

# **PROJET PROGRAMMATION C**

## **Morphing**

## Table des matières

I.Présentation générale.....	2
I.I.Déroulement du programme.....	3
I.II.Modules, fonctions principales et rôles.....	3
Triangulation.....	3
Graphique.....	5
II.Divers.....	5
II.I.Répartition.....	5
II.II.Difficultés.....	6
II.III.Conclusion.....	6

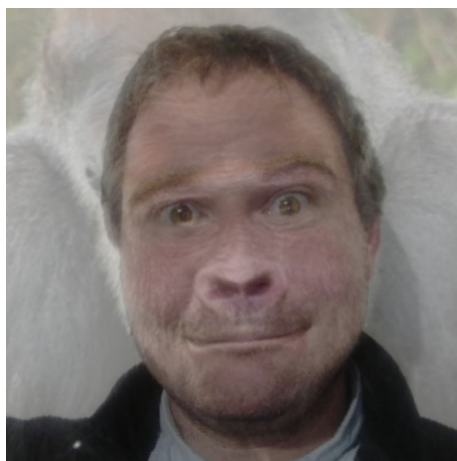
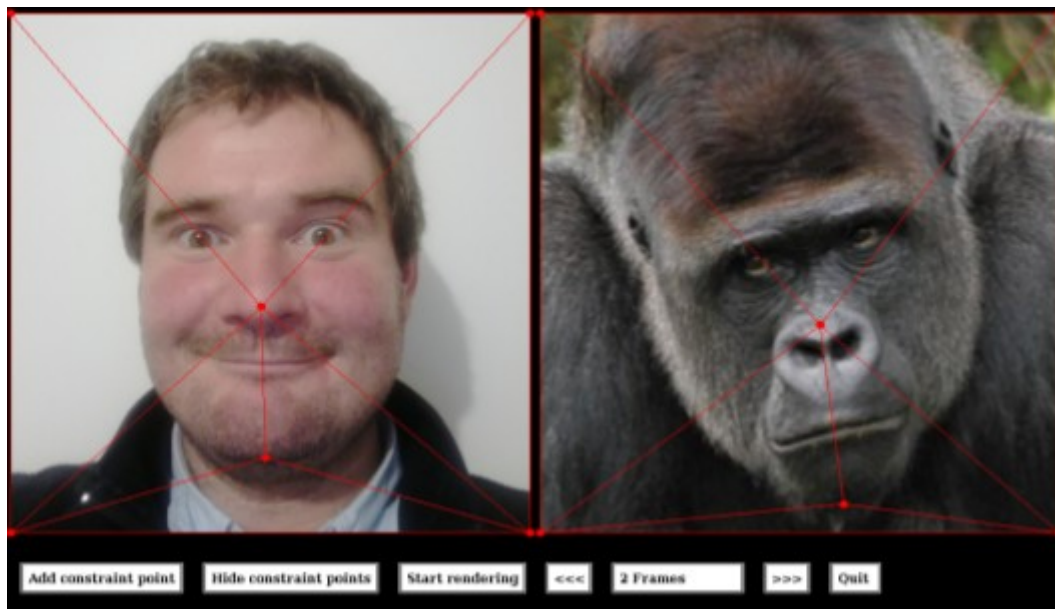
# I. Présentation générale

Le but de ce projet est de créer une application de “morphing”. Le programme permet de mixer deux images en y associant des points d’ancrage similaires (ex : points au niveau d’un œil, sourcil, nez...).

Le programme que nous avons développé ne propose cependant pas toute les fonctionnalités voulues. En effet, nous avons rencontré des difficultés lors de l’implémentation de certaines méthodes, ce qui nous a empêché d’atteindre l’objectif.

Le programme permet d’ouvrir une fenêtre, et de placer les points d’ancrage pour former les premiers triangles.

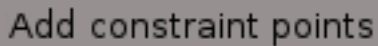
Exemple :



## I.I. Déroulement du programme

Une fois le programme compilé avec la commande *make* on peut lancer le programme avec la commande *./Morphing*. On a alors une fenêtre qui s'ouvre avec deux images qui contiennent chacune deux triangles de base.

Pour rajouter des points, il suffit de cliquer sur le bouton *add constraint points*.



Le programme va attendre que l'utilisateur choisisse un point sur l'image de gauche puis sur l'image de droite. Pendant cette phase, on est obligé de placer les points, les autres actions sont interdites. Une fois les deux points placés, le programme va créer de nouveaux triangles qui serviront par la suite à savoir comment modifier l'image pour arriver au résultat final.

Par la suite on peut continuer à rajouter des points qui vont créer de nouveaux triangles, ou alors quitter le programme en cliquant sur le bouton *Quit*.



Dans ce rendu, notre programme ne rajoute que les triangles. Les fonctionnalités permettant de réaliser les triangulations ont été développés mais leurs implémentations nous ont posé problème (cf. Difficultés rencontrés), ce qui ne les rend pas utilisables.

## I.II. Modules, fonctions principales et rôles

Nous avons décidé de découper le projet en 2 modules (Triangulation et graphique) et un main.

### Triangulation

Dans le module *Triangulation* nous avons tous les algorithmes permettant de calculer ce qui est lié aux triangles. C'est également ici que les structures liées aux triangles sont développées.

```
typedef struct Point
{
    int X,Y;
}Point;
```

La structure *Point* permet de modéliser 2 points x y

```
typedef struct Triangle
{
    Point A, B, C;
}Triangle;
```

La structure *Triangle* permet de modéliser un nouveau triangle grâce à des points de type *Point*.

```
typedef struct TriangleInList
{
    struct TriangleInList* next;
    Triangle current;
}TriangleList;
```

Cette structure permet de créer une liste chaînée de triangles de type *Triangle*.

```
void initializePoint(Point* P, int X, int Y);
```

La fonction *initializePoint*, permet de créer le point que l'utilisateur vient de placer.

```
void initializeTriangle(Triangle* t, Point A, Point B, Point C);
```

La fonction *initializeTriangle* crée un nouveau triangle en fonction de trois points.

```
int pointInTriangle(Point P, Triangle T);
```

La fonction *pointInTriangle* permet de voir si le point placé par l'utilisateur est situé dans un certain triangle.

```
Triangle inThisTriangle(TriangleList* list, Point P);
```

La fonction *inThisTriangle* renvoie le triangle ou l'utilisateur a placé le point.

```
TriangleList* popTriangles(TriangleList* list, Point P);
```

La fonction *popTriangles* permet de former 3 nouveaux triangles en fonction du point placé par l'utilisateur.

```
TriangleList* flip(TriangleList* list, Point P);
```

La fonction *flip* permet de faire flipper l'arête d'un triangle pour remplacer les triangles et de faire une triangulation de Delauney.

Non fonctionnelle

```
TriangleList* initTrianglesRight(TriangleList* listLeft, Point PointLeft, Point PointRight);
```

La fonction *initTrianglesRights* initialise la liste de triangles de l'image de droite en fonction de la liste de triangles de l'image de gauche et des points cliqués par l'utilisateur.

Non fonctionnelle

## Graphique

Le module graphique permet d'afficher le programme.

```
typedef struct Image{  
  — MLV_Image *Image_left;  
  — MLV_Image *Image_right;  
  — MLV_Image *Add_constraint_point;  
  — MLV_Image *Hide_constraint_points;  
  — MLV_Image *Start_rendering;  
  — MLV_Image *Left;  
  — MLV_Image *Right;  
  — MLV_Image *Frames;  
  — MLV_Image *Quit;  
}Image;
```

La structure *Image* permet de définir toutes les images nécessaire au fonctionnement du programme.

Cela permettra de ne charger les images qu'au lancement du programme.

```
Image initImage();
```

La fonction *initImage* permet de charger toutes les images nécessaires au bon fonctionnement du programme.

```
void initFenetre(TriangleList* list_left, TriangleList* list_right);
```

La fonction *initFenetre* permet d'afficher tout le contenu du programme à chaque fois que l'on a une modification de l'utilisateur (c'est-à-dire les différents points placés par l'utilisateur et les triangles que les points forment).

## II. Divers

### II.I. Répartition

Florian RICHARD s'est surtout occupé de la partie codage du module Triangulation tandis que Joachim BOCAZ COEFFE s'est occupé du module graphique.

Nous avons également discuté sur la manière dont on allait implémenter les différentes structures et fonctions selon les points forts de chacun.

## **II.II. Difficultés**

Nous avons rencontré plusieurs difficultés durant le développement du projet. Les premiers problèmes ont été rencontrés lors de la création de la liste des triangles. En effet, nous avons tout de suite opté pour une liste chaînée qui est plus simple à gérer qu'un tableau dynamique. Cependant, cette gestion nous a posé des problèmes lors de l'utilisation de certaines fonctions qui n'accédaient pas à certaines cases mémoires de la liste chaînée à cause des fonctions de gestion de la liste que nous avons mal implémenté.

Une fois ces problèmes réglés, nous avons bloqué sur les flips des triangles. En effet, nous pensons avoir trouvé la manière algorithmique de les gérer mais il nous manque certains éléments qui font que nous ne sommes pas parvenu à les implémenter.

Du fait de l'impossibilité de faire des flips correctement, toute la suite du projet a été mise en échec.

## **II.III. Conclusion**

Ce projet nous a permis de réaliser que certaines problématiques simples de maths peuvent rapidement devenir complexes lors de leurs implémentations dans un langage informatique. Nous avons aussi appris à se départager les tâches dans un temps imparti, nous permettant de comprendre que l'organisation dans le travail est très importante et nous utiliserons cette expérience pour mieux nous organiser lors de nos prochains projets.