U.P.B. Sci. Bull., Series C, Vol. 70, No. 2, 2008 ISSN 1454-234x

EXTENSIONS TO THE MLS METHOD FOR IMAGE DEFORMATION

Cosmin ATANASOAEI[1], Florica MOLDOVEANU[2]

Articolul prezinta mai multe extensii ale metodei MLS de deformare a imaginilor. Propunem o metoda eficienta de constructie a imaginii rezultate si argumentam pentru utilizarea unor functii diverse de ponderare dupa distanta in calculul deformarii. Extensiile sunt integrate intr-un sistem original (IDS), special construit pentru a permite evaluarea obiectiva a diferitelor variatii MLS, utilizand metodele numerice de estimare a erorii de deformare prezentate in finalul lucrarii.

This paper describes some useful extensions to the MLS image deformation method. We propose different weighting functions that can improve the deformation results and an efficient deformed image reconstruction method. Also, a new modular and configurable system (IDS) specially designed for evaluating different MLS variations is presented. It uses our proposed numerical evaluation methods of the deformation results.

Keywords: image deformation, Moving Least Squares, affine transformation, rigid transformation, interpolation, numerical analysis, error field.

1. Introduction

Image deformation is a widely used technique for animation, morphing, medical imaging or advanced user interfaces. All the deformation methods use a set of user defined geometrical handles to manipulate the image by modifying the pixel positions. A detailed description of the most used methods can be found in [1], [2] or [3] and a very interesting application is presented in [4].

The deformation can be defined by a deformation function *f*, computed at pixel level, that gives the expected deformed image when applied to the original image; the *f* function does not change the pixel colors, but move them using the original position of the handles and the user modified positions; as described in [1], the handles can be control points, user specified lines or curves, some coarse grid or even a triangulation. This paper focuses only on using control points, but the results can be applied successfully to other forms of handles.

This paper defines extensions of the Moving Least Squares (MLS) method as described in [1], using ideas from spatial data interpolation methods. Chapter 2

[1] MSc, Dept. of Computers, University POLITEHNCA of Bucharest, Romania, E-mail: accosmin@gmail.com
[2] Prof., Dept. of Computers, University POLITEHNCA of Bucharest, Romania

presents the basic MLS method and the next ones describe our proposed extensions. Chapter 3 argues for the importance of using different weighting functions in computing the *f* function and presents a detailed method for the construction of the deformed image. Chapter 4 describes our modular system for image deformation; the system has configurable components for weighting functions, deformation types and the deformed image construction. Chapter 5 presents our methods for numerical evaluation of the system, which can be successfully used in choosing the best configuration for a given image, image set or specific handles: real-time property, reference triangulation error and MLS error field.

2. The MLS method

This chapter introduces the main ideas of MLS method, using the same notations as in [1]. Let's consider the set of control points having the original positions as *p* and the current positions (modified by moving or dragging the handles) as *q*. The deformation function *f* should transform the set *p* into *q*, such that the deformation should be as "natural" as possible. The properties of this function are borrowed from the spatial data interpolation theory (they find the best values in new positions, by considering a restricted set of known values in some original positions):

- *Interpolation*: $f(p_i) = q_i$;
- *Smoothness*: *f* should be chosen to produce smooth deformations;
- *Identity*: if the handles positions are not modified, the deformed image should be identical with the original.

Accordingly with the interpolation theory, the pixels closer to the *p* should be deformed more than those farther away; the logical consequence of this statement is that the deformation function should be different from pixel to pixel and some weighting (generally based on distance) should be used when computing it.

As described in [1], *f* should minimize the square error of transforming *p* into *q*; additional constraints are added as to use only some transformations (affine, similarity or rigid) and to respect the properties above. For real time results, the deformation function of each pixel *v* in the image should be as simple as possible; in this case, the best candidate is a linear transform as in (1).

$$f(v) = l_v(v) = vM + T \qquad (1)$$

The linear *transformation matrix M* is considered constant, for simplicity; *T* is the *translation vector.* The square error to minimize can be expressed in the form of $l_v(v)$ linear transform, as in (2).

$$\sum_i w_i \left| l_v(p_i) - q_i \right|^2 \quad (2)$$

The weights w_i are computed in most cases using the inverse of the distance between the pixel *v* and some handle; in chapter 3, more details and alternatives for computing weights will be discussed. Merging equations (1) and (2), a single error function to minimize is obtained, but with two independent parameters: *M* and *T.* Given the convex form of the error function (square form), the minimum value is situated on the gradient descend path; equalizing with zero the partial derivate of the error function with respect to *T* parameter – (3), the error function can be reduced to just one parameter: *M* matrix – (4).

$$T = q_* - p_* M \quad (3)$$

$$\sum_i w_i \left| \hat{p}_i * M - \hat{q}_i \right|^2 \quad (4)$$

The auxiliary values used in equations (3) and (4) have the meaning of *weighted centroids* (p_* and q_*) and *difference to centroids* ($\hat{p}_i = p_i - p_*$ and $\hat{q}_i = q_i - q_*$).

Finding *M* using equation (4) is done by equalizing with zero the partial derivate with respect to *M* and having the solution in one simple equation. In [1], there are presented three different types of transformation with additional restrictions:

- *Simple affine*: *M* is a general affine transformation;
- *Similarity:* only uniform scaling, rotation and translation are allowed; the restriction is:
  $$M^T * M = \lambda^2 I_2 \quad (5)$$
- *Rigid transformation:* the most restrictive; scaling is not allowed, only rotation and translation; the restriction is:
  $$M^T * M = I_2 \quad (6)$$

Every transformation type has a simple straightforward solution and major parts in this formula can be pre-computed when handles are defined; more details can be found in [1]. That makes the *f* computation and the deformed image construction a real time process.

3. MLS method extensions

This chapter presents two extensions to the MLS method: the first covers different weighting functions to use for the *f* computation and argues for their importance in the deformation results, often ignored in similar papers; the second presents an efficient method for constructing the deformed image with no gaps between pixels.

Considering the close relation with spatial data interpolation methods (*f* properties, distance based weighting, transformation matrix computing method), computing the deformed image should use the best techniques from these methods, that already have proven their value. The first and likely to mostly influence the deformation results is the way weights used in (2) are computed; in [1] only an inverse distance is presented; in [5] and [6] there is a detailed description of the most widely used methods for spatial interpolation. A very nice presentation of B-Spline interpolation results using different parameters is presented in [7] and shown in Fig. 1.

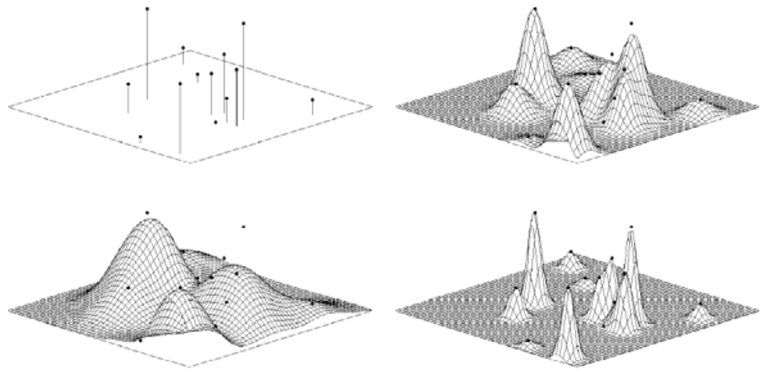

Fig. 1. Interpolation results using different parameters for B-Spline method; from [7]. Scattered points presented in top-left image, interpolation results in the other ones.

If the interpolation results are so different just by using different parameters or methods, there is no doubt that also the deformed image will greatly be influenced by the weights (see equation (2) for details). Having alternatives to choose from, the best one can be identified as to have the best results for a given image and set of handles or even for large sets of images. For example, artists that build animated characters with a smart deformation system could use different weighting variants and choose the configuration that best suits their needs.

Let d_i be the distance between the pixel v and the handle point p_i as $d_i = \|v, p_i\|$. The weight for p_i is a function of d_i, as $w_i = w(d_i)$, with large values for smaller distances: the influence of the handle point is limited locally.

Table 1 presents the most common weighting functions; their number is of course much larger and their mathematical properties are beyond the scope of this paper, further details can be found in [5], [6] and [7]. The influence of weights in interpolation results is given mainly by the form of their definition function: there are exponential variations, linear, quadric or more complicated forms. Test have been done for the functions that use an influence radius parameter to get the optimum value of this parameter; the best choice for influence radius is around the image's diagonal and can be decreased if the number of handles is increased.

Table 1

Examples of the most used weighting functions in interpolation field

Name	Weight function	Parameter	Meaning
Inverse distance weighting	$w_i = \frac{1}{s + (d_i)^{2\alpha}}$	$s > 0$	Adjusting factor for cases where d_i is very small or zero.
		$\alpha > 0$	Power factor, controls the variations of weight with distance. Typical values: 1, 2.
Cressman	$w_i = \max\left(0, \frac{R^2 - d_i^2}{R^2 + d_i^2}\right)$	$R > 0$	Influence radius, above R the influence is zero.
Shepard modified	$w_i = \max\left(0, \frac{R - d_i}{R * d_i}\right)$	$R > 0$	Influence radius, above R the influence is zero.
Gaussian	$w_i = e^{\left(-d_i^2 / R^2\right)}$	$R > 0$	Influence radius, above R the influence is very small.
Distance Linear	$w_i = \max(0, R - d)$	$R > 0$	Influence radius, above R the influence is zero.

An example of the influence which the weighting function has on the deformation results is given in Fig. 2; the nose and mouth area are the easiest to recognize as different. It's easy to see that the chose of weighting function can influence greatly the deformation results; the weight selection should be done using a system as described in section 4 which integrates numerical error evaluation methods like the ones we propose in section 5.

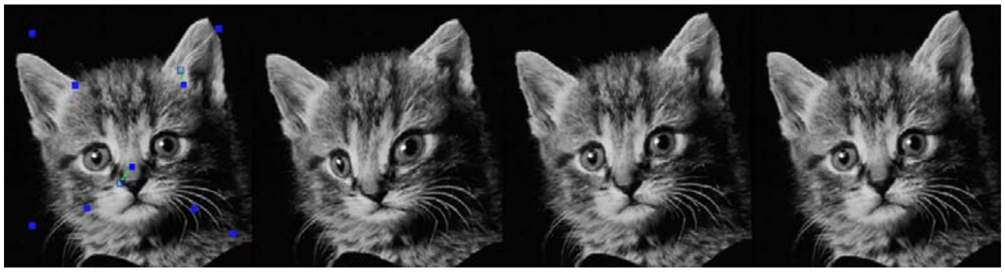

Fig. 2. Deformation results using the affine transformation and varying the weight method; Left to right: initial image with handles, IDW with *s = 0.00001* and $\alpha = 1$, Shepard with *R = 600,* Distance Linear with *R = 300.*

Our proposed second extension is the method for building the deformed image, given the *f* function computed at pixel level. The *forward* methods move pixels from the original image to the deformed image – from *v* to *f (v)*, and applying interpolation techniques to fill gaps, if they appear. The *backward* methods move pixels the other way around and are more complex in this case, needing additional processing steps to compute the inverse of deformation function *f* or to approximate it using interpolation techniques. This paper is focusing only on forward methods.

The easiest forward method is to copy color from pixel *v* in the initial image to pixel *f(v)* in the deformed one – *simple forward*; the problem is that gaps appear where stretching is above 1 pixel limit and, because of approximations in pixel coordinates for the position *f(v)*, pixels remain uncovered.

An example is given in Fig. 3, the right image; because of distortions generated by different deformation functions *f* (computed locally!) pixels remain uncovered. One possible solution is to interpolate the remaining uncovered pixels to their covered neighbors, but additional problems can arise in trying to delimit those pixels from the background – e.g. the bottom of the same image.

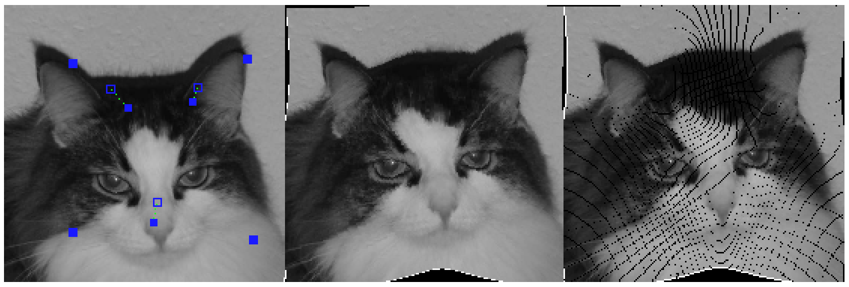

Fig. 3. Deformation results using the similarity transformation;
Left to right: initial image with handles, forward with polygon interpolation, simple forward.

Our proposed forward method is called *forward with polygon interpolation* and its results are shown in Fig. 3, the center image; the deformed image is much better than the simple forward image and the background is left uncovered as it should be.

The main idea of *forward with polygon interpolation* is to fill quadrilaterals generated by deforming squares of adjacent pixels using nearest neighbor interpolation; it can be extended to use more complex interpolation methods. Fig. 4 presents a square of adjacent pixels (left side) that is deformed, using *f* functions computed in its vertices, into a quadrilateral marked with dashed lines; generally the quadrilateral has its vertices in between pixels and the method uses its vertices coordinates in floating point precision. The next step is to compute the pixels that are inside – marked with light gray. In [8] there are details of the most common solution: checking if a point *P* is inside some polygon *PG* is the same as counting the intersections of ray from *P* parallel with Ox axes with *PG* edges and checking if the count is odd. If some candidate pixels is found inside the quadrilateral then its color is computed based on its distances to the vertices and their colors, using an interpolation method; in our tests, the nearest neighbor was used.

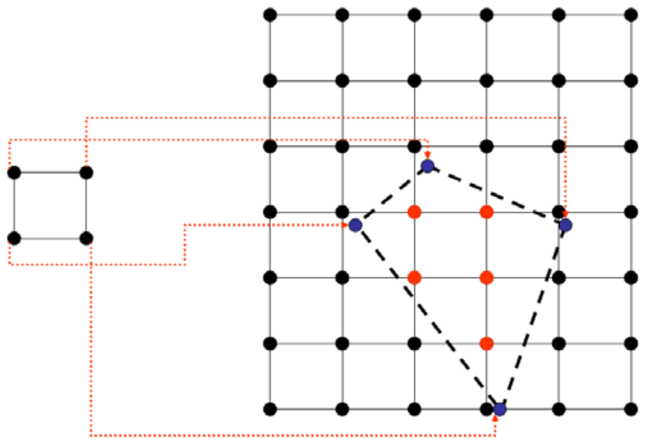

Fig. 4. Details of *forward with polygon interpolation* method.

Our method can be described in pseudo-code as:

```
For each square S of adjacent pixels in the initial image
    Compute quadrilateral Q as applying f for S vertices (floating point coordinates)
    Compute the bounding rectangle box R for Q (pixel coordinates)
    For each candidate pixel P in R
        If P was not marked and P inside Q, then
            Interpolate P's color using Q vertices' colors and mark P.
```

Nearest neighbor method was chosen for the last part because it's a very fast method that can improve the speed of the overall deformation algorithm and because it uses only the colors in the initial image, no new colors are computed and therefore no error is added.

The way deformation function *f* is computed guarantees that quadrilateral *Q* doesn't intersect with its neighbors, therefore pixels colors are computed only once; the simple forward method can compute some pixel's color multiple times, because of approximating *f(v)* floating point coordinates to pixel coordinates. But, the bounding rectangle *R* can intersects with its neighbors and performance problem can appear if the same pixel is checked for intersection with multiple quadrilaterals; the simplest way to solve this problem is to mark pixels already checked and for each candidate pixel in *R* first must be checked if already assigned to some quadrilateral.

By using the pixel marking strategy described as above, we enforce that each pixel is only once considered for the intersection with the quadrilateral *Q* and therefore the computing time is decreased. Also, speedup is achieved by using the fast intersection algorithm as described in [8] and the nearest neighbor interpolation method; this has also the benefit of keeping the deformation error – described in chapter 5, as low as possible.

4. The Image deformation system (IDS)

Image deformation using MLS can be extended, as proved in the previous chapters, with many alternatives for: transformation types (others can be provided, besides the three described in [1]), weighting functions and the deformed image construction algorithms. By incorporating all these variations into a modular system and having a numerical analysis method for the evaluation of the results, experiments can be made to choose the configuration that provides the best deformed image.

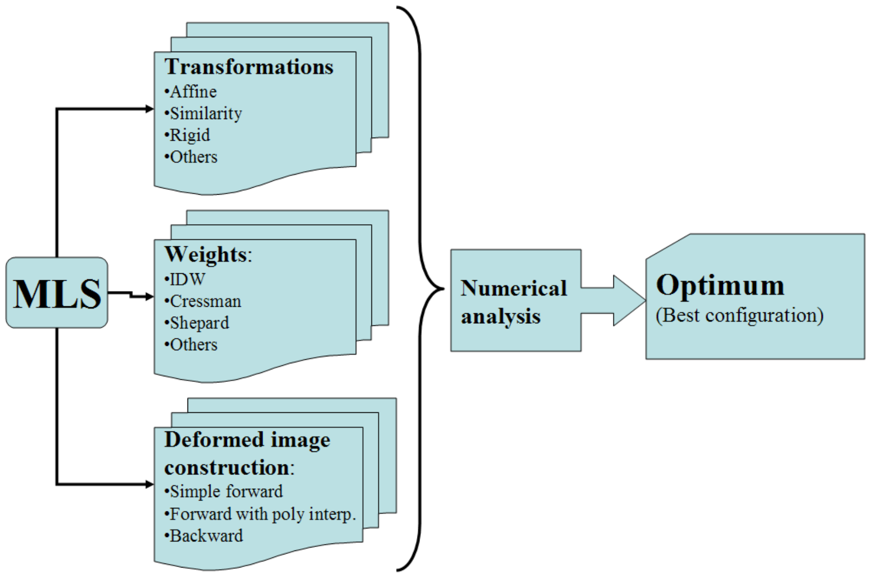

Fig. 5. Image deformation system (IDS) – modules.

Our proposed system is presented in Fig. 5. The numerical analysis module will be discussed in the next chapter; using this module, choosing the best configuration is a matter of running tests and comparing results with respect to error criteria given in the next chapter. Every MLS module can be further extended with methods tuned for speed and quality results that meet better specific application requirements. Being an interactive system, IDS must provide results in real time and to achieve that, every composing module was optimized for speed.

5. Numerical analysis

Usually the evaluation of the image deformation results is done in terms of speed and visual inspection for artifacts; for details see [1], [2] and [3]. In this paper, we argue for *numerical and unambiguous analysis methods* that should take into account subtle changes in results that can be missed by visual inspection or interpreted differently by different persons. We propose two numerical methods for comparing the results of different IDS configurations and an intuitive graphical representation of the error field.

The first method uses relation (2) that MLS is minimizing and gives a statistical approach of how well that error was decreased. Attention must be taken, because in (2) weights are part of the equation and trying to compare different IDS configuration with different weight functions and different domains would lead to erroneous results; therefore the MLS error at pixel level is computed as:

$$E_{MLS}(v) = \sum_i \left| l_v(p_i) - q_i \right|^2 . \tag{7}$$

The next step is to provide statistical information at image level, by considering the average – (8), minimum or maximum error, its standard deviation – (9), per cent of pixels that are within 1x, 2x or 3x of standard deviation from average.

$$E_{MLS}(I) = \frac{\sum_{v \in I} E_{MLS}(v)}{|I|} \tag{8}$$

$$\Gamma_{MLS}(I) = \sqrt{\frac{\sum_{v \in I} \left(E_{MLS}(v) - E_{MLS}(I)\right)^2}{|I|}} \tag{9}$$

The graphical representation for the MLS error field is straightforward: after choosing two colors for maximum and minimum possible values, the color for a given pixel level error is linearly interpolated between the extreme colors; the error map obtained with IDS and built this way is very intuitive as shown in Fig. 6 - the right image. The regions where error is greater (marked with intense gray value) correspond with the four handles that have been dragged; the larger the distance between p_i and q_i the greater the error in the area around them and the error map will be more red intense there too.

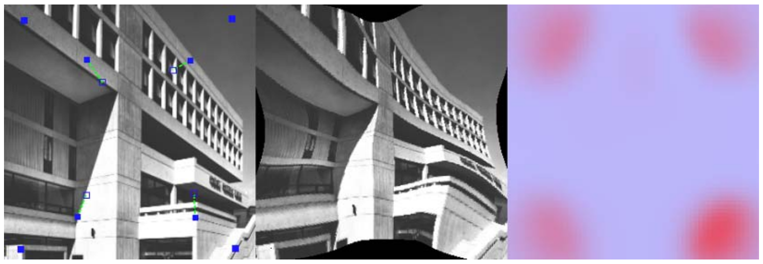
Fig. 6. Left to right: initial image with handles,
Deformed image using similarity transform and IDW, MLS error field.

The second method uses reference points given by the user and builds a Delaunay triangulation; for every triangle the angles and edge lengths are compared in the initial image and in the deformed one. Details for Delaunay triangulation can be found in [9].

Let T_j be a triangle with vertices between the reference points- $T_j = (r_{j(1)}, r_{j(2)}, r_{j(3)})$; the deformed triangle is $T_j^d = (f(r_{j(1)}), f(r_{j(2)}), f(r_{j(3)}))$. For every pair (T_j, T_j^d) the lengths and angles variations are computed using (10) and (11) equations; next, the error statistics are computed for the whole triangulation.

$$E_j^{\Delta} = \frac{1}{3}\sum_{(k,l)} \frac{\left\| (r_{j(k)}, r_{j(l)}) - (f(r_{j(k)}), f(r_{j(l)})) \right\|}{\left\| (r_{j(k)}, r_{j(l)}) \right\|} \tag{10}$$

$$E_j^{\angle} = \frac{1}{3}\sum_{k=1,3} \frac{\left| \cos(\prec(r_{j(k)}) - \cos(\prec(f(r_{j(k)})))) \right| + \varepsilon}{\left| \cos(\prec(r_{j(k)})) \right| + \varepsilon} \tag{11}$$

The graphical representation for triangle errors is done in the same way as in the MLS case. In Fig. 7, the right image is an angular error graphical representation for the same settings used in Fig. 6; the center image depicts the Delaunay triangulation for the user given reference points. The triangles with the greatest errors are not situated as for MLS error field, near dragged handles.

The triangulation is very useful to check how much an object in the image is deformed, primarily how much its shape is distorted and not how the scaling factors vary; this makes the angular error much more useful than the distance error. In simple case of scaling the initial image with 50%, the triangles would not change angles, but only edge distances; in this case the angular error would be zero – which is a good indicator that the algorithm gives the correct results, and the distance error would be very high – not a very useful indicator. The results evaluation should be done, therefore, using only the angular error and its statistical coefficients as in MLS error method.

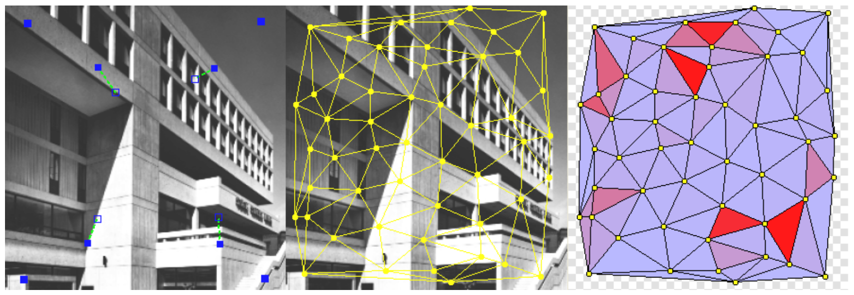
Fig. 7. Left to right: initial image with handles,
Initial image with Delaunay triangulation, angular error graphical representation.

The two proposed methods represent a very useful tool for numerical evaluation of different IDS configurations, but they have limitations.

The most important is that computing errors in real time needs the explicit computation of transformation matrix M and translation vector T which is quite expensive; in the original MLS method, the transformation function f was not computed explicitly, but only the final position for pixel v in the initial image, which allowed speeding up the algorithm - details can be found in [1]. By including the numerical analysis module, the IDS system becomes slower, but we used a coarser grid and an optimized forward polygon interpolation method to reduce this problem; this way we obtained results in real time.

Another limitation is that the two proposed methods do not cover the final step of constructing the deformed image – simple forward and forward with polygon interpolation methods. Further studies should try to find a way of building a new metric for evaluating the error at this step too; one idea would be to use the user reference points (the same as in computing angular errors) as indicators of how much the color field has changed around them.

6. Conclusions

We have proposed several extensions to the MLS image deformation method – weighting functions and forward deformation image reconstruction methods, a system to test and evaluate all the possible configurations, two numerical analysis methods and graphical representations of the deformation error. The IDS system gives the user the flexibility to choose from a wide range of possible configurations the one that best suits its needs. Being built with real-time performance as a priority, the deformed image and the error fields are generated any time the user drags the handles. The system can be further extended by improving existing methods or adding new ones; for example, an algorithm for backward computing the deformed image should be included, to test against the forward variants. Also other methods for computing weights and transformations should be considered in further studies.

The numerical analysis methods presented in this paper helps evaluating and choosing the best configuration for a set o images and handles; fine tuning can be achieved using an image, a set of handles, reference points and IDS configurations as input and random generating deformations and evaluating the results. In this way the best configuration can be determined automatically.

Further studies should focus on extensive evaluation of different MLS variations with our system on different image sets and try to prove that no IDS configuration can perform best in terms of our error estimation methods on every image set. Chapter 3 presents some results that point to this idea, but additional experiments should be done.

The proposed system can be extended to the 3D deformation case; the MLS, Transformation and Weights module should require only small modifications, but the Deformed image construction and Numerical Analysis are much more difficult to extend. Deformation in 3D case can be used in numerous applications such as animation or simulation of complex systems.

Speeding up the IDS system (needed if extended to 3D case) can be done using the GPU unit and memorizing pixel level information in textures; the main problem to be solved is that the information to memorize at pixel level varies linearly with the number of handles and cannot be all stored in textures, because the number of textures that can be used is limited.

R E F E R E N C ES

[1] *Scott Schaefer, Travis McPhail, Joe Warren,* Image deformation using Moving Least Squares, in ACM Transactions on Graphics, **vol. 25**, Issue 3, July 2006, pages 533-540.

[2] *Takeo Igarashi, Tomer Moscovich, John F. Hughes*, As-Rigid-As-Possible Shape Manipulation, in ACM Transactions on Graphics, **vol. 24**, Issue 3, July 2005, pages 1134-1141.

[3] *Marc Alexa, Daniel Cohen-Or, David Levin,* As-Rigid-As-Possible Shape Interpolation, in Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 2000, pages 157-164.

[4] ***, http://www-ui.is.s.u-tokyo.ac.jp/~takeo/research/rigid/index.html.

[5] *Isac Amidror*, Scattered data interpolation methods for electronic imaging systems: a survey, in Journal of Electronic Imaging, **vol. 11,** Issue 2, April 2002, pages 157-176.

[6] ***, http://www.ecmwf.int/newsevents/training/rcourse_notes/DATA_ASSIMILATION.

[7] *Seungyong Lee, George Wolberg, Sung Yong Shin*, Scattered Data Interpolation with Multilevel B-Splines, in IEEE Transactions on Visualization and Computer Graphics, **vol. 3**, Issue 3, July 1997, pages 228-244.

[8] ***, http://local.wasp.uwa.edu.au/~pbourke/geometry/insidepoly/.

[9] ***, http://en.wikipedia.org/wiki/Delaunay_triangulation.