

# Morfarea imaginilor

Marc Rareș-Cristian

## Rezumat

Am implementat un sistem de morfare al imaginilor folosind OpenCV și C++. Utilizăm triangularea de tip Delaunay, transformări afine și interpolare liniară pentru a crea tranziții între două imagini.

## Cuprins

<b>1</b>	<b>Introducere</b>	<b>2</b>
1.1	Funcționalități principale . . . . .	2
<b>2</b>	<b>Arhitectura</b>	<b>2</b>
2.1	Componente . . . . .	2
<b>3</b>	<b>Teorie matematică</b>	<b>2</b>
3.1	Interpolarea liniară . . . . .	2
3.2	Triangularea Delaunay . . . . .	2
3.3	Transformarea afină . . . . .	2
<b>4</b>	<b>Algoritmi</b>	<b>2</b>
<b>5</b>	<b>Implementare</b>	<b>3</b>
5.1	Pipeline-ul de procesare . . . . .	3
5.1.1	Inițializarea . . . . .	3
5.1.2	Gestionarea punctelor . . . . .	3
5.2	Triangularea . . . . .	4
5.3	Keybind-uri . . . . .	4
5.4	Îmbunătățiri Viitoare . . . . .	4
<b>6</b>	<b>Referințe</b>	<b>5</b>

# 1 Introducere

Morfarea imaginilor este o tehnică care creează o tranziție între două imagini prin transformarea graduală a geometriei și aspectului imaginii sursă în imaginea destinație.

## 1.1 Funcționalități principale

- Selecția punctelor din cele 2 imagini
- Triangularea și vizualizarea ei
- Previzualizarea rezultatului
- Exportarea rezultatului în format GIF

# 2 Arhitectura

## 2.1 Componente

- **ImageMorpher**: Componenta care implementează morfarea
- **UIController**: Gestionează interfața
- **Main**: Punctul de intrare

# 3 Teorie matematică

## 3.1 Interpolarea liniară

Pentru un parametru de morfare  $\alpha \in [0, 1]$ , pozițiile punctelor intermediare sunt calculate folosind interpolarea liniară:

$$P_{intermediar}(t) = (1 - \alpha) \cdot P_{sursă} + \alpha \cdot P_{destinație} \quad (1)$$

## 3.2 Triangularea Delaunay

Folosim triangularea Delaunay pentru a crea un mesh care:

- Maximizează unghiul minim în fiecare triunghi
- Evită triunghiurile lungi și subțiri, care pot cauza artefacte

## 3.3 Transformarea afină

Fiecare triunghi este "warp-uit" folosind o matrice de transformare afină, adică prezervă liniile și paralelismul:

# 4 Algoritmi

---

**Algorithm 1** Morfarea imaginilor

---

- 1: **Intrare:** Imaginea sursă  $I_s$ , imaginea destinație  $I_t$ , parametrul de morfare  $\alpha$
- 2: **Ieșire:** Imaginea morfată  $I_m$
- 3: Încarcă și redimensionează imaginile
- 4: Calculează punctele intermediare folosind interpolarea liniară
- 5: Generează triangularea Delaunay
- 6: Inițializează imaginile de ieșire  $W_s$  și  $W_t$  ca matrice de zero
- 7: **for** triunghi  $T_i$  în triangulare **do**
- 8:   Ia vârfurile triunghiului sursă  $T_s^i$
- 9:   Ia vârfurile triunghiului țintă  $T_t^i$
- 10:   Ia vârfurile triunghiului intermediar  $T_m^i$
- 11:   Fa warping  $I_s(T_s^i) \rightarrow W_s(T_m^i)$  folosind transformarea afină
- 12:   Fa warping  $I_t(T_t^i) \rightarrow W_t(T_m^i)$  folosind transformarea afină
- 13: **end for**
- 14:  $I_m = (1 - \alpha) \cdot W_s + \alpha \cdot W_t$
- 15: **returnează**  $I_m$

---

---

**Algorithm 2** Warping la triunghi

---

- 1: **Intrare:** Imaginea sursă  $I$ , vârfurile triunghiului  $V_{src}, V_{dst}$
- 2: **Ieșire:** Triunghiul după warp în imaginea destinație
- 3: Calculează bounding boxes pentru triunghiurile sursă și destinație
- 4: Calculează matricea de transformare afină  $M$  de la  $V_{src}$  la  $V_{dst}$
- 5: Aplică warping afin în regiunea triunghiului sursă
- 6: Copiază pixelii după warp în destinație folosind masca (masca de la bounding box spune care pixeli fac parte din triunghi)

---

## 5 Implementare

### 5.1 Pipeline-ul de procesare

#### 5.1.1 Inițializarea

```
1 bool ImageMorpher::initialize(const std::string& sourcePath,
2                               const std::string& targetPath) {
3     sourceImg = cv::imread(sourcePath);
4     targetImg = cv::imread(targetPath);
5
6     if (sourceImg.cols > MAX_IMAGE_WIDTH) {
7         float ratio = static_cast<float>(MAX_IMAGE_WIDTH) / sourceImg.
8             cols;
9         cv::resize(sourceImg, sourceImg, cv::Size(), ratio, ratio);
10    }
11
12    if (sourceImg.size() != targetImg.size()) {
13        cv::resize(targetImg, targetImg, sourceImg.size());
14    }
15
16    return true;
17 }
```

#### 5.1.2 Gestionarea punctelor

Mentinem perechi de puncte corespunzătoare între imaginile sursă și destinație:

```

1 void ImageMorpher::addCorrespondingPoints(const cv::Point2f&
    sourcePoint,
2                                     const cv::Point2f& targetPoint
    ) {
3     sourcePoints.push_back(sourcePoint);
4     targetPoints.push_back(targetPoint);
5     triangles.clear();
6 }

```

## 5.2 Triangularea

```

1 bool ImageMorpher::calculateTriangulation() {
2     if (sourcePoints.size() < 3) return false;
3
4     cv::Rect rect(0, 0, sourceImg.cols, sourceImg.rows);
5     std::vector<cv::Point2f> points = sourcePoints;
6
7     points.push_back(cv::Point2f(0, 0));
8     points.push_back(cv::Point2f(rect.width - 1, 0));
9     points.push_back(cv::Point2f(rect.width - 1, rect.height - 1));
10    points.push_back(cv::Point2f(0, rect.height - 1));
11
12    cv::Subdiv2D subdiv(rect);
13    for (const auto& point : points) {
14        subdiv.insert(point);
15    }
16
17    std::vector<cv::Vec6f> triangleList;
18    subdiv.getTriangleList(triangleList);
19
20    return true;
21 }

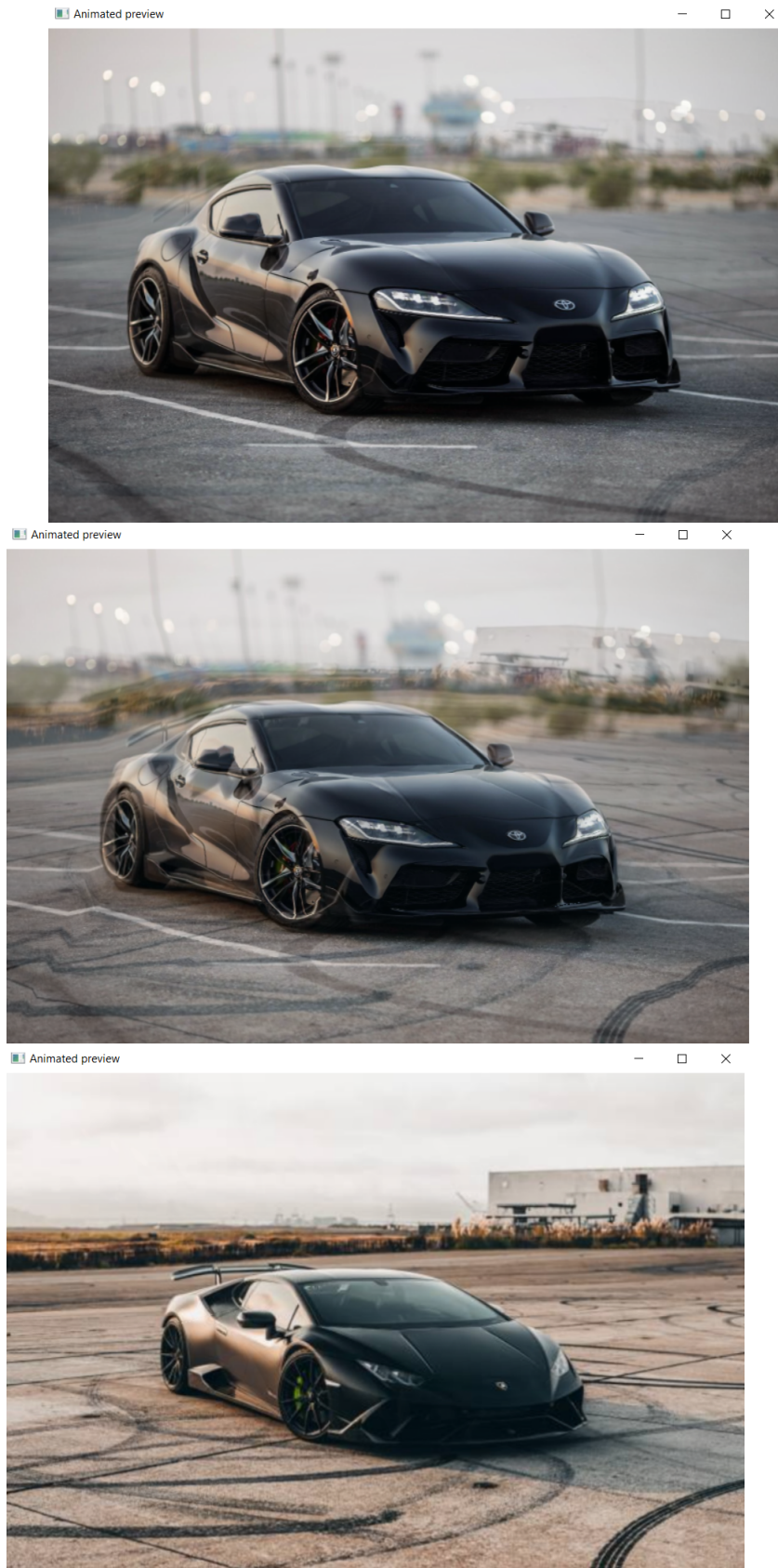
```

## 5.3 Keybind-uri

Tastă	Funcție
S	Salvează punctele în fișier
L	Încarcă punctele din fișier
Z	Scoate ultima pereche de puncte
T	Afișează triangularea
C	Șterge toate punctele
A	Preview
G	Generează GIF-ul
X	Șterge fișierele generate local

## 5.4 Îmbunătățiri Viitoare

- Funcții de interpolare non-liniare (sigmoid)
- Detectia automată a trăsăturilor
- Calcule pe GPU



## 6 Referințe

1. <https://sauravmittal.github.io/computational-photography/image-morphing/>

2. [https://en.wikipedia.org/wiki/Delaunay\\_triangulation](https://en.wikipedia.org/wiki/Delaunay_triangulation)
3. [https://docs.opencv.org/4.x/df/d5b/group\\_imgproc\\_\\_subdiv2d.html](https://docs.opencv.org/4.x/df/d5b/group_imgproc__subdiv2d.html)
4. <https://ffmpeg.org/>