

3. A Study of Requirements, Problems and Possible Solutions

In this section we look at what specific features are required of the program, how these can be implemented, and how knowledge of the domain in which the program is to be used can be exploited. This will enable us to produce a software product that is easier to use and more effective than it might otherwise be.

3.1. Functional Requirements of a Morphing Program

In addition to basic consistency and usability requirements, any image morphing program should be able to take two images, determine a correspondence, and render the result to screen or disk. Where correspondence isn't automatic the user should have the ability to pick out features on each image and to flag them as corresponding to one another. The user should also be able to edit, delete and dissociate features. If an image is being rendered the user needs to be able to specify how far through the morph the rendered image should be.

3.2. Functional and Non-Functional Client Requirements

Requirements were obtained from the Client using non-directed questioning and iterative prompting where it was reasonable to do so. The software they require would be used to observe at what point people stop seeing the face of one species and start seeing another in human-monkey hybrids [HALL04] and follows on from previous research such as [CAMP97] and [ROSS01].

The Client requires an image morphing program capable of taking two faces of different species and generating a progression of hybrids. Input images will be of a common size and in greyscale, output should be as JPEGs. These shall constitute the core mandatory requirements. There is no need to incorporate a viewer into the program, as the Client will use a dedicated tool for image display and data collection.

The Client also expressed an interest in being able to create 'triangular morphs' in which hybrids are synthesised between each of the three pair combinations of three target images. It is also desirable that feature specification should be as simple as possible whilst not losing the ability to create an effective morph.

It is most likely that the program will be run on a Windows PC, but the Client has access to a significant number of Apple Macs, so cross-compatibility is desirable.

The Client's research will be carried out in parallel to work on this project, so it is imperative that a working product is available at the earliest stage possible.

The program should produce morphs as reliably as possible and should crash as little as possible, even in early releases; and where there is a crash the user should be informed graciously and with sufficient information to be able to report the problem. A rendered morph - or preview - should be generated without an inconvenient delay of more than a few seconds, and since old work will need to be loaded into new versions future compatibility is important. Installation should be quick and simple.

3.3. Analysis of Problems and Solution Strategies

We now examine some approaches, including those raised in Chapter 2, which may help to meet the Client's requirements.

3.3.1. The Morphing Interface

With so many possible approaches to the task, one of the first things to decide on is which rendering algorithm will be used. We summarised some major algorithms in Chapter 2:

- Patch based methods are simple to specify and program, and fairly quick to render, but they tend not to look particularly natural.
- Spline meshes are more natural and render quickly, but require more skill on the part of the user.
- Field-based warping is intuitively expressive for the user and relatively quick to program, but rendering is slow.
- Energy minimisation, radial basis, and thin plate spline models are complex to program and slow to render, though feature specification may be easier.
- Finally, freeform deformation techniques can be more intuitive for the user and lead to more accurate feature specification with reasonable rendering times, but are initially more complex and require greater developmental effort.

Each of these techniques presents its own problems, such as foldover and the bottleneck problem in the B-Spline mesh, and ghosting in field morphing. The program should seek to minimise the effects of these artefacts whilst retaining the level of control demanded by the user and not complicating things more than is necessary. The interface could, perhaps, flag a warning if there appears to be a problem in the specification, or even attempt to rectify the issue itself. This could be performed at render time so the user works with one level of feature primitives and the program works with a lower level.

All of these algorithms make reference to some type of feature, whether that be the contours of an object or the control points in a mesh. Throughout this we will continually refer to features, but because we are referring to features on different levels from time-to-time we will make a distinction between three specific types:

- *Morph-level features (strictly warp-level features)* are the most primitive elements with which the warping algorithms operate, and is usually what is meant when this report refers to a ‘feature’, e.g. a point on a mesh, a directed line.
- *User-level features* are the smallest primitives with which the user will normally work. For example, a circle may be a user-level feature, even if it is transparently represented by four morph-level arc features. Usually users will work directly with morph-level features.
- *Image-level features* refers to one or more morph- or user-level features which together represent a higher level visual concept, such as vectors which can be overlaid on an image to mark out the nose. This may also be used to identify corresponding visual features, e.g. the nose on the image.

Like WinMorph, it may be more intuitive to be able to express user-level features at a higher level than the morph-level primitives by constructing primitives out of the morph-level features.

We will continue this section assuming the field warping technique wherever the content cannot be generalised. The field warping technique is expressive and convenient, though many examples citing it could be extended to other techniques.

The software we looked at in Chapter 2 had many varying approaches to feature specification, and none seemed to be immediately intuitive to a new user. The interface must allow features to be constructed, edited, deleted, and possibly paired. This may require the ability to select one or more features. These operations can be actioned by performing particular gestures, operating in a particular mode, or a combination of the two. Since a new user will expect to use the mouse to control features we have a range of gestures available, including movement, scrolling, dragging, and clicking one or more times.

The typical routine for specifying a feature is likely to involve adding the feature and then adjusting it. Feature construction may consist of stamping a point, drawing a line, or composing a polyline, operations usually associated with (left) mouse click and dragging gestures. Similarly, making a selection or adjustment is also usually performed with a click. In order to maintain this consistency the interface should allow to mouse to act in one of several modes, which must be switched between soon after specifying any feature. This is tedious and can confuse the user as to what mode they are in, even with visual cues.

As an alternative, the functionality could be reassigned to other mouse gestures. This would lead to a counter-intuitively over-specified mouse, and could also present problems if the program was run on a Mac, which normally has only one mouse button. The common strategy used by graphics programs such as PaintShop Pro is to employ separate modes, but creating a graphic is less algorithmic than specifying a morph, so the modal approach appears less prominent to the user. A middle-of-the-road alternative would be to combine keyboard modifiers with mouse gestures, as used by *FaceGen Modeller* [SING03], which explains this behaviour to the user with a text label.

Correspondence specification needs careful consideration. It is unusual to work with two images in order to generate just one (or an animation), so it isn't easy to make such an interface intuitive for the user. In the software review we saw two main approaches to specifying correspondence:

- Mirror the features created on one image onto the other where they can be edited. This can lead to clutter and confusion, especially if features overlap, but it ensures correspondence between compatible features.
- Create features independently on each image and then explicitly pair them. This ensures that all features are fitted to the correct places on the images and not simply duplicated and left, but features may be unpaired, or attempts may be made to pair incompatible shapes. This pairing phase requires additional work by the user.

It wouldn't be too difficult to imagine a third approach that combined the convenience of automatic correspondence with the flexibility to express each corresponding feature independently:

- (Semi)-automatic feature correspondence that dynamically creates a correspondence between feature pairs at render-time based upon their attributes.

An increasingly common area of investigation is transition control. In basic morphing techniques each feature will morph at a constant rate across the whole animation. This doesn't look fluid if two morphs are glued on to one another, and moreover, a morph

can be made more interesting by animating different sections at different rates, directing the viewer's attention to specific parts of the image. This may lead to more believable 'transformation' animations, but doesn't produce reliable hybrids (unless applied by modelling a genetic transformation of the hox genes, which affect how and where features develop). For this reason feature transition control will not be a major part of this project.

3.3.2. Automation

We have already noted that to morph any image requires both a set of features and a correspondence between them to be determined. This is a requirement of all morphing techniques and is independent of how the morph comes to be specified, thus an automated technique must still perform both these steps before calling the warp function.

Goals of Automatic Approaches

The aim of an automatic technique is to correctly produce the best morph possible (at least to the standard of a typical user) without significant human intervention and in as short a time as possible. In essence it tries to package together a large chunk of the morphing domain into a program that will generate an animation in a single click.

To produce a quality morph any automated technique must systematically process the two input images in the same way as a master animator. In their thorough and accessible book [GOME99] Gomez et al. suggest ten principles for a good metamorphosis:

- *Attributes transformation:* Properties from one image should be mapped to the same properties in the other image. This relates to determining a correspondence between images.
- *Topology preservation:* There should be continuity in shape – no torn shapes or unrealistic intermediates. This high-level constraint has the effect of eliminating most types of ghosting and requires that corresponding features don't overlap in intermediates. It may be possible to detect this automatically, but avoidance would be more difficult.
- *Feature preservation:* Sharp corners should map onto sharp corners rather than being formed by a join in a compound straight line. For example, a morph between two triangles, one resting on a side and the other resting on a vertex, should involve mapping corresponding vertices rather than splitting sides to form an intermediate hexagon. This is again feature correspondence, but it highlights the possibility that the distance between corresponding features may change significantly over the animation.
- *Rigidity preservation:* Conservation of distances (e.g. skeletons), angles, volumes and convexity as appropriate. This is particularly true when morphing between two conceptually similar images, e.g. a person in two different poses. Conservation of volume and area is quite a high level concept and may be difficult to automatically determine from a two-dimensional image.
- *Smoothness preservation:* Smooth-to-smooth mappings are more effective than folding a smooth boundary. There is also smoothness in the rate of transition. Since smoothing is polish for the morph it won't be a high priority for the automation.
- *Monotonicity:* Changes in volume, area and other object parts should be done monotonically, i.e. in one direction. This means that any progression curves should not have any negative gradients, which ought not to be a problem to automate.

- *Nonlinearity*: Linear interpolations fail to work realistically for rotations, for example a turning arm may be shortened in intermediates. Joining two linear interpolations can produce a jolt, which can be easily spotted both in images and in animation. Since linearity is fast and simple to work with it would be a good starting point for automation, but nonlinear extensions would overcome its inherent limitations.
- *Use of transformation groups*: If the nature of the warp is known then a suitable mathematical interpolation will yield realistic results, e.g. a perspective mapping for an image rotating in a plane not parallel with the eye. Unfortunately automating such a high level concept would require metadata which could not easily be gleaned from the image.
- *Slow-in and slow-out*: Progressions are better starting and finishing slowly. Like nonlinearity, this can evolve from simple interpolations.
- *Avoid morphing leakage*: Warping a foreground image will leak the warp into the background. Separating the images and recombining them post-warp is a good way to solve this. Determining the areas occupied by the images to be morphed between is a natural extension of feature detection, but it wouldn't be unreasonable to start with images with only block colour backgrounds, which can't be geometrically distorted. Automating this would be just beyond the scope of this project.

Detecting Features

One of the main factors prompting the eye to recognise a feature is some sort of visual boundary around it, so it may help to use an edge detection algorithm to decide on key features, or at least the edges surrounding them (which is enough to perform the morph). Many edge detection algorithms exist, and the most common technique is to use a convolution kernel to effectively displace the image and then take it away from the original, highlighting edges as a discretely calculated gradient along the displacement direction. This is usually repeated at different angles with the resultant images combined to produce a gradient map of the image. Steeper gradients are usually shown in white and relatively flat areas in black (though the reverse is also common). An edge usually exhibits a steep gradient, so more pronounced edges appear white. Larger kernels produce better results (since they can simulate more angles of displacement and are less influenced by noise), but require more processing.

An abundance of material exists on edge detection filters, a few of which are listed below with information based mainly on [RUSS02] and [@@@]:

- *The Laplacian filter* is symmetric about the centre both horizontally and vertically with the centre value cancelling out all other values. It serves to highlight edges, but is better at making noise more pronounced.
- *The Roberts' Cross Operator* uses a 2×2 kernel to displace the image by half a pixel in both dimensions, but is also noise-sensitive.
- *The Sobel Operator* is like Roberts' Cross, but uses a 3×3 matrix. This reduces noise by providing smoothing and makes edges more pronounced, but it takes more time to process and the edges may be quite thick. Several rotations are performed then Pythagoras' theorem is applied to approximate the gradient in two dimensions (sometimes reduced to an addition for speed). *Kirch* achieves a similar effect by keeping track of the results as they are calculated.
- *The Canny Operator* @@@ (summarise from source)

It would seem from this that the Sobel family of operators offers a good compromise between pronounced feature detection, noise avoidance and speed.

An edge detection filter might be used for point snapping, in which case two things must be considered for any prospective point – the gradient at that point and the distance from the original point. Points should be on edges with steep gradients (the local maximum) but remain within the locality of the point to be snapped. This could be done by limiting the range of the area searched or by combining an inverse square law with the merits of the prospective points' gradients to give a score which favours nearby edges, even if more pronounced edges may exist a little further away. Such a technique might risk giving corners no priority over edges, even though vertices are more important in specifying a morph.

Determining how a line may be used to approximate an edge is more difficult since it requires that a continuity is established between edge pixels, and a simple consideration of the eight neighbouring pixels would allow a line to have only one of four gradients. An improvement over this would be to identify key pixels and pick lines as paths between these samples. This would mean having to identify significant pixels in the right quantity and density, a job that might lend itself to corner detection algorithms such as [SOJK02], though these algorithms tend to be tested on well-defined artificial corners rather than less obvious natural features such as those on faces. A line could be drawn between two key pixels if the pixels between them suggest that they are joined. Such a method would identify lines even if they were broken or not perfectly straight. A possible algorithm is:

1. Detect corners or key pixels
2. Perform a component-wise edge detection (so edges between different colours of similar brightness are detected)
3. Convert to greyscale
4. Blur (so that there is a leniency when tracking edges)
5. Determine which points should be joined. This could be done by calculating the mean brightness along a direct path between each point pair with any result exceeding a threshold becoming a line.

This could result in parallel lines, traced shapes not being closed and key points being split and merged.

There have been many other attempts at using edge detection and Hough transforms to detect and pair image features, with a fair degree of success. Combining this with biometric data and methodologies could advance such approaches. Similar techniques may work in a semi-automatic environment to bring ease and precision to manual accuracy and flexibility.

Determining Correspondence

Once key features have been identified in two images they must be paired up, but it is unlikely that a convenient correspondence can be established without first harmonising the feature configurations. For example, there may be more key points on a smiling mouth than on one presenting a blank expression since the curved smile would need to be approximated by several small straight lines. A domain-independent approach to establishing a correspondence would first need to identify that these morph features correspond to a higher level image feature for each image, then the two image features must be recognised as corresponding, and finally a correspondence between the morph-level features must be established by adding points and merging lines as appropriate.

There could be several possible mappings of morph-level features. Even if the number of morph-level features is the same, mapping them to each other may still result in unrelated features being paired. Establishing a correspondence between morph-level lines comprising an image-level feature may include:

- 1:1 mappings – A direct correspondence can be established providing that the lines are coherently oriented.
- 1:many mappings – The single line may have to be split or the destination features reduced so that each point has a complimentary point in the other image. It may be inappropriate to divide the single line at regular intervals.
- Many:many mappings – Corners should map to corresponding corners, if they exist, though sections between corners may not map 1:1. It may be possible to do this by considering angles, positions and distances, but it would still be unclear how to turn an ‘N’ into an ‘L’, for instance since there is no obvious way to tell which – if either – of the corners in the ‘N’ correspond to the corner in the ‘L’. Points may need to be added and others removed within segments. Existing points may need to be aligned better. Points on obtuse joints should be removed in preference to those forming acute angles.
- 1:0 and many:0 mappings – no mapping to the destination is possible since the feature doesn’t exist or hasn’t been detected. The risk here is that in the absence of one image-level feature two completely unrelated image-level features may be paired.

This is without considering occlusions, rotations, undetected subtle edges, scratches acting as false edges, changes in feature topology such as a closed eye opening, line intersections (either in keyframes or intermediates), secondary features used to control the morph and further real-life problems.

All this assumes that features are identified on each image independently. It may be easier to identify a correspondence if the features on the second image were determined from those in the first. This would impose a template structure on the second image meaning that key points would simply be perturbed from their positions in the first image, providing the second image is similarly oriented and with common features. We can further abstract away from this by eliminating the first image as a definition for a template and instead use an independent template that can be stamped over each image, perhaps initially by the user.

Possible Implementations

Covell [COVE95] has devised a method similar to some of the above approaches for determining a feature-based correspondence between two images based mainly on orientation and position of edge segments in so-called oriented cartoons, which are similar to the output of an edge detection algorithm with normalised lighting. By making use of a saliency threshold edge segments are identified and then chained, forming clusters. The result produces reasonable morphs that are relatively easy to make corrections to, though in the samples shown in [COVE95] the source features are almost spatially aligned before the correspondence algorithm is actioned.

Given that we have a known domain in which we our program will operate, can we make use of specialist knowledge to simplify the task of specifying the morph? We have already seen how snakes use edge detection to shape themselves to the contours of an image, but if we know what that image is can we make feature specification any easier?

While many animal faces have similar characteristics – eyes, nose, mouth etc, the configuration of these can vary substantially between species, such as the front-facing eyes in predators compared with the side-facing eyes of prey. Each is clearly a face, but can the program use such a generalisation, or is a greater familiarity with each species of face required? This question has parallels with the subject being investigated by the Client – at what point do we stop seeing the face of one species and start seeing the face of another? The Psychological approach suggests that humans have prototypes for different types of faces. These prototypes can be thought of as visual templates that match any face within certain bounds. This idea of a set of templates could be extended back to the program. If a template were superimposed over a face it would be easier to specify features, and this could even be done with a degree of autonomy.

The idea of a template is a step towards a domain-specific approach. It might be possible to use knowledge from a domain such as face recognition to aid both feature detection and correspondence determination.

In an attempt to use morphing for video compression [SIEW95] Siew uses ‘seeds’ to locate key image features between which a correspondence should be established. Siew notes, “for facial images, seeds should be located at the eyes, nose and mouth because they are the most noticeable features of the human face” [SIEW95]. Also of note is a decision to train a neural net (which would be very involved) or use a template-fitting approach involving a measure of potential energy [YUIL92]. The method used involved component-wise edge detection to locate the mouth, from which the nose could then be accurately located and the eyes also found. This approach did involve some significant assumptions about the images, such that they were (human) face portraits, but still coped with beards and glasses. It would take about 2.5 seconds to locate the mouth, nose and eyes of a 98x97 pixel image with the accuracy of detecting features at 85% or better, though time increases with image size (probably in proportion to the number of pixels). Bearded faces had reduced feature detection rates, which might have implications if such an algorithm were applied to monkeys.

To keep costs low a hybrid of a template and localised edge detection filter may work well in a semi-automatic environment in which the user can ‘drop’ the template over the feature it should match against.

3.4. Determining Success

A successful system will meet the Client’s mandatory requirements of a program that can generate a realistic hybrid image at a specified point for any two input images, and then save that image to disk. Ideally feature specification should be as simple and automatic as possible without any side effects. A reasonably advanced system will allow the specification of ‘triangular’ morphs between any two images in a set of three.

The program will be deemed a success if it:

- Meets the mandatory requirements.
- Is agreed as successful by the Client.
- Is used in the research task it was designed for without significant trouble.
- Takes measurable steps towards effective automatic specification of inter-species facial morphs.

In addition, feedback from the Client will be very useful in addressing issues and gauging how well the tool is being received.