AN AUTOMATIC BEAUTIFIER FOR DRAWINGS AND ILLUSTRATIONS

Theo Pavlidis
Christopher J. Van Wyk

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Abstract

We describe a method for inferring constraints that are desirable for a given (rough) drawing and then modifying the drawing to satisfy the constraints wherever possible. The method has been implemented as part of an online graphics editor running under the UNIX† operating system and it has undergone modifications in response to user input. Although the framework we discuss is general, the current implementation is polygon-oriented. The relations examined are: approximate equality of the slope or length of sides, collinearity of sides, and vertical and horizontal alignment of points.

CR Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*geometric algorithms, languages, and systems*; I.4.3 [**Image Processing**]: Enhancement—*geometric correction*; I.4.5 [**Pattern Recognition**]: Clustering—*algorithms and similarity measures*.

1. Introduction

Illustrators often need to create drawings that contain many parallel line segments, right angles, aligned rectangles, and that are in other ways "neat." This has led to the development of graphics editors and languages that permit users to specify an illustration by constraining the relative locations of picture elements like points and lines rather than by giving them explicitly and absolutely [1-3]. These tools grow more cumbersome to use as the complexity of a drawing increases. It would be nice if one had a procedure whereby a sloppily drawn figure could automatically be redrawn more neatly.

One could view interactive editors in which one can snap to rectilinear grid points as implementing a simple kind of beautification. Drawing with a grid has some serious limitations; for example, once a resolution is selected refinement is difficult, and there is a very coarse quantization on the slopes of lines allowed in the drawing. In addition, the idea of "snapping" offers little help in creating computer representations of drawings from paper originals by digitization [4-6], which is necessary in order to integrate electronic and paper graphics. Another potential application of beautification is in the extraction of solid object descriptions from digitized images (range or light data). Edge extraction algorithms produce the equivalent of a line drawing which is rough because of noise in the image [7]. The process for converting that set of lines into a set which is consistent with a projection of the edges of a solid requires transformations that are either identical or similar to those used in beautification. Beautification stands in contrast to specialized kinds of picture transformations. An example of the latter is common in integrated-circuit editors that implement layout, compaction, and routing algorithms [8,9].

In this paper we discuss how we approached the design of a beautification procedure, our experience in its implementation, and how the reactions of users have led us to modify our approach. Section 2 presents what we have come to understand to be the heart of the beautification problem, and the requirements that a beautification procedure must meet. Sections 3-5 present a solution to the problem as we pose it in Section 2. We implemented our solution as part of the PED graphics editor [10], which runs under the UNIX operating system on the AT&T Teletype 5620 dot-mapped display terminal [11]; PED users may invoke the beautification procedure at any time, and the result is displayed in real time. Feedback from these users has been very valuable to our understanding of the problem and this paper reflects what we have learned from public use of a part of our solution.

2. Fundamental Problems and Outline of the Beautification Scheme

Our approach to automatic beautification requires the solution of two problems. First, we need to infer from the initial sketch appropriate constraints to impose on the beautified version. Second, we need to change the initial sketch to satisfy those constraints. To reduce the vagueness of these descriptions so that an algorithmic attempt at solution is at least possible, we shall restrict the problem in a number of ways.

First, we limit our goal to images that can be defined in terms of points. This restriction covers most line drawings. We can express the problem mathematically as follows. Let the points of a drawing—vertices of polygons, centers of circles, etc.—be (x_1,y_1), $\cdots$, (x_n,y_n), and let $\mathbf{X}$ be the vector of coordinates with $\mathbf{X}_{2i-1} = x_i$ and $\mathbf{X}_{2i} = y_i$ for $1 \leqslant i \leqslant n$. Suppose the initial value of $\mathbf{X}$ is $\mathbf{X}^O$. It is tempting to define beautification as choosing a set of constraints $\mathbf{F}_k$ such that

$$\mathbf{F}_k(\mathbf{X}^O) \leqslant \delta,\ k = 1,2,\cdots, \qquad (1)$$

then finding a vector $\mathbf{X}^B$ such that

† UNIX is a Trademark of AT&T Bell Laboratories.

$$\mathbf{F}_k(\mathbf{X}^B) = \mathbf{0},\ k = 1,2,\cdots, \tag{2}$$

and

$$\|\mathbf{X}^B - \mathbf{X}^O\|_\infty < \epsilon, \tag{3}$$

where $\|\cdot\|_\infty$ denotes the maximum-coordinate-norm.

For example, consider the simple constraint that points line up vertically. Suppose x_3 and x_8 differ by less than δ; then we would look for a solution that satisfies the equation

$$x_3 - x_8 = 0\ .$$

Formally, there is only one relation, $\mathbf{F}_1$, in this case expressed as the scalar product of $\mathbf{X}$ with a vector $\mathbf{V}$ whose components are all zero, except for $\mathbf{V}_5 = 1$ and $\mathbf{V}_{15} = -1$.

This formulation of the beautification problem assures that all points stay near their original positions. It offers no insight as to the choice of $\mathbf{F}_k$ (indeed, $\mathbf{F}_1(\mathbf{X}) = \mathbf{X} - \mathbf{X}^O$ is an obvious choice that is appropriate in some cases), but given a suitable $\mathbf{F}_k$, it reduces the second part of the beautification problem to mathematical programming (linear programming in case all constraint equations are linear).

The example of Figure 1 shows a pitfall in this approach. The original drawing contains several lines that are nearly vertical. If, as above, we choose $\{\mathbf{F}_k\}$ to be the constraint that nearly equal x-coordinates should be made equal, then we can obtain a solution that satisfies $\{\mathbf{F}_k\}$ and moves all points by only a small amount. However, the result of this beautification may be unsatisfactory because three of the line segments have been moved on top of each other. This suggests the desirability of imposing negative constraints, which say that picture entities should *not* satisfy a relation, in addition to positive constraints that enforce relationships among entities.† For this example, one possible negative constraint is that distinct points that are close in both x and y coordinates should not be constrained to be equal. Note that using tighter bounds on the size of allowed movement during the beautification does not resolve this problem: The endpoints of the three rightmost lines may be moved by much larger amounts than those of the three leftmost lines and we will still achieve good results.

Formally, we should add a set of nonequalities to the system of Eqs. (2) and (3):

$$\mathbf{G}_k(\mathbf{X}^B) \neq \mathbf{0},\ k = 1,2,\cdots\ . \tag{4}$$

Notice that even if $\{\mathbf{F}_k\}$ and $\{\mathbf{G}_k\}$ are linear, the system of (2-4) cannot be solved by linear programming. Another reason for not using mathematical or linear programming to solve the constraint systems is that they offer little guidance in choosing a large satisfiable subset of a system than cannot be completely satisfied; such systems arise frequently in beautification.

So far we have not addressed the computational feasibility of finding the constraint relations $\{\mathbf{F}_k\}$ and $\{\mathbf{G}_k\}$, and of solving the resulting system of equations, inequalities, and nonequalities. The search for relations among the points or entities of a drawing may become very expensive because of combinatorial growth. This has led us to impose another limitation on our approach: *In our search for approximately satisfied constraints $\{\mathbf{F}_k\}$ among objects, we consider only relations that can be mapped into a single scalar number.*

For the above examples, the constraint is vertical alignment of points, which can be expressed as the equality of x-coordinates. As another example, parallelism among line segments can be expressed as the equality of angles with the horizontal. For each class of such constraint relations we sort the objects according to the value of their scalars and seek objects that are appropriately related in the sorted list. ("Appropriately related" usually, but not always, means "approximately equal.") This ensures that the search for relations among N objects can be performed in time $O(N\log N)$ rather than $\Omega(N^2)$. The class of constraint relations that can be mapped into scalars is surprisingly rich. As an unusual example, consider symmetries along a vertical axis: these can be found by sorting line segments according to angle with the horizontal, then finding pairs of sides whose angles sum to π.

Many of the relations that cannot be found in this way can be discovered and satisfied directly by simpler, but less general, techniques. For example, finding a smooth curve approximating a roughly drawn curve can be expressed as a spline approximation problem ([12], Ch. 12), which can be solved explicitly for the set of points. As another example, consider the termination of curves on other curves as shown in Figure 2. To find picture elements that are thus related requires pairwise examination of objects, an operation of at least quadratic complexity. However, we need consider only objects that are geometrically close to each other, so we can presort the objects by neighborhood or bounding box to find such constraints in small expected time.

We describe in the next section a clustering scheme that creates relations from the sorted lists. From these relations we create a set of constraint equations and inequalities; the constraint relations used in our current implementation are described in Section 4. Section 6 shows how we solve the constraint system using a simple incremental procedure; an important advantage of the approach described there is that it can be used to suggest which constraints should be dropped when the whole constraint set $\{\mathbf{F}_k\}$ cannot be satisfied.

3. The Clustering Scheme

The clustering strategy is described here as a general operation. It may be helpful to think of the concrete example of clustering line segments according to the angle they form with the horizontal so that nearly parallel segments can be found and made parallel.

Suppose the n elements to be clustered have been sorted by the appropriate scalar value; let the elements in this order have scalar values v_i, $1 \leqslant i \leqslant n$. A clustering of the n elements by the scalars v_i into k clusters is an increasing subsequence i_j, $1 \leqslant j \leqslant k$. Setting $i_{k+1} = n+1$ to take care of the boundary condition, we say that the elements of the j^{th} cluster have scalar values v_{i_j} through $v_{i_{j+1}-1}$. Figure 3a shows an example arrangement of elements. While a human observer may readily see two clusters as in Figure 3c, detecting them automatically is difficult.

In general, the goal of clustering is to minimize a

† This is one of the principal changes to the beautifier whose need was made apparent by users' experience.

measure of variation of values of individual elements from certain "central" values ([13], pp. 217 - 252). If V_j is the central value for the j^{th} cluster, then we may wish to minimize

$$E = \max_j [\max_i |v_i - V_j|,\ i_j \leqslant i \leqslant i_{j+1}-1\ ],\ \text{for } 1 \leqslant j \leqslant k. \quad (5)$$

This expression can always be made zero if we select as many clusters as data points, so many statistical techniques for clustering seek to minimize E when an upper bound on k is given as well. Unfortunately, no *a priori* assumption about the number of clusters is reasonable for drawings. In addition, most statistical techniques try hard to assign each object to a nontrivial cluster, an unrealistic expectation for drawings. One alternative to a fixed upper bound on k would be to minimize the number of clusters subject to an upper bound on the width, $|v_{i_{j+1}-1} - v_{i_j}|$, of any cluster. But this can also lead to unexpected results as in Figure 3b: a person sees a central cluster and two outlying singleton clusters, but it is possible to divide the elements into just two clusters and still satisfy the bound on maximum cluster width. This example has led us to augment the strategy of clustering subject to a maximum cluster width with the following rule: *Select a small constant δ (a fraction of the maximum cluster width) and group together all points that form sequences such that the difference in values of adjacent elements is less than δ.* (This scheme can be modified trivially to find relations that do not require approximate equality of values, but some other numerical relation. For example, to search for segments that are nearly perpendicular to each other we need only replace everywhere $|v_{i_{j+1}-1} - v_{i_j}|$ by $|v_{i_{j+1}-1} - v_{i_j} - \pi/2|$.) We offer some statistical justification for relying on the adjacent pair differences.

Assume that two random processes are involved in the creation of a drawing. One is the selection of a *desired* value for an element; the second is the realization of this value.† Fundamental to the idea of beautification is the assumption that the range of possible desired values is much larger than the range of deviations around a desired value. An even stronger assumption is needed to justify the above strategy: The variation among intended values is much larger than the variation among approximations to the same intended value. It would be nice if we had statistical models for drawings so that we could test the validity of these assumptions. In their absence we present some results from probability theory related to the observed pairwise distances among objects.

Parzen ([14], pp. 322-323) gives the following expression for the density of x, the observed maximum range as a fraction of the total range for n samples drawn from a uniform distribution

$$f(x) = n(n-1)x^{n-2}(1-x).$$

The expected value of x is easily computed to be $(n-1)/(n+1)$, so the average distance between samples is $1/(n+1)$. Feller ([15], Section VIII.5) treats unlimited sequences of Bernoulli trials and shows that the maximum range for n samples tends with probability 1 to the limit $\sqrt{c \log\log n}$, where c is a constant proportional to the variance of the distribution. This bound grows very slowly with n, so the average distance between adjacent samples is proportional to $1/n$.

What these results tell us is that if we draw samples from two distributions, one having a much bigger range than the other, we may end up with similar pairwise distance distributions, if we draw more samples from the distribution with the larger range. In our case, let d_1 be the expected pairwise distance between adjacent samples when they are approximations to the same value and d_2 when they are not. Since the number of samples approximating a single value (e.g. all lines at nearly the same angle) is likely to be much smaller than the total (all lines in the drawing), this would compensate for the fact that the error range is much smaller than the total range. Therefore the values of d_1 and d_2 could be quite close! On the other hand, if the distribution of desired values is far from uniform, the values of d_1 and d_2 will differ significantly. This is often the case in many technical drawings where there are few chosen orientations (usually including vertical and horizontal). Therefore we chose δ empirically so that when the distance between adjacent pairs is less than δ it is reasonable to claim that they are approximations to the same intended value.

The lack of reliable statistical models for drawing generation makes it imperative to verify empirically any solutions that are based on statistics. Therefore the implementation of our method—in our case, as part of an online drawing editor—was an essential part of the design.

The discussion above focussed on the formation of groups of elements; now we discuss how these groups are processed to form clusters. Figure 4 shows that elements that are pairwise close should not necessarily be clustered together. Therefore we compare the width of each group with two other constants δ_1 and δ_2. (The maximum expected cluster width is between δ_1 and δ_2.)

- If the width is less than δ_1, we place all of its members into a single cluster.
- If the width is greater than δ_2, we assume we have a "chaining" situation like that of Figure 4, and the group is dissolved: none of its members is clustered. Since we assume that the small differences are intentional, we also impose negative constraints as discussed below to insure that the solver leaves the chained elements alone.
- If the width is between δ_1 and δ_2, the group is modified either by removing one of the end elements, or splitting it into two groups (if the greatest pairwise distance is in the middle of the group); the resulting group or groups become clusters.

Finally, we consider negative constraints. Recall that these arise when elements that wind up in the same cluster should not be constrained to satisfy the cluster value because doing so causes entities of the picture to collapse. In principle we could discover desirable negative constraints by clustering within each cluster according to another criterion. However, for simplicity (and for more flexibility in the selection of negative constraints) we consider all possible pairs of cluster elements; since

† While the creation of any given drawing is (hopefully) a deterministic process, one could collect statistics over a large number of drawings and thus compute probability density functions for various parameters of drawings. These would also be parameters to our beautification process; they would not affect the underlying model.

each cluster has few elements (usually at most 10) the computation cost is not excessive. To express a negative relation between cluster elements, we remove them from the cluster, and tag them to prevent undue modification by the solver. Figure 5 shows an example where this approach is essential. If we are clustering by angle with the horizontal, sides *AB*, *CD*, *DE*, and *FG* are likely to cluster together. While it is permissible to make both *AB* and *FG* vertical, making *CD* and *DE* vertical causes them to collapse together. So these sides have their slopes frozen at their initial value before the solver is allowed to make any modifications to the picture.

4. Constraint Possibilities among Lines and Points

Our current implementation uses the following constraint relations among points, or lines, or points and lines. With each positive constraint we list the corresponding negative constraint. For brevity we use the term *side* for line segment.

1. A set of sides should lie at the same angle to the horizontal, unless they intersect. We modify this relation further so that clusters at angles near a preferred value (currently a multiple of $\pi/4$) are adjusted to have exactly that value.
2. A set of sides with similar slopes should be collinear, unless their projections perpendicular to the common slope intersect.
3. A set of sides should have the same length.
4. A set of points should be horizontally [vertically] aligned, unless they are also vertically [horizontally] aligned.

Figures 6 and 7 each show an original drawing and the same drawing adjusted after the above constraint relations have been found and enforced.

Other relations that fit into this beautification framework, but that we have not implemented, include:

1. A pair of adjacent sides should meet at the same angle as another pair of adjacent sides.
2. A set of points or sides is horizontally or vertically symmetric.
3. A point lies midway (or in general, some fixed fraction of the way) between two other points.

Two examples of relations that do not fit into this framework are:

1. Two curves are tangent.
2. A circle is inscribed in a polygon.

5. Creating the Equations

We describe here how constraint equations are formed from the clusters. The most general equations involve two sides; in the implementation, these are given as explicit endpoints, but for what follows it is convenient to write the endpoints of side i as (x_i,y_i) and $(x_i+\Delta x_i, y_i+\Delta y_i)$, and its preferred angle with the horizontal as θ_i.

Slopes: Clustering is performed according to the angle each side makes with the horizontal. To keep nearly horizontal lines together, angles are not taken in the usual range $[0,\pi)$ but in $[-\pi/10, 9\pi/10)$. Once the clusters are formed, those whose values are close to a preferred direction (currently, a multiple of $\pi/4$) have their values set to exactly the preferred value and clusters that differ by nearly $\pi/2$ have their values set so the difference is exact. Then each side in each cluster has its preferred angle set to the cluster value. Sides in clusters with more than one member, and sides whose angles have been set to a preferred value, are used to generate equations; there will be one equation per cluster member.

Sides within slope clusters that intersect are in danger of collapsing together if the cluster value is imposed on them. The negative constraint—that this should not happen—is imposed by fixing the slopes of such sides to their initial values. This allows the endpoints to move, but prevents complete collapse of the sides.

If the preferred angle, θ_1, is between $\pi/4$ and $3\pi/4$, we generate the equation $\Delta x_1 = \Delta y_1 \cot\theta_1$; otherwise we generate the equation $\Delta y_1 = \Delta x_1 \tan\theta_1$. This keeps all coefficients less than or equal to one in absolute value, and so improves the numerical stability of the solution procedure.

Collinearity: We seek collinear picture elements by clustering within slope clusters, using as the significant scalar the signed distance of each side from a fixed point. (We fix this point at the center of gravity of the endpoints of the sides in the original cluster, thus minimizing the maximum absolute value of the resulting numbers.) From a cluster of k nearly collinear sides, we form $k-1$ equations, each between two sides in the cluster. Because each side also generates a slope equation, the collinearity constraint can be written simply as one of the equations

$$y_{i+1} - y_i = (x_{i+1} - x_i)\tan\theta_i$$

or

$$x_{i+1} - x_i = (y_{i+1} - y_i)\cot\theta_i \ ;$$

again, the choice between them is made so that all coefficients have magnitude at most one.

Length Equality: We cluster sides by their length in the initial drawing. From a cluster of k sides of nearly equal length, we form $k-1$ equations, each between two sides in the cluster. The length of side i is

$$\sqrt{\Delta x_i^2 + \Delta y_i^2} = |\Delta x_i \sec\theta_i| = |\Delta y_i \csc\theta_i| .$$

We can remove the absolute value signs using the approximate values of Δx_i and Δy_i computed from the original drawing. Thus there are four ways to write the length equality constraint as a linear equation; we choose among them as above.

Coordinate Coincidence: To find approximate vertical alignment we need to compare points rather than sides. To simplify implementation we treat points as sides of zero length, and cluster by x coordinate using the side-clustering routine. A cluster of k points with close x coordinates generates $k-1$ constraint equations. If two points with close x coordinates also have close y coordinates, the maximum they will be allowed to move is reduced from the default to one third of the distance between them. This makes it impossible for them to collapse together. Horizontal alignment and non-alignment is detected by a similar process on y coordinates.

6. Solving the Equations

Since it is likely that not all of the constraints produced by the clustering algorithm can be satisfied simultaneously, the first step towards solution is to assign a penalty to each constraint so that more desirable constraints are satisfied first. We currently assign to each equation a penalty equal to the maximum change in one of its variables assuming that the others remain at their original values; this represents a preference for equations that require only a small change to the original coordinates. Note in addition that the equations that freeze the slopes of certain sides have penalty zero, so they will be imposed (and satisfied) first. In general, such equations can simply be assigned a negative penalty to insure that the solver gives them high priority.

Next we process the equations in order of increasing penalty using the algorithm of [16]. Briefly, each coordinate variable is represented at all times as a linear combination of coordinate variables; before any equation has been processed each variable is simply equal to itself. To process an equation we plug in the linear-combination representation of each of its variables and perform arithmetic simplification; if the resulting equation contains any variables, then one is chosen to become dependent on the other variables in the equation; this means that its linear-combination representation will change to one that involves the other variables. Before this change is made final, however, we check to see if it causes the newly dependent variable to move too far from its original position; if that happens we ignore the equation.

This method is not guaranteed to find a solution to the system of (1-3), even if one exists. But it usually does find a solution to a large subsystem thereof, and it is also fast, so it has worked well in practice.

7. Conclusion

The problem of beautifying pictures cannot be solved completely: there will always be changes that would be nice but that are not detected by an automatic procedure. Therefore we have concentrated on a small set of relations that appear to have wide applicability, and on understanding how these relations can be detected and imposed efficiently without causing untoward changes in the picture being beautified.

Figures 8 to 10 illustrate the application of the procedure to a nontrivial drawing. They show the effectiveness of the method and also that it is not always idempotent. One question raised by some readers of drafts of this paper (including the SIGGRAPH referees) is the "rate of success" of beautification. This is an interesting practical question because prospective users might want to know the probability that their drawings will be beautified properly. However, such statistics are not easy to collect and not practically meaningful. Since the graphics editor that calls the beautifier runs on a large number of machines it would require a rather cumbersome mechanism to collect statistics of, say, the number of "undo" operations after each beautification. Even that would not be enough because: (a) users might instead discard the result and read in a backup copy of the original; (b) the beautifier might make some desirable changes in the drawing but not others, so there would be no following overt user action canceling the beautification, even though the results were not fully satisfactory. In extreme cases where there is a spectacular failure we hear directly from disgruntled users, and this is how we found out the need for negative constrains. Most important, the success of beautification depends a lot on how closely the drawing fits the model. The current implementation does quite well on "rectilinear" drawings such as that shown in Figure 8. Thus the probability that a block diagram will be beautified should be quite high, while the current version of beautification is bound to fail on an illustration containing a set of circles that are supposed to be tangent to each other.

Our experience shows that naive (or even sophisticated) statistical methods are bound to produce unintended and undesirable results in practice. We believe that negative constraints are crucial to any beautification procedure if it is to avoid these pitfalls.

Acknowledgements

We thank Brian Kernighan and Doug McIlroy for many useful comments on a first draft of this paper.

References

[1] Sutherland, I. E., "Sketchpad: A Man-machine Graphical Communication System," in *1963 Spring Joint Computer Conference*, reprinted in *Interactive Computer Graphics*, H. Freeman, ed., IEEE Computer Soc., 1980, pp. 1-19.

[2] Van Wyk, C. J., *"A High Level Language for Specifying Pictures,"* ACM Transactions on Graphics, **1**(2) (1982), pp. 163-182.

[3] Nelson, G., "Juno," personal communication.

[4] Pferd, W., and K. Ramachandran, "Computer Aided Automatic Digitizing of Engineering Drawings," *Proc. IEEE COMSAC*, 1978, pp. 630-635.

[5] Harris, J. F., J. Kittler, B. Llewellyn, and G. Preston, "A Modular System for Interpreting Binary Pixel Representations of Line-Structured Data," *Pattern Recognition Theory and Applications*, Proc. of NATO Adv. Study Inst., Oxford, March-April, 1981, J. Kittler, K. S. Fu, and L. F. Pau, eds. D. Reidel Publishing Co., 1982, pp. 311-351.

[6] Pavlidis, T., and Cherry, L. L., "Vector and Arc Encoding of Graphics and Text," *Proc. 1982 Intern. Conference on Pattern Recognition*, 1982, pp. 610-613.

[7] Herman, M., "Generating Detailed Scene Descriptions from Range Images," *Proc. 1985 IEEE International Conference on Robotics and Automation*, 1985, pp. 426-431.

[8] Skinner, F. D., "The Interactive Wiring System," *IEEE Computer Graphics and Applications* **1**(2) (1981), pp. 38-51.

[9] Wallich, P., "A review of engineering workstations," *IEEE Spectrum*, **21**(10) (1984), pp. 48-53.

[10] Pavlidis, T., "PED: A 'Distributed' Graphics Editor," *Proc. Graphics Interface '84*, 1984, pp. 75-79.

[11] Pike, R., "The Blit: A Multiplexed Graphics Terminal," *Bell System Technical Journal* (Part 2), **63**(8) (1984), pp. 1607-1631.

[12] Pavlidis, T., *Algorithms for Graphics and Image Processing*, Computer Science Press, 1982.

[13] Duda, R. O. and P. E. Hart, *Pattern Classification and Scene Analysis,* New York: J. Wiley, 1973.

[14] Parzen, E., *Modern Probability Theory and Its Applications,* New York: J. Wiley, 1960.

[15] Feller, W., *An Introduction to Probability Theory and its Applications,* Volume I. Third Edition. New York: J. Wiley, 1968.

[16] Derman, E., and C. J. Van Wyk, "A Simple Equation Solver and its Application to Economic Modeling," *Software Practice and Experience* **14**(12) (1984), pp. 1169-1181

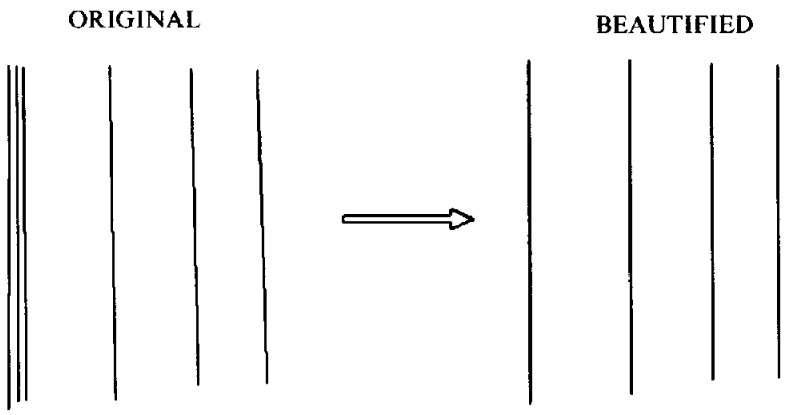

Figure 1: An undesirable beautification.

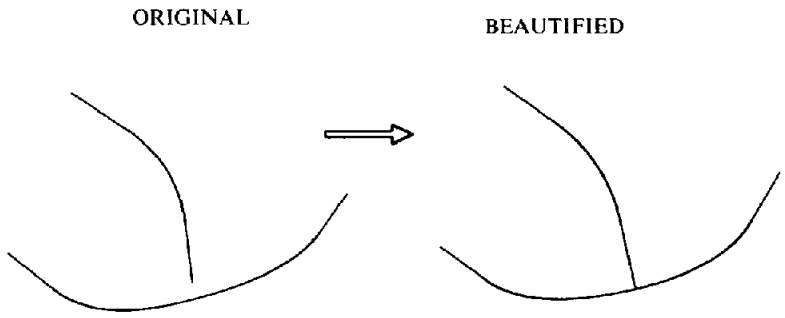

Figure 2: Beautification of curve terminations.

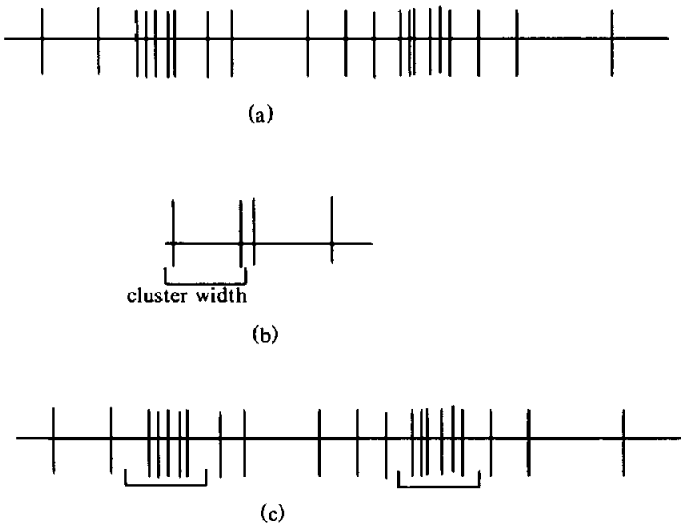

Figure 3: (a) Illustration of a clustering arrangement. The thin vertical lines denote elements. (b) An arrangement where insisting on the minimum number of clusters yields a poor answer. (c) A good clustering of the elements in (a).

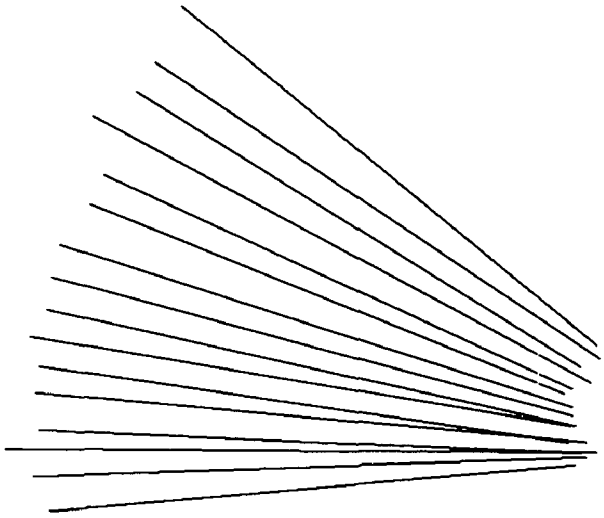

Figure 4: A drawing where clustering is inappropriate.

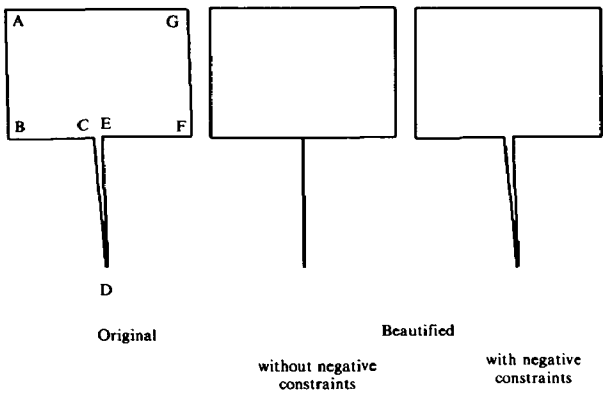

Figure 5: A drawing in which small changes in the slopes of *AB*, *BC*, *EF*, *FG*, and *GA* do not alter the gross shape, while a small change in the slope of *CD* or *DE* does.

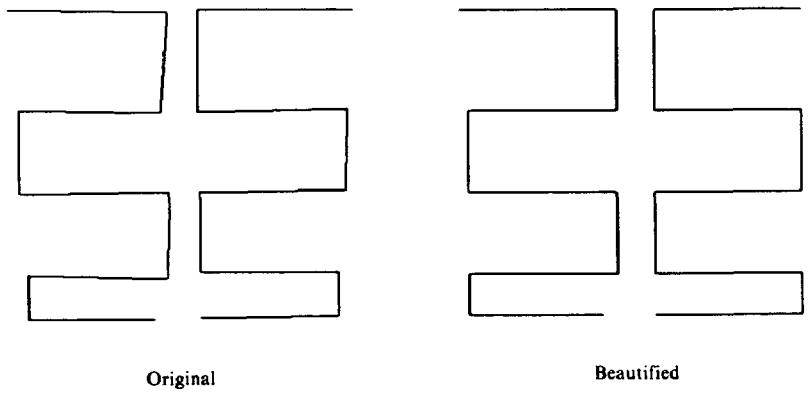

Figure 6: Example of the results of the beautification algorithm.

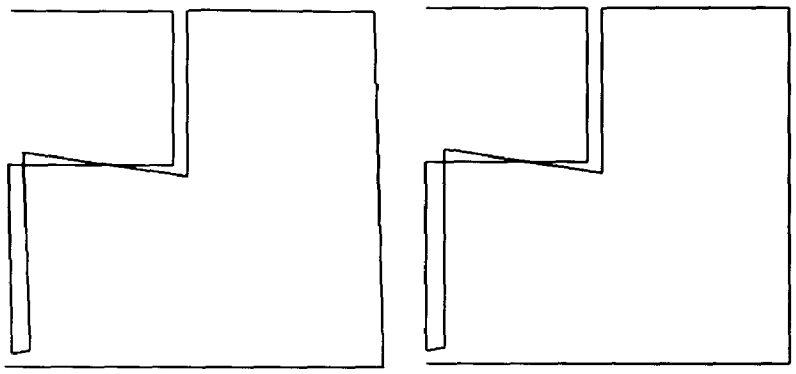

Figure 7: Example of the results of the beautification algorithm. Notice that the use of negative constraints leaves intact the sides that intersect.

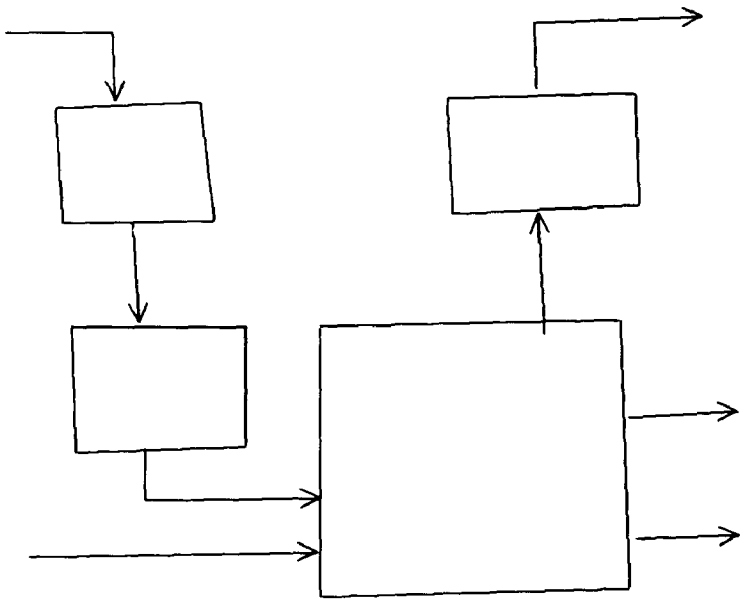

Figure 8: Original drawing

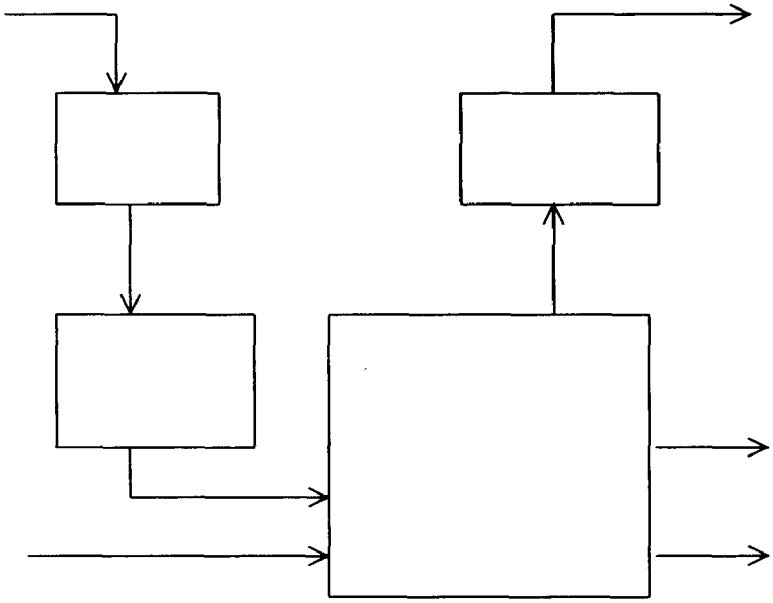

Figure 9: Drawing of Figure 8 after the First Application of the Beautifier

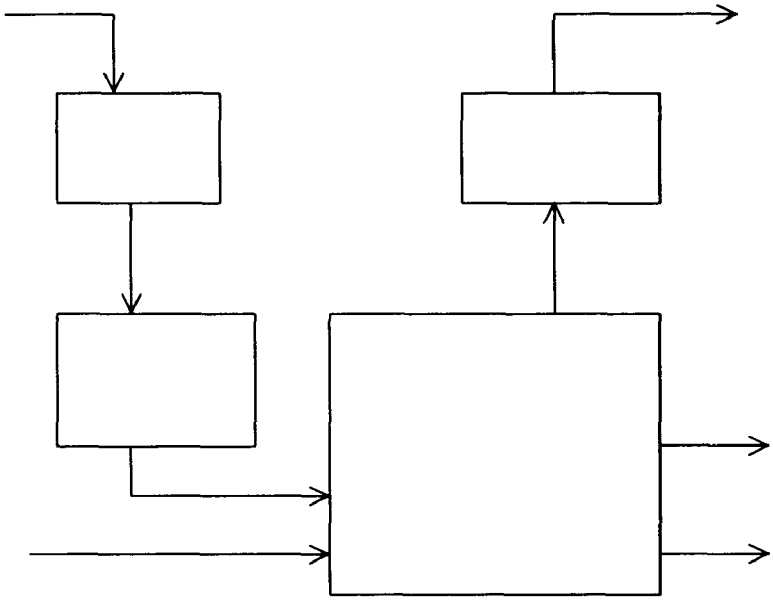

Figure 10: Drawing of Figure 8 after the Second Application of the Beautifier