Rendering Nonphotorealistic Strokes with Temporal and Arc-Length Coherence

Lubomir Bourdev

Department of Computer Science

Brown University

Submitted in partial fulfillment of the requirements for the Degree of Master of Science in the Department of Computer Science at Brown University

Signature (Prof. John Hughes, Project Advisor)

5/2/98
Date

Rendering Nonphotorealistic Strokes with Temporal and Arc-Length Coherence

Lubomir Bourdev

1.0 Objective

This system allows for rendering a silhouette of an object in a frame-to-frame coherent way. The input to the system each frame is a set of silhouette pixels in a rendering of the object and their corresponding silhouette edges in a polygonal model (mesh) of the object. The output is a set of silhouette strokes.

2.0 Introduction

Nonphotorealistic Rendering (NPR) deals with representing pictures and animation in, as the name suggests, a nonphotorealistic fashion. While there is only one way to render a photo-realistic image, in an NPR system we have the freedom to represent it in an unlimited number of ways. By varying the style of rendering, the image composition and the level of detail, by omitting or emphasizing certain parts of the drawing, we can direct the viewer's attention and convey a bias -- something that is not possible in photorealistic rendering systems.

Many nonphotorealistic rendering styles benefit from "*economy of line*" (Markosian, 1997) -- the picture is drawn with as few strokes as possible, omitting parts where detail is not needed and drawing only the "important" strokes -- usually along the silhouette of the object. In this paper we describe an algorithm for drawing silhouette strokes. The strokes can be rendered as a simple polyline, or displaced from it either by a predefined function (e.g. sine waves) or a pattern read from a file.

Our NPR system maintains a balance between **temporal coherence** (disallowing distracting trembling of the strokes over frames) and **arc-length coherence** (maintaining a constant period of repetition of the pattern in the stroke). It is not possible to achieve both temporal and arc-length coherence simultaneously. For example, when an object approaches the camera, its strokes become longer and we need to either stretch the patterns and thus violate arc-length coherence, or insert new patterns and violate temporal coherence. The right behavior depends on the style of rendering. For instance, arc-length coherence is very important when drawing text strokes, since the text may become unreadable if stretched. On the other hand simple styles, like a sine wave, look better when temporal coherence is preserved.

2.1 A Straw Man Approach

One naive, straightforward approach to the problem of rendering the silhouette of a mesh is as follows:

• render all triangles of the mesh in the frame buffer

• check all edges and determine the set of silhouette edges

• connect the adjacent silhouette edges into 3D silhouette strokes

• render the silhouette strokes in the desired style. The frame buffer handles the occlusion problem.

There are several problems with this straightforward approach. First, using the frame buffer to handle occlusion after a style is applied to the stroke often causes occlusion problems. In many styles the stroke may go behind the surface from the camera point of view in which cases it is not drawn (fig. 1). The shape of the stroke is defined on the film plane and there is not a right way to define it in world space

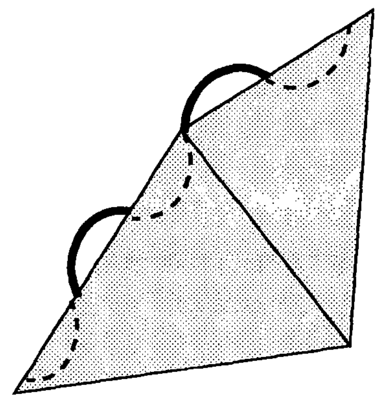


Fig. 1 Using the frame buffer to draw stroke in 3D causes occlusion problems

Another problem is that, although it seems reasonable to assume that adjacent silhouette edges form long connected polylines, this is not so in practice, especially in the nearly planar regions, where some edges are slightly convex and others are slightly concave. The predominant silhouette in such cases consists of semi-occluded edges (fig. 2). Thus the simple idea of connecting adjacent silhouette edges into strokes does not give the desired effect. Moreover if a continuous style is applied along such silhouette edges it would be broken into small discontinuous pieces.

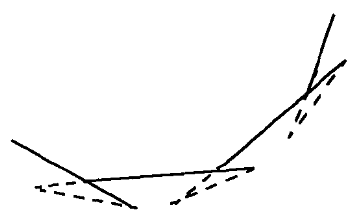


Fig. 2. The predominant silhouette consists of a sequence of semi-occluded convex and occluded concave edges

A third issue involves maintaining temporal coherence. As the object changes position and orientation with respect to the camera, the perceptual difference in a frame-to-frame rendering of the silhouette should be minimized in most styles. In other words, the phase of the silhouette stroke in any region on the object should be maintained as much as possible across frames. The straightforward approach does not provide for any frame-to-frame (temporal) coherence -- the strokes constantly change their length and no phase information is preserved from the previous frame.

A fourth problem is that, just as in hand drawn illustrations, most stroke styles need to be defined in screen space, and not in world space as the straightforward approach does. In other words, the period of the pattern needs to stay constant as the object approaches or moves away from the camera.

The system presented in this paper addresses these four problems.

3.0 Algorithm Overview

Intuitively, to maintain temporal coherence we need to preserve the phase information (i.e. the locations of the "bumps" of the stroke style) across frames. What the viewer perceives as the same stroke in two consecutive frames, however, are in fact two different strokes and in general there is no easy way to associate them -- they don't necessarily span the same set of edges; sometimes a stroke is broken into several new strokes, or several strokes merge into one. The stroke or a part of it may disappear or a new one may appear.

Because of this difficulty, we represent the stroke as a list of smaller units (which we call silhouette particles), each of which maintains local phase information and tries to "survive" and transfer this phase information across frames. Each silhouette particle is associated with one silhouette edge at a time and when its edge becomes non-silhouette, it tries to find another similar silhouette edge that is close to it on the film plane (see Section 6.2). Each frame the particles are partitioned into lists (or rings) of neighboring particles and each of those lists is used to construct a stroke (see Section 6.4). The phase information of a stroke is determined from the local phase information of its particles, their position along the stroke, and the stroke style. Once the phase at each point along the stroke is determined, it is used to update back the phase information for its particles (see Section 6.5).

The algorithm is described in more detailed below.

4.0 Definition of Terms

Silhouette stroke. A 2D-polyline along the projection of the silhouette. The stroke can be non-periodic or periodic. A **non-periodic (homogeneous) stroke** is drawn as a uniform medium, for example a straight line. A **periodic stroke** is drawn by repeating a given **pattern**. The length of the pattern is called a **stroke period**. Any position A along a periodic stroke corresponds to a displacement $t \in [0, Period)$ along its repeated pattern. We say that the **phase** of the stroke at A is equal to t. Strokes can also be non-stretching or stretching. A **non-stretching stroke** always preserves arc-length coherence -- the period of repetition of its pattern is constant along the stroke. A **stretching stroke** allows for variations in its period in order to achieve temporal coherence. This variation may be "smoothed" over time to achieve a balance with arc-length coherence. The speed of smoothing, which intuitively is the weight of temporal vs. arc-length coherence, is called **elasticity**; $elasticity \in [0, 1)$

Silhouette edge. An edge of the mesh that lies on the silhouette from the current point of view. In other words, one of its adjacent faces is front-facing and the other is back-facing. Each edge has a:

- *2D direction* -- a direction along the screen projection of the edge. We gather the screen projections of all silhouette edges and orient them counter-clockwise when observed from the camera point of view.

Let p_1 and p_2 are particles and e_3 and e_2 are their corresponding edges. To consider p_1 as a prospective next (respectively previous) neighbor of p_2 the following conditions are examined:

1. p_2 does not have a next neighbor

2. p_1 does not have a previous neighbor

3. p_1 and p_2 are adjacent

4. e_1 's particle last frame was a next neighbor of e_2 's particle (in the rare case when an edge has more than one particle, this heuristics picks one of them and might fail)

5. e_1 and e_2 are adjacent

6. e_1 and e_2 have similar directions

7. p_1 's beginning pixel is adjacent to p_2 's end pixel.

Conditions 1, 2, and 3 are required -- if any of them fail, then p_1 cannot be a next neighbor of p_2. We use conditions 4 and 5 in the fast decision pass -- if both of them are true, then p_1 becomes the next neighbor of p_2. We perform the second pass for those pairs of particles for which the first pass is unsuccessful. In the second pass we evaluate conditions 4 - 7 and assign a weight to each of them. p_1 becomes the next neighbor of p_2 only if the sum of the four weights is above a certain threshold. The weights and the threshold are manually adjusted for optimal performance.

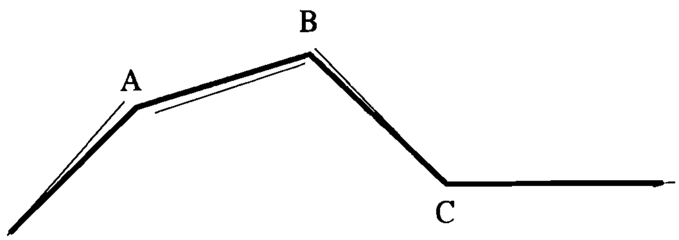


Fig. 6. Regions of silhouette edges covered by their particles (in black) and the silhouette stroke resulting from them (in red)

Although edges e_1 and e_2 are adjacent on figure 5, they are unlikely to become neighbors because of the sharp angle they have (violating condition 6). Of course, e_1 and e_3 are not even considered as prospective neighbors since they are not adjacent (violating mandatory condition 3)

We construct a new silhouette stroke for each doubly-linked list (or ring) of neighboring particles. We compute the 2D polyline of the stroke from the screen projections of the end points of the edge segments corresponding to each particle. If a point on the polyline is not peripheral then there are two edge segment end points corresponding to it and their locations are averaged (points A, B and C on figure 6). As indicated on the figure, although the screen projections of two adjacent edges may share the same endpoint, this is not always true in practice for the regions covered by their particles (the black segments on figure 6). The reasons for this are round-off errors caused by the rasterization (as is the case with the end of edge e_2 in figure 5) as well as the particular rasterization algorithm of the hardware, as explained in the Implementation Details section. In either case those differences are smaller than a pixel size and therefore have no visual effect.

6.5 Computing the phase values along a stroke

After we connect the neighboring particles in strokes, for each stroke we need to re-evaluate the phase values of its particles given their current phase values, their positions along the stroke and the stroke style. This step is skipped for homogeneous strokes and for strokes whose styles don't require temporal coherence.

Let n denote the number of particles in a stroke, d_i denote the arc-length distance along the stroke from its beginning to the beginning of particle i (and thus $d_0 = 0$), h_i denote the phase value of particle i at the beginning of this step[1], and h_i' denote the new phase value of particle i, which is to be computed in this step. Let $d_{i,j} = d_j - d_i$

6.5.1 Non-stretching strokes

Because the period anywhere along a non-stretching stroke is constant, once we compute h_0', we implicitly define the phase value anywhere along the stroke. To compute the initial phase of the stroke, h_0', we average the phases of its particles evaluated at the beginning of the stroke (we use only those particles whose phases are defined).

$$h_0' = avg(norm(h_0 - d_0), 1, \ldots, norm(h_n - d_n), 1)$$

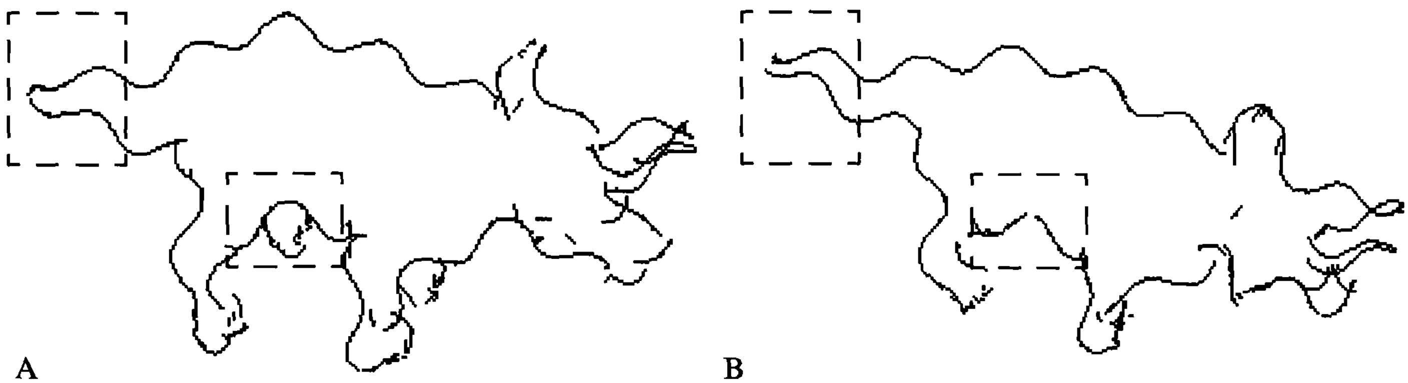
If none of the particles has a defined phase then the stroke is "new" in the image and we can pick any phase as its initial one. After computing the initial phase, we can infer the rest of the phases:

$$h_i' = norm(h_0' + d_i)$$

6.5.2 Stretching strokes

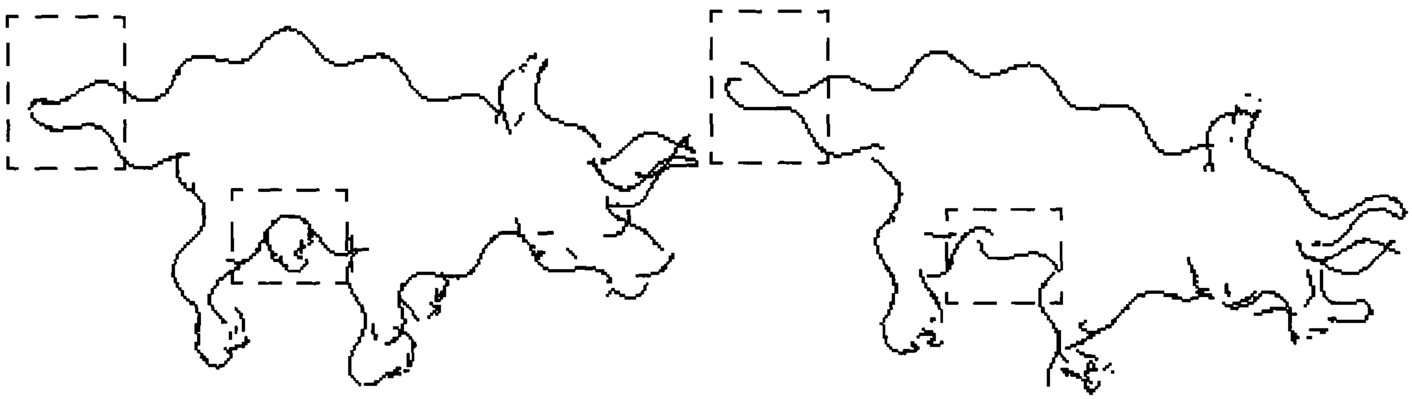
While for non-stretching strokes we preserve arc-length coherence in full, for stretching ones we need to balance it with temporal coherence. Thus, for each particle we need to compute a

1. For clarification, if the edge of particle i was a silhouette edge in the previous frame, then h_i is carried over from the previous frame. Otherwise it is obtained as described in Section 6.2 and may be undefined for some or all particles as is the case for particle p_2 in fig. 5



A B

Fig. 8 Rotation of a triceratops with **enabled** temporal coherence. Original image (A) and after rotation around axis parallel to camera X axis (B)



A B

Fig. 9 Rotation of a triceratops with **disabled** temporal coherence. Original image (A) and after rotation around axis parallel to camera X axis (B)

8.2 Future work

The only way in the current implementation that we can achieve inexactness of the style, typical of hand drawn illustration, is by creating a sequence of "bumps" by hand and treating them as one large pattern. The problem with that approach is that, the larger the pattern is, the harder it is to satisfy temporal coherence. A possible future research project is finding a more elegant way of solving that problem, for example by maintaining a set of patterns and selecting from them at random. Other future projects are implementing a collection of stroke styles and building a user interface for creating hand-drawn patterns.

9.0 Acknowledgments

I would like to thank Lee Markosian and my advisor Prof. Hughes for the bright suggestions they gave me and the many hours they spent on this project. Many thanks to my colleagues Michael Kowalski, Caroline Dahllof, Dave Bremer and Loring Holden.

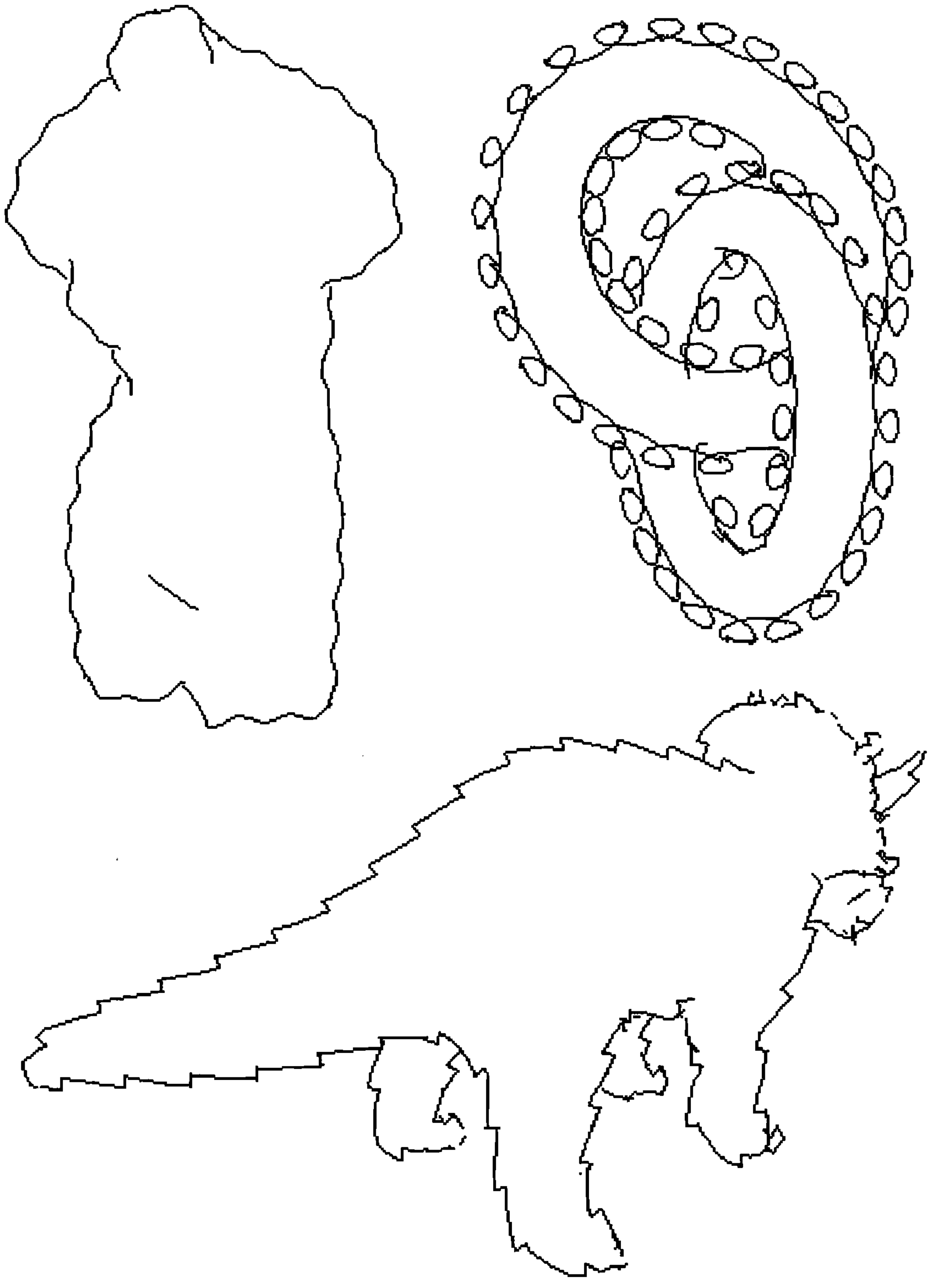


Fig. 10 Snapshots from our NPR system

10.0 References

- L. Markosian, M. Kowalski, S. J. Trychin, L. Bourdev, D. Goldstein, J. F. Hughes Real-Time Non-photorealistic Rendering. In *Proceedings of SIGGRAPH '97*,.pp. 415-420, 1997
- R. Zeleznik, K. Herndon, and J.F. Hughes. Sketch: An interface for sketching 3d meshes. In *Proceedings of SIGGRAPH '96*,.pp. 163-170, 1996
- J.Foley, A. van Dam, S. Feiner, and J.F.Hughes. *Computer Graphics: Principles and Practice,* Addison-Wesley, 1992