt à :

- faire fonctionner votre programme ;
- répondre aux questions de l'enseignant ;
- effectuer des tests ou des modifications demandées par l'enseignant.

2 Niveau 1 : manipulation de polygones

2.1 Module polygone

Il s'agit d'écrire un module¹ `polygone` qui utilise le module `svg` et qui permet de gérer un polygone dont les sommets ont des coordonnées entières. Un polygone sera représenté par la structure de données suivante :

1. Pour chaque module que vous devez écrire, vous devez appliquer les principes vus en CTD concernant la programmation modulaire, notamment l'encapsulation des données et des traitements.

```

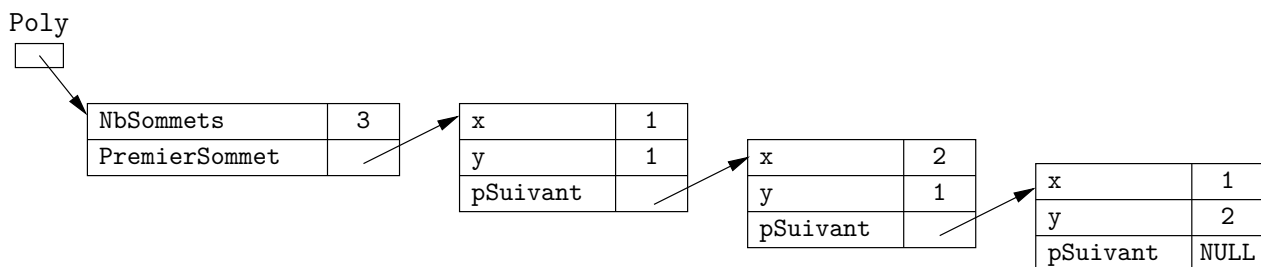
// Sommet d'un polygone
struct sSommet
{
    int x,y; // Coordonnées entières du sommet
    struct sSommet *pSuivant; // Adresse du sommet suivant
};

// Descripteur du polygone pointant vers une liste simplement chaînée de
// sommets
struct sPolygone
{
    int NbSommets; // Nombre de sommets
    struct sSommet *PremierSommet; // Adresse du premier sommet
};

// Type permettant de manipuler un polygone
typedef struct sPolygone *tPolygone;

```

Exemple : la représentation mémoire d'un triangle dont les sommets ont les coordonnées (1,1), (2,1), et (1,2) devra être :



Le module polygone devra comporter les fonctions suivantes :

- `tPolygone PolygoneCreer(void)` : crée un polygone vide ;
- `int PolygoneNbSommets(tPolygone Poly)` : retourne le nombre de sommets de Poly ;
- `void PolygoneAjouterSommetEnFin(int x, int y, tPolygone Poly)` : ajoute un sommet de coordonnées (x,y) à la fin de Poly ;
- `void PolygoneAjouterSommetEnDebut(int x, int y, tPolygone Poly)` : ajoute un sommet de coordonnées (x,y) au début de Poly ;
- `void PolygoneAjouterSommetEnIeme(int x, int y, int i, tPolygone Poly)` : ajoute un sommet de coordonnées (x,y) à la i^e position de Poly (la position d'insertion est comprise entre 0 et le nombre de sommets du polygone, 0 signifiant un ajout au début et le nombre de sommets signifiant un ajout à la fin) ;
- `void PolygoneAfficher(tPolygone Poly)` : affiche à l'écran les coordonnées des sommets du polygone à n sommets Poly sous la forme :

```

n
x1_y1
x2_y2
...
xn_yn

```

Dans le cas du triangle de l'exemple précédent, l'affichage obtenu doit être :

```

3
1_1
2_1
1_2

```

- `void PolygoneSommetIeme(tPolygone Poly, int i, int *px, int *py)` : délivre en sortie (par l'intermédiaire de `px` et de `py`) les coordonnées du sommet qui se trouve à la i^{e} position de `Poly` (la position est comprise entre 0 et le nombre de sommets moins un) ;
- `void PolygoneSommetSupprimerIeme(int i, tPolygone Poly)` : supprime le sommet qui se trouve à la i^{e} position de `Poly` (la position est comprise entre 0 et le nombre de sommets moins un) ;
- `void PolygoneLiberer(tPolygone Poly)` : libère la mémoire occupée par le polygone `Poly` ;
- `tPolygone PolygoneLiref(FILE *f)` : lit dans le fichier de texte d'identificateur `f` les coordonnées des sommets d'un polygone et retourne le polygone ; cette fonction suppose que le polygone est décrit dans le fichier selon le même format que celui de la fonction `PolygoneAfficher` ;
- `void PolygoneEcrireref(tPolygone Poly, FILE *f)` : écrit dans le fichier de texte d'identificateur `f` les coordonnées des sommets du polygone `Poly` sous le même format que celui de la fonction `PolygoneAfficher`.
- `void PolygoneEcrireSvg(tPolygone Poly, tStyle *pStyle, FILE *IdFichSVG)` : écrit le polygone `Poly` dans le fichier au format SVG d'identificateur `IdFichSVG` avec le style d'adresse `pStyle` (cf. les explications dans le fichier `svg.h` ainsi que dans la section 2.2.2 ci-dessous) ; attention, dans la fonction `SvgEcrirePolygone`, le paramètre `x` correspond aux indices des colonnes et le paramètre `y` aux indices des lignes.

Remarque : dans tous les modules, il est fortement conseillé d'écrire des fonctions « auxiliaires » qui ne seront pas publiques ; elles devront donc être de classe statique.

2.2 Test du module polygone

2.2.1 Compilation séparée

Créez le fichier source `testpolygone.c` contenant la fonction `main` afin de tester les différentes fonctions du module `polygone`.

Effectuez la compilation séparée grâce à l'utilitaire `make`. Pour cela, écrivez un fichier de description des dépendances `Makefile1` et lancez les compilations en tapant :

```
make -f Makefile1
```

2.2.2 Visualisation avec le format SVG

En plus de l'affichage des coordonnées d'un polygone grâce à la fonction `PolygoneAfficher` et de l'écriture dans un fichier de texte grâce à la fonction `PolygoneEcrireref`, vous utiliserez la fonction `PolygoneEcrireSvg` pour visualiser un polygone grâce au format SVG². Cette fonction doit faire appel au module `svg` qui permet de créer des fichiers au format SVG et d'y écrire des polygones grâce à la fonction `SvgEcrirePolygone`. La documentation des fonctions fournies est donnée sous la forme de commentaires dans le fichier d'en-tête `svg.h`.

Pour afficher une image vectorielle au format SVG, utilisez le logiciel libre multiplateforme Inkscape³ qui est installé sur les machines de TP. Certains navigateurs Web permettent d'afficher le format SVG mais la limitation du niveau de zoom ne permet pas de visualiser correctement des images de petite taille.

2. Scalable Vector Graphics : https://fr.wikipedia.org/wiki/Scalable_Vector_Graphics

3. <https://fr.wikipedia.org/wiki/Inkscape>

2.2.3 Aide à la détection de fuites de la mémoire

Afin de vérifier que votre programme n'a pas de « fuites » mémoire, c'est-à-dire gère correctement l'allocation dynamique et la libération des zones mémoires, un outil d'analyse dynamique peut vous aider. Sous Linux, vous pouvez utiliser Valgrind⁴. Par exemple, avec l'exécutable `testpolygone`, lancez l'analyse avec Valgrind en tapant⁵ :

```
valgrind ./testpolygone
```

et lisez le résultat pour savoir s'il a détecté des fuites (« *leaks* »).

Sous MacOS, tapez la commande :

```
leaks --atExit -- ./testpolygone
```

et vérifiez le nombre de « *leaks* » à la fin de l'affichage produit.

3 Niveau 2 : manipulation de contours de régions

On suppose ici que l'on dispose d'une image au format PGM qui contient le résultat d'un étiquetage en composantes connexes (régions). Une étiquette est ici représentée par un niveau de gris. Les pixels appartenant au fond de l'image, c'est-à-dire à l'arrière-plan, ont l'étiquette (le niveau de gris) 0. Les pixels appartenant aux objets ont des étiquettes comprises entre 1 et 255. Tous les pixels qui appartiennent à la même région ont la même étiquette.

Par exemple, l'image `im2.pgm` :

```
P2
18 10
255
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 255 255 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 255 255 255 0 0 0 0 0 0 0 0 0 0
0 0 0 255 255 255 255 255 255 0 127 127 0 0 127 127 0 0
0 0 255 255 255 255 255 255 0 0 127 127 0 0 127 127 0 0
0 255 255 255 255 255 255 255 0 0 127 127 127 127 127 127 0 0
0 255 255 255 255 255 0 0 0 0 127 127 127 127 127 127 0 0
0 0 0 255 0 0 0 0 0 0 0 127 127 127 127 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 127 127 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

contient deux régions : celle de droite a l'étiquette 127 et celle de gauche a l'étiquette 255.

Le contour d'une région est constitué des pixels appartenant à la région et situés sur sa bordure, c'est-à-dire ceux qui sont voisins de pixels extérieurs à la région (la notion de voisinage sort du cadre de ce projet et de l'UE).

3.1 Modules matrice et image

Récupérez les modules `matrice` et `image` que vous avez programmés lors des séances de TP.

3.2 Module contour

Il s'agit d'écrire un module `contour` qui utilise les modules `matrice`, `image`, `svg` et `polygone`. Il devra comporter les fonctions suivantes :

4. <https://fr.wikipedia.org/wiki/Valgrind>

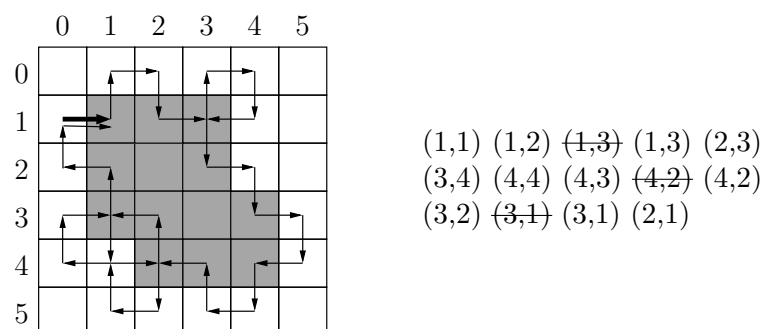
5. Si des paramètres devaient être passés à l'exécutable depuis la ligne de commande, il suffirait de les placer normalement après le nom de l'exécutable.

- `int ContourEtiquettes(char NomFichEtiquettes[], unsigned char Etiquettes[])` : extrait les étiquettes des régions d'une image d'étiquettes stockée dans un fichier au format PGM de nom `NomFichEtiquettes`, les stocke dans le tableau `Etiquettes` et retourne le nombre d'étiquettes trouvées (l'étiquette 0 du fond de l'image n'est pas considérée ici) ; par exemple, pour le fichier `im2.pgm` de l'exemple ci-dessus, cette fonction doit mettre les valeurs 255 et 127 dans le tableau des étiquettes et retourner 2.
- `tPolygone ContourSuivi(tImage Im, unsigned char Etiquette)` : effectue, dans l'image d'étiquettes `Im`, le suivi du contour d'une région d'étiquette `Etiquette` et retourne les pixels du contour de la région sous la forme d'un polygone ; ce suivi doit être réalisé par l'algorithme de la tortue de Papert qui consiste, pour la tortue qui se déplace sur le plan de l'image⁶, à :
 - commencer avec le premier pixel de la région recherchée rencontré lors d'un parcours de l'image ligne par ligne (balayage TV) ;
 - si le pixel courant est dans la région (son niveau de gris vaut l'étiquette considérée), marquer ce pixel comme faisant partie du contour et se déplacer d'un pixel vers la gauche ;
 - si le pixel courant est à l'extérieur de la région, se déplacer d'un pixel vers la droite ;
 - s'arrêter si le pixel courant est celui du départ ;
 - parcourir la liste des coordonnées des pixels trouvés et éliminer les répétitions (doublons) dues aux boucles dans le parcours.

Considérons, par exemple, l'image d'étiquettes `im0.pgm` suivante :

```
P2
6 6
255
0 0 0 0 0 0
0 166 166 166 0 0
0 166 166 166 0 0
0 166 166 166 166 0
0 0 166 166 166 0
0 0 0 0 0 0
```

L'application de l'algorithme de la tortue pour l'étiquette 166 peut être illustré par le schéma suivant :



Dans la partie gauche de cette illustration, les pixels blancs, pour plus de lisibilité, correspondent à ceux qui sont à 0 (le fond de l'image) et les pixels gris sont ceux de la région d'étiquette 166. La flèche plus épaisse que les autres représente la direction de la tortue lorsqu'elle rentre pour la première fois dans la région lors du parcours initial de l'image, ligne par ligne.

La partie droite montre les coordonnées, sous la forme (*<indice de ligne>*,*<indice de colonne>*), des pixels du contour trouvés. Les coordonnées rayées sont celles qui sont supprimées lors de la

6. La gauche et la droite ne signifient pas ici le pixel de gauche ou le pixel de droite quand on regarde l'image, mais sont relatives à la direction de déplacement de la tortue quand elle arrive sur le pixel courant.

dernière étape de l'algorithme (les doublons dus à deux passages successifs sur le même pixel du contour de la région).

Cet algorithme n'est pas parfait : son comportement n'est pas symétrique, il pose des difficultés avec les régions étroites, etc. Mais il a le mérite d'être simple.

- `void ContourExtraire(char NomFichEtiquettes[], unsigned char Etiquette[], int NbRegions, char NomFichContours[])` : localise les contours des régions, dont les `NbRegions` étiquettes sont stockées dans le tableau `Etiquette`, dans l'image d'étiquettes stockée dans le fichier au format PGM `NomFichEtiquettes` et écrit leurs coordonnées dans le fichier de texte `NomFichContours`. Le format du fichier résultat doit être le suivant :


```
NbRegions
NbSommetsRegion1
x1_y1
x2_y2
...
NbSommetsRegion2
x1_y1
x2_y2
...
```
- `void ContourEcrireSurImageSvg(char NomFichContours[], char NomFichImage[], int NbCol, int NbLig, tStyle *pStyle, char NomFichSVG[])` : écrit un ensemble de contours stocké dans le fichier `NomFichContours` sur une image stockée au format PNG dans le fichier `NomFichImage`, de largeur `NbCol` et de hauteur `NbLig`, avec le style d'adresse `pStyle`, dans le fichier au format SVG `NomFichSVG`.

3.3 Test du module contour

Créez un fichier source `contex.c` contenant la fonction `main` afin de tester les différentes fonctions du module `contour`. Créez le fichier `Makefile2` destiné à produire l'exécutable `contex` en prenant en compte les dépendances liées aux différents modules utilisés (`matrice`, `image`, `svg`, `polygone`, `contour`).

Quand vous aurez mis au point les différentes fonctions du module, modifiez le fichier `contex.c` afin de produire un exécutable dont l'appel suivant (n'oubliez pas de vérifier le nombre d'arguments reçus) :

```
./contex im2.pgm im2.png res2.ct res2.svg
```

doit écrire dans le fichier de texte `res2.ct`, au format décrit pour la fonction `ContourExtraire`, les coordonnées des pixels des contours des deux régions de `im2.pgm` et, dans le fichier `res2.svg`, le résultat permettant la visualisation suivante :

