TP03
Noté

Robin Faury

25/01/2021

Abstract

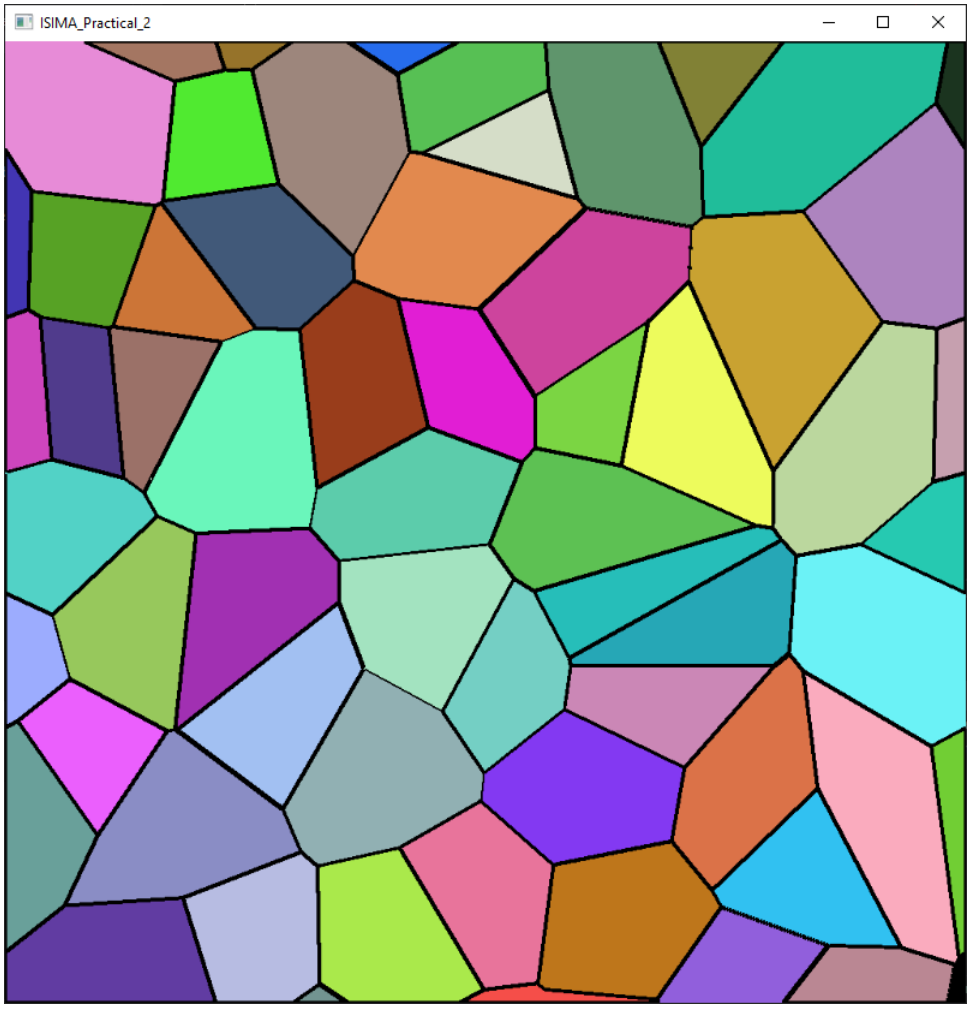

Figure 1: Segmentation d'une image

1 Introduction

1.1 Clone du projet

Les sources du projet sont accessibles à l'adresse suivante :

```
https://github.com/robinfaurypro/GPGPU_ISIMA_2020-2021.git
```

Le fichier CMakeLists se trouve dans le dossier Practical_marked. Ouvrez une console et taper les commandes suivante :

```
git submodule update --init
mkdir build
cd build
cmake -G "Visual Studio 15 Win64" ..
```

La dernière commande est à adapter si vous utilisez Visual Studio 2019 ou un système UNIX. Si tout se passe correctement vous devriez voir une image représentant un diagramme de Voronoi.

1.2 Modalité d'évaluation

Le TP se déroule sur deux séances. C'est une illustration d'un cas réel que l'on peut rencontrer dans l'industrie. Le TP est donc volontairement très long et de difficulté croissante ce qui permet d'obtenir aisément la moyenne.
Le rendu de TP doit être fait avant le 08 mars 2021 sous la forme d'un fichier zip contenant les fichiers suivant :

- Tous fichier sources ainsi que le CMakeLists si vous l'avez modifié. (Le dossier de build ne doit pas être rendu)
- Un compte rendu du TP où vous expliquerez les algorithmes que vous avez mis en place. Certain algorithme ne sont pas parallélisables et donc pas adaptable sur GPU. Il faudra aussi discuter de ces traitements dans le rapport.

Le rendu de TP compte pour 50% de la note finale de la matière.

2 Enoncé

Il vous est proposé dans ce TP d'accélérer le traitement d'image effectuée par le logiciel. Actuellement, le programme lit une image d'entrée, détecte ses contours et remplit les cellules de couleur aléatoire. Tous les algorithmes de traitements d'images sont déjà développé en CPU. Vous devez donc utiliser les technologies GPGPU pour accélérer le traitement. Il est conseillé de suivre la méthode dites Assess, Parallelize, Optimize, Deploy. Les gains de vitesse devrons être exprimé en pourcentage.
Le but du traitement proposé ici est de faire une segmentation d'image. Pour cela, nous devons détecter les arêtes de l'image puis remplir les trous obtenus avec des couleurs aléatoire.

2.1 Chargement de l'image

Dans la fonction Initialization, vous trouverez le code pour charger une image depuis le dossier de ressource. Cette image est ouverte et stocké dans un vector de la structure content nommé image_data_color_. Une texture GPU est aussi créé pour l'affichage du rendu final.

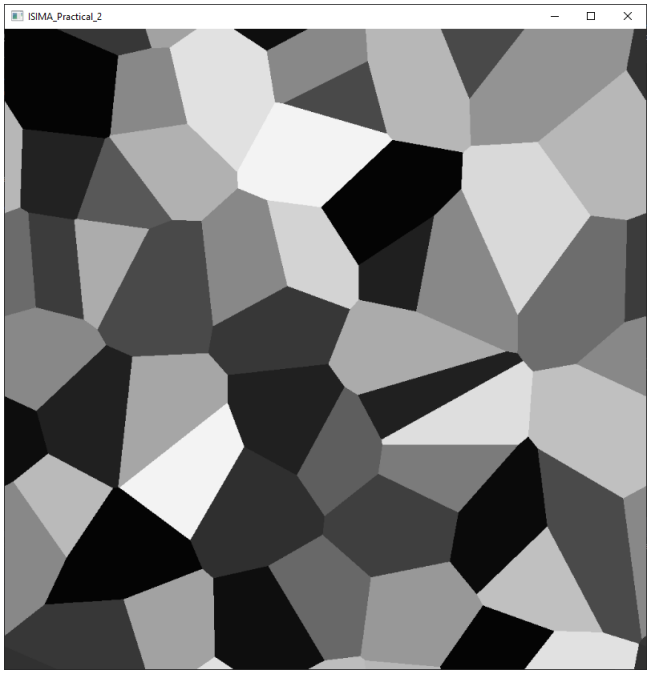

Figure 2: Affichage de l'image d'entrée

2.2 Passage en niveau de gris

Les algorithmes qui suivent ont besoin de travailler sur des textures en niveau de gris. La fonction GrayscaleConversion sert donc à remplir le tableau image_data_grayscale_. Le résultat visuel à la fin de cette étape doit être le même que pour la section précédente.

2.3 Application d'un flou

Souvent, les images d'entrée sont dites bruité ou présente de l'aliasing. Effectuer un flou aide à obtenir un résultat plus précis. Ici l'algorithme est un filtre moyenneur.

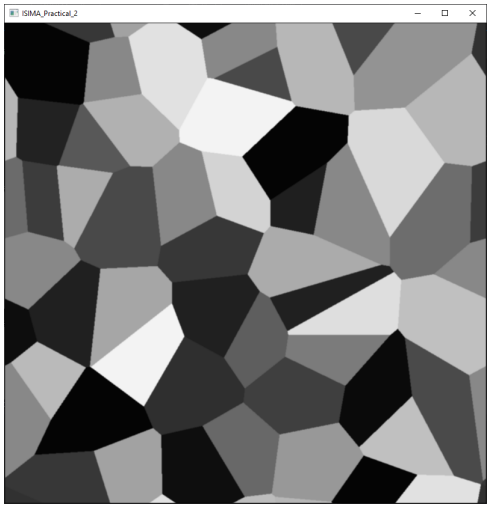

Figure 3: Image flouté

2.4 Détection de contour

La détection de contour s'effectue en utilisant un filtre de Sobel. Cela consiste à effectuer deux convolutions avec les deux noyaux de Sobel sur l'image pour obtenir un vecteur 2D décrivant le gradient de l'image.

Noyau de convolution de Sobel sur X = $\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$

Noyau de convolution de Sobel sur Y = $\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$

Le calcul de la norme de ce vecteur nous donne les contours de notre image.

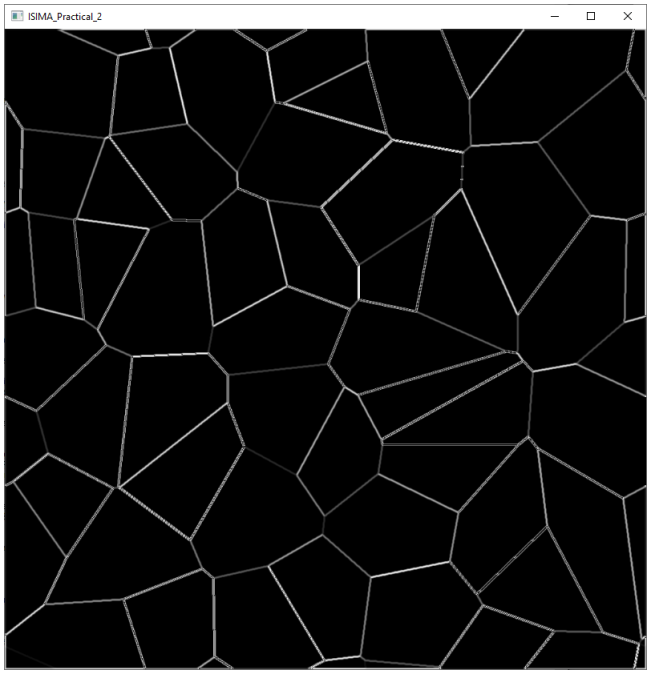

Figure 4: Coutour de l'image

2.5 Application d'un seuil

Pour rendre plus lisible les contours, nous appliquons un seuil. Les valeurs situées au-dessus sont mises à la valeur maximale alors que les valeurs au-dessous sont mises à zéro.

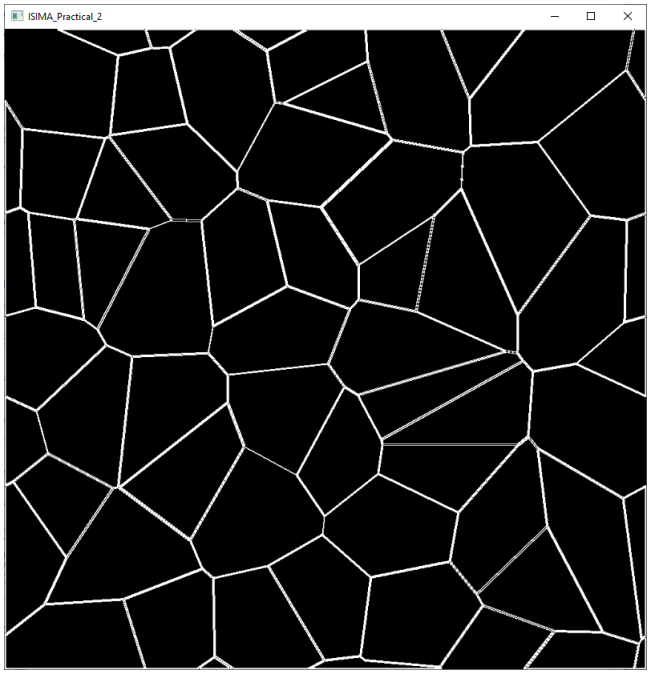

Figure 5: Image binaire

2.6 Ajout de seed

L'algorithme que nous allons mettre en place consiste à placer les graines sur l'image et à les faire grandir. Si deux graines se rencontrent, elles fusionnent. On posera donc des graines de couleur de façon aléatoire sur toute l'image couleur sauf sur les contours.

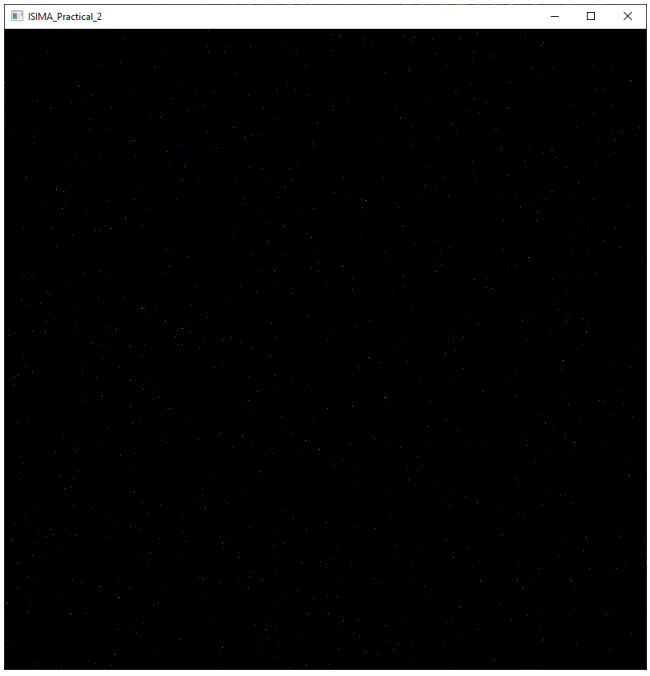

Figure 6: Image couleur avec les graines

2.7 Remplissage des cellules

Nous pouvons donc maintenant faire grossir nos cellules en utilisant l'algorithme de flood fill. Chaque graine va grossir et manger ces voisines dans chaque direction jusqu'à rencontrer un contour.

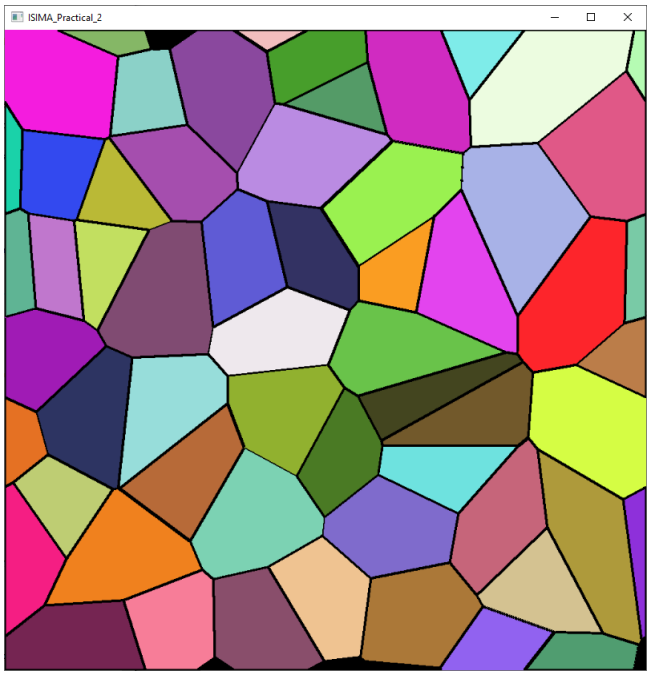

Figure 7: Algorithme de remplissage

2.8 Histogramme de l'image

L'idée maintenant est de compter les cellules. Pour cela on fera l'histogramme de l'image et l'on comptera le nombre de couleur différente.

3 Bonus

On remarquera que le nombre de cellule n'est pas le même d'une frame à l'autre. Cela est dû à l'algorithme de détection de contour. En effet le filtre de Sobel détecte un contour sur les pixels avant la transition et après la transition. Nous avons donc potentiellement des cellules créées entre les bordures. Vous pouvez utiliser la dérivée seconde (filtre de Laplace) qui donne un seul contour par bordure.