DURAND Clovis
MAGNAN Camille

12 Mai 2017

PG208

PG208
Dessin vectoriel

V1

ENSEIRB-Matmeca
Bordeaux - Talence

Table des matières

Table des figures

1 Introduction

L'objectif de ce projet est de mettre en œuvre les notions de base de la programmation orientée objets appliquées au langage C++ que l'on a appréhendé durant l'enseignement de PG208. Pour ce faire nous avons décidé de nous lancer dans le second sujet de projet : **Dessin vectoriel**. L'objectif est de générer une image Bitmap en fonction d'un fichier de description vectorielle de formes. L'enjeu de ce sujet est de comprendre l'intérêt de l'utilisation de classes et d'apprendre à faire interagir les différentes classes entre elles.

1.1 Utilité du langage C++

Le projet dans lequel nous nous lançons est fait pour nous donner un aperçu des possibilités que nous offre le C++. En effet, celui-ci nous demande la création de différentes classes. À partir de ces dernières il nous est possible de leur créer un héritage (classes filles) dans lesquelles les attributs et méthodes sont réutilisables et protégés. De cette façon nous évitons un certain nombre d'erreurs (par exemple la modification des attributs) et améliorons considérablement la lisibilité de notre architecture.

Cette application est appelé `dessin_vect`

1.2 Gestion du cahier des charges

Nous avons donc tenté de réaliser une application permettant de suivre au mieux le cahier des charges qui nous a été fournis. En effet, nous avons développé un programme permettant de générer des images à l'aide d'une description vectorielle de ces images. Ces images seront créées à partir de la description de plusieurs formes géométriques. Nous avons donc programmé le dessin de 9 formes différentes afin d'avoir un large panel de possibilités graphiques : la ligne, le triangle vide, le point, le cercle (vide et plein), le rectangle (vide et plein) et enfin le carré (vide et plein).

Le cahier des charges stipule que nous devons gérer un facteur d'échelle qui modifie la résolution de notre image de sortie. Un facteur de transparence doit aussi être intégré lorsqu'on dessine une forme.

Voici le diagramme de cas d'utilisation qui représente le mode d'opération du point de vue du client :

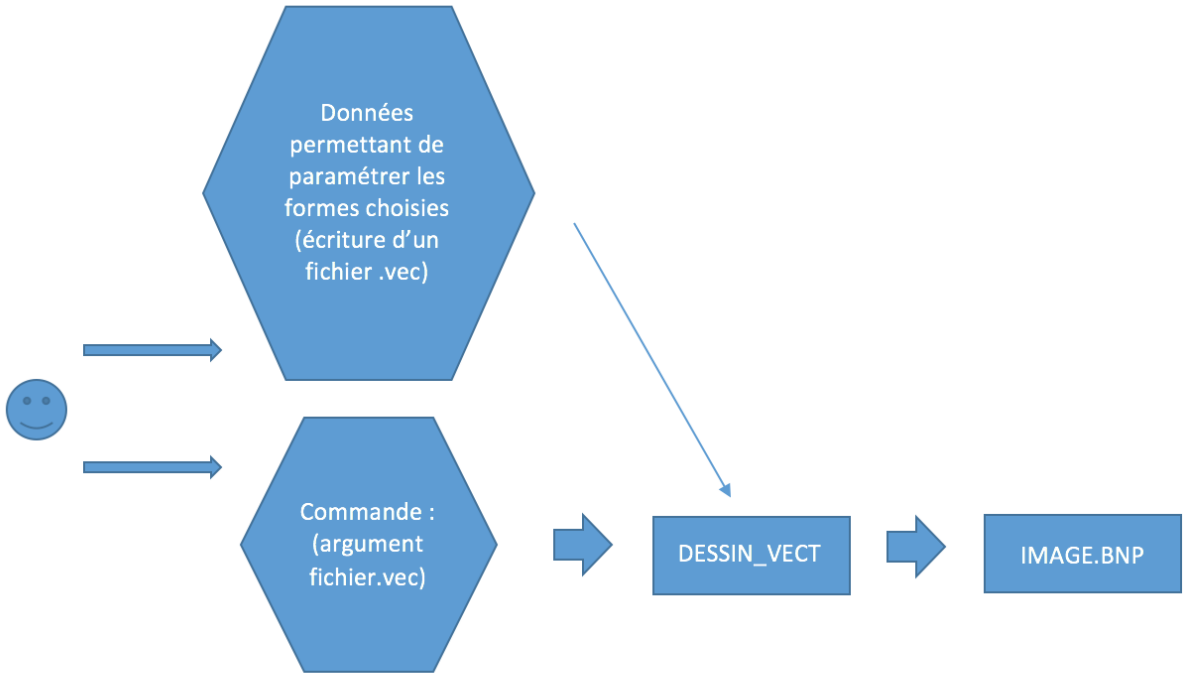

FIGURE 1 – Diagramme de cas d'utilisation de notre programme `dessin_vect`

1.3 Diagramme UML

Le langage UML est un langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode simple pour visualiser la conception d'un système. Nous avons donc utilisé ce langage afin de nous permettre de suivre un cheminement clair et détaillé de notre projet. Ce diagramme permet aussi de comprendre comment les différentes classes ont été construites les unes par rapport aux autres, comme les rapports d'héritage.

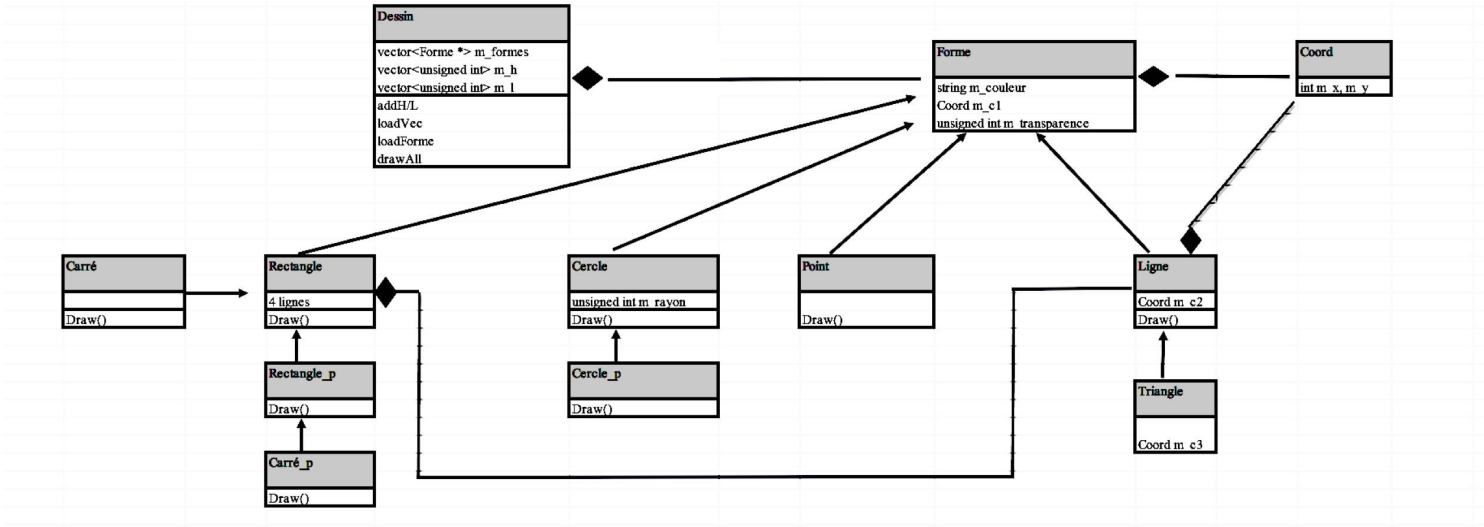

FIGURE 2 – diagramme UML

Le diagramme UML fonctionne comme suit :

— Chaque classe utilisée est explicitée. Les attributs et les méthodes principales y figurent et leurs rapports d'héritage y sont expliqués.

— Quand une classe en utilise une autre comme type d'attribut, un losange et une ligne font le lien entre les deux (le losange est placé sur la classe qui utilise l'autre comme type d'attribut).

— quand une classe hérite d'une autre, une flèche indique la classe mère et part de la classe fille.

D'autres classes - fournies par l'encadrant du projet - sont utilisées dans le projet : CBitmap gère la création d'un fichier bitmap (`.bmp`) à partie d'un objet CImage. CImage créé l'image dans laquelle on va dessiner avec la classe Dessin. On écrit dans CImage soit pixel par pixel (ce que l'on a choisi de faire). Un méthode publique à d'ailleurs été rajoutée dans CImage pour dessiner sur un pixel d'un simple appel de fonction. Les classes en question sont explicitées dans la figure suivante :

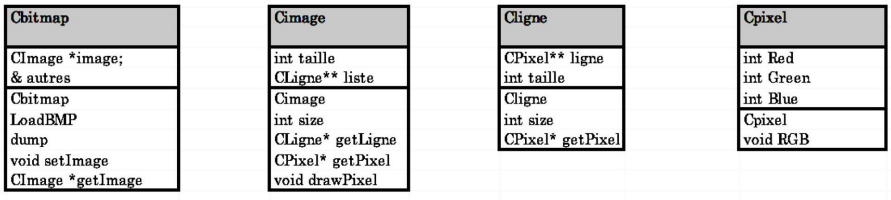

FIGURE 3 – Classes fournies par l'encadrant : CBitmap, CImage, CLigne, CPixel

2 Les classes principales

Grâce au diagramme UML vu précédemment, nous avons pu commencer à écrire le code de notre projet par les classes principales. En effet certaines des classes à programmer sont plus importantes que d'autres. Nous avons fait le choix de créer une classe de coordonnées Coord (non indispensable mais qui nous a permis de nous approprier la notion de position d'un objet dans l'image dès le début) qui rassemble les coordonnées x et y de chaque forme.

Nous avons commencé par concevoir la classe forme contient tous les attributs communs aux futures classes des différentes formes.

Ensuite, nous avons écrit le reste des classes de formes, en commençant par la classe Ligne. Celle-ci est à la base de beaucoup de formes. Par exemple, la méthode `drawLigne()` est utilisée dans toutes les formes (mis à part la classe Cercle et la classe Cercle_p).

Enfin une classe Dessin permet de réunir toutes les formes en un seul objet, de définir les bornes de l'image, et de fournir les informations au dessin de l'image.

2.1 Classe de coordonnées Coord

La classe `Coord` est une classe pratique. Celle-ci ne génère pas une forme, mais permet simplement de généraliser les 2 coordonnées d'un point en un seul objet. Cet objet servira d'attribut pour la classe Forme, ainsi que pour d'autres formes.

2.2 Classe forme Forme

La classe mère Forme regroupe l'ensemble des attributs communs à toutes les classes de formes :

— Coord m_c1 : Cette coordonnée définit la position dans l'image de l'objet. Ce point sert de base pour construire le reste de la forme.

— string m_couleur : La couleur de l'objet. Tous les objets à dessiner doivent avoir une couleur définie, de manière à écrire cette couleur sur un pixel de l'image. Pour ce faire un format RGB est utilisé (0 à 255 pour chacune des trois couleur primordiales).

— unsigned int m_transparence : Ce facteur permet de définir le niveau de transparence de la forme sur le dessin.

Elle regroupe également les méthodes utilisées dans les différentes formes, c'est à dire la méthode générant la couleur des pixels en fonction des valeurs de R, G et B. De cette façon il nous est possible de récupérer la valeur de R, G et B indépendamment afin de déterminer la couleur de notre pixel.

REMARQUE : Nous avons rajouté une fonction `drawPixel` dans la classe CImage qui nous permet de dessiner chaque pixel grâce à un simple appel à cette fonction.

Une des méthodes de la classe Forme nous permet de déterminer le réglage de la couleur du pixel en fonction du niveau de transparence retenu grâce à la formule suivante qui prends en arguments la position du pixel, la couleur à affecter, le niveau de transparence et la couleur du pixel avant application de la nouvelle couleur.

$$Pixel'(x,y) = \frac{(100 - transparence).Pixel(x,y) + transparence.CouleurForme}{100}$$

2.3 Classe Dessin

La classe Dessin est la classe qui englobe toutes les formes dans le but de les dessiner. En revanche, elle à également d'autres fonctions.

Elle est d'abord composée d'un vecteur de pointeur vers des objets de type Forme. Grâce à ce vecteur, on peut rassembler toutes les formes à dessiner dans un seul objet en vue de les traiter. En effet, toutes les formes héritent justement de la classe Forme, et donc un pointeur vers un objet de type Forme peut pointer vers un objet de type Cercle par exemple. Une fois que toutes les formes sont pointées par des éléments du vecteur `m_formes`, un simple appel à la méthode `drawAll()` de la classe Dessin permet de dessiner l'ensemble de l'image dans l'objet de type CImage (classe fournie par le professeur encadrant).

La classe Dessin permet également de capitaliser les informations relatives aux positions extremum des différentes formes dans des attributs réservés pour cela. `m_l` et `m_h` sont des vecteurs d'entiers positifs recensant ces extremums. Après avoir trié ces vecteurs avec un tri à bulles, on peux en extraire le maximum et ainsi obtenir la hauteur et la largeur de l'image de destination.

Enfin, deux méthodes permettent de charger les informations vectorielles des formes depuis un fichier `.vec`. D'abord, `loadVec` est une méthode publique de Dessin qui prends en argument le nom du fichier `.vec`, en extrait une à une les lignes puis les passe en argument à la méthode privée `loadForme`. Cette dernière extrait toutes les informations d'une ligne et ajoute la forme correspondante au vecteur de pointeurs de type Forme de la classe Dessin.

REMARQUE : Les fichiers `.vec` peuvent contenir des commentaires. Les commentaires doivent avoir en premier caractère le "#". En revanche, la gestion des lignes vides dans un fichier `.vec` n'est pas assuré dans cette version de `dessin_vect`.

3 Classes filles de Forme

3.1 Classe Ligne

La classe Ligne hérite de la classe Forme. En plus des attributs provenant de Forme, un autre objet de type Coord permet de définir la deuxième extrémité de la ligne.

Quant à la ligne dans l'image, nous avons décidé de tracer nos lignes à l'aide de l'algorithme généralisé optimisé de Bresenham. Cette méthode permet d'identifier le sens du tracé de la ligne et de déterminer les pixels que l'on doit dessiner afin de s'approcher le plus près possible d'une ligne droite (avec un taux d'erreurs le plus faible possible). Pour cela, nous séparons l'écran en octants. Nous partons d'un point central, puis nous prenons le deuxième point de la ligne et nous calculons les deltas de la droite reliant ces deux points. Grâce à ceux-ci nous déterminons dans quel octant nous nous trouvons et appliquons ainsi l'algorithme de Bresenham dans cet octant.

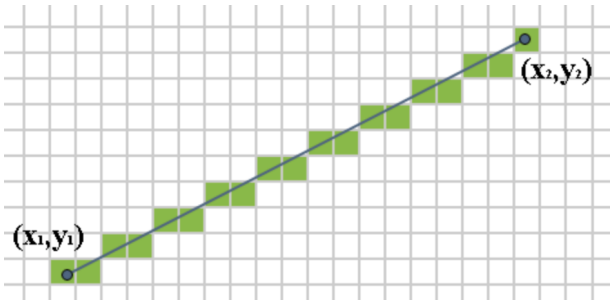

FIGURE 4 – Schéma explicatif de la méthode de Bresenham

3.2 Le Point

Le point hérite de la classe Forme. La seule chose qui diffère de Forme est la fonction `draw` qui permet de dessiner le point. Dans ce projet, le choix qui à été fait quand au dessin du point est de colorier un simple pixel à la coordonnée `m_c1`.

REMARQUE : De part notre choix de la méthode de dessin du point, quand un facteur d'échelle est défini la position du point sera impacté mais pas sa taille.

3.3 Le cercle

3.3.1 Le cercle vide : Cercle

Le cercle hérite de la classe Forme. Un attribut supplémentaire est défini dans Cercle : le rayon `m_rayon`. Nous utilisons ensuite une condition de longueur, telle que si l'on se trouve à une distance r du centre,

alors le pixel est dessiné. Afin d'avoir un tracé suffisamment large à l'œil humain, nous avons ajouté un "epsilon" de sorte que le pixel soit dessiné si il se trouve dans l'intervalle défini : $r \pm \varepsilon$

3.3.2 Le cercle plein : Cercle_p

Le cercle plein Cercle_p hérite de Cercle. Pour le dessiner, on dessine des cercles concentriques de rayon se décrémentant de `m_rayon` jusqu'à 1. Ensuite on colorie un simple pixel au centre du cercle.

3.4 Le rectangle

3.4.1 Le rectangle vide : Rectangle

Le rectangle vide hérite de Forme et est tout simplement créé à partir de 4 lignes. Ces lignes trouvent leurs coordonnées en fonction du point de départ (coin en bas à gauche du rectangle) et des longueurs et hauteurs. Pour le dessiner, il suffit de dessiner les 4 lignes qui sont en attribut.

3.4.2 Rectangle plein : Rectangle_p

Le rectangle plein hérite du rectangle simple et fonctionne à partir de lignes également. En effet, le remplissage se fait en balayant le rectangle dans la hauteur et en dessinant une ligne à chaque incrémentation. En revanche, on ne déclare pas d'objet Ligne à chaque incrémentation.

3.5 Le carré

3.5.1 Le carré vide : Carre

La classe Carre hérite de la classe Rectangle. Pour le construire, on appelle le constructeur de Rectangle avec une hauteur égale à sa longueur. Pour le dessiner, la méthode est la même que pour la classe Rectangle.

3.5.2 Le carré plein : Carre_p

La classe Carre_p hérite de la classe Rectangle_p. Pour le construire, on appelle le constructeur de Rectangle_p avec une hauteur égale à sa longueur. Pour le dessiner, la méthode est la même que pour la classe Rectangle_p.

3.6 Le Triangle `Triangle`

La classe Triangle hérite de la classe Ligne. Le seul attribut supplémentaire est un troisième objet Coord. Pour le dessiner, on créer les lignes manquantes et on dessine toutes les lignes.

4 Dimensions graphiques et facteur d'échelle

Pour régler les dimensions de l'image de destinations, nous prenons les vecteurs d'extremums de Dessin (attributs (m_l) et `m_h`) et nous en extrayons le maximum. Ensuite, on adapte les bornes de notre objet CImage à partir de ces maximums est les passant en argument du constructeur de CImage. Ces extremums sont obtenus lors du chargement des objets dans la méthode `Dessin::loadForme`.

D'ailleurs, avant de récupérer les extremums, on les multiplie par le facteur d'échelle. De cette façon il est appliqué avant la définition des bornes de l'image et le dessin des formes. Le facteur d'échelle est un des arguments de l'exécutable.

5 Utilisation de notre programme `dessin_vect`

5.1 Écriture du fichier .vec

Pour utiliser notre programme, il faut d'abord écrire un fichier de description des formes. Chaque forme à un format particulier. Pour chaque forme, le format est le suivant :

- Le point : `PIXEL:600,600,BLANC,100`. Le mot clé POINT peut également être utilisé. Les arguments sont : x, y, couleur, transparence.
- La ligne : `LIGNE:300,300,500,400,BLANC,100`. Le mot clé LINE peut également utilisé. Les arguments sont : x0, y0, x1, y1, couleur, transparence.
- Le rectangle : `RECTANGLE:300,300,100,200,ROSE,100`. Les arguments sont : x, y, hauteur, largeur, couleur, transparence.
- Le rectangle plein : `RECTANGLEP:300,300,30,50,MARRON,100`. Le mot clé RECTANGLES peut également être utilisé. Les arguments sont : x, y, hauteur, largeur, couleur, transparence.
- Le carré : `CARRE:900,10,75,PINK,70`. Les arguments sont : x, y, hauteur, cote, transparence.
- Le carré plein : `CARRES:900,10,75,PINK,70`. Le mot clé CARREP peut également être utilisé. Les arguments sont : x, y, hauteur, cote, transparence.
- Le cercle : `CERCLE:500,500,75,ROSE,30`. Les arguments sont les suivants : x, y, rayon, couleur, transparence
- Le cercle plein : `CERCLEP:500,500,75,ROSE,30`. Les mots clé CERCLES et DISQUES peut également être utilisé. Les arguments sont : x, y, rayon, couleur, transparence
- Le triangle : `TRIANGLE:100,100,200,200,100,300,BLUE,100`. Les arguments sont : x1, y1, x2, y2, x3, y3, couleur, transparence.

5.2 Exécution du programme

Une fois le programme compilé, on l'exécute avec la commande et les arguments suivants :

```
$ <chemin_vers_executable>/dessin_vect <chemin_vers_.vec/nom.vec> <facteur d'échelle>
```

5.3 Résultats obtenus lors d'essais

Voici deux images que nous avons obtenus lors des essais de notre programme final.

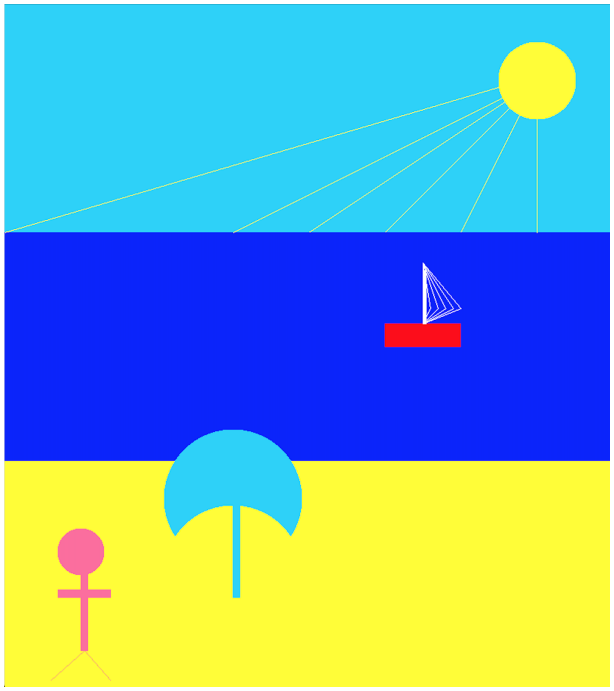

FIGURE 5 – `plage.jpg`

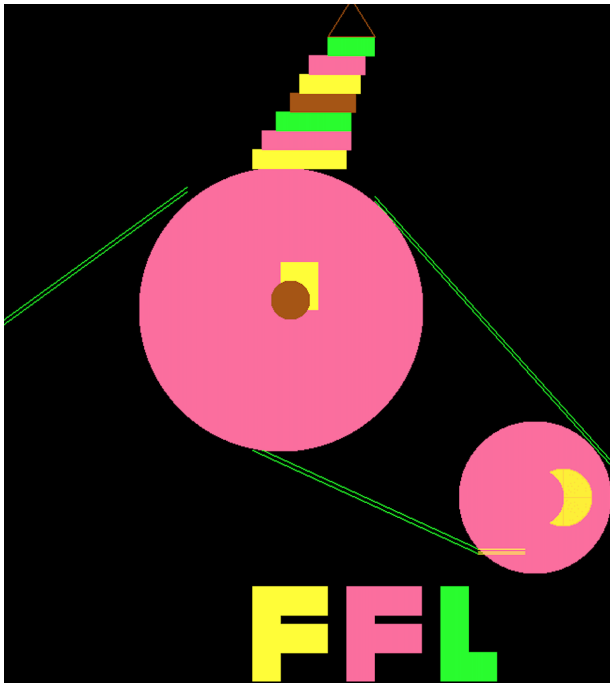

FIGURE 6 – licorne.jpg

6 Conclusion

Ce projet nous a permis d'avoir un aperçu sur les possibilités que nous offre le langage C++. En effet celui celui-ci est un réel avantage quant à la programmation objet. Nous parvenons ainsi à dessiner dans un fichier bitmap des images à partir d'une description vectorielle des formes qui la compose. Néanmoins il est possible d'approfondir ce projet tant dans les formes disponibles (nous nous sommes intéressés au rectangle de travers par exemple) que dans une interface plus ergonomique, comme une interface graphique par exemple.