EUROGRAPHICS 2003 / P. Brunet and D. Fellner
(Guest Editors)

Volume 22 (2003), Number 3

A Few Good Lines: Suggestive Drawing of 3D Models

Mario Costa Sousa Przemyslaw Prusinkiewicz

Department of Computer Science
University of Calgary, Canada

Abstract

We present a method for rendering 3D models in the traditional line-drawing style used in artistic and scientific illustrations. The goal is to suggest the 3D shape of the objects using a small number of lines drawn with carefully chosen line qualities. The system combines several known techniques into a simple yet effective non-photorealistic line renderer. Feature edges related to the outline and interior of a given 3D mesh are extracted, segmented, and smoothed, yielding chains of lines with varying path, length, thickness, gaps, and enclosures. The paper includes sample renderings obtained for a variety of models.

1. Introduction

Humans interpret line drawings remarkably well, being able to perceive and understand 3D object structures from very sparse collections of lines [1]. This is the main reason for the expressive power of line drawings that are used by artists and scientific illustrators to effectively represent the form of 3D objects. Such drawings are termed *pure* line drawings if they consist entirely of lines that define the edges of shapes and use no tones [2, 3] (figs. 1, 2, 3.)

In this paper, we present an automated method for direct non-photorealistic rendering of 3D triangle meshes as line drawings. Our system reproduces the artistic principle of *suggestion* or *indication*, in which lines are used with economy, and the expressive power of illustrations results from engaging the imagination of the viewer rather than revealing all details of the subject. We focus on the *loose* drawing style [4], in which lines are made with gestures that convey a spontaneous rather than a carefully constructed look.

To recreate this style in NPR, we consider three elements of illustrator's thinking [2, 3, 4, 5]:

Shape feature selection, or where to place the lines? Before starting to draw, illustrators thoroughly study the subject to be rendered, focusing on the geometric forms that give the subject its overall shape. They consider both the lines that define the outline of the object (silhouettes and boundaries) and features that define the interior volumes and surfaces, such as creases, ridges, and valleys.

Figure 1: *Beethoven (rendered by our system), looking at violin (rendered by John Blackman [6]).*

Line economy control, or how many lines to place? Illustrators control the amount of lines to be placed by following the principle that "less in a drawing is not the same as less of a drawing" [3]. Extraneous details are visually eliminated, reducing the subject to simple lines depicting key shape features.

Linear phrasing, or how to draw the lines? A significant challenge for the illustrator is to achieve a 3D sense in a drawing, given that a line is by nature a 2D trace of an

object in a plane. To address this challenge, the illustrator shapes and connects the feature lines of the objects in different ways, subtly varies their thickness and lengths, inflects them, breaks them, and puts them in various relations with respect to each other. These actions and their effects are collectively known as *linear phrasing* [3] †.

Figure 2: *Examples of real pen-and-ink* pure *line drawings: flowers* [7]*, a French horn* [6]*, and a woman* [4]

Most of the effects pertinent to linear phrasing can be achieved by a combination of three specific elements:

(1) Weight control consists of suggesting shapes and volumes by drawing lines that are thicker at certain curvatures and junctions. Also, parts in the focus of attention are drawn with thicker lines. For example, observe how the weight control improves the overall shape depiction in Figure 3.

(2) Definition of connectives consists of specifying various ways in which lines relate to each other. As shown in Figure 4, these relations occur when lines are connected together, broken by gaps, or emerge from each other. Again, observe how the insertion of linear connectives improves shape depiction in Figure 3.

(3) Definition of enclosures consists of indicating object outlines with lines that are not completely closed, but nevertheless give a fair idea of the intended shape. Enclosures are typically defined by combining the connectives shown in Figure 4(a). They may include gaps in otherwise continuous linear sequences (fig. 2, in particular the drawings of the plants and woman), and sequences in which the gaps are longer than the visible segments.

† The term *phrase* comes from a musical analogy: just as the musician *phrases* the shape of the melodic material, so too does the illustrator who *phrases* the lines [3]. The medium of choice is usually pen-and-ink with flexible nibs, where all sorts of qualities may be suggested at different parts of the line's track.

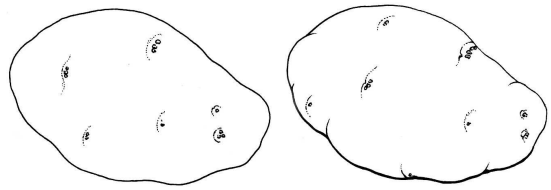


Figure 3: *Real pure line drawing of a potato* [8]*. Left: constant weight outline with inner lines suggesting a more discoid shape. Right: adding more weight to the bottom part of the outline and including short inner lines emerging from different locations in the outline creates a much better visual effect that suggests a rounded shape.*

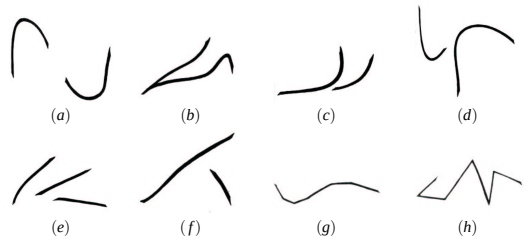


Figure 4: *Relationships of linear connectives common in pure line drawings* [3]*: (a) gaps between sections of a line; (b, c, d) line emerging from a section of another line and following a different direction; (e, f) spaced groups of lines; (g, h) different degrees of continuity in connected line segments.*

2. Previous work

This paper falls into one of the main areas of NPR, namely the creation of line drawings. More specifically, it deals with the automatic rendering of geometric models using a small number of stylized feature lines. The importance of such rendering was initially pointed out by Winkenbach and Salesin [9] and Strothotte et al. [10]. Below, we review object-space methods in which, as in our method, the NPR pipeline directly accesses the geometry of the 3D objects, as opposed to post-processing 2D images. We consider these methods from five points of view.

(1) Line economy control. Strothotte et al. [10] presented a system that allows the user to interactively control the level of detail in selected areas of the rendered image, by increasing or decreasing the number of strokes. The system

enhances these details by varying line styles. Winkenbach and Salesin [9] presented a related semi-automatic approach, in which the user controls the number of strokes.

(2) Shape feature selection. Most research has been devoted to silhouette detection [11, 12, 13, 14, 15, 16]. Other types of lines extracted from the model included ridges and valleys [17], creases [18], and lines related to the principal directions of curvature [13, 19, 20].

(3) Weight control. A typical approach is to represent the strokes as parametric curves $p(t)$ [9, 10, 11, 21]. Offset vectors are then added at particular values of parameter t, resulting in control points for a new parametric curve $q(t)$. The offset vectors depend on factors that influence real drawing strokes (e.g., pressure applied to the drawing point and hand gestures), and can be calculated on the fly or precomputed and stored in lookup tables. Northrup and Markosian [16], Kaplan et al. [22], and Markosian et al. [23] proposed methods in which line widths are scaled depending on the distance from the viewer and the distance from other strokes in the object. In addition, Kaplan et al. [22] proposed a weighting process which scales the silhouette line widths according to the lighting of the object.

(4) Insertion of connectives. The focus has been on techniques for connecting feature edges (mainly silhouettes) to form long and smooth strokes [11, 22, 24], and on the insertion of gaps between stroke segments (fig. 4(a)). Such gaps can be defined by functions that capture the raising and lowering of the drawing point at particular values of parameter t in the parametric representation of a stroke, $p(t)$. As in the case of weight control, these functions can be computed on the fly [9, 10, 21] or queried from precomputed lookup tables [11].

(5) Definition of enclosures. Two approaches have been explored: the explicit insertion of gaps along silhouette lines (as explained above) and the use of a *contrast enhancement* operator [25] that exposes the silhouette of a shaded model by exaggerating the brightness of its contour edges, including the silhouette.

In our work, we include some of these techniques to reproduce pure loose line drawings.

3. Our approach

Our algorithm takes as input a single 3D triangle mesh in the *.obj* format. A *half-edge* data structure [26] is then constructed to maintain edge adjacency information. We do not consider illumination and surface reflectance information, and do not use simulation models for ink and paper. The main steps of the algorithm are performed in the object space, and are as follows:

1. Feature lines are extracted and classified.
2. On this basis, several graphs are constructed as an input for chaining, which is the connection of lines of the same type into sequences.
3. These chains are extruded into 3D (perpendicular to the object's surface), creating ribbons of width dependent on a selected measure of surface curvature.
4. Spline curves are fitted to the edges of the ribbons, resulting in a smooth representation.

The final image is obtained by projecting the ribbons on the projection plane. OpenGL performs the visibility computation (using a Z-buffer) and the final rendering of the ribbons.

In terms of the elements of pure line drawing reviewed in the previous sections, the first step is the calculation of shape features. Line weight is a function of the surface curvature. Level of detail is controlled by feature thresholding and by the length of curves fitted to the chained feature lines. The connectives are not specified explicitly; nevertheless the user can control the final appearance of the image to some extent by changing the order of the spline curves.

The following subsections describe each step in more detail.

3.1. Feature edge classification

Our system distinguishes five types of feature edges: silhouette, boundary, crease, cap, and pit edges. We first examine every edge in the half-edge data structure to identify edges related to the outline of the object.

Silhouette edges are shared by a front-facing and back-facing polygons,

Boundary edges are incident to only one polygon.

Next, we extract three types of interior edges:

Creases depict folds within the object outlines. They correspond to edges shared by two front-facing polygons, whose normal vectors make an angle θ within the limits (min, max) specified by the user; we assume that $0 \leq min \leq \theta \leq max \leq 180$.

Cap and **pit** edges are situated in convex and concave regions of surfaces, respectively (fig. 5) [‡]. In our system, cap edges have both vertices situated in convex areas, and pit edges have both vertices in concave areas.

[‡] Surface geometry can be classified into six categories defined by the values of the principal curvatures: elliptical (convex when both curvatures are positive or concave when both curvatures are negative), hyperbolic (convex in one direction and concave in the other), flat (zero curvature in both directions), and cylindrical (convex when curvature is positive in the non-flat direction or concave otherwise). We have observed that artists tend to focus on depicting elliptical regions [2, 3].

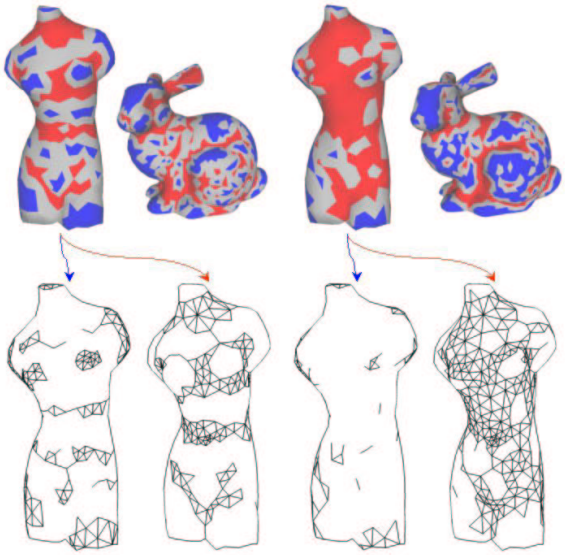


Figure 5: *Results of the two methods for extracting cap and pit edges, implemented in our system. Top row: the extraction of caps (blue) and pits (red) typically results in the formation of* feature regions*. They consist of triangles with all three edges being feature edges. Bottom row: the* networks *of cap and pit edges.*

Curvature Estimation

Curvature estimation of a triangulated mesh is a non-trivial problem [29]. Furthermore, it is not obvious what type of curvature is most appropriate for extracting regions relevant to our application. We experimented with two methods: the first method, proposed by Turk [27], finds approximate maximum curvature; the second method, by Nomura and Hamada [28], selects concave and convex regions using a method related to the calculation of mean curvature. Examples of regions selected using both methods are shown in fig. 5. Both methods led to useful practical results.

3.2. Building chains of feature edges

The edge classification process results in many disconnected feature edges. We now link them together to create *chains* of line segments with the same features. [§] To this end, we build five graphs G, one for each type of feature edges. These graphs can be *undirected* or *directed*; in the latter case, we assign directions to edges at random. Edges forming a chain must have consistent directions, hence chaining of directed edges yields a larger number of shorter chains, compared to the undirected case. This difference eventually translates to different visual quality of the rendered models.

We implement the feature graphs G using the *adjacency list* representation [31]. Thus, for each feature vertex v_i, we store a list of pairs (v_j, e_k), where v_j is a feature vertex adjacent to v_i, and e_k is a feature edge between v_i and v_j. In addition, for each vertex v_i, we store its *indegree* (the number of edges that lead to v_i) and its *outdegree* (the number of edges that emanate from v_i) [31]. For undirected graphs, we assume that every vertex v_i has its *indegree* equal to the *outdegree*. While chaining edges of undirected graphs, the starting points for chains are chosen arbitrarily. In the case of directed graphs, we attempt to start chains at *sources* (vertices with no incoming edges, $indegree = 0$) and finish at *sinks* (vertices with no outgoing edges, $outdegree = 0$). The following two algorithms, valid for both directed and undirected graphs, describe the chain building process:

```
BUILD-ALL-CHAINS(G)
 1  set_of_chains ← EMPTY
 2  while true
 3    do sources ← FindSources(G)
 4       if sources = EMPTY
 5         then return set_of_chains
 6       while sources ≠ EMPTY
 7         do v_s ← PickRandomSource(sources)
 8            C ← BuildSingleChain(v_s, G)
 9            insert C in set_of_chains
10            for all vertices v_i ∈ sources
11               do if v_i.outdegree = 0
12                     then remove v_i from sources
```

```
BUILD-SINGLE-CHAIN(v_s, G)
1  chain ← EMPTY;  v_i ← v_s;  insert v_i in chain
2  while true
3    do (v_j, e_k) ← PickRandomPath(v_i, G)
4       if v_j = EMPTY
5         then return chain
6       insert v_j in chain
7       update indegree and outdegree of v_i and v_j
8       flag edge e_k as visited
9       v_i ← v_j
```

In line 3 of the second algorithm, $PickRandomPath()$ returns $(v_j, e_k) \leftarrow (EMPTY, EMPTY)$ if $v_i.outdegree = 0$, which means that a complete chain has been built (i.e. we have reached a *sink* vertex); otherwise it returns any pair (v_j, e_k) where e_k has not yet been visited. In line 7, for directed graph, we subtract one unit from the outdegree of v_i and one unit from the indegree of v_j. For undirected graphs, we subtract one unit from both the indegree and the outdegree of v_i and v_j.

[§] A problem with extracting silhouette curves from polygon meshes is that the resulting curves may be jagged, because the meshes are just approximations of the underlying continuous surfaces. Although methods to address this problem exist (e.g. [13, 16, 18, 30]), we have not implemented them, because we observed that the silhouette artifacts do not compromise the quality of our results.

Figures 6 and 7 show examples of chain extraction using directed and undirected graphs, respectively. In both figures, graphs nodes are labeled *Vertex ID (indegree, outdegree)*. For a sample **directed graph** (fig. 6), the set of sources is initially equal to {1, 2}, and the set of sinks to {6, 8, 9}. Given that source and path selection are both random processes (line 7 in *BuildAllChains()* and line 3 in *BuildSingleChain()*, respectively), the chaining process may begin with vertex 1, and the first chain may be defined by the vertex sequence {1, 4, 5, 10, 6}. Figure 6 shows the updated values for the *indegree* and *outdegree* of vertices, and the resulting curve to the right of the graph. Before the next chain is extracted, nodes {1,10} will be removed from the graph (lines 10-12 in *BuildAllChains()*), resulting in *sources* = {2} and *sinks* = {6, 8, 9}. At the end, three other chains will be extracted: {2, 4, 3, 9}, {5, 7, 8}, and {5,6}. In the case of an **undirected graph** (fig. 7), let us assume that the chain = {1, 4, 5, 10, 6, 5, 7, 8} has been extracted first. The updated *indegree* and *outdegree* values and the resulting curve are shown in the figure. Before the next chain is extracted, nodes {1, 5, 6, 7, 8, 10} will be removed, resulting in the set of sources equal to {2, 3, 4, 9}. Further chains can be extracted starting with any vertex, and may yield the following chain combinations: {2,4,3,9}; ({2,4}, {4,2}, {4,3,9}); or ({3,9}, {3,4,2}).

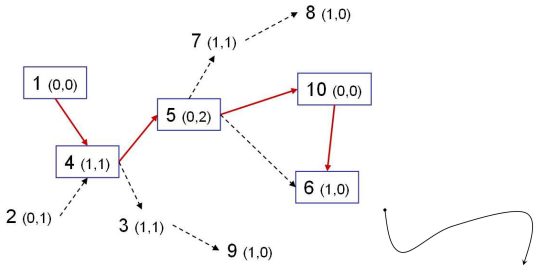


Figure 6: *Example showing one chain extracted from a directed graph. Boxes surround the vertices that are part of the chain.*

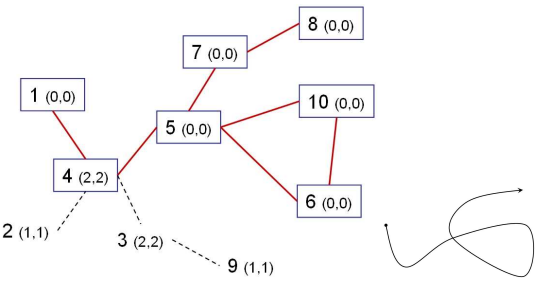


Figure 7: *Example showing one (self-intersecting) chain extracted from an undirected graph.*

3.3. Line weight computation

The choice of how thick the lines should be depends on the level of detail and complexity of shapes, on the illustrator's decisions concerning the emphasis, and on the overall design balance required for the drawing. For the purpose of suggesting 3D shapes, we consider the traditional approach in which illustrators vary ink placement based on the perceived curvature of the subject. In general, more ink is deposited along line sections passing over areas with high curvature [2, 3, 5]. We model this thickness variation by defining 3D polygonal ribbons supported by the mesh (fig. 8). To this end, we extrude (in 3D) each vertex v_i of each chain C in the direction of v_i's normal, thus producing an offset polyline $\overline{C}$. Each vertex $\overline{v}_i$ of $\overline{C}$ is thus defined by $\overline{v_i} = v_i + |w_i| N_i$, where N_i is the normal at vertex v_i, and w_i is a measure of curvature at v_i (subsec. 3.1). When the curvature is small or equal to zero, our system forces w_i to be equal to a user-specified minimum value ε, producing lines of a non-zero thickness.

3.4. Curve fitting

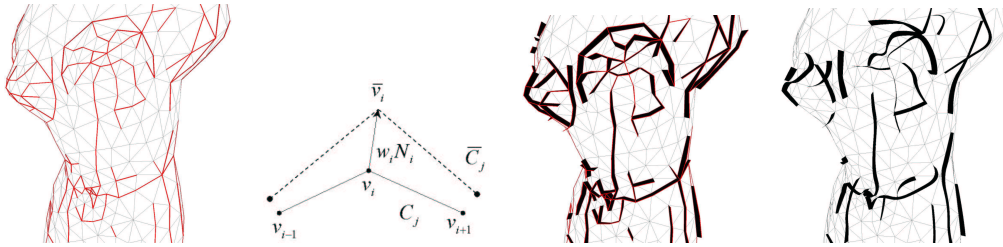
Strassman [32] and Pham [33] have explored the technique of smoothing line strokes by fitting curves to them. We follow the same approach by fitting B-splines to the pair of polylines $(C, \overline{C})$ (i.e., the original chain and its extrusion, fig. 8(c)), which results in a new pair of curves $(S, \overline{S})$ (fig. 8(d)). The user specifies the resolution of the B-spline and the order m of the curve. This controls in the smoothness of the line trajectory, which visually corresponds to the fluidity of ink placement along the line. The curves S and $\overline{S}$ are generated using n control points, equal to the number of feature vertices v_i in the chain C. Two main effects are achieved by simply increasing the order m of a curve segment: (1) the introduction of gaps (enclosure) at the ends where there is insufficient basis to support construction of a higher order curve (figs. 4, 9) and (2) the variation of the extent to which a curve is required to interpolate its control points, which affects the linear connective relations (fig. 4).

3.5. Rendering

In our system, all rendering calls use OpenGL. Visibility computations use the Z-buffer, with the triangles rendered in the background color (white), and the stroke ribbons rendered (in black) as triangle strips formed by the pairs of curves $(S, \overline{S})$. We avoid Z-buffer artifacts (inappropriately clipped or hidden strokes) by following the approach of Rossl et al. [20]. It consists of slightly translating the strokes along the vertex normal (the ribbon extrusion direction). This significantly reduces the stroke artifacts without compromising the quality of the results. The user can control the offset value, but little user intervention was needed while generating images presented in this paper.

4. Results and discussion

All the results were generated on an 1GHz Pentium III with a GeForce2 card. We selected 3D models representing a variety of subjects. For models with less then 5k triangles, the



(a) (b) (c) (c)

Figure 8: *Line weighting (subsec. 3.3) and smoothing (subsec. 3.4) processes: (a) chains C of feature edges (in red) used in subsequent extrude operations. The pairs* $(C, \overline{C})$ *(represented in 2D in (b)) define 3D ribbons across the mesh (c). (d) Curve fitting smoothes the 3D ribbons, creating the visual effects of ink fluidity and linear connectives/enclosure.*

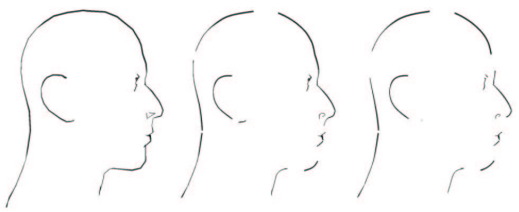


Figure 9: *Enclosure effect introduced by varying the order of the curve m across the silhouette chains (from left to right* $m = 2, 3, 4$*).*

rendering time was below one second (see Table 1 for details). The results show that our approach produces images similar to line drawings executed by hand, as found in artistic and scientific illustrations. To evaluate the system we chose representative pure line drawings (such as the ones from Figure 2) and used our system to duplicate the effect. Our evaluation is thus conducted by observing how close to the original drawings the computer-generated ones are.

Model	*#edges*	*s+b*	*s+b+c*	*all* [27]	*all* [28]
Man's head	2,043	1	1	1	1
Venus	2,127	< 1	< 1	< 1	< 1
Galleon	7,047	1	3	6	6
Bunny	7,473	3	4	5	4
Beethoven	7,663	3	5	9	9
Cow	8,706	2	4	9	11
Fireweed	13,421	13	19	31	30
Sunflower	14,784	22	31	39	39
Hyacinth	17,141	20	31	56	47
Castle	19,675	20	32	59	60
Car	24,969	19	42	95	95

Table 1: *Average times (in seconds) for rendering silhouettes + boundaries (*$s+b$*), silhouettes + boundaries + creases (*$s+b+c$*), and all feature edges. In the latter case, we distinguish between the calculation of caps and pits using the method proposed by Turk* [27] *and the method proposed by Nomura and Hamada (all* [28]*).*

Figure 10 illustrates the steps for rendering pure line drawings using our system. Starting with a 3D mesh, we add silhouettes and progressively render creases, caps and pits. Notice the enclosure at the silhouette suggesting the overall form of the model. Also notice how the strokes suggest hair, face and coat features. The increase in the number of lines is due to the user adjusting threshold values related to feature selection. The user also controls additional scaling of the line weight (subsec. 3.3) and adjusts the order of the curves fitted to the chain of line segments (subsec. 3.4). Figures 1 (left) and 12 show further results obtained using the same 3D model.

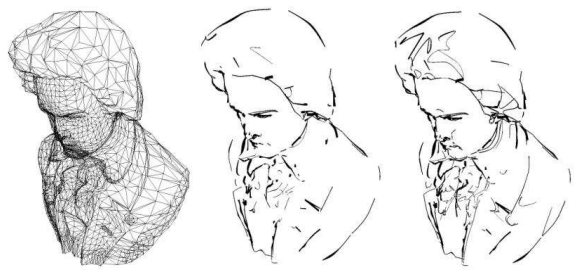


Figure 10: *Example showing typical results while interacting with our system. User selects features for display, adjusts their thresholds and controls the stroke attributes of weight and curve order. In this particular example, we start with the* ***Beethoven*** *mesh, add silhouettes, and gradually increase the number of crease, cap and pit strokes placed in the model.*

Two important effects appear in the final rendering due to the graph organization of chained feature edges (subsec. 3.2): stroke length and path variations. They are described next.

Stroke length variations. The use of directed graphs results in a larger number of chains with shorter line segments, compared to the use of undirected graphs (fig. 11). Both approaches produce good results and the final choice is up to the user. Directed graphs yield many polylines (or chains) that are later shortened or eliminated due to the curve fitting

(a) G_d (b) G_u

	$f1$	$f2$	$f3_{m1}$	$f4_{m1}$	$f3_{m2}$	$f4_{m2}$
$G_d : n_c$	142	178	350	250	340	287
$G_d : l_c$	163	141	438	264	386	314
$G_u : n_c$	27	167	280	186	244	206
$G_u : l_c$	273	142	451	275	395	322

Figure 11: *Example showing the effects of stroke length variations resulting from the chaining of feature edges. In the table, n_c is the total number of chains and l_c is the mean length of chains (number of vertices), that results from chaining silhouettes, creases, caps and pits ($f1$, $f2$, $f3$, $f4$ respectively), using directed (G_d) and undirected (G_u) graphs (subsec. 3.2). For caps and pits, $m1$ and $m2$ refer to the methods of Turk [27] and Nomura and Hamada [28], respectively.*

and curve order control (fig. 11, left). This is the reason why directed graphs are more appropriate to reproduce a *loose* drawing style, with larger distributions of gaps and connectives relations along the lines. Undirected graphs result in a "tighter" drawing where lines seem to be more specifically placed with fewer gaps along them, especially at the silhouettes (fig. 11, right).

Stroke path variations. We use a randomized approach for the chain's source and path selection. This results in a large variation of path configurations, but does not have a significant impact on the overall quality of the final renderings. For example, consider the images shown in Figure 12, which were obtained using the same dihedral threshold angle for extracting creases. Although the extracted paths are different (especially in the hair), all three images make a similar overall impression.

Figure 13 shows how our systems can be used for botanical illustration. Notice how a few strokes with proper weight suggest complex arrangements of plant elements, including different petal shapes and levels of branching. This figure also shows how different perceptions of shape can be achieved by rendering feature edges individually and in combination.

The final set of images (fig. 14) illustrates the operation of our system for four different models. In the **galleon**, observe how strokes suggest the overall shape of the hull. Crease and cap strokes suggest folds on the sail. In the **cow**, enclosures suggest key features of the head, the concave junction of

Figure 12: *Examples of rendering variations due to different random choices in the chaining of feature edges.*

front leg with body, the roundness of the belly, and concave regions at the back. In the **castle**, notice the cluster of lines suggesting the arches, the single lines for the walls, and a few lines suggesting the pointed roof of the tower. Finally, in the **car**, notice the suggestion of the shape of the wheels, the hood, and the foot stand by the door.

5. Conclusions and future work

This paper presents a 3D NPR method that automatically reproduces traditional pure line drawings. The simple combination of chaining, weighting, and smoothing applied to outline and interior feature edges of an input mesh reproduces the effects of ink fluidity, line weighting, connectives and enclosure, and leads to results adding three-dimensional shape suggestion using selected feature lines. This conforms to the main characteristics of *loose* line drawing style using traditional *linear phrasing* techniques. The illustrative power of the resulting images depends on the interplay between the depicted lines and the imagination of the observer.

Future improvements include extending our system with other algorithms for shape feature analysis (including alternative curvature estimation methods). It would also be useful to have automatic selection of threshold values assigned in different regions of the model to allow, for instance, better control of the level of detail (in terms of line quantities and styles) in selected regions of the model. Another interesting path to explore is the reproduction of additional connectives/enclosure effects following the patterns found in traditional drawings used in artistic and scientific illustrations. The criteria we use to judge the quality of the non-photorealistic rendering is solely based on observations of how close to the original drawings are to the computer-generated ones. It is important to conduct more formal evaluations and user studies to provide quality pure line drawing tools for illustrators in art and science.

Figure 13: *Results for botanical illustrations:* ***fireweed*** *model shown in full view (left) and four drawing steps (top) showing (from left to right) silhouettes, boundaries, silhouettes + boundaries, and silhouettes + creases + caps. Bottom: sunflower and hyacinth.*

Acknowledgements

We would like to thank Martin Fuhrer for his help generating plant models, and Brian Wyvill, Patricia Rebolo Medici, Desmond Rochfort, Alan Dunning, and Peter MacMurchy for the useful discussions and advice. We also thank the anonymous reviewers for their careful and valuable comments and suggestions. This research was supported in part by Discovery Grants from the Natural Sciences and Engineering Research Council of Canada.

References

1. C. Christou, J. J. Koenderink, and A. J. V. Doorn, "Surface gradients, contours and the perception of surface attitude in images of complex scenes", *Perception*, pp. 701–713 (1996).
2. W. Crane, *Line Form*. George Bell Sons, London, (1900).
3. P. Rawson, *Drawing*. University of Pennsylvania Press, (1987).
4. G. Simmons, *The Technical Pen*. Watson-Guptill Publications, (1993).
5. E. Hodges, *The Guild Handbook of Scientific Illustration*. Van Nostrand Reinhold, (1989).
6. K. Paterson, *Cambridge Secrets 2*. (Warry-Haired Reader guides), Thanet Press Limited, Margate, (2000).
7. H. Carson, *Illustrators' reference file : compositional elements for artists, renderers, and students*. Wiley, John and Sons, Inc., (1992).
8. K. West, *How to draw plants: the techniques of botanical illustration*. The Herbert Press Limited, London, (1983).
9. G. Winkenbach and D. H. Salesin, "Computer-generated pen-and-ink illustration", in *Proc. of SIGGRAPH 94*, pp. 91–100, (1994).
10. T. Strothotte, B. Preim, A. Raab, J. Schumann, and D. R. Forsey, "How to render frames and influence people", in *Proc. of EuroGraphics 94*, pp. 455–466, (1994).
11. L. Markosian, M. Kowalski, S. Trychin, L. Bourdev, D. Goldstein, and J. Hughes, "Real-time nonphotorealistic rendering", in *Proc. of SIGGRAPH 97*, pp. 415–420, (1997).
12. J. W. Buchanan and M. C. Sousa, "The edge buffer: a data

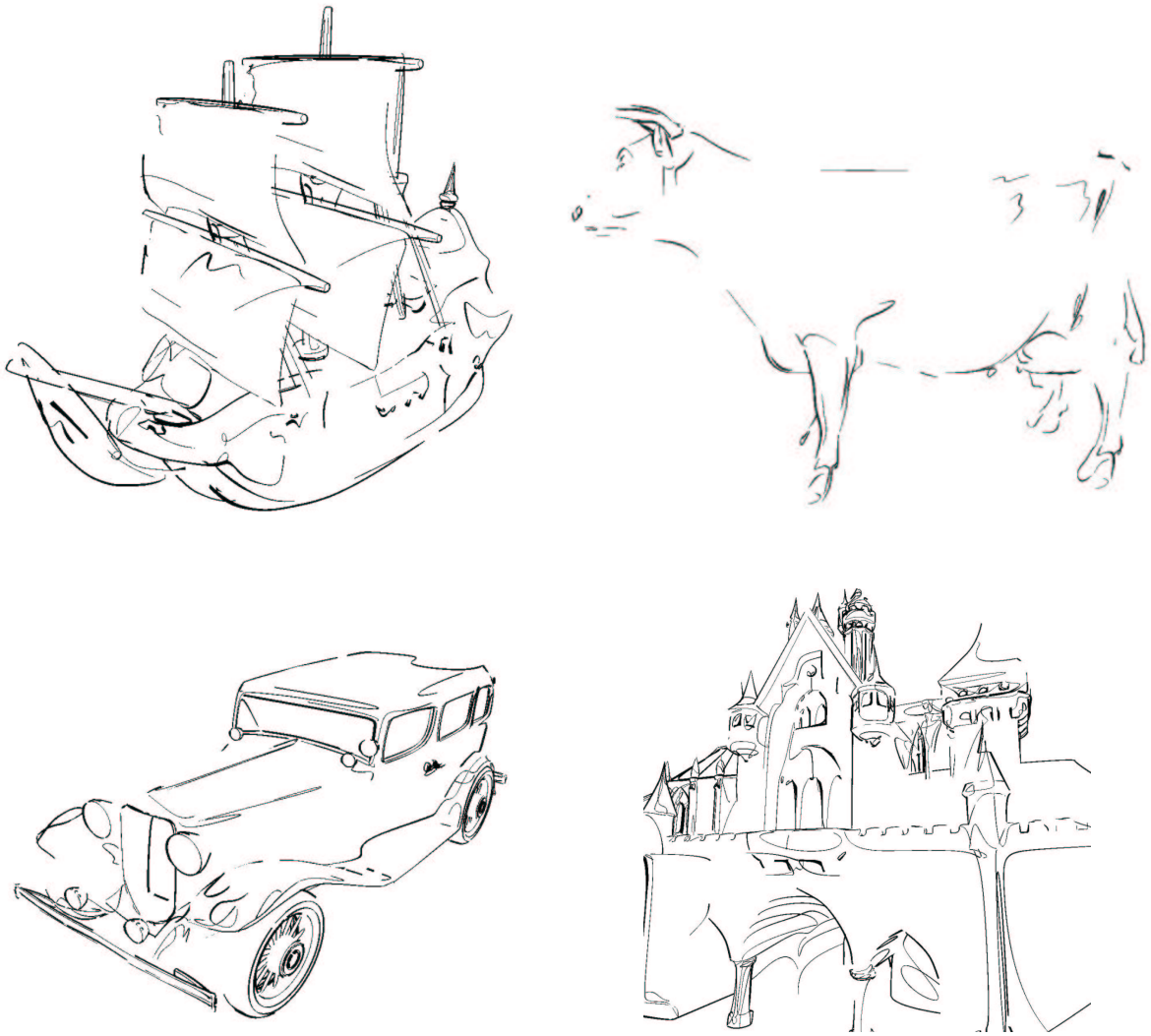


Figure 14: *Examples of drawings generated using our system: a* ***galleon****, a* ***cow****, a* ***car****, and a* ***castle****.*

structure for easy silhouette rendering", in *Proc. of NPAR 00*, pp. 39–42, (2000).

13. A. Hertzmann and D. Zorin, "Illustrating smooth surfaces", in *Proc. of SIGGRAPH 00*, pp. 517–526, (2000).

14. P. Sander, X. Gu, S. Gortler, H. Hoppe, and J. Snyder., "Silhouette clipping", in *Proc. of SIGGRAPH 00*, pp. 327–334, (2000).

15. B. Gooch, P.-P. J. Sloan, A. A. Gooch, P. Shirley, and R. Riesenfeld, "Interactive technical illustration", in *Proc. of the 1999 Symposium on Interactive 3D Graphics*, pp. 31–38, (1999).

16. J. Northrup and L. Markosian, "Artistic silhouettes: a hybrid approach", in *Proc. of NPAR 00*, pp. 31–38, (2000).

17. T. Sasada, "Drawing natural scenery by computer graphics", *Computer Aided Design*, **19**(4), pp. 212–218 (1987).

18. R. Kalnins, L. Markosian, B. Meier, M. Kowalski, J. Lee, P. Davidson, M. Webb, J. Hughes, and A. Finkelstein, "WYSIWYG NPR: Drawing strokes directly on 3d models", in *Proc. of SIGGRAPH 02*, pp. 755–762, (2002).

19. A. Girshick, V. Interrante, S. Haker, and T. Lemoine, "Line direction matters: an argument for the use of principal directions in 3d line drawings", in *Proc. of NPAR 00*, pp. 43–52, (2000).

20. C. Rossl, L. Kobbelt, and H. Seidel, "Line art rendering of triangulated surfaces using discrete lines of curvature", in *Proc. of WSCG 00*, pp. 168–175, (2000).

21. M. C. Sousa and J. W. Buchanan, "Computer-generated graphite pencil rendering of 3d polygonal models", in *Proc. of Eurographics 99*, pp. 195–208, (1999).

22. M. Kaplan, B. Gooch, and E. Cohen, "Interactive artistic rendering", in *Proc. of NPAR 00*, pp. 67–74, (2000).

23. L. Markosian, B. Meier, M. Kowalski, J. N. L.S. Holden, and J. Hughes, "Art-based rendering with continuous levels of detail", in *Proc. of NPAR 00*, pp. 59–66, (2000).

24. T. Isenberg, N. Halper, and T. Strothotte, "Stylizing silhouettes at interactive rates: From silhouette edges to silhouette strokes", in *Proc. of EuroGraphics 02*, pp. 249–258, (2002).

25. A. Majumder and M. Gopi, "Hardware accelerated real time charcoal rendering", in *Proc. of NPAR 02*, pp. 59–66, (2002).

26. M. Mantyla, *An Introduction to Solid Modeling*. Computer Science Press, (1988).

27. G. Turk, "Re-tiling polygonal surfaces", in *Proc. of SIGGRAPH 92*, pp. 55–64, (1992).

28. M. Nomura and N. Hamada, "Feature edge extraction from 3d triangular meshes using a thinning algorithm", in *Proc. of SPIE Conference on Vision Geometry X*, pp. (4476):34–41, (2001).

29. M. Meyer, M. Desbrun, P. Schroder, and A. Barr, "Discrete differential geometry operators for triangulated 2-manifolds", in *Proc. of the International Workshop on Visualization and Mathematics 2002 (Vismath 2002)*, (May 2002).

30. W. Correa, R. Jensen, C. Thayer, and A. Finkelstein, "Texture mapping for cel animation", in *Proc. of SIGGRAPH 98*, pp. 435–446, (1998).

31. R. Sedgewick, *Algorithms in C, Part 5: Graph Algorithms , third edition*. Addison-Wesley, (2001).

32. S. Strassmann, "Hairy brushes", in *Proc. of SIGGRAPH 86*, pp. 225–232, (1986).

33. B. Pham, "Expressive brush strokes", *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, **53**(1), pp. 1–6 (1991).