Thirteenth Eurographics Workshop on Rendering (2002)
P. Debevec and S. Gibson (Editors)

Curve Analogies

Aaron Hertzmann[1] Nuria Oliver[2] Brian Curless[1] Steven M. Seitz[1]

[1]University of Washington [2]Microsoft Research

Abstract

This paper describes a method for learning statistical models of 2D curves, and shows how these models can be used to design line art rendering styles by example. A user can create a new style by providing an example of the style, e.g. by sketching a curve in a drawing program. Our method can then synthesize random new curves in this style, and modify existing curves to have the same style as the example. This method can incorporate position constraints on the resulting curves.

1. Introduction

Designing new styles is one of the most difficult tasks in non-photorealistic rendering (NPR). Typically, one is given a choice of a few rendering algorithms and the ability to adjust different parameters to these algorithms. While this gives the user many options, it may be difficult to find a desired style in this space. Moreover, it is unlikely that highly sophisticated or personal styles will be available.

In this paper, we present methods for learning line styles from examples. With this approach, an artist or end-user may simply draw strokes in the desired style; our system will capture aspects of their style and can generate new drawings in that style. For example, to design an outline style for a "nervous" character, one may draw a jittery stroke, and to design a style for a field of grass, one may draw an example of its jagged silhouette. In this paper, we adopt the terminology of Hertzmann *et al.*[13], and call the approach "curve analogies."

Our work is based on recent algorithms in image texture synthesis, where a new texture is generated such that every pixel neighborhood in the new texture looks like some neighborhood in the example texture. More recently, several methods have been developed that try to learn transformations between images, based on local neighborhood transformations. Our goal is to automatically learn how to generate an output curve from an input curve. This problem is surprisingly difficult, because neighborhood sampling patterns depend on the shape of the output curve. This means we must somehow generate the output curve shape, its neighborhood parameterizations, and its correspondence to the source curve simultaneously, even though each one depends on the others. Moreover, we must also account for possible rotations and translations in drawings. Our approach is to fix a curve correspondence in advance, and then iteratively reestimate curve shape, neighborhood sampling patterns and translational/rotational alignments.

This paper has three main contributions: first, We show how learning curve styles may be formulated as a texture synthesis problem on a parametric domain. Second, we show how curve analogies (i.e. transformations from input to output curves) may be learned from data, by learning specific statistics within curves and relationships between them. Third, we show how these algorithms can be used to capture hand-drawn styles of curves. Although we have focused on 2D curve synthesis for non-photorealistic rendering, we expect that this approach can be extended to other 2D signals, and even generalized to 3D surfaces.

In this paper, we use the term "curve style" to refer to the shape variations in a curve. Our work addresses one part of a larger problem of learning line art styles, including learning the placement of strokes, as well as the physical appearance of media (i.e. the buildup of graphite or paint). We focus on the specific subproblem of curve statistics, and assume that the input and output curves are continuous.

2. Related Work

There are many different ways to design curve styles for illustration. One general strategy is to manually program curve shape procedures, an approach taken by several authors[6, 21, 23, 27]. This approach gives great control to the programmer, but can be somewhat unintuitive. An alternative approach that we adopt is to create curve styles from examples. This approach has the potential to provide an intuitive interface — since the user is required only to draw an example of the desired style — but requires an ef-

fective procedure for capturing style. One approach is to simply warp an example stroke to the desired shape, as done by Hsu *et al.*[16], or to copy displacements to the target shape, as done by Finkelstein and Salesin[9]. These approaches give high quality results with relatively little effort, but may give a distorted result when the target shape is very different from the original shape, or when the base shapes are noisy. Such noise can be filtered out, at the cost of detail in the source shape. Essentially, such approaches require that all texture information resides in the high-frequencies of the texture curve, and all "sweep" information is in the low-frequencies of the target shape. These methods also produce noticeable repetition, unless many example curves are provided. Markosian *et al.*[22] describe a system for placing copies of small, hand-made strokes on a surface; in contrast, we focus on creating long, continuous curves. Computer vision and handwriting recognition researchers have put substantial effort into shape recognition[2, 3, 29]; to our knowledge, these methods have not been applied to modifying shapes or to line art synthesis. Some service bureaus will create a personalized typeface from a handwriting sample; our work generalizes this approach. Freeman *et al.*[11] present perhaps the first system for synthesizing line art based on example styles, and provide an inspiration for our work. This method produces high-quality results, but requires a large training corpus (e.g. over a hundred strokes), only handles strokes of roughly the same lengths as the example strokes, and does not incorporate other constraints (e.g. the method may change the topology of the drawing in undesirable ways). Our work addresses all of these problems. In work concurrent to our own, Kalnins *et al.*[18] describe a user interface for designing rendering styles for 3D models. Our work is very much in this spirit. Their method also includes a curve texture synthesis procedure, but may suffer the same problems as copying displacements (as described above). Our method could be used as a more-powerful curve synthesis procedure within their framework.

A few authors have described ways to learn stroke arrangements. Salisbury *et al.*[23] create hatchings interactively by copying from example hatchings. Chen *et al.*[5] describe a method for learning facial illustration styles from example. Their method does not produce very stylized lines; our method could be used together with theirs to produce stylized line drawings directly from images. Jodoin *et al.*[17] describe a system for learning hatchings as arrangements of strokes; our work is complementary, since we focus on the appearance of individual strokes.

This paper builds on recent work in example-based image texture synthesis. Freeman *et al.*[10] describe an algorithm for learning scene interpretations from examples, and, subsequently, Efros and Leung[8] demonstrated that simple recursive sampling can produce new textures similar to examples. Wei and Levoy[25] demonstrated several accelerations to this technique, including multiresolution synthesis. Efros and Freeman[7], Harrison[12] and Hertzmann *et al.*[13] demonstrated ways to generalize these synthesis algorithms by constraining them with source images. As demonstrated by Efros and

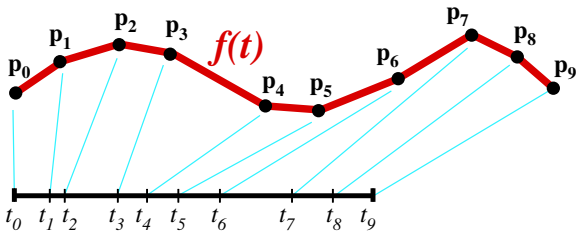


Figure 1: *We represent each curve as a polyline: a list of control points* $\mathbf{p}_i$ *connected by straight line segments. Each control point has a corresponding parameter value* t_i *and specifies that* $f(t_i) = \mathbf{p}_i$*. The curve is evaluated at a parameter value t by linear interpolation between the adjacent control points.*

Freeman and Hertzmann *et al.*, such methods can be used to produce line drawings by example. However, they can be quite slow, since they must process large amounts of training data (i.e. the set of image neighborhoods), often blur and blend curves, and produce resolution-dependent bitmaps (as opposed to line art in vector form). Furthermore, they do not capture rotationally invariant aspects of style. We show that these methods can be transferred to the curve domain, and that this representation is better suited to learning line styles. However, it should be pointed out that our method cannot yet work directly from source images but, rather, requires input curves in vector form.

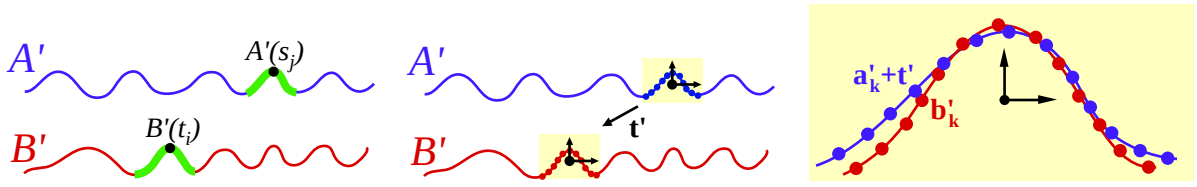
Wei and Levoy[26] and Ying *et al.*[28] describe methods for synthesizing 3D shape by synthesizing displacement textures. These methods model statistics of only the displacements, whereas our method models statistics of the resulting surface shape. Furthermore, we show how to learn transformations between shapes.

At a high level, our constrained texture synthesis algorithms are similar to variational curve modeling. For example, Kobbelt and Schröder[19] describe a subdivision scheme for fair interpolation of a set of constraints. Instead of using a predetermined smoothness energy function, we use a learned energy function; our single-resolution synthesis is analogous to a blurring filter, and our multiresolution scheme is analogous to variational subdivison.

3. Algorithms

In this section, we introduce a family of curve analogies problems and corresponding algorithms. The most general problem statement is as follows: given an example "unfiltered" curve A and example "filtered" curve A', we would like to "learn" the transformation from A to A', and apply it to a new curve B to generate an output curve B'. In short, we wish to generate B' to satisfy the relation

$$A : A' :: B : B' \tag{1}$$



(a) (b) (c)

Figure 2: *Curve synthesis problem statement. (a) Our goal is to produce a curve B' such that the neighborhood around every point $B'(t_i)$ "looks like" some neighborhood in A'. (b) Each neighborhood can be sampled with arc length spacing to get samples $\mathbf{a}'_k$ and $\mathbf{b}'_k$. In order to compare neighborhoods, we estimate a rigid transformation that aligns the samples. Here we show a translation $\mathbf{t}'$ that maps from the coordinate system of the A' neighborhood to that of the B' neighborhood. (c) Once the transformation is determined, the two neighborhoods can be compared in the same coordinate frame. Each neighborhood is sampled with arc length spacing, and the corresponding samples $\mathbf{a}'_k + \mathbf{t}'$ and $\mathbf{b}'_k$ are compared.*

We begin by describing the simplest problem statement and algorithm, and then show how to add more features to create more general algorithms. Specifically, we begin with a curve synthesis algorithm that randomly generates a B' curve in the style of an A' curve. In this case, there are no A or B curves and we are just randomly sampling curves from the distribution implied by A'. Next, we show how to synthesize a shape according to some constraints. We then show how to generalize this method to create curve analogies, and, finally, how multiresolution synthesis can give greater speed, quality and flexibility.

A 2D curve can be written in parametric form as a mapping from a segment on the real line to the plane $f : [t_{min}, t_{max}] \rightarrow R^2$. For convenience, we use piecewise linear curve representations, a.k.a. polylines (Figure 1). Each curve is represented internally by an ordered list of control points $(t_i, \mathbf{p}_i)$, where each parameter value t_i is unique in the list, and the maximum and minimum parameter values in the list define the range $[t_{min}, t_{max}]$. A curve is evaluated at a specific t value by linear interpolation†, and rendered by drawing line segments between each control point. The curve is modified by adding and deleting control points. A curve computed in this representation can easily be converted to other representations, e.g. by resampling or by creating a non-uniform B-spline from the control points.

3.1. Curve synthesis

Problem statement. The curve texture synthesis problem is as follows: given an example curve A', generate a new curve B' of a specified arc length $L_{B'}$ that appears "similar" to the example curve (Figure 2(a)).

Our single-scale curve texture synthesis algorithm is an adaptation of Efros and Leung's image texture synthesis algorithm[8] for processing shape. We use the primed symbols A' and B' in anticipation of Section 3.3, when the unprimed symbols will be introduced in a more general version of the algorithm.

In particular, we use A' to define a cost function over possible curves, and then attempt to generate a new curve B' with minimal cost:‡

$$E(B') = \sum_i \min_j d(B', t_i, A', s_j) \qquad (2)$$

This cost function states that we desire curves for which the neighborhood around each t_i location in B' "looks like" the neighborhood around some s_j in A' (Figure 2(a)). In other words, this cost function measures the "difference" between the local shape of B' around t_i and the local shape of A' around s_j. We restrict the t_i and s_j samples to be taken from a finite set of samples in their respective domains.

Of critical importance is the definition of the neighborhood distance metric $d(B', t_i, A', s_j)$. We represent each neighborhood as a set of K samples $\mathbf{a}'_k \equiv A'(s_k)$ and $\mathbf{b}'_k = B'(t_k)$, $k = \{1..K\}$, taken in unit arc length increments around $A'(s_j)$ and $B'(t_i)$, respectively. In addition, we use tangent features $\Delta\mathbf{b}'_k \equiv (\mathbf{b}'_k - \mathbf{b}'_{k-1})/\|\mathbf{b}'_k - \mathbf{b}'_{k-1}\|$, and $\Delta\mathbf{a}'_k \equiv (\mathbf{a}'_k - \mathbf{a}'_{k-1})/\|\mathbf{a}'_k - \mathbf{a}'_{k-1}\|$, in order to better capture the smoothness properties of the curve. We cannot directly compare the 2D positions of points on the curves B' and A' (as is typically done for image texture synthesis), since translating a curve in 2D should not affect this measurement. More generally, we would like the neighborhood comparison to be invariant to rigid transformations. Hence, we use the distance metric

$$d(B', t_i, A', s_j) = \min_{R', \mathbf{t}'} \sum_k w_k (\|R'\mathbf{a}'_k + \mathbf{t}' - \mathbf{b}'_k\|^2 + w_\Delta \|R'\Delta\mathbf{a}'_k - \Delta\mathbf{b}'_k\|^2)$$

† $f(t) = \begin{cases} \mathbf{p} & \text{if } (t, \mathbf{p}) \text{ is a control point} \\ \frac{(t - t_{lower})\mathbf{p}_{upper} + (t_{upper} - t)\mathbf{p}_{lower}}{t_{upper} - t_{lower}} & \text{otherwise} \end{cases}$
where "upper" and "lower" denote the parameter values of the control points immediately before and after t.

‡ More formally, we infer a density of curves $p(B')$ from A', and then sample from this density. The density is given by $p(B') = e^{-E(B')}/Z$, where Z is a normalization constant. In this sense, our algorithm models the statistics of curves as samples from $p(B')$.

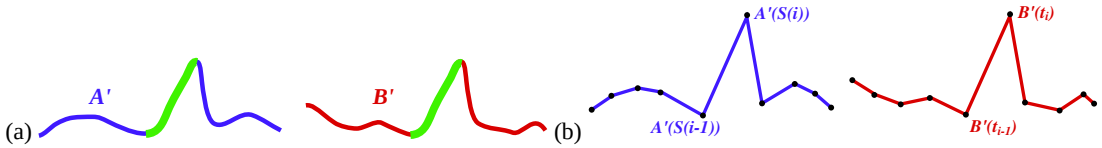


Figure 3: *Curve coherence. (a) We would like to copy continuous segments of the example curve A' to B'. We say that a curve segment is coherent if it was copied from a curve segment in the example curve. (b) For the polyline representation, we can test for coherence by testing if the line segment between $A'(S(i))$ and $A'(S(i-1))$ is identical to the line segment between $B'(t_i)$ and $B'(t_{i-1})$. We approximate this test by measuring the arc lengths $\|A'(S(i))-A'(S(i-1))\|$ and $\|B'(t_i)-B'(t_{i-1})\|$.*

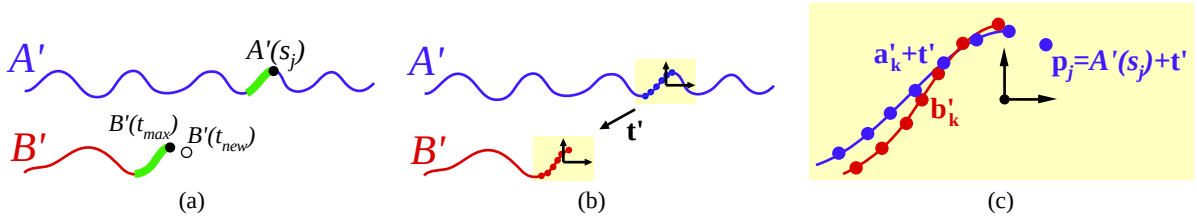


Figure 4: *First pass of the single-scale curve synthesis algorithm. (a) Our goal is to pick a value of $B'(t_{new})$ such that the resulting neighborhood around t_{new} matches some neighborhood in A'. (b) For each example neighborhood, we find the rigid translation that best aligns the two neighborhoods. Here we show a translation $\mathbf{t}'$. (c) The neighborhoods are compared in the transformed coordinate frame, and the candidate $\mathbf{p}_j$ for $B'(t_{new})$ is a copy of $A'(s_j)$ in the local coordinate frame.*

where the rotation R' and translation $\mathbf{t}'$ together define a rigid transformation, and w_k and w_Δ are user-defined parameter settings. If only translation invariance is desired, then R' can be fixed to be the identity matrix; this choice of whether to use rotation invariance depends on the texture being modeled.

Several authors have observed that, for image textures, copying patches of texture gives improved quality[1, 7, 13, 20]; we find the same to be true for curves. Hence, in a similar manner to Hertzmann *et al.*[13], we encourage copying coherent segments from A' to B'. Recall that B' is constructed from control points $(t_i, \mathbf{p}_i)$ so that $B'(t_i) = \mathbf{p}_i$. Let $S(i)$ be the source index for each control point in B', defined as $S(i) = \arg\min_j d(B', t_i, A', s_j)$. We say that a control point $(t_i, \mathbf{p}_i)$ in B' is *coherent* with the preceding control point $(t_{i-1}, \mathbf{p}_{i-1})$ if $S(i-1) < S(i)$ and the curve segment between $A'(S(i-1))$ and $A'(S(i))$ is identical to the curve segment between $B'(t_{i-1})$ and $B'(t_i)$ (Figure 3). We can test for this condition approximately by testing whether the corresponding arc lengths are the same, which, in this case, reduces to $\|A'(S(i)) - A'(S(i-1))\| = \|B'(t_i) - B'(t_{i-1})\|$. We penalize non-coherent control points by multiplying the distance $d(\cdot)$ by $(1+\kappa/D)$ where D is the approximate control point spacing in the curve and κ is a parameter setting.

Algorithm. We now describe a single-scale curve synthesis procedure; pseudocode for the algorithm is given at the end of this section. Since, at the outset, the start of the curve is entirely unconstrained, the algorithm first initializes $B'(t)$ to be an empty curve, and then copies a randomly-chosen segment of three consecutive control points from the A' curve to the B' curve.

The remainder of the B' curve is generated by nearest-neighbor sampling. We generate a new sample after the end of the curve, at $t_{new} = t_{max} + \Delta t$, where t_{max} is the current maximum t value in B', and Δt is a predetermined spacing (Figure 4(a)). Our goal now is to choose the value $B'(t_{new})$ that minimizes equation (2), while holding the rest of the curve shape fixed. As in the texture synthesis algorithms, the algorithm searches for the value $\mathbf{p}^*$ for $B'(t_{new})$ that best matches the resulting neighborhood of some neighborhood j^* in A', while neglecting the other terms in Equation 2 that depend on $B'(t_{new})$:

$$(\mathbf{p}^*, j^*) = \arg \min_{(B'(t_{new}), j)} d(B', t_{new}, A', s_j) \qquad (3)$$

In order to estimate these values, the algorithm iterates over each neighborhood (indexed by j) in the example curve A'. For each j, we compute the new value $\mathbf{p}_j$ for $B'(t_{new})$ that minimizes $d(B', t_{new}, A', s_j)$, and the corresponding cost $d_j = \arg\min_{B'(t_{new})} d(B', t_{new}, A', s_j)$. We set j^* to the index of the neighborhood with the smallest cost d_j, and $\mathbf{p}^*$ to the corresponding position $\mathbf{p}_j$. Finally, we set $B'(t_{new})$ to the value $\mathbf{p}^*$ corresponding to that sample, and the source index $S(i) = j^*$. We introduce randomness in the choice of the source location j^* in the same way as Efros and Leung[8], i.e. we uniformly sample from the set of s values for which the neighborhood measurement is within a fixed proportion $(1+\varepsilon)$ of the optimal choice.

We now describe the computation of $\mathbf{p}_j$ and d_j for each neighborhood j. This search is very similar to the one de-

scribed by Efros and Leung[8], except that neighborhoods must be compared under rigid transformations. First, the method computes the optimal rigid transformation $R'_j, \mathbf{t}'_j$ that aligns the existing neighborhood samples $\{\mathbf{a}_k, \Delta\mathbf{a}_k\}$ to $\{\mathbf{b}_k, \Delta\mathbf{b}_k\}$ (Figure 4(b)), using standard point-matching techniques[14, 15, 24]. This 2D transformation is computed in closed form. A causal neighborhood is used: the neighborhood points in $\mathbf{b}'_k$ after t_i are omitted (since they have not yet been computed), along with their corresponding points in $\mathbf{a}'_k$. Note that if only translation invariance is desired, then R'_j is set to be an identity matrix. With these values for j, R'_j, and $\mathbf{t}'_j$, the cost function can be written

$$E(B') \;=\; w_{new}\|\mathbf{p}_j - (R'_j A'(s_j) + \mathbf{t}'_j)\|^2 + C \qquad (4)$$

where w_{new} is the weight of $B'(t_{new})$ in equation (2), and C contains terms that do not depend on $\mathbf{p}_j$. Hence, the optimal choice for $\mathbf{p}_j$ is given by transforming the position of $A'(s_j)$ to the neighborhood in B': this gives $\mathbf{p}_j = R'_j A'(s_j) + \mathbf{t}'_j$ (Figure 4(c)). Finally, we evaluate the cost d_j for this value $\mathbf{p}_j$, and apply the coherence penalty, if any. We determine whether a candidate is coherent by the approximate test $S(i-1) < s_j$ **and** $\|A'(S(i)) - A'(S(i-1))\| < \frac{3}{2}\|B'(t_i) - B'(t_{i-1})\|$.

This process can be accelerated by precomputing the set of neighborhoods in A'. This sped up our system by a factor of five.

This method is not strictly optimal because the sampling pattern and the rigid transformation all depend on the position of $\mathbf{p}_j$, and vice versa. For curve synthesis, we have not found it necessary to directly account for this interdependence — we first estimate the sampling pattern, then the rigid transformation, then the value of $\mathbf{p}_j$. In later sections, we will describe cases where multiple passes are necessary.

The entire algorithm can be summarized in pseudocode as follows:

```
function SYNTHESIZECURVE(A'):
initialize B' with a randomly-chosen sequence of
        three control points from A'
while ARCLENGTH(B') < L_B'
    t_new ← t_max + Δt
    compute the curve samples b'_k around B'(t_max)
    for each s_j in a discrete set in the domain of A', do:
        compute the curve samples a'_k around A'(s_j)
        compute R'_j, t'_j that align {a_k, Δa_k} to {b_k, Δb_k}
        p_j ← R'_j A'(s_j) + t'_j
        compute d_j for the values of p_j, R'_j, and t'_j
        if p_j is not coherent, d_j ← d_j(1 + κ/D)
    d_min ← min_j d_j
    compute the set of candidates {j : d_j(1+ε) ≤ d_min}
    choose j* by uniform sampling from the candidates
    insert control point (t_new, p_j*) into B'
    S(i) ← s_j*
return B'
```

Once an initial shape for this curve has been generated, it can be refined by making successive passes over the curve. The algorithm for this is very similar to the first pass: it iterates over every control point in B', collects the full neighborhood around that control point, and finds the best matching full neighborhood in the example data.

3.2. Curve synthesis with constraints

Problem statement. We now describe a way to add constraints to curve synthesis. We allow two kinds of constraints:

- **Soft constraints** specify that the curve should pass *near* specific positions. Each soft constraint adds a term of the form $w_c\|B'(t_c) - \mathbf{q}_c\|^2$ to the cost function in Equation (2), where c is the index of the constraint.
- **Hard constraints** specify that the resulting curve *must* pass through specific positions $\mathbf{q}_c$. A hard constraint is specified as $B'(t_c) = \mathbf{q}_c$ and can be viewed as the limit of a soft constraint as $w_c \to \infty$.

We represent a constraint as a tuple $\langle t_c, \mathbf{q}_c, w_c\rangle$, where $w_c = \infty$ for hard constraints.

Note that a pair of hard constraints $\langle t_1, \mathbf{p}_1, \infty\rangle$ and $\langle t_2, \mathbf{p}_2, \infty\rangle$ specifies only that the curve must pass through $\mathbf{p}_1$ and $\mathbf{p}_2$ in that order; the arc length of the curve between those two points may vary widely depending on the style of the training data. In practice, the arc length will also depend on the search procedures used.

Algorithm. The algorithm for curve synthesis with constraints is quite similar to that without. In addition to the example curve A', a set of constraints $\langle t_c, \mathbf{p}_c, w_c\rangle$ is also provided. For this version of the algorithm, we assume that the first point on B' is constrained; we use the constraint to initialize the curve. The more general case can be handled by synthesizing forward from the first constraint, and then synthesizing backwards from the last constraint. The single-scale synthesis is not robust to arbitrary arrangements of constraints (for example, it has little chance of reaching a constraint that is far from the previous constraint), so we use the multiscale synthesis algorithm (to be described in Section 3.4) whenever constraints are present. We only introduce synthesis with constraints in isolation to make the presentation cleaner.

For each pass of the algorithm, we iterate over the same t values as before, but also insert control points (and perform synthesis) at the constraints. Unconstrained control points are synthesized exactly as in the previous algorithm. Control points specified by hard constraints are immediately replaced with the position of the hard constraint.

For control points with soft constraints, we must modify the search procedure according to the modified cost function. In the case of one soft constraint $\langle t_c, \mathbf{q}_c, w_c\rangle$ when $t_c = t_{new}$, equation (4) becomes

$$E(B') \;=\; w_{new}\|\mathbf{p}_j - (R'_j A'(s_j) + \mathbf{t}'_j)\|^2 + w_c\|\mathbf{p}_j - \mathbf{q}_c\|^2 + C$$

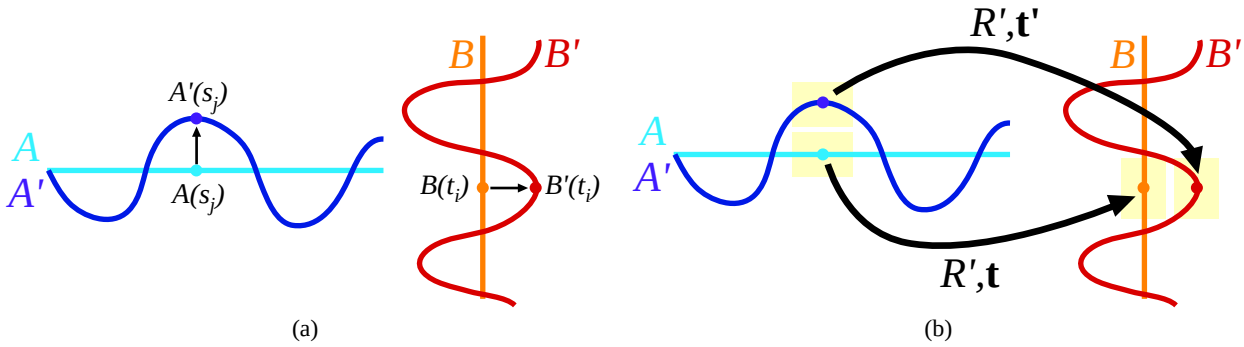


Figure 5: *Curve analogies problem statement. Here we show an example using rotation invariance, where the B curve is rotated 90 degrees from the A curve. (a) Our goal is to generate a B' curve, such that, for every point t_i, there is some point s_j such that: the neighborhood around $B'(t_i)$ looks like the neighborhood around $A'(s_j)$, the neighborhood around $B(t_i)$ looks like the neighborhood around $A(s_j)$, and the offset between B and B' matches the offset between A and A'. The offsets are measured between the neighborhood centers-of-mass. (c) These comparisons are performed with respect to the rigid transformation that best matches the curves. We use the same rotation matrix R' to map between the different curves, but allow different translations* $\mathbf{t}$ *(between A and B) and* $\mathbf{t}'$ *(between A' and B'). A different rigid transformation will be estimated for each candidate.*

where w_{new} is the weight of the target point; we always use $w_{new} = 1$ in practice. The overall outline of the algorithm is the same: we compute the set of candidates $\mathbf{p}_j$ and their associated costs d_j, and choose the candidate with the least cost. However, the candidates are now computed by optimizing the above quadratic equation:

$$\mathbf{p}_j = \frac{w_{new}(R'_j A'(s_j) + \mathbf{t}'_j) + w_c \mathbf{q}_c}{w_{new} + w_c} \tag{5}$$

The position of $\mathbf{p}_j$ is no longer a direct copy of the example position as in Efros and Leung's method — the constraint acts like a spring that "tugs" the value of $B'(t_{new})$ toward $\mathbf{q}_j$.

For a specific neighborhood and transformation, the cost in equation (4) acts like a soft constraint of the form $\langle t_{new}, R'_j A'(s_j) + \mathbf{t}'_j, w_{new} \rangle$. It is convenient to view this as a "virtual constraint;" in subsequent sections, we will add more terms that can be written as virtual constraints. In our implementation, we convert all energy terms into virtual constraints, and then compute $\mathbf{p}_j$ as a weighted combination $\mathbf{p}_j = \sum_c w_c \mathbf{q}_c / \sum_c w_c$, for each constraint c, real or virtual. The cost terms $\sum_c w_c \|\mathbf{p}_j - \mathbf{q}_c\|^2$ are then added to d_j. This unified treatment of constraints makes it easier for one piece of code to handle the different possible cost functions. Of course, our technique could be generalized to allow non-quadratic cost, for example, by computing $\mathbf{p}_j$ by numerical optimization.

3.3. Curve analogies

Problem statement. The curve analogies problem is as follows: given an example "unfiltered" curve A and example "filtered" curve A', we would like to "learn" the transformation from A to A', and apply it to a new curve B to generate an output curve B' (Figure 5(a)). In short, we wish to generate B' to satisfy the relation

$$A : A' :: B : B'$$

Of course, there are infinitely many plausible ways to interpret a training pair; in this paper, we pursue a "textural" form of analogy, similar to the image methods[7, 12, 13]. However, the shape and position relationships of 2D curves present some additional complications.

First, we need some way of representing the correspondence between A and A', and between B and B'. We do this by requiring the A, A' curves to have the same parameterization, and generating B' with the same parameterization as B. Hence, a point $A(s_j)$ corresponds to the point $A'(s_j)$, and $B(t_i)$ corresponds to $B'(t_i)$. Note that this places no constraint on the relative arc lengths of the curves. If the example curves do not have the same parameterization, then they are first reparameterized.

More importantly, we need some way to measure shape relationships between input and output curves. For example, one approach would be to represent the A' and B' curves as displacement vectors from the A and B curves respectively. However, this representation would be very sensitive to noise in the shape of the A and B curves — noise in the B curve would distort the resulting B'.

Intuitively, the problem can be viewed as simultaneously optimizing for three goals (Figure 5(a)):

- The *shape* of B' should "look like" the shape of A',
- The *shape* of B' should have the same relationship to the shape of B as the shape of A' has to A, and
- The relative *positions and orientations* of the B' curve should have the same relationship to B as the positions and orientations of A' do to A.

The first two goals directly correspond to the matching terms in the Image Analogies algorithms; we optimize these by using the same point and tangent features for B and B' as in Section 3.1. However, the third goal is new, and specific to shape modeling.

We measure the positional relationships by measuring the offsets between curves. We use neighborhood centers-of-mass as a sort of "summary" of the local neighborhood position; we experimented with several alternatives (such as using offsets between corresponding curve points) and found centers-of-mass to be the most effective. Let $\mathbf{a}_k, \mathbf{a}'_k, \mathbf{b}_k, \mathbf{b}'_k$ be neighborhood samples in the four curves around some t_i and s_j sample points, and define the centers-of-mass as $\bar{\mathbf{a}} = \sum w_k \mathbf{a}_k / \sum w_k, \bar{\mathbf{a}}' = \sum w_k \mathbf{a}'_k / \sum w_k, \bar{\mathbf{b}} = \sum w_k \mathbf{b}_k / \sum w_k, \bar{\mathbf{b}}' = \sum w_k \mathbf{b}'_k / \sum w_k$, and let R and $\mathbf{t}$ be the rigid transformation that aligns the neighborhood of A to B. As before, these samples are taken in unit arc length increments around the neighborhood centers; the $\mathbf{b}'_k$ values are resampled from the portion of the B' curve that has already been generated. We want the offset from A to A' to match the offset from B to B' (Figure 5(b)), which we express as

$$d_{OFFSET}(B, B', t_i, A, A', s_j) = \|R'(\bar{\mathbf{a}}' - \bar{\mathbf{a}}) - (\bar{\mathbf{b}}' - \bar{\mathbf{b}})\|^2$$

For example, when R' is the identity matrix and A' is above A, B' should be above B.

The new cost function for the output curve becomes

$$E(B') = \sum_i \min_j \left[d(B', t_i, A', s_j) + w_B d(B, t_i, A, s_j) + w_{OFFSET} d_{OFFSET}(B, B', t_i, A, A', s_j) \right] \quad (6)$$

where the three terms here correspond to the three goals described above, with weights w_B and w_{OFFSET}. The sum of the three desires is also multiplied by a coherence term. The weight w_B controls the relative importance of the B and B' curves in the matching.

As stated, this cost function allows different rotations and translations for the input and output curves. We have found it works best to use separate translations $\mathbf{t}$ and $\mathbf{t}'$ for matching A to B and A' to B', respectively, but to use the same rotation from A to B as from A' to B'.

Constraints may be included in the analogy just as in curve synthesis; note that the constraint t values correspond to the t values on the B curve.

In many cases, we are not interested in the shape of the B curve, only in its overall sweep. In this case, we set w_B to be zero, which corresponds to disabling the "second goal" described above.

Algorithm. The analogy algorithm is the same as curve synthesis, but with two important changes to the cost function in equation (2), due to the two new cost terms.

First, the A/B curve matching term (6) affects the curve alignment and the computation of d_j. For example, when generating the value of $B'(t_{new})$, if the neighborhood of $B(t_{new})$ has a sharp concave curve, then we prefer to choose an example where $A(s_j)$ also has a sharp curve. The optimal translations $\mathbf{t}$ and $\mathbf{t}'$ are computed for each pair of curves separately, and then a single R' is computed that aligns the pairs of curves.

Second, the curve offset term affects both the matching penalty d_j and the computation of the optimal $\mathbf{p}_j$. Since $\mathbf{p}_j$ is a candidate for $B'(t_{new})$, we can rewrite the offset term as

$$d_{OFFSET}(B, B', t_i, A, A', s_j) = m^2 \left\| \mathbf{p}_j - \frac{1}{m} \left(\bar{\mathbf{b}} + R'(\bar{\mathbf{a}}' - \bar{\mathbf{a}}) - \frac{\sum_k w_k \mathbf{b}'_k}{w_j + \sum_k w_k} \right) \right\|^2$$

where we have defined $m = w_j / (w_j + \sum_k w_k)$ is the proportion of mass of $\bar{\mathbf{b}}'$ due to $\mathbf{p}_j$, so that $\bar{\mathbf{b}}' = m\mathbf{p}_j + \frac{\sum_k w_k \mathbf{b}'_k}{w_j + \sum_k w_k}$. This is another quadratic error term, and can be converted into a virtual constraint

$$\left\langle t_i, \frac{1}{m} \left(\bar{\mathbf{b}} + R'(\bar{\mathbf{a}}' - \bar{\mathbf{a}}) - \frac{\sum_k w_k \mathbf{b}'_k}{w_j + \sum_k w_k} \right), w_{OFFSET} m^2 \right\rangle$$

and included when solving for $\mathbf{p}_j$. This also means that we do not begin the curve with random initialization. For the first sample in B', there are no $\mathbf{b}'$ values, and the virtual constraint reduces to $\langle t_i, \bar{\mathbf{b}} + R'(\bar{\mathbf{a}}' - \bar{\mathbf{a}}), w_{OFFSET} \rangle$.

The analogy algorithm terminates when the maximum t value in B' is the same as in A'. Multiple passes may be added to the curve analogy procedure as before, by considering full neighborhoods in B', and iterating over all control points in B'.

The single-scale analogies algorithm can be summarized in pseudocode as follows:

function SYNTHESIZEANALOGY(A, A', B, *constraints*):
initialize B' as an empty curve
for each constraint $\{t_c, \mathbf{q}_c, w_c\}$, **do**:
  insert $(t_c, B(t_c))$ as a control point into B'
for each control point $(t_i, \mathbf{p}_i) \in B$, **do**:
  compute samples $\mathbf{b}_k$ and $\mathbf{b}'_k$ around $B(t_i)$ and $B'(t_i)$
  for each s_j in a discrete set in the domain of A', **do**:
    compute samples $\mathbf{a}_k$ and $\mathbf{a}'_k$ around $A(s_j)$, $A'(s_j)$
    compute the $R'_j, \mathbf{t}_j, \mathbf{t}'_j$ that align $\{\mathbf{a}_k, \Delta\mathbf{a}_k, \mathbf{a}'_k, \Delta\mathbf{a}'_k\}$ to $\{\mathbf{b}_k, \Delta\mathbf{b}_k, \mathbf{b}'_k, \Delta\mathbf{b}'_k\}$
    get all constraints $\{\langle t_i, w_c, \mathbf{q}_c \rangle\}$ for t_i
    $\mathbf{p}_j \leftarrow \sum_c w_c \mathbf{q}_c / \sum_c w_c$
    $d_j \leftarrow w_c \|\mathbf{p}_j - \mathbf{q}_c\|^2$
    if $\mathbf{p}_j$ is not coherent, $d_j \leftarrow d_j(1 + \kappa/D)$
  $d_{min} \leftarrow \min_j d_j$
  compute the set of candidates $\{j : d_j(1+\varepsilon) \le d_{min}\}$
  choose j^* by uniform sampling from the candidates
  insert control point $(t_i, \mathbf{p}_{j^*})$ into B'
  $S(i) \leftarrow s_{j^*}$
return B'

In addition, we define a function REFINEANALOGY($A, A', B, \hat{B}'$) that performs the same

task as the above procedure, but using $\hat{B}'$ as the initialization of B'. This means that full, non-causal neighborhoods are computed instead of causual neighborhoods. Multipass refinement can be performed by first synthesizing with SYNTHESIZEANALOGY and then repeatedly applying REFINEANALOGY.

3.4. Multiresolution curve synthesis

There are a number of problems with the single-scale algorithms just described. First, we would like to be able to use large neighborhood sizes, which can be computationally expensive. Second, we would like to propagate constraints from the end of the output curve to the beginning, which may require several passes over the curve. Finally, there is a dependence on the Δt step size parameter that can produce poor results when processing analogies, since the step size is not directly related to arc length. We address these problems with a multiresolution synthesis procedure, based on the work of Wei and Levoy[25]. The idea is to generate a Gaussian pyramid of curves with the same statistics as a Gaussian pyramid of curves corresponding to A'. The Gaussian pyramid of curves is a list of L curves A'_ℓ such that $A'_{\ell-1}$ is a blurred and subsampled version of A'_ℓ, and $A'_L = A'$. This definition of curve pyramid is analogous to Gaussian pyramids used for images[4].

Problem statement. In multiresolution curve synthesis, we are given an example A' curve, and synthesize an output curve B'. This is done by synthesizing a Gaussian pyramid of curves from coarse to fine, by sampling from the distribution implied by the Gaussian pyramid of the A' curve. The final output B' is the finest level of the output pyramid.

The coarsest level of the output pyramid is a single-scale curve synthesis problem based on the coarsest level of the input pyramid. For the remaining levels of the pyramid, the relationships can be expressed as an analogy:

$$A'_{\ell-1} : A'_\ell :: B'_{\ell-1} : B'_\ell$$

where ℓ indicates the pyramid level, $\ell-1$ is the next coarser level above ℓ.

Any constraints placed on the output curve must be applied to the higher levels as well. However, because the coarser levels are smoothed versions of the finer levels, maintaining the constraints precisely may distort the texture in undesirable ways. For example, if the coarsest level of A' is a straight line, then this texture cannot meet constraints that the finer level can. The texture would have to bend to meet non-colinear constraints, which would not be a valid sample of the texture. We address this problem by softening the constraints: the weight w_c of each constraint $\langle t_c, \mathbf{q}_c, w_c \rangle$ is replaced with $(w_c^{-1} + (L-\ell)^{-1})^{-1}$ at level ℓ, where we define $\infty^{-1} = 0$ for hard constraints.[§]

Algorithm. These observations lead to the following multiresolution curve synthesis algorithm. First, compute the coarsest level of the B' pyramid by curve synthesis from the coarsest A'. In order to remove the dependence on the Δt for the remaining levels, we resample the curve at unit arc length intervals after synthesis. We then compute the remaining levels of the pyramid using the analogy algorithm from Section 3.3. Each finer level of the pyramid is initialized with the coarser level, and resampled to have a control point density proportional to the neighborhood sampling density before applying the analogy refinement. If any constraints are present, the corresponding blurred versions are used at each level.

The multiresolution synthesis algorithm can be summarized in pseudocode as follows:

function SYNTHESIZECURVEMULTIRES(A'):
create Gaussian pyramid of A'
$\hat{B}'_0 \leftarrow$ SYNTHESIZECURVE(A'_0)
$B'_0 \leftarrow$ RESAMPLE($\hat{B}'_0, 1$)
for each remaining level ℓ, from coarest to finest, **do:**
  $\hat{B}'_\ell \leftarrow$ RESAMPLE($B'_{\ell-1}, .5(1.2)^{\ell-L}$)
  $B'_\ell \leftarrow$ REFINEANALOGY($A'_{\ell-1}, A'_\ell, B'_{\ell-1}, \hat{B}'_\ell$)
return B'_L

The function RESAMPLE resamples a curve to have the specified arc length between control points.

3.5. Multiresolution curve analogies

The most general version of our algorithm is a direct extension of the preceding algorithms. In this case, synthesis at the finer levels of the pyramid can be viewed as a "generalized analogy:"

$$A_\ell, A'_{\ell-1} : A'_\ell :: B_\ell, B'_{\ell-1} : B'_\ell$$

The new curves simply add the corresponding terms to the cost function — each curve pair has a matching term, and there are two offset terms, to ensure that B'_ℓ has consistent relationships with B_ℓ and $B'_{\ell-1}$. Each of these terms creates corresponding terms in the neighborhood matching and the virtual constraints; the resulting algorithm is a trivial generalization of the previous algorithms. This involves generalizing REFINEANALOGY to take multiple pairs of A, B curves that samples neighborhoods for alignment and comparison in each corresponding A, B pair.

The multiresolution analogies algorithm can be summarized in pseudocode as follows:

[§] This softening comes from viewing a constraint as a Gaussian prior probability density over the position of $B'(t_c)$, with variance $\sigma_c^2 = w_c^{-1}$. Softening corresponds to convolving the prior with another Gaussian of variance $(L-\ell)^{-1}$; the resulting Gaussian density has variance $(w_c^{-1} + (L-\ell)^{-1})^{-1}$. Hard constraints have zero variance.

function SYNTHESIZEANALOGYMULTIRES(A, A', B):
create Gaussian pyramids of A, A' and B
$\hat{B}'_0 \leftarrow$ SYNTHESIZEANALOGY(A_0, A'_0, B_0)
$B'_0 \leftarrow$ RESAMPLE($\hat{B}'_0, 1$)
for each remaining level ℓ, from coarest to finest, **do:**
  $\hat{B}'_\ell \leftarrow$ RESAMPLE($B'_{\ell-1}, .5(1.2)^{\ell-L}$)
  $B'_\ell \leftarrow$ REFINEANALOGY($\{A_\ell, A'_{\ell-1}\}, A'_\ell, \{B_0, B'_{\ell-1}\}, \hat{B}'_\ell$)
return B'_L

3.6. Parameter settings

In practice, the parameters must be chosen carefully to yield good results. However, we found that a single set of parameters is effective for a large class of styles — we were able to use almost the same settings for all of the results shown in this paper, as described below. All distance measurements are in units of image-space distance.

When creating Gaussian pyramids, we resample each level to have a spacing of $D = .5(1.2)^{\ell-L}$ between samples. We set the neighborhood sampling spacing (i.e. the distance between adjacent $\mathbf{a}_k$ or $\mathbf{b}_k$ samples) to be $D/4$ for the level being synthesized. Note that the neighborhood size should be large enough to capture the details of the style in A'. We typically use neighborhoods of $n = 81$ or $n = 101$ samples wide. The sample weights w_k are approximations to a nonnormalized Gaussian density: $w_k = \binom{n}{k} / \binom{n}{(n-1)/2}$. Since this weighting depends only on k, the Gaussian has a larger spread at coarser pyramid levels. The weight of the coarse scale $w_{B'^{\ell-1}} = 2$. Typical parameter values include $w_{OFFSET} = 3.8$ for analogies and $w_{OFFSET} = 19$ for curve synthesis; $w_\Delta = 100$ or $w_\Delta = 100,000$; $\kappa = 1/2$, $\Delta t = 2D$ or $\Delta t = 4D$.

The weight of source B curve is set to $w_B = 2$ or $w_B = 0$ depending on whether the shape of the B curve is important. If $w_B = 0$, then only the position and orientation information of B is used.

4. Applications and Experiments

We now show several applications of the curve analogies framework. We used a simple 2D drawing interface to create the curves and specify styles and constraints, although our methods can be applied to any 2D curves (such as object silhouettes rendered from a 3D model). Each of the individual curves here took from a few seconds to about 1 minute to generate using a 1 GHz Pentium 4, depending on the length of the B' curve. This time could probably be substantially reduced by hand-optimizing the code. We used multiresolution synthesis in all experiments. Rotational invariance is used for all figures except where noted.

Figure 6 illustrates curve synthesis using translation invariance — a long, loopy curve is automatically generated in the style of a short one. Figure 7 shows this texture used in an analogy to apply this texture to the sweep of another curve. Since we are only interested in the sweep of the curve, we set w_B to zero.

Figure 9 demonstrates a situation where the curve of A is used, and thus w_B is set to 2. The algorithm generates a hand-drawn style based on the example, with different features at corners, vertical edges, and so on. Translation invariance is also used to capture orientation-dependent behavior. This means that, for example, the portions of B that were drawn right-to-left are treated differently from those that were drawn left-to-right. Both A and B were drawn clockwise. This explains the indentations that appear below the horizontal part of B': there are no examples of long horizontal, right-to-left lines in A, so the long horizontal, right-to-left lines are replaced with a series of corners.

Figure 10 illustrates the use of constraints to compose drawings. Processing each edge separately produces curves that do not intersect at the endpoints, and thus changes the topology of the drawing (Figure 10(b)). We created endpoint constraints for each set of nearby endpoints; processing with these constraints forces the corners to meet and preserves the source topology.

Figure 11 shows a larger composition, made by applying various curve style to elements of a line-drawn composition. Similar methods could be used to for rendering silhouettes extracted from 3D models in hand-drawn styles.

Figure 12 shows another example where the shape of the source curve is used, and thus w_B is set to 2. Here, we create a filter that makes the shape less rigid and more flowing. For example, the bend at the beginning of the "M" is copied to the beginning of the "B."

5. Discussion and Future Work

We have presented a technique for learning transformations of 2D shapes from examples. This method can be applied to a wide variety of input shapes, and thus we expect this method to be useful whenever hand-made stroke styles are desired, such as in hatching and 3D silhouette rendering. There are many potential avenues for improvement and enhancement:

Learning parameters. Our current system includes an unfortunate number of user-tuned parameters. It would be more desirable to fit a parametric model that automatically learns all parameters from the data. However, it is not at all clear what model would be appropriate for representing arbitrary styles; in particular, traditional time-series models often have difficulty representing sharp features without using a very large number of parameters (and thus requiring a vast corpus of training data).

Alternative representations. The polyline representation does not represent smooth curves very efficiently; it should be straightforward to translate our methods to a smooth curve representation (e.g. B-splines). Furthermore, such a

representation would be much faster, since fewer control points would be required to represent a curve. However, such a system would be more complicated to implement, and might have difficulty capturing sharp features, unless they are explicitly included in the curve representation. It would also be interesting to experiment with displacement vectors[9, 18]. We speculate that they may give good results in some cases.

Drawing systems. Our system can be viewed as a first step toward a larger system that generates many curves or begins with images or 3D geometry as input, such as in the work of Kalnins *et al.*[18] and Jodoin *et al.*[17]. For example, our system currently assumes that every input curve produces exactly one output, and thus cannot learn styles with broken curves, such as the overdrawing common in sketchy styles. Additional types of constraints could be added, in the form of new cost terms. For example, we could specify constraints on interactions between curves or between a curve and an image.

Additional features. Additional features of the curve — such as pressure, thickness, and brush texture — could be easily added to the synthesis as additional features in the curves A', B' (although more training data may be required in order to capture greater variability). Closed curves can easily be modeled by allowing neighborhood sampling to wrap around the curve.

Animation. Curve analogies can be used for defining line styles for non-photorealistic animation. If stroke temporal coherence is desired, then corresponding terms could be added to the energy functions. However, whereas flickering in full-frame painting styles may be very distracting, flickering in line-drawn animation is often acceptable or even desirable, since it gives a sense of life to otherwise still characters. Many classic line-drawn animations did not use temporal coherence of the form that has recently become a goal of NPR research.¶

3D signal processing. We believe that the curve analogies approach could be extended to learning probability distributions over 3D surfaces, for recognition and 3D signal processing applications.

References

1. Michael Ashikhmin. Synthesizing Natural Textures. *2001 ACM Symposium on Interactive 3D Graphics*, pages 217–226, March 2001.

2. Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape Matching and Object Recognition Using Shape Contexts. *IEEE Trans. Pattern Anal. Machine Intell.*, 24(2), April 2002. To appear.

3. Andrew Blake and Michael Isard. *Active Contours*. Springer, 1998.

4. P. J. Burt and E. H. Adelson. A multiresolution spline with application to image mosaics. *ACM Transactions on Graphics*, 2(4):217–236, October 1983.

5. Hong Chen, Ying-Qing Xu, Heung-Yeung Shum, Song-Chun Zhu, and Nan-Ning Zheng. Example-based Facial Sketch Generation with Non-parametric Sampling. *Proc. International Conference on Computer Vision*, 2001.

6. Cassidy Curtis. Loose and Sketchy Animation. In *SIGGRAPH 98: Conference Abstracts and Applications*, page 317, 1998.

7. Alexei A. Efros and William T. Freeman. Image Quilting for Texture Synthesis and Transfer. In *Proceedings of SIGGRAPH 2001*, pages 341–346, 2001.

8. Alexei A. Efros and Thomas K. Leung. Texture Synthesis by Non-parametric Sampling. In *IEEE International Conference on Computer Vision*, pages 1033–1038, September 1999.

9. Adam Finkelstein and David H. Salesin. Multiresolution Curves. *Proceedings of SIGGRAPH 94*, pages 261–268, July 1994.

10. W. T. Freeman, E. C. Pasztor, and O. T. Carmichael. Learning Low-Level Vision. *Intl. J. Computer Vision*, 40(1):25–47, 2000.

11. William T. Freeman, Joshua B. Tenenbaum, and Egon Pasztor. An example-based approach to style translation for line drawings. Technical Report TR99-11, MERL, February 1999.

12. Paul Harrison. A Non-Hierarchical Procedure for Re-Synthesis of Complex Textures. *Proc. WCSG*, 2001.

13. Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image Analogies. *Proceedings of SIGGRAPH 2001*, pages 327–340, 2001.

14. Berthold K. P. Horn. Closed Form Solution of Absolute Orientation using Unit Quaternions. *Journal of the Optical Society of America*, 4(4):629–642, April 1987.

15. Berthold K. P. Horn, Hugh M. Hilden, and Shahriar Negahdaripour. Closed Form Solution of Absolute Orientation using Orthonormal Matrices. *Journal of the Optical Society of America*, 5(7):1127–1135, July 1988.

16. Siu Chi Hsu and Irene H. H. Lee. Drawing and Animation Using Skeletal Strokes. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 109–118, July 1994.

17. Pierre-Marc Jodoin, Emric Epstein, Martin Granger-Pichi, and Victor Ostromoukhov. Hatching by Example: a Statistical Approach. *NPAR 2002 : Second International Symposium on Non Photorealistic Animation and Rendering*, June 2002. To appear.

18. Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes, and Adam Finkelstein.

¶ To name a few: "The Big Snit" by Richard Condie, "Special Delivery" by John Weldon and Eunice Macaulay, "Billy's Balloon" by Don Hertzfeldt, the animations of Bill Plympton, "Crac" by Frédéric Back, and the feature film "My Neighbors, the Yamadas" by Isao Takahata.

WYSIWYG NPR: Drawing Strokes Directly on 3D Models. In *Proceedings of SIGGRAPH 2002*, July 2002. To appear.

19. Leif Kobbelt and Peter Schröder. A multiresolution framework for variational subdivision. *ACM Transactions on Graphics*, 17(4):209–237, October 1998.

20. Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics*, 20(3):127–150, July 2001.

21. Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein, and John F. Hughes. Real-Time Nonphotorealistic Rendering. In *SIGGRAPH 97 Conference Proceedings*, pages 415–420, August 1997.

22. Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Loring S. Holden, J. D. Northrup, and John F. Hughes. Art-based Rendering with Continuous Levels of Detail. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, pages 59–66, June 2000.

23. Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. Interactive Pen–And–Ink Illustration. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, pages 101–108, July 1994.

24. Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Machine Intell.*, 13(4):376–380, 1991.

25. Li-Yi Wei and Marc Levoy. Fast Texture Synthesis Using Tree-Structured Vector Quantization. *Proceedings of SIGGRAPH 2000*, pages 479–488, July 2000.

26. Li-Yi Wei and Marc Levoy. Texture Synthesis Over Arbitrary Manifold Surfaces. *Proceedings of SIGGRAPH 2001*, pages 355–360, August 2001.

27. C. I. Yessios. Computer drafting of stones, wood, plant and ground materials. In *Computer Graphics (Proceedings of SIGGRAPH 79)*, volume 13, pages 190–198, August 1979.

28. Lexing Ying, Aaron Hertzmann, Henning Biermann, and Denis Zorin. Texture and Shape Synthesis on Surfaces. *Proc. 12th Eurographics Workshop on Rendering*, pages 301–312, June 2001.

29. Song-Chun Zhu. Embedding Gestalt Laws in Markov Random Fields. *IEEE Trans. Pattern Anal. Machine Intell.*, 21(11), November 1999.

Figure 6: *Shape synthesis with translation invariance. The example shape is shown on the left, and the output shape on the right.*

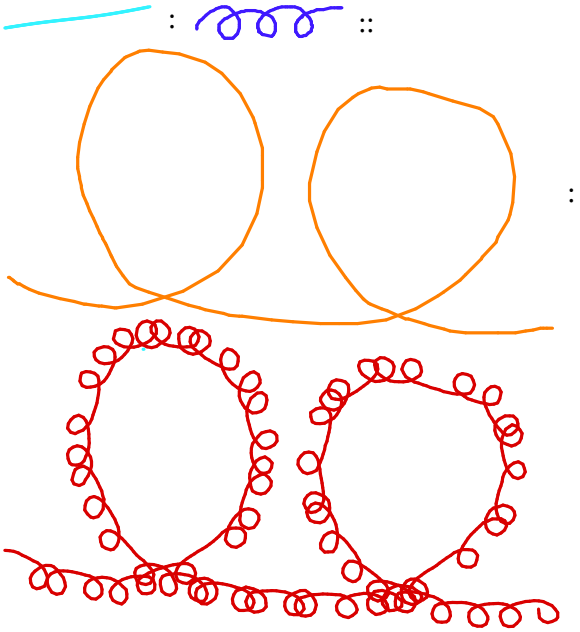


Figure 7: *Shape analogy with rotational invariance. The B' curve is generated by analogy to the other curves.*

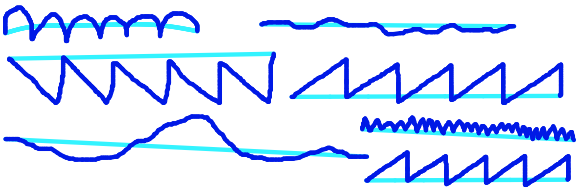


Figure 8: *Source A and A' curves used in Figure 11.*

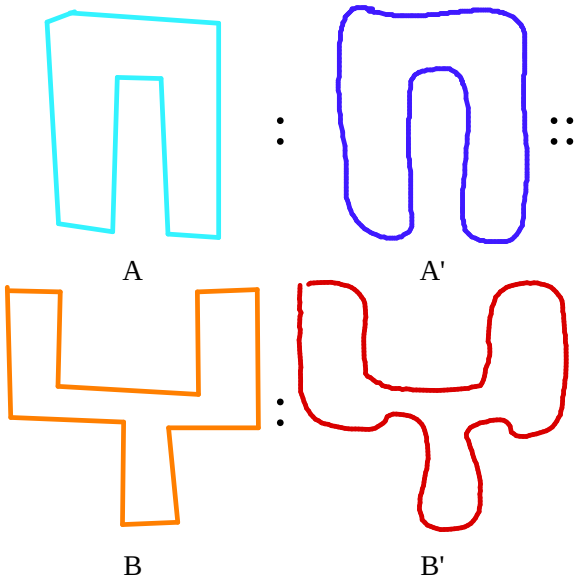


Figure 9: *Shape analogy with translation invariance. The B' curve is generated by analogy to the other curves.*

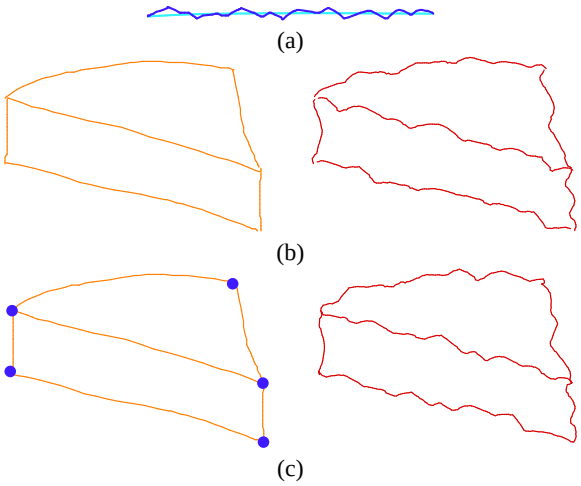


Figure 10: *Shape analogy with constraints. (a) Source A and A' pair. (b,Left) Source B curves. (b,Right) Applying the style to each curve separately produces curves that do not intersect at their endpoints. (c,Left) Source B curves with hard constraints. (c,Right) Applying the style with constraints forces the curves to meet at their endpoints.*

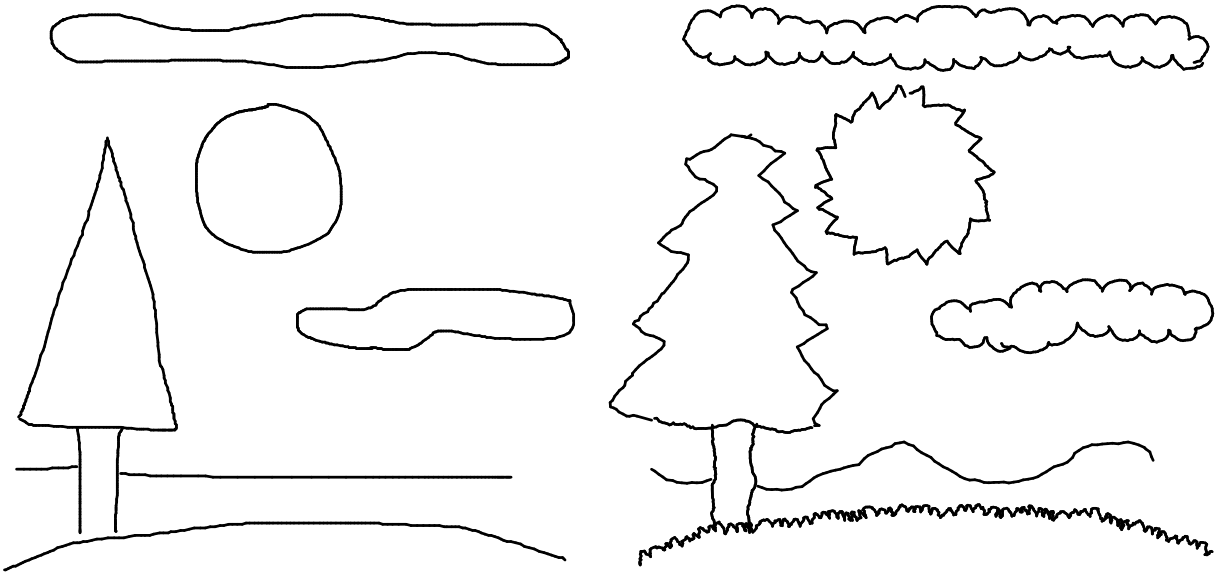


Figure 11: *Combining shape analogies. The curves in the left image were processed with styles learned from the examples in Figure 8, and composited manually to produce the right image. The character and texture of the example strokes are transfered to the source drawing. Rotational invariance was used for all examples.*

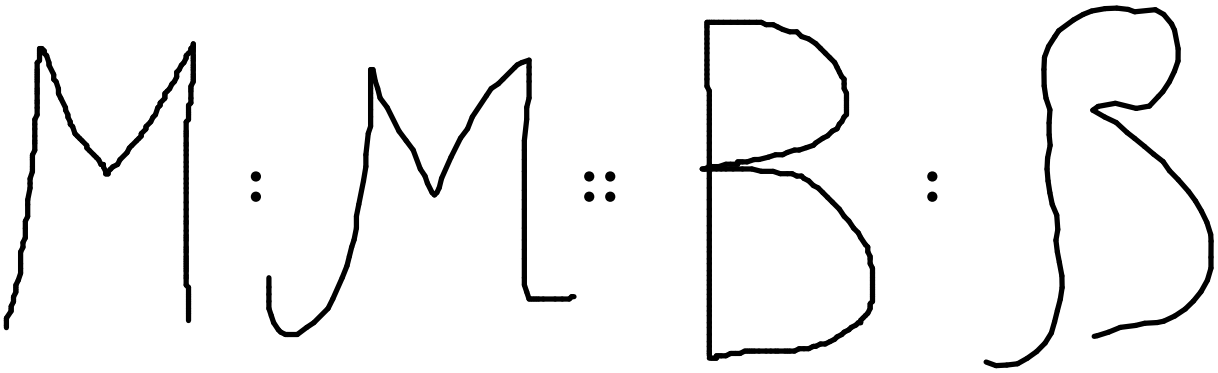


Figure 12: *Shape analogy with rotational invariance, applied to handwriting. The output curve is generated by analogy to the other curves. Here, we create a filter that makes the shape less rigid and more flowing. For example, the bend at the beginning of the "M" is copied to the beginning of the "B."*