An Optical Flow Technique for Image Metamorphosis

Minas E. Spetsakis
Dept. of Computer Science, York University
4700 Keele Street, North York, ONTARIO
CANADA, M3J 1P3
tel (416) 736-5053
fax (416) 736-5872
e-mail minas@cs.yorku.ca

ABSTRACT

This paper presents an application of optic flow estimation to image metamorphosis, which is used in video production. A stable optical flow algorithm was designed to compute image deformation that is insensitive to the constant intensity assumption. The computed deformation is applied to the images to transform one smoothly into the other. The method presented automates the labor intensive process of specifying the deformation manually.

1. Introduction

Image metamorphosis or *morphing* is a very useful technique in computer graphics and animation. It is routinely used commercially for special effects. In a recent paper Beier and Neely [3] described how this is done by professional animators at Pacific Data Images and Wolberg [16] how it is done by their colleagues at Industrial Light and Magic. Despite the differences the two methods consist mainly of the following steps.

* The operator specifies via a GUI the deformation. The result is a sparse description of the deformation.
* The system computes the complete deformation field from the sparse data.
* The system warps the image and interpolates the color.
* The cycle is repeated until the result is good.

The procedure can be applied to still images or on individual frames from movies. Major portion of the time is spent on manually specifying the deformation.

The support of the NSERC (App. No. OGP0046645) is gratefully acknowledged.

2. Using flow to do morphing

In this paper we present a method to estimate the deformation field using optical flow algorithms developed for vision. These algorithms compute the amount of deformation (flow) between successive images in an image sequence. These algorithms are often considered unstable for the simple reason that when used with structure from motion algorithms the end result is unstable. If one is concerned only with the deformation field then several flow algorithms work fairly well [2] in the sense that the needle maps look subjectively correct which is sufficient for morphing. It is also relatively straightforward to take two images from the same sequence and transform one into another. This technique is used by several video compression schemes like MPEG.

Something similar can be done for morphing. Take the images of two people and compute the deformation between them and then use this to morph one into another. The problem is that although the images

are not completely different (e.g. both contain faces) the difference is large enough to confuse most flow algorithms [8, 10]. As opposed to the flow algorithms used for structure from motion, algorithms for estimating the deformation for morphing need not worry that much about the aperture problem (the ambiguity of the deformation field when examining a small patch of an image as viewed through an aperture) or accurate detection of discontinuities. But they should work well for pairs of images where the constant intensity assumption is grossly violated and the displacement is large and varying. We tried several algorithms and the one that worked best uses affine model for flow [14, 15, 5] with Gabor filtering [12]. The Gabor filters are particularly good at capturing the motion of shapes without much regard to slow variations of intensity [7] and the affine model can accommodate the varying flow field within the large support of the Gabor filters.

3. Affine flow algorithm

We start from the fundamental constraint for optical flow which is based on the constant intensity assumption

$$I_x u + I_y v + I_t = 0$$

which cannot be used directly. The three main problems are that it is one equation for two unknowns, requires derivatives and is very sensitive to the constancy of the intensity. The first problem we can cure by assuming that the flow does not change much in a small neighborhood and try to fit u and v in every neighborhood using least squares. We can generalize it, if we notice that this is equivalent to fitting u and v to linear combinations of the constraints in this neighborhood. If the linear combinations are the same for every neighborhood in the image then the linear combinations can be expressed as convolutions with a set of filters.

Assume that we have a series of filters g_q for $q = 1..q_{\max}$. We can try to minimize

$$SSE = \sum_q \left\{ I_x^{(g)} u + I_y^{(g)} v + I_t^{(g)} \right\}^2 \qquad (1)$$

where the superscript (g) denotes convolution of the image I with the filter g_q, in this case a Gabor filter, u and v are the components of the flow and the subscripts denote partial derivatives with respect to x, y and t. This is similar to what is used in [6] which gives very stable results with slow varying flow. But since we use Gabor filters with quite big support, what we do in effect is blend constraints from a large area of the image and we cannot assume that the flow remains constant over the whole area. But we can relax the restriction considerably, by using the assumption that the flow can be, more or less, described by an affine equation in a small neighborhood. This basic idea has been used in various forms in the past [4, 11, 5]. The flow then takes the form

$$u(x,y) = u(x_0,y_0) + u_x(x_0,y_0)\cdot(x-x_0) + u_y(x_0,y_0)\cdot(y-y_0)$$

$$v(x,y) = v(x_0,y_0) + v_x(x_0,y_0)\cdot(x-x_0) + v_y(x_0,y_0)\cdot(y-y_0)$$

This will allow the size of the filter (or the variation of the flow within a given area) to be considerably larger. Then

$$\begin{aligned} &I_{err}(x_0,y_0;x,y) = \\ &I_x(x,y)u(x_0,y_0) + I_x(x,y)u_x(x_0,y_0)\cdot(x-x_0) + \\ &I_x(x,y)u_y(x_0,y_0)\cdot(y-y_0) + \\ &I_y(x,y)v(x_0,y_0) + I_y(x,y)v_x(x_0,y_0)\cdot(x-x_0) + \\ &I_y(x,y)v_y(x_0,y_0)\cdot(y-y_0) + \\ &I_t(x,y) \end{aligned}$$

We now take a linear combination over a small region defined by function g_q to get the affine residual

$$\begin{aligned} &I_{aerr}(x_0,y_0) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I_{err}(x_0,y_0;x,y)g_q(x-x_0,y-y_0)\,dx\,dy = \\ &I_x^{(g)}(x_0,y_0)u(x_0,y_0) + I_y^{(g)}(x_0,y_0)v(x_0,y_0) + \\ &I_x^{(gx)}(x_0,y_0)u_x(x_0,y_0) + I_y^{(gx)}(x_0,y_0)v_x(x_0,y_0) + \\ &I_x^{(gy)}(x_0,y_0)u_y(x_0,y_0) + I_y^{(gy)}(x_0,y_0)v_y(x_0,y_0) + \\ &I_t^{(g)}(x_0,y_0) \end{aligned}$$

where the superscripts mean convolution with the corresponding filter. The filter $g_q x$ comes from terms like

$$I_x^{(gx)}(x_0,y_0) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I_x(x,y)\cdot(x-x_0)\cdot g_q(x-x_0,y-y_0)\,dx\,dy$$

Then the sum of squared absolute errors is

$$SSE = \sum_q \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} |I_{aerr}|^2 \qquad (2)$$

where

$$I_{aerr} = I_x^{(g)}u + I_y^{(g)}v + I_x^{(gx)}u_x + I_y^{(gx)}v_x + I_x^{(gy)}u_y + I_y^{(gy)}v_y + I_t^{(g)}$$

We used absolute values because the filters are complex valued Gabor filters. Notice that some of the convolutions are with $g_q(x,y)\cdot x$ etc. The x multiplier comes from the derivative terms. The Euler equations for this minimization are

$$\begin{aligned} I_x^{(g)} I^*_{aerr} - \frac{\partial}{\partial x}\left(I_x^{(gx)} I^*_{aerr}\right) - \frac{\partial}{\partial y}\left(I_x^{(gy)} I^*_{aerr}\right) = 0 \\ I_y^{(g)} I^*_{aerr} - \frac{\partial}{\partial x}\left(I_y^{(gx)} I^*_{aerr}\right) - \frac{\partial}{\partial y}\left(I_y^{(gy)} I^*_{aerr}\right) = 0 \end{aligned} \quad (3)$$

where the star superscript denotes complex conjugate and I_{aerr} is defined in Eq. (2). A common difficulty with such algorithms is the computation of derivatives. It is well known that the derivatives amplify the higher frequencies which are dominated by noise. But this is not the whole story. The derivatives are numerically hard even with synthetic images where noise is not a problem. In [9] is shown that the accuracy of the numerical differentiation depends on the method used. If a two tap filter is used (finite differences) the result is accurate for the lower frequency components of the signal (about one sixth of the spectrum) and less and less accurate for higher frequencies. The accuracy can be increased using more expensive filters (more taps). In effect using more taps one can increase the part of the spectrum that the computation is accurate.

But using more expensive filters will not solve all the problems. The main difficulty is introduced by terms like

$$\frac{\partial}{\partial x}\left(I_x^{(gx)} I^*_{aerr}\right)$$

where we take the derivative of what is essentially the product of three signals: $I_x^{(gx)}$ is a signal and I^*_{aerr} is composed of sums of products of two signals: derivatives of the image and derivatives of the flow components. The width of the spectrum of the product of two real signals is more or less the sum of the widths of the factors of the product. To see this consider an image that is just a cosine $\cos\omega_1 x$. If we multiply it by another image that is also a cosine $\cos\omega_2 x$, then the result will contain the $\cos(\omega_1+\omega_2)$. So the spectrum of the result will be wider than each of the components.

If, on the other hand we use Gabor filters the bandwidth could even decrease. A signal that has gone through convolution with a Gabor function

$$e^{j(k_x x + k_y y) + \frac{x^2+y^2}{2\sigma^2}}$$

can be written in the following form

$$s_1 = e^{j(k_x x + k_y y)} f_1(x,y)$$

where f_1 is a bandlimited signal. If we multiply two signals $s_1 s_2^*$ the complex powers of e vanish and we get $f_1 f_2^*$ which with the proper choice of the Gabor parameters can have a narrower band than the original signals s if $\sigma|k| \geq 3$.

The Eq. (3) is a linear equation where unknowns are the components of u avd v. We used the CG (Conjugate Gradient) method to solve this linear system. Among the several advantages of CG is that it can be implemented using only high level image operations which can be done by overloading the multiplication and dot product operations. We used as preconditioner the 2×2 block diagonal matrices. The implementation was done on MediaMath, a system for interactive image manipulation and algorithm prototyping and the whole program was about 200 lines of MediaMath code [13].

This flow method was tested on several moving image sequences and the results were very good [12]. The formalization of the method is flexible, so several ideas of other algorithms can be included. In Fig. 1, we

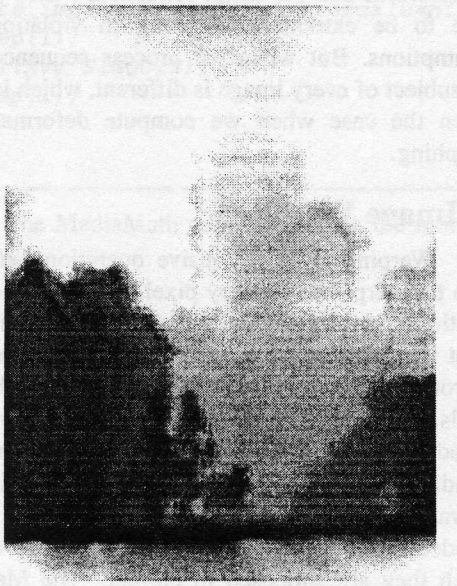

Fig. 1 The picture show the inverse of the magnitude of the flow, which is proportional to the depth map because the camera translates sideways. The flow varies between 20 and 30 pixels and the original images are shown in Fig. 2.

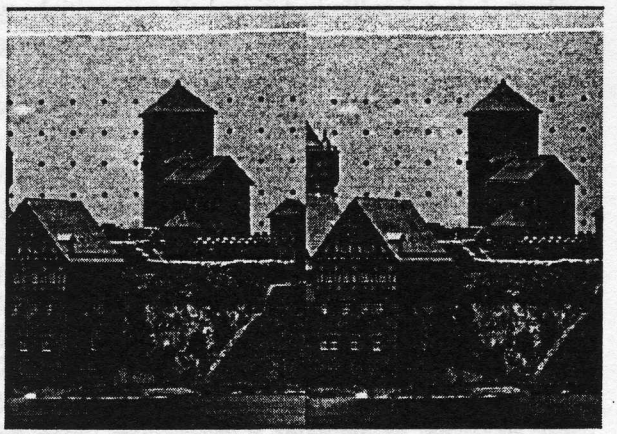

Fig. 2. The two images of the castle sequence (numbers 3 and 4 from the database of CIL at CMU) cropped.

show the results on two images from the CIL database at CMU. The algorithm that produced these images used a hierarchical approach similar to [1], a regularization factor with a small λ, and sophisticated derivatives [9]. The σ of the Gabor functions were rather small, which works best for image sequences where we do not have to be extremely forgiving to violation of the assumptions. But when we process sequences where the subject of every image is different, which is by definition the case when we compute deformations for morphing.

4. Image Warping

Warping is an expensive operation because we have to interpolate at every pixel. We used linear interpolation which proved sufficient for the quality of the input images. The warping is done in two stages: first we round the displacement vectors and warp by whole pixels. This is a very cheap operation. Then we correct by adding the derivatives times the residual of the rounding doing in effect linear interpolation. The derivatives have to be themselves warped by the rounded flow as well so that they follow the pixel which they are supposed to correct. The MediaMath [13] code is shown in Fig. 3.*

It is easy to see that this can be extended to second order interpolation if the results are not satisfactory by computing the second order derivatives and multiplying them by second degree monomials of du and dv. The results using the second order interpolation were only slightly better than the linear one and third order were virtually indistinguishable from the second order.

*MediaMath syntax is very similar to C

5. Animating the morphing

To produce the morphing we need the displacements from image one to image two and from image two to image one. Then we calculate the intermediate images deforming both images towards each other, so that they "meet" in between and we take a weighted average (cross dissolve). By varying the weights and the amount of deformation we can create a smooth transition from one image to the next. The MediaMath code is shown in Fig. 4.

6. Results

We used the algorithm to morph the images of various people. The image sequences could be viewed using MediaMath and xv (about one image every three seconds) or on an SGI using moviemaker/movieplayer. Nine of the frames are shown on Fig. 5. For comparison we show the result of the fifth frame using Horn and Schunck algorithm [8] in Fig. 6. The results with the Gabor-affine algorithm are much better.

7. Conclusions

We presented an approach to morphing that does not require a human operator by using flow techniques from computer vision. We designed a flow algorithm that is forgiving to gross violations of the intensity constancy assumption and does not suffer from problems with derivatives. It worked well for images that have similar features but it performed poorly on images that did not (e.g. one of the two faces had glasses). We plan to extend the approach to moving images and introduce a simple way for the operator to guide the flow algorithm in cases that the images have parts that are extremely different. Since the computation of the deformation is much easier and accurate between images of the same sequence the user guidance in one frame could be transfered to the next frames.

References

1. P. Anandan, "A Computational Framework and an Algorithm for the Measurement of Visual Motion," *Intl' J. of Computer Vision* **2** pp. 283-310 (1989).

```
function linwarp_fimg(im,u,v)
"Warps the image using linear interpol."
{
  local im12,im12_x,im12_y,ru,rv,du,dv;
        /* Integer warp the image */
  im12   = intwarp_fimg(im,ru=round_fimg(u),
                        rv=round_fimg(v));
        /* and its derivatives     */
  im12_x = intwarp_fimg(D_x(im),ru,rv);
  im12_y = intwarp_fimg(D_y(im),ru,rv);
  du     = u - ru;
  dv     = v - rv;
        /* add the derivative times the */
        /* residual of the rounding */
  im12  += im12_x*du;
  im12  += im12_y*dv;
  im12;                 /* return im12 */
};
```

Fig. 3. Mediamath code for warping using linear interpolation.

```
uv=Gb_A_flow(im1,im2,:sig=0.0, :lam=10.0,
                    :maxits=10,
                    :solver='ConGrad,
                  :scls = 3,
                  :scl1 = 0.7,
                  :w_s_prod = 3.0,
                  :oriens=4);

uv_i=Gb_A_flow(im2,im1,:sig=0.0, :lam=10.0,
                  ...
                  :oriens=4);

xv = spawn("xv", "morph");

for (i=1; i<=10; i++)
{
  local ru, rv, du, dv, uv_s, uv_si, temp;

  uv_s = uv*(i/10.0);
  uv_si= uv_i*((10-i)/10.0);

  im12=linwarp_fimg(im2,uv_s->u,uv_s->v);
  im12 *= (10-i)/10.0;
  temp=linwarp_fimg(im1,uv_si->u,uv_si->v);
  temp *= i/10.0;
  im12 += temp;

  write_img(im12, "morph",:rescale=nil);
  sleep(1);
  kill(xv, SIGQUIT);
  sleep(1);
};
```

Fig. 4. The MediaMath code to animate the morphing using *xv*.

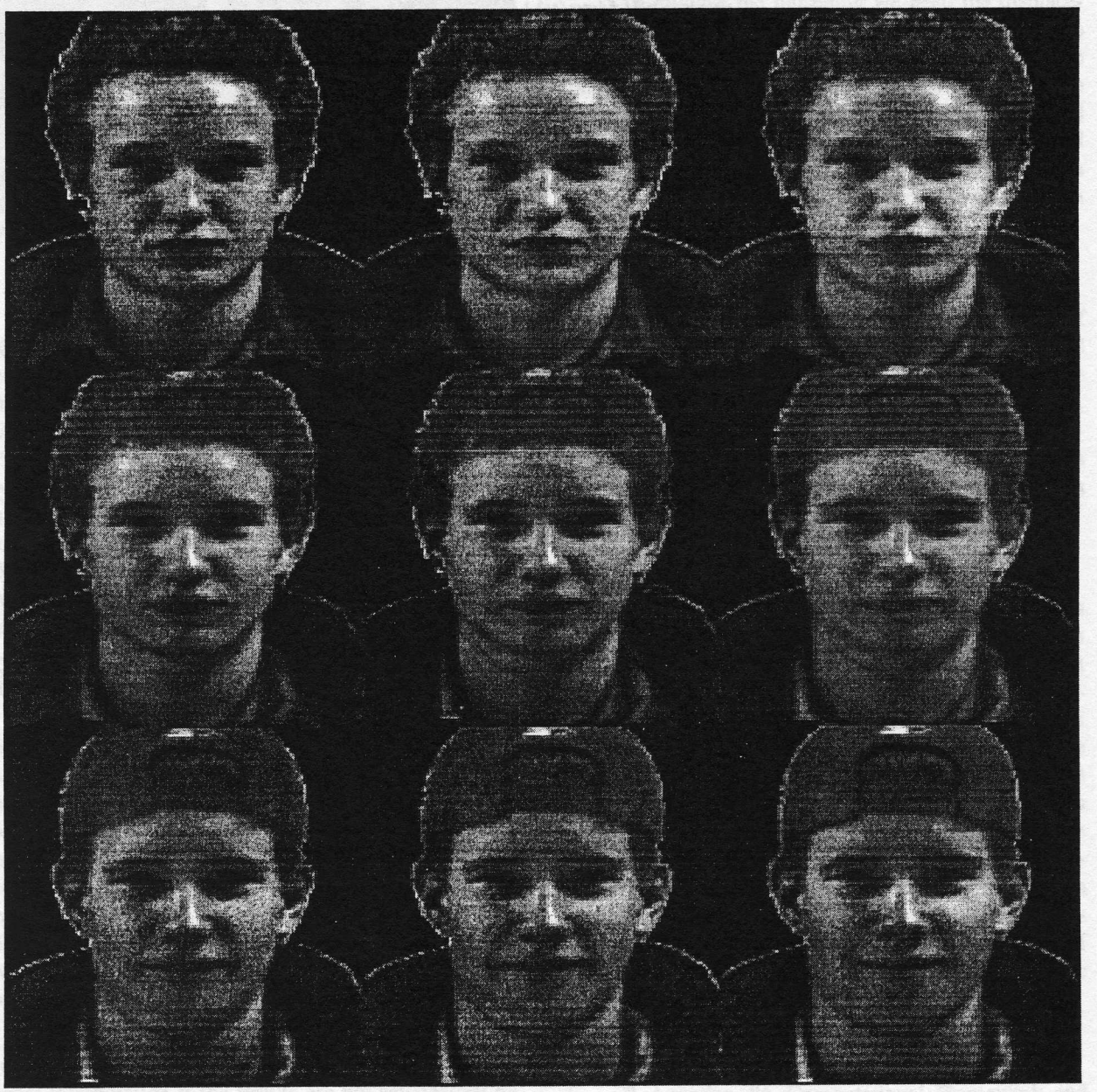

Fig. 5. Nine images of one young male morphing into another.

Fig 6. The fifth image using Horn and Schunck algorithm.

2. J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of Optical Flow Techniques," *Int'l Journal of Computer Vision* **12** pp. 43-77 (1994).
3. T. Beier and S. Neely, "Feature Based Image Metamorphosis," *SIGRAPH*, pp. 35-42 (1992).
4. M.J. Black and P. Anandan, "A framework for the robust estimation of optical flow," *ICCV*, pp. 231-236 (1993).
5. M. Campani and A. Verri, "Computing Optical Flow from an Overconstraint System of Linear Equations," *CVPR*, pp. 22-26 (1990).
6. Hsiao Jing Chen, Yoshiaki Shirai, and Minoru Asada, "Obtaining optical flow with multi orientation filters.," *CVPR*, (1993).
7. D. J. Fleet and A. D. Jepson, "Computation of component image velocity local phase information," *Intl' Journal of Computer Vision* **5** pp. 77-104 (1990).
8. B. K. P. Horn and B. G. Schunck, "Determining Optical Flow," *Artificial Intelligence* **17** pp. 185-204 (1981).
9. E. Karabassis and M. E. Spetsakis, "An Analysis of Image Interpolation Differentiation and Reduction using Local Polynomial Fits," *CVGIP: GMIP (accepted for publication)*, (1992).
10. B. Lucas, *Generalized Image Matching by the Method of Differences*, PhD Dissertation, Dept. of Computer Science, Carnegie Mellon University (1984).
11. F. Meyer and P. Bouthemy, "Region based matching in an image sequence," *ECCV*, pp. 476-484 (1992).
12. M. E. Spetsakis, *An optical flow estimation algorithm that uses Gabor filters and affine model for flow*, York TR CS-94-06 (1994).
13. Minas Spetsakis, "MediaMath: A reasearch environment for vision research," *Vision Interface*, pp. 118-126 (1994).
14. A. Verri, M. Straforini, and V. Torre, "Computational aspects of motion perception in natural and artificial vision systems," *Phil. Trans. R. Soc. Lond.* **B 337** pp. 429-443 (1992).
15. P. Werkhoven and J. J. Koenderink, "Extraction of Motion Parallax Structure in the Visual System," *Biological Cybernetics* **63** pp. 193-199 (1990).
16. G. Wolberg, *Digital Image Warping*, IEEE Computer Society Press Monograph (1990).