Rapport - Projet PRIM 2022

Loïc XU

08/01/2023

Table des matières

1 Sujet du projet

Le but de ce projet est d'implémenter un interpréteur pour un petit langage graphique. L'interpréteur devra lire sur l'entrée standard (stdin) un fichier qui a sur sa première ligne un entier représentant la taille de l'image à produire ainsi qu'en deuxième ligne une suite de caractères dont certains représentent des actions à faire. Ces actions permettront à la fin d'avoir une image au format PPM. Cette image en question sera retournée sur la sortie standard (stdout). L'idéal serait que le programme qu'on nommera exec puisse afficher l'image associée au fichier test.ipi grâce à la commande suivante : ./exec < test.ipi | display

L'image sera calculée à partir d'une machine à états. Un état de cette machine sera constitué de :

— deux entiers indiquant la position courante du curseur
— deux entiers indiquant la dernière position marquée
— un élément qui indique la direction du curseur (Nord, Est, Sud, Ouest)
— un multiensemble de couleurs qui représente le seau dans lequel sont mélangées des doses de couleurs pour produire la couleur courante

— un multiensemble d'opacités qui représente le seau dans lequel sont mélangées des doses d'opacités pour produire l'opacité courante
— une pile de calques de taille au plus 10, chaque calque étant une grille de pixels de la taille de l'image à produire.

Initialement, l'état de la machine est :
— les positions courante et marquée sont toutes les deux en (0,0)
— la direction est Est
— les seaux de couleurs et d'opacités sont vides tous les deux
— la pile contient un seul calque ; chaque pixel de ce calque est formé de la couleur (0,0,0) et de l'opacité 0

Certains caractères spécifiques pourront modifier l'état de la machine :
— n : Ajoute la couleur noire (c'est-à-dire (0,0,0)) dans le seau de couleurs
— r : Ajoute la couleur rouge (c'est-à-dire (255,0,0)) dans le seau de couleurs.
— g : Ajoute la couleur verte (c'est-à-dire (0,255,0)) dans le seau de couleurs.
— b : Ajoute la couleur bleue (c'est-à-dire (0,0,255)) dans le seau de couleurs.
— y : Ajoute la couleur jaune (c'est-à-dire (255,255,0)) dans le seau de couleurs.
— m : Ajoute la couleur magenta (c'est-à-dire (255,0,255)) dans le seau de couleurs.
— c : Ajoute la couleur cyan (c'est-à-dire (0,255,255)) dans le seau de couleurs.
— w : Ajoute la couleur blanche (c'est-à-dire (255,255,255)) dans le seau de couleurs.
— t : Ajoute l'opacité transparente (0) dans le seau d'opacités.
— o : Ajoute l'opacité complète (255) dans le seau d'opacités.
— i : Vide les seaux de couleurs et d'opacité.
— v : Avance la position du curseur d'un pas dans la direction courante ; si le bord de l'image est atteint, on repasse de l'autre coté.
— h : Tourne la direction courante dans le sens horaire.
— a : Tourne la direction courante dans le sens anti-horaire.
— p : Met à jour la position marquée en y mettant la position actuelle du curseur.
— l : Trace une ligne entre la position du curseur et la position marquée
— f : Remplit la zone de même couleur autour du curseur, en la remplaçant par la couleur courante
— s : Ajoute un nouveau calque dans la pile de calques ; chaque pixel de ce nouveau calque sera formé de la couleur (0,0,0) et de l'opacité 0 ; s'il y a déjà 10 calques dans la pile, ne fait rien.
— e : Fusionne les deux calques les plus hauts dans la pile ; il y aura donc un calque de moins dans la pile ; s'il n'y a qu'un seul calque, ne fait rien.
— j : Découpe le calque situé juste en dessous du sommet de la pile en utilisant comme masque les opacités de celui au sommet ; retire celui au sommet ; s'il n'y a qu'un seul calque, ne fait rien.
— On ne prendra pas en compte tous les autres caractères ainsi qu'aucune majuscule.

La plupart des algorithmes ci-dessus est donnée, il faudra alors les implémenter.

Une fois le fichier en entrée complètement lu, on prendra le calque situé en haut de la pile pour obtenir l'image finale. On ne prendra pas en compte les opacités pour l'image finale.

2 Implémentation

2.1 Définition des types

La première chose que nous allons faire est de définir les types qu'on va utiliser par la suite dans un module nommé etat_de_machine_initial.

Le sujet nous donne des indications :

— Une composante est un entier non signé sur 1 octet, (j'ai choisi un unsigned char au vu de l'opacité qui est entre 0 et 255).

— Une couleur est formée de trois composantes pour le rouge, le vert et le bleu (on va faire un enregistrement couleur de trois composantes (r,g,b)).

— Une opacité est donnée par une composante ; 0 correspond à la transparence totale, tandis que 255 correspond à l'opacité totale.

— Un pixel est formé par une couleur et une opacité (on va faire un enregistrement pixel avec une couleur et une opacité)

— Un calque est une grille carrée de pixels, de la taille de l'image à produire au final (on va prendre un tableau 2D de pixel, le type calque sera donc un pixel**)

J'ai aussi défini les types suivants :

— pour le seau de couleur : une liste chaînée composée de maillons (enregistrement d'une couleur et d'un pointeur de maillon qui pointe vers le prochain maillon), car on ne sait pas combien de doses de couleurs vont être mis dedans surtout avec des gros fichiers comprenant beaucoup de caractères. Un tableau statique n'était alors pas optimisée car si trop petit, on ne pourra pas tout stocker, si trop grand, on gâche de la mémoire (on aurait pu prendre un tableau dynamique avec redimensionnement de la taille à chaque ajout mais cela semblait moins naturel qu'une liste chaînée.

— pour le seau d'opacité : de même mais avec des maillons composés d'une opacité et d'un pointeur

— pour les coordonnées permettant de se repérer sur les calques : un enregistrement coordonne de deux entiers x et y

— pour la direction : une énumération pour pouvoir utiliser switch dessus (switch ne fonctionne pas avec les string, on aurait aussi pu utiliser le première caractère de chaque point cardinal)

— pour la pile de calques : un enregistrement pile avec un tableau statique puisque sa taille maximale est imposée à 10, et avec cela un entier "top" qui est l'indice du sommet de la pile.

2.2 Initialisation de l'état de la machine

Avant d'initialiser l'état de la machine, il nous faut d'abord la taille de l'image. On sait qu'on peut la récupérer grâce à la première ligne du fichier en entrée standard. On déclare donc dans la fonction main un entier nommé taille_image qui stockera la taille de l'image. Pour cela on implémente une fonction recup_taille_image qui ne prend aucun argument et retourne la taille de l'image (dans le module vu précédemment), on utilise la fonction fgets qui permet de récupérer la première ligne du fichier à tra-

vers un buffer, puis on utilise sscanf pour affecter la valeur en question dans taille_image.

Une fois obtenue, on va créer une fonction creer_calque_initial (dans le module vu précédemment) qui prend en argument la taille de l'image trouvée grâce à la fonction précédente et retourne une variable de type calque dont chaque pixel est de couleur (0,0,0) et d'opacité 0.

On définit aussi deux variables de type coordonnee qui sont la pos_courante et la pos_marquee qu'on initialise tous les deux à (0,0), un cardinal direction définit à Est, une liste chaînée de couleurs pour le seau de couleurs et une liste chaînée d'opacités pour le seau d'opacités égales au pointeur NULL. On initialise aussi notre pile initial de type pile avec à l'intérieur un calque dont chaque pixel est de couleur (0,0,0) et d'opacité 0 et pour l'indice du sommet on le met à 0.

2.3 Fonctions utilisées pour les sceaux

Maintenant, que l'état de la machine est initialisé, il faut faire les fonctions qui peuvent modifier ce dernier qu'on implémentera dans le module fonction_sceau. Commençons par les fonctions pour les seaux, on implémente la fonction cons_couleur qui prend en argument le seau de couleur et une couleur à ajouter, elle retourne le seau de couleur avec la couleur à ajouter au sommet, pour cela on créer un nouvel maillon qui pointe vers l'ancien sommet du seau de couleur. On fait de même avec cons_opacite pour le seau d'opacités. Il y a aussi une procédure qui vide les seaux par effet, il suffit de donner en argument l'adresse du seau de couleur et du seau d'opacité. On n'oubliera pas de free les maillons avant.

2.4 Fonctions utilisées pour les pixels/calques

On passe maintenant au plus gros module fonction_pixel.

La première fonction est la fonction de calcul du pixel courant qui prend en argument le seau de couleurs et le seau d'opacités et retourne un pixel calculé à partir des seaux. La formule étant donnée dans le sujet, on l'implémente. J'ai eu une grosse difficulté sur cette fonction qui paraît assez simple. En effet, si pour la moyenne des composantes de couleur, on fait

$$\frac{somme.des.composantes.de.chaque.maillon}{longueur.de.la.chaine}$$

On a un problème puisque par exemple dans le cas du rouge et jaune cela donnerait pour la composante rouge : $\frac{255+255}{2}$. Or, puisque c'est un unsigned char cela fera $255 + 255 = 254$ puis diviser par deux ie 127. Bien entendu, la moyenne de 255 et 255 est 255. Pour résoudre ce problème, j'ai utilisé trois variable int pour faire les calculs . J'ai ensuite affecté à mes composantes ces trois variables.

La deuxième fonction est la fonction qui permet au curseur d'avancer d'une unité. Ici, l'énumération de la direction est pratique puisqu'on peut directement faire un switch

sur la direction. J'ai eu deux problèmes pour cette fonction, le premier est que je pensais que le (0,0) se trouvait en bas à gauche comme en mathématiques et donc que l'ordonnée allait vers le haut. Je me suis vite rendu compte de l'erreur lorsque j'ai testé mon programme sur le fichier "simple.ipi". Le second est par rapport à l'opérateur "%", je pensais que cela retournait le reste de la division euclidienne comme un modulo, mais j'ai été surpris de voir des entiers négatifs pour mes coordonnées après l'utilisation de celui-ci. Une recherche sur Google m'a éclairé et j'ai donc alors fait ma propre fonction modulo qui retourne le reste de la division euclidienne. Ceci m'a été utile pour passer de l'autre côté lorsque mon curseur était au bord.

Ensuite, nous avons les fonctions pour tourner le curseur dans le sens horaire et anti-horaire. De même, j'ai utilisé un switch sur la direction pour adapter les 4 cas possibles. Puis, il y a une fonction pour mettre à jour la position marquée.

Par ailleurs, pour faire une image, il faut colorier les pixels, d'où la fonction pour tracer une ligne entre la position courante et la position marquée. J'ai suivi l'algorithme donné dans le sujet. Pour la fonction remplissage, qui fonctionne comme sur les éditeurs d'images classiques, la version donnée dans le sujet est récursive. Elle fonctionne pour les petites images mais entraîne une erreur de segmentation pour les grandes images. Pour pallier ce problème, j'ai implémenté une pile statique de coordonnées de taille maximale 1 000 000 qui stocke les coordonnées voisins à traiter. On traite le sommet de la pile et on continue jusqu'à qu'elle soit vide. Je gâche beaucoup de mémoire en faisant ceci (a priori, moins qu'avec la version récursive puisqu'elle fonctionne sur base.ipi, là où la fonction récursive échoue). J'ai tout d'abord tenté avec une pile dynamique type liste chaînée ou avec un tableau dynamique. Cependant, j'avais une erreur de segmentation à chaque fois que je lançais le programme et je n'ai toujours pas trouvé l'erreur liée à ça, j'ai donc préféré mettre un tableau statique avec une grosse capacité et un entier pour l'indice du sommet pour avoir un programme exécutable, pourquoi 1 million ? C'est pour tenter de réussir à remplir potentiellement une image de taille 1000 pixels si jamais l'occasion se présente dans un des fichiers ipi qu'on a pas eu. L'implémentation des piles de coordonnées est dans le module pile_coordonnee.

Il reste des fonctions de fusion et de découpage de calques. J'ai juste implémenté en suivant les algorithmes du sujet.

2.5 Format d'entrée

On récupère les caractères un par un avec la fonction getc qui retourne un caractère ou EOF en cas de fin de fichier ou d'erreur. On utilise une boucle while qui continue à récupérer les caractères jusqu'à la fin. (Elle termine a priori car sinon le fichier aurait une taille infinie.) Puis on applique un switch à chaque tour de boucle et selon le caractère, on exécute une fonction implémentée. Bien entendu, puisqu'on a pas fait de cas pour les caractères indésirables et majuscules, on ne les traite pas. La fonction feof vérifie qu'on est bien en fin de fichier en sortie de boucle et non pas qu'on a eu une erreur car getc peut retourner EOF sur une erreur.

2.6 Production de l'image finale

On renvoie l'image au format PPM. Un fichier PPM est structuré de telle manière :
P6
longueur largeur
valeur du pixel (généralement 255)
longueur x largeur pixels

Il nous suffit donc de fprintf et fwrite sur la sortie standard (stdout).

2.7 Remarques

./exec < test.ipi | display ne fonctionne pas de mon côté car je n'arrive pas à créer le raccourci display.

J'ai donc fait de mon côté : ./exec < test.ipi
pour afficher sur la sortie standard
et :
./exec < test.ipi > image.ppm
pour obtenir un fichier image.ppm et l'ouvrir manuellement

Pour vérifier les fichiers d'exemple, j'ai utilisé "diff -s test.ppm image.ppm" sur le terminal linux.