Université Paris-Est Marne-la-Vallée
Master 2 Logiciel
2017/2018

Projet C++
Figures géométriques

MOREAU Ameline
VIEIRA NORO Kevin
ZECCHINI Aymeric

Table des matières

Introduction

L'objectif du projet est d'implémenter des figures géométriques (Ligne, Cercle, Rectangle et Triangle) et ensuite de pouvoir appliquer des transformations sur celle-ci. Ces figures sont contenues dans une image, qui elle-même peut subir des transformations et inclure des images de plus petite aire.

Le point le plus important du projet est d'utiliser une bonne architecture, et d'utiliser un maximum de fonctionnalité de c++11 et c++14.

Les figures devront être lues à partir d'un fichier et devront si possible, être visualisable à l'aide d'une interface graphique.

Manuel d'Utilisation

Lancement du Main

Lorsque l'on exécute le Main, les choix suivants sont définis :

- Ligne : test la création et transformation d'une ligne
- Cercle : test la création et transformation d'un cercle
- Rectangle : test la création et transformation d'un rectangle
- Triangle : test la création et transformation d'un triangle
- Lire Fichier : test la création et transformation d'un fichier Images/image1.txt
- **Démonstration : scénario qui présente toutes les fonctionnalités demandées**.

Import de fichier

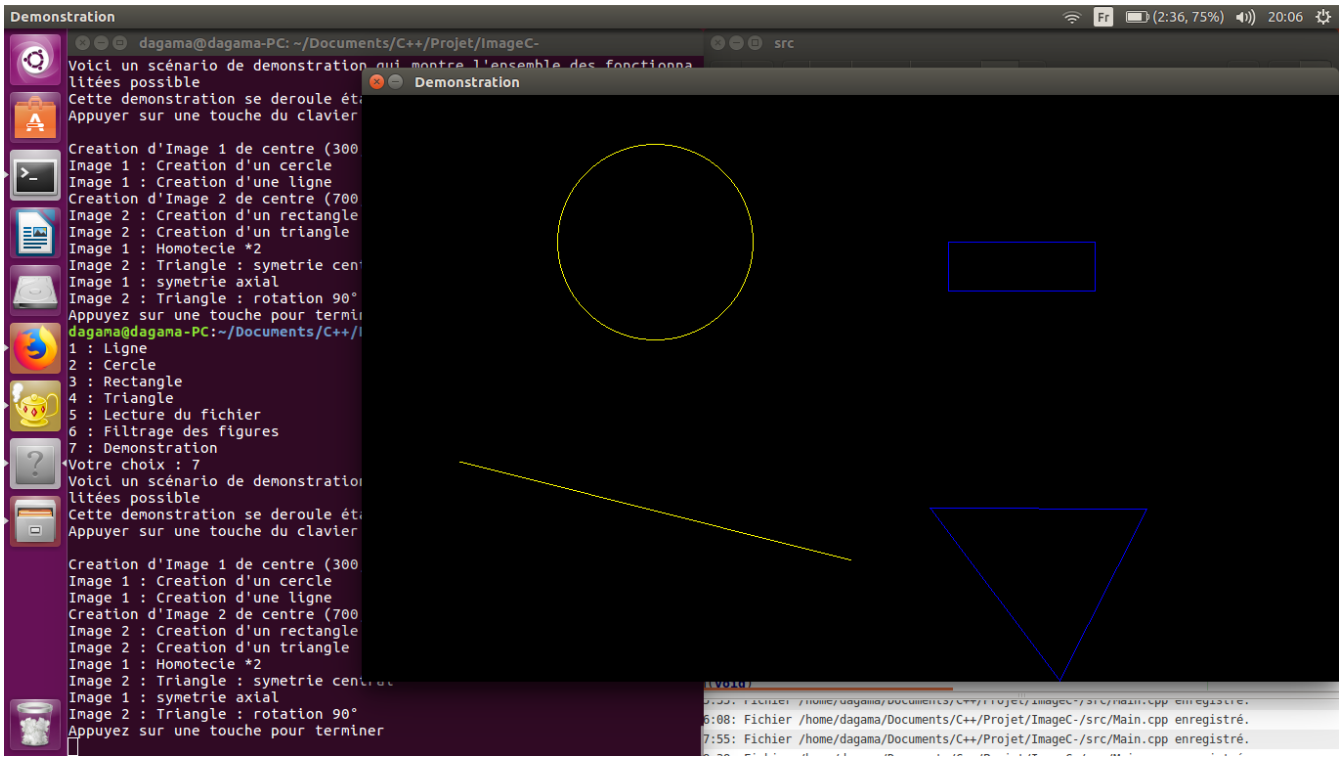
Il est possible d'importer une figure à partir d'un fichier. L'import d'un fichier se fait à partir du fichier « Images/image1.txt ». Le format à respecter est le suivant :

- Image pour X Y pour dire que c'est une image de centre X Y
- Circle X1 Y1 X2 Y2 pour un cercle de centre X1, Y1 et de rayon X2, Y2 (distance avec le centre)
- Triangle X1 Y1 X2 Y2 X3 Y3
- Rectangle X1 Y1 X2 Y2 Point haut gauche bas droit
- Line X1 Y1 X2 Y2

Il suffit de mettre une Image pour définir ses enfants puis écrire un END pour dire que l'on a terminé d'inséré les enfants de l'image parente. Il doit y avoir autant d'END que d'image.

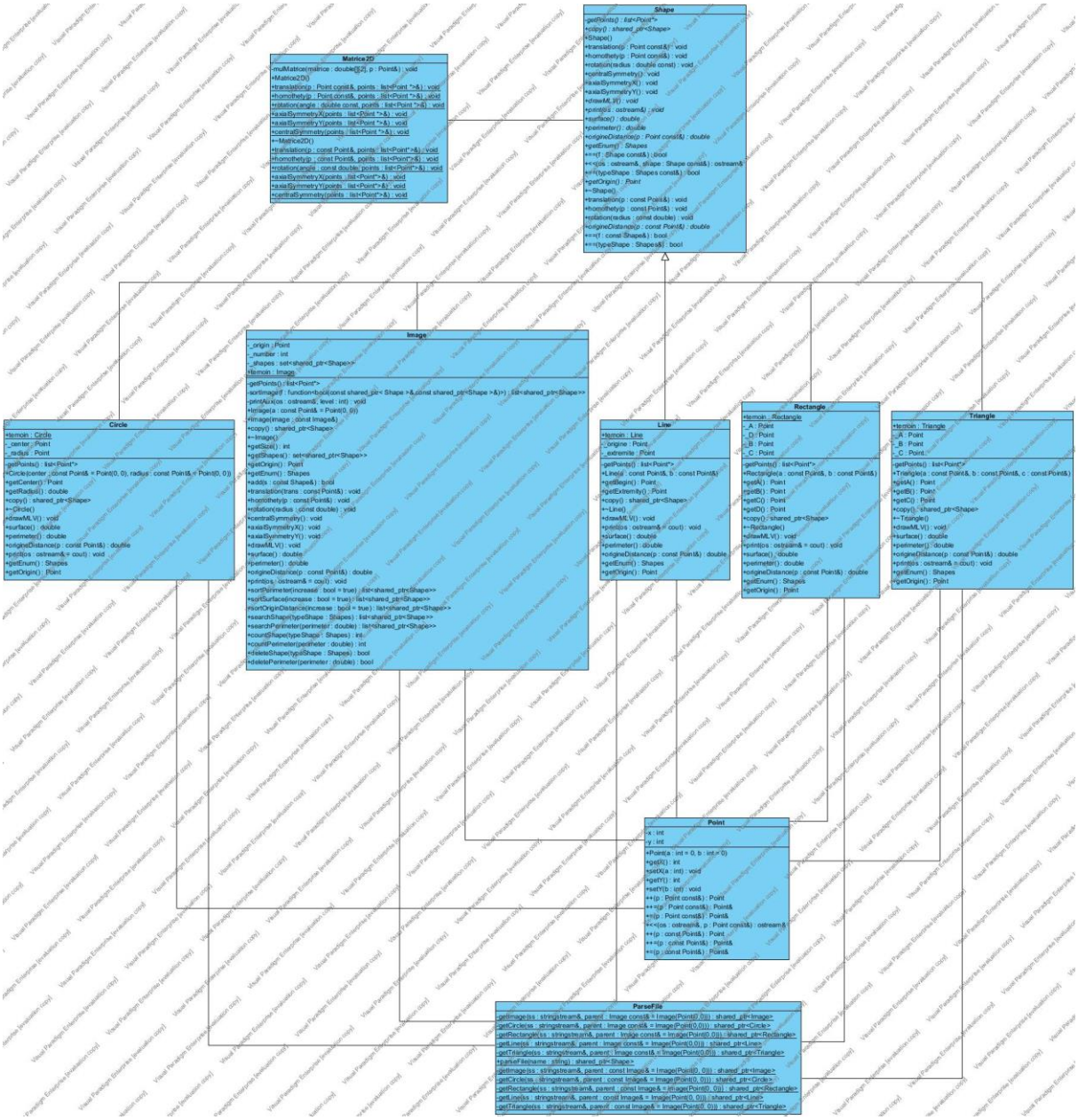
Documentation

Nous avons généré la documentation de notre programme à l'aide d'oxygen. Elle est consultable dans le répertoire docs/oxygen/html/index.html.



Architecture Fonctionnelle

Diagramme UML



Nous pouvons visualiser que nous avons un ensemble de figures, Rectangle, Cercle, Triangle, Ligne et les Images qui héritent de Shape. Sur une Shape nous pouvons faire un ensemble de transformations géométriques définies dans le sujet. Il est également possible d'importer une Shape à l'aide d'un fichier et d'un parseur que nous avons créé.

Container Image

Une image est une figure qui doit contenir d'autres figures. Pour ce faire nous avons utilisé le design pattern composite et nous avons utilisé un set de la stl. Cela permet d'avoir une liste sans doublon. Quand nous ajoutons une figure dans une image, nous nous assurons que la surface est bien plus petite. Afin que qu'une image incluse dans une image ne deviennent pas plus grande, nous nous assurons de faire une copie car sinon elle pourrait être potentiellement insérée en dehors de l'image qui la contient, puis modifiée et donc avoir une surface plus grande.

Nous souhaitons également pouvoir rechercher, compter, supprimer des figures. Comme cela n'a de sens que pour les Images car ce sont les containers, nous avons définies des méthodes dans cette classe qui permettent de faire cela. Afin de ne pas avoir à dupliquer du code, nous avons fait des méthodes privé qui prennent des fonctions de tri, de comparaisons… et nous avons décliné les appelles à ses méthodes privés avec des lambda pour répondre à toutes les attentes.

Description des classes

Image, **Circle**, **Rectangle**, **Triangle**, **Line** et **Point** permettent de représenter des figures. Elles héritent toutes de **Shape** qui permet de factoriser le code et d'éviter les redondances de code car les figures ont toutes le même comportement. **Shape** permet d'effectuer des transformations à l'aide de la méthode « getPoints » redéfinie par chaque figure qui renvoie leurs points et de la classe **Matrice2D** qui permet d'effectuer des transformations sur des listes de point. Shape impose de redéfinir également différentes méthodes notamment de dessin. Nous avons une classe **ParseFile** qui permet de parser un fichier pour en créer une figure.

Matrice de transformation

Pour pouvoir effectuer des transformations, nous avons appliqué les matrices. Pour chaque point, nous multiplions la matrice de transformation avec chaque point de la figure. Il est nécessaire de tenir compte de l'origine qui n'est pas forcément (0,0). Pour remédier à ce problème, avant les appels nous soustrayons la distance des points à l'origine afin de les ramener à (0,0) puis nous les remettons à leurs place une fois la transformation effectué.

Pointeurs partagés

Afin de remédier au problème d'allocation, nous avons utilisé un container pour les images de type « set<shared_ptr<Shape>> ».

Structures de données

Rectangle

Un rectangle est construit à partir de deux points, le point en haut à gauche et le point en bas à droite. Avec ces deux points nous connaissons les coordonnées des deux autres points.

Cercle

Un cercle est représenté par deux points. Le centre et un point qui est sur l'extrémité du cercle. Le rayon est calculé à partir de la distance entre le cercle et l'extrémité.

Ligne

Une ligne est représentée par deux points.

Triangle

Un triangle est représenté par 3 points

Image

Une image est représentée par un point qui est son origine. Elle contient également un set de figure.

Shape

Shape est une classe abstraite qui propose des méthodes de transformations à partir d'une liste de point ainsi que des méthodes à redéfinir de dessin.

Shapes

Shapes est une énumération qui permet à chaque figure d'implémenter une méthode « getShape » qui renvoie le type de la figure. Ceci est très pratique pour utiliser le parser. En effet, nous transformons une ligne en figure et à l'aide de cette méthode nous savons récupérer son type. Nous aurions pu utiliser aussi des dynamic_cast mais ce n'était pas élégant et trop lourd à écrire pour notre parser.

Fonctions

Import d'une figure

Pour transformer du texte en une figure, nous avons défini un protocole simple. Nous avons utilisé une file d'attente, et lorsque nous rencontrons une image nous l'ajoutons à cette liste. Elle devient le parent et nous insérons les figures dans cette image jusqu'à tomber sur un END. Dans ce cas nous passons à l'image suivante si il y en a une et nous insérons dedans les nouvelles figures jusqu'à rencontrer à nouveau un END.

Exemple :

```
Image 50 50
END
Image 100 100
Image 100 100
END
Circle 100 100 50 50
Triangle 500 500 600 600 500 600
END
Rectangle 200 200 400 400
Line 200 200 600 600
END
```

Transformation

Nous pouvons appliquer différentes transformations :

void Matrice2D::translation(const Point & p, list<Point *> & points) : qui va permettre de translater une figure de p.x et p.y.

void Matrice2D::homothety(const Point & p, list<Point *> & points) : qui va faire une homothétie.

void Matrice2D::rotation(const double angle, list<Point *> & points) : rotation de la figure

void Matrice2D::axialSymmetryX(list<Point *> & points) : symétrie axiale sur X

void Matrice2D::axialSymmetryY(list<Point *> & points) : symétrie axiale sur Y

void Matrice2D::centralSymmetry(list<Point *> & points) : symétrie centrale

void mulMatrice(double matrice [][2], Point & p) : multiplie un point à l'aide d'une matrice.

Pour chaque transformation hors mis la translation, nous appliquons pour chaque point une multiplication des points de la figure par une matrice propre à la transformation en question. Afin de gérer l'origine, il faut s'assurer qu'avant l'appel, les points soient soustraits pour être placé par rapport à (0,0) puis refaire l'addition pour leur remettre à leurs origines initiales.

Trier, Rechercher, Supprimer

Nous pouvons trier, rechercher et supprimer par rapport à une liste de figure.

list<shared_ptr<Shape>> Image::sortImage(function<bool(const shared_ptr<Shape> &, const shared_ptr<Shape> &)> f) : C'est une méthode privé qui va permettre de trier une image selon la function de comparaison passé en paramètre.

list<shared_ptr<Shape>> Image::sortPerimeter (bool increase): Cette method va permettre de trier selon l'ordre croissant ou décroissant les périmètres.

list<shared_ptr<Shape>> Image::sortSurface (bool increase): Trie une liste par rapport à la surface de la figure selon l'oridre croissant ou décroissant.

list<shared_ptr<Shape>> Image::sortOriginDistance (bool increase): Trie selon la distance à l'origine

list<shared_ptr<Shape>> Image::searchShape (Shapes typeShape): Renvoie toutes les figures d'une image d'un certain type.

list<shared_ptr<Shape>> Image::searchPerimeter (double perimeter): Renvoie toute les figures supérieur à un périmètre.

int Image::countShape (Shapes typeShape): Renvoie le nombre de figure d'un certain type. On renvoie la taille de la liste trouvé avec searchShape

int Image::countPerimeter (double perimeter) : Compte le nombre de figures supérieur à un périmètre.

bool Image::deleteShape (Shapes typeShape): Supprime les figures d'un certain type et renvoie vrai si il y a eu des suppressions.

bool Image::deletePerimeter (double perimeter): supprime les figures avec un périmètre supérieur à une valeur.

Conclusion

C'est un projet à l'apparence simple, mais qui dans le fond nous a permis d'exploiter et de découvrir les fonctionnalités de c++ que nous ne maitrisions pas parfaitement. Nous avons pu découvrir la puissance de c++ mais qui peut très vite prendre le dessus si nous ne comprenons pas le fonctionnement des différents principes. L'une des plus belles nouveautés que nous ne connaissions pas sont les pointeurs intelligents, en effet c'est un vrai atout par rapport au c qui facilite grandement la vie. Nous avons aussi du appliqué le principe SOLID, qui est un principe fondamentale en programmation car c'est aussi ce qui va assurer l'évolutivité et la maintenabilité simple d'un programme.

Bibliographie

https://upload.wikimedia.org/wikipedia/commons/6/65/W3sDesign_Composite_Design_Pattern_UML.jpg

http://www.drdobbs.com/cpp/the-c14-standard-what-you-need-to-know/240169034

https://stackoverflow.com/questions/38060436/what-are-the-new-features-in-c17

https://discourse-production.oss-cn-shanghai.aliyuncs.com/original/3X/c/e/ce2e1fe51865660aaa835bd2674c1920e49f4c82.pdf

http://www.cplusplus.com/info/history/

https://www.youtube.com/watch?v=1OEu9C51K2A

https://www.youtube.com/watch?v=fX2W3nNjJIo