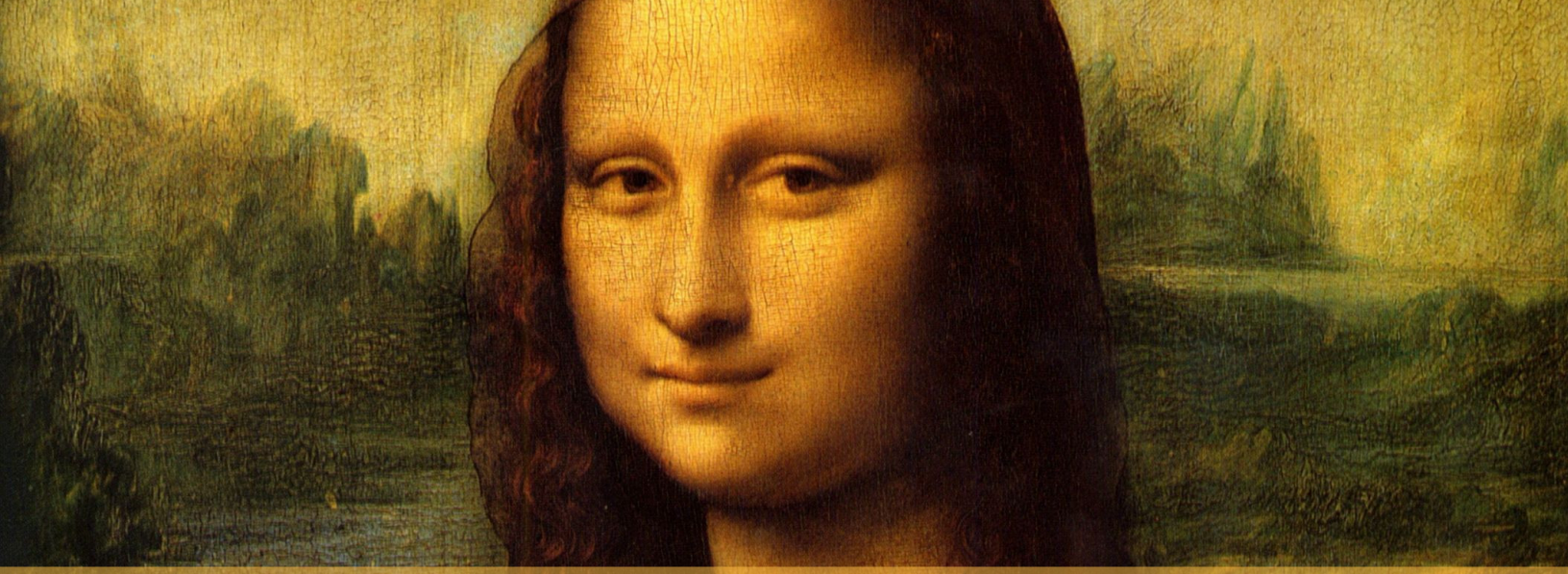

Reconstitution de la Joconde grâce à des polygones

Monte-Carlo et Regroupement de pixels

Lothair Kizardjian

PROJET D'IA HIVER 2018, UNIVERSITÉ PARIS DAUPHINE

HTTPS://GITHUB.COM/LOTHAIRKIZARDJIAN/AIPROJECT

1. Monte-Carlo

1.1 Generation d'une solution

Avant de me lancer de l'écriture du code du projet j'ai d'abord effectué quelques recherches sur internet pour voir ce qui avait déjà été fait et quelles méthodes avaient été employés. J'ai trouvé le site suivant : https://alteredqualia.com/visualization/evolve/ qui lui même s'est inspiré du code de Roger Alsing mais l'a adapté. En effet il n'utilise pas d'algorithme génétique mais part d'un invidivu qu'il fait muter aléatoirement et garde la meilleur version qu'il continue de faire muter, c'est donc une application d'une méthode de Monte-Carlo. J'ai décidé d'adapter cette solution au problème qui m'était donné à savoir :

> "Construire une image la plus semblable possible à une image cible. En particulier le but du projet est de trouver une image qui sera le plus semblable possible au tableau la Joconde de Léonard de Vinci à l'aide d'au plus 50 polygones convexes."

Je suis donc parti d'une liste de n (compris entre 0 et 50) polygones convexes ayant un nombre de sommets ainsi qu'une couleur et une opacité aléatoire (cf Figure 1.1).

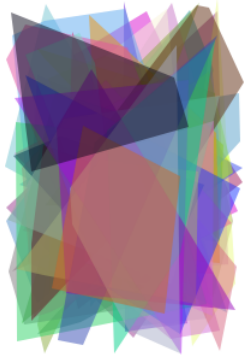

Figure 1.1: Exemple de génération d'un Individu

1.2 Mutations

Ce groupe de polygones représente donc un *Individu*, qui est une solution au problème donné. J'ai ensuite fait subir à cet individu des modifications. Ces modifications se font par l'intermédiaire de *normal*mutations qui peuvent être très variées. J'ai réparti ces différentes mutations dans 3 catégories bien distinctes :

- les *softMutations* : Ces mutations ne modifient qu'un seul paramètre d'un polygone aléatoire (R,G,B,A,X,Y) d'un très faible pourcentage par rapport à la valeur qu'ils possèdent. R,G et B correspondant à la couleur du polygone, A son opacité et finalement X et Y la coordonné x ou y d'un sommet du polygone.
- les *mediumMutations* : Ces mutations réatribuent à un seul paramètre d'un polygone aléatoire une nouvelle valeur aléatoire. Elles peuvent aussi ajouter ou supprimer un sommet à un polygone ou leur faire subir une tranlsation.
- les *hardMutations* : Ces mutations réatribuent une couleur et une opacité aléatoire à un polygone en plus de modifier les coordonnées X et Y d'un sommet aléatoire de ce polygone. Elles peuvent aussi supprimer un polygone aléatoire ou ajouter un nouveau polygone aléatoire.

L'idée est donc de faire subir à l'Individu de départ une mutation, si l'individu obtenu après la mutation est meilleur qu'avant (c'est à dire qu'il ressemble plus à l'image objectif) alors on le garde et continue les mutations sur ce nouvel individu. On répète ensuite ce schéma jusqu'à obtenir une solution qui est acceptable.
J'ai aussi remarqué que l'utilisation uniquement des mediumMutations permettaient d'obtenir les meilleurs résultats le plus rapidement donc je n'utilise ni les softMutation ni les hardMutation.
Cependant un problème ce pose : lorsqu'un polygone subit une mutation qui change sa forme il se peut qu'il ne soit plus convexe s'il possède plus de 3 sommets. Après quelques recherches j'ai pu trouver un algorithme qui permet de calculer l'enveloppe convexe d'un groupe de points, c'est *La marche de Graham* (j'ai utilisé une version codée en Java qui était disponible en opensource sur Github, cf bibliographie). Cet algorithme renvoie un nouvel ensemble de points qui contient tout ceux passés en paramètres et permet donc de construire un polygone convexe après mutation sur sa forme. Je recalcule donc l'enveloppe convexe du polygone muté si sa forme a été modifiée ce qui me permet de ne conserver que des polygones convexes. Avec cet algorithme j'obtient des résultats assez satisfaisant :

Figure 1.2: Résultat avec 50 polygones de 7 sommets max en 27 minutes

1.3 Optimisation des solutions de bases

Cependant j'ai remarqué que selon l'Individu de base les résultats finaux peuvent être plus ou moins bons. En effet on peut converger plus rapidement si on part avec un Individu de base qui possède des polygones ayant déjà des couleurs ou des formes acceptables. J'ai donc modifié certains paramètres lors de la génération des polygones convexes : premièrement ils sont tous invisibles (opacité 0). Ensuite ils peuvent soit commencer avec des couleurs aléatoires (pas de changement) soit tous être blanc soit tous être noir. Pour la joconde j'ai remarqué que s'ils commencent tous en étant noir l'algorithme converge plus rapidement; en effet comme l'image possède surtout des couleurs sombres si on commence avec des polygones noirs la fitness sera déjà bien plus proche de l'image objectif.

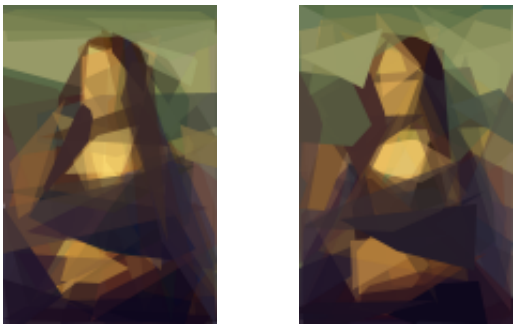

Figure 1.3: Respectivement après 15 minutes et 20 minutes de calcul

Au vu de ces résultats j'ai continué à chercher d'autres possibilité pour améliorer ma solution de base. C'est alors que j'ai pensé à faire du regroupement de pixels. Créer n groupes de pixels qui contiennent initialement un pixel pris aléatoirement sur l'image (on ne peut pas prendre plusieurs fois le même pixel) puis pour chaque groupe de pixels ajouter ses voisins qui n'appartiennent pas déjà dans un groupe si sa couleur n'est pas trop différente de la couleur moyenne du groupe de pixel concerné. Une fois que tous les pixels appartiennent à un groupe, pour chaque groupe de pixel je génère une enveloppe convexe qui regroupe chaque pixel du groupe et je créé un nouvel Individu qui aura ces polygones comme liste de départ.
Pour avoir une répartition uniforme des groupes de pixels sur l'image j'ai utilisé le *poisson sampling* qui n'ajoute un nouveau groupe de pixel que si la distance entre celui-ci et tout les autres est supérieur à *delta*, le problème est de trouver le delta approprié, pour l'instant je lui donne des valeurs arbitraires car je n'ai pas trouvé de formule adéquate pour le calculer. Mes premiers résultats ne sont pas très concluant mes permettent tout de même d'obtenir des images assez ressemblantes où on commence déjà à distinguer un peu plus de détails :

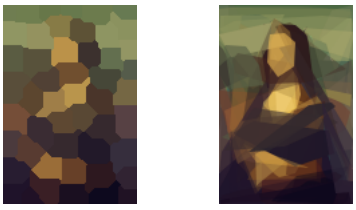

Figure 1.4: Respectivement le cluster et le résultat obtenu depuis ce cluster après 100 minutes

Cependant après optimisation du code j'ai réussi à obtenir un cluster beaucoup plus détaillé et qui permet d'obtenir une solution de base très ressemblante à l'image objectif et ainsi je commence l'algorithme avec une fitness très basse qui met permet d'obtenir en quelques minutes une solution très satisfaisante :

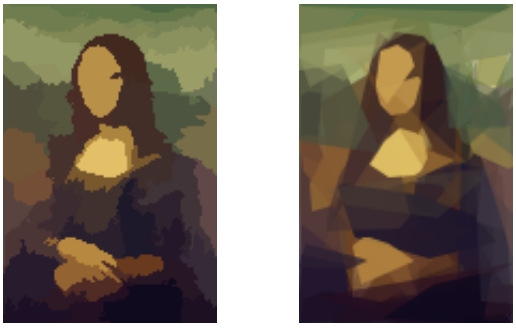

Figure 1.5: Respectivement le cluster et le résultat obtenu depuis ce cluster après 10 minutes

Voici le résultat pour l'image en dimension 200x298 (les précédentes étant pour la 100x149) :

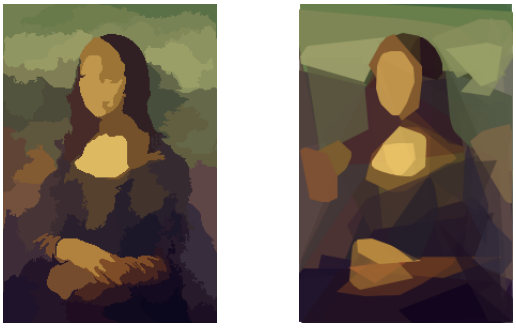

Figure 1.6: Respectivement le cluster et le résultat obtenu depuis ce cluster après 2 minutes

1.4 Difficultés

Lorsque j'ai commencé ce projet j'étais en premier lieu parti sur de l'algorithme génétique mais j'ai abandonné car je n'arrivais pas à avoir des résultats ressemblant, de plus mon algorithme était assez lent. D'autre part, trouver les mutations à faire et comment les faire pour respecter les contraintes (pas plus de 50 polygones, ils doivent être convexe ... etc) s'est revélé prendre beaucoup de temps, mais le fait d'avoir trouver une version de l'algorithme de la marche de Graham codé en Java a considérablement réduit ce temps. Finalement, pour ce qui est du clustering j'ai du apprendre à utiliser les Threads en JavaFX car les Threads simple de Java ne sont pas supportés par JavaFX. De plus trouver un algorithme de clustering assez rapide était tout aussi compliqué, en effet mes première versions étaient très lentes et prennaient beaucoup trop de temps pour une image 200x298. L'algorithme actuel permet d'avoir un temps de calcul raisonnable pour cette même dimension d'image mais pour des plus grandes le temps de calcul augmente considérablement.

Personnellement je n'ai pas trouvé ce projet très difficile, je l'ai surtout trouvé intéressant, il m'a donné envie de m'investir et de me renseigner, j'ai d'ailleurs beaucoup appris.

1.5 Bibliographie

Algorithme de mutation d'un individu composé de polygones :
https://alteredqualia.com/visualization/evolve/

Calcul d'une enveloppe convexe d'un polygone :
https://fr.wikipedia.org/wiki/Calcul_de_l'27enveloppe_convexe

Code source de l'algorithme de la marche de Graham :
https://github.com/bkiers/GrahamScan

Répartition uniforme de point dans un espace (poissin sampling):
https://www.cs.ubc.ca/~rbridson/docs/bridson-siggraph07-poissondisk.pdf