Exercise 3. MLS For Curves, Meshes and Images

Dongho Kang

16-948-598

April 2, 2017

MATLAB R2016b version was used for coding and testing:

MathWorks, MATLAB R2016b (9.1.0.441655)
64-bit (maci64)

The ***code*** directory contains the followings:

part1_2.m script .m file for exercise part 1.2.

part1_3.m script .m file for exercise part 1.3.

part1_4.m script .m file for exercise part 1.4. (optional problem)

part1_5.m script .m file for exercise part 1.5. (optional problem)

part2.m script .m file for exercise part 2.

functions dir function .m files

- ***Fx2D.m*** function .m file for calculating $f(\mathbf{x}) = c_0$ for each 2D $\mathbf{x}$ points on the grid.
- ***CalC0.m*** function .m file for $f(\mathbf{x}) = c_0$ for given $\mathbf{x}$ array (subfunction).
- ***VisCompSigma.m*** function .m file for visualization of curve plot (part 1.2) for given value of sigma.
- ***FxGradFx3D.m*** function .m file for calculating $f(\mathbf{x})\nabla f(\mathbf{x})$ for given 3D $\mathbf{x}$ point array.
- ***FxGradFx3D_ANN.m*** function .m file for speeding up calculating $f(\mathbf{x})\nabla f(\mathbf{x})$ for given 3D $\mathbf{x}$ point array.

FxGradFx3D_RIMLS.m function .m file for calculating $f(\mathbf{x})\nabla f(\mathbf{x})$ for given 3D $\mathbf{x}$ point array using advanced method.

AffineTransform.m function .m file for affine transformation.

SimilarTransform.m function .m file for similarity transformation.

RigidTransform.m function .m file for rigid transformation.

FvToImg.m function .m file for creating forward warping image.

BackwardWarp.m function .m file for creating backward warping image.

data dir Pre-generated data and images for testing.

p_array.mat saved control points (part 2).

q_array.mat saved deformed points (part 2).

F_array.mat calculated $f(\mathbf{x})$ values (part 1.2). The file was used for debugging.

files dir provided funtions and files. Several images and .off files were additionally added for testing.

For running each .m script, check dependencies (especially for ***part1_4.m*** and ***part1_5.m***) and adjust parameters first. Note that **these scripts only work properly in MATLAB R2016b environment.** More details are stated in the *Running* section of each parts.

1 EXERCISE PART 1: CURVE AND SURFACE RECONSTRUCTION USING MLS

In this exercise, moving least squares (MLS) were used for reconstruction of curves (part 1.1 and part 1.2) and smoothing meshes (part 1.3, part 1.4, part 1.5):

- derivation of the mathematical formula of the MLS based surfaces (part 1.1).
- implementing the derived $f(x)$ and plotting the curves for the given three data sets (part 1.2).
- derivation of $\nabla f(\mathbf{x})$ and smoothing provided meshes by $\mathbf{v}' = \mathbf{v} - f(\mathbf{x})\nabla f(\mathbf{x})$ (part 1.3).
- speeding up mesh smoothing by using Approximate Nearest Neighbor Library (part 1.4).
- implementing the sharp feature preserving mesh smoothing proposed on the paper "Feature preserving point set surfaces based on non-linear kernel regression".

1.1 CURVE DERIVATION

In the moving least square problem, given $f(\mathbf{x}) = \mathbf{b}(\mathbf{x})\mathbf{c}$, the local weight function ϕ_i, and the local functions $f_i(\mathbf{x})$, the local error function is defined as follows:

$$\begin{aligned} E_{\mathbf{x}}(\mathbf{c}) &= \sum_i \phi_i(\mathbf{x})\big(f(\mathbf{x_i}) - f_i\big)^2 \\ &= \sum_i \phi_i(\mathbf{x})\big(\mathbf{b}(\mathbf{x_i})^T\mathbf{c} - f_i\big)^2 \end{aligned} \tag{1.1}$$

In this problem, define $\mathbf{c}$ and $\mathbf{b}$ as $\mathbf{c} = c_0$ and $\mathbf{b} = 1$ as the exercise instruction suggested. Concerning the local weight function $\phi_i(\mathbf{x})$ and the local functions $f_i(\mathbf{x})$, gaussian weight function $\phi_i(\mathbf{x})$ and reproduced local functions are used:

$$\phi_i(\mathbf{x}) = \exp\Big(-\frac{\big\|\mathbf{x}-\mathbf{x_i}\big\|^2}{\sigma^2}\Big) \tag{1.2}$$

$$f_i(\mathbf{x}) = \mathbf{n_i}^T(\mathbf{x}-\mathbf{x_i}) : \tag{1.3}$$

Thus, the moving least square problem can be described as follows:

$$\arg\min_{c_0} E_{\mathbf{x}}(c_0) = \arg\min_{c_0} \sum_i \exp\Big(-\frac{\big\|\mathbf{x}-\mathbf{x_i}\big\|^2}{\sigma^2}\Big)\big(c_0 - \mathbf{n_i}^T(\mathbf{x}-\mathbf{x_i})\big)^2 \tag{1.4}$$

The closed form solution can be obtained by differentiation:

$$\frac{dE_{\mathbf{x}}(c_0)}{dc_0} = \sum_i 2\exp\Big(-\frac{\big\|\mathbf{x}-\mathbf{x_i}\big\|^2}{\sigma^2}\Big)\big(c_0 - \mathbf{n_i}^T(\mathbf{x}-\mathbf{x_i})\big) = 0 \tag{1.5}$$

$$\sum_i \exp\Big(-\frac{\big\|\mathbf{x}-\mathbf{x_i}\big\|^2}{\sigma^2}\Big)c_0 = \sum_i \exp\Big(-\frac{\big\|\mathbf{x}-\mathbf{x_i}\big\|^2}{\sigma^2}\Big)(\mathbf{n_i}^T(\mathbf{x}-\mathbf{x_i})) \tag{1.6}$$

$$f(\mathbf{x}) = c_0 = \frac{\sum_i \exp\big(-\frac{\|\mathbf{x}-\mathbf{x_i}\|^2}{\sigma^2}\big)\big(\mathbf{n_i}^T(\mathbf{x}-\mathbf{x_i})\big)}{\sum_i \exp\big(-\frac{\|\mathbf{x}-\mathbf{x_i}\|^2}{\sigma^2}\big)} \tag{1.7}$$

1.2 CURVES PLOTTING

1.2.1 DESCRIPTION

As $f(\mathbf{x})$ was derived in part 1.1, for given 2D sample points and its normal vectors, implicit MLS surface (curve for $\mathbb{R}^2$ domain)can be obtained:

$$S = \{\mathbf{x} \mid f(\mathbf{x}) = 0, \nabla f(\mathbf{x}) \neq 0\} \tag{1.8}$$

For different data sets, and different σ, $f(\mathbf{x})$ was calculated for each $\mathbf{x}$ which is generated by the MATLAB function **meshgrid** and it was visualized as a color map. The implicit MLS surfaces are described as set of the points which of the value of $f(\mathbf{x})$ is 0. These points represent fitted curve.

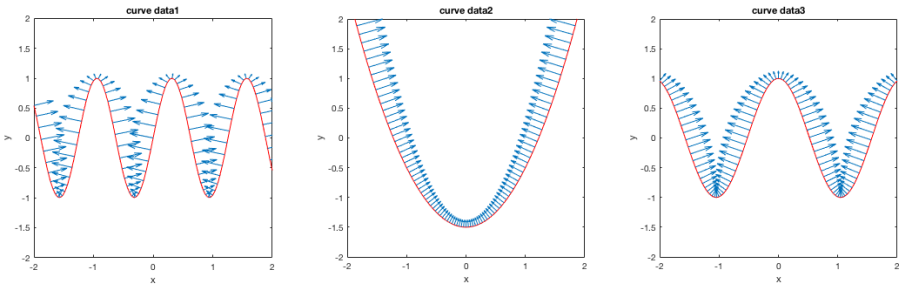
1.2.2 RUNNING

Run the script ***part1_2.m*** after setting the parameter *sigmas* for σ value.

1.2.3 RESULT

The data sets which contains 2D sample point coordinates and normal vectors were tested:

Figure 1.1: Visualization of the data sets



For each data sets, color maps and the fitted curves (green curves) were plotted with different values of σ:

Figure 1.2: The color map and fitted curve of data set 1

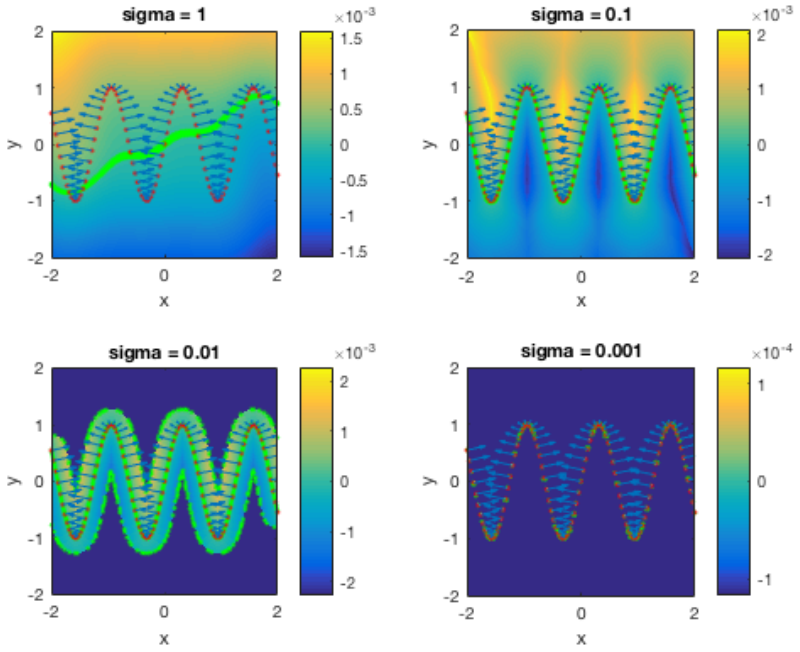


Figure 1.3: The color map and fitted curve of data set 2

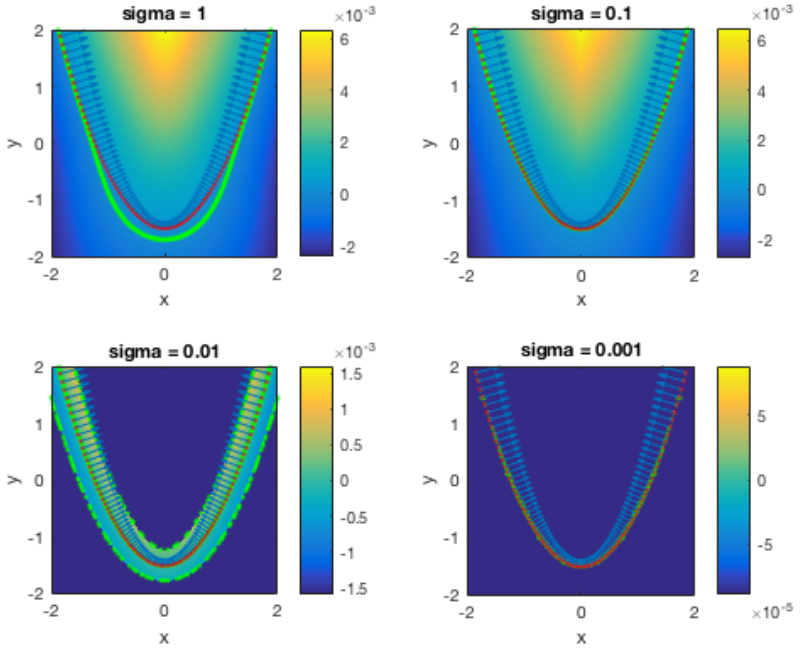
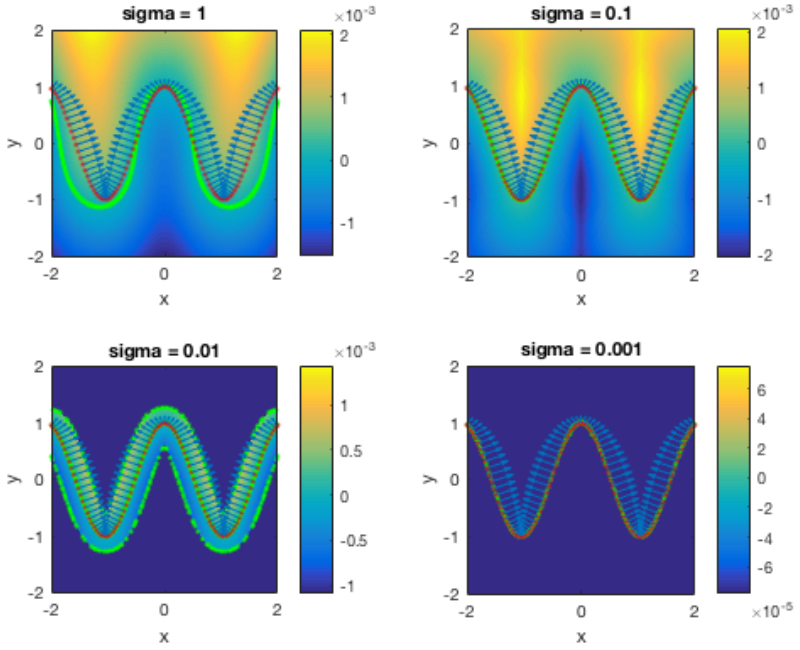


Figure 1.4: The color map and fitted curve of data set 3



1.2.4 DISCUSSION

- for every data sets, $\sigma = 0.1$ **showed the best result** i.e. the implicit MLS surfaces (curves) were fitted well with sample points with $\sigma = 0.1$.
- **the parameter σ corresponds to the moving window size.**
- when sigma becomes smaller than $\sigma = 0.1$, the size of the window is too small to get a proper curve.
- besides, as sigma becomes larger than $\sigma = 0.1$, the size of the window is too large, so the curve does not fit well with sample points.

1.3 SMOOTHING MESHES

1.3.1 DESCRIPTION

In the part 1.3, mesh smoothing algorithm based MLS algorithm was implemented. The provided mesh data is composed with vertex coordinates(3D sample points) and normals vectors. The projection of each vertex coordinate $\mathbf{v}$ to the smoothed coordinates of the mesh vertices $\mathbf{v}'$ follows $\mathbf{v}' = \mathbf{v} - f(\mathbf{x})\nabla f(\mathbf{x})$.

FORMULATION OF PROJECTION The derivation of $f(\mathbf{x})$ has been done in the part 1.1 (the equation 1.7). As taking gradient to $f(\mathbf{x})$, $\nabla f(\mathbf{x})$ can be obtained as follows:

$$\begin{aligned}
\nabla f(\mathbf{x}) &= \nabla \frac{\sum_i \phi_i(\mathbf{x})\left(\mathbf{n_i}^T(\mathbf{x}-\mathbf{x_i})\right)}{\sum_i \phi_i(\mathbf{x})} \\
&= \frac{\nabla\left(\sum_i \phi_i(\mathbf{x})\left(\mathbf{n_i}^T(\mathbf{x}-\mathbf{x_i})\right)\right)\left(\sum_i \phi_i(\mathbf{x})\right) - \left(\sum_i \phi_i(\mathbf{x})\left(\mathbf{n_i}^T(\mathbf{x}-\mathbf{x_i})\right)\right)\nabla\left(\sum_i \phi_i(\mathbf{x})\right)}{\left(\sum_i \phi_i(\mathbf{x})\right)^2} \\
&= \frac{\left(\sum_i \phi_i(\mathbf{x})\mathbf{n_i} + \sum_i \mathbf{n_i}^T(\mathbf{x}-\mathbf{x_i})\nabla\phi_i(\mathbf{x})\right) - \left(f(\mathbf{x})\sum_i \nabla\phi_i(\mathbf{x})\right)}{\sum_i \phi_i(\mathbf{x})} \\
&= \frac{\sum_i \left(\phi_i(\mathbf{x})\mathbf{n_i} + \left(\mathbf{n_i}^T(\mathbf{x}-\mathbf{x_i}) - f(\mathbf{x})\right)\nabla\phi_i(\mathbf{x})\right)}{\sum_i \phi_i(\mathbf{x})}
\end{aligned} \tag{1.9}$$

MATLAB IMPLEMENTATION Since the formulation for $f(\mathbf{x})\nabla f(\mathbf{x})$ is derived, smoothed coordinates of the mesh vertices $\mathbf{v}'$ can be calculated with certain value of σ: $\mathbf{v}' = \mathbf{v} - f(\mathbf{x})\nabla f(\mathbf{x})$. After several iterations, $\mathbf{v}'$ converges.

1.3.2 RUNNING

Run the script ***part1_3.m*** after setting the parameters.

sigma_array : sigma values for each data sets. Each rows correspond to each data sets.

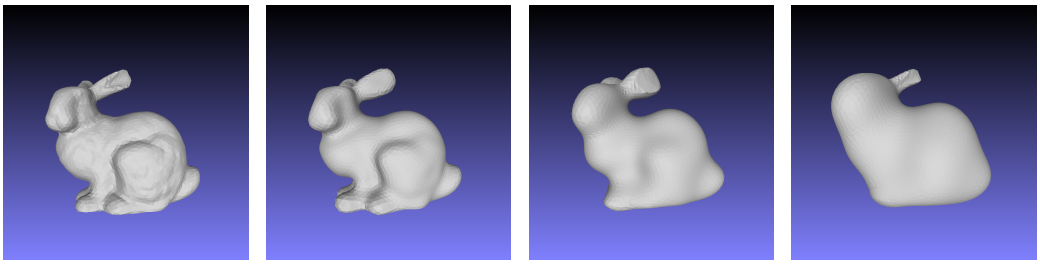
max_iter : the number of maximum iteration.

threshold : threshold for convergence. The default value is 10^{-4}.

1.3.3 RESULT

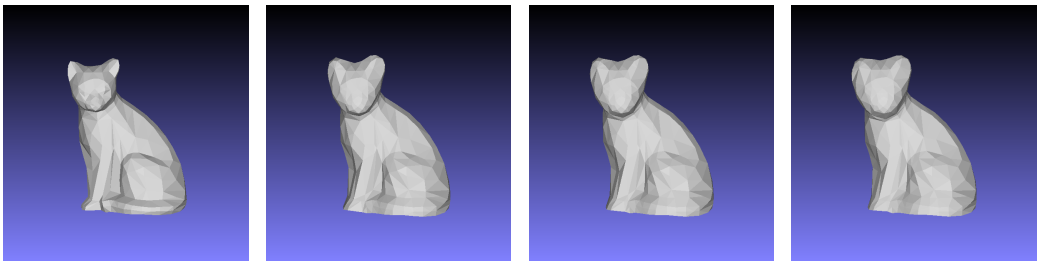
The result of mesh smoothing with different value of σ is as follows:

Figure 1.5: The smoothed mesh of the bun.off data set



(a) original (b) $\sigma = 0.01$ (c) $\sigma = 0.02$ (d) $\sigma = 0.03$

Figure 1.6: The smoothed mesh of the bun.off data set



(a) original (b) $\sigma = 80$ (c) $\sigma = 90$ (d) $\sigma = 100$

1.3.4 DISCUSSION

- comparing the results, **as σ increases, the amount of mesh smoothing increases.**
- remark that the parameter σ corresponds to the moving window size. Large value of σ means smoothing area is large.

1.4 SPEEDING UP MESH SMOOTHING

1.4.1 DESCRIPTION

In part 1.4, Approximate Nearest Neighbor Library was used to speed up mesh smoothing. By using the library function **frsearch**, $f(\mathbf{x})\nabla f(\mathbf{x})$ can be calculated for sample points 3σ away from the query point **x** only.

1.4.2 RUNNING

The MATLAB script ***part1_ 4.m*** corresponds to mesh smoothing with ANN library.
Before running the script, MATLAB ANN wrapper should be installed. Please note the followings.

- refer to ANN wrapper github repository: `https://github.com/jefferislab/MatlabSupport/tree/master/ann_wrapper`
- **randint** function used in ANN wrapper is deprecated in MATLAB 2016b version. If **randint** function of ANN wrapper library makes error during running time, change it to **randi**.
- although changing **randint** to **randi**, ***test_ ann_ class.m*** file which included in ANN wrapper library still does not work properly. However **frsearch** works without any problem.
- before running ***part1_4.m*** script, add ANN wrapper path.

1.4.3 RESULT

The running time for each data set with different σ is as follows:

Table 1.1: The running time of mesh smoothing

File	σ	Normal	ANN
bun.off	0.01	43.38	16.34
	0.02	36.18	47.18
	0.03	40.46	115.17
bunny.off	0.01	34.62	12.45
	0.02	34.67	46.14
	0.03	35.11	108.84
bunny2.off	0.01	34.90	15.28
	0.02	35.72	46.41
	0.03	36.21	129.90
cat.off	80	1.34	1.66
	90	1.15	1.19
	100	1.14	1.23

The discussion regarding the result follows on next page.

1.4.4 DISCUSSION

When σ is small, running time is reduced by **frsearch**, because the number of the points to calculate $f(\mathbf{x})\nabla f(\mathbf{x})$ becomes smaller. However, as σ is getting larger, advantage of using **frsearch** is not considerable anymore. Besides, finding neighboring points becomes overhead in this case, thus, running time even increased than before.

1.5 ADVANCED METHOD

1.5.1 DESCRIPTION

In part 1.5, sharp feature preserving mesh smoothing by Robust Implicit Moving Least Squares(RIMLS) method was implemented. In order to reduce the running time, the algorithm was vectorized.

Several implementation choices are determined as follows:

- for the spatial weight function $\phi_i(\mathbf{x})$, gaussian function was used.

$$\phi_i(\mathbf{x}) = \exp\Big(-\frac{\|\mathbf{x}-\mathbf{x_i}\|^2}{h_i^2}\Big)$$

- h_i determined as follows (same value was used for $i = 1, 2, \ldots, N$):

$$h_i = \frac{\sum_i^N 1.4 \times d_i}{N}$$

  where d_i is the minimum distance from sample point $\mathbf{v_i}$ to other sample points.

- $\sigma_r = 0.5$ was used.

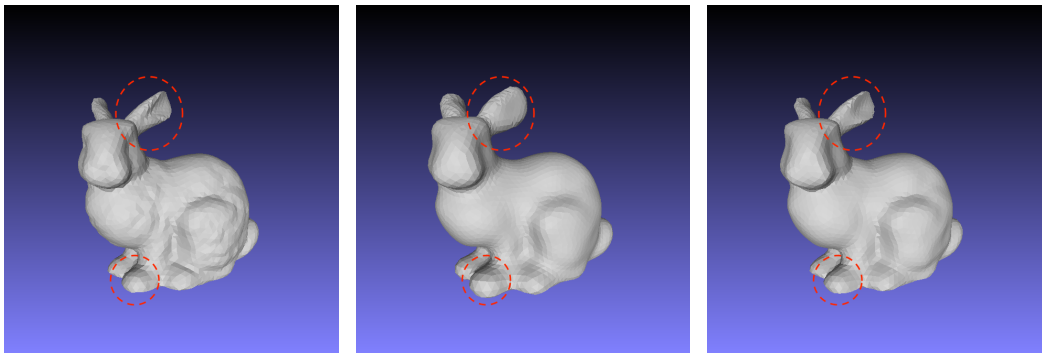
1.5.2 RUNNING

Before running the script ***part1_5.m***, MATLAB ANN wrapper should be installed. Please refer to part 1.4.
Since the results are very sensitive to parameters, they were tuned carefully. Do not change any parameter in the script.

1.5.3 RESULT

The following figures are smoothed mesh of the same data set. One is by normal moving least square (Implicit Moving Least Squares; IMLS) which was implemented in part 1.3 and the other is by advanced method, the RIMLS implemented in part 1.5.

Figure 1.7: Comparison of the smoothed mesh of the bun.off data set



(a) original (b) smoothed by normal IMLS (c) smoothed by RIMLS

The differences are salient especially for ear and foot part of the bunny model. The RIMLS smoothes similar amount but preserve sharp edges.

2 EXERCISE PART 2: IMAGE DEFORMATION USING MOVING LEAST SQUARES

2.0.1 DESCRIPTION

For part 2, moving least square used for image deformation. Three deformation model was adapted:

- affine transformation
- similarity transformation
- rigid transformation

The weights w_i for a control point $\mathbf{p_i}$, its deformed position $\mathbf{q_i}$ and an image point $\mathbf{v}$ is given as:

$$w_i = \frac{1}{\left|\mathbf{p_i} - \mathbf{v}\right|^{2\alpha}} \tag{2.1}$$

p_* and q_* are defined as follows:

$$\mathbf{p}_* = \frac{\sum_i w_i \mathbf{p_i}}{\sum_i w_i} \tag{2.2}$$

$$\mathbf{q}_* = \frac{\sum_i w_i \mathbf{q_i}}{\sum_i w_i} \tag{2.3}$$

AFFINE TRANSFORMATION For affine transformation, deformation function f_a is as follows:

$$f_a(\mathbf{v}) = (\mathbf{v} - \mathbf{p}_*)\left(\sum_i \hat{\mathbf{p}_i}^T w_i \hat{\mathbf{p}_i}\right)^{-1} w_j \hat{\mathbf{p}_j}^T \tag{2.4}$$

SIMILARITY TRANSFORMATION For similarity transformation, deformation function f_s is as follows:

$$f_s(\mathbf{v}) = \sum_i \hat{\mathbf{q}_i}(\frac{1}{\mu_s} A_i) + \mathbf{q}_* \tag{2.5}$$

$$A_i = w_i \begin{pmatrix} \hat{\mathbf{p}_i} \\ -\hat{\mathbf{p}_i}^\perp \end{pmatrix} \begin{pmatrix} \mathbf{v} - \mathbf{p}_* \\ -(\mathbf{v} - \mathbf{p}_*)^\perp \end{pmatrix}^T \tag{2.6}$$

RIGID TRANSFORMATION For similarity transformation, deformation function f_r is as follows:

$$f_r(\mathbf{v}) = \sum_i \hat{\mathbf{q}_i} A_i \tag{2.7}$$

where, A_i is from equation (2.6).

BACKWARD WARPING As deforming image and mapping each 2D point $\mathbf{v}$ of original image to deformed position $f(\mathbf{v})$ (f is f_a for affine, f_s for similarity and f_r for rigid transformation), artifacts can be made since deformed image points might be non-integer. In order to avoid these artifacts, backward warping was implemented.

Since the value of the deformation function $\mathbf{v}' = f(\mathbf{v})$ is obtained for every original image points, the inversed function f^{-1} can be obtained by regression. With f^{-1}, for each point of deformed image $\mathbf{v}'$, now $\mathbf{v} = f^{-1}(\mathbf{v}')$ can be calculated. If calculated $\mathbf{v}$ vector contains non-integer value, simply interpolated to nearest neighbor. For the regression, MATLAB function **griddata** was used.

2.1 RUNNING

Run the script ***part2.m*** after setting the parameters.

alpha : the α value of the weight w_i

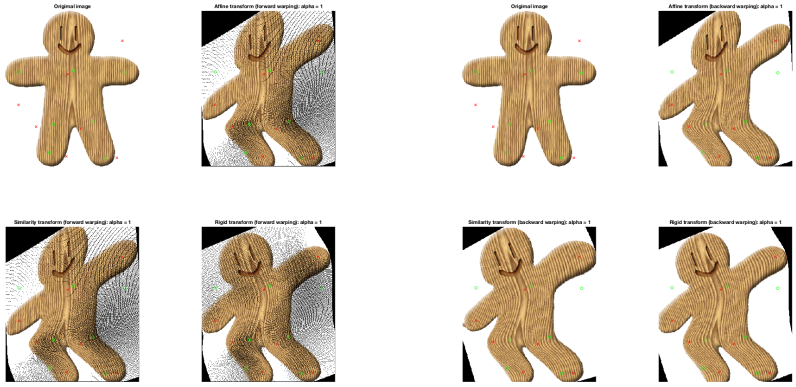
img_idx : 1 for ginger.png as an input, 2 for kerr.jpg

saved_control_pts : for ginger.png, saved control points can be used. (for debugging)

2.2 RESULT

The results of the deformation with $\alpha = 1$ is as follows.

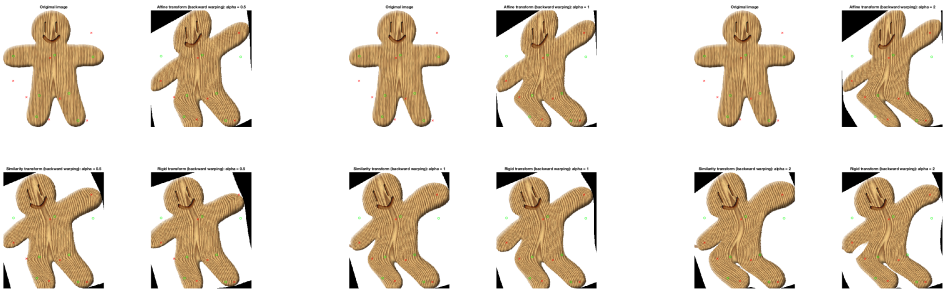
Figure 2.1: ginger.png for different transformation model



(a) forward warping (b) backward warping

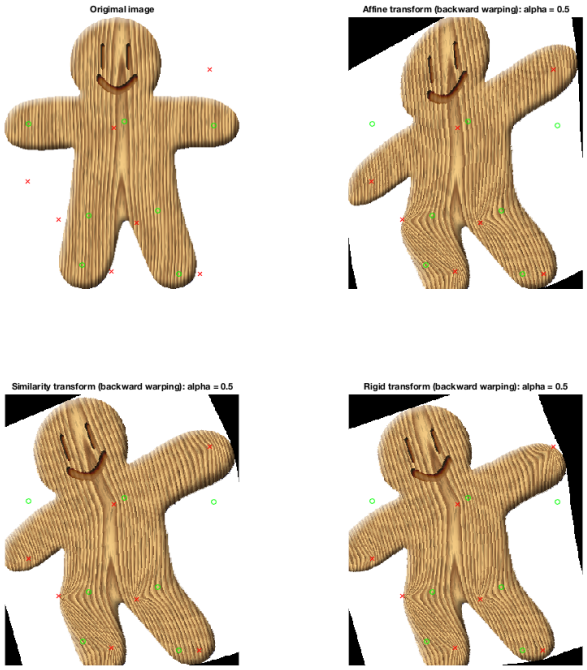
Comparing for different value of α:

Figure 2.2: ginger.png with different α

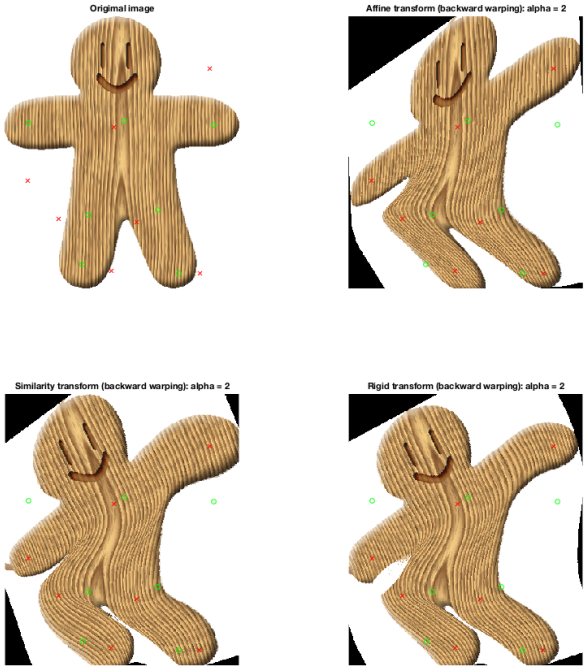


(a) $\alpha = 0.5$ (b) $\alpha = 1.0$ (c) $\alpha = 2.0$

Figure 2.3: ginger.png with $\alpha = 0.5, 2.0$ large images



(a) $\alpha = 0.5$



(b) $\alpha = 2.0$

As the α increases, control points $\mathbf{p_i}$ tend to follow deformed points input $\mathbf{q_i}$ more. Finally, adapted the algorithm to another image which was carefully chosen :)

Figure 2.4: kerr.jpg with $\alpha = 1.0$

2.3 Discussion

- adapting backward warping, the artifacts (black dots) were disappeared.
- as the α increases, control points $\mathbf{p_i}$ tend to follow deformed points input $\mathbf{q_i}$ more. α determines weight w_i for each control point.
- rigid transformation is the most natural.