Eurographics Symposium on Rendering (2005)
Kavita Bala, Philip Dutré (Editors)

Geometric clustering for line drawing simplification

P. Barla, J.Thollot and F. X. Sillion

ARTIS GRAVIR/IMAG INRIA †

Figure 1: *The two stages of our method. Lines of the initial drawing (left) are first automatically clustered into groups that can be merged at a scale ε (each group is assigned a unique color). A new line is then generated for each group in an application-dependent style (at right, line thickness indicates the mean thickness of the underlying cluster).*

Abstract

We present a new approach to the simplification of line drawings, in which a smaller set of lines is created to represent the geometry of the original lines. An important feature of our method is that it maintains the morphological structure of the original drawing while allowing user-defined decisions about the appearance of lines. The technique works by analyzing the structure of the drawing at a certain scale and identifying clusters of lines that can be merged given a specific error threshold. These clusters are then processed to create new lines, in a separate stage where different behaviors can be favored based on the application. Successful results are presented for a variety of drawings including scanned and vectorized artwork, original vector drawings, drawings created from 3d models, and hatching marks. The clustering technique is shown to be effective in all these situations.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation Picture/Image Generation

1. Introduction

Line drawing is an important aspect of modern graphics; it allows an intuitive depiction of complex scenes with a remarkable economy of means. This is probably due to the ability of the human visual system to perceive shape from intensity discontinuities.

Artists have since long learned to use this perceptual property to provide stunningly expressive pictures, by cleverly tuning line density across their drawings. However, most of the time in computer graphics, lines do not come with an appropriate density. Simply scaling a drawing, for instance for displaying on low-resolution devices, creates a need for density adjustment; moreover, density reduction in 3d is not yet mature and most non-photorealistic rendering (NPR) systems extract far too many lines. Thus there is a need to adapt the number of lines in a drawing, otherwise the effectiveness of such a representation may be compromised.

There is not a single way to simplify a set of lines, depending on the envisioned application. In the context of density reduction, we may want to adjust the line density of a drawing where too many lines project in a given region of the image. This is needed when scaling a line drawing, as well as when rendering from a 3d scene. In this context only the most "significant" lines should be drawn. Level-of-detail (LOD) representations for line-based rendering (contours and hatching), where the number of lines must vary with scale, constitute another simplification approach. Finally, in the context of progressive editing (sometimes called over-sketching), the user refines a curve by successive sketches. This can be viewed as an iterative simplification of the set of line sketches provided by the user.

In this paper, we analyse the properties shared by these three applications and propose a common solution.

† ARTIS is a research team of the GRAVIR/IMAG laboratory,a joint unit of CNRS, INPG, INRIA and UJF.

1.1. Problem statement

We consider a drawing to be a digital image composed of a number of vectorized 2d lines. Such images can be obtained in various ways: by scanning and extracting lines from a hand-made drawing; by direct digital creation using appropriate input devices (mouse, tablet, etc); by detecting contours in an image [ZT98]; or by rendering a 3d scene in a line style [SS02, GG01, GTDS04]. We thus limit our approach to static 2d drawings.

We are focusing on simplification of such static 2d drawings, *i.e.*, the creation of another set of lines containing fewer lines than the original set. We propose a generic approach for this type of problem, where simplification is controlled by a single distance-based scale parameter ε.

Of course, this rather restricted view (in which spatial proximity is used as the main discriminating criterion) implicitly assumes that all lines belong to a coherent set, over which simplification can be carried out using a very low level semantic description. In particular this approach is not tailored to regular structures or other higher order arrangements. However, nothing prevents the user from preprocessing the data to organize lines in different categories and to apply our method to simplify independently each category.

1.2. Related work

We now review relevant work involving line drawing simplification. Two research fields deal with line drawing treatments but are beyond the scope of this paper: beautification of a drawing that essentially tries to satisfy some geometric constraints to correct technical diagrams; and simplification of a single curve that deals with maintaining the global shape of a curve while decreasing its resolution. We do not consider these two fields but rather concentrate on techniques that simplify a set of curves without imposing a predefined model.

Progressive drawing tools [IMKT97, Bau94] are useful in the context of sketch-based modeling, or within vector graphics packages such as Adobe Illustrator™. These dedicated tools assist the user in adjusting the shape of a line: they are essentially semi-automatic, work iteratively and are not designed to edit more than one line at a time. Thus they are not easily adapted to other, non-progressive applications.

Several algorithms have been proposed to control the density of lines in 3d renderings. Deussen et al. [DS00] present a simplification technique dedicated to trees and vegetation, which relies on their intrinsic hierarchy, and works in object space. Preim et al. [PS95] and Wilson et al. [WM04] measure density in image space in order to limit the number of lines drawn for complex objects. Similarly, Grabli et al. [GDS04] introduce density measures in image space, used to select the most significant lines. The use of information extracted from the 3d scene (silhouettes, creases, etc.) allows them to evaluate this "significance" and order the lines by decreasing priority. The simplification process is then carried out by deleting the least significant lines. Similar approaches have been proposed in the context of resolution-dependent display and printing, since they are related to halftoning [SALS96, ZISS04]. Here again the authors add a notion of priority to ensure that the most important lines are drawn first for any tone level and then offer a selection mechanism of some lines among the original ones.

In the field of illustration, Winkenbach et al. [WS94] introduced the notion of indication: complex textures are only fully rendered in certain places of the drawing at an appropriate density, to suggest the complexity of a pattern (such as a brick wall).

Several papers about non-photorealistic rendering deal with level-of-detail rendering for animated scenes. Praun et al. [PHWF01] present an image-based method to handle LOD in hatching. Their *tonal art maps* (TAMs) are mip-mapped textures allowing real-time display of hatching styles. Other LOD creation systems have been proposed, such as the "WYSIWYG NPR" system [KMM*02] that lets the user specify the appearance of the object for several view points. Relevant LODs are then blended for a given view.

The methods presented above **only delete** the less significant lines and do not consider any **perceptual** aspect of line simplification. On the other hand, perceptual grouping approaches provide effective ways of consistently grouping lines, even if they do not adress the problem of simplification directly. Most of the work in this area deals with the extraction of closed paths in drawings [Sau03, EZ96], focusing on grouping criteria such as good continuation and closure. However, other criteria are more relevant for simplification purposes, e.g. proximity and parallelism. Unfortunately, even if each criterion has been studied in isolation [Ros94], their relative influence is yet to be determined.

1.3. Contributions

Our two main contributions reside in an attempt to model the common properties of target applications of line drawing simplification while allowing various simplification behaviors: contrary to previous methods, we construct a partition of the original set into **consistent groups** that can be replaced by an **entirely new line**. To this end, we draw inspiration from perceptual grouping.

We decompose the process into two main stages (see Fig. 1): a clustering stage in which we group the original lines and a geometric stage where a new line is created for each group. While the former is entirely automatic and common to all applications, the latter is oriented toward the specific needs of each of the envisioned applications. Our approach is general in the sense that it considers a set of minimal and low level goals shared by those applications. Therefore, it is not able to deal with higher-level structures like Winkenbach and Salesin [WS94].

We begin by describing our methodology in Section 2. The common, automatic clustering stage is presented in depth in Sections 3 and 4. Our contribution here is the definition of a modular algorithm that clusters any kind of line.

Section 5 shows how to adapt our clustering algorithm to different needs, giving simplification results in the contexts of density reduction, LOD and progressive drawing; for lines coming from different sources: scanned drawings or non-photorealistic renderings. Finally we discuss limitations of our method in Section 6.

2. Methodology

We now present the principles of our simplification method, including formal definitions that will help to clarify our approach.

2.1. Input lines

We define a line-drawing as a set of 2d lines holding a set of attributes (color, thickness, style parameters, etc.), without any assumption on their nature. Thus a line l is only defined by two end points and a continuous path between them.

$$l : [0,1] \rightarrow \mathbb{R}^2 \times \mathbf{A}$$

where $\mathbf{A}$ is the space of attributes.

In the following, we will denote by $l|_{[a,b]}$ the part of l restricted to $[a,b]$ with $0 \leq a,b \leq 1$. We are not interested in an exact parameterization of lines, but only on their geometric properties. Therefore in the rest of the paper, we will refer to a line l indiscriminately to refer to the set of geometric points that constitutes it.

2.2. Objectives

As already stated, our main goal is to create a set of lines containing fewer lines than the original one. For that, we first need to control the amount of simplification accomplished by our method. Our target applications all have a single common parameter: the simplification scale. This scale, which we denote by ε, is thus the only parameter needed by our approach. Intuitively, we only simplify the existing information at a scale smaller than ε, keeping all the information present at a larger scale.

In the applications we envision, we first need to ensure that the overall configuration of the original drawing is respected in the simplified one. Regarding perceptual grouping, this means taking **proximity** and **continuation** effects into account. To this end, we impose a **coverage** property which consists of creating new lines only in regions where initial lines can be found.

However, we are not only focusing on the line positions, but also on their shape. We want the new lines to respect the way the initial lines have been created. For instance, in Fig. 2-(a), the new line folds onto itself to cover the original lines that form a fork (Y-shapes), although no such fold was initially present. We prefer a solution such as the one shown in Fig. 2-(b), using one more simplified line, but with more fidelity to the global shape of the original lines. The same problem can be found in other examples, such as hatching groups used to shade regions (see Fig. 2-(c),(d)). In order to preserve the shape of the original drawing, we thus need another perceptual property: we want the new line and the clustered lines to be **parallel** at the scale ε. To do that, we impose a **morphological** property on simplified lines that prevents them from folding onto themselves.

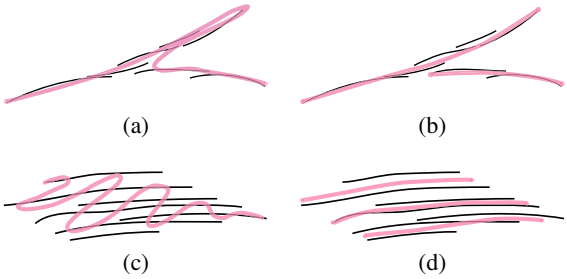

Figure 2: *(a) The simplified line (in pink) has a fold while the initial group (in black) does not - (b) Two simplified lines are preferred to represent this group - (c) The simplified line does not reflects the initial orientation and shape of the group - (d) Using three simplified lines better maintains the shape of the hatching group.*

Finally, still following perceptual grouping, we want to be able to reject the simplification of a pair of lines if their attributes (e.g. **colors**) are too different.

Following our objectives, we now give formal definitions related to our simplification approach.

2.3. Definitions

We begin by the definition of an ε-line and use it to define a group that can be simplified by a single line at the scale ε.

Following our **morphological** property, an ε-line is a line that does not fold onto itself at the scale ε. It corresponds to the fact that, for each point of l, there is no point along the normal at a distance less than ε that also belongs to l (see Fig. 3). We assume l to be G^1 in order to ensure that its normal is uniquely defined at each point:

Definition 1 Let l be a line. l is an ε-line if and only if l is G^1 and

$$\forall p \in l,\ \nexists q \in l,\ \begin{cases} q = p + \sigma \vec{n}_l(p) \\ \sigma = ||p-q|| \leq \varepsilon \end{cases}$$

where $\vec{n}_l(p)$ is the normal vector of l at point p.

This definition is equivalent to saying that l does not intersect either of its two offset curves $l^{+\varepsilon}$ and $l^{-\varepsilon}$ (see Fig. 3).

We now define an ε-group as a group of lines that can

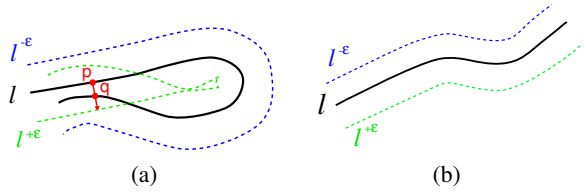

Figure 3: *(a) q is along the normal at p thus the line l has a fold and hence is not an ε-line - (b) l is an ε-line.*

be simplified by a single ε-line, as stated in our **coverage** property (see Fig. 4):

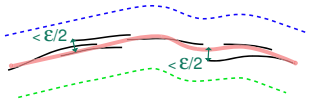

Figure 4: *An ε-group is a group (in black) that can be covered by an ε-line (in pink) at the scale ε.*

Definition 2 A group of lines G is an ε-group if and only if there exists an ε-line l such that[†]:

$$d_{SH}(l,G) < \frac{\varepsilon}{2}$$

where d_{SH} is the symmetric Hausdorff distance defined between two sets of points by:

$$\begin{aligned} d_{SH}(P,Q) &= \max(h(P,Q),h(Q,P)) \\ h(P,Q) &= \max_{p\in P}(\min_{q\in Q}||p-q||) \end{aligned}$$

An ε-group is thus a group that meets the proximity, continuation and parallelisms requirements at the scale ε.

2.4. Our approach

Following our definitions, our approach states that a simplified line-drawing is a set of ε-lines that covers the original drawing at a scale ε. We first need to cluster the original lines in a set of ε-groups before being able to create any new line. Therefore our simplification method is organized in two stages:

1. A clustering stage first groups the lines of the original drawing in ε-groups. No line is created at this stage and the process is entirely automatic using a greedy algorithm to iteratively group the original lines.
2. A geometric stage then creates a single line for each cluster of the original drawing. For each of the three target applications, we use the clusters differently and apply dedicated strategies.

[†] $\frac{\varepsilon}{2}$ ensures that two lines of the same ε-group are at a distance smaller than ε

The clustering stage is the main contribution of this paper, thus it is presented in detail in the next two sections. We then give in Section 5 various examples of the geometric stage and show that our approach can address specific applications without demanding too much effort on the user side.

3. Clustering

We use a greedy algorithm to partition the set of input lines. It is based on the iterative clustering of pairs of ε-lines and maintains the ε-group property during the entire process: clustering a pair of ε-lines that each represent an ε-group results in a new ε-line that represents the merged group. The first step consists of converting the original lines into ε-lines by splitting them at their points of intersection with their offset curves (see Section 4.1). Our clustering algorithm then iteratively clusters the pairs of ε-lines (Section 3.1) that have the minimum error (Section 3.3) until no more clusters can be created. At each step we store the hull of the clustered pair in order to take into account the result of previous clusterings in the next steps (Section 3.2).

3.1. Clustering pairs of ε-lines

Following our definitions, a pair of ε-lines (l_1,l_2) can be clustered if and only if (l_1,l_2) is an ε-group. This definition is not constructive: it only says that there must exist an ε-line that covers (l_1,l_2). We now describe a way of building this new ε-line from l_1 and l_2.

3.1.1. Possible configurations for (l_1,l_2) to be an ε-group

Observation 1 The coverage property (Def. 2) implies that if (l_1,l_2) is an ε-group then there exists a point p_1 (resp. p_2) of l_1 (resp. l_2) such that $||p_1-p_2|| < \varepsilon$. If not, it would not be possible to find a line l such that $d_{SH}(l,(l_1,l_2)) < \varepsilon/2$.

We call *overlapping zones* the portions where l_1 and l_2 are at a distance less than ε, formally defined as:

Definition 3 An overlapping zone, Z, is a pair of line portions $(l_1|_{[a_1,b_1]}, l_2|_{[a_2,b_2]})$ such that:

$$d_{SH}(l_1|_{[a_1,b_1]}, l_2|_{[a_2,b_2]}) < \varepsilon$$

where $\{a_1,a_2\}$ and $\{b_1,b_2\}$ are the extremities of Z.

Observation 2 The morphological property (Def. 1) implies that (l_1,l_2) is an ε-group if it is not a fork. Indeed, if it were the case, then any line l representing (l_1,l_2) would have to fold onto itself.

This implies that the overlapping zones must not fork, thus there must be at least one of the two lines that ends at each extremity of the zone. A simple forking configuration is shown in Fig. 5-(a). Other forking configurations exist, but are not represented because we mainly direct our attention to valid zones (Fig. 5(b) and (c)).

We call such an overlapping zone a *path* and define it by:

(a) Fork (b) Four possible configurations with one path (c) Two paths: closed curve

Figure 5: *Possible configurations of a pair of ε-lines. Only (b) and (c) can form an ε-group.*

Definition 4 A path on a pair of lines (l_1, l_2) is a maximal overlapping zone, Z, such that there is at least one extremity of l_1 or l_2 at each extremity of Z.

Knowing that each line has 2 extremities, there are five combinations for the paths between two lines, illustrated in Fig 5-(b), (c). Thus, if a pair of lines does not correspond to one of these five combinations, it is not an ε-group.

The only configuration that corresponds to a closed curve is when a pair of lines (l_1, l_2) has two paths (see Fig 5-(c)). For the sake of brevity, we will not detail this case in the following, as it is essentially equivalent to the others. However Section 3.4 shows that our algorithm correctly handles it.

3.1.2. Building the new ε-line

Now that we have identified valid configurations (paths), we build the new line l, and make sure that it is an ε-line, i.e. that it respects Definition 1. We create an ε-line l that passes from l_1 to l_2 and lies in the middle of the path.

l is obtained by concatenating the portions of lines outside the path (in purple) with a line created inside the path by interpolating between l_1 and l_2 from one extremity to the other (in pink).

Such a construction insures that l is an ε-line outside the path since l_1 an l_2 are themselves ε-lines. However for zones inside the path, some particular cases when l is not an ε-line exist. Indeed, l may fold onto itself if the curvature of l_1 or l_2 is too close to $1/\varepsilon$. In these cases, (l_1, l_2) is simply not considered as an ε-group.

3.2. Building the hull of an ε-group

The new line l we created only ensures that (l_1, l_2) is an ε-group; we cannot use it iteratively since it would not take into account the error made by the clustering of l_1 and l_2. Indeed, consider an ε-line l_3; determining if (l_1, l_2, l_3) is an ε-group is not directly equivalent to determining if (l, l_3) is an ε-group.

In order to propagate the clustering result of (l_1, l_2) to l, we define the hull of an ε-group by assigning a varying thickness to the ε-line that represents it. This thickness describes the result of the clustering of two or more lines and is used in subsequent clusterings (see Fig. 6).

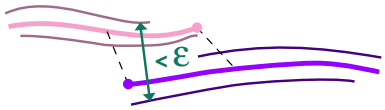

Figure 6: *To decide if the four thin lines are an ε-group we use their two representatives (in pink and purple) and compute the error measure (see Section 3.3) between the farthest lines, which will be represented by the hull.*

For each point $l(x)$, the points of the hull $l^+(x)$ and $l^-(x)$ are obtained by taking the extremal intersections along the normal with (l_1, l_2) (see Fig. 7).

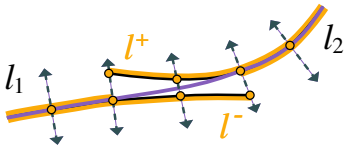

Figure 7: *The hull (in orange) of a line l (in purple) representing a pair of lines (l_1, l_2) (in black) is defined by the farthest points of (l_1, l_2) along each normal of l (dashed line).*

All the definitions given in the previous section are easily extended by considering the two hulls instead of the two ε-lines. Indeed, to decide if a pair of ε-lines (l_1, l_2) is an ε-group we only need to compute distances between pairs of points. By considering $l_1^+, l_1^-, l_2^+, l_2^-$ for the distance computation, we can determine the overlapping zones and then the paths between l_1 and l_2 while taking into account the two ε-groups they already represent. Therefore, while computing overlapping zones between two ε-lines, the distance between $l_1(x_1)$ and $l_2(x_2)$ will be taken as:

$$\max \{ ||l_1^+(x_1), l_2^+(x_2)||, ||l_1^+(x_1), l_2^-(x_2)||, \\ ||l_1^-(x_1), l_2^+(x_2)||, ||l_1^-(x_1), l_2^-(x_2)|| \}$$

Note that the hull of an ε-line l is not defined on points p where there is no intersection with l_1 and l_2 and the normal at p. For those points, we project the closest points of the two hulls as shown in Fig. 8. This will only have an impact on the error computed on the hull as explained in the next section and our choice favors pairs of aligned lines (i.e., those with **good continuation**).

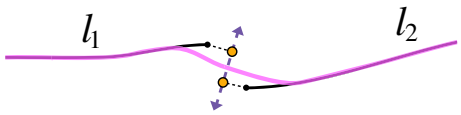

Figure 8: *In places where there is no intersection with the normal, we project the closest points on the pair of lines.*

3.3. Error measure of an ε-group

In order to use a greedy algorithm we now need to choose which pairs of ε-lines we want to cluster at each step. To this end we define an error measure.

Intuitively we want to cluster the closest lines first. By closest we mean not only spatially close but also with similar attributes. We first show how to compute the spatial error, then we explain how to incorporate an attribute error in order to orient the simplification toward a given application.

When computing the spatial error of a pair (l_1, l_2) of lines, we want to favor pairs of lines that could be clustered with the smallest possible ε. We thus define the spatial error $E_s(l_1, l_2)$ of an ε-group relative to the ε-line l chosen to represent it by the maximum thickness of the hull associated with l. This heuristic favors the clustering of the thinest groups first. This error is normalized between 0 and 1 using a division by ε:

$$E_s(l_1, l_2) = \max_{x \in [0,1]} ||l^+(x) - l^-(x)||/\varepsilon$$

The user can also define an attribute error measure $e_a(p_1, p_2)$ (normalized between 0 and 1) for a particular attribute space if he or she wants to take it into account in the clustering process. For the single attribute we used in our implementation (i.e., color), we found that a mean was better than a max to give a good estimation of the total error between two groups. This gives the following attribute error:

$$E_a(l_1, l_2) = \int_0^1 e_a(l^+(x), l^-(x))dx$$

The spatial and attribute error measures are then classically combined in a multiplicative way to give the error measure $E(l_1, l_2)$:

$$E(l_1, l_2) = 1 - (1 - E_s(l_1, l_2)) * (1 - E_a(l_1, l_2))$$

The attribute error is only computed for ε-groups, that is groups of lines that can be spatially clustered. In order to forbid clustering if the attributes of the lines of the group are too different, we add the constraint that for an ε-group (l_1, l_2) to be clustered, it must satisfy $E(l_1, l_2) < 1$.

3.4. Closed curves

Most of this method holds for closed curves and the algorithm is very similar. However, we need to implement some additional processes. First, the lines closed at the scale ε are detected. Those are the lines whose endpoints are at a distance less than ε. Moreover, when identifying paths, if the path configuration found in Fig. 5-(c) arises, the resulting ε-line becomes closed.

4. Implementation details

We have implemented the greedy iterative clustering by an edge collapse algorithm applied on a graph whose edges represent pairs of ε-lines which are ε-groups.

4.1. Preprocessing input lines

The lines we take as input can be of any kind. The only constraints are that we need to sample them. In our implementation, we use regularly-sampled Catmull-Rom splines. For all distance computations involving such samples, we use an acceleration grid of cell size ε, allowing us to quickly find candidate samples.

In order to initialize the algorithm, we need to convert an initial line l into an ε-line. To do that, we follow l, progressively creating its two offset curves, and we split l as soon as it crosses one of the already created offset curves. Note that this splitting process gives different results depending on the extremity at which one starts. We have not found any remarkable difference in the results; however, one may want to choose a more symmetric way of splitting. After this initialization step, each line is its own hull.

4.2. Building the graph

Once the input lines have been converted into ε-lines, we build a graph with a node for each input ε-line, and whose edges represents pairs of lines that can be clustered, i.e. that are ε-group.

A pair of ε-lines can only be clustered if it corresponds to one of the configurations shown in Fig 5-(b),(c). Thus, for an ε-line pair, if there are more than two extremities at a distance greater than ε from the other ε-line, we can reject it directly, saving a lot of computation time.

In practice, we compute a hull for each potential cluster and store it on the corresponding edge along with its error.

4.3. Updating the graph

Then, at each step of the algorithm, we collapse the edge with minimum error and update the graph edges locally. Collapsing an edge is done by creating a new node that stores the edge's ε-line and hull. The collapsed edge is deleted and by definition of a hull, we only have to inspect the edges incident to the collapsed nodes. Those edges are removed from the graph and new edges are created between the new node and its neighbors. We also compute the attributes of the new ε-line by linearly interpolating the attributes of the two original ε-lines.

Finally, the two collapsed nodes are removed from the graph. But instead of deleting these nodes, we keep them in a history of collapse sequences which is stored as a tree under the newly created node. This gives us access to the underlying input lines and the collapsing scheme of each cluster.

The algorithm stops when no more clusters can be created.

5. Results

In this section, we give some results to illustrate the overall simplification process, i.e. both clustering and geometric stages for each of the target applications: density reduction, level-of-detail and progressive drawing[‡]. The geometric stage is clearly a more specialized operation since the choice of the new line to be drawn is left to the chosen strategy. We implemented two "standard", pre-defined strategies:

- **Average line:** the new line interpolates all the original lines in the cluster (with application-defined weights);
- **Most significant line:** the new line is one of the original lines, chosen according to an application-defined priority measure (base on length, nature...).

The former supports a broad range of simplification behaviors, while the latter gives a simple selection/deletion scheme.

In the worst case, the total process increases quadratically with the number of input lines at a fixed scale parameter ε. In practice, for our examples, it ranges from several seconds to a minute. For each example we give the number of input lines and resulting clusters. The simplification scale is shown by a circle of diameter ε.

Density reduction Fig. 9 shows a straightforward illustration of our approach. Lines have been extracted from a scanned line drawing. The user chooses a simplification scale ε and the lines are simplified. We applied an average line strategy without smoothing the results, so that simplified lines exhibit the ε-line of each group.

Fig. 11 shows a similar scenario that takes the color attribute into account. The attribute error is a $L^*a^*b^*$ color distance.

Fig. 12 shows the use of categories to separate lines of different nature: external contour on one side and internal and suggestive contours [DFRS03] on the other side. In examples coming from 3d renderings like this one, we make use of object IDs and line nature to detect categories automatically. This allows for the use of two different geometric strategies: for the external contour an average line is drawn, whereas the longest line of each cluster is drawn for the internal and suggestive contours. Moreover, the external contour

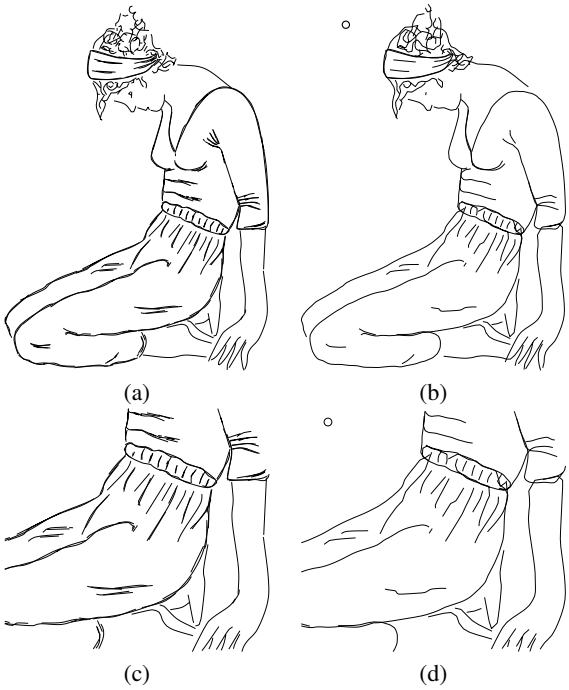

Figure 9: ***Density reduction:*** *(a) The original scanned and vectorized drawing: 357 input lines - (b) The resulting simplification: 87 clusters - (c,d) Zoom on the above images. The scale* ε *is indicated by the circle in the upper left corner.*

is simplified at a larger scale than the other lines. Resulting lines are better organized and keep the most salient features of the model.

Level-of-detail Fig. 10 shows an example of a LOD sequence produced with our approach. Progressively scaling down a drawing is equivalent to choosing an increasing ε. Thus we apply a series of simplifications with an ε step, each time starting from the previous, finer level. Here again, two different geometric strategies are used: the average line for the contour and the longest line for the hatchings. To do that, we created five categories by hand: one for the contours, and one for each of the four orientations for hatchings. Note that although no particular treatment was applied to preserve tone across simplifications, this result is quite convincing. Tone preservation could be explicitly included in the method at the geometric stage, by choosing appropriate line attributes such as width and/or color.

Progressive drawing Fig. 13 shows a drawing sequence using our progressive drawing tool. Here, the clustering algorithm is applied iteratively: The user chooses a sensitivity ε and draws a sketched line over an initial drawing; the lines are then simplified; and finally, the resulting lines constitute the initial drawing for the next step. This tool requires

[‡] A video showing an oversketching session and two example LODs (including the tree of Fig. 10) is available at http://artis.imag.fr/Publications/2005/BTS05a/

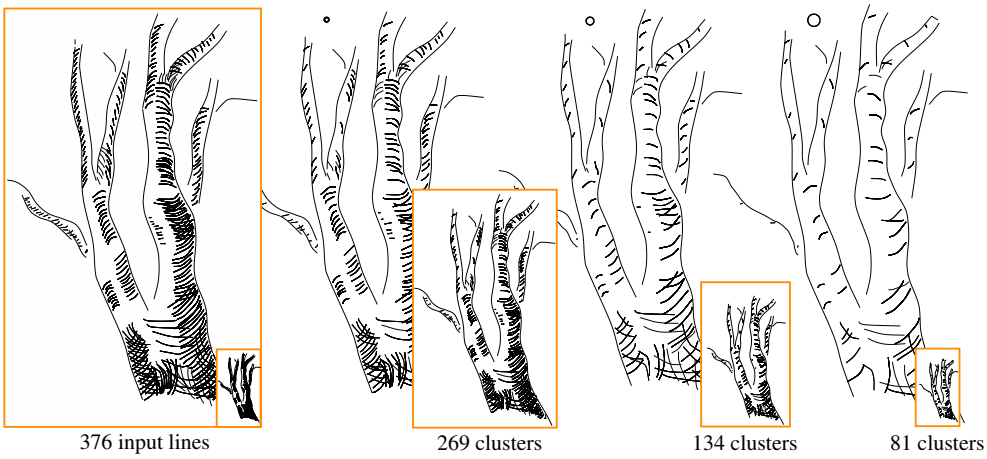

Figure 10: ***LOD:*** *A series of LODs made by progressively increasing* ε*. Using different categories and geometric strategies prevents undesired hatching lines from merging. Compare the small resized images with (right) and without (left) simplification.*

an additional feature: we only want the simplification to be done between initial lines and the new sketch. Thus the input lines are organized in two sets: the initial lines and the new sketched line. During the clustering, only the edges between pairs of nodes that lie in different sets are built.

Finally we choose a priority-based strategy because we want the last drawn line to have a greater priority than initial lines. In practice, that consists of using an average line strategy, giving greater weights to samples belonging to the last drawn line. This is made possible by the history tree stored at each cluster. We found this tool to be very intuitive, particularly for modifying lines coming from 3d renderings or extracted from images.

6. Discussion

In this paper we opted to remain very general, trying to find the common properties of some target simplification methods. However, it is clear that such a low-level method can still be specialized to adapt to other specific applications. In particular, we believe that the separation of the clustering and geometric stages is crucial for all simplification methods.

Other attributes than color could be used in the attribute error definition. However, we did not consider input lines exhibiting wiggling patterns and implicitly assumed that they come at an appropriate scale. The problem of extracting the so-called natural scale of a line has been previously adressed (e.g. [Ros98]).

Our method is invariant under rotation, scale, and translation, since it operates only on euclidean distances between pairs of points. However it has two limitations: it is not transitive and prevents simplifying forks. The former means that simplifying a drawing at scale ε_1, then simplifying the result at scale $\varepsilon_2 > \varepsilon_1$ is not guaranteed to provide the same result as a direct simplification at scale ε_2. However, this is not a problem in the applications we envision, for instance generating a discrete set of LOD representations. The latter assumes that the problem of forks is rather separate from geometric clustering (appearing at a higher level of processing and depending on the application) and thus is left as a post process.

The choice of a greedy algorithm for clustering implies that we only reach a local optimum in general. This turns out to be sufficient in practice for the applications we have tested. Other optimization techniques could be used if reaching a global optimum is important.

The evaluation of a simplification method for line drawings is not an easy task. Indeed, there is no simple and obvious quality measure for a simplified drawing. Visual evaluation involves a number of high-level interpretation processes, which are difficult to model and quantify. Our approach offers the convenience of a guaranteed geometric criterion: the resulting drawing is "within a distance ε" from the original drawing. The direct evaluation of the result is the number of clusters. However, in an attempt to provide finer evaluation tools we identified two other criteria. First, the reduction in the number of lines composing the drawing; Second, the variation of the total arc-length in the drawing. Both are strongly related to the geometric strategy chosen for an application: keeping a line per cluster clearly decreases the total number of lines, and the arc-length may strongly vary depending on the new lines created. For instance, in Fig. 9 the number of lines was divided by 4 and the arc-length reduced by 30%.

In the examples shown, we observe that a purely distance-based simplification should generally not be applied to all lines of the drawing at once, because there are categories of lines that should not be clustered. one example is lines

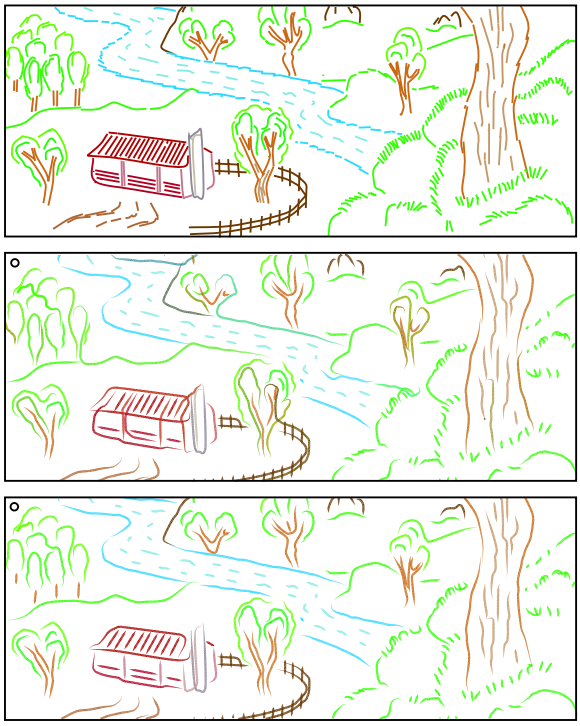

Figure 11: *Top: input drawing. Middle: simplified drawing without taking color error into account during the clustering stage. Bottom: taking color error into account better preserves the original drawing (see the fence and the tree, the trunks and the leaves...).*

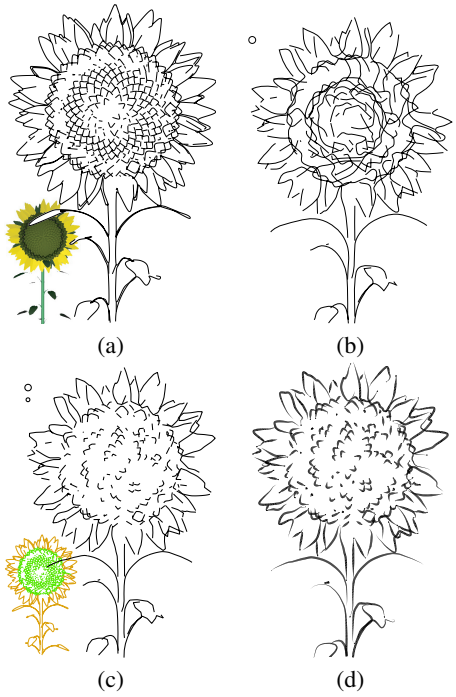

Figure 12: ***Simplification of a 3d rendering:*** *(a) 3d model and its line rendering using silhouettes and suggestive contours (531 input lines) - (b) Simplification without any category (256 clusters) - (c) Using two categories (external and internal contours) each with a different scale and geometric strategy (294 clusters) - (d) Same result in a calligraphic style.*

depicting different objects placed near each other. Segmenting the drawing and applying the simplification algorithm to each category is better for the scope of an automatic process. Naturally this raises the question of how to segment or define the categories automatically, which is beyond the scope of this paper.

Finally, we think that our approach could be extended to animation. The idea would be to guide the clustering stage temporally, not only to ensure temporal coherence, but also to cluster lines that go together across time. This could be accomplished by incorporating another perceptual grouping criterion: common fate, which states that the visual system tends to group elements with similar velocity.

7. Future work

Our approach can be extended in many ways. Forks, where a line separates into two distinct lines, are not handled explicitly in our technique. We plan to take them into account in the density reduction application, along with new geometric strategies. We plan to extend our LOD system to more elaborate transitions between levels: since we keep all the history of agglomerations, we have all the information needed to realize geometric morphs instead of blends. Moreover, we will add new features to our oversketching tool (such as alignment, anchoring, etc.).

Another interesting issue lies in the coupling of our method with contour detection in images, with applications in medical imagery and image-based rendering. Our approach is well adapted to these applications since it is able to incorporate any kind of data associated with the extracted lines (gradient, color, etc.) and to take them into account in the simplification process.

Finally, a long-term goal is to adapt our approach to the simplification of animated line drawings, taking into account the observations made at the end of Section 6.

8. Conclusions

We have presented a new, generic approach to the simplification of line drawings, that supports a large variety of applications. The clustering stage identifies groups of lines that capture the morphological structure of the original drawing; the geometric stage builds the actual lines that will represent this structure in the final drawing.

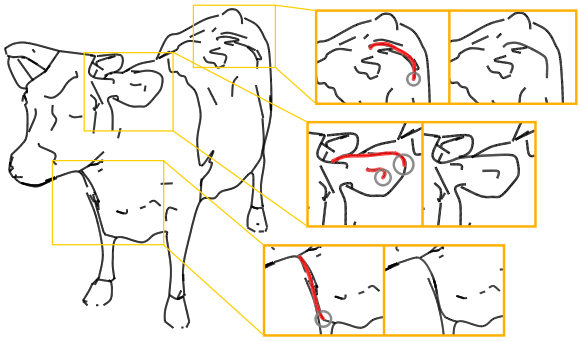

Figure 13: ***Oversketching:*** *A 3d model has been rendered in a line drawing style. The user adds new lines (in red) which are clustered with the old ones; the scale ε (gray circle) can be changed at each step.*

The low level and generic nature of this process makes it a solid foundation for many applications. We have demonstrated three possible scenarios: Density reduction of a set of lines where input lines, possibly classified in categories, are replaced by a smaller set of lines; an automatic level-of-detail system for line drawings with specific behaviors for silhouette lines and hatching groups; and a progressive drawing tool, which provides an intuitive and flexible interaction environment where the user creates or modifies existing lines with drawing gestures on the canvas.

Acknowledgement

We would like to thank Gilles Debunne for his help on the making of the video. This paper benefited greatly from suggestions given by the members of the ARTIS team and Lee Markosian. This work was supported in part by Region Rhone-Alpes (DEREVE project and EURODOC grant).

References

[Bau94] BAUDEL T.: A mark-based interaction paradigm for free-hand drawing. In *UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology* (New York, NY, USA, 1994), ACM Press, pp. 185–192. 2

[DFRS03] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S., SANTELLA A.: Suggestive contours for conveying shape. *ACM Transactions on Graphics 22*, 3 (July 2003), 848–855. 7

[DS00] DEUSSEN O., STROTHOTTE T.: Pen-and-ink illustration of trees. *Proceedings of SIGGRAPH* (2000). 2

[EZ96] ELDER J. H., ZUCKER S. W.: Computing contour closure. In *ECCV (1)* (1996), pp. 399–412. 2

[GDS04] GRABLI S., DURAND F., SILLION F.: Density measure for line-drawing simplification. In *Proc. of Pacific Graphics* (2004). 2

[GG01] GOOCH, GOOCH: *Non-Photorealistic Rendering*. AK-Peters, 2001. 2

[GTDS04] GRABLI S., TURQUIN E., DURAND F., SILLION F.: Programmable style for npr line drawing. In *Rendering Techniques 2004 (Eurographics Symposium on Rendering)* (june 2004). 2

[IMKT97] IGARASHI T., MATSUOKA S., KAWACHIYA S., TANAKA H.: Interactive beautification: A technique for rapid geometric design. In *UIST (ACM Annual Symposium on User Interface Software and Technology)* (1997), pp. 105–114. 2

[KMM*02] KALNINS R. D., MARKOSIAN L., MEIER B. J., KOWALSKI M. A., LEE J. C., DAVIDSON P. L., WEBB M., HUGHES J. F., FINKELSTEIN A.: WYSIWYG NPR: drawing strokes directly on 3d models. In *SIGGRAPH 2002* (2002). 2

[PHWF01] PRAUN E., HOPPE H., WEBB M., FINKELSTEIN A.: Real-time hatching. In *SIGGRAPH 2001, Computer Graphics Proceedings* (2001), Fiume E., (Ed.), pp. 579–584. 2

[PS95] PREIM B., STROTHOTTE T.: Tuning rendered line-drawings. In *WSCG'95* (February 1995), pp. 228–238. 2

[Ros94] ROSIN P.: Grouping curved lines. In *5th British Machine Vision Conf* (York, 1994), pp. pp. 265–274. 2

[Ros98] ROSIN P. L.: Determining local natural scales of curves. *Pattern Recognition Letters 19*, 1 (1998), 63–75. 8

[SALS96] SALISBURY M., ANDERSON C., LISCHINSKI D., SALESIN D. H.: Scale-dependent reproduction of pen-and-ink illustrations. *Computer Graphics 30* (1996), 461–468. 2

[Sau03] SAUND E.: Finding perceptually closed paths in sketches and drawings. *IEEE Trans. Pattern Anal. Mach. Intell. 25*, 4 (2003), 475–491. 2

[SS02] STROTHOTTE T., SCHLECHTWEG S.: *Non-photorealistic computer graphics: modeling, rendering, and animation*. Morgan Kaufmann, San Francisco, CA, USA, 2002. 2

[WM04] WILSON B., MA K.-L.: Representing complexity in computer-generated pen-and-ink illustrations. In *NPAR* (2004). 2

[WS94] WINKENBACH G., SALESIN D.: Computer-generated pen-and-ink illustration. *Proc. SIGGRAPH* (1994). 2

[ZISS04] ZANDER J., ISENBERG T., SCHLECHTWEG S., STROTHOTTE T.: High quality hatching. *Computer Graphics Forum 23*, 3 (2004), 421–421. 2

[ZT98] ZIOU D., TABBONE S.: Edge detection techniques - an overview. *International Journal of Pattern Recognition and Image Analysis 8* (1998), 537–559. 2