Ecole Publique d'Ingénieurs en 3 ans

Rapport

PROJET FONDEMENTS DE L'INFORMATIQUE

le 7 avril 2022,
version 0.1

PICQUE Kylian & STEIMETZ Tangui
Informaticien en herbe

kylian.picque@ecole.ensicaen.fr
tangui.steimetz@ecole.ensicaen.fr

www.ensicaen.fr

TABLE DES MATIERES

TABLE DES FIGURES

TABLE DE COULEURS

Explication des algorithmes utilisés pour la création d'une table de couleur et ses sous-tables. (Cf. *table.c* & *table.h*).

1. Les tables de couleurs

Les images tables sont des images qui contiennent un certain nombre de couleurs disposées sur une seule lignes, qui font office de références des couleurs présentes sur une image.

Dans le projet, elles sont décrites par une structure *color_table* composée : d'une part des couleurs de l'image unidimensionnelle, de son nombre de couleurs et d'autre de son statue (table original ou duplicata).

1.1. Méthode de travail sur la structure de la table de couleurs

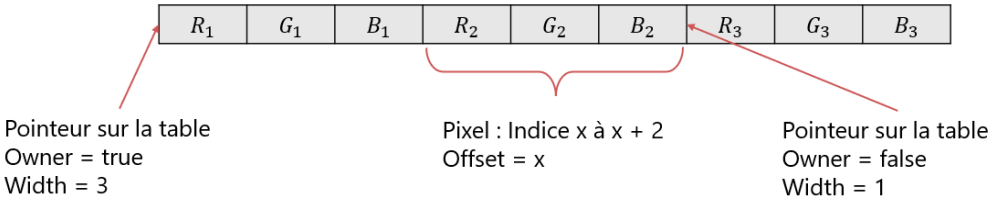

Figure 1 - Méthode de travail sur l'image unidimensionnelle

Le choix retenu a été de travailler sur un tableau unidimensionnel dans lequel les pixels sont alignés à la suite. C'est une disposition simple mais qui implique de devoirs prendre en compte que le tableau soit trois fois plus grand car les pixels sont des blocs de 3 par 3. Une autre solution aurait été de faire un tableau de pointeurs pointant vers des tableaux de taille 3, solution écartée pour éviter un nombre trop grand d'appel à la fonction *malloc*.

L'implantation des fonctions commençant par le suffixe *get* et *set* qui, respectivement, récupèrent une valeur et l'assignent à la bonne position dans la structure tout en vérifiant l'entrée. Cela permet également de travailler sur la structure *color_table* dans d'autres fichiers, notamment pour les fonctions testes.

1.2. Fonctions liées aux tables de couleurs

1.2.1. Création d'une table

La création des tables de couleurs s'est faite en trois étapes :

- 1ère étape : allocation de la mémoire pour le pointeur de la structure et initialise à 0 ou NULL le reste des variables de la structure par la fonction *init_color_table*.

- 2nd : allocation de mémoire pour la liste de couleurs de la table assigne les pixels de l'image (voir pointeur sur la table *Figure 2*).
    - Ces deux allocations ont été faites avec des tableaux dynamique afin de pouvoir modifier leur taille selon le nombre de couleurs présent dans l'image table. Cependant, il faudra la libérer correctement.
- 3ème étape : assigne les variables pour distinguer les tables entre-elles (largueur et owner).

1.2.2. Destruction d'une table de couleurs

La fonction *destroy_color_table* détruit le pointeur sur la table si ce n'est pas un duplicata, et libère le pointeur.

1.2.3. Duplication d'une table de couleurs

La fonction *color_table_duplicate* initialise une nouvelle table puis pointe vers la table de référence décalée d'un indice (voir pointeur sur la table, owner = false *Figure* 2).

1.2.4. Récupération d'un pixel dans la table de couleurs

L'algorithme de récupération de pixel pointe vers le tableau de référence à la position du pixel voulu grâce à une fonction *get,* ce qui lui permet de simplifier l'extraction du pixel selon la dimension dans laquelle on travaille (s'adapte aux images en nuances de gris).

1.2.5. Trie de la table de couleurs suivant un axe de référence

La fonction *qsort* est une fonction de trie déjà existante sur les bibliothèques standards du C. Nous avons décidé de l'utiliser par souci de d'efficacité et de simplicité. Cela nous a forcé à faire les fonctions qu'elle prend en argument pour chaque couleur (rgb) afin qu'elle puise trier selon un axe précis.

INVERSION DE COULEURS PAR METHODE TRIVIALE

2. Méthode Triviale

Explication des algorithmes utilisés pour l'inversion de couleurs par méthode triviale (Cf. *trivial_main.c*, *color_switch.c & third_party_function .c*).

2.1. Initialisation des variables de travails

On a chargé les photos (tables et images) sur des variables images avec la fonction *init_image* qui regroupe plusieurs fonctions de la bibliothèque donnée. Cette technique nous a permis de condenser le programme et de pouvoir charger plus facilement des images.

2.2. Inversion de couleurs

La fonction *color_switch_trivial* (Cf. *color_switch.c*) procède en deux étapes succinctes :

2.2.1. Recherche du plus proche voisin

La méthode de comparaison triviale charge chaque pixel de l'image originale pour déterminer le pixel de la table le plus proche de celui-ci. Pour cela elle mesure la distance entre le pixel de l'image traité avec tous les pixels de la table. Une structure (*save_pixel*) a été élaborée dans le but de simplifier la retenue du pixel le plus proche et de sa distance par rapport au pixel de référence. On a choisi la méthode la plus « triviale » possible c'est-à-dire un parcours linéaire de la table.

2.2.2. Ecriture des pixels

La fonction écrit ensuite le plus proche pixel de table, obtenu précédemment, dans une nouvelle image. Elle réitère ces étapes jusqu'à ce que l'image originale soie complétement parcourue. Enfin l'image obtenue par ce processus est convertie en photo puis enregistrée dans un dossier.

TABLE DE COULEURS PAR KDTREE

Explication des algorithmes utilisés pour la méthode de rangement des données d'une table de couleurs sous forme d'un kd arbre (Cf. *kdtree.c*).

3. Les kd arbres

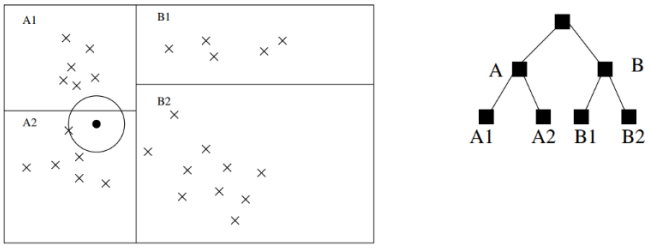

Figure 2 – Exemple de kd-arbre avec 4 feuilles. La recherche de la couleur (.) nous conduit sur la feuille A2 – créateur : BRUN Luc du document projet fondements de l'informatique.

UN **KD-ARBRE** (OU **KD-TREE**, POUR K-DIMENSIONAL TREE) EST UNE STRUCTURE DE DONNEES DE PARTITION DE L'ESPACE PERMETTANT DE STOCKER DES POINTS ET D'EFFECTUER DES RECHERCHES (RECHERCHE PAR PLAGES, PLUS PROCHES VOISINS, ETC.). C'EST UN CAS PARTICULIER D'ARBRE BSP (BINARY SPACE PARTITION TREES). - (Wikipédia - arbre kd).

3.1. Méthode de travail sur la structure du kd arbre

Les même implantations et méthodes de travails ont été utilisé que pour la table de couleurs (Cf. partie 2.1) pour les mêmes avantages et pour garder une cohérence.

3.2. Fonctions liées à la création d'un kd-arbre

Chaque étape de création du kd-arbre a été décomposé en fonctions claires et courtes pour rendre plus lisible les étapes de création. Cela implique beaucoup d'appels à des fonctions tiers et une taille du code assez conséquente.

3.2.1. Création d'un kd-arbre

C'est une fonction récursive qui fait appel à toutes les fonctions qui suivent pour assigner les valeurs nécessaires à un nœuds du kd-arbre ainsi que ses fils s'ils existent. Elle ne consiste qu'à créer un seul nœud suivant les arguments qui lui sont donnés. L'implantation de la variable statique globale *height_kdtree* permet de connaître la hauteur de l'arbre. C'est une variable utile pour l'exploration de l'arbre mais ajoute une possibilité de confusion avec une variable locale et augmente la taille du code dû au quantifieur *static*.

3.2.2. Initialisation d'un kd arbre

La fonction *init_kdtree* initialise le pointeur de la structure et met à 0 (ou NULL) le reste des variables. Elle permet d'alléger la fonction appelante (qui l'appelle) et d'utiliser NULL comme une condition d'arrêt pour l'exploration des nœuds. Mais également en cas d'erreur de voir quelle variable n'a pas été assigné.

3.2.3. Recherche de l'axe de découpe

La fonction privée au module *choice_reference_axis* récupère les taux minimums et maximums des axes présents dans une table grâce à des tris successifs afin de calculer les projections pour trouver quel axe a la plus grande.

3.2.1. Recherche du plan de coupe médian par rapport à l'axe de coupe

Les fonctions *centrales_line_sorted* et *central_line_unsorted* privées au module trouvent le point médian d'un tableau qui fait office de plan de coupe. Le préfixe *sorted* signifie qu'elle prend en argument une table de couleurs précédemment triée à l'instar de la seconde fonction pour plus de flexibilités.

3.2.2. Création des deux sous-tables suivant le plan de coupe médian

La fonction *split_color_table* duplique à l'aide de la procédure *color_table_duplicate* le tableau en deux. Les sous tables sont passées en argument par pointeurs pour ne faire qu'une fonction et moins de déclarations.

3.2.3. Assignement des kd-arbres fils au kd-arbre parent

La fonction *assign_sons* est appelée uniquement si la largeur du tableau du kd-arbre parent est supérieure au seuil. Elle demande la création d'un nouveau nœud, ce qui créé un appel récursif entre deux fonctions. C'est une façon simple de créer un arbre mais il y a un risque de faire déborder la pile et possède une complexité élevée.

3.3. Destruction d'un kd-arbre

La fonction *destroy_kdtree* libère un kd-arbre de manière récursive en commençant par ses fils gauches puis droits s'ils ne pointent pas vers *NULL*. Ensuite il se libère lui-même si c'est le premier nœud i.e. que le *owner* de la table associée soit fixé à *true*, sinon il sera détruit par son nœud parent. Cela permet d'éviter de libérer deux fois un pointeur et d'être sûr que tout l'arbre soit libéré. Elle traite en priorité les nœuds de plus bas niveau ce qui a pour avantage de libérer l'arbre de manière ordonnée.

INVERSION DE COULEURS PAR KDTREE

4. Méthode via kd arbre

Explication des algorithmes utilisés pour l'inversion de couleurs par recherche dans un arbre-kd (Cf. *kdtree_main.c* & *color_switch.c*). Seule la méthode de recherche du plus proche voisin diffère de la méthode triviale (Cf. 4.).

4.1. Recherche des plus proches voisins

La fonction *kdtree_get_nearest_color* du fichier *kdtree.c* trouve le pixel le plus proche de celui donné en argument, dit *reference_pixel*, et l'envoie dans le *buffer*. L'utilisation des variables statiques permet de conserver leurs valeurs lors de la remonté récursive. Elle suit les étapes suivantes :

- Descend dans le nœud le plus bas qui théoriquement contient le pixel le plus proche. La sauvegarde du chemin est faite par un tableau de variables statiques énumérées (gauche ou droite) alloué suivant la hauteur de l'arbre. La position du nœud courant est connue par une variable statique qui est mise à jour à chaque montée ou descente dans l'arbre. Ces combinaisons de variables permettent d'éviter d'emprunter un chemin déjà exploré lors d'un dépassement du plan de coupe (Cf. Figure 2) ;

- Une fois dans cette feuille, une fonction calcule la longueur entre le pixel recherché et ceux de la table de couleurs et la compare avec la distance de la variable *save_pixel* passé par pointeur en argument. Cette variable statique sauvegarde le pixel le plus proche parmi toutes les feuilles explorées. Cependant il faut penser à l'initialiser pour sa première utilisation.

- S'il y a besoin d'effectuer une recherche dans un nouveau nœud, on utilise le chemin sauvegardé pour explorer l'autre fils en reprenant les précédentes étapes.

COMPARAISON DES METHODES / JEU D'ESSAIS

5. Comparaison de deux facteurs

On utilise des scripts pour compiler et exécuter les différentes méthodes d'inversion de couleurs qui font guise de comparaisons et de jeu d'essais.

5.1. Mesure de l'influence du cardinal sur la méthode des kd-arbres

Afin de voir l'influence du cardinal sur l'efficacité de la méthode des kd-arbre. Nous avons décidé de créer une script bash pour répéter l'inversion d'une image à partir d'une table de couleurs. Ce script demande d'entrer le nom de l'image et de la table à étudier et réalise un graphique du temps d'inversion en fonction de la valeur du cardinal. Voici deux exemples :

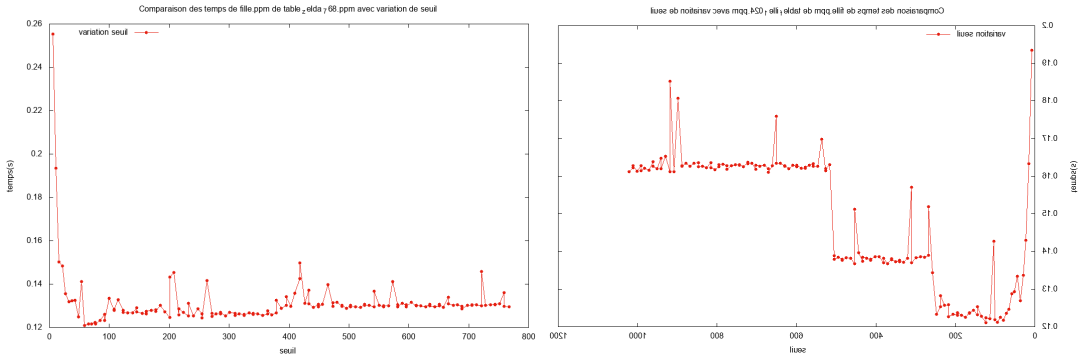

Figure 3 – Influence du cardinal sur le temps de recherche du plus proche voisin. Inversion de fille.ppm par rapport à la table table_fille_1280.ppm puis par rapport à table_lenna_512.ppm

On peut constater que le cardinal exerce une influence sur le temps d'exécution du programme. On observe que les courbent semblent converger ou font des paliers. On peut donc se servir du cardinal le plus adapté en fonction de l'image à inverser et sa table de couleurs.

5.2. Mesure de l'efficacité des deux méthodes

On a cherché à comparer le temps d'inversion des deux méthodes à travers un script bash. Celui-ci demande le nom d'une image pour l'inverser avec toutes les tables de couleurs fournies dans le répertoire de tables. Voici deux exemples :

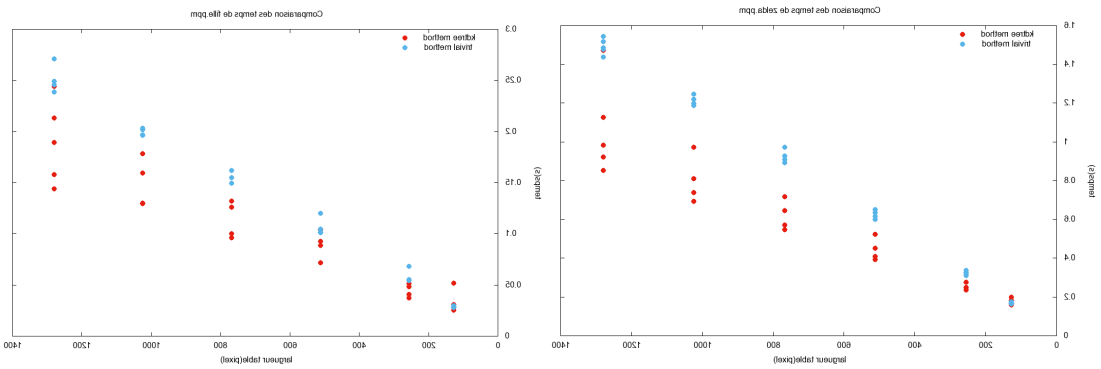

Figure 4 – Comparaison des temps de recherche du plus proche voisin sur l'image zelda.ppm et fille.ppm par rapport à toutes les tables disponibles.

On peut noter que la méthode du kd-arbre est plus rapide que la triviale. Mais le temps de recherche pour la méthode triviale est régulier à l'instar du kd-arbre qui dépend de la composition des couleurs de la tables.

5.3. Les fichiers de test des structures de travail

On essayer de tester nos fonctions qui s'appliquent sur les tables de couleurs et les tables (Cf *test_tables* & *test_kdtree*).

5.3.1. Les tests des tables de couleurs

Nous avons d'abord voulu voir si la structure *color_table* était fonctionnelle en lui affectant des valeurs issues de la photo table_fille_128.ppm. Cela a permis par la même occasion de tester les fonctions d'initialisation, de duplication et de trie. Nous avons fait le choix d'afficher sur le terminal les valeurs plutôt que de les comparer car il est compliqué de créer une référence.

5.3.2. Les tests des kd-arbres

De la même manière que pour les tables nous avons mis à l'épreuve les fonctions liées aux kd-arbres. Nous avons imposé un cardinal arbitraire qui semblait le mieux fonctionner de façon empirique.

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Ecole Publique d'Ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04

ministère éducation nationale enseignement supérieur recherche

Normandie Université

Cti Commission des titres d'ingénieur

cdefi Conférence des Directeurs des Écoles Françaises d'Ingénieurs

European Accreditation of Engineering Programmes EUR-ACE®

CONFÉRENCE DES GRANDES ÉCOLES