# RAPPORT

## Projet Morphing

ING1 GI

**Barrouillet Margaux**
**Garcia Lukas**
**Plaut-Aubry Baptiste**
**Destouches Nolan**
**Philipponneau Thomas**

CY TECH

# SOMMAIRE



Lien du GitHub : https://github.com/BaptistePlautA/ProjetGL3

Lien Drive du rapport (pour voir les gif) : RAPPORT

# Morphing de formes unies simples

## *Explication de notre 1ère démarche :*

*1er diagramme de classe : voir dossier Annexes (diagramme_classe_part_1bis_simple.mdj)*

Initialement, nous avions opté pour une approche qui faisait appel à des canevas, les formes de base ainsi que les formes intermédiaires étaient dessinées. Dans cette méthode, les formes étaient considérées comme des tableaux de points, reliés entre eux lors de l'affichage.

L'utilisateur avait le choix parmi une liste de formes que nous lui proposions pour les formes de début et de fin. Pour générer le GIF, chaque forme intermédiaire était dessinée sur un canevas, puis enregistrée en jpg, et un GIF était créé avec toutes les images.

Cependant, cette approche présentait un inconvénient majeur en limitant l'utilisateur aux formes que nous lui proposions, réduisant ainsi considérablement ses options. En se penchant sur les méthodes suivantes, nous avons réalisé que cette approche serait trop lourde et peu réutilisable. Ainsi, nous avons revu notre diagramme de classe et repensé la manière dont nous manipulons les informations. Ce changement nous a permis d'adopter une approche plus flexible et efficiente, offrant ainsi à l'utilisateur un plus large éventail de possibilités et facilitant la réutilisation du code pour les futures méthodes.

## *Démarche finale :*

*2e diagramme de classe : voir dossier Annexes (diagramme_classe_part_1_simple.mdj)*

Désormais, l'utilisateur a la possibilité d'importer le fichier de son choix, contenant une image représentant la forme de début ou de fin qu'il souhaite utiliser.
Dans cette nouvelle approche, nous considérons chaque image comme un tableau 2D de pixels. Ce changement permet à l'utilisateur de définir ses propres formes, offrant ainsi une expérience plus flexible et adaptable à ses besoins spécifiques. Ensuite, l'utilisateur peut sélectionner ses points de contrôle sur les formes de début et de fin, ainsi que le nombre d'images par secondes et le délai (en millisecondes) entre les transitions pour le gif final.

En termes de traitement, nous calculons le déplacement des points de contrôle de la manière suivante : nous utilisons les coordonnées du premier point (celui de départ) dans la map qui stocke les points de contrôle de départ et les coordonnées du premier point (celui d'arrivée) dans la map qui stocke les points de contrôle d'arrivée.

Nous utilisons ensuite la formule mathématique suivante :

$$dx = \frac{x2 - x1}{nombre\ d'étapes}\ et\ dy = \frac{y2 - y1}{nombre\ d'étapes}$$

Nous obtenons ainsi le déplacement à effectuer sur le point de départ, il suffit alors d'ajouter ces coordonnées au point. Puis, nous reformons les images intermédiaires en conséquence : nous commençons par recolorer l'image en blanc, nous permettant d'effacer tous les tracés précédents, avant de basculer les pixels correspondants aux nouveaux points (calculés précédemment) en noir.

Nous utilisons ensuite l'algorithme de Bresenham pour tracer les lignes entre chaque point consécutif de la map. Il permet de déterminer les pixels à colorier en incrémentant les coordonnées x et y et en calculant quel pixel est sur la bonne droite pour le colorier. Nous avons choisi cet algorithme car il est efficace, rapide et ne nécessite que peu de ressources pour effectuer les calculs. C'est ainsi que notre forme a les contours tracés en noir, nous permettant de la colorer en parcourant tous les pixels du tableau et en jouant sur les conditions de coloration des pixels voisins.

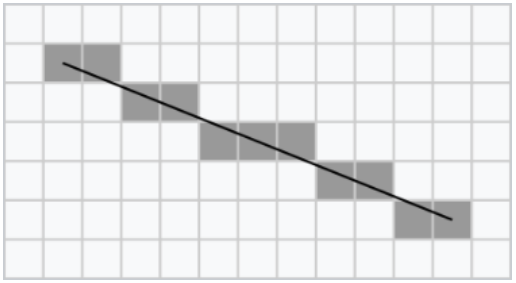


*illustration du résultat de l'algorithme de Bresenham*

Une fois cette étape terminée, nous enregistrons les images et les utilisons pour créer un GIF animé, offrant ainsi à l'utilisateur une visualisation fluide et dynamique de la transition entre les formes de début et de fin. De plus, l'utilisateur peut, s'il le souhaite, visualiser les différentes images une à une pour observer chaque étape de la transition de l'image de début à celle de fin dans le dossier Formes après la génération du GIF.

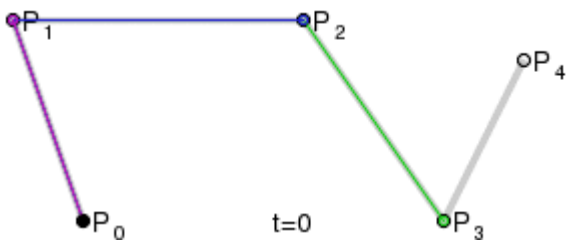
# Morphing de formes arrondies

Pour le morphing des formes arrondies, nous avons réutilisé une partie du code du morphing des formes simples. La structure générale du programme avec le calcul des images intermédiaires, la génération des images en format JPG, et la conversion de ces images en un fichier GIF animé, reste les mêmes. La principale différence se trouve dans la manière dont nous traçons les formes. Au lieu d'utiliser des segments de lignes droites pour créer les formes, nous avons utilisé les courbes de Bézier pour obtenir des formes arrondies. Elles sont définies par une série de points de contrôle et une équation mathématique qui détermine la trajectoire de la courbe.

Nous utilisons l'équation cubique qui fonctionne avec quatre points de contrôle et qui permet donc d'avoir une flexibilité et un contrôle des courbes plus important. Pour une courbe de Bézier cubique l'équation est :

$$B(t) = (1-t)^3 P_0 + 3 (1-t)^2 t P_1 + 3 (1-t) t^2 P_2 + t^3 P_3$$

Où $P_0$, $P_1$, $P_2$, et $P_3$ sont les points de contrôle, et t est un paramètre qui varie de 0 à 1. Cette équation nous permet de calculer les coordonnées des points de la courbe de manière très précise. En échantillonnant la courbe à des intervalles réguliers, nous obtenons une série de points intermédiaires que nous utilisons pour dessiner les courbes.




*exemple de construction de courbe de Bézier (GIF)*

En intégrant ces courbes de Bézier dans notre logique de morphing, nous avons pu générer des formes arrondies qui évoluent progressivement d'une forme initiale à une forme finale. Chaque étape de l'animation est calculée en utilisant les mêmes techniques que pour les formes simples, avec l'ajout de la logique pour les courbes de Bézier. Ainsi, les images intermédiaires sont générées et stockées en format JPG, puis assemblées en un GIF animé de la même manière qu'auparavant.

Du côté utilisateur, celui-ci place ses points de contrôle de la même manière que lors du morphing simple. La différence réside dans l'ajout des points intermédiaires. Ces points intermédiaires, situés entre deux points de contrôle classiques, permettent à l'utilisateur de déterminer la courbure de la courbe. Nous avons dessiné la courbe formée par les points de contrôle et les points intermédiaires sur l'image (interface) afin que l'utilisateur puisse positionner ses points avec précision.

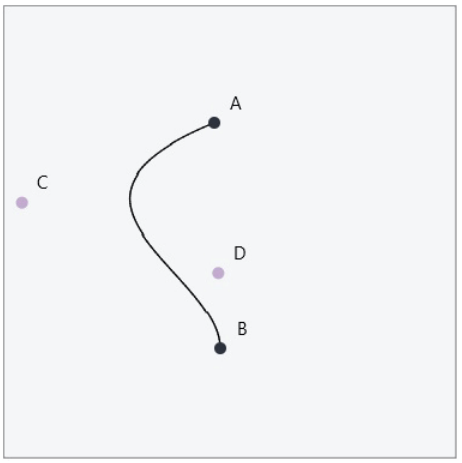
Nous avons également revu notre diagramme de classes pour améliorer l'efficacité du code et éviter les redondances. En observant que plusieurs méthodes se répétaient entre le contrôleur pour les formes simples et celui pour les formes arrondies, nous avons ajouté une méthode abstraite. Cette approche nous a permis d'optimiser la structure du code en regroupant les fonctionnalités communes et en réduisant la duplication du code.

*2e diagramme de classe : voir dossier Annexes (diagramme_classe_part_2_arrondi.mdj)*

De la même manière que pour le morphing simple, l'utilisateur peut retrouver dans le dossier Formes, les images qui correspondent à chaque étape de transition de l'image de début à celle de fin.

## *Limitations techniques :*

Pour le déplacement des points intermédiaires, il semble y avoir un décalage que nous n'avons pas réussi à régler lorsque ces points sont placés trop proches du bord du carré dans lequel l'image est contenue. Autant pour celle de départ que celle d'arrivée. Cela a pour cause de déformer la courbe sur l'interface.

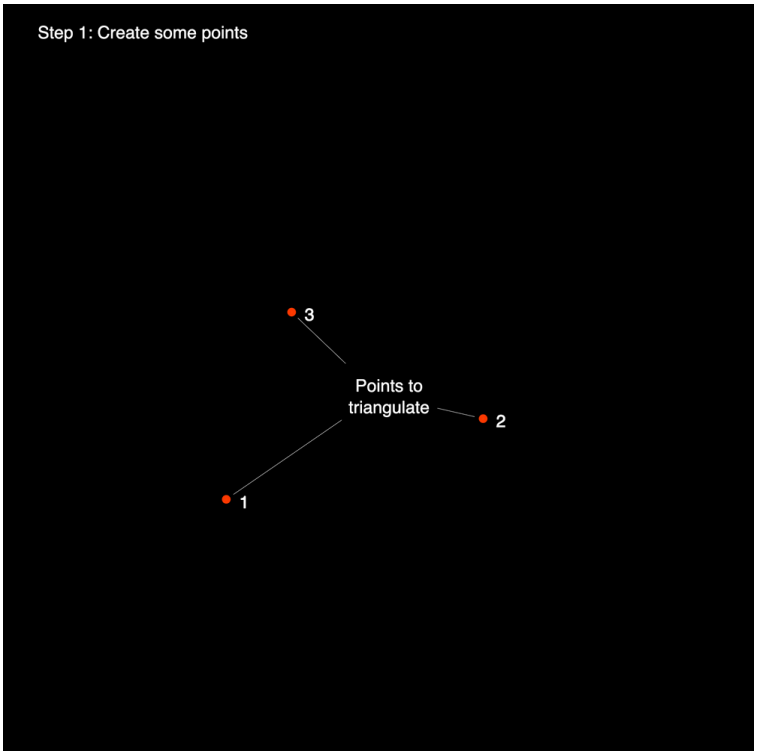


# Morphing d’images

*Explication de la méthode utilisée :*

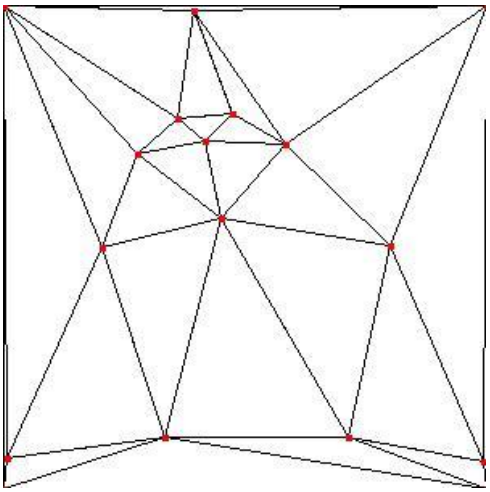
Notre idée de base pour le morphing d’images complexes était d’utiliser la première méthode suggérée par le sujet : le maillage de Delaunay. Le but du maillage est de diviser notre image globale en un ensemble de triangles issus des différents points de contrôle placés par l’utilisateur afin de faire une succession d'interpolations selon des zones fixes (respectivement un triangle de départ associé à un triangle d'arrivée).

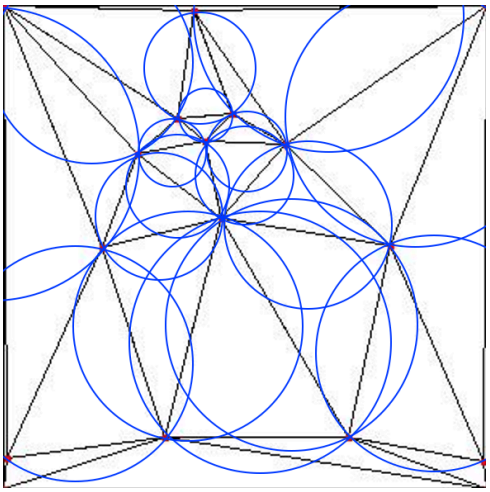
Pour calculer cette triangulation, nous utilisons l’algorithme de Bowyer-Watson.




Source du GIF : 8. **Bowyer-Watson Algorithm for Delaunay Triangulation**

En suivant et en modélisant cette technique, nous obtenons des triangles dont chaque cercle circonscrit est sur trois sommets du supertriangle associé sans qu’aucun point ne soit contenu à l’intérieur. On sélectionne donc et supprime successivement les “mauvais triangles” (ceux qui ne respectent pas cette condition).

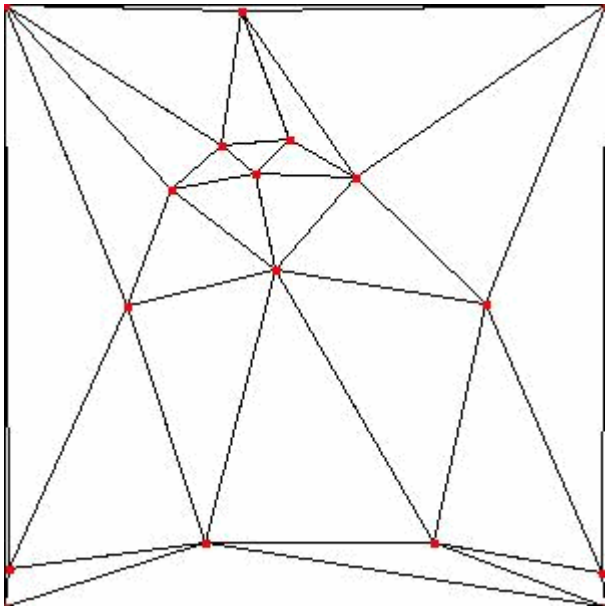




Application de cet algorithme : on obtient un maillage qui respecte ces conditions en utilisant l'algorithme de Bowyer-Watson. Cette triangulation n'est effectuée que pour la première étape du morphing et n'est pas respectée pour les étapes intermédiaires et l'étape finale. Finalement on peut simplifier la transformation par un maillage de départ qui respecte les conditions précédentes et qui déplace les points de départ et d'arrivée selon le nombre d'étapes.

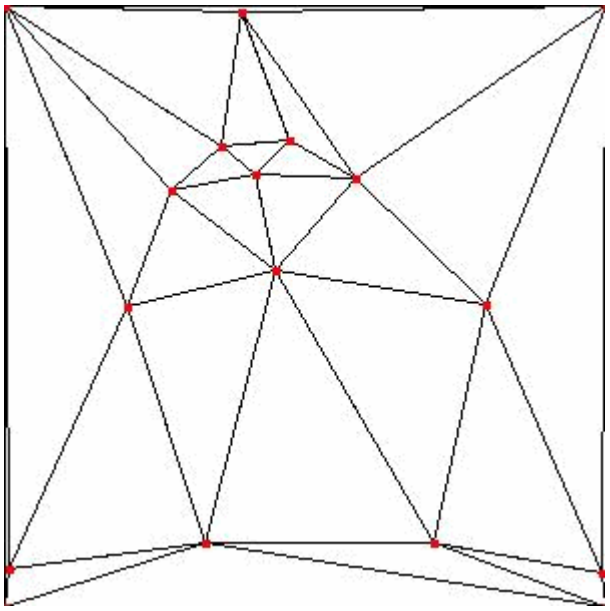
### *Réalisation de la méthode :*

Sur le principe simple, cette transition nous a posé quelques soucis au départ à cause d'une méconnaissance du fonctionnement de transition. En effet, à l'origine, nous réalisions une nouvelle triangulation à chaque étape. Cela avait pour résultat de recalculer les arêtes à chaque itération, d'en rajouter/enlever à certains endroits. Si sur le principe cela ne semble pas être problématique, cela fausse complètement la coloration à l'étape suivante, et crée des artefacts visuels désagréables.



La conséquence directe est donc la perte d'information sur le positionnement du triangle : dans le GIF de gauche, on peut voir des apparitions/disparitions d'arêtes due à un nouveau calcul de triangulation de Delaunay, ce qui provoque logiquement des pertes d'informations sur les textures au niveau de la coloration. (Remarque : ce n'est pas le seul problème présent sur le morphisme ci-dessus.)

Sur les GIFs ci-dessous, nous avons corrigé notre triangulation, cependant il reste un problème. En effet, l'image finale est difforme. Nous avons réalisé ensuite que la raison de cette difformité était liée au fait que nous refaisons une triangulation lors de la dernière étape.



Finalement, après avoir enlevé cette dernière triangulation et bien placé nos points de contrôle, nous obtenons le résultat suivant :

*Coloration :*

Afin de réaliser la coloration de notre morphisme, nous avons utilisé le concept de coordonnées barycentriques. Ces dernières sont particulièrement adaptées à notre problème de morphisme, car les coordonnées d'un point calculé avec des coordonnées barycentriques changeront peu au cours de notre interpolation de triangles et de points. Cela permet de garder une cohérence au niveau du calcul de pixels de couleurs et leurs changements de coordonnées au fil du temps.

Nous appliquons donc la formule suivante, où P est un point quelconque appartenant à notre triangle, et A, B, C nos sommets du triangle respectifs.

$$\begin{cases} \alpha = \frac{Aire(PBC)}{Aire(ABC)} \\ \beta = \frac{Aire(APC)}{Aire(ABC)} \\ \gamma = \frac{Aire(ABP)}{Aire(ABC)}. \end{cases} \qquad \alpha + \beta + \gamma = 1$$

*https://fr.wikipedia.org/wiki/Coordonn%C3%A9es_barycentriques*

En théorie nos trois coordonnées respectent la condition précédente, mais en pratique, il arrive qu'il y ait des "outliers" pour plusieurs raisons différentes : (points sur le triangle, perte d'information…). Cela peut donc provoquer des artefacts visuels sur le morphisme que l'on modélisera par défaut par des pixels noirs.

Pour calculer le changement de couleur subit par notre pixel, nous utilisons la formule suivante :

$$((1-t)r + tr', (1-t)g + tg', (1-t)b + tb'$$

Ici, *t* correspond au ratio entre notre étape actuelle du morphisme et la dernière étape : r, g, b représentent les composantes RGB de notre pixel de départ à la position (x,y) et r', g', b' représentent les composantes RGB de notre pixel de fin. La formule nous assure donc une transition propre (sur le papier) entre nos pixels.
Pour pouvoir garder en mémoire la position de départ et finale de chaque pixel, on regardera dans quel triangle est un pixel et on modifiera son aspect et sa couleur selon les transformations que subira le triangle dans lequel il est contenu.

Pour le diagramme de classe : les classes triangle et point sont utilisés dans les classes du morphing complexe. À noter que l'on récupère la classe "Point" utilisée dans les morphing précédent en lui ajoutant un index, qui est initialisé à zéro dans le constructeur si non précisé (pour le cas d'un morphisme simple et arrondi).
*3e diagramme de classe : voir dossier Annexes (diagramme_classe_part_3_complexe.mdj)*

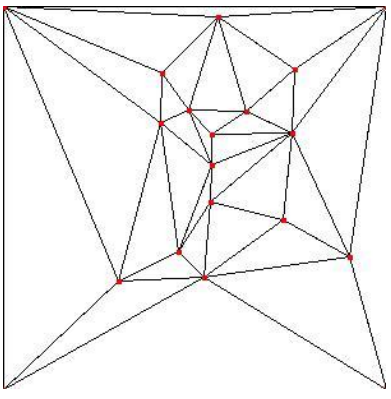
## *Rendu post-morphing :*

Lors du morphing l'application va créer un GIF "resultatMorphing.gif", et dans le même dossier courant un dossier "FormesComplexes", contenant toutes les images intermédiaires du morphing.
Un dossier "TriangulationDelaunay" dans lequel on peut retrouver toutes les étapes de notre triangulation et interpolations est aussi créé, depuis les points de départ choisis par notre utilisateur pour la figure de gauche, jusqu'à leur emplacement final sur la figure de droite. Cela nous permet de montrer l'approche mathématique à ceux qui souhaiteraient comprendre le morphing complexe.

## *Limitations techniques :*

Comme expliqué précédemment, il arrive que des artefacts visuels se glissent dans notre morphisme d'image. Si nous faisons de notre mieux pour les limiter, ils apparaissent parfois, et semblent reproduire en partie les arêtes de nos triangles.





*Morphisme initial (25 Frames)* *- Frame 22 du morphisme* *- Frame 22 de la triangulation*

Les pixels noirs en bas à gauche de la frame 22 sont des artefacts visuels qui ne semblent pas faire sens avec la triangulation de Delaunay de la même frame.
Une autre source d'exception apparente peut apparaître lors du positionnement des points de contrôle trop proches de la bordure de notre image. À noter que cette erreur ne survient pas lorsque l'on place les points de contrôle manuellement.

# Répartition des tâches

Concernant la répartition des tâches, nous avons pris l'habitude, chaque matinée, de discuter des réalisations accomplies, de faire le point sur les prochaines étapes à entreprendre, de répondre aux éventuelles interrogations, et d'attribuer de nouvelles tâches si les précédentes ont été terminées. Ces sessions matinales servent également à assurer une communication claire et cohérente entre les membres de l'équipe, et à unifier le code si nécessaire, notamment lorsque des méthodes ont été finalisées. Dans un souci de collaboration et de compréhension mutuelle, nous veillons à ce que chaque membre participe activement au développement du code et acquière une compréhension approfondie de l'ensemble du projet.

Nous avons également tenu à jour un fichier Google Sheets dans lequel nous renseignons les tâches quotidiennes faites par chacun. *Voir dossier Annexes (repartitionTravail.pdf)*
De plus, nous avons organisé notre travail autour d'un Projet GitHub commun, sur lequel chacun possédait sa branche afin de travailler ou modifier des parties de l'application principale.

Après avoir pris connaissance du sujet, nous avons brainstormé pour décider de notre approche. Ensemble, nous avons travaillé sur la première phase du morphing simple afin de comprendre le principe du morphing et les attentes du projet. Ensuite, nous avons réparti le travail en fonction des différentes méthodes à implémenter, en veillant à avancer sans prendre de retard et en maintenant une bonne communication pour éviter les conflits de code. Nous avons créé au début du projet une première version fonctionnelle de l'interface, que nous avons modifiée en fin de projet pour la rendre plus pratique pour l'utilisateur.

# Bilan personnel

## *Plaut-Aubry Baptiste*

Ce projet sur le morphisme d'image aura été très enrichissant et il m'aura beaucoup appris, tant sur le plan des connaissances brutes (Java, JavaFX) que sur le plan du travail en équipe. J'ai trouvé le sujet plutôt pertinent car à l'instar du projet météo en 2ème année de prépa qui nous donnait un avant goût de la Data, celui-ci nous permet de faire connaissance avec le concept d'IA, d'une certaine manière. (même si ça n'est pas aussi poussé que le projet de reconnaissance faciale des GMI sur ce domaine bien évidemment).

Cela aura aussi été l'occasion de mettre en pratique l'apprentissage du Java que nous avons reçu cette année avec nos premiers pas dans la programmation objet.
La complexité du projet nous aura obligé à faire des recherches approfondies, partager et rechercher ensembles sur des sujets communs complexes, chose que l'on a assez peu l'occasion de faire en temps normal.

De plus, la complexité et l'envergure du projet nous a poussé à utiliser et comprendre des notions dont on pouvait peut être négliger l'importance en cours (l'utilité des toString pour les nombreux débuggages, les moyens de réduire la répétition de code pour une meilleure lisibilité, le transtypage..).

Le travail en groupe avec des membres choisis aléatoirement aura été inhabituel mais aura été utile pour nous pousser à réfléchir à des moyens de fonctionner ensemble efficacement et sans habitudes préétablies.

## *Garcia Lukas*

Ce projet a été très intéressant pour ma part, car il m'a permis de voir une utilisation du java pour un rendu concret, et visuel grâce au JavaFX. J'ai également été amené à me plonger dans les différentes documentations des librairies, réflexe et habitude que je n'avais pas bien développés jusque là mais qui maintenant me semble comme une évidence lors de l'utilisation de celles-ci. De plus, ce projet m'a permis de me rendre compte de l'importance de la réutilisation du code et de l'utilité des classes abstraites.
L'utilisation de nos compétences développées en AOO pour la réalisation de diagrammes de classes nous aura également été extrêmement utile, car ces diagrammes nous ont permis de constater que nous partions sur une mauvaise route pour la réalisation du projet.

Le travail en groupe a été géré assez facilement bien que les groupes aient été aléatoires, car nous avons pris rapidement l'habitude de faire des réunions chaque matinée pour faire un état de l'avancement, des difficultés de chacun et de ce qu'il nous restait à faire. L'entraide a été plus qu'utile car elle nous a offert plusieurs points de vues et plusieurs réflexions différentes sur une même problématique.

## *Philipponneau Thomas*

La réalisation de ce projet en Java et JavaFX a considérablement enrichi mes connaissances et compétences.
J'ai eu la chance de pouvoir approfondir ma maîtrise des concepts fondamentaux de Java et j'ai appris à structurer le code de manière plus efficace pour que l'ensemble des membres du groupe puissent comprendre et modifier le code si besoin. Nous avons séparé les tâches pour parvenir le plus rapidement possible à la résolution du problème donc la structure du code est un pilier fondamental pour une bonne compréhension et un bon partage des tâches au sein du groupe.
En travaillant avec JavaFX, j'ai découvert comment gérer des événements et utiliser leurs propriétés pour rendre les interfaces plus dynamiques et ergonomiques.

Le travail en groupe a été particulièrement bénéfique, permettant un échange constant de connaissances et de compétences qui m'ont permis de progresser considérablement en JavaFX. Le travail de groupe m'a aussi permis d'obtenir des points de vue extérieurs lorsque je me retrouvais bloqué.

De plus, une collaboration étroite avec l'ensemble des membres du groupe m'a permis de travailler et d'améliorer ma communication et ma capacité à coordonner les tâches, ce qui a été crucial pour respecter les délais et assurer la qualité du projet.

Un aspect particulièrement fascinant du projet a été le développement d'une fonctionnalité de morphing de formes simples, arrondies et d'images. Me permettant de découvrir tout un pan du traitement d'image. En somme, ce projet a été une expérience extrêmement enrichissante, m'offrant l'opportunité de développer de nouvelles compétences techniques et de bénéficier des avantages du travail collaboratif.

## *Destouches Nolan*

Ce projet m'a permis d'évoluer sur plusieurs plans, tout d'abord le sujet proposé m'a dans un premier temps un peu effrayé car je ne voyais pas comment résoudre ce type de problème à l'aide des outils mis à notre disposition. Cependant, au fil du temps, j'ai pu me rendre compte de la puissance du Java et du JavaFX dans un cas concret, nous devions trouver nos propres réponses en partant de nos connaissances en programmation et cela a fonctionné.

Grâce à la richesse de ce langage et des informations que l'on peut trouver sur Internet nous avons pu trouver des solutions efficaces et nous surpasser. Je sens que je suis beaucoup plus à l'aise avec le Java et le JavaFX en règle générale et que ma façon de réfléchir en programmation orienté objet est plus fluide. De plus, certaines notions jusqu'alors assez floues pour moi sont maintenant plus claires.

En ce qui concerne le travail en groupe, le fait de devoir produire quelque chose avec des personnes avec lesquelles on n'a pas l'habitude de travailler et que l'on ne connaît pas forcément m'aura aidé à améliorer ma communication. En effet, même si le début n'était pas très fluide et efficace, à force de discuter à propos de ce qui allait et de ce qui n'allait pas, les choses se sont améliorées et ont fini par créer un groupe homogène dans lequel chacun sait ce qu'il a à faire et le fait. De plus, j'ai pu utiliser pour la première fois GitHub et apprendre comment ce type de service fonctionne.

## *Barrouillet Margaux*

Ce projet de morphing en Java a été une opportunité pour appliquer les notions vues en cours dans un contexte concret. Il m'a permis de comprendre l'importance de l'optimisation et de la structuration du code, ainsi que la nécessité de bien réfléchir avant de commencer à coder afin d'éviter les faux départs.

J'ai également consolidé mes compétences en JavaFX et approfondi mes connaissances sur les contrôleurs pour mieux gérer les interactions entre les différentes parties de l'application, notamment lors de la mise en place des points de contrôle pour les formes arrondies, ce qui fut complexe mais enrichissant.

Pour ce qui est du travail en équipe, nous avons réussi à bien communiquer et à nous organiser efficacement pour mener à bien ce projet, même si nous n'avions jamais travaillé ensemble auparavant. Le sujet du morphing était particulièrement intéressant. J'ai beaucoup appris sur les différents types de morphing, les calculs nécessaires et les techniques associées. Cela m'a permis de découvrir des aspects techniques et théoriques que je connaissais peu voire pas du tout.

# Références


1.Princeton University
**Feature-Based Image Metamorphosis : [en ligne]**,
Thaddeus Beier & Shawn Neely
https://www.cs.princeton.edu/courses/archive/fall00/cs426/papers/beier92.pdf

2.MIT - EECS
**Image Warping and Morphing : [en ligne]**, Frédo Durand, Bill Freeman
https://groups.csail.mit.edu/graphics/classes/CompPhoto06/html/lecturenotes/14_WarpMorph.pdf

3.Stellenbosch University
**Morphing in Two Dimensions: Image Morphing : [en ligne]**, Magdil Delport
https://scholar.sun.ac.za/server/api/core/bitstreams/7bd8bcd1-07da-44d9-80ad-bde89af4fddc/content

4.**Triangulation de polygones : [en ligne]**, Achraf Othman
https://www.achrafothman.net/docs/mesh3d.2tnsi.chapitre%204.pdf

5.FernUniversitat Hagen,Universität Bonn
**Java Applets for the Dynamic Visualization of Voronoi Diagrams : [en ligne],**
https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=ae3821f8345c87ad452ae11f2cd49148ae68d329

6.**Voronoi diagrams and natural neighbor interpolation : [en ligne],**
Mike Bourgeous
https://blog.mikebourgeous.com/2021/06/09/voronoi-diagrams-and-natural-neighbor-interpolation-explained-in-video/

7. Université de Marne la Vallée, ESIPE
**Projet de programmation C - Morphing : [en ligne]**
http://www-igm.univ-mlv.fr/~borie/esipe/morphing.pdf

8. Gorilla Sun
**Bowyer-Watson Algorithm for Delaunay Triangulation : [en ligne]**,
Ahmad Moussa
https://www.gorillasun.de/blog/bowyer-watson-algorithm-for-delaunay-triangulation/

9.GeekForGeeks, VIT AP University Amaravati
**Image Warping and Morphing : [en ligne],**"preetikintali"
https://www.geeksforgeeks.org/image-warping-and-morphing/

10.Wikipedia
**Algorithme de tracé de segment de Bresenham : [en ligne]**
https://fr.wikipedia.org/wiki/Algorithme_de_trac%C3%A9_de_segment_de_Bresenham

11. Wikipedia
**Coordonnées barycentriques : [en ligne]**
https://fr.wikipedia.org/wiki/Coordonn%C3%A9es_barycentriques

12.Wikipedia
**Courbe de Bézier : [en ligne]**
https://fr.wikipedia.org/wiki/Courbe_de_B%C3%A9zier

13.GeekForGeeks
**Bresenham's Line Generation Algorithm : [en ligne],**Shivam Pradhan
https://www.geeksforgeeks.org/bresenhams-line-generation-algorithm/