Vector text-based editor

LEMOINE Mathurin
DING Chengfeng

2023 – Rentrée décalée

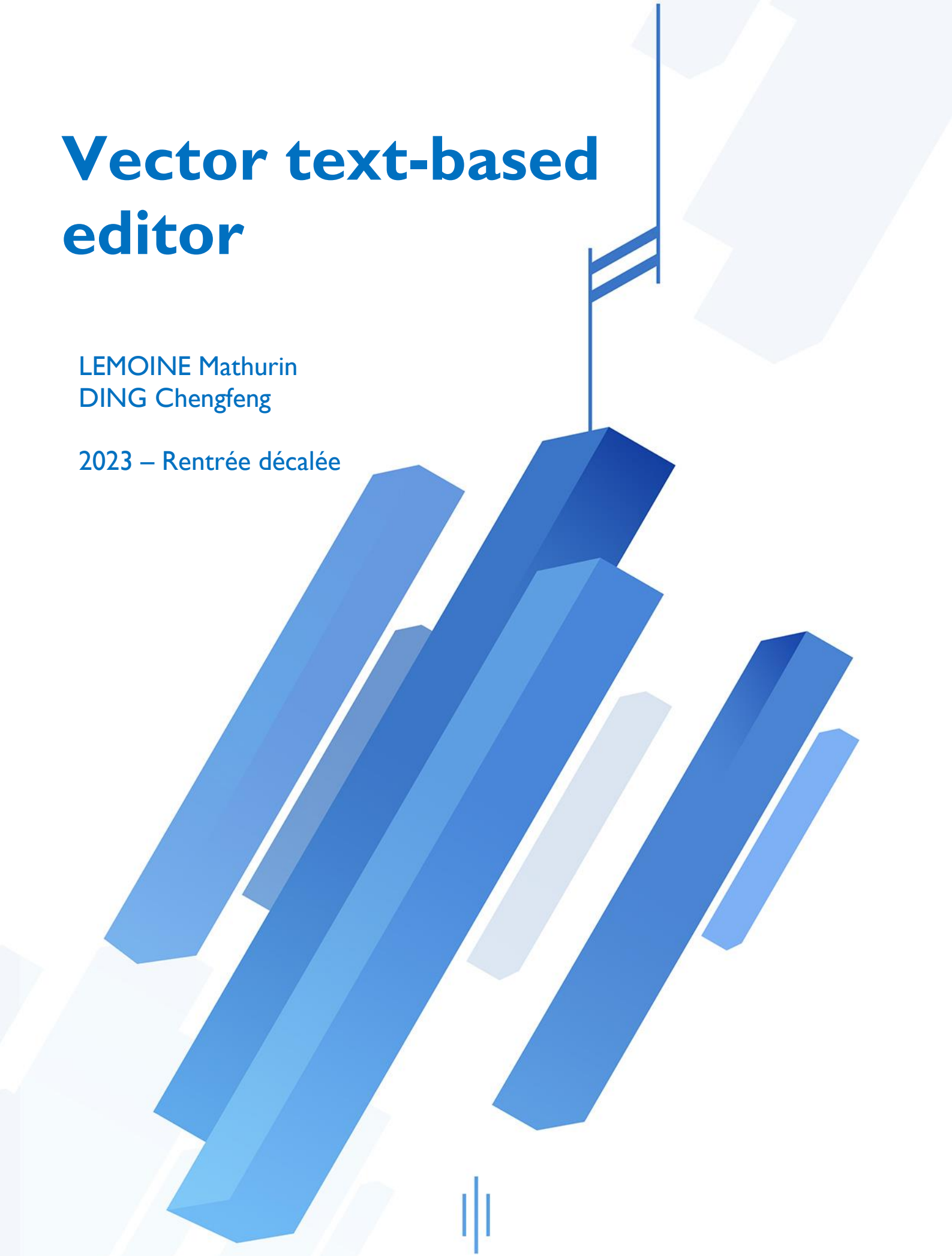

Table des matières

Introduction

L'objectif de ce projet est de développer une application de dessin vectoriel, basée sur un mode textuel. Nous devons offrir aux utilisateurs une expérience intuitive et pratique pour la création d'images vectorielles simples. Le code peut réaliser des dessins en utilisant un ensemble de commandes spécifiques, tout en affichant les résultats sous forme de texte.

Le projet s'est axé sur trois aspects fondamentaux :

- Gestion de la mémoire et des données : Nous devons élaborer une représentation efficace des informations nécessaires à notre application. Il est crucial de déterminer comment stocker les données relatives aux formes géométriques créées par les utilisateurs, de manière à pouvoir les récupérer et les afficher ultérieurement de manière fiable et précise.

- Interface en ligne de commandes : Nous devons fournir aux utilisateurs une interface conviviale qui leur permettra d'effectuer diverses actions autorisées par l'application, telles que la création de formes, la superposition de plusieurs formes et la modification des formes présentes.

- Affichage en mode textuel : L'une des caractéristiques clés de notre application est de représenter visuellement les images vectorielles créées par les utilisateurs à l'aide de caractères textuels (# et .). Nous devons mettre en œuvre les algorithmes de dessin appropriés pour afficher ces images de manière claire et esthétique, en utilisant des techniques adaptées à la représentation en mode texte.

Présentation des différents modules du projet

Les structures de données des différentes formes

Dans cette partie, l'objectif est de définir comment stocker un point, une ligne, un carré, un rectangle, un polygone et un cercle et par la suite définir une structure générique (void *) « shape » qui pointera vers un type de forme existant, tout en évitant d'avoir à reécrire le même code pour chaque type de forme.

En effet, le type void * peut donc pointer vers n'importe quel type de variable (ici : line, circle, square etc...). A noter qu'il faut bien faire attention à caster le type de pointeur retourné par une fonction void * pour pouvoir l'utiliser correctement.

```c
typedef struct {
    Point * center;
    int radius;
} Circle;

Circle * create_circle(Point * center, int radius);
void delete_circle(Circle * circle);
void print_circle(Circle * circle);
```


Etudions maintenant un exemple de structure de donnée pour le cercle :

Le cercle est ici caractérisé par un centre qui prend le type point (lui-même composé de deux entiers qui représentent sa position en x et en y). En plus de ce centre, il faut aussi un entier qui correspond au rayon.

Ensuite, pour chaque forme, il faut aussi penser à la façon dont nous allons créer, afficher et supprimer ce type de forme via des fonctions.

Pour résoudre le défi de supprimer une forme dont le type générique est inconnu à l'avance, nous avons opté pour l'utilisation d'un bloc de contrôle "switch case". Cette décision s'est avérée évidente car elle présente plusieurs avantages : lisibilité, polyvalence et efficacité.

Le bloc de contrôle "switch case" permet de comparer une expression à une liste de cas possibles et d'exécuter le code correspondant au cas qui correspond à l'expression. Dans notre cas, l'expression sera le type générique de la forme que nous souhaitons supprimer.

De plus, cette approche est polyvalente car elle peut être facilement étendue pour prendre en charge différents types de formes. Chaque cas dans le bloc "switch" peut être ajouté ou modifié pour gérer un type spécifique de forme, ce qui rend le code adaptable et flexible.

```c
void print_shape(Shape *shape) {
    printf("SHAPE : ID = %d\n", shape->id);
    switch (shape->shape_type) {
        case POINT:
            print_point(shape->ptrShape);
            break;
        case LINE:
            print_line(shape->ptrShape);
            break;
        case SQUARE:
            print_square(shape->ptrShape);
            break;
        case RECTANGLE:
            print_rectangle(shape->ptrShape);
            break;
        case CIRCLE:
            print_circle(shape->ptrShape);
            break;
        case POLYGON:
            print_polygon(shape->ptrShape);
            break;
    }
}
```

Enfin, l'utilisation du "switch case" peut également être efficace en termes de performances. La comparaison directe du type générique de la forme avec les cas spécifiques évite les opérations de vérification qui surchargeraient le code et l'exécution du code.

Affichage de l'image à l'écran

La zone de dessin

Dans ce module, il faut créer une zone de dessin à l'écran pour ensuite y afficher les formes. Pour ce faire, nous créons une matrice (ou tableau 2D) dans lequel chaque valeur correspond à un pixel de coordonnées (x;y)

```c
struct area {
    unsigned int width; // Nombre de pixels en largeur ou nombre de colonnes ( axe y )
    unsigned int height; // Nombre de pixels en hauteur ou nombre de lignes ( axe x )
    BOOL ** mat; // Matrice des pixels de taille ( width * height )
    Shape * shapes[SHAPE_MAX]; // Tableau des formes
    int nb_shape; // Nombre de formes dans le tableau shapes ( taille logique )
};
typedef struct area Area;
```


On définit donc la structure de données area avec une taille de dessin, une matrice 2D d'entiers (le type bool est défini comme un entier mais permet de bien décrire que l'on attend une valeur binaire : 0 ou 1). Ensuite, dans cette zone de dessin, il faut aussi stocker toutes les formes associées, d'où la présence d'un tableau de pointeurs vers des formes accompagné du nombre de formes pour connaître la taille du tableau précédent.

La représentation des formes

Une fois la zone de dessin créée, il faut maintenant, pour chaque forme, remplir la matrice avec les pixels de chaque forme pour ensuite la représenter et la montrer à l'utilisateur.

Comme chaque forme possède différents paramètres et méthodes de construction, nous devons donc créer une fonction par forme.

Ici, prenons l'exemple de la fonction pour le polygone :

```c
void pixel_polygon(Shape *polygon, Pixel ***pixel_tab, int *nb_pixels) {
    Polygon *p = (Polygon *) polygon->ptrShape;
    int max_size = polygon_max_size(p);
    *pixel_tab = malloc(sizeof(Pixel *) * max_size);
    *nb_pixels = 0;
    // itere sur les points du polygone
    for (int j = 0; j < p->n; j++) {
        Shape * line_shape = create_line_shape(p->points[j]->pos_x, p->points[j]->pos_y,p->points[(j + 1) % p->n]->pos_x, p->points[(j + 1) % p->n]->pos_y);
        Pixel **line_pixels = NULL;
        int line_nb_pixels = 0;
        pixel_line(line_shape, &line_pixels, &line_nb_pixels);
        // copie les pixels de la ligne dans le tableau de pixels du polygone
        for (int i = 0; i < line_nb_pixels; i++) {
            (*pixel_tab)[(*nb_pixels)++] = line_pixels[i];
        }

        free(line_pixels);
    }
}
```

codesnap.dev

Les paramètres sont : le polygone (la forme est constituée d'un ID, d'un type et d'un pointeur vers la structure polygon elle-même composée d'un tableau de points et d'une taille). Nous voulons, ici, modifier la matrice pixel_tab pour pouvoir la retourner dans la fonction draw_area qui permet d'assigner les pixels à l'area définie au préalable. Enfin nous allons modifier une variable nb_pixels qui va compter le nombre de pixels et nous permettre d'itérer à travers la matrice pixel_tab.

Dans cette fonction, nous allons itérer à travers la liste de points du polygone et créer des lignes en utilisant la fonction définie de base pour les lignes et stocker les pixels que nous renvoie cette fonction dans la matrice pixel_tab. Enfin nous libérons la mémoire de la ligne temporaire avant de recommencer avec d'autres points. L'utilisation du modulo permet de relier le dernier et le premier point sans avoir à traiter le cas particulier en dehors de la boucle for.

Gestion de l'entrée utilisateur

Dans le cadre de notre projet, qui vise à être utilisé par des personnes n'ayant aucune connaissance en programmation, il est essentiel de fournir une interface utilisateur à la fois complète, mais également simple et intuitive à utiliser.

La structure "commande" est conçue pour capturer, stocker puis exécuter les instructions fournies par l'utilisateur. En stockant toutes ces informations dans une structure organisée, nous facilitons le traitement et l'exécution des commandes ultérieurement.

Par la suite, nous utilisons une fonction d'exécution qui prend la structure "commande" comme argument et effectue les actions appropriées en fonction des informations fournies. Par exemple, elle peut créer une nouvelle forme, supprimer une forme existante, afficher la zone de dessin, ou exécuter d'autres actions en fonction des commandes spécifiées.

Conclusion : Appréciation personnelle et problèmes rencontrés

Lors de ce projet, nous avons rencontré quelques difficultés lors de l'affichage de la zone de dessin. L'origine du souci venait d'une erreur dans la gestion de mémoire qui « coupait » une partie de notre zone de dessin au moment d'assigner les pixels des formes à dessiner.

Lors de la partie de gestion de commandes, il nous a fallu trouver un moyen d'intégrer les fonctions des autres modules dans ce dernier module et notamment créer une area qui puisse être appelée dans la fonction d'exécution des commandes. Au final nous avons bien réussi à obtenir le résultat satisfaisant et faire fonctionner l'ensemble des fonctions pour l'area créée.

Approches personnelles :

Mathurin : Ce projet a été un réel plaisir et j'ai vraiment aimé travailler dessus. Cela m'a permis d'appliquer ce que nous apprenions pendant les cours et de voir une application réelle pour « sentir » le code et le comprendre.
J'ai appris à travailler sur un projet avec des délais, à gérer les échecs et à utiliser au mieux mes forces
Pour faire mieux je retravaillerais probablement les algorithmes de pixels pour optimiser et rendre plus efficace le projet. Je
Dans l'ensemble, je suis heureux et fier du travail réalisé pendant la durée du projet.

Chengfeng :

Dans ce projet en C sur les images vectorielles, j'ai compris que les images vectorielles sont basées sur des formules mathématiques et aussi que j'ai aussi découvert comment manipuler ces images en utilisant des structures de données appropriées, comme les points, les lignes etc.