CG_2_ImageWarping Report

PB21010479 Caoliwen Wang

March 10, 2024

1 Problem Statement

We need to implement image warping. We have implemented interactive mouse dragging, allowing users to designate certain points as fixed and move others to desired positions. These selected points are collectively referred to as control points$(\boldsymbol{p}_i, \boldsymbol{q}_i)$, $\boldsymbol{p}_i, \boldsymbol{q}_i \in \mathbb{R}^2$, $i = 1, \ldots, n$. Taking these control points as input, our goal is to output the deformed entire image. Essentially, the problem entails determining a distortion function $\boldsymbol{f}$ given n interpolation conditions:

$$\boldsymbol{f}(\boldsymbol{p}_i) = \boldsymbol{q}_i, i = 1, \ldots, n$$

We then apply $\boldsymbol{f}$ to all points in the image to determine the positions of the deformed points. Finally, we assign the pixel values of the points before distortion to the corresponding points after distortion, achieving the rendering of the deformed image.

2 Propose Algorithms

The essence of this problem lies in finding the interpolation function. Here, we present three different approaches:

1. IDW: Inverse distance-weighted interpolation methods;
2. RBF: Radial basis functions interpolation method;
3. MLS: Moving least squares interpolation method.

I need to clarify that the Moving Least Squares (MLS) method is an assignment in a mathematical experiment course. Here, it has been implemented once again using C++.

2.1 IDW

We need to define some local interpolation functions $\boldsymbol{f}_i(\boldsymbol{p}) : \mathbb{R}^2 \to \mathbb{R}^2$, which satisfy the following conditions:

$$\boldsymbol{f}_i(\boldsymbol{p}_i) = \boldsymbol{q}_i.$$

To be more specific,

$$\boldsymbol{f}_i(\boldsymbol{p}) = \boldsymbol{q}_i + \boldsymbol{D}_i(\boldsymbol{p} - \boldsymbol{q}_i). \tag{2.1}$$

In equation (2.1), $\boldsymbol{D}_i : \mathbb{R}^2 \to \mathbb{R}^2$, which satisfies that $\boldsymbol{D}_i(\boldsymbol{0}) = \boldsymbol{0}$.

Choose $\boldsymbol{D}_i$ to be a linear transformation. We can define it in a more accurate way. Define the energy function:

$$E_i(\boldsymbol{D}_i) = \sum_{j=1, j\neq i}^{n} \sigma_i(\boldsymbol{p}_j)\|\boldsymbol{q}_i + \boldsymbol{D}_i(\boldsymbol{p}_j - \boldsymbol{p}_i) - \boldsymbol{q}_j\|^2. \tag{2.2}$$

Let the $\boldsymbol{D_i}$ minimize the energy function.

$$\frac{\partial E_i}{\partial \boldsymbol{D}_i} = \sum_{j=1, j\neq i}^{n} \sigma_i(\boldsymbol{p}_j)[\boldsymbol{q}_i + \boldsymbol{D}_i(\boldsymbol{p}_j - \boldsymbol{p}_i) - \boldsymbol{q}_j] \cdot (\boldsymbol{p}_j - \boldsymbol{p}_i)^\top = 0. \tag{2.3}$$

So $\boldsymbol{D_i}$ can be calculated in the following way.

$$[\sum_{j=1, j\neq i}^{n} \sigma_i(\boldsymbol{p}_j)(\boldsymbol{q}_j - \boldsymbol{q}_i)(\boldsymbol{p}_j - \boldsymbol{p}_i)^\top] \cdot [\sum_{j=1, j\neq i}^{n} \sigma_i(\boldsymbol{p}_j)(\boldsymbol{p}_j - \boldsymbol{p}_i)(\boldsymbol{p}_j - \boldsymbol{p}_i)^\top]^{-1} \tag{2.4}$$

We use the Eigen library to compute the inverse of a matrix.
The complete interpolation function is

$$\boldsymbol{f}(\boldsymbol{x}) = \sum_{i=1}^{n} \omega_i(\boldsymbol{x})\boldsymbol{f}_i(\boldsymbol{x}) \tag{2.5}$$

$\omega_i : \mathbb{R}^2 \to \mathbb{R}$ is defined:

$$\omega_i(\boldsymbol{x}) = \frac{\sigma_i(\boldsymbol{x})}{\sum_{j=1}^{n} \sigma_j(\boldsymbol{p})}. \tag{2.6}$$

$$\sigma_i(\boldsymbol{x}) = \frac{1}{\|\boldsymbol{x} - \boldsymbol{x}_i\|^\mu} \tag{2.7}$$

μ is a constant.

2.2 RBF

Assuming the interpolation function f is in the form of a combination of radial basis functions as follows:

$$f(\boldsymbol{p}) = \sum_{i=1}^{n} \boldsymbol{\alpha}_i R(\|\boldsymbol{p} - \boldsymbol{p}_i\|) + \boldsymbol{A}\boldsymbol{p} + \boldsymbol{b}. \tag{2.8}$$

In equation (2.8),

1. $R(\|\boldsymbol{p} - \boldsymbol{p}_i\|)$ consists of n radial basis functions, such as the option $R(d) = (d^2 + r^2)^{\mu/2}$, where the coefficients $\boldsymbol{\alpha}_i \in \mathbb{R}^2$ are to be determined;

2. $\boldsymbol{A} \in \mathbb{R}^{2\times 2}$ and $\boldsymbol{b} \in \mathbb{R}^2$ are the to-be-determined affine parts.

The mapping has $2(n + 3)$ degrees of freedom, and the interpolation conditions are:

$$f(\boldsymbol{p}_j) = \sum_{i=1}^{n} \boldsymbol{\alpha}_i R(\|\boldsymbol{p}_j - \boldsymbol{p}_i\|) + A\boldsymbol{p}_j + \boldsymbol{b} = \boldsymbol{q}_j, \quad j = 1, \ldots, n. \tag{2.9}$$

Provided are $2n$ constraints. Optional additional constraints include:

$$\begin{pmatrix} \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_n \\ 1 & \cdots & 1 \end{pmatrix}_{3\times n} \begin{pmatrix} \boldsymbol{\alpha}_1^\top \\ \vdots \\ \boldsymbol{\alpha}_n^\top \end{pmatrix}_{n\times 2} = \boldsymbol{0}_{3\times 2}. \tag{2.10}$$

Based on the above constraints, we can formulate the following matrix equation to solve for the parameters to be determined:

$$\begin{pmatrix}
R(\|\boldsymbol{p_1}-\boldsymbol{p}_1\|) & \cdots & R(\|\boldsymbol{p_1}-\boldsymbol{p}_n\|) & 0 & \cdots & 0 & \boldsymbol{p}_1[0] & \boldsymbol{p}_1[1] & 0 & 0 & 1 & 0 \\
R(\|\boldsymbol{p_2}-\boldsymbol{p}_1\|) & \cdots & R(\|\boldsymbol{p_2}-\boldsymbol{p}_n\|) & 0 & \cdots & 0 & \boldsymbol{p}_2[0] & \boldsymbol{p}_2[1] & 0 & 0 & 1 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
R(\|\boldsymbol{p_n}-\boldsymbol{p}_1\|) & \cdots & R(\|\boldsymbol{p_n}-\boldsymbol{p}_n\|) & 0 & \cdots & 0 & \boldsymbol{p}_n[0] & \boldsymbol{p}_n[1] & 0 & 0 & 1 & 0 \\
0 & \cdots & 0 & R(\|\boldsymbol{p_1}-\boldsymbol{p}_1\|) & \cdots & R(\|\boldsymbol{p_1}-\boldsymbol{p}_n\|) & 0 & 0 & \boldsymbol{p}_1[0] & \boldsymbol{p}_1[1] & 0 & 1 \\
0 & \cdots & 0 & R(\|\boldsymbol{p_2}-\boldsymbol{p}_1\|) & \cdots & R(\|\boldsymbol{p_2}-\boldsymbol{p}_n\|) & 0 & 0 & \boldsymbol{p}_2[0] & \boldsymbol{p}_2[1] & 0 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & \cdots & 0 & R(\|\boldsymbol{p_n}-\boldsymbol{p}_1\|) & \cdots & R(\|\boldsymbol{p_n}-\boldsymbol{p}_n\|) & 0 & 0 & \boldsymbol{p}_n[0] & \boldsymbol{p}_n[1] & 0 & 1 \\
\boldsymbol{p}_1[0] & \cdots & \boldsymbol{p}_n[0] & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 \\
\boldsymbol{p}_1[1] & \cdots & \boldsymbol{p}_n[1] & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 \\
1 & \cdots & 1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 \\
0 & \cdots & 0 & \boldsymbol{p}_1[0] & \cdots & \boldsymbol{p}_n[0] & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & 0 & \boldsymbol{p}_1[1] & \cdots & \boldsymbol{p}_n[1] & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & 0 & 1 & \cdots & 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0
\end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{\alpha}_1[0] \\ \vdots \\ \boldsymbol{\alpha}_n[0] \\ \boldsymbol{\alpha}_1[1] \\ \vdots \\ \boldsymbol{\alpha}_n[1] \\ a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \\ b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} \boldsymbol{q}_1[0] \\ \vdots \\ \boldsymbol{q}_n[0] \\ \boldsymbol{q}_1[1] \\ \vdots \\ \boldsymbol{q}_n[1] \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{2.11}$$

In equation (2.11),

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix},$$

$\boldsymbol{p}_i[0]$ represents the x-coordinate of point $\boldsymbol{p}_i$, $\boldsymbol{p}_i[1]$ represents the y-coordinate of point $\boldsymbol{p}_i$, as well as $\boldsymbol{q}_i[0]$, $\boldsymbol{q}_i[1]$, $\boldsymbol{\alpha}_i[0]$, $\boldsymbol{\alpha}_i[1]$.

We solve the expression (2.11) using the Eigen library and return the solution vector, which contains the coefficients to be determined.

2.3 MLS

In this project, I just finished the rigid transformation, but complete processes mentioned in the paper are listed.

The known deformation points are denoted as $\boldsymbol{p}$, and their images are denoted as $\boldsymbol{q}$, stored in $n \times 2$ matrices $\boldsymbol{p}$ and $\boldsymbol{q}$ respectively, where n represents the number of fixed points. Based on the given $\boldsymbol{p}$ and $\boldsymbol{q}$. We define:

$$l_v(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{M} + \boldsymbol{T}. \tag{2.12}$$

The transformation needs to minimize the following energy function:

$$E = \sum_i \omega_i |l_v(\boldsymbol{p}_i) - \boldsymbol{q}_i|^2. \tag{2.13}$$

attains its minimum value, where ω_i represents the weight vector, a $1 \times n$ vector defined as:

$$\omega_i = \frac{1}{|\boldsymbol{p}_i - \boldsymbol{v}|^{2\alpha}}. \tag{2.14}$$

From here, it can be seen that this weight vector depends on $\boldsymbol{v}$ and $\boldsymbol{p}$.

Substituting the assumption equation (2.12) into the energy function (2.13), we can obtain:

$$E = \sum_i w_i \left|\boldsymbol{p}_i M + \boldsymbol{T} - \boldsymbol{q}_i\right|^2. \tag{2.15}$$

Taking the derivative of Equation (2.15) with respect to $\boldsymbol{T}$, and since the energy function attains an extremum, we have:

$$\frac{\partial E}{\partial \boldsymbol{T}} = 2\sum_i \omega_i \left|\boldsymbol{p}_i \boldsymbol{M} + \boldsymbol{T} - \boldsymbol{q}_i\right| = 0. \tag{2.16}$$

Then

$$\boldsymbol{T} = \boldsymbol{q}^* - \boldsymbol{p}^* \boldsymbol{M}. \tag{2.17}$$

We define $\boldsymbol{q}^*$ and $\boldsymbol{p}^*$:

$$\begin{aligned} \boldsymbol{p}_* &= \frac{\sum_i w_i \boldsymbol{p}_i}{\sum_i w_i}. \\ \boldsymbol{q}_* &= \frac{\sum_i w_i \boldsymbol{q}_i}{\sum_i w_i}. \end{aligned} \tag{2.18}$$

We can find that $\boldsymbol{p}^*$ and $\boldsymbol{q}^*$ are only up to ω, $\boldsymbol{p}$ and $\boldsymbol{q}$. Then

$$l_v(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{p}^*)\boldsymbol{M} + \boldsymbol{q}^*. \tag{2.19}$$

At this point, we can rewrite the energy function as:

$$E = \sum_i w_i \left|\hat{\boldsymbol{p}}_i \boldsymbol{M} - \hat{\boldsymbol{q}}_i\right|^2. \tag{2.20}$$

$\hat{\boldsymbol{p}}_i$ and $\hat{\boldsymbol{q}}_i$ are defined:

$$\begin{aligned} \hat{\boldsymbol{p}}_i &= \boldsymbol{p}_i - \boldsymbol{p}^*. \\ \hat{\boldsymbol{q}}_i &= \boldsymbol{q}_i - \boldsymbol{q}^*. \end{aligned} \tag{2.21}$$

Therefore, our core task is transformed into solving the matrix $\boldsymbol{M}$, and the computation of different transformation matrices $\boldsymbol{M}$ varies slightly. Later, for the function $l_v(\boldsymbol{x})$. let's define:

$$\boldsymbol{f}(\boldsymbol{v}) = l_v(\boldsymbol{v}). \tag{2.22}$$

2.3.1 Affine Transformation

$$E = \sum_i w_i \left(\hat{\boldsymbol{p}}_i \boldsymbol{M} - \hat{\boldsymbol{q}}_i\right)\left(\hat{\boldsymbol{p}}_i \boldsymbol{M} - \hat{\boldsymbol{q}}_i\right)^\top = \sum_i w_i \left(\hat{\boldsymbol{p}}_i \boldsymbol{M}\boldsymbol{M}^\top \hat{\boldsymbol{p}}_i{}^\top - \hat{\boldsymbol{q}}_i \boldsymbol{M}^\top \hat{\boldsymbol{p}}_i{}^\top - \hat{\boldsymbol{p}}_i \boldsymbol{M} \hat{\boldsymbol{q}}_i{}^\top + \hat{\boldsymbol{q}}\hat{\boldsymbol{q}}_i{}^\top\right). \tag{2.23}$$

Taking the derivative with respect to $\boldsymbol{M}$ on both sides, we have:

$$\frac{\partial E}{\partial \boldsymbol{M}} = 2\sum_i \omega_i (\hat{\boldsymbol{p}}_i{}^\top \hat{\boldsymbol{p}}_i \boldsymbol{M} - \hat{\boldsymbol{p}}_i{}^\top \hat{\boldsymbol{q}}_i) = 0. \tag{2.24}$$

Then

$$\boldsymbol{M} = (\sum_i \hat{\boldsymbol{p}}_i{}^\top \omega_i \hat{\boldsymbol{p}}_i)^{-1} \cdot (\sum_j \omega_j \hat{\boldsymbol{p}}_j{}^\top \hat{\boldsymbol{q}}_j). \tag{2.25}$$

$$\boldsymbol{f}_a(\boldsymbol{v}) = (\boldsymbol{v} - \boldsymbol{p}^*)(\sum_i \hat{\boldsymbol{p}}_i{}^\top \omega_i \hat{\boldsymbol{p}}_i)^{-1} \cdot (\sum_j \omega_j \hat{\boldsymbol{p}}_j{}^\top \hat{\boldsymbol{q}}_j) + \boldsymbol{q}^*. \tag{2.26}$$

Note that if interactivity is required, i.e., users can achieve real-time animation effects by changing the position of $\boldsymbol{q}$, for ease of computation, we can rearrange (2.26) as follows:

$$\boldsymbol{f}_a(\boldsymbol{v}) = \sum_j \boldsymbol{A}_j \hat{\boldsymbol{q}}_j + \boldsymbol{q}^*. \tag{2.27}$$

$\boldsymbol{A}_j$ is defined:

$$\boldsymbol{A}_j = (\boldsymbol{v} - \boldsymbol{p}^*)(\sum_i \hat{\boldsymbol{p}}_i{}^\top \omega_i \hat{\boldsymbol{p}}_i)^{-1} \cdot (\omega_j \hat{\boldsymbol{p}}_j{}^\top). \tag{2.28}$$

Note that the definition of $\boldsymbol{A}_j$ is independent of the value of $\boldsymbol{q}$, so it only needs to be computed once.

2.3.2 Similar Transformation

However, affine transformations may result in uneven distribution or discontinuities, which are not typically observed in real-life objects. On the other hand, similarity transformations only involve translation, rotation, and uniform scaling. Therefore, we examine the mapping function under similarity transformations, where the core remains to find the matrix $\boldsymbol{A}$.

Similarity transformations require the matrix to possess the property ($\boldsymbol{M}^\top \boldsymbol{M} = \lambda^2 \boldsymbol{I}$). If we denote:

$$\boldsymbol{M} = (\boldsymbol{M}_1 \quad \boldsymbol{M}_2).$$

Then, $\boldsymbol{M}_1^\top \boldsymbol{M}_1 = \boldsymbol{M}_2^\top \boldsymbol{M}_2 = \lambda^2$ and $\boldsymbol{M}_1^\top \boldsymbol{M}_2 = 0$. They mean that

$$\boldsymbol{M}_2 = \boldsymbol{M}_1^\perp. \tag{2.29}$$

We can change the form of energy function:

$$E = \sum_i \omega_i \left| \begin{pmatrix} \hat{\boldsymbol{p}}_i \\ -\hat{\boldsymbol{p}}_i{}^\perp \end{pmatrix} \boldsymbol{M}_1 - \hat{\boldsymbol{q}}_i{}^\top \right|^2. \tag{2.30}$$

$$\frac{\partial E}{\partial \boldsymbol{M}_1} = 2 \sum_i \omega_i \begin{pmatrix} \hat{\boldsymbol{p}}_i \\ -\hat{\boldsymbol{p}}_i{}^\perp \end{pmatrix} (\begin{pmatrix} \hat{\boldsymbol{p}}_i{}^\top & -(\hat{\boldsymbol{p}}_i{}^\perp)^\top \end{pmatrix} \boldsymbol{M}_1 - \hat{\boldsymbol{q}}_i{}^\top) = 0. \tag{2.31}$$

We can get the value of $\boldsymbol{M}_1$:

$$\boldsymbol{M}_1 = \frac{1}{\mu_s} \sum_i \omega_i \begin{pmatrix} \hat{\boldsymbol{p}}_i \\ -\hat{\boldsymbol{p}}_i{}^\perp \end{pmatrix} \hat{\boldsymbol{q}}_i{}^\top. \tag{2.32}$$

$$\mu_s = \sum_i \omega_i \hat{\boldsymbol{p}}_i \hat{\boldsymbol{p}}_i{}^\top. \tag{2.33}$$

$$\boldsymbol{M} = \frac{1}{\mu_s} \sum_i \omega_i \begin{pmatrix} \hat{\boldsymbol{p}}_i \\ -\hat{\boldsymbol{p}}_i{}^\perp \end{pmatrix} \begin{pmatrix} \hat{\boldsymbol{q}}_i{}^\top & -(\hat{\boldsymbol{q}}_i{}^\perp)^\top \end{pmatrix} \tag{2.34}$$

Similarly, to enhance user interactivity, we can improve the expression as follows:

$$\boldsymbol{f}_s(v) = \sum_i \hat{\boldsymbol{q}}_i (\frac{1}{\mu_s} A_i) + \boldsymbol{q}^*. \tag{2.35}$$

$$\boldsymbol{A}_{\boldsymbol{i}} = \omega_i \begin{pmatrix} \hat{\boldsymbol{p}}_i \\ -\hat{\boldsymbol{p}}_i{}^\perp \end{pmatrix} \begin{pmatrix} \boldsymbol{v} - \boldsymbol{p}^* \\ -(\boldsymbol{v} - \boldsymbol{p}^*)^\perp \end{pmatrix}^\top. \tag{2.36}$$

We can find that A_i is only up to $\boldsymbol{p}$ and $\boldsymbol{v}$.

Similarity transformations have a good property of preserving angles, thus resulting in more ideal outcomes compared to affine transformations. However, they may lead to the enlargement of the range for parts of points that are distant from the transformation points.

2.3.3 Rigid Transformation

In previous work, it was found that achieving more realistic deformations requires approaching rigid transformations as closely as possible. The author proposed Lemma 2.1 and provided the following concise expression:

$$\boldsymbol{f}_r(\boldsymbol{v}) = \sum_i \hat{\boldsymbol{q}}_i \boldsymbol{A}_i. \tag{2.37}$$

$$\boldsymbol{f}_r(\boldsymbol{v}) = |\boldsymbol{v} - \boldsymbol{p}^*| \frac{\boldsymbol{f}_r(\boldsymbol{v})}{|\boldsymbol{f}_r(\boldsymbol{v})|} + \boldsymbol{q}^*. \tag{2.38}$$

$\boldsymbol{f}_r(\boldsymbol{v})$ is the sought-after rigid transformation function. Compared to similarity transformations, rigid transformations utilize normalization, thus leading to increased computational complexity.

3 Programming

3.1 Class

One of the primary objectives of this task is to recall knowledge related to C++ classes, including inheritance and polymorphism. The class diagram for the code to be written is shown in Figure 1.

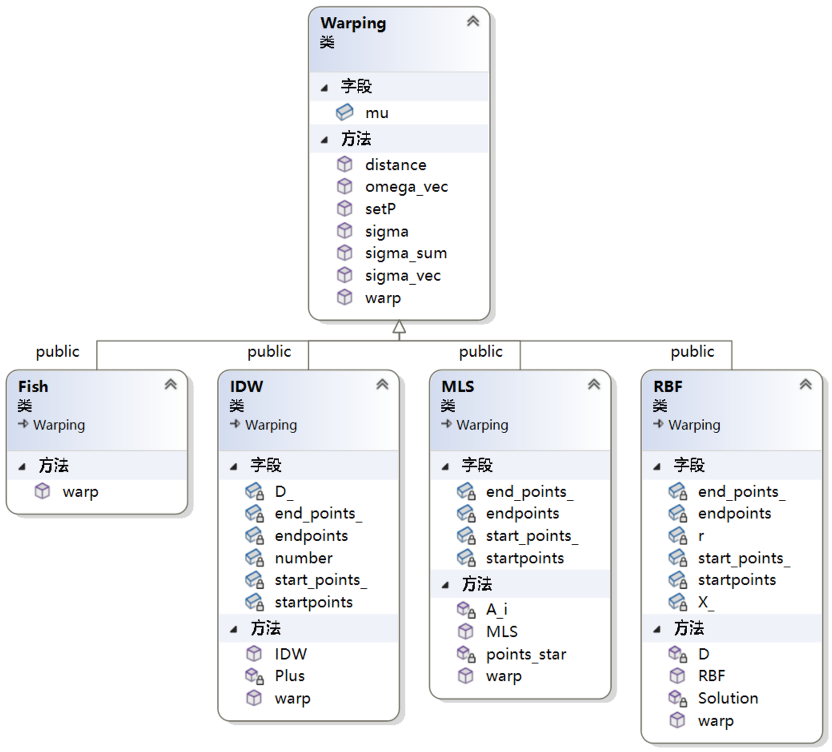


Figure 1: Class diagram

We have established four subclasses of Warping, each representing a type of deformation (Fish, IDW, MLS, RBF). The Warping class contains some basic functions: SetP, which transforms ImVec2 vectors to pair type vectors; distance, which represents the distance function; omega_vec, sigma_sum, sigma_vec, and sigma, all related to weight calculation. The warp function is a virtual function, with its specific implementation in the subclasses.

We have created a shared pointer std::shared_ptr<Warping> warp_, which can point to different subclasses of Warping. This demonstrates inheritance and polymorphism, as when the warp function is called, the specific implementation in the subclass is invoked based on the type of object pointed to by warp_.

To pass the information of the initial and final points into the warp function, we use a constructor. Additionally, during construction, elements unrelated to the points to be mapped, such as the solution vector in RBF, are already computed.

3.2 Improve Code Efficiency

When I initially ran the code, I found the speed to be very slow. Upon inspecting the code, I discovered that due to computing some quantities unrelated to the points in the constructor, there were redundant computations of matrices unrelated to the points each time a point was mapped. This inefficiency arises because the computational complexity of solving matrix equations using the Eigen library is on the order of n^3. As for a 512×512 pixel image, the increased computational workload is extremely large. Therefore, we precompute some values unrelated to the points in the constructor, significantly improving the efficiency of the code.

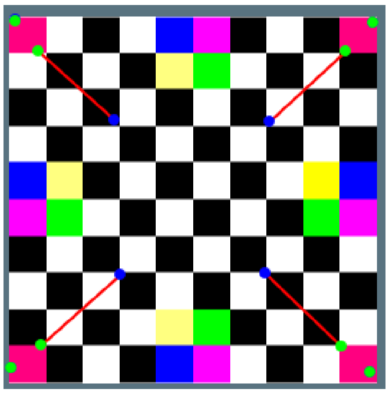
It is important to note that in the MLS algorithm, each function is related to the points to be mapped, so calculations can only be performed in the warp function. However, it is still possible to optimize the efficiency of the code by precomputing functions such as weight vectors.

4 Experimental Results

The method of operation has not changed. Simply click on "Warping" in the interface, and you will see four options: "fisheye", "IDW", "RBF", and "MLS". The "fisheye" option does not require selecting points. However, for the other methods, you need to first select points and then perform the transformation. It's important to note that selecting only one transformation point will result in strong distortion in the image!

We tested the IDW and RBF methods using the provided grid image. The results of the original image and IDW algorithm or RBF algorithm tests are shown in Figures 2, 3 respectively.

We use the ginger man to test MLS. The result is shown in Figure 4.


(a) original image 1

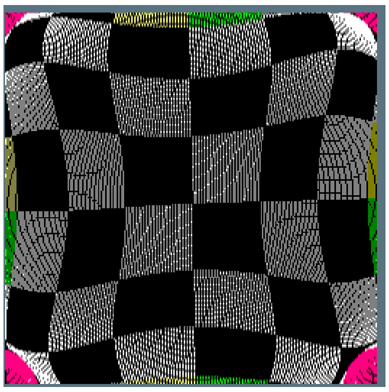
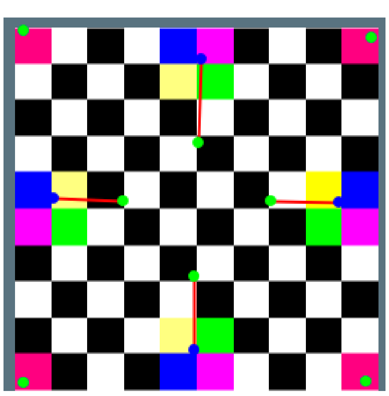

(b) IDW

Figure 2: IDW example


(a) original image 2

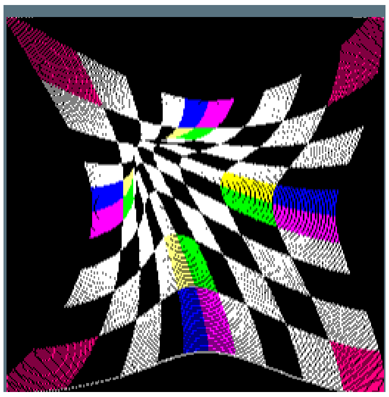
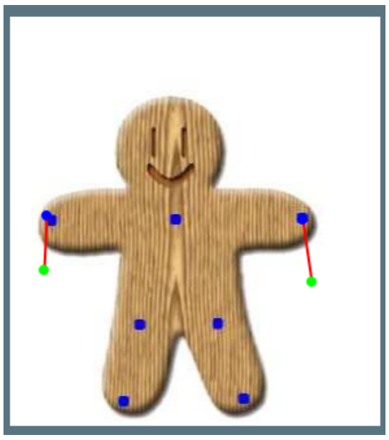

(b) RBF

Figure 3: RBF example


(a) original image 3

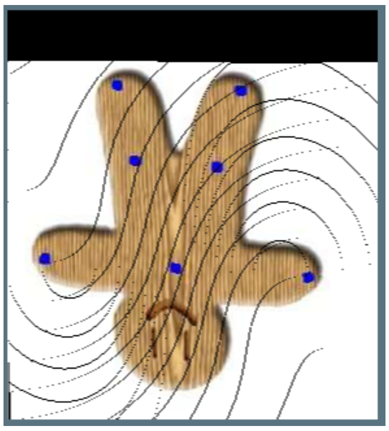

(b) MLS

Figure 4: MLS example

5 Summary

In this experiment, we achieved interactive image deformation, gaining a deeper understanding of the essence of image deformation and revisiting concepts such as inheritance, polymorphism, and constructors in C++ classes. However, there are several issues with the algorithms we implemented:

1. The deformation results exhibit numerous ”black spots”: In the image rendering algorithm, we resorted to forcefully rounding floating-point values at the end. This resulted in the algorithm being neither smooth nor necessarily complete, causing multiple pixel overlaps and the occurrence of ”black spots”. This issue can be addressed using the ANN library, an optional component. The library facilitates searching nearby points, enabling further interpolation to achieve smoother and more accurate results.

2. Algorithm robustness: In our experiments, we observed that for certain specific point selections, significant distortions might occur. However, the algorithm ensures the mapping of the source starting points to the target ending points, indicating that the robustness of the algorithm requires further exploration.