EUROGRAPHICS 2008 / G. Drettakis and R. Scopigno
(Guest Editors)

Volume 27 (2008), Number 2

Efficient and Dynamic Simplification of Line Drawings

Amit Shesh and Baoquan Chen [†1]

[1]University of Minnesota–Twin Cities

Abstract

In this paper we present a pipeline for rendering dynamic 2D/3D line drawings efficiently. Our main goal is to create efficient static renditions and coherent animations of line drawings in a setting where lines can be added, deleted and arbitrarily transformed on-the-fly. Such a dynamic setting enables us to handle interactively sketched 2D line data, as well as arbitrarily transformed 3D line data in a unified manner. We evaluate the proximity of screen projected strokes to simplify them while preserving their continuity. We achieve this by using a special data structure that facilitates efficient proximity calculations in a dynamic setting. This on-the-fly proximity evaluation also facilitates generation of appropriate visibility cues to mitigate depth ambiguities and visual clutter for 3D line data. As we perform all these operations using only line data, we can create line drawings from 3D models without any surface information. We demonstrate the effectiveness and applicability of our approach by showing several examples with initial line representations obtained from a variety of sources: 2D and 3D hand-drawn sketches and 3D salient geometry lines obtained from 3D surface representations.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation Line Rendering
Keywords: Line drawings, dynamic simplification, hierarchical simplification

1. Introduction

Line drawing has been a traditional art that aims to produce abstract and "sketchy" illustrations to convey just the right amount of information and hide unnecessary features. They can be created from several forms of 2D and 3D data to cater to specific application requirements. In this paper, we present a pipeline to produce static renditions and coherent animations of line drawings efficiently in a dynamic setting.

The nature of dynamism may change with the form of data and the desired application. When 2D static line drawings are zoomed in or out, simplification to preserve shape and stroke density is a primary issue. Computer-based sketching using tablet devices creates data naturally in the form of lines. In several artistic and conceptual design applications it is desirable to retain the original free-hand nature of sketches instead of trying to refine them into formal geometry [ZHH96, SC04]. Simplifying and rendering such sketches as line drawings while being able to interactively add and delete strokes and transform them adds an extra challenge. By supporting such a scenario, our pipeline can act as a front-end to many existing sketching applications [Kal05, KHR04, DXS*07, IOOI05].

Simplifying and rendering line drawings on-the-fly obtained from 3D data presents two unique challenges. First, as 3D line data may be subject to an arbitrary sequence of rigid, deforming and projective transformations, simplifying them efficiently on-the-fly is difficult. The second challenge concerns the issue of visibility. Many techniques to create line drawings from 3D geometry in the form of meshes [SP03, GDS04, WM04, JNLM05, HZ00], points [XC04] and volumes [BKR*05] have been proposed in the past. Most previous work use attributes of the underlying surface geometry to address visibility. However in many cases, using an underlying surface to determine visibility between strokes may be inappropriate (e.g. freehand strokes drawn on planes, as in Mental Canvas [DXS*07] and 3D6B [Kal05]) or even impossible (interactively created wire frame CAD models). In addition to stroke simplification, our pipeline generates,

[†] {asheshlbaoquan}@cs.umn.edu

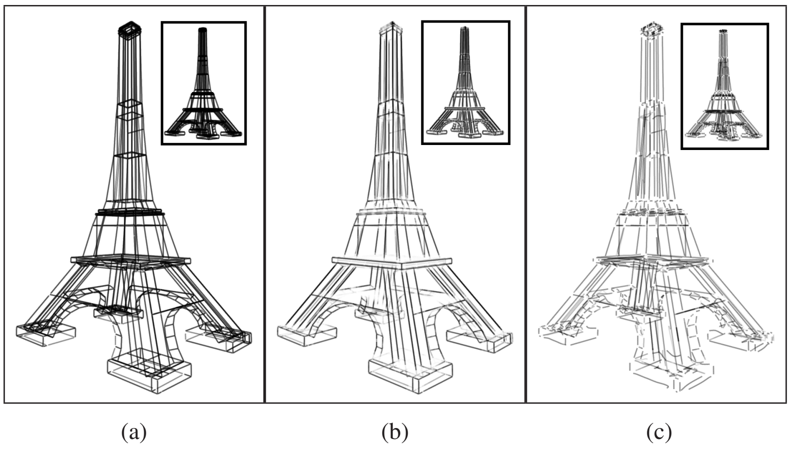


Figure 1: *Line drawings of the Eiffel Tower. Each column shows the Eiffel Tower zoomed out to the same level (thumbnail) and the thumbnail magnified. (a) the original wireframe model. (b) the tower rendered using only visibility cues that reduce some of the clutter. Without simplification, this figure is rendered at a very low frame rate. (c) the tower rendered after simplification and with visibility cues. Simplification further reduces the clutter when zoomed out to such a level in the thumbnail in (c) to retain the overall shape of the tower but hide the detailed truss structures.*

without *any* surface information, local visibility cues for 3D line data that create a global occlusion effect. Thus, we show that it is possible to generate many kinds of line drawings without using any surface information. Due to this, our pipeline can be used to generate line drawings from 3D models, even with available surface representations by pre-extracting salient geometry lines (Elber [Elb95] provides a simple approach for this extraction). This creates the potential to create a generic lightweight line-based representation to create line drawings, saving memory bandwidth and simplifying graphical processing for conventional 3D models.

Efficient and dynamic line simplification is one of our main contributions. There are two generic principles governing the process of line simplification: *proximity* (strokes near each other should affect each other) and *continuity* (the integrity of each stroke must be preserved) as described by Barla *et al.* [BTS05]. To adhere to these principles, some form of a level-of-detail hierarchy of strokes can be built based on proximity so that continuity is maintained during rendering and animation. For static 2D line models (from vectorized images or previously drawn digital sketches), such a hierarchy can be generated off-line and only once. To support progressive sketching and editing sessions or to render 3D line models, this hierarchy *must* be created dynamically and interactively whenever strokes move on the screen (e.g. view point changes) or are added/deleted. To incorporate all these scenarios, we use a state-of-the-art data structure called a deformable spanner [GGN06] for efficient dynamic queries to ascertain proximity of screen projected line strokes. We use these queries to create and maintain a time-coherent hierarchy for dynamic line merging and splitting, and generate appropriate visibility cues at no extra cost.

2. Related Work

The issue of line simplification that is central to our pipeline has been addressed in several ways in NPR. Strokes are simplified in a line drawing to maintain tone and overall shape and possibly for rendering efficiency. Tone is quantified by measuring local screen-space density [GDS04], often for static line drawings [WM04]. Simplification is typically done by creating a level-of-detail (LOD) hierarchy of lines. Lines are prioritized and rendered according to this hierarchy. Creation of these levels of detail under various settings is the subject of many papers. Tonal art maps are used by Praun *et al.* [PHWF01] as an image-based method to create LODs in hatching. In the field of illustration, Winkenbach *et al.* [WS94] introduce the concept of indication to prioritize lines in textures to achieve control over density, albeit semi-automatically. With a sketch drawn in real time as its input, WYSIWYG-NPR [KMM*02] takes a user-centric approach by relying on manual specification of LODs when they cannot be extracted automatically. All these approaches simplify strokes using an accept/reject scheme, i.e. removing strokes to simplify the drawing. An interesting approach by Cole *et al.* [CDF*06] uses *priority buffers* to modulate line density for localized simplification effects. Their focus however, is on user-assisted customization of line drawings created from static models, and hence may not be suitable for automatically simplifying interactively sketched line drawings.

Our work is partly inspired by that of Barla *et al.* [BTS05] who merge strokes by using screen-space proximity and color-based clustering. However there are three primary differences between their work and ours. First, their hierarchy can be created only once for a given drawing (possibly off-line) as they assume that drawings can only be zoomed in or out. We create and manage a time-coherent hierarchy on-the-fly to allow an arbitrary sequence of **2D/3D** transformations in an interactive and dynamic setting. Secondly, they rely on a manual classification of strokes as contour and hatching and employ different simplification methods for them. We present a unified and automatic simplification strategy, whose advantages are discussed in Sections 4.2 and 6. Lastly, a change in their input parameters requires the hierarchy to be completely re-calculated (their ε-lines and ε-groups change with ε). Our implementation based on a spanner does not suffer from this restriction. This is an advantage because such parameters are model-dependent and often have to be determined interactively by trial-and-error.

3. Overview

Representation: Our input is an unordered set of 2D or 3D line strokes. Each stroke is in general a one-dimensional (parametric) curve. We first sample each stroke uniformly (in terms of its arc length) into a set of points and then represent the input model at two levels: strokes consisting of points and an independent set of points from all strokes. As we will see later, this dual representation is useful for proximity determination and stylization during rendering. Every point stores the tangent to the stroke at that point, its screen-space position and orientation (projected tangent), and rendering attributes like thickness, color, transparency, stroke texture id, etc. Our processing pipeline works fully in screen space.

Stroke Proximity: Given a set of strokes projected on the screen, we first determine which strokes are (at least partially) "near enough" to other strokes to affect their appearance. We define a parameter δ as the maximum distance between two strokes for them to affect each other. This parameter can be changed interactively. Section 4.1 explains stroke proximity in detail. Proximity calculations are also used to generate visibility cues to mitigate depth ambiguity and visual clutter. This is explained in Section 4.4.

Stroke Pairing And Simplification: To disambiguate multiple interactions (a single stroke can affect and be affected by many strokes), we pair every stroke with one other stroke that it affects the most, according to proximity, color, local gradient and extent of overlap as a percentage ρ. This parameter can also be changed interactively. We merge two paired strokes into a single stroke. Section 4.2 explains how we pair strokes and simplify them.

Hierarchy Maintenance: Stroke simplification results in a dynamic binary tree hierarchy. Whenever points move on the screen, this hierarchy must be updated and an appropriate level in it must be chosen for rendering. Section 4.3 discusses how this hierarchy is maintained.

Rendering: Every point in the sampled stroke is rendered as an oriented, alpha-textured quad with the stroke's thickness and color. Each point's opacity modulates that of its texture to produce the final result. The texture can be changed to produce various styles.

4. Simplification and Rendering

In this section we explain our pipeline in detail. Sections 4.1, 4.2 and 4.3 discuss simplification, while Section 4.4 discusses generation of visibility cues.

4.1. Proximity

Line simplification can be thought of as a localized process, where strokes are affected only by other strokes near them. If a shape-defining feature like a contour is specified using many small strokes (a common sketching style), then proximity groups such small strokes together so that they may be approximated by a single long stroke. A local group of hatching strokes may be approximated by fewer strokes to maintain stroke density as the user zooms out. Thus, proximity is an effective measure of identifying groups of strokes that may be approximated by a different set of strokes.

To determine which strokes to simplify, one must determine which strokes are near each other in screen space. Data structures employed for this purpose must support adding and removing strokes efficiently. Moreover, as the strokes move incrementally on the screen, proximity should be re-calculated quickly to maintain interactive rendering rates.

Given a set of curves, it is difficult to efficiently determine proximity as they move. We resort to a divide-and-conquer approach–we pool the underlying points of every stroke, solve the proximity problem for points and then interpret the results at the stroke level. In general, we estimate the expected "pairability" $E(S,T)$ between strokes S and T as

$$E(S,T) \propto Co(S,T) * \sum_{(p,q):p\in S, q\in T, d(p,q)\leq\delta} (\vec{p}.\vec{q}) \qquad (1)$$

where p and q are screen points with screen-space (normalized) tangents $\vec{p}$ and $\vec{q}$ and $d(.)$ is the Euclidean distance. Co(.) is the color similarity between S and T, but may include other suitable metrics as well. If two strokes are very near each other, then more pairs will be found, leading to a greater expected value. Thus $E(S,T)$ measures how "near", "locally parallel" and similar in color S and T are.

Many solutions for point proximity problems exist in computational geometry [CK95, Epp00]. As strokes can be added, deleted, or dynamically modified, we require a data structure that supports efficient nearest-point queries under motion, dynamic insertion and deletion. Hence, we use the $(1+\varepsilon)$-deformable spanner by Gao *et al.* [GGN06].

$(1+\varepsilon)$-deformable Spanner

For a set of points in $\mathbb{R}^d$, an s-spanner is a graph on the set such that any pair of points is connected via some path in the spanner whose total length is at most s times the Euclidean distance between the points. A $(1+\varepsilon)$-deformable spanner (for any $\varepsilon > 0$) is a sparse spanner that is suitable for dynamic sets of points. Because of its hierarchical construction and sparseness, a $(1+\varepsilon)$-spanner efficiently "repairs" itself incrementally whenever points move *continuously*. Thus, in the context of rendering, it works on the notion of using results from the previous frame to determine those for the current frame. Moreover it supports dynamic insertion and deletion of points, making it suitable for our setting. Specifically, for a set of n points in $\mathbb{R}^d$ bounded by an aspect ratio α (the ratio of the maximum and minimum distance between two points), the $(1+\varepsilon)$-spanner supports insertion and deletion of a point in $O(\frac{h}{\varepsilon^d})$ time, where $h = O(\log_2 \alpha)$. The nearest-point query – given a set of points, for each point p, enumerate all points within a distance δ from p is defined as a standard operation on this spanner. For k such pairs, the $(1+\varepsilon)$-spanner supports this query in $O(k+n)$ time. Our pipeline uses this data structure *as-is* (like a black box). We refer the reader to Gao *et al.* [GGN06] for further details.

We project all strokes onto the screen and build the $(1+\varepsilon)$-spanner on these projected 2D points. After any screen-space movement, we update(repair) the spanner and query it to return all pairs of points that are within a distance δ (Section 3) from each other. As long as the movement is *incremental* (points do not move by a large distance abruptly), the updating operation is efficient irrespective of the actual nature of movement. Thus our pipeline is not limited to simple rigid transformations–*any* incremental motion can be handled. Please see the accompanying video for an example.

The parameter ε controls the extent of approximation of level i in its parent level $i-1$ (we chose $\varepsilon = 16$ empirically for all results in this paper). Our experiments indicated that increasing the value of ε increases the time to "repair" the spanner during every frame.

Henceforth, all screen-projected points are represented in lower case while all strokes are represented in upper case. A screen-projected point p has a screen-space orientation $\vec{p}$.

Point Pairing

All pairs returned by the spanner are not useful in determining stroke-stroke proximity (e.g. the spanner returns pairs of adjacent points along the same stroke due to the absence of connectivity information in it). Intuitively, we try to select pairs that are near to each other, have similar colors and screen-space orientations.

During every frame, we build two tables from all the pairs returned by the spanner: $C(p,S)$ that maintains the "closest" point in every stroke S to a given point p, and $Q(S,T)$ that maintains the geometric likelihood that strokes S and T will be paired together (Q is the summation part of Equation 1). Let $p \longrightarrow q (p \in S, q \in T)$ be a candidate pair of points returned by the spanner. If $S = T$, we reject the pair. Otherwise, we determine a score $Sim(p,q)$ proportional to the Euclidean distance and color difference between p and q. We update the entries $C(p,T)$ and $C(q,S)$ with $Sim(p,q)$ if necessary. Then we add $\vec{p} \cdot \vec{q}$ to $Q(S,T)$ and $Q(T,S)$. Thus $Q(S,T)$ maintains a score that is proportional to the number of pairs between points in S and T respectively, modulated by the similarity between their local gradients. It may be noted that C and Q are sparse as they contain data only for points and strokes that are near each other.

4.2. Stroke Pairing and Simplification

Given updated sparse tables C and Q, we now consolidate the results to pair strokes. Simplification happens at the stroke level, maintaining continuity wherever appropriate. We maintain continuity geometrically by preventing strokes from disintegrating. Although *apparent* continuity (interpreting multiple short strokes as a single long stroke) is maintained, it is not guaranteed (please refer to Section 6)[†].

Intuitively, we pair stroke S with a stroke T if $\rho\%$ of the points of S are paired to some point in T with comparable corresponding local gradients. We select an ordered pair of strokes (S,T) for simplification if $Co(S,T)Q(S,T) \geq \rho\,|S|$. [‡] Our formulation allows us to handle fork cases (neither of the end points of S pair with T) seamlessly. Since entries in Q are averaged over a stroke, they vary smoothly as points move continuously. Thus simplification is also incremental and smooth, providing temporal coherence.

It is important to note that once (S,T) are paired and simplified to U in a particular frame, no other strokes can pair with either S or T until U again separates into them. Thus in the case that multiple strokes have sufficient overlap with a single stroke S to trigger simplification, the first of these strokes (in the order in which they were created) is paired with S, ruling out any other pairs with S during that frame.

Stroke Simplification

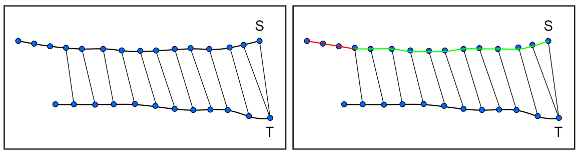
The problem of simplifying two strokes into one is difficult in general, because some correspondence between their points must be known for any interpolation scheme. This problem occurs in many applications: simplifying strokes in NPR, representing rough over-traced sketches by a representative stroke, etc. It is often addressed by classifying strokes and employing multiple simplifying strategies.

If the two strokes were parameterized in a common domain, the parametrization would serve as an implicit correspondence. There are existing methods to fit curves to noisy point data [Gos00] that parameterize points in a common domain. The strategy is to define a reference parameterized curve. Then for every point, the point on the curve that is nearest to it is determined and parameterized accordingly. While this seems inefficient for an interactive setting, we reuse the point pairs from the spanner to determine this correspondence.

Noting that at least $\rho\%$ of points of S are paired to some point in T for a pair (S,T), we choose T as our reference curve and parameterize it in $(0,1)$ using arc-length parametrization. For every point $p \in S$, if $C(p,T)$ exists, the

[†] Maintaining the integrity of special strokes like dashed lines depends on how they are represented in the application–if passed to the pipeline as one long stroke, continuity is maintained.

[‡] It may happen that $Co(S,T)Q(S,T) \geq \rho\,|S|$ but $Co(T,S)Q(T,S) \leq \rho\,|T|$. This happens if S is shorter than and so overlaps only a small portion of T.



parameter value at p is trivially known. This scheme divides S into a set of alternating parameterized (green above) and (possibly) unparameterized (red above) segments. We parameterize these segments using interpolation (if it lies between two parameterized segments) or extrapolation (otherwise). Another pass through S makes this parametrization consistent (monotonically increasing/decreasing). Then we sort the points in S and T according to the common parameter and fit a curve through this point set (a similar technique for parameter propagation was used by Kalnins *et al.* [KDMF03] to produce coherent silhouette animations).

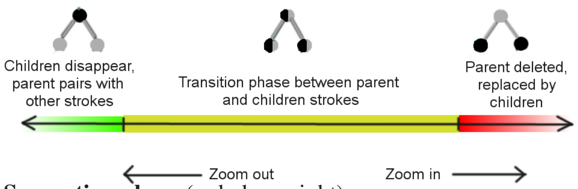
In regions where S and T do not coincide, this point set may cause a zig-zagged curve. To minimize this effect we choose Rational Gaussian curves [Gos95] for interpolation. RaG curves work by centering a Gaussian function at every control point and using these weights to calculate interpolating points on the curve. They offer good control over local and global smoothness by changing the standard deviations of the interpolating Gaussians (we use $\sigma = \frac{3}{|S+T|}$ for all models). We use pre-computed Gaussian tables to improve speed. Figures 2 and 4 show how this single simplification scheme works well for both contour strokes (maintaining shape in the Gandhi and boat sketches) and hatching strokes (hull of the boat).

For a pair (S,T) simplified to a stroke U, U records a representative pair of points $(p_S \in S, q_T \in T)$ and the Euclidean distance $d_{first}(p_S,q_T)$ between them at the time of simplification. They are used to decide whether to move down the hierarchy (replace U with S and T).

4.3. Hierarchy Maintenance

Simplification creates a localized hierarchy of strokes. Strokes should smoothly merge into their parent, or should smoothly separate into their children for coherent animation.

Based on strokes S and T simplified into stroke U with the parameters (p_S,q_T) and $d_{first}(p_S,q_T)$ as explained previously and distance $d_{curr}(p_S,q_T)$ in the current frame, a life cycle of U can be defined in terms of the following phases:


Children disappear, parent pairs with other strokes
Transition phase between parent and children strokes
Parent deleted, replaced by children
Zoom out
Zoom in

Separation phase (red phase;right)

In this phase characterized by $d_{curr}(p_S,q_T) > l * d_{first}(p_S,q_T)$, S and T are apart enough to exist as separate strokes. Thus we descend one level down the hierarchy and render them instead of U. Points of S and T are added to the spanner, and those of U are deleted. $l > 1$ provides a wider hysteresis loop (so that points are not repeatedly added to/deleted from the spanner between frames).

Transition phase (yellow phase;middle)

In this phase characterized by $k * d_{first}(p_S,q_T) < d_{curr}(p_S,q_T) \leq l * d_{first}(p_S,q_T)$, U, S and T are rendered as if in transition between the two levels. Their opacity is governed by $d_{curr}(p_S,q_T)$, such that $\alpha(S) = \alpha(T) \propto \frac{d_{curr}(p_S,q_T)}{d_{first}(p_S,q_T)}$, $\alpha(U) = 1 - \alpha(S)$. $0 \leq k \leq 1$ creates a smooth transition.

Simplified phase (green phase;left)

In this phase characterized by $d_{curr}(p_S,q_T) \leq k * d_{first}(p_S,q_T)$, S and T are too near each other to exist as separate strokes. If they were in the transient phase in the previous frame (they would have had low opacity then to be in this phase now), we stop rendering them and set U at full opacity. Points of S and T are removed from the spanner, and those of U are added.

We implement this scheme using two lists. At every frame, one list is marked as the current list. Any strokes created during that frame (by simplification) are added to this list. We check every stroke in the list, and either push it (yellow phase) or its children (red phase) into the other list. The lists are swapped in the next frame. Any stroke that has already been paired (i.e. its ancestor exists in the list) is neglected.

To summarize, the following high-level operations are performed in order during every frame:

1. Verify the hierarchy. (Section 4.3)
2. Update the spanner, get and process all pairs. (Section 4.1)
3. Determine strokes pairs and simplify them. (Section 4.2)
4. Render the strokes.

4.4. Visibility Cues

If surfaces are available in case of 3D models, visibility cuing by occlusion may be performed easily in some cases. However in the general case of 3D curves, such occlusion may not be possible or desirable even if the surfaces that they lie on are available. For example, consider two 3D curves that lie on two known surfaces. How does one decide the surface bounds to get occluding polygons? Is it always conceptually sensible for one curve to occlude another?

We generate cues to alleviate depth ambiguity and visually de-emphasize distant parts of the model by further modulating the transparency of every stroke. Our technique is similar to the *haloed line effects* proposed by Appel *et al.* [ARS79] and Elber [Elb95]; however our pipeline implements it at no extra cost. For every intersection point of two strokes on the screen, we locally increase the transparency of the

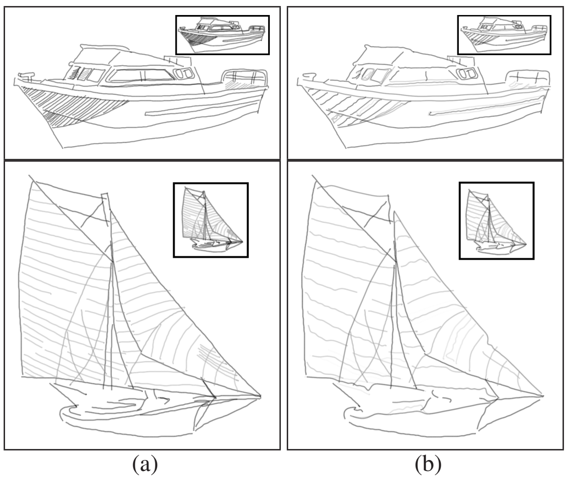


Figure 2: *Results on 2D sketches of objects. Column (a) shows the original sketch zoomed out to a certain level (thumbnail), while column (b) shows the sketch zoomed out and simplified to the same level (thumbnail) and the thumbnail magnified. The top row shows a sketched boat* ($\delta = 4, \rho = 85$)*, while the bottom row shows a sketched sailboat(*$\delta = 8, \rho = 85$*).*

stroke that is behind by a certain amount. Thus, the stroke appears "lighter" or hidden behind the stroke that is in front. These intersection points can be approximated from the pairs of points returned by the spanner. Such local modulation of transparency values results in a global visibility cuing effect. In Figure 3(b), the right wheel of the cart appears occluded due to these cues. In Figure 1(b), these cues reduce clutter when the Eiffel Tower is rendered from different distances.

5. Results

Figure 2 shows how our system can be used to render 2D sketches. These sketches were drawn on a tablet PC, using a simple program that allowed the user to draw strokes by tracing a photograph. Row 1 shows the 2D sketch of a boat. Notice how the hatching strokes on the hull and the contours of the boat are simplified correctly with our unified simplification technique. Row 2 shows another 2D sketch of a sailboat. The thumbnail shows how the wrinkles on the sail were simplified upon zooming out.

Figure 4 shows how our system can be used to render creative sketches directly drawn on computer by an artist. Typically, such sketches are ambiguous and are drawn by over sketching with many small strokes. The first row shows a character sketch of Mahatma Gandhi. Notice how various silhouettes are drawn with multiple smaller strokes. Column (b) shows our simplified rendering where such strokes are consolidated into fewer strokes, thereby preventing them from thickening and darkening. The second row shows a sketch of a lamp. While sketching, the artist went over the same regions with strokes of multiple colors, due to which there is some color mixing as the sketch is zoomed out

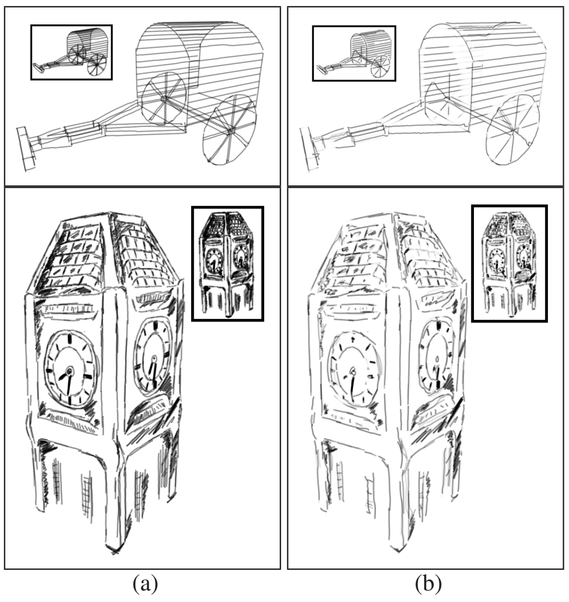


Figure 3: *Results on 3D sketches (created using the 3D6B interface [Kal05]). Column (a) shows the original sketch zoomed out to a certain level (thumbnail). Column (b) shows the sketch zoomed out and simplified to the same level(thumbnail) and the magnified thumbnail. First row: 3D sketch of a cart (*$\delta = 5, \rho = 85$*). Second row: two facades of a clock tower (*$\delta = 5, \rho = 80$*).*

(thumbnail in (c)). Some parts (yellow shading on the lamp shade) were drawn using a few long, winding strokes. The lamp simplifies into (d) if such strokes are taken in their entirety. It can be seen that such strokes are not simplified significantly as their length mitigates sufficient overlap with any other stroke to trigger simplification. To improve this, we first segment such strokes by monitoring abrupt curvature changes. Simplification after such segmentation is shown in (e). Notice how the lamp is now simplified to a greater extent and resembles the original sketch more (the two thumbnails), in terms of stroke placement and density.

Figure 5 shows a comparison with a result from Barla *et al.* [BTS05] by using the data of one of their sketches. Our pipeline simplifies the hatching strokes similar to theirs (without explicitly labeling them so). Result (c) was obtained by fixing δ and ρ so that zooming out to the same size as the thumbnail in (b) results in an image that looks similar to the original sketch in terms of shape and tone. Notice how the overall appearance of the thumbnail of (c) matches that of the original sketch. In particular, notice the locally darker hatches on the big branch on the right that are somewhat preserved in our thumbnail. This is reflected in the sparse dark stroke on this branch in the magnified thumbnail in (c).

Figure 3 shows how 3D sketches (here drawn using the 3D6B interface [Kal05]) can be rendered using our system. Notice how our visibility cues correctly depict the depth by

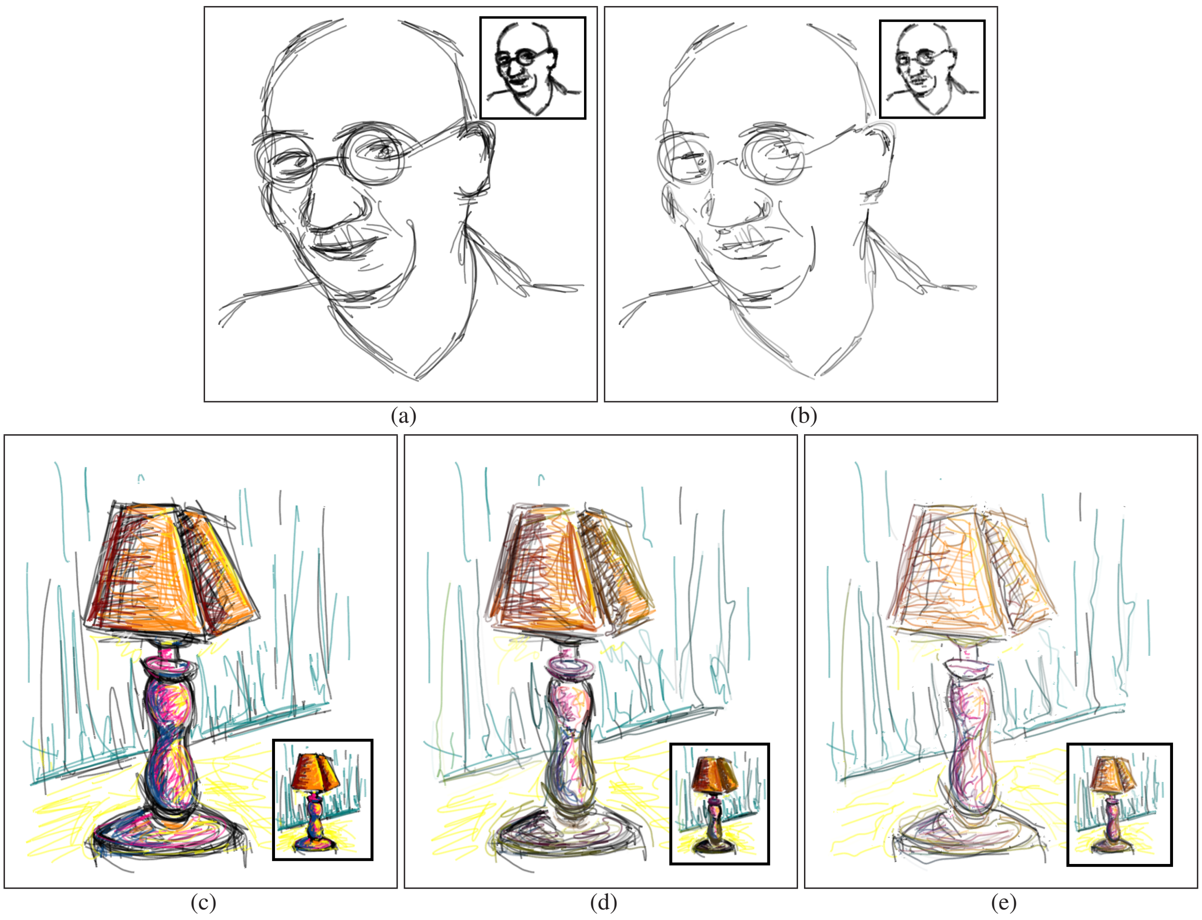


Figure 4: *Results on artistic sketches. First row: (a) a sketch of Mahatma Gandhi made by an artist, zoomed out to a certain level (thumbnail). (b) the simplified sketch zoomed out to the same level (thumbnail) and the thumbnail magnified ($\delta = 5, \rho = 80$). Notice the typical over-tracing sketching style in the sketch that is suitably simplified by our rendering. Second row: (a) an artistic sketch of a lamp, zoomed out to a certain level (thumbnail). This sketch was made by the artist by sketching repeatedly with overlapping strokes of various colors, giving it a composite appearance. Notice how parts of the lamp shade are sketched using a single winding stroke. If such strokes are taken in their entirety, the simplified result appears as in (b)($\delta = 6, \rho = 85$). To improve on this, we break the stroke into segments so that they may be simplified, producing the result in (c). Notice how the strokes in the lamp shade have been simplified correctly due to this segmentation.*

"occluding" the right wheel of the cart (first row). In the second row , notice how the tone of the dense strokes on the roof, the shingle pattern and the shape of the clocks are retained. Figure 1 shows how our system can be used to render 3D wireframe CAD models ($\delta = 3, \rho = 80$). This Eiffel Tower model was obtained from Google 3D Warehouse [goo] and rendered without any surface information.

Table 1 shows the average frame rates for the results shown in this paper. These frame rates were calculated for a resolution of 950×950 on a desktop machine with a 3.0 GHz Intel Pentium 4 processor and 1 GB RAM. The implementation is fully CPU-based. As the screen size decreases, the number of points in the spanner decreases due to simplification, leading to faster frame rates.

Our system maintains consistent stroke density and creates smooth transitions as the model moves. The accompanying video shows how various models shown here simplify as they are transformed, preventing strokes from smearing each other. Temporal coherence can be appreciated more by closely observing parts of the video where a model is simplified but rendered at its original size, as it undergoes smooth and often unnoticeably gradual transitions.

6. Qualitative Analysis and Limitations

The main highlight of our pipeline is that it works in a dynamic setting for an arbitrary sequence of transformations. Towards this goal, the most expensive operation is to determine proximity between strokes, which we perform efficiently with the $(1+\varepsilon)$-spanner. The cost of dynamism

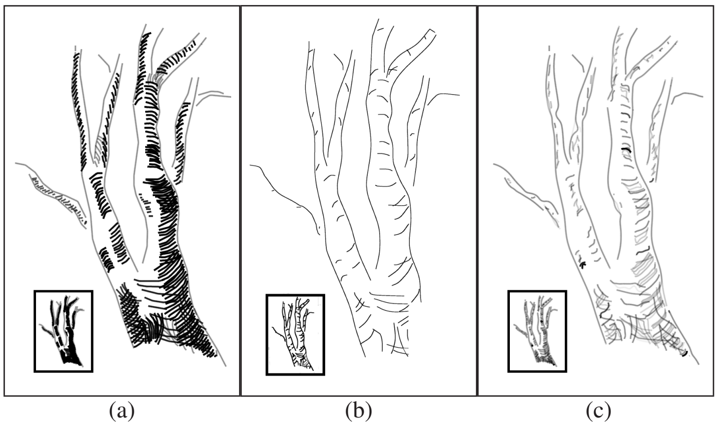


Figure 5: *Comparison with previous work. (a) A sketch from Barla et al. [BTS05]. (b) Their result upon zooming out. (their result was darkened to better compare with our rendering style. The number and shapes of strokes were unchanged. Reproduced with permission from the authors) (c) Our result when zoomed out to the level shown in the bottom left thumbnails ($\delta = 6, \rho = 90$). Our simplification preserves the apparent tone produced by the hatching strokes as in their result, shown in (b). The result was obtained by fixing δ and ρ empirically and zooming out till a satisfactory image at the same size was obtained.*

Model	$N_{strokes}$	N_{pts}(initial)	$FR_{avg}(fps)$
Sailboat	281	4817	7.7
Boat	359	6000	9.24
Gandhi	757	7197	12.52
Lamp	2559	31444	3.16
Tree	2376	34424	11.3
Clock tower	961	21445	2.69
Cart	162	8851	5.45
Eiffel	2742	22119	5.11

Table 1: *Performance results. $N_{strokes}$: number of strokes in the model, $N_{pts}(initial)$: number of points initially added into the spanner. All frame rates were captured at a* 950×950 *resolution.*

principally comes in the form of per-frame updates of the spanner, which is why the performance of our pipeline is interactive, but not real-time. It may be possible to improve the performance of the spanner if only specific transformations are allowed. Implementation of the spanner *as-is* on the GPU will greatly boost performance without compromising on generality. We have reserved this for the near future.

The main advantage of a unified simplification strategy (Section 4.2) is that strokes need not be annotated (e.g. "contour" and "hatching") explicitly. In a sketching session, this may have to be done manually which would seem contextually unnecessary. However the notion of *continuity* may be compromised in some cases. Consider a (implied) long stroke that is drawn using two strokes A and B overlapping end-to-end. If a third stroke C pairs with A to create a simplified stroke D, then D and B may not look like a continuous long stroke as A and B were meant to be. As small values of δ are normally used to prevent over-simplification, such an effect is usually not obvious. However it is theoretically possible.

The two parameters δ and ρ offer limited control over subjective simplification effects. Both δ and ρ are applied globally, i.e. to the entire (projection of) model. Although this approach makes them intuitive to change, it makes local customizations difficult. For example, it is not possible to simplify a local region more while retaining the original simplification elsewhere. A possible solution could be to allow additional localized simplification effects like those by Cole *et al.* [CDF*06] as a post-process. Another issue concerns the "constancy" of δ, i.e. although δ can be changed by the user, it is independent of the actual scale of the model. This creates artifacts when the model is scaled down greatly. At that stage the model may be over-simplified. A better strategy could be to modulate the value of δ with the overall scale of the model, but achieving a proper control is difficult because this correspondence is often model-dependent. Such a strategy may also make δ less intuitive to change.

Another issue concerns our stroke simplification algorithm. Our parameter propagation when creating a merged stroke is based on a simple merging procedure. This may cause strokes to jump (spatially, not temporally) from the unpaired parts to those that are paired. This, along with our method of determining the standard deviations of the RaG curves sometimes creates wiggly strokes (some strokes in Figure 5(c)).

7. Conclusions and Future Work

In this paper, we introduce a pipeline that performs line simplification and visibility cuing to create line drawings from purely 2D/3D line-based models obtained from various types of data. By using a $(1+\varepsilon)$-spanner to determine stroke proximity, we dynamically build and maintain a stroke hierarchy based on the high-level principles of proximity and continuity. We also generate localized visibility cues at no extra cost to produce a globally less ambiguous and uncluttered line drawing. The line drawings thus created are geometrically meaningful and temporally coherent.

Synthesis of line drawings is a complementary and more difficult problem, in which strokes are synthesized (instead of simplified) to achieve the same goals of shape and tone preservation. In the future, we wish to extend our pipeline to synthesize line drawings. In addition to line-based NPR, we would like our pipeline to support other styles of abstract drawings like point stippling. Point-based and line-based NPR effects are often treated exclusively because of operations required to realize each of them. We envision a hybrid pipeline that feeds off a common efficient representation of underlying data, supports both styles simultaneously and switches between them interactively, and that offers intuitive controls for an artistic user.

Acknowledgements

Support for this work includes a University of Minnesota Digital Technological Center Seed Grant, a Microsoft Research Gift, NSF CCF-0238486 (CAREER) and NSF DMS-0528492 (MSPA-MCS) and a generous travel grant by the Department of Computer Science at the University of Minnesota. We thank Betsy Wall for the Gandhi and lamp sketches and Poonam Shanbhag for the tower sketch. We also thank the anonymous reviewers for their critique and useful comments.

References

[ARS79] Appel A., Rohlf F., Stein A.: The haloed line effect for hidden line elimination. *Proc. SIGGRAPH* (1979), 151–157. 5

[BKR*05] Burns M., Klawe J., Rusinkiewicz S., Finkelstein A., DeCarlo D.: Line drawings from volume data. *Proc. SIGGRAPH* (2005), 512–518. 1

[BTS05] Barla P., Thollot J., Sillion F.: Geometric clustering for line drawing simplification. *Proc. EGSR* (2005), 183–192. 2, 6, 8

[CDF*06] Cole F., DeCarlo D., Finkelstein A., Kin K., Morley K., Santella A.: Directing gaze in 3D models with stylized focus. In *Proc. EGSR* (2006), pp. 377–387. 2, 8

[CK95] Callahan, Kosaraju: Algorithms for dynamic closest pair and n-body potential fields. In *SODA: ACM-SIAM Symp. Discrete Algorithms* (1995). 3

[DXS*07] Dorsey J., Xu S., Smedresman G., Rushmeier H., McMillan L.: The mental canvas: A tool for conceptual architectural design and analysis. *Proc. Pacific Graphics* (2007), 201–210. 1

[Elb95] Elber G.: Line illustrations in computer graphics. *The Visual Computer 11*, 6 (1995), 290–296. 2, 5

[Epp00] Eppstein D.: Spanning trees and spanners. In *Handbook of Computational Geometry*, Sack J.-R., Urrutia J., (Eds.). Elsevier, 2000, ch. 9, pp. 425–461. 3

[GDS04] Grabli S., Durand F., Sillion F.: Density measure for line-drawing simplification. In *Proc. Pacific Graphics* (2004). 1, 2

[GGN06] Gao J., Guibas L. J., Nguyen A.: Deformable spanners and applications. *Comput. Geom. Theory Appl. 35*, 1 (2006), 2–19. 2, 3, 4

[goo] Google 3d warehouse. http://sketchup.google.com/3dwarehouse/. 7

[Gos95] Goshtasby A.: Geometric modelling using rational gaussian curves and surfaces. *Computer-Aided Design 27*, 5 (1995), 363–375. 5

[Gos00] Goshtasby A.: Grouping and parameterizing irregularly spaced points for curve fitting. *ACM Trans. Graph. 19*, 3 (2000), 185–203. 4

[HZ00] Hertzmann A., Zorin D.: Illustrating smooth surfaces. *Proc. SIGGRAPH* (2000), 517–526. 1

[IOOI05] Ijiri T., Owada S., Okabe M., Igarashi T.: Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints. *Proc. SIGGRAPH* (2005), 720–726. 1

[JNLM05] Jeong K., Ni A., Lee S., Markosian L.: Detail control in line drawings of 3D meshes. *The Visual Computer 21*, 8-10 (September 2005), 698–706. 1

[Kal05] Kallio K.: 3D6B editor: Projective 3d sketching with line-based rendering. In *Proc. Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2005), pp. 73–79. 1, 6

[KDMF03] Kalnins R. D., Davidson P. L., Markosian L., Finkelstein A.: Coherent stylized silhouettes. In *Proc. SIGGRAPH* (2003), pp. 856–861. 5

[KHR04] Karpenko O., Hughes J. F., Raskar R.: Epipolar methods for multi-view sketching. In *Proc. Eurographics Symposium on Sketch-Based Interfaces and Modeling* (2004), pp. 167–174. 1

[KMM*02] Kalnins R., Markosian L., Meier B., Kowalski M., Lee J., Davidson P., Webb M., Hughes J., Finkelstein A.: Wysiwyg npr: drawing strokes directly on 3d models. *Proc. SIGGRAPH* (2002), 755–762. 2

[PHWF01] Praun E., Hoppe H., Webb M., Finkelstein A.: Real-time hatching. *Proc. SIGGRAPH* (2001), 581. 2

[SC04] Shesh A., Chen B.: SMARTPAPER–an interactive and user-friendly sketching system. *Proc. Eurographics* (2004), 301–310. 1

[SP03] Sousa M. C., Prusinkiewicz P.: A few good lines. In *Proc. Eurographics* (2003), pp. 381–390. 1

[WM04] Wilson B., Ma K.-L.: Rendering complexity in computer-generated pen-and-ink illustrations. In *Proc. NPAR* (2004), pp. 129–137. 1, 2

[WS94] Winkenbach G., Salesin D.: Computer-generated pen-and-ink illustration. *Proc. SIGGRAPH* (1994), 91–100. 2

[XC04] Xu H., Chen B.: Stylized rendering of 3d scanned real world environments. In *Proc. NPAR* (2004), pp. 25–34. 1

[ZHH96] Zeleznik R., Herndon K., Hughes J.: Sketch: an interface for sketching 3d scenes. *Proc. SIGGRAPH* (1996), 163–170. 1