Learning style translation for the lines of a drawing

William T. Freeman
Mitsubishi Electric Research Labs and Artificial Intelligence Laboratory, MIT,

Joshua B. Tenenbaum
Department of Brain and Cognitive Sciences, MIT
and
Egon Pasztor
Mitsubishi Electric Research Labs and Media Laboratory, MIT

1. INTRODUCTION

An important task of pattern recognition is identifying or synthesizing stylistic variations on a signal independently of its underlying content [Omohundro 1995; Hofstadter 1995; Tenenbaum and Freeman 2000; Brand and Hertzmann 2000; Rose et al. 1998]. For example, one may want to differentiate stylistic variations in handwriting from the underlying content, or letters. We study this problem for the particular task of rendering line drawings. We want to allow the user of a drawing program to apply a new style to selected lines of the drawing, while keeping the "content", or the overall shape of those lines, constant.

Present systems typically provide such style control through parametric or interactive manipulations. For example, the thickness of the line can be adjusted, or the Fourier spectrum of the x-y coordinates of the line can be modified to yield a different line character ([Finkelstein and Salesin 1994; Unama et al. 1995; Bruderlin and Williams 1995; Hsu and Lee 1994]). User guidance is required in the rendering or font appearance models of [Winkenbach and Salesin 1994; Zongker et al. 2000]. 3-d object properties are used for the line-rendering methods of [Hamel and Strothotte 1999]. But even advanced parametric manipulations represent a small subset of the possible stylistic variations that a designer might consider. Figure 1 shows a set of line strokes each made in a range of different line styles. This range of styles would be very difficult to describe parametrically. Yet we would like to build a system which supports content-preserving translation across such different styles of lines. See, for example, [Borgman 1977], for examples of lines drawn in different styles.

We propose a learning-based approach to this problem. The system designer collects examples of many different lines, each drawn in the various styles that the system is to contain (the "training data"). An artist, not a programmer, decides what it means to render a particular line over a range of styles. The system refers to this training data in order to automatically translate lines made by a user into a particular desired style.

Our system has similarities to the interpolation-based style-modification system of [Rose et al. 1998]. Their "adverbs and verbs" correspond to what we call "style and content". However, we emphasize automatic operation at runtime; given new lines, the system must both identify the corresponding content (training line), as well as translate to a novel style. Our example-based approach in the domain of line modification is analogous to example-based texture transfer methods [Hertzmann et al. 2001; Efros and Freeman 2001].

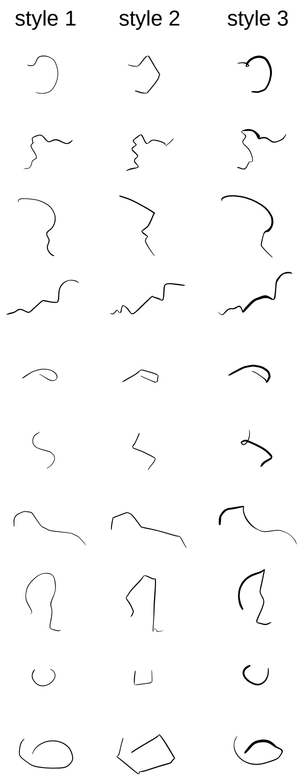


Fig. 1. Training data. An artist drew the same lines in several different line styles, which we call generic, jaggy, and brushy. 123 lines were in the training set, in each style. By computer these were scaled in size over 6 scales, ranging from 0.3 to 2.0, over 4 angles, and flipped left/right. All possible combinations of such manipulations ($6 \times 4 \times 2 = 48$) applied to the 123 hand drawn lines yielded a training set of 5904 line elements.

2. THE TRANSLATION ALGORITHM

We process drawings that are represented as a collection of lines. Each line follows some path on the page, with a thickness that can vary along the length of the line. We represent each line as a vector describing spline control points, with 3 numbers for each control point, indicating the x and y positions relative to the line origin, and a local thickness parameter. During a training phase, an artist draws a collection of line strokes, each in some number of different styles. We had an artist draw a training set of 123 lines, each in 3 different styles. Some elements of this training set, T, are shown in Fig. 1.

The user makes a drawing of lines in a style that we assume is represented as one of the styles in the training set, say style 1. Here is the crucial problem we need to solve at runtime: how should we use the training data to translate that line to style 2, the desired output style? The algorithms we discuss all involve two steps. The first is *fitting*, describing the input image in terms of the strokes in a similar training set style (style 1). The second step is *translation*, converting that description to one involving the lines of the output style (style 2, in this case).

Consider first a very simple algorithm, the *nearest neighbor* algorithm. In the fitting step, from all the lines in the training set for style 1, we look for the one closest to the input line.

(We measure distance between lines by the Euclidean distance in a vector representation of the line's spline control points and thicknesses, described below). Call the closest training example the *i*th line. The training set includes line *i* in each of the styles, and so, for the translation step, we simply output the training example of line *i* in style 2. (Note this is not the same as outputting the line in style 2 that is most similar to the input line).

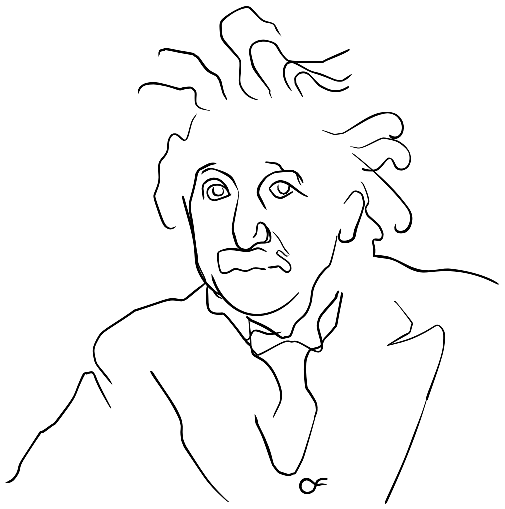
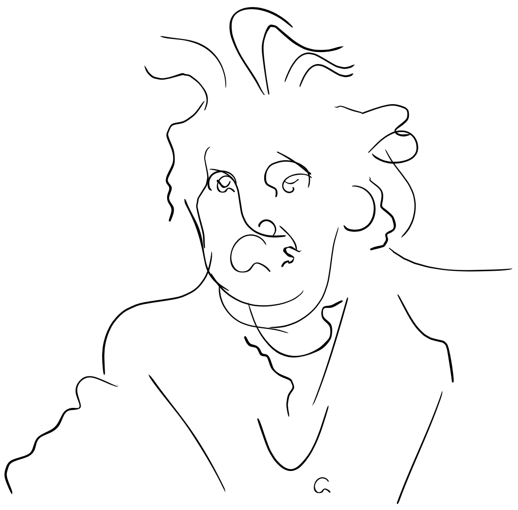


Fig. 2. Line drawing traced from photograph of Einstein.

Figure 2 shows a test image we wish to translate to a different style. Figure 3 (a) shows the nearest neighbor fit of the input drawing in a "generic" style of the training set. (In this and all subsequent examples, each line segment of the drawing is fit and translated separately.) Figure 3 (b) shows the drawing translated into a "jaggy" style using the nearest neighbor algorithm. This algorithm *translates* style well (i.e. the output lines (b) look like jaggy versions of the nearest neighbor fits (a)), because the nearest neighbor fit to each input line is represented explicitly in the artist-designed training set in both styles. But it doesn't *fit* the content of the input drawing well (compare (a) with Fig. 2), because the algorithm is only able to use lines that are already in the training set. Thus nearest neighbor is too limited to support content-preserving style translation.

To be of practical use, an algorithm must have the expressiveness to represent lines not explicitly put into the training data. Consider a second algorithm, the *linear combination* algorithm. In order to fit a broader range of lines, this algorithm fits the input line with the best linear combination of all the training lines in style 1. Define the matrix $\mathbf{A_s}$ to have as its columns each of the training lines of style *s*. If the input line is the vector $\mathbf{y}$, then we find the coefficients $\mathbf{x}$ yielding the least squares solution to $\mathbf{y} = \mathbf{A_1}\mathbf{x}$, or $\mathbf{x} = \text{pinv}(\mathbf{A_1}) * \mathbf{y}$. This pseudo-inverse solution (pinv) can be over-determined, if the number of linearly independent lines in the linear combination exceeds the number of line parameters to be estimated, or under-determined, if the converse is true. For each case, there is a standard least-squares solution; in Matlab, these are both incorporated into the function pinv. To render this input line in style 2, we simply output $\mathbf{A_2}\mathbf{x}$, a linear combination of lines in style 2 weighted by the same coefficients used to describe the input line in style 1.


(a)

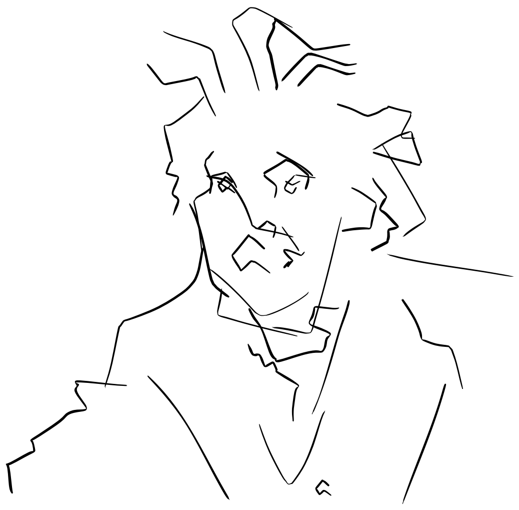
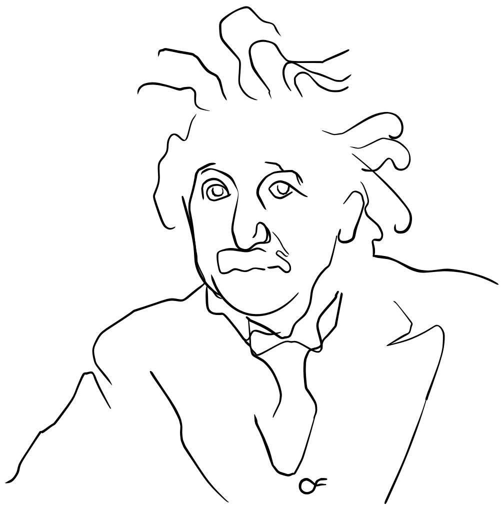

(b)

Fig. 3. (a) Nearest neighbor fit of original image to style 1. The fit does not describe the original image very well. (b) Nearest neighbor algorithm translation to style 2. The style translation is very good, being a look-up into the line strokes that the artist deemed were the style 2 translations of the best-fitting style 1 lines.

(We expect that other regression methods, such as radial basis functions (e.g. [Rose et al. 1998]) could also be applied).


(a)

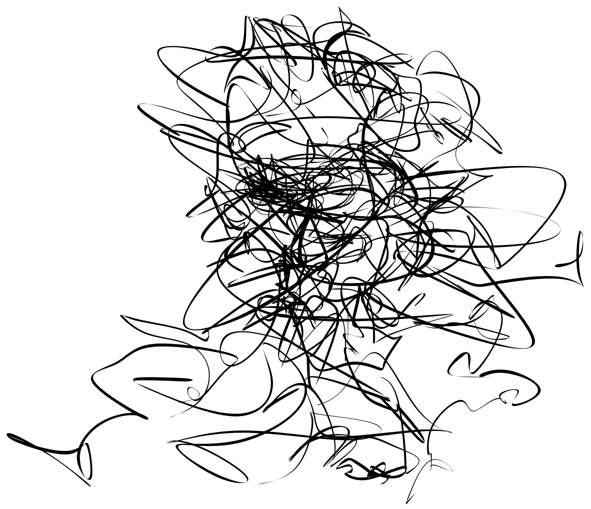

(b)

Fig. 4. (a) Linear algorithm fit of original image to style 1. The optimum linear combination of all training lines yields a high fidelity fit. (b) Linear algorithm translation to style 2. Taking a linear combination of the 5904 content elements in style 2 yields an unintelligible result; the translation is very poor.

Even with a small training set, the linear combination algorithm can fit an input line **y** quite well. Fig. 4 (a) shows the linear fit to our test image for style 1 of the training set T. The fit is very good; the result is almost indistinguishable from the original image, Fig. 2. However, the translation is terrible. Rendering $\mathbf{A_2}\mathbf{x}_i$ using the coefficients $\mathbf{x}_i$ found for each line of the test image gives Fig. 4 (b). In general, linear combinations of a large set of the style-translated training lines will not maintain their style, because characteristic features such as inflections or loops are scrambled when averaged together across many lines.

These simple experiments demonstrate a tension between “goodness-of-fit” and “goodness-of-translation” that confronts any example-based approach to content-preserving style translation. (This is similar to the “overfitting” problem in machine learning [Bishop 1995]). The nearest neighbor algorithm focuses on one extreme, achieving excellent style translation on a very limited set of lines which in general cannot fit the content of the input drawing. The linear combination algorithm goes to the other extreme, achieving an excellent fit to the content of input lines but in a form that cannot be generically translated across style.

Our solution to this dilemma is based on the observation that linear combinations of *sufficiently similar* lines often do maintain their style. This is particularly true if the vector representation of lines appropriately models their salient features, as discussed below in the section on representation. Then, if we only take linear combinations of similar lines, we may be able to both fit the input image, as well as translate those fit lines well to the desired output style.

This intuition is the basis for the algorithm we present here, the *K-nearest neighbor* algorithm for style translation. For an input line $\mathbf{y}$ in style 1, we find the K examples in the training set of style 1 that are most similar (in Euclidean distance) to that line. Let these K examples be the columns of a matrix, A_1^y. We then find the least squares solution $\mathbf{x}$ to $\mathbf{y} = A_1^y\mathbf{x}$, which describes the linear combination of the K training examples that best explains the input line. We translate this fit into style 2 by taking the same linear combination of the style-translated training examples. That is, we generate a matrix A_2^y by replacing each column of A_1^y with the style-translated version of that example found in the training set. Then $A_2^y\mathbf{x}$ describes the original input line $\mathbf{y}$ translated into style 2.

The K-nearest neighbor algorithm interpolates between the two extreme approaches presented above. When $K = 1$, it reduces to simple nearest neighbor. When K equals the size of the training set, it reduces to a simple linear combination. As we show below, intermediate values of K find an acceptable balance between goodness-of-fit and goodness-of-translation, yielding good style translations of an input drawing. In practice, it is easy to find a value of K that achieves this balance by starting at $K = 1$ and increasing K until a satisfactory fit is obtained. For the training set sizes we use here, we have typically set K between 3 and 8. In general, the optimal values of K will increase with the size of the training set. Figure 5 illustrates the fitting and translation tradeoffs with varying K for this drawing. We used $K = 6$ for all the results shown in this paper.

Figure 6 shows the results of K-nearest neighbor style translation for $K = 6$. Panel (a) shows the fit of the Einstein image using a linear combination of the closest 6 lines in the style 1 training set to each line of the original image. Taking a linear combination of several nearby examples allows us to *fit* the input data well. Panels (b) and (c) show the Einstein drawing translated into “jaggy” and “brushy” styles, styles 2 and 3, respectively. Combining only nearby examples allows the re-synthesized output lines to maintain their styles well. The fitting and translation took 4 seconds in a Matlab implementation on a

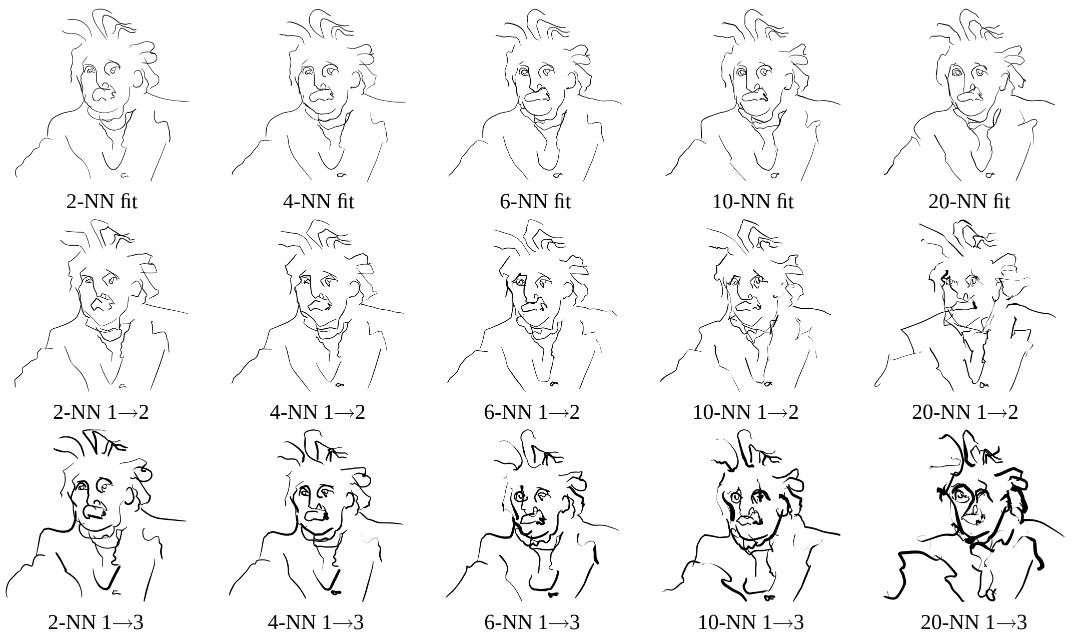


Fig. 5. Effect of parameter *K* in the K-nearest-neighbor fitting algorithm on fitting to style 1 (generic), translation to style 2 (jaggy), and translation to style 3 (brushy). The fitting accuracy increases with increasing *K* since linear combinations of more lines are used to synthesize the fitted lines. The translation accuracy decreases with increasing *K*, since linear combinations are taken of more and more disparate lines.

1Ghz PC.

Our algorithm is an application of a general learning method known as locally weighted regression [Cleveland and Loader 1995; Atkeson et al. 1997], where a weighting function which decays with distance restricts over which examples regression is performed. In the context of shape synthesis, Darrell used a related local learning approach to learn a mapping from end-point position to the rendered image of an articulated linkage [Darrell 1999]. In his work, as in this, simple methods are used to synthesize high-dimensional data from training examples, provided care is taken to only interpolate over the relevant training examples. The K-nearest neighbor approach can also be viewed as a locally adaptive generalization of the bilinear models used in [Tenenbaum and Freeman 2000] to separate style and content for various synthesis and analysis problems.

The remaining sections present the vector representation of lines that we use, how we generated the training set, more results of the K-nearest neighbor algorithm, and conclusions and future work.

3. LINE REPRESENTATION

The K-nearest neighbor algorithm requires a vector representation of lines, and in particular, one that encourages a linear combination of similar lines in a certain style to maintain the stylistic quality of those component lines. We first fit an open uniform cubic b-spline to a set of finely-sampled points along each line [Rogers 1990]. (The origin of this coordinate system is the mean position of all the finely-sampled points on the line). This allowed us to adjust the amount of data used to represent each line by fitting to fewer or

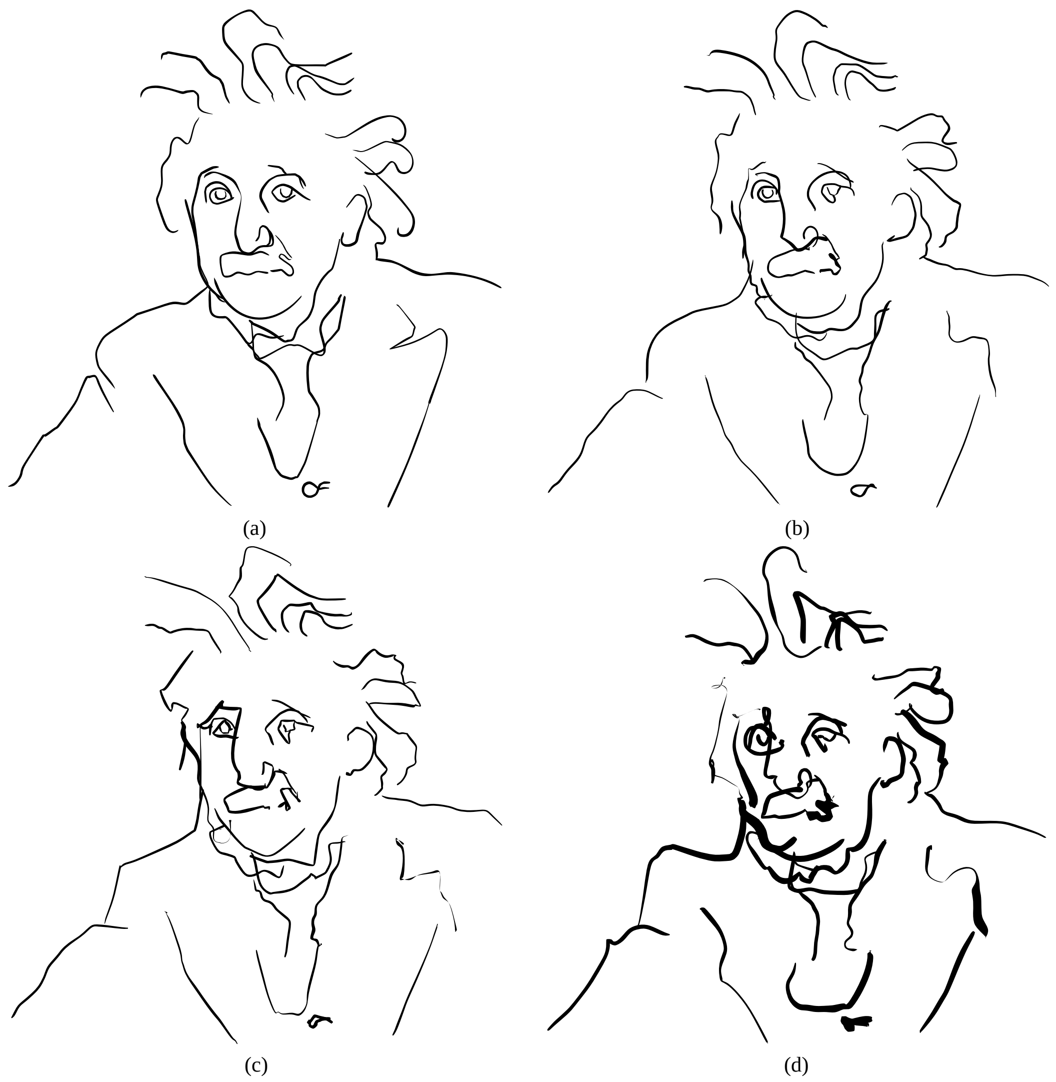


Fig. 6. Style translation using the K-NN algorithm. (a) Original line drawing. (b) 6-NN algorithm fit to image data in style 1. While not exactly the same as the original, the fit is quite faithful. (c) 6-NN algorithm image translation to style 2. In addition to fitting well, this algorithm translates well to the new style. (d) Image data translated to style 3, where the line quality of the training data in that style is still maintained.

more numerous control points. Because an open uniform cubic b-spline can be described as a sequence of connected cubic bezier curves, this representation can be easily translated into Postscript format, readable by many drawing tools. The K-nearest neighbor algorithm sees each stroke as a point in a vector space with one dimension for each component of each control point. If each stroke was represented as a spline with 20 control points, each an [x, y , thickness] triple, then the K-nearest-neighbor algorithm would see that stroke as a point in a 60-dimensional space. We subtract the mean position, averaged over the line, of the control points before fitting the lines to the training database (also mean zero lines).

After style translation, we addd back in the original means for each line.

The shape of a spline fit to data can be tuned by appropriately selecting the parametric value t assigned to each data point [Rogers 1990]. To best preserve the shape of each curve, we distributed t in a nonuniform fashion, such that greater intervals of t were clustered around areas of higher curvature. (Denoting arclength by s, the function relating dt/ds to the local curvature had the form of a saturating (sigmoidal) nonlinearity. Defining τ as the angular difference between sequential line segments, we used $\frac{dt}{ds} = 1 + \frac{\pi}{2} + \text{atan}(2.2 * (3.5 + |\tau|))$, for τ in degrees.) Splines fit with t distributed in this manner tend to have control points clustered around bends and corners, and thus the salient features of a line are represented more accurately than would be possible with an even control point spacing.

Ideally, to maintain style, stylistic features such as bends or loops should be put in register from one line to the next, and then averaged. If they are mis-registered, their average will blur and distort each feature. Adaptively placed control points will tend to devote the same number of control positions to an uneventful straight segment, independent of its length. Thus the stylistic features will be more nearly in register with one another between two perceptually similar lines. Figure 7 illustrates this. The average of the two similiar lines, using evenly spaced control points, loses a stylistic feature, the sharp corner. Averaging with more control points devoted to the corner region maintains the stylistic characteristic of the line. (In order for this averaging to work, the number of bends should be the same in both lines. However, if the number of bends were different, then the distance between the two lines would, in general, be large and the lines would not be selected for averaging by the K-nearest neighbors algorithm.)

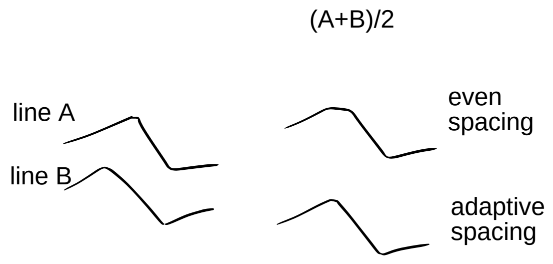


Fig. 7. Left: two lines in the same style which are similar to each other. We would like to represent them in such a way that their vector average another line in the same perceived style. Top: Average using a spline with equally spaced control points. The features of the two lines are blurred. Bottom: The average using a spline with adaptively space control points. This tends to place the interesting feature points into similar dimensions.

4. TRAINING SET

It is not realistic to expect an artist to draw enough examples to span the diversity of lines expected in the input image. We amplified the training set by taking into account operations, such as scale change, over which we expected the style translations to be invariant.

Simard et al. [Simard et al. 1994] use a modified distance metric, the tangent distance, to impose classification invariances in a classifier. Another different approach could be to rotate every line to a canonical position and scale, both during training and during processing before comparing lines. However, this can introduce problems for curving lines which have no canonical orientation.

We selected a different approach which allows for more explicit manipulation of line orientation and scale. Our approach to imposing invariances is to replicate the initial training set over all the desired invariances. In this case, we replicated each of the 123 hand-drawn training examples over 6 spatial scales, 4 rotation angles, and a mirror reflection, yielding a training set of 5904 line elements. Figure 8 shows the benefits both in fitting and style translation of replicating the training dataset over several different size scales. (A particularly large or small input drawing may require uniform re-scaling of its lines for them to be within the sizes of the training set lines.)

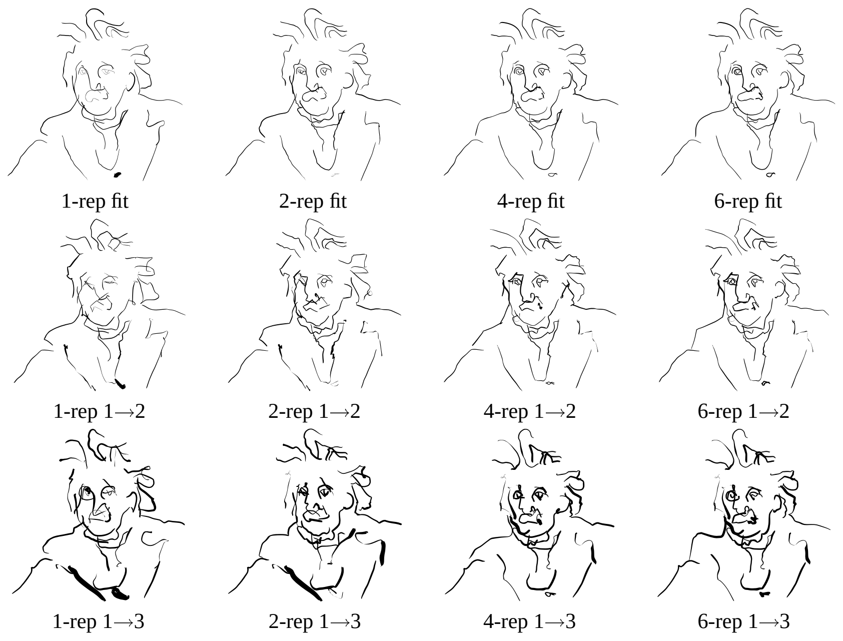


Fig. 8. Effect of the number of scale replications of the database on fitting and style translation (style 1⟶2 and style 1⟶3). The multiplicative scale factors by which the position and thickness parameters in the line representation were scaled are shown in brackets: 1-rep(lication), no scale replications: [1]; 2-rep: [0.7 1]; 4-rep: [0.5 0.7 1 1.4] ; 6-rep: [0.3 0.5 0.7 1 1.4 2].

5. RESULTS

Figure 6 shows the result of the K-NN style translation algorithm applied to the Einstein drawing. Figures 9, 10, 11, 12, show four other line drawings, their fits to the "generic" style of the the training set, and their translation into the "jaggy" and "brushy" styles, using the same training set and value of $K = 6$. In general, the style translation is good, balancing the goals of fitting the original with translating to the target style.

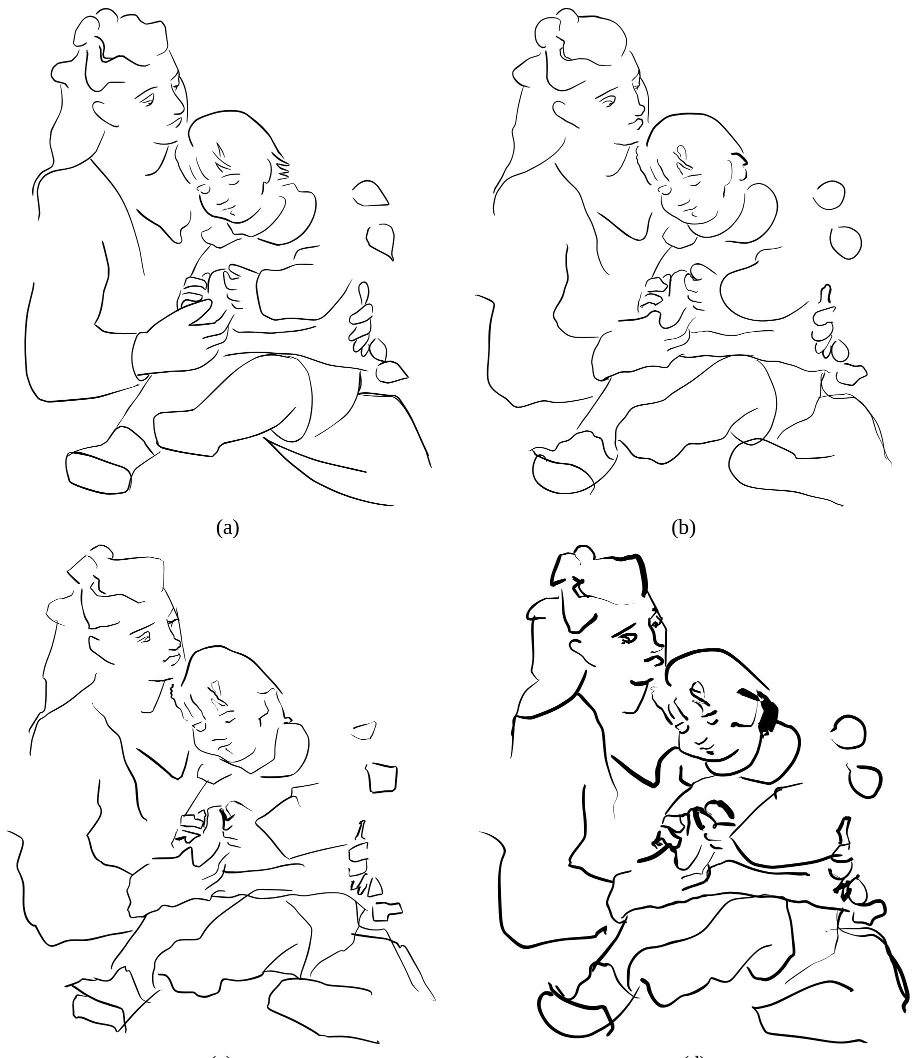


Fig. 9. 6-NN style translation example. (a) A tracing of the lines of Picasso's "Mother and Child". (b) Fit to style 1. (c) Translation to style 2, and (d) to style 3, both using the 6 nearest neighbor algorithm, and fitting assuming the original was in style 1.

Some insight into the limitations of the algorithm can be obtained by examining what appear to be mistakes in the human figures of Figures 6 and 9. Because the algorithm does not know anything about the shapes of fingers, noses, or other real-world objects depicted in these drawings, it may inadvertently distort the contours of these objects in ways which are stylistically consistent but physically unrealistic. Because each line is modeled independently, relational properties between lines such as parallelism or perpendicularity may not be preserved. In addition, this method cannot convert between a thick line and many cross-hatched lines, or utilize notions of 3-d shape in the style translation. Dramatic changes in line style may require more information about the context of the line within the drawing. Such limitations would be expected with any low-level approach to style translation that treats a drawing as just a set of lines with no additional structure, and are not a distinctive problem with learning-based approaches.

6. CONCLUSIONS AND FUTURE WORK

We have described a system for learning-based style translation in the domain of line drawings. The system is based on a simple and fast K-nearest neighbor algorithm, which is expressive enough to fit new input data, but constrained enough to translate that fit to different styles. Successful style translation also depended on finding an appropriate representation for the input lines, which allowed linear combinations of similar lines to maintain the consistent stylistic quality.

Our example-based approach has a number of advantages over conventional parameteric approaches: First, it can handle new styles drawn by the user, regardless of how difficult it may be to describe parametrically. A standard parametric approach to style translation such as power spectrum modification ([Finkelstein and Salesin 1994]) would have difficulty with the style translations presented here, because crucial shape information encoded in the phase component of the frequency signal is ignored.

Second, the repertoire of an example-based system can be easily extended by the user at any time. While our system produces good results with manageable training set sizes (around 100 user-entered examples per style), these results can certainly be improved with more training data. To add a new style, a user only needs to draw examples of the basic set of training lines in that style. A user who works mainly with drawings of a certain class of objects (e.g. faces or mechanical parts) can supplement the default training set with contours from these objects, in order to encourage the system to more faithfully preserve their essential shape under style translations.

Finally, the example-based approach may be generalized to modify the style of other kinds of graphics objects, such as the font of a letter or the movement style of an animated character. The problem of having to compromise between quality of fit to a wide range of signals and quality of style translation is just as crucial in these domains, and the simplicity and generality of the K-nearest neighbor algorithm makes it a promising approach in these other domains. Of course, as in the domain of line drawing, successful example-based style translation in these other domains will also require a representation of the input signals which allows linear combinations of similar inputs to maintain their style.

REFERENCES

Atkeson, C. G., Moore, A. W., and Schaal, S. 1997. Locally weighted learning. *Artificial Intelligence Review 11*, 11–73.

Bishop, C. M. 1995. *Neural networks for pattern recognition.* Oxford.

Borgman, H. 1977. *Drawing in ink.* Watson–Guptill publications.

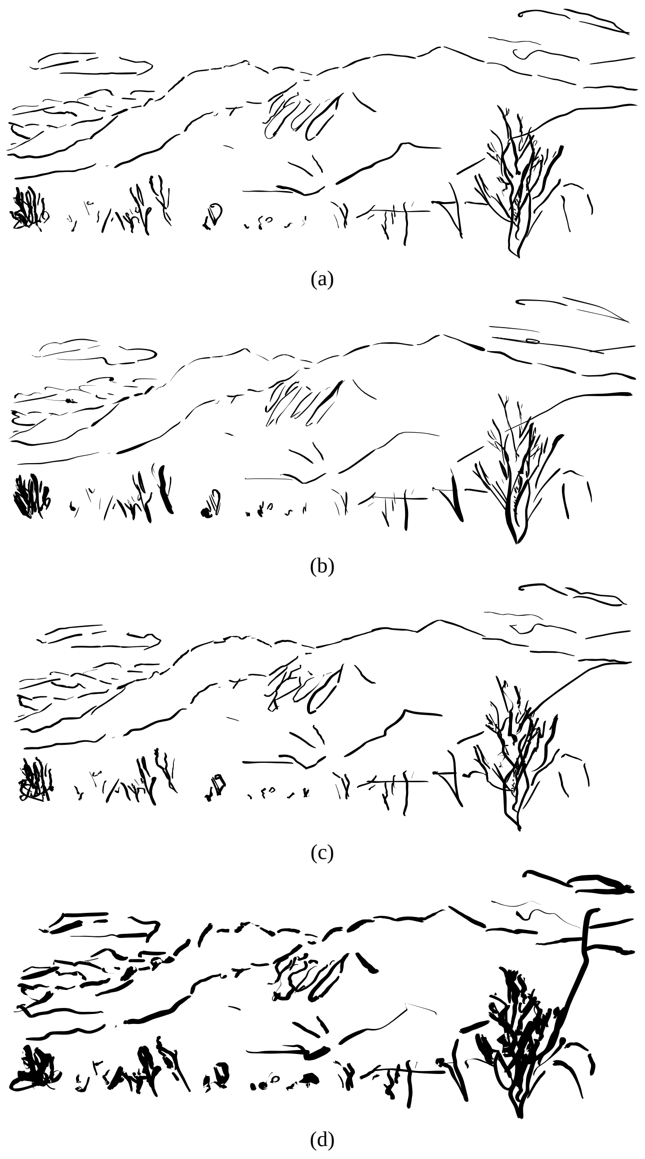


Fig. 10. 6-NN style translation example. (a) Original drawing. (b) Fit to generic training style (style 1) (c) translation from style 1 to style 2 (jaggy) (d) translation from style 1 to style 3 (brushy).

BRAND, M. AND HERTZMANN, A. 2000. Style machines. In *Proc. SIGGRAPH 2000*. 183–192. In *Computer Graphics*, Annual Conference Series.

BRUDERLIN, A. AND WILLIAMS, L. 1995. Motion signal processing. In *ACM SIGGRAPH*. 97–104. In *Computer Graphics* Proceedings, Annual Conference Series.

CLEVELAND, W. S. AND LOADER, C. 1995. *Smoothing by local regression: principles and methods.* Springer.

DARRELL, T. 1999. Example based image synthesis of articulated figures. In *Adv. in Neural Information Processing Systems*, M. S. Kearns, S. A. Solla, and D. A. Cohn, Eds. Vol. 11. MIT Press, 768–774.

EFROS, A. A. AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *ACM SIG-*

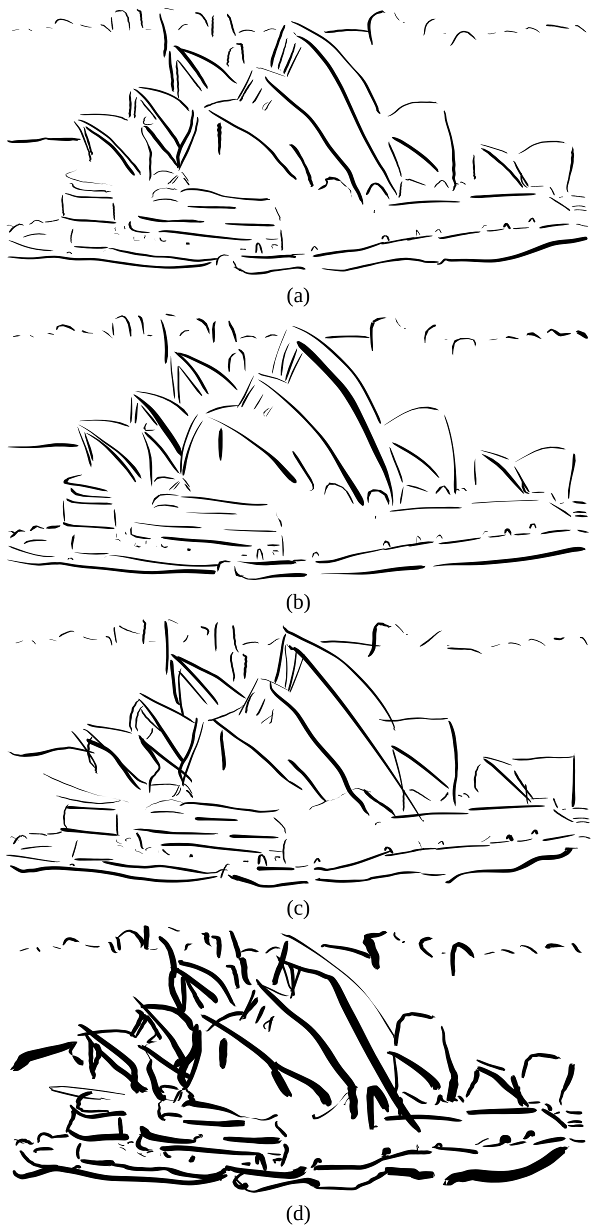


Fig. 11. 6-NN style translation example. (a) Original drawing. (b) Fit to generic training style (style 1) (c) translation from style 1 to style 2 (jaggy) (d) translation from style 1 to style 3 (brushy).

GRAPH. In *Computer Graphics* Proceedings, Annual Conference Series.

FINKELSTEIN, A. AND SALESIN, D. 1994. Multiresolution curves. In *ACM SIGGRAPH*. 261–268. In *Computer Graphics* Proceedings, Annual Conference Series.

HAMEL, J. AND STROTHOTTE, T. 1999. Capturing and re-using rendition styles for non-photorealistic rendering. In *Computer Graphics Forum*. Vol. 18. 173–182.

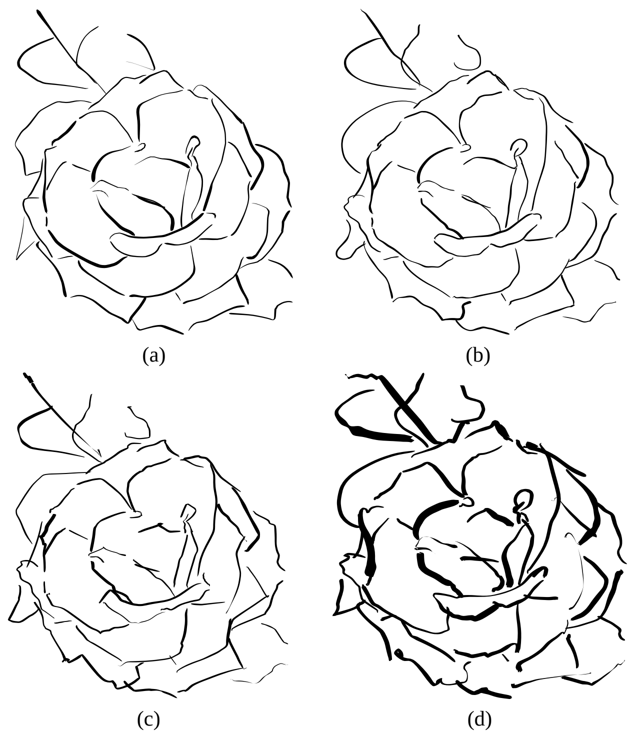


Fig. 12. 6-NN style translation example. (a) Original drawing. (b) Fit to generic training style (style 1) (c) translation from style 1 to style 2 (jaggy) (d) translation from style 1 to style 3 (brushy).

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *ACM SIGGRAPH*. In *Computer Graphics* Proceedings, Annual Conference Series.

HOFSTADTER, D. 1995. *Fluid Concepts and Creative Analogies.* Basic Books.

HSU, S. AND LEE, I. 1994. Drawing and animation using skeletal strokes. In *ACM SIGGRAPH*. In *Computer Graphics* Proceedings, Annual Conference Series.

OMOHUNDRO, S. M. 1995. Family discovery. In *Adv. in Neural Information Processing Systems.* Vol. 8. 402–408.

ROGERS, D. F. 1990. *Mathematical elements for computer graphics.* McGraw-Hill Inc.

ROSE, C., COHEN, M., AND BODENHEIMER, B. 1998. Verbs and adverbs: multidimensional motion interpolation. *IEEE Computer Graphics and Applications 18,* 5.

SIMARD, P. Y., CUN, Y. L., AND DENKER, J. S. 1994. Memory-based character recognition using a transformation invariant metric. In *IEEE 12th International Conference on Pattern Recognition.* Vol. 2. IEEE, Jerusalem, Israel, 262–267.

TENENBAUM, J. B. AND FREEMAN, W. T. 2000. Separating style and content with bilinear models. *Neural Computation 12*, 1247 – 1283.

UNAMA, M., ANJYO, K., AND TAKEUCHI, R. 1995. Fourier principles for emotion-based human figure animation. In *ACM SIGGRAPH*. 91–96. In *Computer Graphics* Proceedings, Annual Conference Series.

WINKENBACH, G. AND SALESIN, D. 1994. Computer-generated pen-and-ink illustration. In *ACM SIGGRAPH*. In *Computer Graphics* Proceedings, Annual Conference Series.

ZONGKER, D., WADE, G., AND SALESIN, D. 2000. Example-based hinting of truetype fonts. In *ACM SIGGRAPH*. In *Computer Graphics* Proceedings, Annual Conference Series.