Report on Image warping

Zhan Zhang, PB17000123

August 16, 2019

Abstract

This document summarized the algorithms of our image warping solution for further study, and there is a detailed description about the implementation of these algorithms.

1 Algorithms

The term "Image Warping" describes the methods for deforming images to arbitrary shapes. By using the control points, the prolem of image warping is essentially the problem of scattered data interpolation.

Input: n pairs (p_i, q_i) of control points, $p_i, q_i \in \mathbb{R}^2, i = 1, ..., n$.

Output: An at-least-continuous function $f : \mathbb{R}^2 \to \mathbb{R}^2$ with $f(p_i) = q_i, i = 1, ..., n$.

1.1 Inverse distance weighted interpolation

Inverse distance-weighted interpolation methods were originally proposed by Shepard and improved by a number of other authors, notably Franke and Nielson.

For each control point p_i, we use local approximation $f_i(p) : \mathbb{R}^2 \to \mathbb{R}^2 with f_i(p_i) = q_i, i = 1, ..., n$ and set the interpolation function a weighted average of these local approximations:

$$f(p) = \sum_{i=1}^{n} w_i(p) f_i(p)$$

where $w_i(p)$ is the weight function, which must satisfy the condition:

$$w_i(p_i) = 1, \sum_{i=1}^{n} w_i(p) = 1, \text{and } w_i(p) \geq 0, i = 1, ..., n$$

These conditions guarantee the property of interpolation. Shepard proposed the following simple weight function:

$$w_i(p) = \frac{\sigma_i(p)}{\sum_{j=1}^{n} \sigma_j(p)}, \text{ with } \sigma_i(p) = \frac{1}{d(p, p_i)^\mu}$$

where $d(p, p_i)$is the distance between p and p_i.

Usually, we set $\mu = 2$. In order to figure out our local approximation $f_i(p)$. We simpliy use the linear local approximations, and corresponding error function $E_i(f)$ as below:

$$f_i(p) = q_i + T_i(p - p_i), \text{ with } T_i = \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix}$$

$$E_i(f) = \sum_{j=1, j\neq i}^{n} \sigma_i(p_j) \cdot \| q_i + \left(\begin{smallmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{smallmatrix}\right)(p_j - p_i) - q_j \|^2$$

Using the partial derivatives with respect to the $t_{kl\ k,l=1,2}$, to obtain the minimum of the error function and setting the derivatives to zero, we can get four linear equations with four unknows, easily solved for the t_{kl}.

1.2 Radial basis function transform image warping

Transformations based on radial basis functions have proven to be a powerful tool in image warping. It has the expression below:

$$f(p) = A(p) + R(p)$$

where $A(p) = Mp + b$ is a D affine transformation (M is a 2 x 2 real matrix), and $R(p)$ is a radial transformation defined by $R(p) = (R_x(p), R_y(p))$

$R_x(p), R_y(p)$ are both radial functions of the form:

$$R(p) = \sum_{i=1}^{n} a_i g_i(\|p - p_i\|)$$

$g : \mathbb{R}^+ \to \mathbb{R}$ is a univariate function, termed the radial basis function.

So, $f(p)$ is determined by $2(n+3)$ coefficients: 6 for the affine part, and $2n$ for the radial components. We define an imagewarping transformation based on the mapping of n anchor points:

$$f(p_i) = q_i, \text{ for } i = 1, 2, ..., n$$

These interpolation conditions translate into $2n$ linear equations in the coefficients of the $f(p)$, thus leaving 6 degrees of freedom (3 for each dimension).

By adding the additional equations, one can mark two sets of anchor points. One set (the affine set, red line in project) determines the affine part of the mapping, while the other set (the radial set, green line in project) controls the generation of changes in the facial expression. Note that these sets may intersect, and may even be identical.

Controlling the affine component, $A(p)$, is carried out according to the number of anchor points in the affine set, as follows. If no points are specified, the affine component is the identity mapping; if one point is specified - translation; two points - translation and scaling; three points - general affine transformation; more than three - general affine transformation determined by a least-square approximation procedure. Then, $R(p_i) = q_i - A(p_i), i = 1, ..., n$ can be solved by choicing the radial basis functions like

$$g(d) = (d^2 + r^2)^{\pm \frac{1}{2}}$$

Following a suggestion by Eck, we used individual values r_i for each point p_i, conputed from the distance to the nearest neighbor: $r_i = min_{i \neq j} d(p_j, p_i)$.

2 Implementation

Press "ControlPoints" to set (p_i, q_i). In my project, blue points stands for the orignal points, p_i; and lightgreen points stands for the transformed points, q_i. By pressing the left button on the mouse, we can set the control points in the affine set, which showed in the form of red lines; and by pressing the right button on the mouse, we can set

the control points in the radial set, which showed in the form of green lines. However as for the IDW_ImageWarping, they are all the same.

Press "IDW" or "RBF" to apply different image warping methods. Or, one can press "CP" to compare the difference between this two methods with the same control points.

2.1 IDW_ImageWarping

As a global method , it has a time complexity of $O(nN)$, which makes my project rather slow.

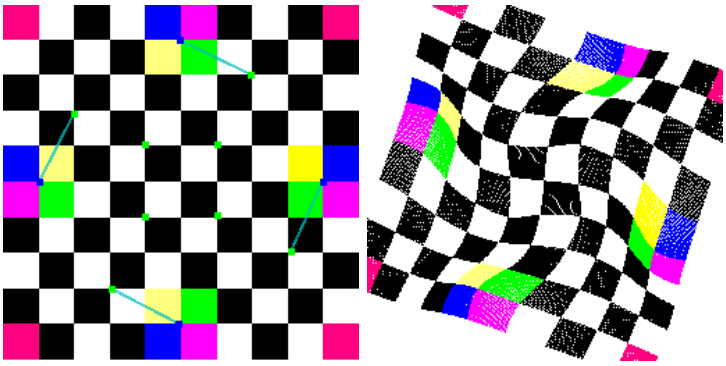

Figure 1: IDW_ImageWarping

2.2 RBF_ImageWarping

Like all global interpolation methods, all control points have to be taken into account in radial basis functions. Thus, the algorithmic complexity is $O(nN)$.

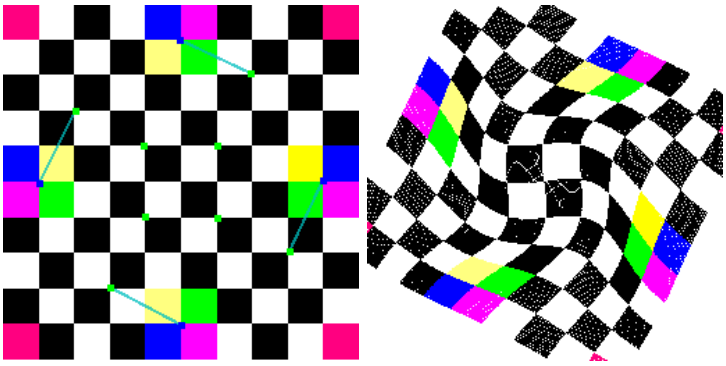

Figure 2: RBF_ImageWarping

Here comes the differences between RBF results with points in the affine set and RBF results without points in the affine set:

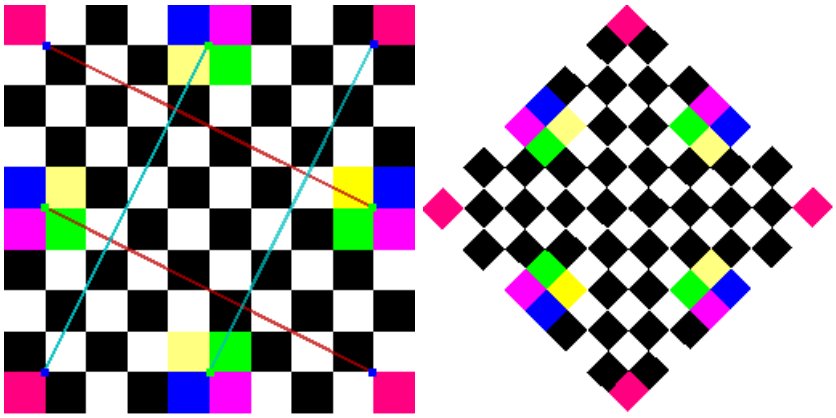

Figure 3: RBF with points in the affine set

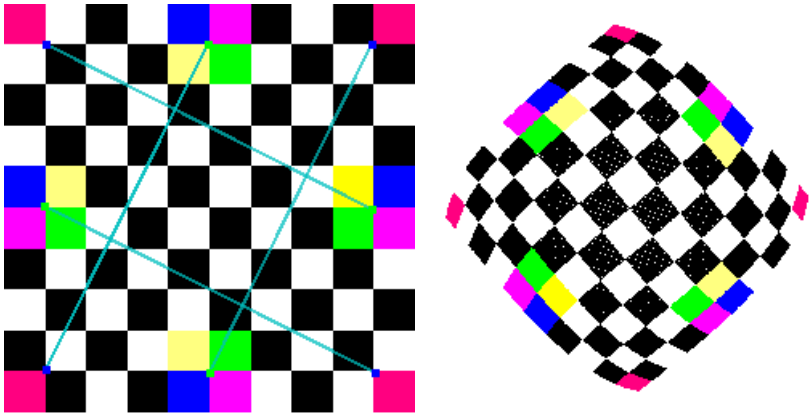

Figure 4: RBF without points in the affine set

2.3 Compare

From the figure below, we can simpliy feel that IDW is a little bit good at the fold-over control. On the other hand, the image warped by RBF is semed more flexible and ductile.

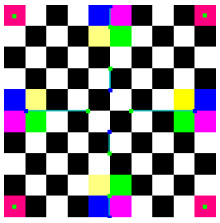

Figure 5: the same control points

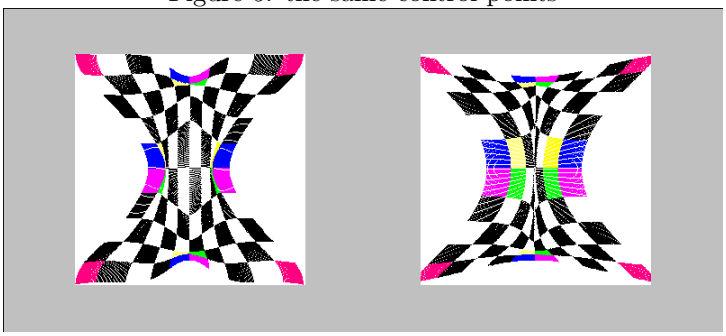

Figure 6: IDW(left) vs RBF(right)

Image warping, an additional problem to that of interpolation is that of maintaining the one-to-one property of the warping transformation. Each point in the transformed image should correspond to only one point in the original image and vice-versa. If the one-to one property is not satisfied the warped function will have the appearance of being folded, rather than only stretched. However, image fold-over still appears in each methods:

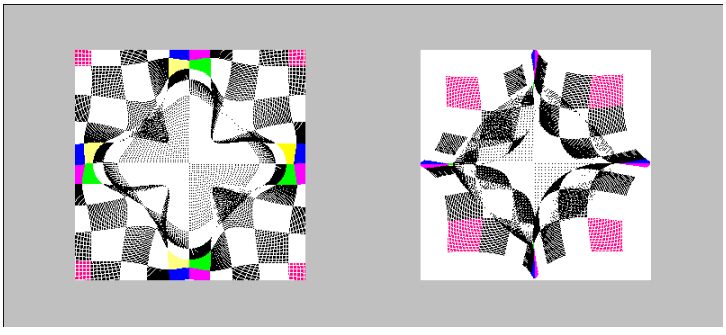

Figure 7: IDW(left) and RBF(right) both fold over

It's necessary to point out that many white lines appeared in the image, which were caused by the prperty of function $f(p)$.

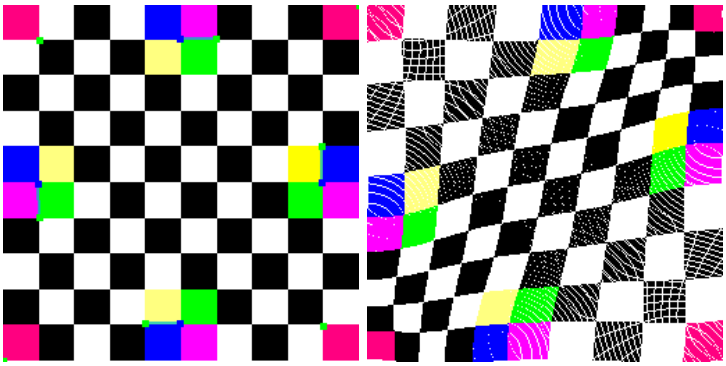

Figure 8: white lines in IDW

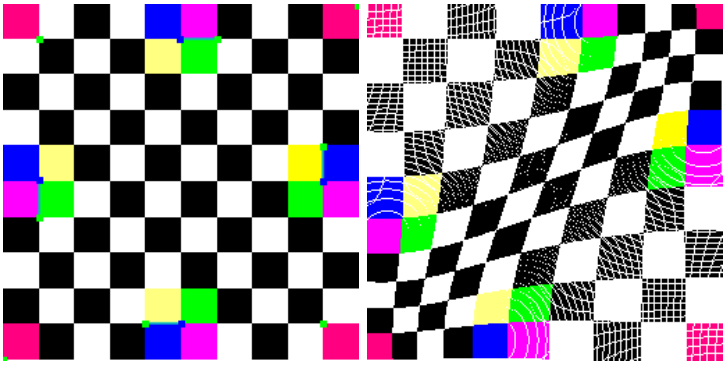

Figure 9: white lines in RBF

Since f is at least continuous fuction and for each scatter point (i, j), we can find $(i', j') = f(i, j)$, if the Jacobian $J_{ij} > 1$, then the image will be magnified at that local place. So, there must be some dots (i, j) cannot map to, then these dots will appear white. And f is continuous, so the sequence of dots is continuous too in some aspects, which means they seemed like a white line.

3 Improvement

3.1 FillHole

In order to fix the white line problem of my warping class, I figure out a simple way to fill the white hole. By using the pixel near the target point, the color of the white hole can be easily approximated.

3.1.1 Linear Search

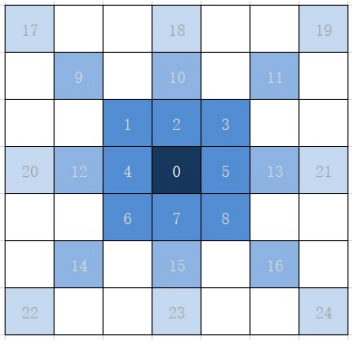

Figure 10: Importance level of near by points with the boundary of 3

The picture above is the near points I usually take into consideration. The number reffers to the order of every pixel when we considering. The target point's number is 0, it will be considered first, and if it's not the hole point we'll keep its color. If it's white hole point, we will use the first not hole pixel's color to approximate.

These are some comperasions:

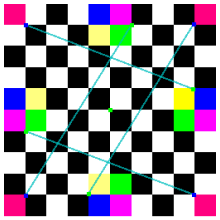

Figure 11: the control points of RBF

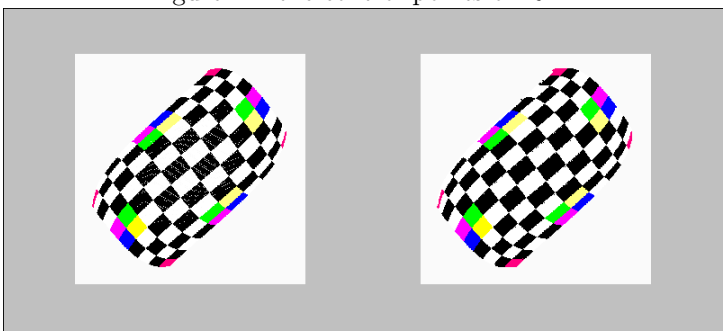

Figure 12: original(left) vs hole-filled(right)

We can find that the white lines disappered. But because of the drawback of the method, the picture seems some kind of rough and vague, especially at the brim of every rectangle.

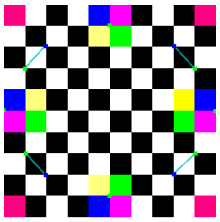

Figure 13: the control points of IDW

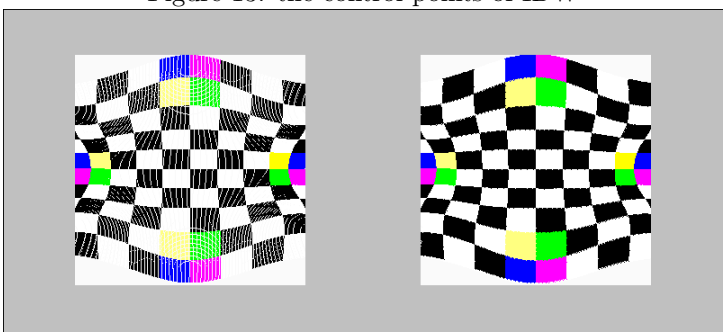

Figure 14: original(left) vs hole-filled(right)

Since the warping of image can be extramely distorted, I estimate the searching boundary by using the information contained in controlpoints:

$$bound = \max \frac{\|q_i - q_j\|}{\|p_i - p_j\|} + 1$$

Thus, fillhole method works in whichever cases. Here is the extrame case:

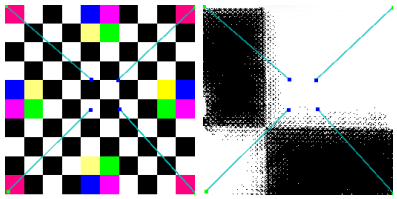

Figure 15: extrame case

It seems like every pixels are in the shape of ' * ' same as the general shape of searching points.

3.1.2 k-Nearest Neighbour

Anothor way I figured out is using k-nearest neighbour. By build up a kd-tree, we can get k-nearest points of the target white point. Then we use the outcome color in the

nearest points set to approximate the white hole's color.

These are some comperasions:

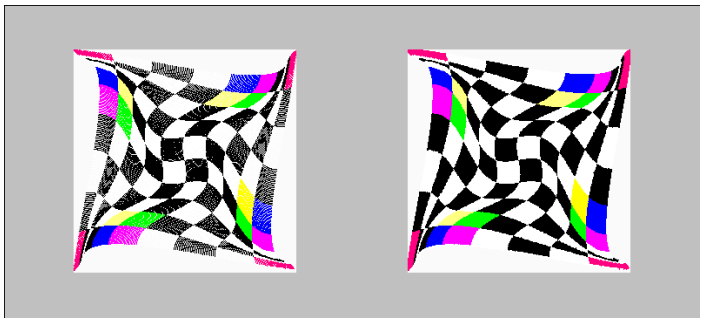

Figure 16: RBF: original(left) vs hole-filled(right)

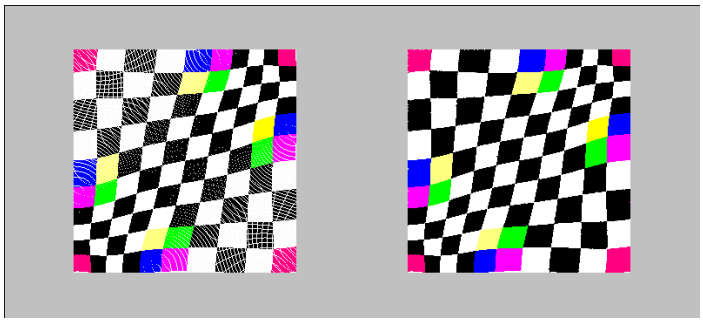

Figure 17: IDW: original(left) vs hole-filled(right)

Here is the extrame case:

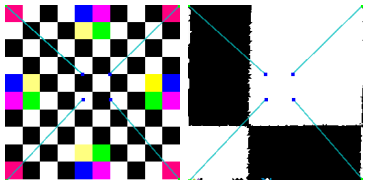

Figure 18: extrame case

It is better than the liner-search methods we first proposed. And the color filled is more reliable and accurate.