Image Morphing

One mainstay of computer animation is the image morph, whereby one image changes smoothly into another. This effect is widely used in movies and music videos, and has been around since at least 1903. Of course it was not done digitally in those days, and even today many movie studios mix digital effects with conventional animation techniques.

The term morph refers to an image transformation that occurs in steps, and contains not only a change in the intensity of each pixel from the starting to the ending image, but also a warping of the image grid. We usually refer to the intensity change as a "blend", or "dissolve", and the movement of pixels as a "warp".

Blend or Dissolve

Suppose we have two images, and wish to animate a sequence in which the first image changes smoothly in intensity to the second. A good example of this might be a horror movie where a human turns into a werewolf at the exact moment of a full moon. You might shoot two still images of an actor - the first would be his normal face and the second with wolf makeup. Because the locations of all of his features are the same in both images, a simple blend of the two images might be a realistic enough effect for your purpose.

So, how might you suggest this transformation cinematically? The exact intermediate image $f_{1/2}(x,y)$ between the initial image $f_0(x,y)$ and $f_1(x,y)$ is simply

$$f_{\frac{1}{2}}(x,y) = (f_0(x,y) + f_1(x,y))/2$$

and it would look somewhat like both images. But you probably want the translation to appear more smooth, so you could generate a whole series of images where each image in the sequence $f_q(x,y)$ is calculated from

$$f_q(x,y) = f_0(x,y)*(1-q) + f_1(x,y)*q$$

where q is the fraction, ranging from 0 to 1, that describes how far along you are in the sequence.

Each image $f_q(x,y)$ is then displayed in sequence, using as many values for q as you need for the desired appearance of the animation. As q approaches zero, you obtain a result similar to the initial image, and as q approaches 1, you get the final image.

Technically, you have created a series of blends that differ only in the amount of the initial and final image

intensities you use.

Matlab details

Creating, storing, and displaying blends in Matlab is particularly simple. Some example code to create a simple movie might be:

```
for nframe = 0:10,
   q = n/10;
   f = f0*(1-q) + f1*q;
   disbytebw(f);
   M(nframe+1) = getframe;
end;
```

The data structure M contains each scene as saved by the command getframe. f0 and f1 are the initial and final images in the blend sequence.

To play the movie, the command

```
movie(M, -10)
```

will play the movie 10 times, with the minus sign indicating that the movie is to be played backwards and forwards, instead of just forward.

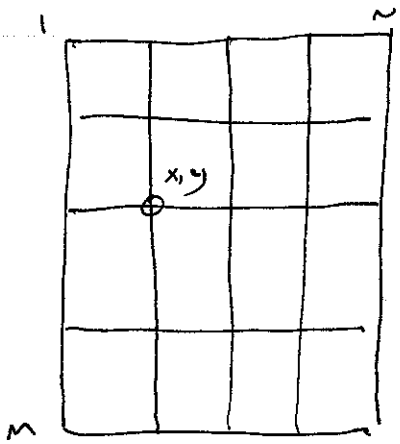
Some sample scenes from a blend for the two images above are:

So, the blend is simply a fade of the intensity of each pixel from its initial value to its final value, with any number of steps in between.

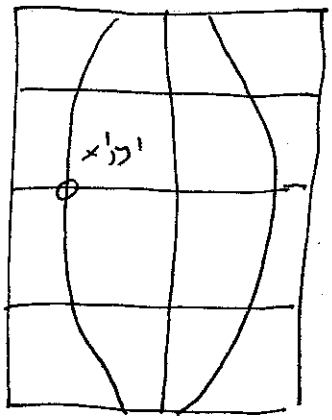
For most sequences, though, you will want to create several intermediate scenes first and then blend between those. In the werewolf example we discussed above, you might want to shoot the actor in his natural state, with half the wolf makeup on, and then with all makeup. Two blends then makes the complete sequence. This will result in a much more reliable and believable movie sequence.

Warps

The other component of a morph is a warping of the grid that an image uses. This allows us to change the shape as well as the intensity of objects in a scene. Consider a square grid of points that changes shape by expanding the points in the middle of the image horizontally:



original grid



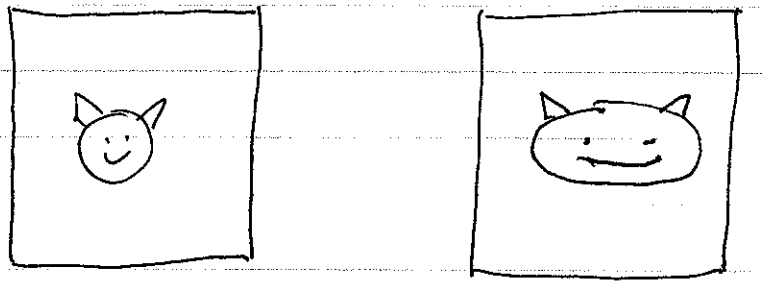
final grid

A point originally at position (x, y) is distorted, or moved to new position (x', y'). The above transformation may be represented analytically by the following equations:

$$x' = x + a \cdot \left[x - \frac{N}{2} \right] \sin\left(\frac{y\pi}{N} \right)$$

$$y' = y$$

An image subject to this transformation might then look like this:



Before After

To generate a series of intermediate images as we did in the blend, Then, we would first define a displacement vector for each point, such that

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} x_{disp} \\ y_{disp} \end{pmatrix}$$

where (x_{disp}, y_{disp}) gives the change in location each pixel undergoes in the transformation. In our previous example,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a(x - \frac{N}{2}) \sin\left(\frac{y\pi}{M}\right) \\ 0 \end{pmatrix}$$

Intermediate scenes can then be generated as a function of our fractional displacement q according to

$$f_q(x,y) = f\left(x + qa(x - \tfrac{N}{2}) \sin\left(\tfrac{y\pi}{M}\right), y + 0\right)$$

Similar code to what we had before generates the warp.

One problem with the simple implementation we had above is that if the movement of a pixel during a step is greater than one in x or y, we can easily end up with "holes" in the output image, if we define the transformation backwards to the above, that is if we use the input points

to calculate output points. But by using the output location as the independent variable, we will always fill in the image completely.

Numerically we might not want to calculate the location arguments for every time step, since they are all related to each other linearly. This is especially true in Matlab since it is an interpreted language, and does matrix operations quickly. Thus we can precalculate the total displacement at each ~~time~~ point in space, and scale these according to the q value.

Morphs

Suppose we now want to blend one image to another that is similar to, but geometrically different from, the original. A good example here is a morph from one face to another. Both faces have two eyes, two ears, etc, but they are ~~located in~~ located in different positions. A simple blend won't produce physically believable intermediate images, for example the halfway image will four eyes instead of two:

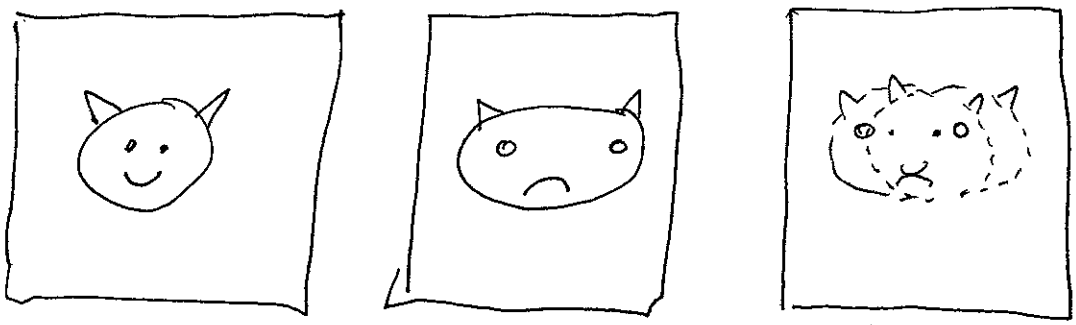
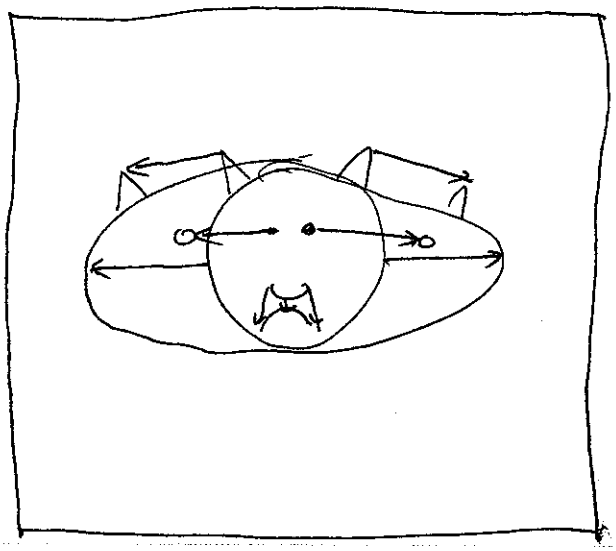


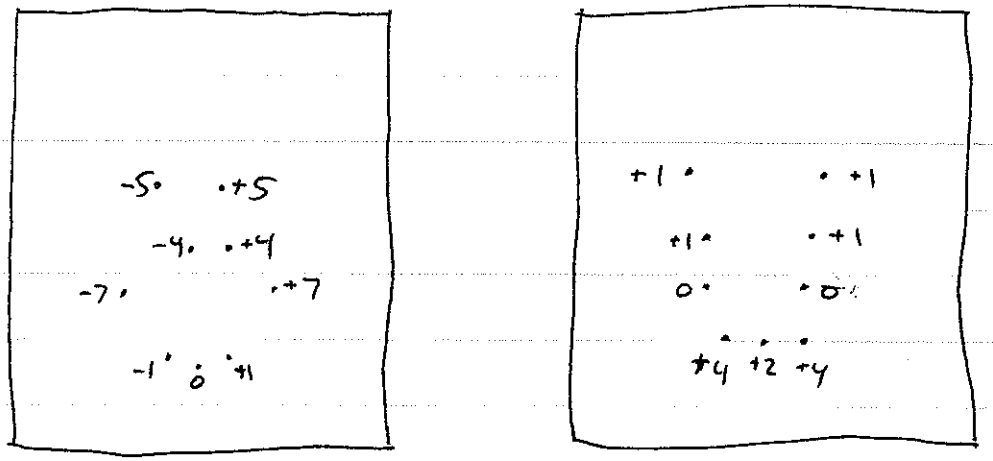
Image 1 Image 2 Halfway Image

How can we make a better image transition sequence?

Suppose we define a transformation to go from the location at each feature in image 1 to its counterpart in image 2. We can think of this as a set of vectors connecting locations in the 2 images. Plotting both images together, we can represent the transformation at the given points like this:



These point define for us how particular points in image 1 map to image 2. We can think of this as representing two functions, one giving the x transformation as a function of location, and one giving the y transformation as a function of location, but only being defined at a very few locations. Plotted as a matrix, we might picture these as:



X transformation Y transformation

But we want to move all the points in the image, not just the few for which we have tie points. So, we need a way to interpolate these functions between the known data. Polynomial curve fits are one method, which we can do fairly easily in matlab, and they work fairly well for smooth functions. But for more complicated functions we will have to use more sophisticated methods that often are quite slow in Matlab. We'll talk about these more later.

Computing a morph

So, to compute a morph we need to combine both of the above warps and blends. Assume we have derived two matrices X and Y that give us the displacements at each image location to warp from image 1 to image 2. We once again compute the intermediate frames as in the blend, changing both the pixel locations and intensities by the same factor q.

Some pseudo-code in matlab might look like:

```
% Assume X and Y contain the x and y displacement functions,
that f0 is the initial image, and also that f1 is the
final image.

for nframe = 0:10,
    q = nframe/10;

    for i = 1:N,
     for j = 1:M,

% sample f0 to the intermediate frame coordinates
    f0q(i,j) = f0(i+q*X(i,j), j+q*Y(i,j));

% sample f1 to the intermediate coordinates
    f1q(i,j) = f1(i+(1-q)*X(i,j), j+(1-q)*Y(i,j));

% now blend these together
    f = f0q*(1-q) + f1q*q;
end;end;
    disbytebw(f);
    M(nframe+1) = setframe;
end;
```

Once again this can be displayed using the movie command.