Spatio-Temporal Analysis for Parameterizing Animated Lines

Bert Buchholz[1] Noura Faraj[1] Sylvain Paris[2] Elmar Eisemann[1] Tamy Boubekeur[1]
[1]Telecom ParisTech - CNRS [2]Adobe

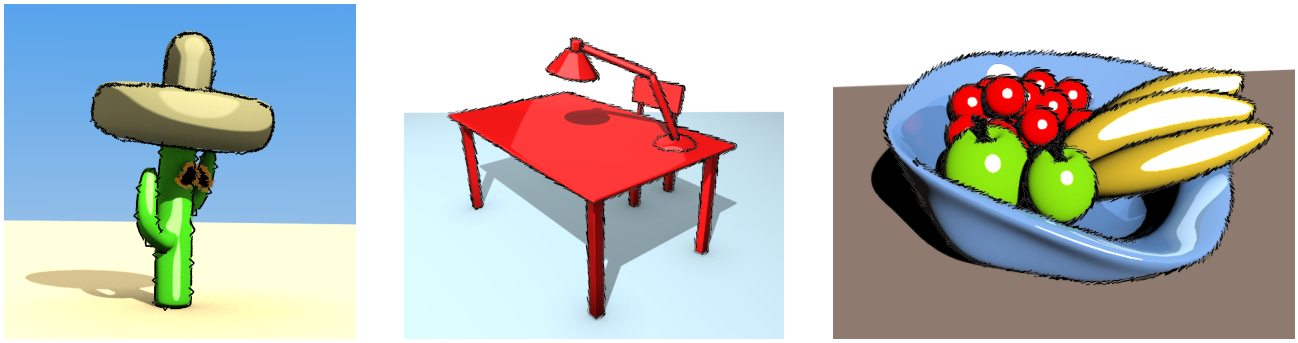


Figure 1: *Our algorithm builds a temporally consistent parameterization for lines extracted from an animated 3D scene.*

Abstract

We describe a method to parameterize lines generated from animated 3D models in the context of animated line drawings. Cartoons and mechanical illustrations are popular subjects of non-photorealistic drawings and are often generated from 3D models. Adding texture to the lines, for instance to depict brush strokes or dashed lines, enables greater expressiveness, e.g. to distinguish between visible and hidden lines. However, dynamic visibility events and the evolving shape of the lines raise issues that have been only partially explored so far. In this paper, we assume that the entire 3D animation is known ahead of time, as is typically the case for feature animations and off-line rendering. At the core of our method is a geometric formulation of the problem as a parameterization of the space-time surface swept by a 2D line during the animation. First, we build this surface by extracting lines in each frame. We demonstrate our approach with silhouette lines. Then, we locate visibility events that would create discontinuities and propagate them through time. They decompose the surface into charts with a disc topology. We parameterize each chart via a least-squares approach that reflects the specific requirements of line drawing. This step results in a texture atlas of the space-time surface which defines the parameterization for each line. We show that by adjusting a few weights in the least-squares energy, the artist can obtain an artifact-free animated motion in a variety of typical non-photorealistic styles such as painterly strokes and technical line drawing.

Keywords: animated line drawing, temporal coherence

1 Introduction

Line drawing is a popular rendering style commonly used for mechanical illustrations, cartoons, and sketches and, in many cases, these are derived from three-dimensional models. For instance, many 3D CAD softwares applications offer line drawing as one of the stylized rendering options. In line drawing, shapes are represented by a few carefully selected lines such as silhouettes [Hertzmann and Zorin 2000], suggestive contours [DeCarlo et al. 2003], or apparent ridges and valleys [Judd et al. 2007]. The simplest approach is to render lines as continuous and textureless, as if one used a ball-point pen with even pressure. But often, the lines are textured to offer greater expressiveness – for instance simulating brush strokes, dashed lines, or calligraphic curves. In this paper, we are interested in parameterizing the lines such that such textures can be applied to them.

A typical work session starts by creating a 3D animation. While many research studies still work on improving this step, we use existing tools such as Maya and Blender and assume that a 3D animation is available. The next step is to select the lines to be drawn. In this work, we focus on the case where these lines are silhouettes. We describe a procedure to track silhouettes across frames in Section 2. Without this grouping, each line would be parameterized independently of the others, which produces "sliding" artifacts, e.g. brush strokes look "disconnected from the underlying 3D model". Once the lines are grouped through time, we compute a parameterization over these lines. This is the main focus of our paper, it is described in Section 3. Our goal is to define how the texture is mapped onto a line. This parameterization is directly responsible for the stretch and compression of the texture and for its motion. As such, it plays a critical role in the look of the rendered animation. Once we have parameterized the lines, the last step is to render the animation, for which we use standard techniques.

The difficulty of parameterizing animated lines comes from the fact that they are extracted from a three-dimensional model whereas the final drawing lives in the 2D image plane. We seek a temporally consistent parameterization such that the drawing looks like it follows the actual motion of the scene. However, there is no general solution to this problem. For instance, if one constrains the 2D lines to exactly follow the 3D model, effects like foreshortening may lead to unsightly texture distortion and brush stroke may become overly compressed or stretched. On the other hand, avoiding distortions can lead to significant motion inconsistencies. In our approach, we formulate temporal coherence as a least-squares optimization problem where each constraint is weighted by parameters that let users control the look of the final output. Another major difficulty comes from lines that merge and split, which introduces topological discontinuities in the drawing. Ignoring these events yields unpleasing "popping" artifacts, e.g. a brush strokes suddenly becomes two strokes. Kalnins et al. [2003] have shown that such discontinuities can be handled by extending them in time, e.g. by always drawing

two strokes even if, in some frames, it is a single connected line. Because Kalnins's approach works in real time, this strategy is only partially effective. It prevents popping due to disocclusion but not artifacts due to occlusions. Further, it also tends to over-segment lines, creating a large number of short strokes. We will discuss this difficulty in more detail (Section 4). Unlike Kalnins and colleagues, we assume that the whole animation is known from the start. Although this choice makes our method inapplicable in a game context, it makes it more suitable for feature animations where quality is paramount. In this respect, our approach is complementary to previous work.

Our approach works in the space-time domain that is the product of the (x, y) image space with the time axis t. For each line to be drawn, we consider the space-time set of 1D lines spanned by the line throughout the animation, which we name $\ell(t)$. It can be seen as a 2D surface in the (x, y, t) domain and we cast the problem of parameterizing $\ell(t)$ as the parameterization of this 2D space-time surface. This is the central idea of our paper. This construction allows us to account for the 3D motion of the input model and the distortion of the 2D line texture, and we can ensure the complete coverage of the line. Further, we handle splitting and merging events with a graph embedded in the line space-time surface. Our approach enables the reuse of the line discontinuities for several events, thereby drastically reducing the number of actual strokes needed in a given animation.

1.1 Related Work

A wealth of articles deal with rendering line drawings from 3D models. Several methods have been proposed to decide where to draw lines on a given 3D model, e.g. [Hertzmann and Zorin 2000; DeCarlo et al. 2003; Judd et al. 2007; Cole et al. 2008; Grabli et al. 2010]. Whereas most of these studies focus on static scenes, DeCarlo et al. [2004] describe a technique specific to animation. In this work, we focus on line parameterization and illustrate our approach on silhouette lines. We will also discuss how it can be extended to other lines.

A few methods specifically target the rendering of animations as line drawings. However, most of them produce uniform lines with no texture [Lee et al. 2007; Winnemöller et al. 2006]. The most related work to ours is by Kalnins et al. [2002; 2003] who also focus on temporally consistent textured lines. We share their goal of adding texture onto lines which greatly improves the expressiveness of the drawing. For instance, texture can be used to represent dotted lines, brush strokes, or even surface details such as cactus needles. However, Kalnins et al. demonstrate that a naive parameterization yields unacceptable results with texture sliding on the object surface. They address this problem in the context of real-time interactive simulation, where only the next frame is known at a point. Although they demonstrate convincing results in a number of cases, this partial knowledge of the animation limits their ability to adapt to topological events. For instance, they cannot deal with appearing discontinuities because these events are unknown until they occur [Kalnins 2004, § 4.5]. This kind of "unexpected event" is inherent to any real-time scenario. In this paper, we explore a different, complementary case in which the entire animation is known in advance. In this context, our approach can "anticipate" appearing discontinuities and adapt to them. In particular, our parameterization does not produce popping artifacts when an occlusion occurs and requires fewer cuts along lines, thereby rendering longer and more visually pleasing strokes.

Depending on the parameterization, the line texture may be compressed or stretched producing unpleasant renderings. Bénard et al. [2010] address this issue by generating a multi-scale texture. The proposed parameterization focuses on the same real-time scenario as Kalnins et al. [2003] and shares the same limitations inherent in this setup.

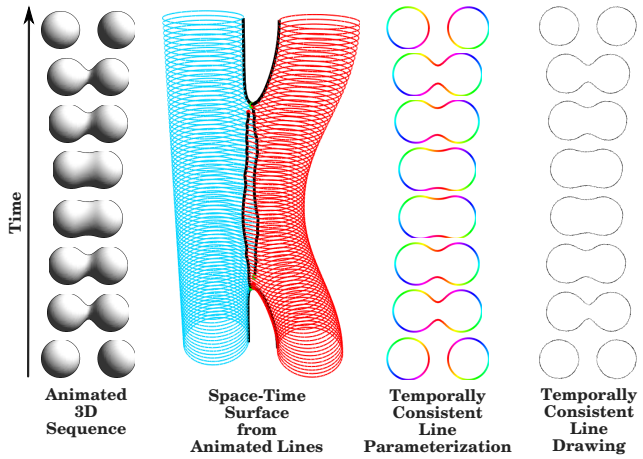


Figure 2: Principle: *a temporally consistent parameterization of a line drawing is computed by parameterizing the space time surface it defines over time and used for texturing.*

Depending on the view distance, lines may become too close or even start overlapping, altering shape perception. Shesh and Chen [2008] solve this problem by defining a line hierarchy and replacing multiple overlapping strokes by a single "average" line.

Temporal consistency has also been studied for the texture that represents the interior of the models and the canvas on which the illustration is drawn, e.g. [Cunzi et al. 2003; Kim et al. 2008; Bénard et al. 2009]. However, the 1D nature of lines raises specific challenges, as the topology of 1D curves significantly differs from the topology of a 2D canvas.

1.2 Contributions

In this paper, we introduce the following contributions.

Space-time formulation We formulate the parameterization of a animated line as the parameterization of the space-time surface that it swept during the animation (see Fig. 2).

Least-squares optimization of geometric constraints We express our objectives as a series of geometric constraints. We represent each of them by a least-squares energy term and the trade-off between the constraints is controlled by a small set of meaningful parameters.

Discontinuity reuse We show that line discontinuities can be reused to limit the number of cuts made to the lines. We control the trade-off between cut reuse and temporal coherence with a simple parameter for the maximum allowed sliding.

1.3 Overview

Input Our model takes as input an animated 3D model $\mathcal{M}(t)$ where t is the time variable defined over an interval $\mathbb{T}$, and a camera that is represented by the function π that projects 3D points onto the image plane. We assume that $\mathcal{M}$ has a temporally consistent parameterization, that is, for any point $\mathbf{P}$ of the model, we can compute its trajectory $\mathbf{P}(t)$ during the animation. In practice, these conditions are always satisfied when the data come from a

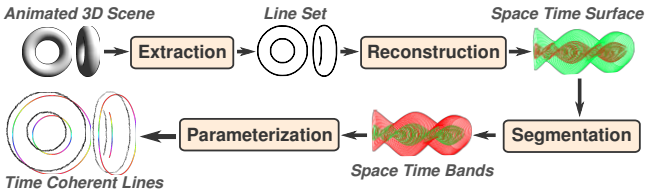


Figure 3: Overview: *lines are extracted independently for each frame of the animation, before being grouped as plausible corresponding lines from frame to frame. Lines may split or merge during the animation and create a graph which is subsequently decomposed in bands corresponding to single lines over time. These bands, or space-time lines, are finally parameterized independently under user control to provide time-coherent parametric lines.*

3D modeler in which a base mesh is deformed while its topology is preserved. The mesh vertices and faces are naturally linked through time, which implicitly ensures a temporally consistent parameterization of the 3D model. We also propose a heuristic to cope with simple models that do not have such parameterization, e.g. meta-balls. We also assume that the camera is given as input, that is, we have a projection function π that maps 3D points onto the image space.

Objective We seek to parameterize the 2D lines that correspond to the silhouettes of $\mathcal{M}(t)$. We process the lines one by one. We name $\mathcal{L}(t)$ a 3D line at the surface of $\mathcal{M}$ at time t. As t changes, $\mathcal{L}$ may move on $\mathcal{M}$. We name ℓ the 2D projection of $\mathcal{L}$, that is, $\ell = \pi(\mathcal{L})$. In this work, we seek a parameterization of ℓ that is temporally consistent. Formally, we aim to define a function $f(u,t)$ such that at any time $t \in \mathbb{T}$, f is a continuous one-to-one mapping between the parameter space, e.g. $u \in [0;1]$, and $\ell(t)$.

Strategy We formulate the parameterization of a line ℓ as the parameterization of the space-time surface $\mathcal{S}$ swept by ℓ during the animation, that is, $\mathcal{S} = \bigcup_t \ell(t)$. First, we construct $\mathcal{S}$ from the lines extracted at each frame. Then, we expose how we deal with temporal coherence when the topology of the lines does not change over time. We formulate desired properties such as temporal coherence, lack of distortion, and line coverage in geometric terms expressed on $\mathcal{S}$. Then, we translate this problem into a least-squares optimization that can be solved with a sparse linear system. In a second part, we cope with splitting and merging events of these lines. To avoid popping, we cut lines and propagate the resulting discontinuities. We show how to use the same cut for several discontinuities to avoid over-segmenting the lines. With our space-time formulation, the cuts are geodesic lines on $\mathcal{S}$ and our handling of discontinuities corresponds to decomposing $\mathcal{S}$ into charts with a disc topology. Finally, we present and discuss representative results of our approach.

2 Building the Space-Time Surface

In this section, we expose how to build the space-time surface $\mathcal{S}$ generated by a line ℓ during the animation. First we describe a technique that we use on simple examples. Then we expose a robust method that copes with complex topological changes and more realistic models in the case where the lines can be defined as the level set of scalar functions defined on the mesh, as is the case for silhouettes.

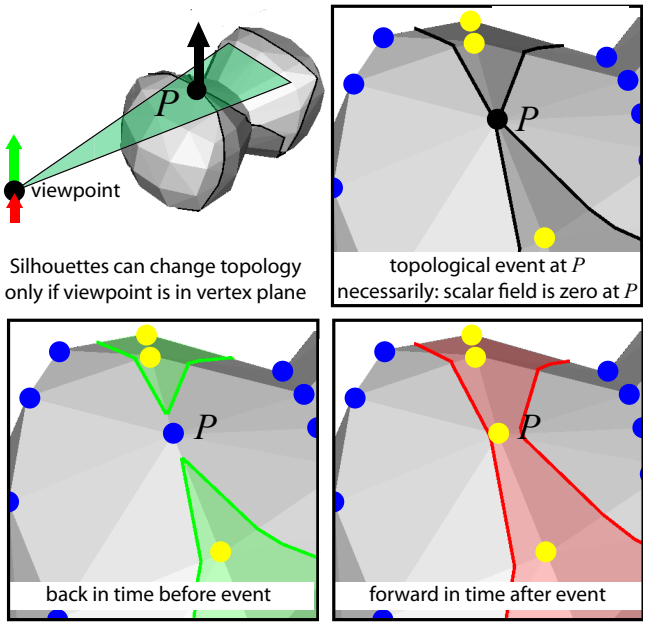


Figure 4: *Robust topology change detection for lines which can be modeled as the zero level sets of a scalar function (e.g. silhouettes).*

2.1 Simple Construction

For educational examples such as Figure 2, we follow a simple approach to build the space-time surface $\mathcal{S}$. We consider the lines extracted in two consecutive frames and use a voting scheme to decide which lines should be paired, i.e. which lines are actually adjacent on $\mathcal{S}$. Each vertex of each line casts a vote for the nearest vertex in the other frame according to the image space distance. For a given line, we observe the votes cast by its vertices and link it to the line in the other frame that received the most votes. This process creates a graph where lines extracted at each frame are the nodes and where the arcs indicate how to build the surface. Topology changes, i.e. when lines split or merge, are detected when a line is linked to more than one line in an adjacent frame.

The advantage of this approach is that it only assumes that we can extract lines at each frame. For instance, it can deal with an animated 3D model inconsistently meshed from frame to frame, which allows for processing meta-balls as shown in Figure 2. Moreover it can handle any kind of line drawing. However, it also relies on the fact that the image distance represents well the evolution of the lines on the 3D model which may not always be true for more complex models. We describe a robust approach that handles these cases in the following section.

2.2 Robust Construction

We assume that the input model $\mathcal{M}$ is a triangular mesh. If it is not, we convert it before processing it. The silhouette of $\mathcal{M}$ is made of one or several closed loops. First, we characterize when topological changes occur, i.e. when loops split or merge. Silhouettes are characterized by a zero dot product between the mesh normal $\mathbf{n}$ and the view direction $\mathbf{V}$. We estimate a normal $\mathbf{n}$ at each vertex and linearly interpolate it over the faces. With this scheme, at most one silhouette line can cross a triangular face: if $\mathbf{n} \cdot \mathbf{V}$ has the same sign for all three vertices, there is no silhouette, and if the sign changes, one silhouette line crosses the two edges with different signs. Since a topological event corresponds to two or more lines being in con-

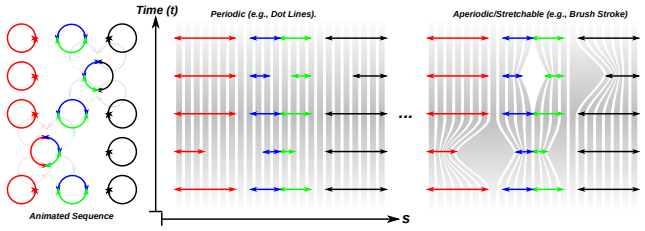


Figure 5: *Space-time parameterization of a line set: our variational formulation offers intuitive control w.r.t. to the texture function type (stretchable or periodic).*

tact, this cannot happen inside a face and must occur at a vertex. Thus, we only need to examine vertices and time instants where $\mathbf{n} \cdot \mathbf{V} = 0$. Assuming that the camera and mesh move linearly between frames, this amounts to finding the zero-crossing of a linear function. For each possible candidate, we check whether there is more than one line going through the vertex. If that is the case, we mark the vertex and time instant as a topological event (Fig. 4).

Once we have listed all the events, we build the space-time surface $\mathcal{S}$ by considering what happens between two frames at t and $t+\Delta t$. There are two cases. If there is no topological event, the silhouette loops have moved over the mesh without splitting or merging. In this case, we label the mesh with the sign of $\mathbf{n} \cdot \mathbf{V}$ at t and its sign at $t + \Delta t$. Mesh regions swept by the silhouette between the two frames have opposite signs, and since there is no topological events and the camera moves along a segment, these regions are disconnected. In particular, the linear camera movement ensures that a vertex can change its sign at most once. Using these properties, we build $\mathcal{S}$ by pairing lines at t and $t+\Delta t$ that are linked by a mesh region with opposite $\mathbf{n} \cdot \mathbf{V}$ signs. In the other case when there are one or more events between the two frames, we split the time interval so that a single event happens at time t_0 in each interval. We extract the loops at $t_0 - \epsilon$ just before the event, and at $t_0 + \epsilon$ just after it (Fig. 4). The $t_0 - \epsilon$ lines can be linked to the lines in the earlier frames using the no-event case. The same applies to the $t_0 + \epsilon$ lines and the later frames. We also link the $t_0 - \epsilon$ lines to the $t_0 + \epsilon$ lines to reflect the change of topology. If we split the interval between two frames to isolate events, we concatenate the information of all sub-intervals and only represent the links between the lines at t and the lines at $t + \Delta t$. Figures 8, 10, 11 and 9 show examples of our space-time surface reconstructed by our approach.

Discussion Our robust reconstruction relies on the fact that lines are a zero level set of a scalar function defined on the mesh $\mathcal{M}$. In the case of silhouettes, the function is $\mathbf{n} \cdot \mathbf{V}$. While not all lines can be expressed as a zero level set, several others lines fall in this category [Stroila et al. 2008]. This paper focuses on silhouette parameterization, and we believe that studying other types of lines is a natural extension for our work in the future. In particular, the above ideas can be applied directly to albedo and specular curves. Currently, our implementation supports rigid motions only but we do not expect any major difficulty to extend it to non-rigid transformations.

3 Parameterizing a Line Over Time

In this section, we explain how to find a temporally consistent parameterization of a line ℓ. For now, we assume a single open 1-manifold line evolving over time with no topological events. The case of multiple lines and topological events is discussed in the next section. We first discuss the parameterization in geometric terms before translating it into a discrete optimization problem.

3.1 Geometric Formulation

Lines as Time Slices During the animation, a single open line ℓ sweeps a space-time surface $\mathcal{S}$ that has a disc topology. We seek a uv parameterization of $\mathcal{S}$, that is, a function $f(u, v)$ such that $f(\mathbb{U}, \mathbb{V}) = \mathcal{S}$ with $\mathbb{U}$ and $\mathbb{V}$ the spaces on which u and v are defined. Because our goal is to parameterize the lines $\ell(t)$ which are "slices" of $\mathcal{S}$ along planes orthogonal to the t axis, we impose that the v parameter is the time variable t. That is, we seek a function $f(u, t)$ such that for a given $t_0 \in \mathbb{T}$, $f(\mathbb{U}, t_0) = \ell(t_0)$.

Temporal Coherence To ensure the coherence between the 3D motion and the 2D drawing, the trajectory of a point on the line should match the trajectory of its corresponding 3D point. This implies that the speed of the 3D model projected in the image plane should be equal to the speed of the line. Formally, at a given time t_0, we seek:

$$\frac{\mathrm{d}}{\mathrm{d}t}\pi\big(\mathbf{P}(t)\big) = \frac{\mathrm{d}}{\mathrm{d}t} f\,(u_0, t) \tag{1}$$

where u_0 is the parameter of the projection of $\mathbf{P}(t_0)$ on $\mathcal{S}$ at t_0, i.e. $f(u_0, t_0) = \pi(\mathbf{P}(t_0))$.

Coverage and Distortion We consider two practical cases, *aperiodic* textures such as brush strokes, that stretch to cover the line, and *periodic* patterns such as dotted lines that repeat to ensure coverage (see Figure 5).

$\triangleright$ For aperiodic textures, we seek a parameterization function f that maps the $[0; 1]$ interval onto ℓ. That is, at every time t, we want $f([0; 1], t) = \ell(t)$. Since f is continuous, it is sufficient to consider $\mathbf{p}_1$ and $\mathbf{p}_2$, the end points of ℓ.

$$\{\mathbf{p}_1(t), \mathbf{p}_2(t)\} = \{f(0, t), f(1, t)\} \tag{2}$$

To ensure this coverage, the texture has to be stretched or compressed. Large variations in this stretching/compression yields unsightly results and we would like this distortion to be uniform along the line, which means:

$$\left\| \frac{\mathrm{d}}{\mathrm{d}u} f(u, t) \right\| = \mathrm{length}\big(\ell(t)\big) \tag{3}$$

$\triangleright$ With periodic patterns such as dotted lines, coverage is not an issue since we can repeat the texture as much as needed. Because of this property, we do not need to stretch or compress the texture as in the aperiodic case and we seek to preserve the original aspect of the texture. We name a the length of the pattern, and to prevent distortion, we seek:

$$\left\| \frac{\mathrm{d}}{\mathrm{d}u} f(u, t) \right\| = a \tag{4}$$

3.2 A Least-Squares Approach

In general, the constraints described above cannot be satisfied simultaneously. We use a least-squares approach to find a trade-off. For aperiodic textures, this corresponds to:

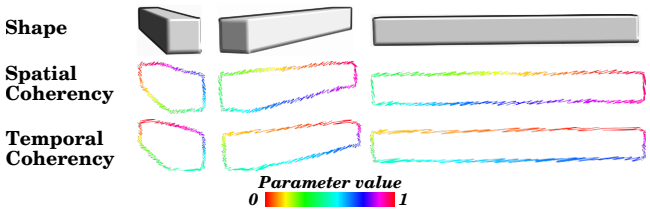


Figure 6: *From spatial to temporal coherency control.*

$$\begin{aligned}\arg\min_f \quad & w_{\text{dis}}\left[\left\|\frac{\mathrm{d}f}{\mathrm{d}u}\right\| - \text{length}(\ell)\right]^2 \\ & + w_{\text{end}}\left[\big(\mathbf{p}_1(t) - f(0,t)\big)^2 + \big(\mathbf{p}_2(t) - f(1,t)\big)^2\right] \\ & + w_{\text{proj}}\left[\frac{\mathrm{d}}{\mathrm{d}t}\pi\big(\mathbf{P}(t)\big) - \frac{\mathrm{d}}{\mathrm{d}t}f(u_0,t)\right]^2\end{aligned} \tag{5}$$

where w_{proj}, w_{dis}, w_{end} control the relative importance of each constraint. With periodic patterns, this becomes:

$$\begin{aligned}\arg\min_f \quad & w_{\text{dis}}\left[\left\|\frac{\mathrm{d}f}{\mathrm{d}u}\right\| - a\right]^2 \\ & + w_{\text{proj}}\left[\frac{\mathrm{d}}{\mathrm{d}t}\pi\big(\mathbf{P}(t)\big) - \frac{\mathrm{d}}{\mathrm{d}t}f(u_0,t)\right]^2\end{aligned} \tag{6}$$

3.2.1 Discretization

In this section, we discretize Equations 5 and 6 so that they can be minimized with standard linear optimization toolkits. First, we regularly sample the time t every Δt and use i to denote the i^{th} frame of the animation, that is, the frame at $t = i\Delta t$. For simplicity, we use an i subscript for entities related to frame i, i.e. $\ell_i = \ell(i\Delta t)$. We also sample each ℓ_i with n_i points $\{\mathbf{r}_{i,1}, \ldots, \mathbf{r}_{i,n_i}\}$. We use $\text{length}(\mathbf{r}_{i,j}, \mathbf{r}_{i,j+1})$ to denote the length of the segment of ℓ_i between $\mathbf{r}_{i,j}$ and $\mathbf{r}_{i,j+1}$. The unknowns of our optimization problem are the scalar values $u_{i,j}$ such that $f(u_{i,j}, i\Delta t) = \mathbf{r}_{i,j}$.

Temporal Coherence Given a point $\mathbf{r}_{i,j}$, we seek to find its corresponding points in frame $i+1$. We consider its 3D counterpart $\mathbf{R}_{i,j}$, that is, $\pi(\mathbf{R}_{i,j}) = \mathbf{r}_{i,j}$, and the 3D line $\mathcal{L}_i$ on which it is located. We search for the corresponding point $\mathbf{R}' \in \mathcal{L}_{i+1}$ at the next frame. If there is no topological event between i and $i+1$, we know that the two lines $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$ are separated by a region $\mathcal{R}$ of the model $\mathcal{M}$ where the line passed, i.e., where $\mathbf{n} \cdot \mathbf{V}$ changed signs in the case of silhouettes (§ 2.1). We find $\mathbf{R}'$ as the closest point $\mathbf{R}_{i,j}$ when considering only paths within $\mathcal{R}$. We implemented this as a shortest path problem on the mesh edges within $\mathcal{R}$. It is also useful to use the image metric instead of the 3D distance since we seek to preserve the coherence in the image plane. Although the camera may be moving, we found it sufficient to use the projection $\pi_{i+\frac{1}{2}}$ at the middle time instant to estimate distances. If more accuracy is needed, one can subdivide the time interval. If there are topological events, we subdivide the frame interval such that each sub-interval has no event and apply the same pairing algorithm within each of them. In practice, we reuse the same subdivision as when we built the space time surface (§ 2.1). Once we find the closest 3D point $\mathbf{R}'$, we project it onto the image plane to get $\mathbf{r}' = \pi_{i+1}(\mathbf{R}')$. We name j' the index of $\mathbf{r}'$ and define:

$$E_{\text{proj}} = \sum_i \sum_j \big(u_{i,j} - u_{i+1,j'}\big)^2 \tag{7}$$

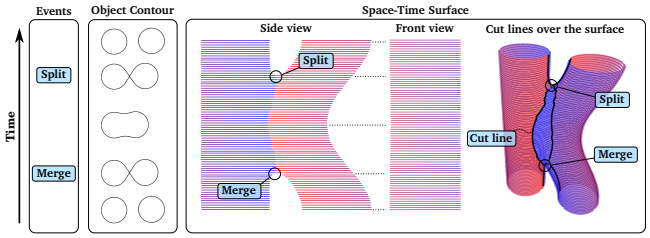


Figure 7: *Line parameterization for a scene with dynamic geometry and topology. Our discontinuity-reuse algorithm pairs the split and the merger, which limits the number and extent of the cuts.*

Coverage and Distortion (Aperiodic Case) The coverage constraint (Eq. 2) is $u_{i,1} = 0$ and $u_{i,n_i} = 1$ for all i. The corresponding energy term is:

$$E^{\text{a}}_{\text{end}} = \sum_i [u_{i,1}^2 + (u_{i,n_i} - 1)^2] \tag{8}$$

The discrete version of the distortion constraint (Eq. 3) is:

$$u_{i,j+1} = u_{i,j} + \frac{\text{length}(\mathbf{r}_{i,j}, \mathbf{r}_{i,j+1})}{\text{length}(\ell_i)} \tag{9}$$

for all i and all $1 \leq j < n$. The corresponding energy term is:

$$E^{\text{a}}_{\text{dis}} = \sum_i \sum_{j=1}^{n_i - 1} \left(u_{i,j+1} - u_{i,j} - \frac{\text{length}(\mathbf{r}_{i,j}, \mathbf{r}_{i,j+1})}{\text{length}(\ell_i)}\right)^2 \tag{10}$$

Distortion (Periodic Case) The equations are similar except that the length of the texture is a and we do not impose the $[0,1]$ coverage.

$$E^{\text{p}}_{\text{dis}} = \sum_i \sum_{j=1}^{n_i - 1} \left(u_{i,j+1} - u_{i,j} - \frac{\text{length}(\mathbf{r}_{i,j}, \mathbf{r}_{i,j+1})}{a}\right)^2 \tag{11}$$

Putting it Together We assemble all the terms to obtain the final energy in the aperiodic case:

$$E^{\text{a}}_{\text{total}} = w_{\text{proj}} E_{\text{proj}} + w_{\text{dis}} E^{\text{a}}_{\text{dis}} + w_{\text{end}} E^{\text{a}}_{\text{end}} \tag{12}$$

and in the periodic case:

$$E^{\text{p}}_{\text{total}} = w_{\text{proj}} E_{\text{proj}} + w_{\text{dis}} E^{\text{p}}_{\text{dis}} \tag{13}$$

This is a classical least-squares energy where the $u_{i,j}$ variables are unknown. This linear system is sparse and can be solved in the least-squares sense. In our implementation, we use a sparse Cholesky factorization.

4 Handling Discontinuities with Cuts

The parameterization algorithm presented in the previous section handles a single open line, that we assume that the space-time surface $\mathcal{S}$ has a disc topology. However, in general $\mathcal{S}$ has a more complex topology and lines split or merge when a topological event occurs. Our strategy is to decompose $\mathcal{S}$ into disc-like charts so that splits and mergers always happen at chart boundaries. Intuitively, when two lines meet, instead of merging them, we keep them separated by cutting $\mathcal{S}$. While the produced lines are shorter, this ensures that there is no discontinuity within a line. Further, we pair mergers and splits so that they share cuts, thereby minimizing the length of these cuts. Figure 7 shows an example of the cuts.

We detect the split and merger points during the construction of the space-time surface (§ 2). We extend these points in time to produce lines free of discontinuities. A possible option is to propagate each cut in time individually. Although this would generate discontinuity-free lines, this would also overly segment the lines and generate visually unpleasing results. We propose a better approach where forward cuts emerging from mergers are combined with backward cuts coming from splits. Considering all possible combinations of forward and backward cuts would generate a combinatorial explosion. We propose a simple heuristic that yields satisfying results in practice.

Once we have detected all the mergers and splits, we analyze the sequence from the first frame to the last, and process the events in order. We describe the case of a merger $\mathbf{k}_i \in \ell_i$, splits are handled symmetrically. We assume that users have specified a parameter σ that represents the maximum sliding that can be introduced for cut reuse. Similar to § 3.2.1, we find the point $\mathbf{k}'_{i+1} \in \ell_{i+1}$ that corresponds to $\mathbf{k}_i$. We collect the points $\mathbf{r}_{i+1,k} \in \ell_{i+1}$ such that $\text{length}(\mathbf{k}'_{i+1}, \mathbf{r}_{i+1,k}) \leq \sigma$ and such that there is no cut between them and the propagated $\mathbf{k}'_{i+1}$ point. If one of them is a split, we pair it with $\mathbf{k}_i$ and nothing needs to be done. If several of them are splits, we pick the one with shortest length. If none of them is a split, we propagate the $\mathbf{r}_{i+1,k}$ to the next frame and iterate the process until we find a split at frame $i+n$. During this process, we keep track of the $\mathbf{k}'$ points that directly corresponds to the initial cut point $\mathbf{k}_i$. If we find several splits at the same time, we pick the point $\mathbf{k}^\star_{i+n}$ for which the distance $\text{length}(\mathbf{k}^\star_{i+n}, \mathbf{k}'_{i+n})$ is the smallest. Finally, we build the shortest path between $\mathbf{k}_i$ and $\mathbf{k}^\star_{i+n}$ on the space-time surface $\mathcal{S}$ and perform a cut along it.

5 Results

We have applied our approach to several scenes with various degrees of dynamism and complexity, ranging from simple static scenes with moving viewpoint to deforming geometry with dynamic topology. Some minor jittering can be observed because we handle visibility at the vertex level – this could be addressed by computing visibility at each pixel at the expense of slower computation times. We use the CHOLMOD solver [Chen et al. 2009] to handle the linear system associated with each space-time line. Solving the least-squares systems takes about 5 seconds for a mesh made of 20k vertices over a sequence of 100 frames. The algorithm's speed depends heavily on the number of split and merge events, and may require up to several minutes for few seconds of animation. Figures 8, 9, 10 and 11 show sample results that we obtain with various shapes. Even with seemingly simple models, the space-time surface can often be complex and nontrivial to analyze, including numerous splits and mergers. Nevertheless, our approach finds a temporally consistent parameterization each time. Our current implementation of the line tracking algorithm handles rigidly moving objects and we are planning to extend it to non-rigid transformations, which requires only technical adaptations since the algorithm itself does not assume this condition. Beside this, our simple proximity-based tracking can handle simple non-rigid animations and demonstrates that our parameterization approach also works in this case. For rendering purpose, we exploit the visibility of the lines stemming from the 3D model to either hide the occluded ones or apply a different style to them.

6 Conclusion

We have described a method to produce temporally-consistent line parameterization that can be used to render animated line drawings. The cornerstone of our approach is the introduction of the space-time surface that represents an animated line. Our approach is grounded on a geometric analysis of temporal consistency based on this surface. We translate this spatio-temporal analysis into a discrete least-squares problem. We have shown that the user can control the trade-off between temporal coherence and distortion using a few meaningful parameters. Furthermore, we avoid over-segmentation by pairing complementary topological discontinuities. Together, these elements enable temporally consistent parameterizations that we demonstrate by rendering several complex animations as line drawings.

Acknowledgements. We would like to thank Forrester Cole, Joelle Thollot and Adam Finkelstein. This work has been partially funded by an Institut Telecom Future & Rupture grant, the ANR KidPocket project and the 3DLife European Network of Excellence.

References

BÉNARD, P., BOUSSEAU, A., AND THOLLOT, J. 2009. Dynamic solid textures for real-time coherent stylization. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*, 121–127.

BÉNARD, P., COLE, F., GOLOVINSKIY, A., AND FINKELSTEIN, A. 2010. Self-similar texture for coherent line stylization. In *NPAR 2010: Proceedings of the 8th International Symposium on Non-photorealistic Animation and Rendering*, ACM Press.

CHEN, Y., DAVIS, T. A., HAGER, W. W., AND RAJAMANICKAM, S. 2009. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Software 35*, 3.

COLE, F., GOLOVINSKIY, A., LIMPAECHER, A., BARROS, H. S., FINKELSTEIN, A., FUNKHOUSER, T., AND RUSINKIEWICZ, S. 2008. Where do people draw lines? *ACM Transactions on Graphics 27*, 3 (Aug.), 88:1–88:11.

CUNZI, M., THOLLOT, J., PARIS, S., DEBUNNE, G., GASCUEL, J.-D., AND DURAND, F. 2003. Dynamic canvas for non-photorealistic walkthroughs. In *Graphics Interface 2003*, 121–130.

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

DECARLO, D., FINKELSTEIN, A., AND RUSINKIEWICZ, S. 2004. Interactive rendering of suggestive contours with temporal coherence. In *NPAR 2004*, 15–24.

GRABLI, S., TURQUIN, E., DURAND, F., AND SILLION, F. X. 2010. Programmable rendering of line drawing from 3d scenes. *ACM Transactions on Graphics 29*, 2 (Mar.), 18:1–18:20.

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating smooth surfaces. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 517–526.

JUDD, T., DURAND, F., AND ADELSON, E. 2007. Apparent ridges for line drawing. *ACM Transactions on Graphics 26*, 3 (July), 19:1–19:7.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: Drawing strokes directly on 3D models. *ACM Transactions on Graphics 21*, 3 (July), 755–762.

KALNINS, R. D., DAVIDSON, P. L., MARKOSIAN, L., AND FINKELSTEIN, A. 2003. Coherent stylized silhouettes. *ACM Transactions on Graphics 22*, 3 (July), 856–861.

KALNINS, R. D. 2004. *Interactive stylization for stroke-based rendering of 3D animation*. PhD thesis, Princeton University.

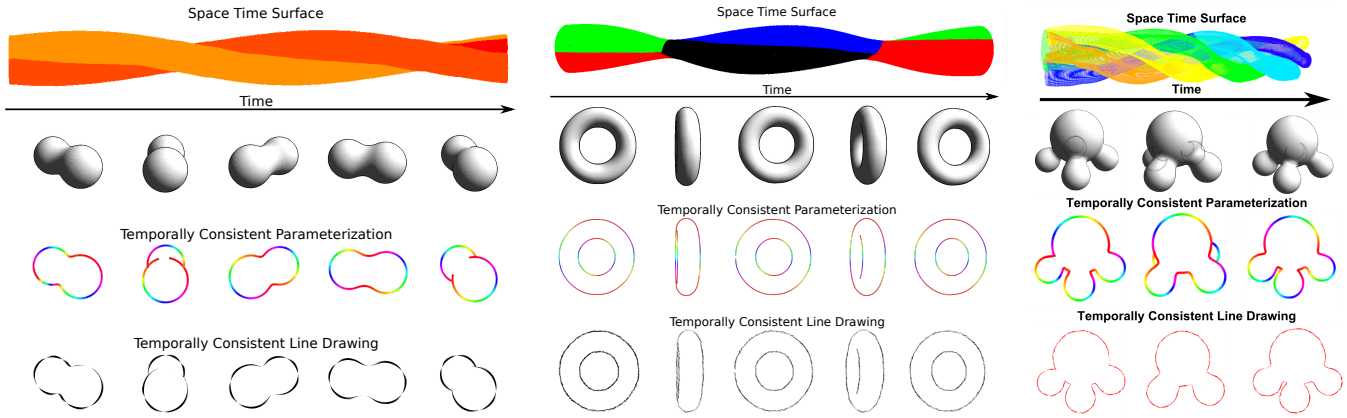
KIM, Y., YU, J., YU, X., AND LEE, S. 2008. Line-art illustration of dynamic and specular surfaces. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia) 27*, 5.

LEE, Y., MARKOSIAN, L., LEE, S., AND HUGHES, J. F. 2007. Line drawings via abstracted shading. *ACM Transactions on Graphics 26*, 3 (July), 18:1–18:5.

SHESH, A., AND CHEN, B. 2008. Efficient and dynamic simplification of line drawings. *Computer Graphics Forum 27*, 2, 537–545.

STROILA, M., EISEMANN, E., AND HART, J. 2008. Clip art rendering of smooth isosurfaces. *IEEE Trans. Vis. Comput. Graph. 14*, 1, 135–145.

WINNEMÖLLER, H., OLSEN, S. C., AND GOOCH, B. 2006. Real-time video abstraction. *ACM Transactions on Graphics 25*, 3 (July), 1221–1226.



(a) "peanut" shape (b) torus (c) "multipod" shape

Figure 8: *Even simple shapes can generate complex lines. The silhouette of a peanut folds in nontrivial ways (a), the inner and outer silhouettes of a torus repeatedly merge and split (b), and the multipod generates many visibility events (c).*

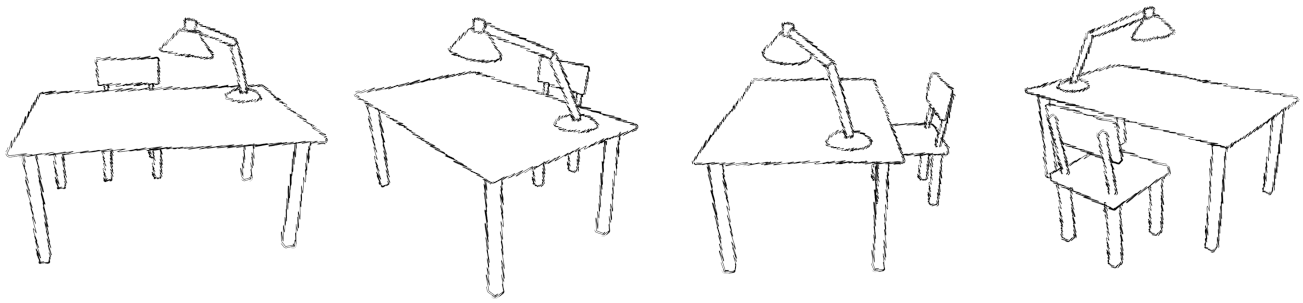


Figure 9: *Four frames of a desk model, with lines only.*

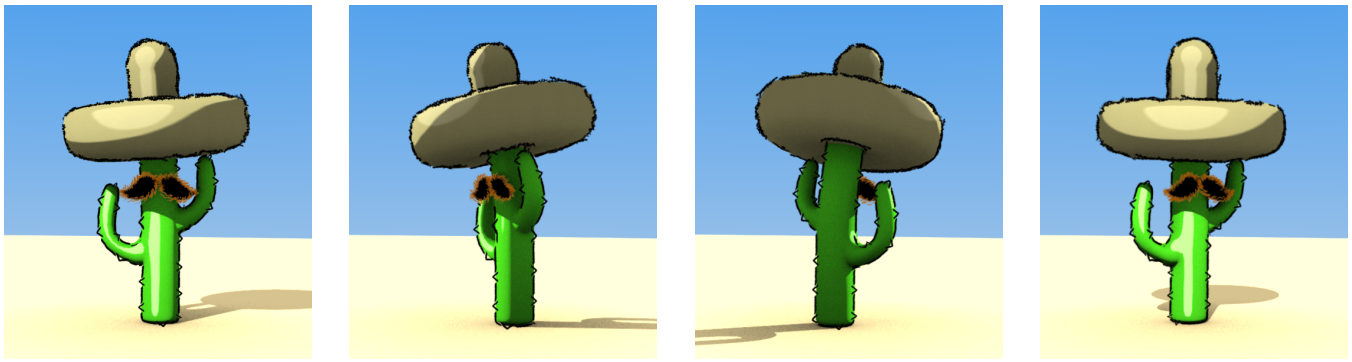


Figure 10: *Four frames of a more complex example: the cactus scene.*

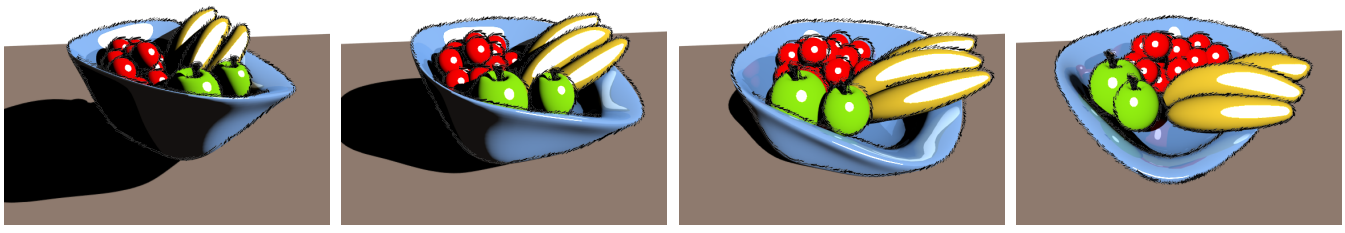


Figure 11: *Four frames of an additional example.*