THE JOURNAL OF VISUALIZATION AND COMPUTER ANIMATION, VOL. 3: 31–43 (1992)

Choreographed Image Flow

KARL SIMS
Thinking Machines Corporation, 245 First St., Cambridge, MA 01242, U.S.A. and
Digital Animation Laboratory, 725 N. Highland, Hollywood,CA 90038, U.S.A.

SUMMARY

Choreographed fluid-like motions have been created and used to animate two dimensional images. Techniques are presented for generating these animations by repeatedly warping images using vector field representations of arbitrary flowing motions. Tools that allow interactively creating and aligning vector flow-fields with images are also described. These methods lend themselves well to highly parallel computation, and have been implemented using the Connection Machine System.

1. INTRODUCTION

Many techniques have been developed for warping and distorting images for various purposes in areas such as image processing, computer vision, medical imaging and computer graphics. Here, *warping* will mean a resampling or mapping of a 2D source image onto a 2D destination image. Texture mapping is a common computer graphics use of warping in which images are projected onto 3D surfaces and then onto the image plane.[1,20] Developments have been made towards increasing the efficiency of certain kinds of warping transformations, such as those used for texture mapping.[2–6]

The techniques presented here include uses of arbitrary warping transformations and repeated warping of images to generate frames of animation.

A simulation of the planet Jupiter in the film *2010* contained an example of animation that could be considered as repeated image warping. A 2D particle system was used to model fluid motion and warp an image of Jupiter's surface. The initial states of the particles were set by vortex descriptions, fluid dynamics modelling was used to flow the particles, and the surface was remapped by re-rendering the moved particles.[7]

Oriented patterns have been analysed to automatically create flow fields that align with the directions of anisotropy in the patterns.[8] These flow fields can then be used to manipulate the original image.[9]

The warping motions described in this paper are completely choreographed. The animator has control over the synthesis of each detail of motion. No automatic analysis of the image to be animated is performed. Motions can be aligned to the patterns, or not, as desired by the choreographer. No dynamics modelling is used, but dynamically correct looking motions can still result with careful design. Motions are represented in two dimensions, but three-dimensional looking motion can still

1049–8907/92/010031–14$12.00

Received 3 July 1991
Revised October 1991

be created. Although the tools used here to flow and warp images have some characteristics of particle systems,[10] they are probably more appropriately classified as image processing techniques. They have been used to animate a selection of Leonardo da Vinci's deluge drawings (Figure 1).

In this paper, capital letters will be used to represent 2D arrays such as images or vector fields and small letters will be used for single scalars or vectors. Bold type will be used to represent vectors and plain type will be used for scalars. For example, $\mathbf{X}$ might be an image of colour vectors, X an image of single grey values, $\mathbf{x}$ a vector and x a scalar. Subscripts will be used to represent indexed elements of arrays and co-ordinate pairs will usually be concisely represented as co-ordinate vectors. For example, if $\mathbf{x} = [u, v]$ is a co-ordinate vector, then $\mathbf{V}_{\mathbf{x}}$ would be the pixel of the image, $\mathbf{V}$, at $[u, v]$. A bold zero, $\mathbf{0}$, is used to mean a vector field with all zero elements.

1.1. Forward and inverse mapping

An image warping operation can be specified by defining the relationship between the source and destination image co-ordinates. Forward mapping defines the destination co-ordinates as a function of the source co-ordinates. Inverse mapping defines the source co-ordinates as a function of the destination co-ordinates.

If the vector $\mathbf{x} = [u, v]$ denotes a pair of co-ordinates in the source image domain, and $F(\mathbf{x})$ is a forward co-ordinate mapping function which transforms the input source co-ordinates $\mathbf{x}$ into the output destination co-ordinates $\mathbf{x}'$,

$$\mathbf{x}' = F(\mathbf{x})$$

then the warping operation from a source n-dimensional image vector, $\mathbf{V}$, to a destination image vector, $\mathbf{V}'$, can be described as:

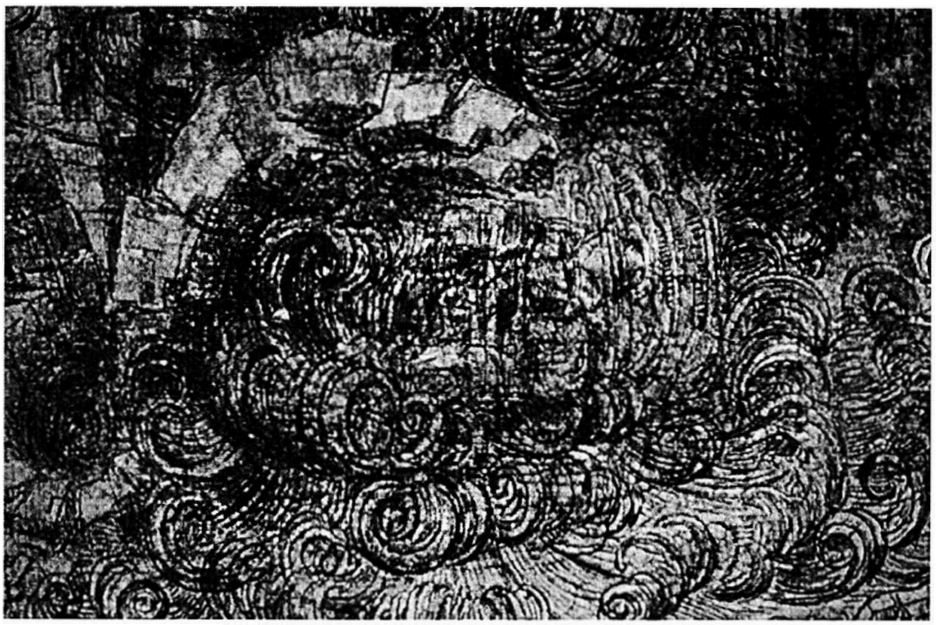

Figure 1. A deluge drawing of Leonardo da Vinci

for each source element $\mathbf{x}$:
$\mathbf{V}'_{F(\mathbf{x})} = \mathbf{V}_{\mathbf{x}}$

In this simple implementation, the destination image will probably have serious aliasing problems. If the source image is expanding, there will be some holes in the destination image from unset destination elements. If it is contracting, destination elements may be set by more than one source element. Quantization of the destination co-ordinates will also contribute to aliasing. Some solutions to these problems will be discussed below. An arrow will be used to imply that the elements of $\mathbf{V}'$ at all destination co-ordinates are somehow set and filtered.

$$\mathbf{V}'_{F(\mathbf{x})} \leftarrow \mathbf{V}_{\mathbf{x}}$$

If $G(\mathbf{x})$ is an inverse co-ordinate mapping function which transforms destination co-ordinates, $\mathbf{x}'$, into source co-ordinates, $\mathbf{x}$,

$$\mathbf{x} = G(\mathbf{x}')$$

then the warping operation can be performed directly with unique source co-ordinates calculated for each of the destination co-ordinates:

for each destination element $\mathbf{x}'$:
$\mathbf{V}'_{\mathbf{x}'} = \mathbf{V}_{G(\mathbf{x}')}$

Aliasing problems can also occur with inverse mapping. If the source image is contracting, the destination element should receive an average over an area of the source image instead of a single element. If the source image is expanding, the destination should receive a value by interpolating among at least four source element neighbours. Texture mapping is a common example of inverse mapping. Summed area tables,[11] mip mapping [12], and other techniques,[13–14] have been used extensively as successful solutions to these inverse mapping aliasing problems.

Some forward mapping functions are separable into a series of one dimensional transformations. This can permit the warping computation to be performed with increased efficiency.[2–4] Separability will not be considered in this paper because the warping functions used will be arbitrary and not necessarily possible to separate.

Section 2 of this paper presents a forward mapping function to perform arbitrary image warping. Section 3 introduces an operation to alter vector fields with an inverse mapping component. Section 4 addresses methods for the creation of complex fluid-like motions. Some techniques for visualizing and constructing vector fields will be described in section 5. And finally, data parallel implementation, results, and future extensions of this work will be discussed.

2. IMAGE WARPING

Image warping with arbitrary forward mapping functions has been implemented using a lookup array. The lookup array is a vector field containing the relative displacements for each element in the source image. If the vector field describing the warp to be performed is $\mathbf{W} = [W^u, W^v]$, then the forward co-ordinate mapping function is

$$\mathbf{x}' = \mathbf{x} + \mathbf{W}_{\mathbf{x}}$$

and a destination image, $\mathbf{V}'$, can be set from a source image, $\mathbf{V}$, by:

for each source element $\mathbf{x}$:
$\mathbf{V}'_{(\mathbf{x} + \mathbf{w}_{\mathbf{x}})} \leftarrow \mathbf{V}_{\mathbf{x}}$

A function *warp* will be a useful definition. It will perform this warping operation above such that

$$\mathbf{V}' = warp(\mathbf{V},\mathbf{W})$$

Since *warp* can remap any image of n-vectors, $\mathbf{V}$, it can be applied to grey-scale images, 3-element colour images, colour images with mattes, or even other 2-element vector fields.

As previously mentioned, forward mapping can result in aliasing problems from expansions, contractions, and quantization of destination co-ordinates. *Warp* performs anti-aliasing using a weighted average to handle quantizations and contracted regions, followed by a diffusion to fill in holes from expanded regions.

First, each source pixel calculates its destination coordinates $\mathbf{x}'$ at subpixel accuracy, using the element, $\mathbf{W}_{\mathbf{x}}$, of the warp vector field of relative displacements. The four neighbouring destination pixels nearest to $\mathbf{x}'$ are determined and weights equal to one minus the distances of $\mathbf{x}'$ to each of them are calculated.

Then, each source pixel sends its value to each of the four destination pixels premultiplied by the corresponding weight. The destination pixels sum these products received from some number of source pixels. The four weights themselves are also sent to the destination pixels such that each destination also receives a sum of all of the weights sent to it. The destination pixels then divide their total weighted value sums by their total weight sums to get final values containing weighted contributions from any source samples mapped nearby. This is similar to resampling an image using a triangle filter.

This forward mapping technique deals well with contractions in the vector field because arbitrarily large areas can be appropriately averaged down into smaller regions. However, if the vector field contains expansions or if the border of the image is pulled back, some destination elements may not receive any values or weights. A diffusion process is performed to spread the values to destination elements that received few or no weighted values from those that did until every element has a value that is an appropriate weighted average of mapped source values.

For each diffusion iteration, each pixel adjusts its weight sum and its weighted value sum using averages of those quantities from its four adjacent neighbours. This is an iterative blur in which pixels force their values and weights on each iteration to be a percentage of their original values and weights, where the percentage equals their original weights. When the new weight sum of a pixel is sufficiently large, that pixel stops being affected by diffused values from its neighbours. The diffusion stops when all the new weights have reached sufficient values. Finally, the diffused weighted values are divided by the diffused weights to give the final warped values.

A potential disadvantage of this diffusion method is that if an image is being expanded anamorphically, the diffusion will still spread values at equal speeds in

each direction. This could cause some inappropriate blurring. Interpolating the source image and the vector field to generate a small grid of samples (2×2 or 3×3) for each pixel can alleviate this problem if necessary with some additional cost in computation time. An adaptive approach might be preferable, where the local number of necessary samples is determined by first estimating the amount of local expansion in the vector field.

In an attempt to create a series of images for an animation sequence we might repeatedly warp an image, $\mathbf{I}^0$, by the same flow-field, $\mathbf{W}$, containing a small amount of movement. If the images $\mathbf{I}^n$, $[n = 0, 1, 2, 3. . .]$, are the successive frames of an animation:

$$\mathbf{I}^0 = image$$
$$\mathbf{W} = flow\ field$$
$$\mathbf{I}^1 = warp(\mathbf{I}^0, \mathbf{W})$$
$$\mathbf{I}^2 = warp(\mathbf{I}^1, \mathbf{W})$$
$$\mathbf{I}^3 = warp(\mathbf{I}^2, \mathbf{W})$$

This might look interesting for the first few frames, but each *warp* would cause irreversible blurring due to resampling and filtering. Since each frame depends on the previous, this is not acceptable.

3. IMAGE FLOWING

This section presents a method for repeatedly rewarping an image with significantly less degradation than shown above. Instead of each frame, $\mathbf{I}^n$, depending on a *warp* from the previous frame, $\mathbf{I}^{n-1}$, the effects of multiple *warps* are preserved at sub-pixel accuracy in a vector field so the frames can all be calculated by warping only the original image, $\mathbf{I}^0$.

A function *flow* will help achieve this. It increments a vector field by the value of a second vector field at the relative location of the vectors in the first vector field.

If $\mathbf{W} = [W^u, W^v]$ is now a source 2D vector field, $\mathbf{W}'$ is a destination 2D vector field, and $\mathbf{F} = [F^u, F^v]$ is a 2D vector flow-field, then

$$\mathbf{W}' = flow\,(\mathbf{W}, \mathbf{F})$$

where *flow* performs the following operation:

for each destination element $\mathbf{x}$:
$$\mathbf{W}'_{\mathbf{x}} = \mathbf{W}_{\mathbf{x}} + \mathbf{F}_{(\mathbf{x} + \mathbf{w}_{\mathbf{x}})}$$

If the flow-field, $\mathbf{F}$, is used to warp an image, $\mathbf{I}^0$, three successive times:

$$\mathbf{I}^3 = warp(warp(warp(\mathbf{I}^0,\mathbf{F}),\mathbf{F}),\mathbf{F})$$

the same result with much less blurring can be achieved with three *flow* operations followed by a single *warp*:

$$\mathbf{I}^3 = warp(\mathbf{I}^0, flow(flow(flow(\mathbf{0},\mathbf{F}),\mathbf{F}),\mathbf{F}))$$

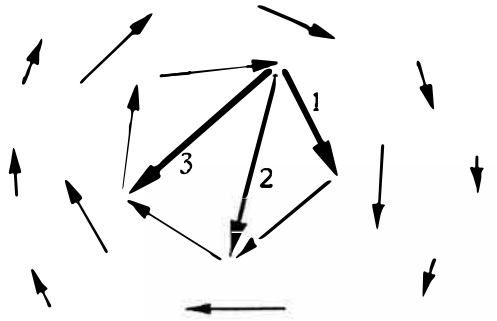

Figure 2.

The bold arrows labelled 1, 2, and 3 in Figure 2 represent the successive contents of a single element of the vector field, **W**, being flowed three times by the vector field, **F**, represented by the plain arrows. Each flow operation pushes the co-ordinate elements of **W** one step further along in the patterns described by **F**. Note that *warp*(**X**,**0**) = **X**, and *flow*(**0**,**F**) = **F**.

The flow calculation has an inverse co-ordinate mapping component. Like *warp*, this is an arbitrary mapping defined by the relative displacement elements in the provided vector field, but unlike *warp*, it is an inverse mapping instead of a forward mapping, and it causes a vector field to be incremented using a second vector field. It is performed as follows.

First, each destination element, $\mathbf{W}_x$, calculates the source vector co-ordinates, $\mathbf{x} + \mathbf{W}_x$ at sub-pixel accuracy. Values from the four neighbouring elements of the source vector field, **F**, nearest these co-ordinates are retrieved. Bi-linear interpolation between these gives the final value, $\mathbf{F}_{(x + W_x)}$which is then added to the first vector argument for its new value $\mathbf{W}'_x$ (see Figure 3).

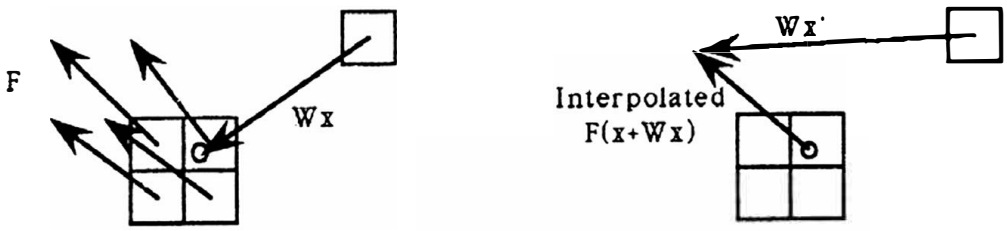

Figure 3.

Some of the inverse mapping aliasing problems discussed above could still occur using this procedure. Techniques to retrieve an average over an area of the source vector field have not been implemented because the vector fields used did not contain many discontinuities or high frequencies. Similar to the technique suggested above for *warp*, interpolation between vector field elements to generate and average multiple samples could be performed if necessary to alleviate this problem.

Combining the use of the *flow* and *warp* operations will allow the frames of an animation to be calculated as before. *Warp* will always be performed on the original image but with an incrementally updated warping vector field:

$\mathbf{I}^0$ = *image*
$\mathbf{F}$ = *flowfield*
$\mathbf{W}$ = $\mathbf{0}$

$\mathbf{W}$ = *flow*($\mathbf{W}$,$\mathbf{F}$)
$\mathbf{I}^1$ = *warp*($\mathbf{I}^0$,$\mathbf{W}$)

$\mathbf{W}$ = *flow*($\mathbf{W}$,$\mathbf{F}$)
$\mathbf{I}^2$ = *warp*($\mathbf{I}^0$,$\mathbf{W}$)

$\mathbf{W}$ = *flow*($\mathbf{W}$,$\mathbf{F}$)
$\mathbf{I}^3$ = *warp*($\mathbf{I}^0$,$\mathbf{W}$)

4. SECOND ORDER FLOWING

More complex types of motions can be created than by simply rewarping an image by a single flow-field. A useful variation is to also put the flow-field into motion by using a second-order flow-field.

A simple, quickly degrading version of second order flowing using only *warp* would be as follows:

$\mathbf{I}^0$ = *image*
$\mathbf{F}$ = *flowfield*
$\mathbf{F}^2$ = *2nd order flowfield*

For each frame *n*:

$\mathbf{F}$ = *warp*($\mathbf{F}$,$\mathbf{F}^2$)
$\mathbf{I}^n$ = *warp*($\mathbf{I}^{n-1}$,$\mathbf{F}$)

A higher quality version using both *flow* and *warp* would be

$\mathbf{I}^0$ = *image*
$\mathbf{F}$ = *flowfield*
$\mathbf{F}^2$ = *2nd order flowfield*
$\mathbf{W}$ = $\mathbf{0}$
$\mathbf{W}^2$ = $\mathbf{0}$

For each frame *n*:

$\mathbf{W}^2$ = *flow*($\mathbf{W}^2$,$\mathbf{F}^2$)
$\mathbf{W}$ = *flow*($\mathbf{W}$,*warp*($\mathbf{F}$,$\mathbf{W}^2$))
$\mathbf{I}^n$ = *warp*($\mathbf{I}^0$,$\mathbf{W}$)

$\mathbf{W}^2$ is repeatedly updated to contain the cumulative effect of $\mathbf{F}^2$, which warps the elements of $\mathbf{F}$, while $\mathbf{W}$ is updated to contain the cumulative effect of $\mathbf{F}$, which in turn warps $\mathbf{I}^0$, such that each new frame can still be warped from the initial image.

A variation of 2nd order flowing is achieved by adding $\mathbf{F}^2$ to the warped $\mathbf{F}$ before updating $\mathbf{W}$:

For each frame n:

$$\mathbf{W}^2 = flow(\mathbf{W}^2,\mathbf{F}^2)$$
$$\mathbf{W} = flow(\mathbf{W},warp(\mathbf{F},\mathbf{W}^2) + \mathbf{F}^2)$$
$$\mathbf{I}^n = warp(\mathbf{I}^0,\mathbf{W})$$

This causes the second order flow to move the image as well as the first order flow. Without adding $\mathbf{F}^2$, the first order flow will move through the image without the second order flow also pulling the image along.

Many other possibilities for complex types of motions exist. Layers of flowing motion can be composited together by flowing images with matte channels. Flow-fields can be scaled to vary the speed of motion.

5. FLOW-FIELD CONSTRUCTION

In the previous sections, the use of flow-fields to warp and animate images has been described. In this section the interactive creation of flow-fields that can be used to give desirable motion will be addressed.

The contents of a constructed flow-field should align well with the image that it will warp. Large motions over the entire image should be possible as well as small detailed motions. Ideally, the interface for flow-field manipulation should be intuitive, visual, and efficient, such that complex motions can be created on a practical time scale. An attempt to achieve these goals is described.

Visualization of a flow-field is an important component of this flow-field construction system. It can be achieved in two ways. First, the magnitude of the flow-field elements can be displayed as a monochromatic image. This allows the other colour channels of a full colour display to be used for other information. Plate 1 shows the original image displayed in blue with the flow-field magnitudes aligned and displayed in green. The red channel remains available for interactive use.

Secondly, the elements of the flow-field can be transformed into a full colour image and displayed communicating the flow-field direction as well as its magnitude. The vector direction of the flow-field is used to calculate the hue and the vector magnitude controls the brightness, as shown in Plate 2.

The interactive process for building a flow-field is as follows: First, a flow-field to be constructed is created and initialized, either to $\mathbf{0}$, or by copying the contents from an existing flow-field.

An *active region* is defined using a paint system. A soft-edged brush paints over a region where the flow-field will be altered. The *active region* is essentially a matte containing percentages for each pixel of the image or each vector of the flow-field. The active region is set from a single colour-channel of the painted image. The other two colour channels can contain the image to be warped, or an image representing the partially constructed flow-field as described above, so the *active region* can be carefully aligned to the image or to the flow-field as shown in Figure 4.

Three operations, *translate, expand* and *vortex*, are used to alter the flow-field using the active region and various parameters. They first create a temporary flow-field using specified parameters, then matte it onto the flow-field being constructed using the active region.

Translate simply creates a region of motion in a specified constant direction.

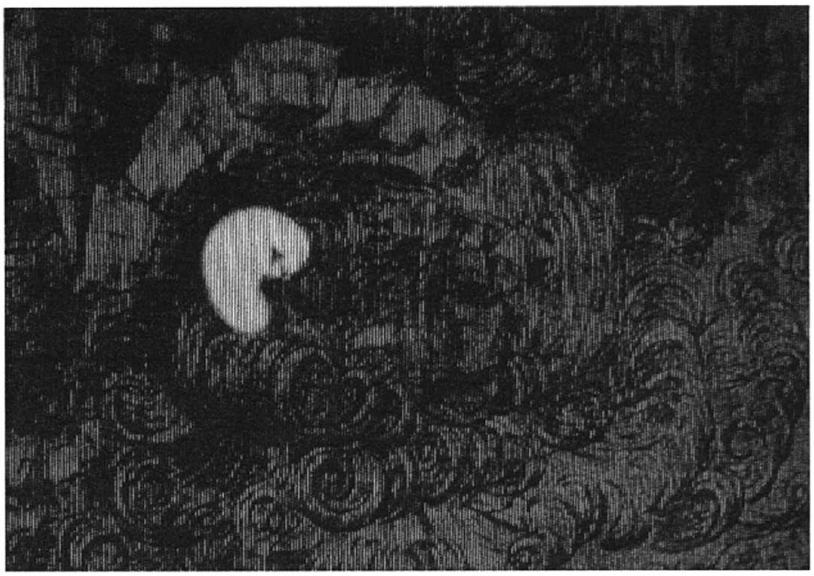

Figure 4. An active region *is defined by painting an area*

Expand creates a region of expanding motion by scaling outward from a specified centre. The centre of reference can be entered numerically, pointed to with a mouse (Figure 4), or automatically calculated as the centroid of the region. A contraction can be created using the *expand* operation with a negative scale parameter (see Figure 5).

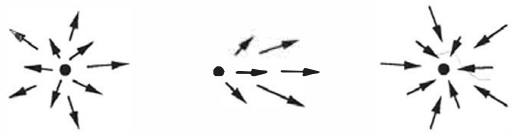

Figure 5. Expansions and contractions

The *vortex* operation creates a region of vortex motion using the parameters *centre, magnitude* and *tightness*. The centre can be specified with the same options as *expand*. An angle θ is calculated from the distance, *r*, between each element and the vortex centre.

$$\theta = \frac{magnitude}{r^{tightness}}$$

The *tightness* parameter usually varies between 1·0 and 2·0. The final vectors are calculated by rotating each element's co-ordinates about the centre by the angle θ, and then subtracting the previous value. An optional *stretch* parameter allows non-uniformly scaling the vortex motion to create elliptical vortices and can help achieve 3D looking motions (see Figure 6). A problem that can occur in vortex motions, especially if the flow-field is later scaled to alter the speed of motion, is that the

Figure 6. Vortices

vortices may slightly expand or contract at the centres due to the quantization of an ideally continuous rotation into translational segments.

The function that composites these motions onto the flow-field can have some variation. The matte of the active region can be scaled by different amounts for weighted averaging between the previous flow-field and new flow-fields created with the operations above. Since the matte is painted with soft-edged brushes smooth transitions between different regions in the resulting vector field usually occur. The active region matte can optionally be blurred to cause even smoother transitions if necessary.

Another utility is used to create multiple regions of similar motion more easily. Multiple separate regions are painted at once with a soft-edged brush. A simple connected component labeling technique is used to find each isolated region. One at a time, each separate region is treated as the *active region*, and a *vortex* or *expand* operation is performed using it with its centroid as the centre of reference (see Figure 7). This allows faster creation of complex motion with many similar details, such as clouds with multiple expanding regions, or water with multiple vortices.

Some other techniques for achieving 'painting' of flow-fields were attempted but were not successful. One method tried was directly painting a colour representation of a flow-field, as is used to visualise it, and then transforming that into the flow-field, but this was not successful because creating smooth washes between colours

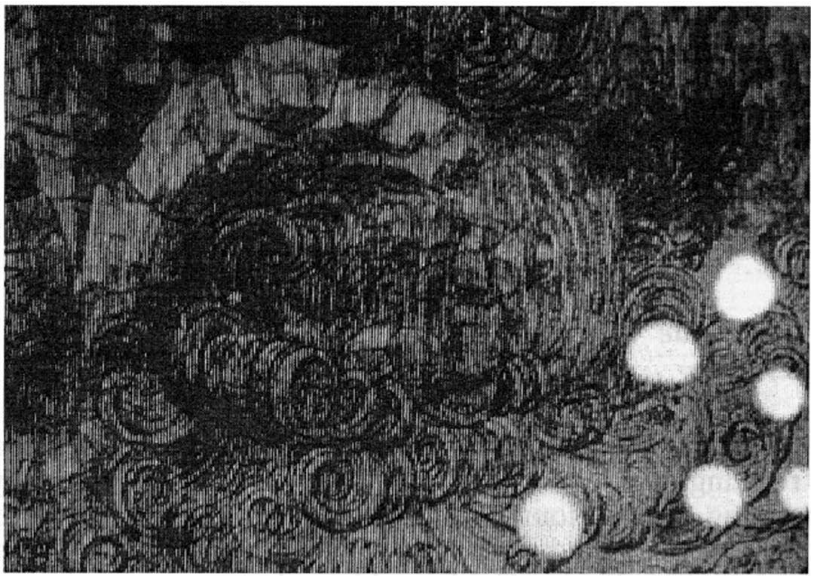

Figure 7. Multiple regions are painted to be similarly processed

in arbitrary shapes was time consuming and not intuitive. Setting the flow-field using the brush velocity was also attempted but was not used because it was difficult to accurately and smoothly control the brush velocity with the paint system available.

Although it might still be useful to extend the library of utilities beyond those mentioned here, they have been used to successfully create a wide variety of motions.

6. PARALLEL IMPLEMENTATION

These techniques have been implemented using a Connection Machine System (CM-2), a data parallel supercomputer consisting of between 4K and 64K processors.[15–17] Special routeing hardware allows general communication between processors. A virtual processor mechanism is used to simulate more processors than the physical number so the virtual machine size can vary depending on the number of data elements in the application. For example, a 512 × 512 image in an 8K processor machine would require a *virtual processor set* with 32 virtual processors per processor to contain the image, with one pixel in each virtual processor.

The images and vector fields described above are processed in parallel in a single 2D virtual processor set with dimensions equal to the image dimensions. Each virtual processor contains one element of every image and vector field in use, such that the elements in different images or vector fields with equal indices are in the same virtual processor.

Three primitive interprocessor communications are used for choreographed image flowing: *send, get,* and *news. Send* delivers a value, from each selected processor, to a specified destination processor-address. *Get* retrieves a value, for each selected processor, from a specified source processor-address. *Send* can also combine values sent to equal destinations in various ways. *Send with add* causes the values sent to equal destinations to be summed as they are delivered. The forward mapping *warp* function is implemented in parallel using *send with add*, and the inverse mapping *flow* function is implemented using *get. News* (North, East, West, South) performs nearest neighbour grid communications and is used for the diffusion component of the *warp* function.

Images of 256 × 256 resolution can be processed and tested using the *flow* and *warp* functions on the Connection Machine system at near real time rates.

7. RESULTS

These tools have been used to create animations of fluid-like motions applied to the deluge drawings of Leonardo da Vinci (Figure 1).[18] The methods using second order flow-fields described above were applied for these animations. A first-order flow-field was constructed which usually contained the smaller details of motion such as vortices and slight expansions and contractions. A second-order flow-field was constructed usually with motions on a larger scale. This allowed hierarchical motions to be created such as vortices of vortices.

Fluid motions of wind and water were created using many vortices and a second order flow-field to flow the vortices.

Boiling cloud motions were created using multiple small expansions and contractions, and a second order flow containing some larger elliptical vortices, expansions, and contractions.

Although accumulating the effect of successive warping with *flow* helps maintain image quality, eventually the image becomes distorted well beyond the original. Expanding areas become blurry and vortices get twisted up. A useful trick to generate longer sequences of continuous animation is to flow the image both forwards and backwards. A forward sequence is generated by flowing an image until it becomes distorted beyond acceptability. Then, a backwards sequence is generated by negating the flow-fields before proceeding. The two sequences are spliced together, back to back, with the second sequence reversed, so that the original image is in the centre, giving twice as much undistorted duration.

8. FUTURE WORK AND CONCLUSION

There are many possible extensions to the utilities described in this paper. An elaborate paint system specifically designed for 'painting' 2D flow-fields instead of colours could be developed. Instead of just painting the *active regions* and performing operations on them, complex brushes of different shapes and sizes with various parameters might be used to create flowing regions of different types. The ability to copy and transform regions of flow to other regions would also be useful.

One might also imagine vector fields that cause different kinds of effects beyond the second order flowing described, such as an acceleration field, which, instead of warping a velocity-field, would cause its elements to be incremented.

A fluid dynamics simulation could be incorporated to generate flow-fields automatically and help generate more realistic 2D fluid motions.

Extending this work to three dimensions would be an ambitious project because of the vast amount of memory required to store 3D vector fields of significant resolution, but the same methods used above for warping 2D images could be used for 3D volume warping. 3D flowing motions created with this system might also be applied to flexible 3D objects.[19]

In conclusion, techniques have been presented for the interactive creation of arbitrary flow-fields that can be used to generate animation from repeated image warping. The functions *warp* and *flow* allow repeated warpings of an image to be performed with a minimum of quality degradation. The processes used are easily parallelizable and have been implemented on the Connection Machine system. As data parallel computation becomes more affordable and available, techniques such as these will hopefully become more practical and useful.

ACKNOWLEDGEMENTS

Thanks to Mark Whitney for providing an application that motivated the existence of this system. Thanks to John Whitney Jr. and Digital Animation Laboratory for the environment to produce this work, and for permitting it to be published. Thanks to John Ornelas for being a successful user of this system. And thanks to Lew Tucker, Guy Steele, Jim Salem, Dave Sheppard, Gary Oberbrunner, and Thinking Machines Corporation, for support and encouragement.

REFERENCES

1. J. Blinn and M. Newell, 'Texture and reflection on computer-generated images', *Communications of the ACM*, **19**, (10), 542–547 (1976).
2. E. Catmull and A. R. Smith, '3-D transformations of images in scanline order', *Computer Graphics*, **14**, (3), 279 (1980).

3. A. R. Smith, 'Planar 2-pass texture mapping and warping', *Computer Graphics,* **21**, (4), 263 (1987).
4. G. Wolberg, 'Separable image warping with spatial lookup tables', *Computer Graphics,* **23**, (3), 369 (1989).
5. K. M. Fant, 'A nonaliasing, real-time spatial transform technique', *IEEE Computer Graphics and Applications,* **6**, (1), 71–80 (1986).
6. A. Paeth, 'A fast algorithm for general raster rotation', *Graphics Interface '86 Proceedings*, May 1986, pp. 77–81.
7. L. Yaeger and C. Upson, 'Combining physical and visual simulation—creation of the planet Jupiter for the film 2010', *Computer Graphics,* **20**, (4), 85–93 (1986).
8. M. Kass and A. Witken, 'Analyzing oriented patterns', *Computer Vision, Graphics, and Image Processing,* **37**, 362–385 (1987).
9. M. Kass and A. Witken, *Knot Reel*, Siggraph'86 Video Review.
10. W. T. Reeves, 'Particle systems—a technique for modeling a class of fuzzy objects', *ACM Transactions on Graphics,* **2**, (2), April 1983, reprinted in *Computer Graphics 1983*, pp. 359–376.
11. F. Crow, 'Summed-area tables for texture mapping', *Computer Graphics,* **18**, (3), 207–212 (1984).
12. L. Williams, 'Pyramidal parametrics', *Computer Graphics,* **17**, (3), 1–11 (1983).
13. P. Heckbert, 'Filtering by repeated integration', *Computer Graphics,* **20**, (4), 315 (1986).
14. A. Glassner, 'Adaptive precision in texture mapping', *Computer Graphics,* **20**, (4), 297 (1986).
15. Thinking Machines Corporation, *Connection Machine Model CM-2 Technical Summary*, technical report, May 1989.
16. W. D. Hillis, *The Connection Machine,* MIT Press, 1985.
17. W. D. Hillis, 'The connection machine', *Scientific American,* **255**, (6), 108 115 (1987).
18. K. Sims, *Excerpts from Leonardo's Deluge*, SIGGRAPH '89 Video Review.
19. C. Reynolds, *Ductile Flow,* SIGGRAPH '90 Video Review.
20. P. Heckbert, 'Survey of texture mapping', *IEEE Computer Graphics and Applications,* **19**, No. 10, 56–67 (1986).

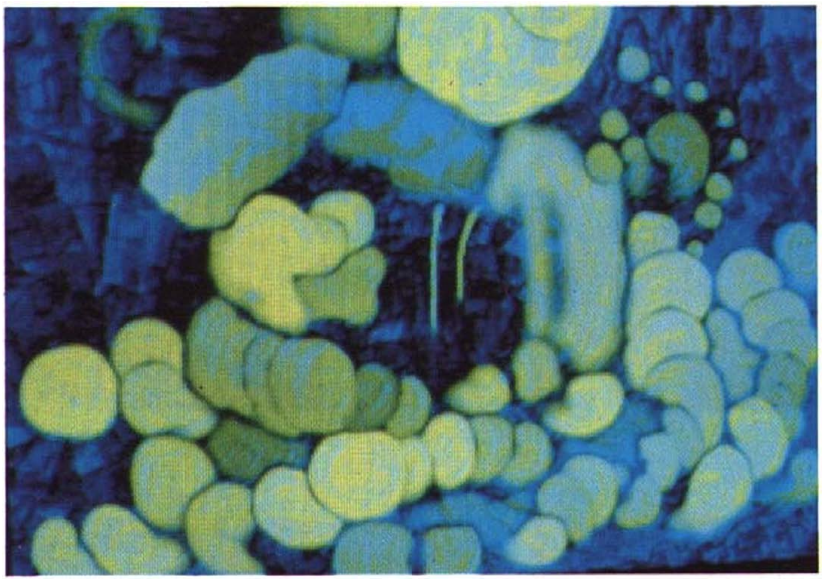

Plate 1 (Sims). Completed flow-field is displayed in green with the drawing in blue

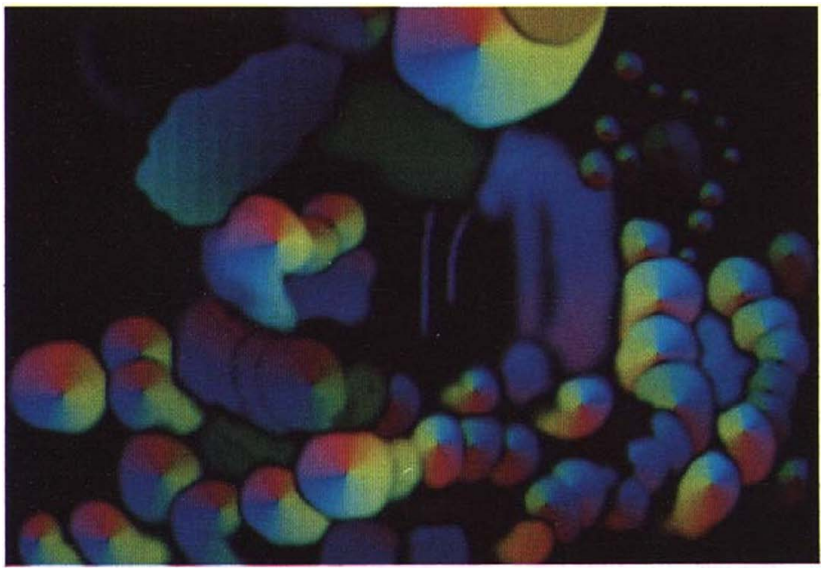

Plate 2 (Sims). Completed flow-field is visualized using a full colour representation