The Photo-Realistic Synthesis of Novel Views from Example Images

by

Stephen Lines

Submitted to the Department of Brain and Cognitive Sciences in partial fulfillment of the requirements for the degree of

Master of Science in Brain and Cognitive Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1996

© Massachusetts Institute of Technology 1996

Signature of Author

Department of Brain and Cognitive Sciences
May 28, 1996

ARCHIVES

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

JUL 11 1996

LIBRARIES

Certified by

Tomaso Poggio
Uncas and Helen Whitaker Professor, Department of Brain and Cognitive Sciences
Thesis Supervisor

Accepted by

Gerald E. Schneider
Chairman, Departmental Committee on Graduate Students
THE PHOTO-REALISTIC SYNTHESIS OF NOVEL VIEWS FROM EXAMPLE IMAGES

MS Thesis, December 10, 1995
Stephen Lines

Novel views of an object such as a human face or body can be synthesized semi-automatically from multiple example views using techniques from machine learning, computer vision, and image processing. To the extent that the process of image synthesis can be automated, one may say that a graphical model of the object is "learned" from the example views. As a means to producing computer graphics, this approach is novel for beginning and ending with 2-dimensional photo-realistic images, without relying on a 3-dimensional model anywhere in the process. In these and other respects, this work may be regarded as a "neurobiologically motivated" approach to computer graphics. This technology may be applied to computer graphics and interactive multimedia (e.g., the creation and animation of virtual actors), new human-machine interfaces (e.g., talking heads), computer-aided design (e.g., interactive simulations), forensic identification, and very low bandwidth teleconferencing. I have developed a system that reads in a variable number of example images and allows the user to synthesis novel views within an $N$-dimensional parametric image space. It does so by creating a re-usable module or "image network" that essentially contains some of the examples, the pixel-wise correspondences among them, and a mapping function for translating user-defined parameters into novel synthetic images. These modules or networks play the role of 3D models without requiring any explicit 3D information; they are created automatically, and may be enhanced using a combination of manual and automated tools. In the proposed thesis, I will explain the capabilities and limitations of this system (called NIN, for N-dimensional Image Net). I will describe the general problems it can solve and the particular techniques and algorithms it uses to solve them. I will also characterize where and how it fails and what might be done to improve its capabilities, performance, and user interface. A demonstration of some of the results of this research is available on the World Wide Web (http://www.ai.mit.edu/people/spraxlo/).

This thesis describes research done within the Center for Biological and Computational Learning in the Department of Brain and Cognitive Sciences, and at the Artificial Intelligence Laboratory. This research is sponsored by grants from the Office of Naval Research under contracts N00014-91-J-1270 and N00014-92-J-1879; by a grant from the National Science Foundation under contract ASC-9217041. Support for the A.I. Laboratory's artificial intelligence research is provided by ONR contract N00014-91-J-4038. Tomaso Poggio is supported by the Uncas and Helen Whitaker Chair at the Whittaker College, Massachusetts Institute of Technology. Stephen Lines is supported by an AASERT Fellowship from ARPA.
1 The Problem

In its most general form, the problem is to synthesize many new views of an object from a few example views. Here I address a more particular problem, which may be formulated as building a system to do the following: Given M example images of a face, a whole body, or some other object, construct from them an N-dimensional image space \((N < M)\) and an approximation network for synthesizing novel views of the object anywhere within this space.

From parameter values defined and assigned to the example set by a human user, such a network may be said to "learn" a non-linear mapping from a low-dimensional parameter space to a relatively high-dimensional space of novel images [25], [26], [22], [5]. In addition to static views of pose or expression, characteristic motions may also be learned from an appropriately labeled sequence of views [16]. Animating the face or creature is then a matter of feeding the network a sequence of parameters; it renders the novel views by interpolating shape and texture information from the examples. This rendering process may be regarded as a variety of multi-dimensional image-warping or "morphing." It depends crucially on having some form of feature-based or pixel-wise correspondences among the example views. This last, the "correspondence problem," has been a key and difficult problem in computer vision since the 1960s.

The overall problem may thus be broken down into four sub-problems: finding correspondences, parametrizing the example image space, learning a mapping between parameters and the synthetic image space, and then rendering the final images. I will discuss each in turn.

1.1 The Correspondence Problem

When we know what a thing looks like, we can usually visualize it in many different views. Beyond simply recognizing an object in various configurations and from many different angles, we seem to be able to generate all these views, or at least some stereotypical, characteristic ones, in our imaginations.

Conversely, when we see two images as representing different views of the same object, we distinguish that object from its surroundings and identify it as a thing more or less invariant through variable views. Moreover, we can often visualize some transformation taking one imaged view into the other, whether this involves changes in the object, changes in our viewing conditions, or whatever.

Either way — whether generating new views of a familiar object, or else learning something about an object from different, changing views

— the visualization depends on at least implicitly solving the correspondence problem: Which points in the different images correspond to each other as representing the same points in the thing viewed? If the goal is to generate new views, one might reframe the problem as to where to place the visible parts of the thing in its various images (pixels from parts, so to speak). If the goal is to understand what is common to different views, or how different images depict the same thing, one might reframe the problem as how to transform one imaged view into another while minimizing the apparent distortion of the parts of the depicted object (parts from pixels). Actually, these formulations may be interchanged and the relations of these means to their goals will still make sense, for the underlying correspondence problem is the same, even if there are two fundamentally opposed approaches to addressing it.

The first approach is to infer 3D structure from the example images and use it to build a 3D model of the object. New views are then generated by manipulating this 3D model, and images are rendered by a process of projecting these new views onto a 2D image plane.

The second approach is to dispense with any explicit representation of 3D structure, and to search for correspondences directly between the example images. New views are then generated by interpolating among the inferred transformations within the image plane, and rendering may be accomplished without projection or any other explicit 3D transformations.

Either approach has its advantages and disadvantages, which I will discuss throughout this thesis. The problem of inferring 3D structure from example images is in general quite difficult, and traditional computer graphics usually avoids it altogether. Humans seem to be able to solve it, more or less, but it is not clear how, or even how much 3D information is actually processed. Humans make use of other perceptual resources besides monocular image analysis, yet they are still very good at judging absolute (as opposed to relative) depth. One might guess that the human visual system, or more specifically its capacity for visualization, makes use of both approaches. They may even interact more or less in a loop: computed correspondences between views sharing more or less the same image plane may aid in inferring 3D structure, and inferred 3D structure may aid in computing correspondences between images (or, parts from pixels from parts). It might also be valuable to know more than we do now about the brain's internal representations of view-correspondences and 3D structure — or lack thereof.

Computer graphics, however, has tradition-
ally emphasized only a part of the former approach, that of constructing 3D models and rendering new views by first performing all transformations in 3D, then projecting the transformed model onto an image plane. Only a few recent exceptions explore image-based methods, for example [18] [10]. Nor has computer vision been of much help. So far, it has offered little hope of a robust, general strategy for extracting 3D information from 2D images. The task of building the object models for computer graphics has thus fallen to the imagination and skilled work of artists, sometimes enhanced by physical simulations. The use of such models only partially solves the correspondence problem, or solves it only in the forward direction: Where the modeled object’s parts go in a rendered image. Correspondences are known only among images generated from the same model.

1.1.1 Computer Graphics with 2D Correspondences

This thesis explores the second approach: How well can one synthesize new views of an object from example images without attempting to construct a model of its 3D structure and environment?

In the absence of a 3D model of camera position, lighting effects, and so on, the objective of photo-realism dictates that the synthesized images inherit many of their properties as images from the example images themselves. Mathematically, the problem may be formulated in terms of estimating functions for the interpolation and extrapolation of images from sparse data, that is, from the example images and their assigned parameter values; I will come back to this. The correspondence problem here is to come up with suitable or “natural” transformations between one image and another without any explicit representation of a third dimension. By “natural” transformations, I basically mean continuous transformations that can be combined with each other and used to generate new images which are perceived as different realistic views of the same thing. Even simple morphing between two images faces a similar correspondence problem, and the solution, though typically arrived at by time-consuming manual intervention, entails essentially vectorizing both images (see [5]).

In principle, the correspondences need not be pixel-wise; they could be based on features (generic or expected) or perhaps on (overlapping) regions. Using pixel-wise correspondences gives one the advantage of having a low-level representation that is straightforward and uncumbered by symbols or other high-level constructs. It also benefits from much prior research and development in the computer vision community. In practice, everything else depends on the accuracy of the correspondences, and pixel-wise correspondences are certainly the most practical to use.

One of the best techniques for computing pixel-wise correspondences has proven to be a coarse-to-fine gradient-based optical flow algorithm using Laplacian pyramids (Lucas & Kanade 1981, Bergen et al 1992). Two major advantages of using an optical flow algorithm like this one to find the correspondences are that it is automatic and that one need not make many assumptions about the object imaged or the process of image formation.

The result of computing optical flow between two same-size images may be regarded as a mapping from a monocular view seen through an aperture to a different view seen through the same aperture at an earlier or later time. For every pixel location in the domain image, a corresponding point in the range image is estimated, which need not be exactly at a pixel center. In general, this mapping is neither injective nor surjective, and thus is not invertible. Self-occlusions of the object imaged, for instance, are likely show up in the mapping as roughly delimited areas of expansion, contraction, or fold-over.

Nevertheless, if images are regarded as feature vectors, one color or other value associated with each pixel location, then a dense, pixel-wise correspondence mapping between two images is equivalent to the vector difference between their associated feature vectors:

$$\Delta y = y_2 - y_1$$  \hspace{1cm} (1)

where $\Delta y$ is a vector containing the $x$ and $y$ components of the flow. Note that $\Delta y$ may also be regarded as the vector representation of $y_2$ relative to $y_1$, which is precisely what is computed by an optical flow algorithm:

$\Delta y \leftarrow \text{optical-flow}(\text{img}_1, \text{img}_2)$

One may thus treat the correspondence mapping as a vector field and interpolate between the domain and range image as a function of the unit interval: associate each pixel location in the domain with the difference between it and the corresponding point in the range image; at $\tau = 1$, replace each domain pixel by a color value sampled from the corresponding point in the range image, and at $\tau \neq 1$, linearly (or otherwise) interpolate between the original value of the domain pixel and that sampled from the interpolated location in the range image. For example,

$$y' = y + \tau \Delta y.$$  \hspace{1cm} (2)
Under the right conditions, the resulting sequence will appear as a transformation between two views of the same object, or as a “morph” between two different objects (see figure 3.)

Notice that although the resulting sequence may be replayed forward or backward, the lack of bijectivity in the correspondence mapping means that the domain and range are not interchangeable, even if the domain is regarded (like the range) as a continuous area rather than as a discrete grid of pixel locations. This non-invertibility limits the composability of mappings among several images and also makes for difficulties in rendering.

1.2 The Parametrization Problem

Parameter values assigned to the example images by a user are used to construct the multi-dimensional space in which new images are synthesized. The simplest way is to treat each parameter as an orthogonal dimension in an N-dimensional Euclidean unit cube, although this presupposes having examples close to each corner. A trivial instance is the 1-dimensional “morphing” problem between only two example images. The single parameter may be thought of as time; at \( t = 0 \) is one image, which is completely replaced by the other at \( t = 1 \). A face module with three independent parameters — degree of head rotation, degree of smile, and degree of mouth openness — could be constructed from eight examples representing all possible combinations of 0s or 1s assigned each parameter. Here the parameter valuation task amounts only to labeling each example image as frontal closed-mouth smile, left-rotated open-mouth no-smile, and so on. If one of the examples were rotated, say, only about 80% as much as the other rotated examples, it could be assigned 0.8 instead of 1.0 rotation, and synthesizing images near that corner would then entail not only interpolating between example images, but also extrapolating beyond that particular example. More generally, the shape of a synthetic image parameter space might be the convex hull of all the assigned parameter points embedded in a Euclidean space of the appropriate dimensionality. Or, if one allows extrapolation beyond the examples, then the parameter space might be regarded as a fuzzy shape extending this convex hull by a more or less definite amount.

Difficulties in parametrization arise when different features are not independent over their whole range. For instance, some people never fully smile without squinting. If the image net module is constraint only to produce fully smiling faces that look like the examples, i.e., squinting, then smile and squint are not independent. If they are made independent, then synthetic views can be made which may look very much like the person’s face, but with an expression they never actually show, namely, fully smiling with their eyes wide open.

What the parameters represent depends both on how the user assigns them and on the actual contents of the example images. Instead of rotation, mouth openess, and smile, the same set of face examples might be labeled with parameters representing distraction, surprise, and joy. Transforming between these two parametrization would be equivalent to changing coordinate bases. If the example images are all placed at the corners of a 3-dimensional cube, then (trilinear) interpolation is trivial; but they need not be. More general and powerful techniques may be used to construct an interpolation space from any assignment of parameter values to the examples. The pairing of examples and parameter values can be regarded as the data for approximating a function from sparse known values. Parametrization is then just the first step in a larger problem of learning, where learning from examples is formulated as approximating a function from sparse data. This learning problem is the topic of the following subsection.

1.3 The Learning Problem

In order to synthesize the “right” novel image from user-specified parameters, the system must somehow obtain a mapping from the space of control parameters to the space of possible output images. The available data are the example images and their user-assigned parameter values. Obtaining this mapping from these data is the learning problem. As indicated above, it can also be formulated in terms of multivariate function approximation from sparse data.

In the simplest cases, such as a 1-dimensional morph or bi-linear interpolation among four example images placed on the corners of the unit square, the learning problem is trivial. If we can solve it for more complex cases, then we need not place such restrictions on the set of example images, and the user’s choice of parameters need not depend as much on the particular set of example images. The control parameter space, for instance, need not be a Euclidean hypercube, and number of example images need not be 2^N, where N is the number of dimensions in this space.

To fit the usual representations of learning problems, the example images must be reduced to feature vectors. The most straightforward representation is simply one feature per pixel, originally suggested for visual recognition (Shausha, 1991). As discussed above in the section 1.1, a dense, pixel-wise correspondence mapping between two images is equivalent to
the vector difference between their associated feature vectors, which can be estimated by computing optical flow. Solving the low-level correspondence problem thus eliminates the need for high-level feature identification and texture mapping techniques. In their place, it makes possible direct mappings from control parameters to images through low-level, pixel-wise synthesis algorithms. The problem of learning this mapping may then be formulated as one of approximating a vector field.

Consider the problem of approximating a vector field \( y(x) \) from a sparse data set, to wit, from the \( M \) examples \( (y_i, x_i) \), \( i = 1 \cdots M \), where in our case the \( y_i \) are the vectorized example images and the \( x_i \) are the (user-assigned) control parameters for those images. For the approximation scheme, one may choose a regularization network or a generalized regularization network (see Girosi, Jones and Poggio, 1993), that is, a network with one "hidden" layer and linear output units (see for instance Poggio and Girosi, 1989). Consider the case of \( M \) examples, \( n \leq M \) centers, input dimensionality \( d \) and output dimensionality \( q \). Then the approximation is

\[
y(x) = \sum_{i=1}^{n} c_i G(x - x_i)
\]

where \( G \) is the chosen Green's function. This may be a radial basis function, like the Gaussian, or a spline, like some tensor product spline. The equation is equivalent to the network of figure 4, and can be rewritten in matrix notation as

\[
y(x) = Cg(x)
\]

where \( g \) is the vector with elements \( g_i = G(x - x_i) \).

Define \( G \) as the matrix of the Green's function evaluated at the \( M \) examples, to wit, the matrix with elements \( G_{i,j} = G(x_i - x_j) \). Then the "weights" \( c \) are "learned" from the examples by solving

\[
Y = CG.
\]

where \( Y \) is defined as the matrix in which column \( i \) is the example \( y_i \). \( C \) is defined as the matrix in which row \( m \) is the vector \( c_m \). Thus \( x \) is a \( d \times 1 \) matrix, \( C \) is a \( q \times n \) matrix, \( Y \) is a \( q \times M \) matrix and \( G \) is a \( n \times M \) matrix. Then the set of weights \( C \) is given by

\[
C = YG^+.\tag{6}
\]

Note that the vector field \( y \) is approximated by the network as the linear combination of the example fields \( y_i \) (see Appendix 2 in Girosi, Jones and Poggio, 1993).

That is:

\[
y(x) = YG^+g(x)
\]

which can be rewritten as

\[
y(x) = \sum_{i=1}^{M} b_i(x)y_i
\]

where the \( b_i \) depend on the chosen \( G \), according to

\[
b(x) = G^+g(x)
\]

Radial basis functions make a good choice for the Green's \( G \) function because the resulting approximation schemes tend to be fairly robust with regards to unevenly spaced examples. This is essential when one does not have complete control over the acquisition of example images, and in general it allows for a greater variety of examples to be used. Another advantage of radial basis functions is that their form does not depend on the dimensionality of the control vector \( x \). Different control parameters can thus be used with the same image networks.

Such schemes are still susceptible, however, to the "curse of dimensionality," namely that in the worst case, the overall complexity of the learning problem increases exponentially with the number of dimensions in the example space (i.e., in our parameter space).

One way to attempt to circumvent the "curse of dimensionality" is to partition the problem into several smaller ones. An example would be to create separate networks for controlling the mouth and eyes of one face, and compositing the synthesized parts into a single whole output image.

Machine learning could also be extended to other areas of the system. If the general problem is to automate the creation of novel views from examples, then machine learning techniques might profitably be used to:

1. Recognize problem areas in the correspondence mappings and aid in correcting them.
2. Automate the selection and parametrization of examples images from a large pool of candidates, such as all the frames of a film.
3. Monitor and in some ways improve the image quality of the synthesized views.

1.4 The Rendering Problem

Once we have constructed correspondence mappings among the example images, a parametrized control space, and a mapping from the control space to the novel view space, it remains only to synthesize the new images from
the relevant image data. This image data contains both shape and texture information, and both must be mapped and interpolated. This is the rendering problem.

When we speak of “morphing” one face or animal into another, or more generally, of interpolating between two images, there are basically two ways we expect what we see to change: the projected, 2D shape of one face, animal, or whatever will change into the shape of the other, and, simultaneously, so will the apparent surface properties, or texture.

By shape I mean all the geometric properties of both the 3D object and of its 2D projections or views in the example images. Changes in shape will also subsume small changes in position and posture, as well as many changes that are only apparent, in that they occur in the 2D images as a result of viewing angle, distance, foreshortening, and so on, and not (only) as a result of movement or deformations of the 3D object. Since NIN contains no explicitly 3D representations, it can only interpolate 2D shape data. A straightforward and fairly robust representation of the shapes of a group of similar objects is a vector of offsets relative to a standard shape, or reference image. This is precisely the information stored in the vectorized example images and their differences

$$\Delta y_{1-k} = y_1 - y_k \approx \text{optical-flow}(E_1, E_k)$$

By texture I mean all the optical properties of the object as it appears in an image, which can be captured by sampling small patches of pixels. For our purposes, this also includes shading and reflectance information. Using 3D models, one can separate intrinsic surface properties (fine details in shape and pigmentation) from accidental properties due to light and shade, reflections of nearby objects, and so on. Using 2D information only, all these effects must be interpolated the same way, and a shadow on one side of a face, say, can only be controlled if there are examples both with and without the shadow. If we were starting with only a 3D model instead of example images, and all the textures were generated artificially from scratch, then we might store them discretely and index them with a look-up table. As we are usually sampling from entire images rather than from isolated patches, the natural representation of texture for our purposes is in the form of the vectorized images themselves. Textures may then be sampled from anywhere within the example images, and in any size, shape, or order.

This distinction between shape and texture may not be wholly natural and absolute, but they are independent from each other in that the shape of an object in one image can be gradually transformed into the shape of another object (or of itself seen from a different viewpoint) while retaining its original texture, and vice versa, it can retain its shape while changing texture. Separating texture from shape can be useful not only for tasks of image synthesis, but also for tasks of image analysis such as face recognition and pose estimation [6].

In the case of 1-dimensional “morphing,” there is a simple solution to the rendering problem: warp both of the original images toward each other and then merely cross-dissolve between them. For the case of N dimensions with examples at each of the $2^N$ corners of the N-dimensional hypercube, pixel values in the synthesized image would generally be obtained by sampling and interpolating pixel data multilinearly from all $2^N$ examples. Less orderly example sets require more sophisticated pixel sampling and color interpolation schemes which are similar to those outlined in the section on the learning problem. In any number of dimensions, if the correspondence and rendering tasks are done well, it can appear that it is the objects themselves which change shapes and skins or move and grow into each other, not merely their pictures.

In our view-based approach to computer graphics, the central task of rendering is image warping. There are basically two different ways of approaching the image-warping task: forward and backward. In the forward orientation, the problem is whether to send pixel data into the synthesized, “target” image. In the backward, or inverse, orientation, the problem is whence to fetch pixel data from the example images. The inverse orientation is often used for simple problems of image warping because it can usually be implemented efficiently as a one-pass algorithm: for every pixel location in the output image, fetch or compute the appropriate color value and put it there. Mathematically, the color is a function of output location. If the color values are to be obtained from multiple correspondence mappings, then the domain for all these mappings must be the spatial area of the output image. In terms of vectorized images, the pixel grid of the output image is the natural coordinate reference frame. If optical flow is to be used to morph between two images, then, the flow must be computed from the final image to the initial one. But if the final image is to be of a synthetic view, not any of the example views, then computing flow this way is impossible: the reference images do not yet exist. The correct correspondence mappings from the reference frame of the output image back to the examples can only be obtained by approximate methods that involve warping and com-
bining the optical flow fields from the example images to each other. In short, the final stages of inverse rendering are likely to be simple, but obtaining suitable correspondence mappings may rely on costly approximations.

Forward warping or rendering is often more complicated at the stage of valuating pixels, but may bear a simpler relationship to the modeling or pre-rendering stages. It is the natural choice when the coordinate reference frame is that of the source image(s), not that of the target. It is much simpler to interpolate or otherwise combine forward correspondence mappings from one reference image to two other example images than it is to combine two backward correspondence mappings from two examples back to a common reference image. Algorithms for forward image synthesis are in general not one-pass; if pixel data is sent from each of the source images to one target image, collisions and holes are likely to occur, that is, pixel locations in the output image that receive multiple values or none at all (see fig. 15). The usual methods for resolving collisions and filling holes make use of one or more accumulator arrays and multiple passes through them to evaluate the final output pixel values. In short, the final stages of a forward rendering procedure may be ambiguous and expensive to attempt to resolve, but it may be much easier to obtain the correct forward correspondence mappings.

When the correspondences between the example images are represented by dense, pixellike, flow fields, the memory and processing requirements for manipulating these fields are bound to be on the same order as that for the images themselves. Given the trade-offs indicated above, both forward and backward image synthesis have their advantages and disadvantages. Experimentally, the choice of forward or backward rendering usually makes little difference in simple cases like one-dimensional “morphing.” But for synthesis problems with more than two control dimensions, forward rendering tends to be faster and produce results equal or better in quality to those of inverse rendering.

Output image quality depends first and foremost on the accuracy of the correspondences between the original example images. But it also depends on the accuracy of the approximations used to compose these together to create new correspondence mappings or image warping fields, and it is affected as well by general issues of signal processing, as in sampling and filtering. It is best to pose as few resampling and rendering processes as possible between the first digitization of the example views and the final output of synthetic views. The minimum of one pixel sampling process and one output pixel valuation process can be attained if one possesses mappings directly from the example image to novel views. This is the ideal situation, provided that these mappings themselves can be obtained accurately and efficiently. Indeed, the main advantage of representing the correspondences in the reference frames of the original example images is to avoid extra steps of sampling and filtering in the rendering phase. More will be said about these topics in the section on system design.

A simple algorithm for inverse vectorized image warping, essentially \( y' = y + \tau \Delta y \), is given by the following pseudo-code:

\[
\begin{align*}
  \text{for all } j : 0 \leq j < \text{height} & \\
  \text{for all } k : 0 \leq k < \text{width} & \\
  j' &= j + \text{RINT}(\tau \cdot \Delta y[j][k]) \\
  k' &= k + \text{RINT}(\tau \cdot \Delta x[j][k]) \\
  y'[j][k] &= y[j'][k']
\end{align*}
\]

A simple algorithm for forward vectorized image warping, essentially \( y' = y \star \tau \Delta y \) with no remedy for holes or collisions, is given by the following pseudo-code:

\[
\begin{align*}
  \text{for all } j : 0 \leq j < \text{height} & \\
  \text{for all } k : 0 \leq k < \text{width} & \\
  j' &= j + \text{RINT}(\tau \cdot \Delta y[j][k]) \\
  k' &= k + \text{RINT}(\tau \cdot \Delta x[j][k]) \\
  y'[j'][k'] &= y[j][k]
\end{align*}
\]

In practice, color values are interpolated bilinearly or biquadratically on subpixel image locations both when fetched from source images and when pitched to a target image. Forward warping also requires at least one accumulation array of the same dimensions as the target image. This accumulator is used to register collisions and holes, and may be used in a subsequent pass to fill the holes and arbitrate collisions.

There are basically two alternatives for resolving the ambiguities that result from collisions, that is, pixel data sent from different places in one source image to the same place in the target image. One way is to prioritize the source locations, as in computing their average and simply sampling the color value there (thus effectively inverting the correspondence mapping, at least locally, and reverting there to inverse warping). The other way is to select from all the color values sent to the same place, as in simply taking the last or the nearest “hit,” or, better, weighting each color value inversely to the nearness of the “hit” and taking the average. This latter alternative is sometimes called “snowballing”: sampled or computed pixel values are pitched at the target grid like colored snowballs; each one flattens out and sticks there, its mass spread out over several grid cells. This target grid may be identical to the lattice of pixel locations, or a refinement thereof. It is actually not just one grid
but several accumulations arrays superimposed on top of each other. One array accumulates the total mass from all the snowballs impacting it, in proportion to their overlap on each cell. Another accumulates the snowball’s color values as multiplied by these fractional masses. After the accumulation pass is finished, the value in each cell of the color accumulator is divided by the value in the same cell of the mass accumulator. The results in an image where all collisions have been resolved into weighted averages. Grid cells that fail to accumulate sufficient mass are either assigned a value in that same division pass (e.g., the same value as the last evaluated pixel) or else are marked as holes to be filled in during a third pass. For code fragments showing practical (but not fully optimized) algorithms, see figures 23 and 24.

2 System Design

NIN, the N-dimensional Image Net system, was designed to read in \( M \) example images \((N < M)\) along with their parameter values, and then automatically create an N-dimensional image synthesis module. This module can be used to generate arbitrarily many novel images and is saved to disk in the form of separate re-combinable components. Once the example images and parameters are specified, all user intervention, such as interactive editing of the correspondences, is purely optional. Throughout the system, the automatic methods are the default. Figure 12 gives a schematic diagram of the system showing both the basic and the optional inputs and outputs.

The required inputs are just the example images paired with example parameters. Optional inputs include all the network modules saved from the output of previous sessions, as well as any interactive interventions the user makes during the current session. If the original example images are to be cropped or otherwise transformed before correspondences are computed, then naturally the labels or any other means of specifying how they are to be derived from the examples must also be specified.

The default outputs are the components of an image network module: the pairwise dense correspondence mappings and the learning network specifications and weights. Optional outputs include (sequences of) synthesized images along with the parameter values used to generate them, intermediate images created as by-products of the synthesis process, and any image markings that result from using the interactive editing tools, for example, the coordinates of the mesh vertices described below in section 2.1.5.

The user interface consists of multiple windows and attached pop-up menus. A different menu pops up from each of these different window classes: the main window for displaying synthesized images, pair-wise or triple-wise editing windows, pre-warping windows, and control panels which generally contain sliders and buttons for specifying the parameters and modes for synthesizing new images in the main window.

Many options are available both from the command line from and the pop-up menus. These control any preprocessing of the example images, some variables in the computational optical flow, which approximation algorithms are used for learning and rendering, and so on. Details in the operations of the interactive tools are chosen from their own windows’ pop-up menus.

2.1 Implementation

NIN was implemented in C for reasons of compatibility and speed. The graphical user interface was built on top of X Windows using Silicon Graphics’ native GL functionality, but its modular design and object-oriented functional interface conventions would make it easy to port it to C++ and OpenGL. Different libraries separate the functions computing optical flow from those handling the tasks of numerical learning and images processing.

Its modular design also allows the different subsystems to operate independently of each other. For example, once an N-dimensional synthesis module is loaded, the rendering system only requires set of N valid parameters to synthesize a new view. These can be sent over a network by some other application, which effectively becomes an alternate input interface. Applications which analyze digitized drawings or photographs of faces have been used to drive the rendering system, making the resulting whole into a system for animating a virtual actor.

2.1.1 The Typical User Session

A typical user session begins with a command line specifying the example images files, options such as forward or inverse rendering, and any auxiliary data files whose names differ from those derived in a standard way from the names of the example image files. The main window appears containing the first automatically synthesized image, and from there on the user can open and close other windows or choose many different options and procedures to generate new views. The session ends when the main window is closed again.

Below is an example run-through with four example images of a face. It is a fairly typical session, but is meant to illustrate both the strengths and weakness of this approach to view synthesis.
1. From a Unix shell, enter the command line
nin A B C D -rwBT. Here A, B, C and D are the names of the RGB example image files; say A is a frontal pose with a neutral expression and the mouth open, B is the same expression but with the mouth closed and the entire head turned about 20° left, C is the same as A except that it is smiling (frontal pose, mouth open, smiling) and D is the same as B except that it too is smiling (rotated 20° left, mouth closed, smiling). The options are as follows: r means read the files containing the pair-wise correspondences if they exist, w means write them if they do not already exist, B means use both forward and backward warping algorithms (which will require having both forward and backward correspondence mappings), T is a cropping option, instructing NIN to look for label files containing the eye locations and to cut out from each example a square sub-image centered around the eyes.

If the appropriate correspondences are already saved, they will be read from disk; otherwise, they are automatically recomputed. Because of the B-option, there are six of them: the three forward mappings $A \rightarrow B, A \rightarrow C, A \rightarrow D$, and the three backward mappings $A \leftarrow B, A \leftarrow C, A \leftarrow D$.

2. Once the pair-wise correspondence mappings are obtained (which is equivalent to vectorizing the images and computing their vector differences), they can be combined to make mappings for synthesizing a new vector image. The default for using forward correspondences from four example images is just bilinear combination with each example placed at a corner of the unit square. This is equivalent to choosing the product of the absolute differences in each control dimension as the Green's $G$ in equation 3. Backward warping from four examples is somewhat more complicated, and involves an extra step of first warping the correspondence fields themselves. With the B-option, the novel views closer to the reference or "trunk" image A are rendered with forward warping; those within a distance of 0.33 from one of the other examples are rendered with backward warping. One way or another, the correspondence mappings are combined, and this step ends with the appearance of the first synthetic view in NIN's main window.

3. The user enters control parameters to synthesize new views, either by moving sliders or clicking on various $x, y$ positions in the main window itself as if it were the 2-dimensional unit square. Due to the content of the four bi-linearly interpolated example images, the pose (head rotation) is independent from variation in expression of smiling vs. not smiling, but not is not independent from that of the mouth being open or closed. The face can be made to smile whether looking straight ahead or turned anywhere between 0 and 20 degrees, but it closes its mouth as it turns. At the full rotation of 20°, its mouth is can go from neutral to smiling, but it cannot open, because there are no examples of the face turned with its mouth open. To make mouth-opening independent from head-turning using the simple multi-linear scheme would require four more examples.

4. Optional use of the interactive tools: If the synthesized images are acceptable, the user may skip this step. Rotating a human head by 20°, however, may result in significant occlusion and dis-occlusion. The most useful tool for correcting any resulting miscorrespondence is the Grid Editing Tool, described in section 2.1.5. Any corrections made using this or one of the other tools can be saved and for later sessions.

5. At anytime, all the views synthesized so far can be reviewed individually or as a "movie" sequence. The user may also choose to save any or all them, along with the parameter values used to create them. Conversely, if a sequence of the parameter values alone is saved, then these values may be fed back into NIN to regenerate the same digital movie.

2.1.2 Advantages and Disadvantages of Using Dense Correspondences

NIN's greater advantages, however, accrue in the multi-dimensional case of three or more examples. As noted above, there are any number of ways of morphing or otherwise interpolating between only two images. Getting from there to a general purpose engine for synthesizing new views with several independent dimensions is trickier. First of all, ad hoc representations of pairwise correspondences must be recast into a form sufficiently general and robust to be combined with each other. Likewise, the procedures for combining them into a new, parametrized mapping and using that to render the synthetic images must also be general, robust, and efficient.

Computing at least one dense pixelwise correspondence mapping for each example image
may appear costly, but this results in a general, low-level representation and it is much more feasible to automate this (e.g., using optical flow) than it would be to automate a set of high-level feature finders of sufficient generality, however compact the resulting representation might be. It is clear that the equivalent of dense pixel-wise correspondences will be needed in at least one stage of the overall computation, and any automatic feature finder is likely to require as much or more computation than optical flow. Furthermore, special case feature finders such as face detectors, eye-finders or eye-trackers, or face expression estimators may be designed to work on top of optical-flow type correspondence mappings and benefit both in speed and accuracy.

Storing the dense correspondence mappings can also be costly — two floats for every image pixel. Keeping them in their raw form amounts to spending memory/storage to gain speed. In a sense, then, NIN’s automated correspondence finding using a multi-scale, gradient-based optical-flow algorithm is the early vision subsystem of a memory-based computer visualization system.

### 2.1.3 Advantages and Disadvantages of Using Optical Flow

The main advantages of using optical flow to obtain the correspondence mappings are that it is automatic and general. Without it, automatic methods could not possibly be the default throughout the system; with it, a single user can quickly accomplish what might be a very tedious or virtually impossible task if he were restricted to manual methods.

Some of the disadvantages actually stem from the fact that optical flow does not make many assumptions about the images to which it is applied. One of the few assumptions that it does make (at least in its derivation) is that the disparities between the images are small. The correspondence problem is essentially ill-posed, and computing optical flow always depends on some form of regularization and/or relaxation. There is always a bias toward correspondences showing smaller disparities, and on large disparities, optical flow may entirely fail. Take the case of a human head rotated so that in one image the right eye pupil is located in the same absolute location as the left eye pupil is in the other. A general purpose algorithm based on comparing local differences will tend to match the left and right pupil in that location, even if that conflicts with remote evidence and leaves the other eyes unmatched. In the absence of other constraints, such as a pre-defined face model directing it to match pairs of eyes, as opposed to finding unspecified similarities, it is unlikely that an absolute-distance-biased algorithm will find a global solution correctly matching both eyes. In a case like two images of a picket fence, even a globally perfect solution may incorrectly match similar but displaced fence posts. Such aperture and fence-post problems are well-known in computer vision, yet admit no general solutions.

Further problems arise from occlusion. Consider a frontal view of a face with both ears visible, and then another with the head turned 45°. One ear will have disappeared behind eye and cheek, and the other’s appearance will have changed from a narrow, featureless oblong into a foreshortened variation on the rounded, intricate shape we typically picture as “human ear.” An algorithm that searches explicitly for left and right ears might give up on the disappeared ear, find the rotated ear by a method as simple as convolution with a template, yet only find its frontally-viewed counterpart using cues such as location relative to the eyes. The current state of the art might be fooled by a little finger held against the side of the head. An optical flow algorithm, on the other hand, cannot decide that one ear has disappeared and the other has almost completely changed its appearance. It must map the pixels of either frontally-viewed ear to something, sense or no sense. Or, computing flow the other way, from the rotated to the frontal image, almost all the pixels from the visible, ear-shaped ear might best be mapped into a narrow strip along the outside edge of its frontally-viewed counterpart. As for the pixels representing the other ear, there aren’t any. Optical flow alone cannot create something from nothing — call this dis-occlusion — and strictly speaking, there is no sound solution to the problem of one ear corresponding.

One way to find regions of miscorrespondence between two examples A and B is to warp the texture of A all the way into the shape of B, and then subtract the resulting image $B^A$ from B to form the difference image

$$D = B \ast B^A \equiv \max(0, \min(256, 127 + B - B^A))$$

(10)

Any very bright or dark spots in $D$ are regions of miscorrespondences in shape and/or texture (see fig. 13).

Another way of viewing regions of miscorrespondence is to select and graphically highlight a region of the trunk image and then highlight the corresponding region in a branch image, as in figure 14. And yet another way is to look for clumps of holes in one of the accumulator arrays used for forward warping (see fig. 15).
2.1.4 Remedies for Poor Correspondence

As a remedy to those miscorrespondences typically arising from aperture and fence-post type problems, one might consider either modifying one's automatic correspondence-finding algorithm, or else modifying the example images so that they better fulfill its requirements. An example of the former is to vary the number of levels in the pyramids used for coarse-to-fine flow estimation. An example of the latter is to pre-warp images of faces so that the left and right eyes are already more or less aligned.

As a remedy to correspondence errors arising from dis-occlusion (i.e., lack of correspondence due to missing information), one might try adding information to the system that is not readily extracted from the example images themselves. Rather than modify one's correspondence-finding algorithm, it makes more sense to incorporate this information on a separate level above that of correspondences. For instance, if an ear appears in some examples images of a face and not others, the representation of that face should prioritize the information available from those with ears when rendering that ear in a synthetic view.

I have built interactive tools for applying both remedies. The role of the user ranges from mouse clicking on a few key points in each example image, such as the eyes, to manually defining all correspondences using variable meshes or chains of polygons. Miscorrespondences can usually be corrected just by providing the optical-flow estimating function with a few hints, as it were. Non-correspondences (the kind of errors due to dis-occlusion) are more difficult to correct, and as a last resort, the user may define corresponding regions to be texture-mapped. Short of that, non-correspondences can often be handled in the same way as miscorrespondences. The most useful of these tools are semi-automatic, meaning that they depend both on user input and on some of the automatic functionality that succeeds unassisted in simpler cases. Ideally, the automatic functionality should assist the user as much as possible, while the user input should serve not to replace but to guide the automatic computation, with the net effect that the combination works better than either fully manual or fully automatic tools would alone.

The interactive tools include:

1. point-picking tools for labeling features or corresponding points in all the examples (e.g. the eyes or center of the head). The choosed points are used to determine affine transformations for image alignment, image translation ("panning"), scaling ("zoom"), cropping, and so forth.

2. contour-tracing an line-picking tools for manually putting edges or piece-wise linear chains in correspondence with each other.

3. triangle- and quadrangle-picking tools for defining and selecting polygons and chains of adjacent polygons within an image for patch-to-patch texture mapping or recomputation of optical flow on these regions as isolated from their surroundings.

4. correspondence-grids and mesh-warping tools for putting entire images in correspondence with each other by interpolating from the vertices of grids or quadrangular nets superimposed over all example images.

The simplest of these tools only requires the user to choose one or more corresponding points in each example image. In two dimensions, one pair of corresponding points defines a rigid translation. This may be used, for example, to align an arbitrary number of images around their specified centers. Two pairs define an affine transformation, that is, translation, rotation, and scale. Labeling both eyes in a set of images, for example, allows all the eyes to be aligned. This at least prevents the fence-post error of confounding left and right eyes, though it may introduce different errors; equalizing the intra-ocular distance, for instance, will tend to scale up images of turned heads, where the intra-ocular distance should be decreased due to foreshortening. Once the correspondences are found among the affinely transformed examples, the inverse affine transformations may be applied to take both the images and the correspondence fields back into their original reference frames. Adding a third point to the eyes, e.g. the tip of the nose or the center of the head, might be used to estimate the 3D rotation of the head, and thus allow one to equalize the scale independently from intra-ocular distances. Four or more manually chosen correspondence points can be used to define more complex pre-warping transformations. Insofar as these transformations are used to aid in constructing a correspondence mapping, the key insight is that optical flow works best between images with small disparities. Thus any set of invertible transformations can be applied to a set of images to reduce their disparities prior to computing optical flow, and if the inverse transformations are then applied to both the images and the computed flow fields, one ends up with correspondence mappings accurate over disparities too great for optical flow to accommodate by itself.

As integrated into my system, these point-correspondence tools require the user to pick
all the points by hand. However, there exist fairly reliable algorithms for automatically finding such facial features as the eyes, tip of the nose, corners of the mouth, etc. (Brunelli and Poggio, 1992; Hallinan, 1990; Stringa, 1992). Features not peculiar to faces, such as centroids, occluding edges, and T-junctions, can also be found. Thus, at least for geometrically simple objects or the face as a special case, these point-correspondence tools could be partially or completely automated.

The contour-tracing and linear-chain tools are better-suited for defining or editing the correspondences between line drawings or cartoons (see [18] [14]). Beier & Neely used weighted line-segments to define all correspondences in a linear morphing scheme between two images. A variation of this scheme could be used to blend isolated-point and dense-field correspondence techniques in NIN, but has never been fully implemented.

Corresponding regions, rather than isolated points or linear chains, can also be user-edited. Arbitrarily shaped polygons can be formed as chains of adjacent triangles, and selected either for straightforward texture-mapping between each corresponding triangle, or else for an isolated re-computation of optical flow. On the screen, the chains of triangles are color-coded for ease of identifying corresponding vertices and edges between two example images. The erroneous regions in the automatically computed optical flow are replaced by the edited correspondences; optionally, the boundaries between the old and newly edited regions are smoothed using a Gaussian blending function (see figure 16.)

Simply re-computing optical flow on an isolated region of miscorrespondences can sometimes correct them. The region in question is enclosed within polygons in both the domain and range image, and these polygons are cut out and embedded within identical, solid gray bounding boxes. Optical flow is computed between these boxes, and the new correspondences only within the selected polygonal region in the domain image is patched back into the global flow field (with or without smoothing). This procedure succeeds even on some of the simpler cases of occlusion, for instance, on a nose that occludes part of itself and part of the cheek as the head turns. The reason it succeeds has mainly to do with the removal of distracting or competing features nearby, although global effects should not be ruled out.

[[ Insert Figures showing corrected nose: example-A bad-Tween example-B nose-cut-A good-Tween nose-cut-B ]]

2.1.5 Interactive Grid Editing Tool

One disadvantage of such isolation and patching techniques is that patching together separately computed correspondence mappings may result in large holes or fold-over points in the range, correcting some errors only to create others. The alternative to picking isolated points or bounded regions is to edit the correspondence mapping as a whole. One efficient way to do this is to superimpose a deformable polygonal mesh over each image, where the correspondences at vertices are regarded as exact, and at all other points the mapping is interpolated from the nearest vertices. The accuracy of such an interpolated mapping is limited, of course, by the coarseness of the mesh; few vertices contain less information. Accuracy need not be sacrificed, however, if the deformable mesh correspondences are used as only an intermediate stage in a complete re-computation of optical flow. The mesh correspondences can be used to warp the example images so that they are more similar, decreasing the disparities over which optical flow would otherwise fail. If the pre-warping transformations are invertible, then, as outlined above, both the warped examples and the correspondences mapping constructed among these warped examples can be transformed back into the original, unwarped coordinate frame. I have implemented tools to do this using a variably coarse mesh of convex tetragons with bilinear interpolation from the corners. In experiments with photographic images of faces, this technique has been used to correct not only miscorrespondences such as eyebrows being matched to eyelashes, but also a significant degree of dis-occlusion due to head rotation and mouth-opening. The procedure consists of the following steps:

1. Compute optical flow from the reference example, or “trunk,” to another example image, the “branch.” (The pixel locations of the trunk constitute the initial domain of the correspondence mapping, while the sub-pixel locations of the branch constitute its initial range). (See figure 17.)

2. Test these correspondences by synthesizing images between the trunk and the branch. If satisfactory, stop; otherwise:

3. Superimpose a regular mesh of rectangles, say a 16 by 16 grid, on the trunk. On the branch, superimpose a mesh of quadrangles whose vertices are located at precisely the points computed by optical flow to correspond to the vertex locations in the trunk. Optionally, these branch vertices are constrained to move simultaneously with their corresponding trunk vertices, thus remain-
ing in in correspondence according to the current mapping. Other constraints, such as convexity and no fold-over, may be imposed here to regularize the edited correspondence mapping, and to allow principled rearrangement of several vertices at once.

4. Pre-warp the branch toward the trunk, using only the information in the mesh. Target vertices move to become aligned with the trunk vertices absolutely, and all points between vertices are interpolated. (See figs. 18, 20, and 19).

5. Compute optical flow from the trunk to the just pre-warped branch.

6. Apply the inverse of the pre-warping transformation to the range (i.e., the branch points) of the newly computed optical flow to transform it into a correspondence mapping from the trunk to the original branch image.

7. Repeat from step 2.

This semi-automatic technique has clear advantages over either fully automatic or fully manual techniques. The sparse, manually chosen correspondences serve as a guide for (re)computing the optical flow more accurately. The (re)computed optical flow contains much more detailed information than the (relatively sparse) correspondence mesh, and would be too tedious to construct by hand. Thus, the sparse, user-edited correspondence mesh is used as a guide for the automatically computed dense mapping and vice versa. Within this framework, any of the marking tools can be enhanced by pre-computing optical flow, and vice versa, any of them can be applied in conjunction with pre-warping techniques to improve the accuracy of the optical flow.

3 Experimental Results

Experiments have been performed with many different example sets, ranging in size from 1 to 28 images, and in dimensionality from 1 to 3 control parameters. The subject matter has mainly been photographed human faces, although experiments have also been done using photographs of full-length human figures, cartoon faces and figures, figurative and abstract paintings and drawings, and artificial test images. The most promising domain for this example-based approach to computer graphics is probably that of complex, non-rigid objects. It would seem that if these techniques can be applied successfully to human faces, then they will also work for many other deformable objects.

3.1 The 1-Dimensional Case

In the 1-dimensional case of only two example images—call them $A$ and $B$—there are many different ways to warp between them: using correspondences computed from $A$ to $B$, from $B$ to $A$, both ways, or even using an invertible map derived from both by iterative or other approximation procedures. Once the mapping itself is computed, inverse rendering processes are faster and tend to produce somewhat sharper in-between images, or “tweens,” near the endpoints. But forward rendering processes are not far behind in speed, and tend to produce sharper tweens in the middle. Given a single dense correspondence mapping, say from $A$ to $B$, the best image quality may actually be obtained using forward warping to render tweens near $A$ and inverse warping for tweens closer to $B$. Given correspondence mappings in both directions, the forward mapping $A \rightarrow B$ and the backward mapping $A \leftarrow B$, which in principle is equivalent to $B \rightarrow A$, the best quality may be obtained by forward warping from both endpoints towards the middle. If, however, there are errors in the forward mappings, then the resulting errors in the rendered images with increase linearly from these endpoints, possible showing up as a slight “jump” between those images rendered using $A \rightarrow B$ and those rendered using $A \leftarrow B$.

Generically, 1-dimensional image interpolation amounts to a simple cross dissolve $C$ between the gray levels of two warped intermediate images: one of the texture (i.e., the pixel-level information) of $A$ warped towards the shape of $B$, and one of the texture of $B$ warped towards the shape of $A$. We may represent this as follows, with $x$ as the interpolation parameter:

$$\Delta y \leftarrow \text{optical-flow}(A,B)$$
$$C(x) \leftarrow (1-x) \text{warp}(A, \Delta y, x) + x \text{warp}(B, -\Delta y, 1-x)$$

Ideally, $C$ is an exact reproduction of $A$ at $x = 0$, and at $x = 1$, $C = B$.

Correspondence mappings between only a single pair of images are also the easiest to improve using manual or semi-automatic editing tools. Selecting and editing corresponding points or contours by hand, however, rapidly becomes tedious with increasing numbers of control points and images. I implemented some of the correspondence editing tools to work on example sets three images at a time, but have rarely found them to be more useful than those which work on only two at a time.
In the 1-dimensional case, NIN's main advantage over manual morphing techniques such as Beier & Neely's is that it is automatic, thus saving the user from the tedious task of choosing correspondences by hand. Supplementing optical flow with interactive methods, such as the semi-automatic grid tool, allows a user to find correspondences across disparities that would be much more difficult using only manual tools. Nor does NIN depend on any precise registration of the image pairs. Rotation, scaling, sheering and other transformations meant to bring the example images into alignment with each other can be subsumed by NIN's more general pre-warping transformations. The pre-warping transformations can be inverted prior to rendering the final synthesized views, or left as is. As with ordinary morphing techniques, sequences of tweens from one example to the next can be chained together to create digital movies. For example, see 22.

Some 1-D experiments beyond simple morphing included using correspondences among three examples to create a non-linear interpolation scheme between any two of them. As indicated above and illustrated in figs. 5, there are several different ways to make a one-dimensional interpolation sequence from three views of an object. Call the example views A, B, and C. The simplest scheme is first to interpolate linearly from A to B, and then from B to C. If B does not seem like part of a "straight path" from A to C, then the interpolation sequence will appear to "turn a corner at B."

A second way is to compose two of the pairwise correspondence mappings into another pairwise mapping. If the forward mappings are \( f : A \rightarrow B \), \( g : B \rightarrow C \), and \( h : A \rightarrow C \), then in general \( g \circ f \neq h \), but \( g \circ f : A \rightarrow C \) may be used in place of \( h \) to interpolate linearly between \( A \) and \( C \) without sampling any pixels from \( B \). There will be no appearance of \( B \) or of any "corner turning" in the sequence, although the likelihood of miscorrespondences increases the more different \( B \) is from either \( A \) or \( C \).

A third way to interpolate from \( A \) to \( C \) is to use the corresponding points in \( B \) as control points for fitting non-linear curves between corresponding points in \( A \) and \( C \) (again using the maps \( f \) and \( g \) but not \( h \)). Depending on how the interpolation scheme is formulated, one can then interpolate from \( A \) to \( C \) by way of \( B \) to speak, with or without actually passing through \( B \). With three example images, it is straightforward to fit a quadratic mapping; with four or more, cubic splines can be used to create a smooth path going near but not necessarily through all the intermediate images. The sequence partially shown in figure 21 (http://www.ai.mit.edu/people/spraxlo/m/movingSnear.mp4) was created from eight example images using quadratic mappings, three examples at a time.

3.2 Rotation as a Special Case

Experimenting even with different example sets for 1-D interpolation, one can already compare the difficulties of varying one control parameter instead of another. Jumping ahead to experiments with two control dimensions, one can then make the comparison directly. Either way, what one finds with faces is that achieving realistic, glitch-free rotation of the head about the axis of the neck is generally more difficult than varying smile, eye movement, or any other expression. The only other variation as challenging as rotation is making the mouth open to expose whole teeth, the tongue, and so on. Making the eyes open from a completely closed view is similar, but less problematic. What is common to rotation and opening is dis-occlusion. In the full-frontal view with the mouth and eyes closed, there simply are no pixels representing the colors, textures, and shapes of teeth, pupils, or the ears as seen from the side. In practice, one obtains better results warping forward from open mouth to closed mouth (or inverse warping from closed to open), but even a bag of many such tricks do not constitute a general solution. Since rotation out of the image plane presents the most difficulties, is not specific to faces, and is easily quantifiable, I will most often discuss rotation as the main obstacle to achieving seamless, flawless view synthesis.

In principle, even the problems arising from dis-occlusion can be solved simply by starting from more example images. Exactly how many examples, and how they are best distributed in the input space, however, are empirical questions.

Thus first basic question is then: How large a rotation can be covered using only two example images?

Using fully automatic, optical-flow-based methods, we can generally make a face rotate about 10 to 15 degrees without problems. Using different, but also automatic, methods, Werner et al and others have shown images of inanimate object, not of faces) interpolated through 5°-rotations [32]. The first signs of failure with a rotating head usually show up in the nose or the ears. The eyes are often next; as noted above, if a face rotates far enough that an eye in one example image appears in the same location as the other eye in the other example, then in the absence of any facial model to prevent it, an automatic correspondence finding algorithm like optical flow will tend to confound those different
eyes as the same. Since the separation angle of
the eyes with respect to the axis of the heads
rotation is about 35 to 45 degrees, we should
expect fully automatic methods to fail at head
rotations of much greater than half of 40°.

In practice, optical flow correspondences be-
tween images of a rotating face produces notice-
able errors before this a priori bound of 20°, typ-
ically due to self-occlusion around the nose and
ears. Bright illumination can result in specular-
ities, or rather, spots of overexposed glare, on
the tip of the nose and the cheeks which cannot
easily be filtered out; like the eyes, these spots
might be erroneously identified with each other
by optical flow.

If one allows one example image half-way be-
tween the two end-points, and then composes
the optical flow correspondences to obtain one
mapping as in fig. 5, then one can usually in-
crease the total rotation to about 18 or 20 de-
grees without significant (i.e. noticeable) errors.

Using the interactive, semi-automatic tools
to edit the correspondences, one can achieve
greater rotation than this from only two exam-
ple images. Up to about 30°, the main problem
is to correct for dis-occlusions of the nose and
ear or the side of the head. The limiting factor
is almost always the dis-occlusion of the side of
the head, and it is not so much the perspectival
changes of shape as it is the exposure of whole
new features, such as hair on the side of the head
that is behind the ear in a fully frontal view.

Finally, the very best results are achieved by
composing interactively edited flow mappings
together. Using the resulting mapping and sam-
ping pixels from only the two example images
at the endpoints, one can then get about 40° of
rotation without any glaring distortions. At this
point, it is not the correspondence mapping as
such that is the limiting factor, but the linear
or near linear interpolation of this mapping in
the rendering process. That is, fine-tuning the
 correspondence mapping, even where its accu-
 racy can locally be improved, ceases to improve
the quality of the synthesized intermediate im-
ge; the non-linear changes due to occlusion are
too pronounced, and the textures appropriate
for the intermediate images appear in neither of
the images at the two extremes.

We must conclude, then, that even though we
can do much better than the Werner et al. [32]
and others, still there are limits. The best one
can hope for when sampling texture from only
two images of a face is about a 40° disparity,
regardless of how accurate the correspondences
are. The best one can hope for with fully auto-
matic methods (at least using optical flow, which
is the best technique we have found for automatic-
cally computing correspondences) is much less,
about 20°.

The obvious way to increase the range of rota-
tion is to start with more examples. It is a sim-
ple matter to chain 2-image morphs end-to-end
to interpolate all the way around a stationary
head or any other object (although the rounder
it is, the easier, again on account of the prob-
lems that result from abrupt occlusions.) And as
discussed above, more sophisticated interpo-
lation schemes involving splines may also be used
to create virtual views circumnavigating a head.
From our point of view, the interesting questions
are which of these results in the case of one con-
trol dimension can be generalized and hold up
in two or more dimensions.

3.3 The Multi-Dimensional Case

NIN's greater advantages, however, accrue in
the multi-dimensional case of three or more ex-
amples. As noted above, there are any number
of ways of interpolating between only two im-
ages. Getting from there to a general purpose
engine for synthesizing new views with several
independent dimensions is trickier, and this is
where our solutions to the four sub-problems of
section 1 all come together.

The case of two control dimensions with four
example images of a face is illustrative of NIN's
general functionality. Some of the practical con-
siderations have already been discussed in the
context of a typical user session (section 2.1.1).
Ideally, the four examples are naturally repre-
semed as the four corners in rectangular sub-
space of all possible views of one object, as in
figure 6, which shows the four possible combi-
nations of two orthogonal control variables, ro-
tation and smile. The representation as a rect-
angle is natural in that the two rotated examples
are rotated to the same degree, as are the two
smiling examples, one of them rotated, the other
not. It is trivial in such cases to scale the rect-
gle of variations into the unit square, and the
obvious bilinear interpolation scheme will then
map any point \((x,y)\) within it to a new view
constructed from the examples.

To carry this out this scheme, we first com-
pute correspondences between one of the exam-
ple images \(E_1\) and each of the other three exam-
ple images, \(E_i\) for \(i = 2,3,4\). In parameter space, \(E_1\)
is the reference image at \((x,y) = (0,0)\), \(E_2\) is at
\((0,1)\), and so forth.

For strictly forward warping, the space of
pixel locations in \(E_1\) is the domain of all the cor-
respondence mappings; for strictly inverse war-
ping, the domain of each mapping is the space of
pixel locations in \(E_i\).

Consider the case of forward warping, with

\[
\Delta y_{1 \rightarrow i} \rightarrow \text{optical-flow}(E_1, E_i)
\]
for $i = 2, 3, 4$, and $\Delta y_{1-i} = 0$. A warping from $E_1$ to the synthesized image $S$ may then be computed by bilinear interpolation as

$$\Delta y_{1-s} \leftarrow \sum_{i=1}^{4} b_i(x, y) \Delta y_{1-i}. \quad (11)$$

where

$$b_1 = (1 - x)(1 - y) \quad b_2 = x(1 - y) \quad b_3 = (1 - x)y \quad b_4 = xy.$$

The result $\Delta y_{1-s}$ is a mapping from the discrete lattice of pixels in $E_1$ to the continuous area of $S$. If $\Delta y_{1-s}(j, k) = (x_j - j, y_k - k)$, then the pixel value actually sent to the location $(x_j, y_k)$ in $S$ will be

$$S(x_j, y_k) \leftarrow \sum_{i=1}^{4} b_i(x, y) \times E_i(j + \Delta y_{1-s}^x(j, k), k + \Delta y_{1-s}^y(j, k)).$$

Note that there is nothing here to prevent many pixel values from being sent to one location $S$, nor any guarantee that all locations in $S$ will be sent values. Collisions must be resolved and holes filled using accumulator arrays and a second pass through the warped image data.

Inverse warping does precisely the opposite: for every pixel location in $S$, fetch exactly one value constructed from the examples. The inverse mapping from $S$ back to $E_1$ is

$$\Delta y_{1-s} \leftarrow \sum_{i=1}^{4} b_i(x, y) \Delta y_{1-i}.$$

Since the images $E_2, E_3,$ and $E_4$ are already in correspondence with $E_1$, we can compute warpings from $S$ back to the other three examples as

$$\Delta y_{i-s} \leftarrow \Delta y_{1-s} - \Delta y_{1-i}$$

for $i = 2, 3, 4$. Finally, we combine the examples using bilinear interpolation

$$S(x, y) \leftarrow \sum_{i=1}^{4} b_i(x, y) \text{warp}(E_i, \Delta y_{1-s}, 1)$$

No second pass is required to resolve collisions or fill holes, but the computation of the inverse mappings $\Delta y_{i-s}$ from $E_i$ to $S$ is more expensive in both time and memory than the forward mapping $\Delta y_{1-s}$.

The simple bilinear form here for the $b_i$ is due to the examples set's natural representation as occupying the four corners of a square. However, even if they do not naturally map onto the corners of a square, any four examples which have in common at least two independent variations, such as degree of rotation and degree of smile, can be used as the basis of a 2-dimensional subspace; it is only a matter of setting up a suitable approximation scheme. However, if all the examples where the face is rotated to one side also showed smiling, then rotation and smiling would not independent from each other and it would basically be impossible to use these examples to synthesize a view where the head is turned and the mouth is not smiling. Lesser difficulties arise when the examples show independent variations but not all the same degree. If the example of the rotated, smiling face were rotated only half as much as that of the face rotated, not smiling, then the natural shape of the synthetic image space obtained by linear interpolation alone would not be a rectangle but a quadrilateral as in figure 7. It would be possible to synthesize a view of the face smiling and rotated to the same degree as that of the rotated, non-smiling example, but this would essentially require some form of extrapolation and the resulting image would likely suffer some degradation for it. Finally, if the examples are equally spaced and do form the four corners of a rectangle, then for such a small problem domain, the various commonly used Green's functions in eq. 3 will all give very similar results.

Some results from 2-dimensional, 4-example trials are shown in figures 8 - 10. In short, our approach works - within limits. Perhaps the best way to characterize its capabilities is to note when and where these limits are reached.

The first limitation to inquire about is whether the same range of rotation achieved in the 1-D control space can be carried over to multiple dimensions. That is, if smiling and other expressions are added to a rotating face and controlled independently, will that effectively decrease the believable range of rotation? Qualitatively, this question is central to the basic viability of this approach to computer graphics: Does is scale well under added dimensions? Quantitatively, this question may be framed in terms of error analysis. It is clear that the errors in each dimension will be somehow compounded in the multi-dimensional case, but how?

### 3.3.1 Error Analysis in the 2D Case

The short answer is that some rotational range is lost, but not much. The deterioration in synthesized image quality is basically what we should expect from our consideration of the 1-D case in sec. 3.2, including the differences between forward and inverse warping. Let us again consider the bilinear case of four example images at the corners of a square (fig. 6) with forward warping. Since forward warping from $E_1$ to $b_1$ is least stable at the far endpoint $E_k$, and the effects of errors in the mapping increase linearly...
as we warp linearly toward $E_k$, we should expect the greatest errors to occur at the far corner diagonally across from $E_1$. It is evident in eq. 11 that the errors from the individual mappings $E_1 \rightarrow E_k$ combine additively, weighted by the bilinear coefficients $b_i(x, y)$. If each of the $E_1 \rightarrow E_k$ were equally (likely to be) erroneous, except for the identity $E_1 \rightarrow E_1$, which has zero error, then the (probable) error would be maximized wherever $b_1 = 0$, i.e., anywhere along the two edges of the square connecting the other three examples. However, the example farthest away from $E_1$ in parameter space, $E_4$ at $(x, y) = (1, 1)$, is also the farthest away from $E_1$ in image space (e.g., it is not only rotated or smiling, but both). We should therefore expect the errors in $E_1 \rightarrow E_4$ to be larger than those in either $E_1 \rightarrow E_2$ or $E_1 \rightarrow E_3$, so that the synthesized image with the greatest error will actually be that at $(x, y) = (1, 1)$. We should conclude that no rotational range is lost at $(x, y) = (1, 0)$, for this only reproduces the 1-D case with the two examples $E_1$ and $E_2$. The extreme at $(x, y) = (1, 1)$ will also be the same as in the 1-D case with only the two examples $E_1$ and $E_4$, but if the rotation (or other variation) is large, the quality is likely to suffer compared to the rotation-only case. Experience bears this out. The errors could be decreased if the example $E_4$ were rotated less than that $E_2$, but then we no longer have separable linear controls, where the number of degrees rotated is always directly proportional to the $x$ parameter. There is thus a trade-off: either reduce the range of rotation globally, or else sacrifice linear control.

3.3.2 Faces with Three Control Dimensions

The simplest set-up having 3 control dimensions is a set of 8 example images naturally represented in image space as the corners of a cube. The 4 corners on one side of the cube might be frontal images of a face with smiling and neutral expressions with the mouth both open and closed; the opposite side could then be the same expressions with the head rotated 20° to the left, as in fig. 11. The obvious parametrization and trilinear interpolation – with, say, the frontal, neutral, mouth-closed example at $(x, y, z) = (0, 0, 0)$ – then allows every point within the cube to correspond to an interpolated view of the face.

A typical demonstration of NIN with 3 control dimensions actually uses 12 example images, the same 4 frontal views rotated 25° both left and right. The image space may be conceived as two 8-example cubes adjacent to each other and sharing one side, namely the side whose 4 corners are the 4 frontal views. From a single reference image, there must be 11 correspondence mappings to the other example images; for rotations greater than about 20°, the best results are obtained if the mappings to the rotated examples are composed and/or edited from automatically computed mappings between less rotated examples. For instance, the correspondences used to make the virtual view sequence in figure 2 used composed correspondences from two intermediate rotated views in each direction, for a total of 28 examples. The extra 16 examples were used only to obtain correspondences, then discarded. The same idea has been used with video streams: correspondences are computed between successive frames (backward or forward) and composed into correspondences between frames that are relatively far apart; the intermediate frames and correspondences need not be stored [2]. As in the 2D case, both experience and quantitative error analysis indicate image quality suffering the most at the far corners, which correspond in the simple, trilinear scheme to the parameter values of $(x, y, z) = (1, 1, 1)$.

In terms of features, the problems areas of a face independently rotating, opening and closing its mouth, and changing from one expression to another, can all, once again, be traced back to occlusion and dis-occlusion. Shape distortions and glitches in image quality such as aliasing or slight pixelations tend to show up around the ears in patches of hair that are visible in some example views and not in others. In the example set used in figure 2, optical flow gave dis-correspondences in the plane of the neck which joins the head; head rotation causes 3D changes in shape there, but the texture is homogeneous, lacking in small features and visible gradations where a gradient-based optical flow algorithm can grab on or gain traction, so to speak.

The problem of a mouth opening and exposing the tongue and teeth is a complex one, having at least three different aspects: 1) shape: the lips are subject to pronounced changes in shape and texture, including stretching, compression and wrinkling, and foreshortening, as well as self-occlusion; 2) texture: in example images where the mouth is fully closed, there are no regions of "tooth" to be correctly put into correspondence with the toothy areas in the open-mouth examples or properly sampled for the shape, color, and texture of teeth; 3) shading: even under stable, uniform external illumination, the lighting inside the mouth varies depending on depth and on what the lips are doing, so that the tongue may effectively change color or disappear between different examples. One noticeable manifestation of these difficulties are small movements of texture on the sur-
face of the lips orthogonal to the direction of the overall motion; this is partly because in the actual motion of the lips, each tiny patch of skin does not necessarily take the most direct path to where it is going, and if it did, it might cross the paths of its neighbors. Like the scales on the back of a winding snake, these surface elements of lip stretch apart and press close to each other as the lips change shapes. Another manifestation is an apparent bleeding of color and texture information from the lips onto the teeth, which sometimes gives the appearance of a lipstick smudge. This is easily understood if one considers the problem of warping the texture of a closed-mouth example $A$ into the shape of an open-mouth example $B$. There are no pixels in $A$ representing teeth, and the only ones in the vicinity are red ones representing the lips; these are the pixels warped into the toothy areas of $B$, and the overall effect in the warped image ranges from a closed mouth with very thick lips to and open mouth whose interior is a solid blur of lip-toned red. If this warped image is cross-dissolved with $B$, the resulting teeth will appear pink. In practice, the optical flow correspondences from $B$ to $A$ (open to closed) will tend to be better than those from $A$ to $B$, and using $B$ as the reference image will produce better tzwems anywhere between $B$ and $A$.

The interactive correspondence editing tools need to be used most often to correct precisely these difficulties with mouths opening, hidden ears or hair becoming visible, and texturally plain regions such as the join of neck and jaw changing of shape due to foreshortening. Rotation of the head is still by far the greatest source of difficulties, and close to the rotational limit, small, local errors that the tools cannot fix begin to grow. Also as in the 2D case, the effect of adding another control dimension and doubling the number of examples seems to be to make these residual errors add together, not amplify each other. These errors remain local, and spatially separated distortions or glitches do not affect each other at all. Thus it should not be the least difficult to add another dimension for controlling eye opening or eye movement, more difficult to add another expression involving the mouth (e.g. roundness), and most difficult to add another dimension of whole-head movement, such as rotation up and down.

3.4 System Performance

As indicated above, this image-based approach to computer graphics may also be thought of as a memory-based approach to visualization; as seems to be the case with the brain, large chunks of short and long term memory are often spent to save time and complexity of computation, and the required transformations for generating virtual views are computed using trainable networks rather than through complex simulations of detailed 3D models.

Basically, the memory and storage requirements grow linearly with the number of examples, whereas processing time grows linearly with the number of control dimensions. The complexity of the learning problem increases exponentially with the number of dimensions in the input space, but the costs of the learning subsystem are dwarfed by those of the correspondence mapping and rendering subsystems.

The relationship of NIN's input and output image quality remains fairly constant with varying image sizes, partly due to the multiresolution processing built-in to the optical flow algorithm. Given sufficient memory and resolution in the example images, NIN is capable of synthesizing high-quality images in sizes typical of digitally processed feature films frames (1500 x 900 pixels) and even of PhotoCD (3K x 2K pixels). At present, the best commonly available technology for obtaining high-resolution digital images seems to be scanning 35mm (or larger) color film stock onto PhotoCD. But even consumer-grade flatbed scanners are capable of digitizing images of greater color-depth and much greater effective spatial resolution that those of (analog) broadcast television, provided one starts with reasonably good quality prints and pays proper attention to issues of signal processing. Indeed, even with 256 x 256 pixels frames, digital "movies" made with NIN sometimes appear artificial precisely because all the images in the moving sequence remain in sharp focus; the perceptual misce might be corrected, as it is in feature film animations, by adding in motion blur computed between successive frames.

3.4.1 Memory Requirements

The sizes of the images used most often during development, as well as to and to produce the figures in this thesis, ranged from 256 x 256 to 512 x 512 pixels, that is, from pixel dimensions nominally close to those of NTSC television (525 x 500 interlaced) to somewhat less than typical personal computer screens (640 x 480).

Let us take 300 x 300 pixel x 24-bit examples as typical; each one is 270 kilobytes. Optical flow is computed on the examples' gray levels represented as floats; each one of these is 300K, or a little more, if the edges are padded with a 5- to 10-pixel border to minimized boundary effects. (Since many floating point operations will be performed on this image data, some time might be save by also converting each of the RGB color channels to floats as well, but at
one 32-bit float per pixel per color channel, that would cost 1,080K per example. In NIN, this conversion is optional.) The automatically computed flow fields store two floats (x and y displacements) for each pixel in the reference example; compared to this, the storage required for the mesh vertices and other data used by the interactive tools is quite small, as is that for the approximation networks. The total memory requirement prior to rendering is then about \((7M + 8(M - 1))P\) kilobytes, where \(M\) is the number of examples and \(P\) is the number of pixels per example.

### 3.4.2 Computation of Correspondences

Using a multi-scale, gradient-based optical flow algorithm such as that of Lucas & Kanade [17], the time to compute optical flow between two gray scale images scales close to linearly in the number of pixels.

### 3.4.3 Rendering Speed

To render an image \(S\) from an \(N\)-dimensional network, pixels are typically sampled from \(2^N\) examples, and an equal number of correspondence mappings will be combined into the new mapping between the reference example \(E_1\) and \(S\). For inverse warping, there is the additional cost of computing the correspondence mappings from \(S\) back to the other examples,

\[
\Delta y_{i \rightarrow S} = \Delta y_{1 \rightarrow S} - \Delta y_{1 \rightarrow i}
\]

for \(i = 2, \ldots, 2^N\). Thus the total cost per image using the most straightforward form of forward warping is roughly \(O(2^N)\), whereas that of inverse warping is closer to \(O(2^{N+1})\).

Again using examples of input and output dimensions 300 \(\times\) 300 pixels \(\times\) 24-bits, average times for NIN to synthesize an image and display it on a 50 MHz SGI Indigo Elan are, using forward warping, 1.01 seconds in 1D (2 examples), 1.77 seconds in 2D (4 examples), 2.95 seconds in 3D (8 examples). Using inverse warping, they are 0.678 seconds in 1D (2 examples), 1.60 seconds in 2D (4 examples), 3.30 seconds in 3D (8 examples). Somewhat faster speeds can be achieved on an SGI Indy with the 174 MHz R4400.

### 3.5 Suggestions for Further Experiments

Past experiments with NIN indicate that some further control dimensions would be harder to add than others. Making the eyes open and close on a face that already opens and closes its mouth, smiles and frowns, and turns left and right would be very easy, whereas making the whole head also nod up and down is likely to prove difficult outside a certain range, when occlusion begins to be conspicuous. Changing the facial expression by varying the width of the mouth would likely fit somewhere between these in degree of difficulty. A more interesting additional dimension might be personal identity, that is, to a network one person’s face that rotates, opens its mouth, smiles, and also changes into another person’s face with the same degrees of freedom and ranges. The virtual face at any fixed value between the first and second real person’s face would also be able to rotate, open its mouth, smile, or whatever, and would be a controllable, artificial, yet photo-realistic face.

A second set of experiments would be to further the work done with full-length bodies. Here the progression in complexity would go from various stationary poses, such as the semaphores used by flag persons, to the positions and movements typical of such activities as aerobic exercises. Earlier work with cartoon characters indicates that characteristic motions themselves might be learned from an appropriately labeled sequence of views and varied along a continuum, as from walking to running [16].

A different kind of experiment would be to attempt to measure the quality of the synthesized views of faces and other things in cognitive terms. For although some computational measures of the quality of a synthesized images are possible, although the ultimate test is cognitive in nature. This makes the system a potentially interesting test-bed for algorithms relevant to image understanding and human perception (in particular, communication via facial expressions). Questions concerning optimal example sets and network structure are also interesting both from experimental and a theoretical points of view.

### 3.6 Suggestions for Further System Development

As we have seen, the quality and range of NIN’s output is limited mainly by that of the examples; given examples of sufficiently high resolution and dense distribution over the image space, NIN can interpolate synthetic images to any specification. Most of the costs – both human and mechanical – are directly proportional to the resolution and number of examples used, i.e., to the total number of pixels in the example images, and the simplest N-dimensional scheme, where the examples naturally fall on the vertices of the N-dimensional hypercube, gives a sort of soft lower bound on the required number of examples as \(2^N\). We can break this down further by noting that as the separations among the examples approach their maximum viable distances, more human intervention is needed to correct the automatically computed correspondence mappings. The need for intervention
may be reduced by adding intermediate examples, but with increased costs in storage and processing. Any strategy for improving NIN should thus weigh the costs of human intervention against the marginal costs of procuring, storing, and processing more example images. The latter, “mechanical” costs tend to increase linearly with the number of examples (or exponentially with the number of dimensions). The “human” costs, or the user’s time and effort, are more difficult to estimate, but whereas a more or less fixed amount of the user’s time goes into assigning parameters or otherwise preparing each example image, the time it takes the user to correct miscorrespondences when the distances between examples are stretched beyond the range of the automatic tools rises much more sharply.

The major trade-off, then, is between “human” and “machine time,” where the latter is apparently exponential in the number of dimensions. Two competing strategies for improving NIN are then to:

1. devise schemes that do not demand an exponential increase in the number of examples (or pixels) with each added control dimension (although the user might have to do more)

2. reduce the cost of user intervention per example image and automate as much as possible (the number of examples and all “mechanical” costs will increase)

Let us consider the first strategy first. The number of examples required to construct an interpolation network for spanning an N-dimensional subspace of the views is basically the Cartesian product

$$M = \prod_{j=1}^{N} M_j$$

(12)

where \( M_j \) is the number of examples it takes to span the \( j \)th dimension. For example, if every variation were represented as a continuum between only 2 endpoints, then the number of examples required would be \( M = 2^N \). To make an expressionless face rotate appreciably both left and right, however, requires at least three examples: the left and right rotated views plus one in the center. To add the capability of smiling, three more examples are needed for a total of six: left, right, and center views both neutral and smiling. Then to add a frown in the same dimension as neutral and smiling would require only 3 more examples (9 total), whereas to add an independent dimension like opening and closing the mouth (both neutral and smiling) would require 6 more examples (12 total), and to add both would require a total of 18 examples.

Now let us assume we have already constructed this 18-example network and want to add the independent dimension of making the eyes open and close. If all the desired gradations between eyes open and closed can be interpolated from only two examples in every pose, then the naive solution of 12 would require 18 more examples; if examples of the eye half-open were also needed, that would be 36 more examples (48 total). But what if the appearance of the eyes is totally independent of the state of the mouth? Then adding control over eye-openness would require only 3 more examples, one for each of the 3 example rotations (6 more, if eyes-half-open examples are needed). That is,

$$M = M_{rot} \times (M_{\text{mouth}} \times M_{\text{mouth}} + M_{\text{eyes}})$$

More realistically, smiling might make the eyes squint, while frown might make them appear “long,” but these variations might still be independent of whether the mouth is open or closed, and the result is less than exponential growth.

The problem then is to integrate the products of the separately controlled networks for synthesizing eyes and mouths into novel views of a single face. Perhaps the most promising approach would involve compositing the eyes as a separate layer over the rest of the face, and blending at the boundaries. I have not implemented such a scheme in full, and therefore have no real results to report. Some of the pieces of this kind of component separation and re-blending have been implemented in NIN’s texture mapping tools, Ezzat’s MS Thesis work [2], Graf et al’s system [13], and generally in “re-projection” techniques that map the 2D texture of an actual person’s face onto a generic 3D model. These results are encouraging, but have yet to be tried at high resolution in NIN, where careful attention will have to be given to boundary matching and blending. Besides the eyes, the insides of the mouth might also be synthesized as a separate layer and controlled by its own network. Although the tongue is usually at least partially obscured, its position relative to the teeth is an important cue for speech perception, and control of that should probably be a priority in creating digital talking heads.

Separating and selecting different layers is likely to require more user interaction, and could be aided by enhancements and extensions of the existing interactive editing tools. Some tasks of contour tracing and boundary regularization might best be accomplished using active “snakes” [15] [1].

The second alternative, to minimize the role of the user even if this requires using many more examples, calls for a thorough-going program of automation using further techniques from com-
puter vision, machine learning, and general rule-based heuristics. This would start with tools for automatically extracting and parametrizing views from large sets of potential example images. For faces, some suitable algorithms already exist for detection, recognition, feature tracking, pose estimation, and expression analysis (e.g. [7] [4] [29] [11] [2].

In order for any automatic method to correct its decisions or "learn" from trial and error, it must have some sort of evaluation function. At one end of the spectrum would be functions for detecting intrinsic flaws in the correspondence mappings, such as areas of fold-over, or even for finding areas of poor image quality (the criteria for the later would have to be relative and heuristic). At the other end would be functions comparing NIN's output with predictions (or even whole other images synthesized) from a generic or particular 3D model. Although constructing the 3D model would defeat the purpose of NIN for some applications, for others, like high-end film production, it might not.

Between these two ends of the spectrum, there are other ways of evaluating NIN's correspondence mappings and synthetic views with the aim of improving them. As previously explained, there are usually several different ways of synthesizing an image to fit a given set of parameter values, e.g., using forward or inverse warping. If the correspondence mappings are all correct, there should be very little difference among the images made in these different ways. Where there are pronounced differences, at least one of the mappings is wrong. These areas are easily isolated and may either be marked and brought to the attention of a user or else re-mapped using an automatic method using a criterion of convergence among the differently made synthetic view. NIN incorporates some fledgling steps in this direction, but nothing which yet merits being described as a separate tool.

Another useful extension to NIN would be to make a separate "play-only" application, that is, a smaller program that only plays image networks already created with the main application. If a remote side has already downloaded the player, then sending it a pre-constructed network module followed by a stream of control parameters could be a form of extremely low bit-rate image transmission.

Finally, the storage requirements of a network module might be further reduced by applying standard compression methods to both the example images and the correspondence vectors. The correspondences are currently represented by two 32-bit floats per pixel, but 8 bits gives a maximum displacement of 25.6 pixels at a res-

olution of 1/10th of a pixel, which should be more than sufficient. A standard method like LZW or even a variably lossy scheme like JPEG could then be applied to the 8-bit fields to further save space. Even greater savings might be realized if the dense, pixel-wise correspondence fields can be adequately encoded as sparse sets of coefficients for a spline-based or other compact representation.

4 Conclusion

NIN has been used to demonstrate the viability of a memory-and-learning-based approach to computer graphics, at least for such uses as creating the graphics components of virtual talking heads. Given sufficiently high-resolution examples, photo-realism in the synthesized views is attainable, allowing the user to create film-quality special effects. At lower resolutions, NIN may be used to produce view-based models of faces or simple creatures which are sufficiently compact and robust to be distributed and animated with small "player" programs.

The main advantages of NIN are that it is predominantly automatic and that it does not require very special example images or particular skill on the part of the user to create and animate realistically rendered, parametrically controlled figures. Within a central range (which varies with the set of example images, especially with the number and distribution of the reference examples), NIN's construction of a reusable image network module and synthesis of virtual views can be fully automated. The interactive, semi-automatic tools can then be used to extend this range to where it is bounded not by the accuracy of correspondences, but by more fundamental limits due to occlusion.

The main disadvantage of NIN is the variations in the synthetic views are limited in kind and extent by the number and range of a set of original example images. Even when examples are readily available, the time and space required for processing and storing them along with the correspondence mappings will in many cases eventually outweigh the advantages. In the case of creating and animating photo-realistic human or animal faces, however, this trade-off is somewhat moot, since traditional 3D graphics has not yet succeeded in rendering convincingly realistic faces from scratch. Within certain problem domains, then, such as creating and animating personal avatars or photo-realistic talking heads, NIN provides not only an alternative to traditional computer graphics, but also at least part of a state of the art solution.
References


Figure 1: Twelve example views used to make a 3D image net module. This module was used to make the digital movie of the following figure.
Figure 2: Thirty-six frames taken from a 100-frame digital movie made from the 12 example images shown in figure 1. Now viewable on the World Wide Web:
Figure 3: Examples of 1D interpolation along automatically computed optical flow fields.
Figure 4: The most general network with one hidden layer and vector output.

Figure 5: Three ways of using the correspondence mappings $f : A \rightarrow B$ and $g : B \rightarrow C$ to synthesize images $S$, which are interpolated between $A$ and $C$ with or without sampling texture from $B$: (a) interpolate straight from $A$ to $B$ and then from $B$ to $C$; (b) use the composed mapping $h = g \circ f$ to interpolate straight from $A$ to $C$, without passing through the shape of or sampling any texture from $B$; and (c) use the corresponding points in $B$ as control points for fitting non-linear curves between corresponding points in $A$ and $C$, quadratics for example, and then interpolate along them.
Figure 6: A simple 2D image interpolation scheme. The examples are assigned the parameter values $(0,0), (1,0), (0,1), and (1,1)$, which is equivalent to placing them at the four corners of the unit square. Any point $(x, y)$ within the unit square then maps naturally to a new image synthesized via bilinear interpolation among the correspondence fields from the reference image at $(0,0)$ to the other three. Choosing new parameter values outside the unit square causes in extrapolation, which produces caricatures at first, but eventually image disintegration.

Figure 7: Example sets need not fit the corners of a hypercube. Networks based on radial basis functions are better suited than multi-linear interpolation schemes to such cases.
Figure 8: One example of a 2D image net. The dimensions are rotation and smile. Above are the 4 examples; all the images below are synthetic, reproducing the examples in the corners and the half-way views between them.
Figure 9: Another example of a 2D image net. Although any point \((x, y)\) in the unit square maps to a unique and real-seeming expression, it is not clear what to name as the dimensions of variation.
Figure 10: Another example of a 2D image net. Although any point \((x, y)\) in the unit square maps to a unique and real-seeming expression, it is not clear what to name as the dimensions of variation.
Figure 11: A simple 3D image interpolation scheme. The examples are assigned the parameter values corresponding to the corners of the unit cube, and, similarly to the 2D case, any point \((x, y, z)\) within this space maps naturally to a new view.

Figure 12: Schematic showing NIN's basic inputs, outputs, and functional modules. The inputs in the dotted boxes are optional; here \(m = 0 \ldots M\) and \(k = 1 \ldots M\) index the examples, while \(r = 1 \ldots R\) indexes the synthesized novel views. Required inputs: Example images paired with example parameters. Optional inputs: Parameters to control the generation of novel views, plus any previously computed and saved correspondence mappings, mesh coordinate files saved from the grid tool, and other re-loadable outputs. Outputs (final products): individual synthetic views or digital movie sequences \(I_r\); these may be either naturalistic or for special effects. Other outputs (not shown): the reloadable, re-combinatory components of an image-network: dense pairwise correspondence mappings, cropped, pre-warped input images (optional) pairwise mesh coordinates, network specifications and weights, synthesized image sequences, log files (including parameter sequences), etc.
Figure 13: Regions of miscorrespondences in shape and/or texture can be found by warping the texture of a source image all the way into the shape of a target image and then taking the difference between the target and warped image. Above, the middle left image is the upper right image (source) warped to the shape of the upper left image (target); below that is the difference between target and warped image (see eq. 10). The right column is similar.
Figure 14: Corresponding regions may be highlighted to spot mis-correspondences.
Figure 15: An illustration of forward warping with hole-filling. The face $A$ in the upper left is being morphed into the face $B$ at the lower right; in reading order, the images are: the original source $A$, a tween half-way between $A$ and $B$, the raw pixel value accumulator (all pixel values were divided by 3 and clipped to the range 0 - 255 for display), the weight accumulator (note that the bright ridges are in the same places as those in the value accumulator), the normalized, hole-marked image made by dividing the value accumulator by the weight accumulator pixel by pixel, and, finally, the synthesized destination image made by filling in the holes in the previous image. If the correspondence mapping is perfect, this last should be the same as $B$. 
Figure 16: A region around the nose has been selected as a polygon made formed as a chain of four adjacent triangles. The triangle's vertices are color-coded; transparency and color-blending are used for visibility.
Figure 17: Optical flow in pseudocolor. The bottom left image represents the $z$-displacement between from the upper left to upper right image: the neutral gray in the background shows zero displacement; left displacement is bright, right displacement is dark. Similarly, the bottom right image represents displacements along the $y$-axis.
Figure 18: These images, all taken from windows in NIN, show some aspects the grid tool. As explained in the text, the corresponding grids in the middle images were used to warped the top images into those on the bottom; the bottom right image, for example, is more or less in the shape of the upper right image, rendered using pixel values from the upper left.
Figure 19: The grid tool for semi-automatic lay editing correspondences superimposes a quadrangular grid over two example images. The locations of the vertices in the image on the left correspond to those on the right in the current mapping.
Figure 20: A closer view of the results of mesh-warping. The deformed grid of the previous figure was used to bilinearly warp the frontal view face into the rotated pose, i.e., this image shows the shape of the rotated face with the texture of the frontal view.
Figure 21: The images in the top row are the 7 examples; those in the bottom row are tweens.

Figure 22: The simplest 1D chain $A \rightarrow B \rightarrow C$. In dictionary order, the first, fourth, and last images are the three examples; the others are tweens.
void rowWarpBilUnc ( unsigned char **dat, unsigned char **src,
                      float *x, float *y,
                      int dx, int dy, int dw, int dh,
                      int sx, int sy, int sw, int sh, dbl trs)
{
   register int ix, iy;                 register float pixv1, pixv2;
   register unsigned char *sp;          unsigned char *dp;
   int j, k, scanN = src[1] - src[0] - 1;
   int dx1 = dx + dw, tx = sx + sw - 2;
   int dy1 = dy + dh, ty = sy + sh - 2;
   float *px, *py, fx, fy;
   for (j = dy, dat += j, vx += j, vy += j; j < dy1; j++) {
      dp = &dat[j] + dx, pX = &vX++ + dx, pY = &vY++ + dy;
      for (k = dx; k < dx1; k++) {
         fy = j - trs * spY++;
         iy = IFLOR(fy);
         if (iy > ty) iy = ty, fy = 1.0F; /* bot edge */
         else if (iy < sy) iy = sy, fy = 0.0F; /* top edge */
         else fy -= iy;
         fx = k - trs * spX++;
         ix = IFLOR(fx);
         if (ix > tx) ix = tx, fx = 1.0F; /* right edge */
         else if (ix < sx) ix = sx, fx = 0.0F; /* left edge */
         else fx -= ix;
         sp = &src[iy] + ix;
         pixv1 = sp[1]*fx + (1.0 - fx)*sp;
         sp += scanN;
         pixv2 = sp[1]*fx + (1.0 - fx)*sp;
         pixv1 += (pixv2 - pixv1) * fy;
         *dp++ = IPRIMARY(pixv1);
      }
   }
}

Figure 23: A simple inverse warping function using bilinear pixel sampling. In practice, multiple source images are sampled and warped to one destination image at the same time.
Figure 24: A simple forward warping function using bilinear pixel splattering. In practice, multiple source images are sampled and warped to one destination image at the same time.