Stylizing Ribbons: Computing Surface Contours with Temporally Coherent Orientations

Nora S Willett*, Fernando de Goes*, Kurt Fleischer*, Mark Meyer* and Chris Burrows*
*Pixar Animation Studios
Emeryville, CA

Abstract—Line work is a core element for the stylization of computer animations used by recent shows. However, existing stylization techniques are limited to edge treatments based on brush strokes or textures applied solely on top of curves. In this work, we propose new stylization effects by offering artists direct control over the inside and outside of surface contours. To this end, we introduce a method that creates *ribbons*, geometry strips of possibly varying width, that extrude from each side of the surface contour with temporally coherent orientations. Our contributions include the generation of spatially and temporally consistent normal orientations along visible contours and a trimming routine that converts arrangements of offset curves into ribbons free of intersections. We demonstrate the expressiveness and versatility of stylized ribbons by applying various effects on both character and shadow edges from animation sequences.

1 Introduction

In recent years, there has been a resurgence of interest in stylized looks that blend 2D and 3D visual effects in computer animations. Some, like *Klaus* [1], use a 2D style of animation with added 3D elements for a multi-dimensional effect. Others, like *Spiderman: Into the Spiderverse* [2] and *Soul* [3], incorporate 2D line work to enrich the visuals of 3D characters. In all of these instances, new stylization tools were developed to assist artists [4], [5], complementing research efforts in non-photorealistic rendering [6] and video stylization [7], [8].

An important aspect defining a stylized look is the rendering of surface contours. These curves can be used to convey shapes, emphasize expressions and motions, or depict visual aesthetics [9], [10]. For instance, the short animation *Paperman* [11] emulated hand-drawing by overlaying toon-shaded objects with tapered lines. While many techniques have focused on extracting contours based on occlusion or geometric rules, curve stylizations are achieved mainly through brush strokes (see the survey of [12]). As a result, these methods are limited to generating effects directly over edges, with no separate control of the outside or inside of the contours.

To apply a style on either the inside or outside of a surface contour, an orientation along the curve is necessary. At first glance, it is tempting to compute this orientation from the ordering of points along the curve or from the surface normals. Unfortunately, both processes suffer from temporal inconsistencies. For instance, the ordering of points can change over time causing the orientation to flip. Alternatively, surface normals can be copied to the curves so that the outside is in the same direction as the normals projected to the camera canvas. These projected normals work well in some cases (e.g., silhouette contours [13]) when curves have correspondences to single locations on the surface with normals nearly parallel to the camera plane. However, for other curves such as surface intersections, these normals are ambiguous as they lie between different areas of the 3D model (Figure 5b). Ideally, these ambiguities should be resolved into temporally coherent curve normals across frames while matching the orientation of any connected silhouette curves. Additionally, surface normals can be skewed relative to the camera plane, leading to projection degeneracies (Figure 5a). Altogether, these issues make the computation of spatially smooth and temporally coherent orientations for surface contours challenging.

Offsetting oriented surface contours also requires special trimming treatments. In particular, extruding curves to only one side can extend over other curves thus creating distracting visual clutter, especially near sharp angles (Figure 2b). To resolve overlaps between single-sided offsets, one must trim the extruded geometry by accounting for the curve arrangement in the camera plane (Figure 2c). Additionally, offset curves must also be adjusted in areas of high curvature in order to produce smooth geometry strips with no self-intersections.

In this work, we present a novel technique that enables the stylization of the inside and outside of surface contours (Figure 1). Given a collection of surface contours, our algorithm generates oriented geometry strips, which we refer to as *ribbons*, that define a temporally stable parametrization on each side of the curves. Our approach is centered on two main contributions. First, we introduce an algorithm that interpolates normals smoothly both along curves and between frames, thus producing temporally consistent curve orientations. Next, we describe a strategy for offsetting surface contours towards the inside and outside of the curve normals while trimming folds and intersections. Equipped with these ribbons, we showcase a variety of new line stylizations on 3D animations, including edge effects on characters and shadows.

2 Related Work

In this section, we position our work with respect to existing techniques for the extraction, tracking, and stylization of surface contours. We also refer the reader to the survey by Bénard and Hertzman [12] for a thorough discussion of curve stylization.

Before stylization, contour curves must be detected over 3D objects. These curves are often extracted from silhouettes (a.k.a.,

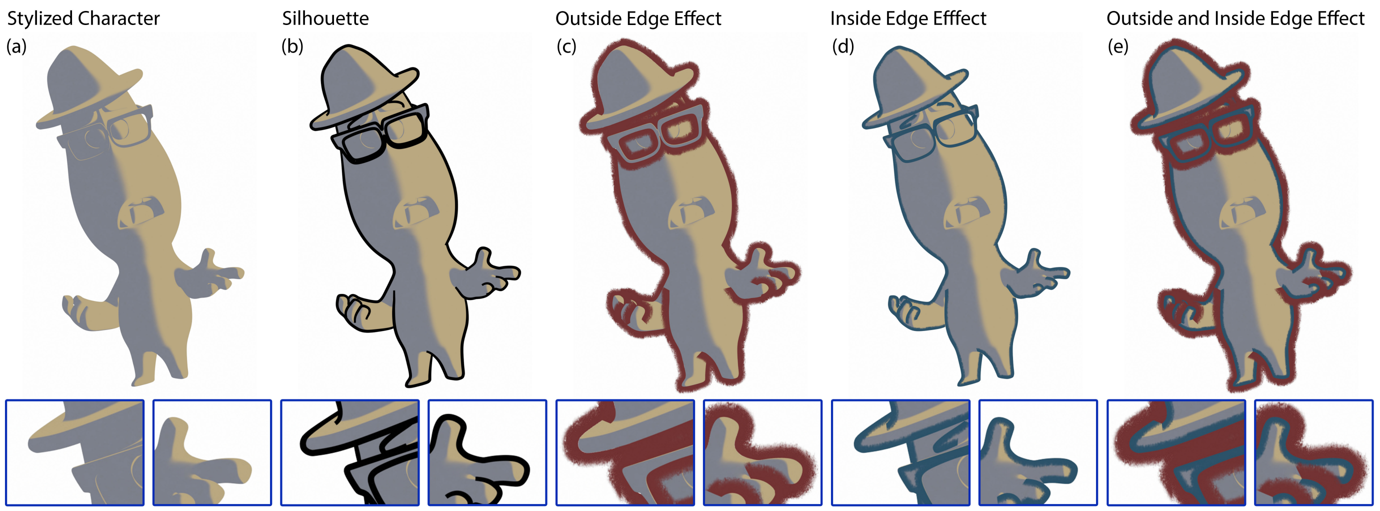

Fig. 1. **Surface Contours with Temporally Coherent Orientations:** Starting with a surface mesh and a camera viewpoint (a), we describe a technique that generates smooth visible contours (b) with normal orientations consistent both along curves and across frames. With these oriented curves, we construct trimmed ribbons, a thick outside one (c) and a thinner inside one (d). Our method allows artists to apply new stylized effects on either side of surface contours (e). ©Pixar

occluding contours) [14] and surface-to-surface intersections [15]. They may also be rigged or computed based on curvature rules such as suggestive contours [16], ridges and valleys [17], and apparent ridges [18]. For performance, sometimes contours are computed directly on the GPU [19]. Some techniques also generate curves directly in the 2D rasterized image [20], [21]. Others use neural networks for curve extraction [22] and stroke stylization [23]. Our work receives any combination of these curves as input and augments them with a camera-aligned orientation.

An important component of edge stylization is the visibility test. The simple approach of checking visibility by ray tracing from the camera origin is sensitive to the surface tessellation and often leads to fragmented curve pieces [14]. To alleviate this issue, Bénard et al. [15] propose a local remeshing strategy that refines the surface tessellation favoring contour consistency at the cost of increased computation. Liu et al. [24] improve that tessellation method for better performance and proven mesh consistency. An alternative method is to propagate visibility hints along curves using a graph of the curve arrangement in the camera canvas [25]. Planar graphs further generalize this propagation scheme by also accounting for the visibility of the 2D regions demarcated by the image curves [26]. We use a voting method similar to Grabli et al. [27] that estimates the percentage of visible points scattered near

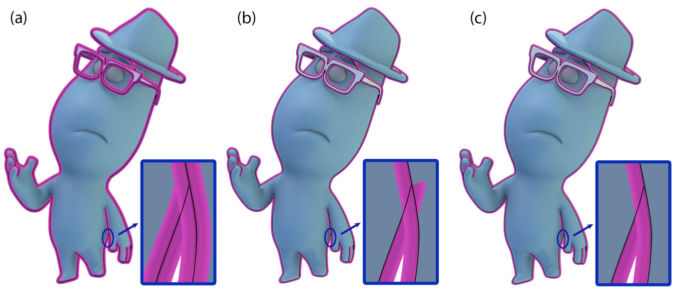

Fig. 2. **Importance of Ribbon Trimming:** When a stroke of constant width is applied on both sides of a curve, no trimming is necessary (a). However, when the stroke is applied on only one side, undesirable overlap between ribbons is introduced (b). We present a method for trimming ribbons that prevents overlaps (c). ©Pixar

the contour curves. By doing so, we keep the efficiency of ray tracing tests while producing clean and smooth surface contours.

Once curves are extracted for each frame, they must be tracked over time in order to enable temporally consistent styles. To this end, Kalnins et al. [21] advect curves in 3D attached to the surface mesh followed by 2D splitting and merging operations. Similarly, the method of Whited et al. [13] carries strokes along the 3D animation and provides inbetweening interpolation and editing tools. The works of Karsch and Hart [26] and Bénard et al. [28] adapt the concept of active contours so that 2D curves expand and shrink over time. In Buchholz et al. [29], a space-time parametrization is optimized to fit the evolution of surface contours, while Bénard et al. [30] splats curve parameterizations onto an auxiliary screen-space buffer for temporal look-ups. In common, these methods can only track curve centerlines with no normal orientation.

The problem of oriented contours was previously investigated by Lengyel et al. [31] in the context of fur creation along outside silhouettes. To break up the outside edge of the model, this method generates a face in the normal direction at every surface edge to which a fur texture is applied. However, this approach introduces noise inside the model and gaps in the textured edges around areas of high curvature. Another technique was used on the short film *Out* [32] by computing signed distance functions from the mask of object IDs outputted by a renderer. This method to determine the inside and outside of closed curves and models can use signed distance fields or winding number generation. Although simpler, this approach produces 2D orientations only along the outer silhouette ignoring self-occlusions, e.g., a hand crossing the torso of a character.

The estimation of oriented normals is also a typical task in reconstruction pipelines from raw point clouds [33], [34]. By assuming that points approximate a manifold, normal orientation can be inferred through spectral methods [35] or stochastically by tracing rays against a capsule that encompasses the unoriented points [36]. Neural networks have also been considered for learning oriented normals from point clouds [37], [38]. Unfortunately, these methods do not account for normal consistency between frames. Our approach for orienting contours is inspired by the

Algorithm 1 Pseudocode to compute oriented ribbons.

```
 1: function COMPUTING VISIBLE CURVES (§3.1)
 2:     Extract surface contours
 3:     Split curves at sharp angles and intersections
 4:     Compute visibility probability per curve point
 5:     Keep visible points and curve sections
 6:     Merge curves close together
 7:     Remove colinear edges
 8:     Delete small loops and short edges
 9: function NORMAL ORIENTATION ENFORCEMENT (§3.2)
10:     for each frame in animation do
11:         Set normal direction to angle bisectors
12:         Compute normal orientation and probability
13:         if not first frame then
14:             Advect normals from previous frame
15:             Update normal probability
16:         Optimize normals using Markov Random Field
17: function RIBBON TRIMMING (§3.3)
18:     for each connected group of curves do
19:         Compute medial axis from curve points
20:         Compute width edges by offseting curves along normals
21:         Trim width edges against medial axis
22:     Remove overlaps between different curve groups
23:     Trim T-junctions
```

work of Hoppe et al. [39], which calculates spatially smooth tangent planes for point clouds via a graph optimization. We adapt this formulation to compute spatially and temporally coherent normals for 2D curves.

Another common step in curve stylization is the thickening of curves into strokes. When extruding the width of a curve, difficulties arise in areas of high curvature as the expanded width may create self-intersections within valleys or gaps nearby cusps [40]. This problem is also encountered by vector graphics packages when converting strokes to filled primitives, which was recently addressed by Nehab [41] and Kilgard [42]. However, while following the vector format guidelines, existing solutions do not trim overlaps and are limited to expand the curve equally on both sides. In our work, we address the case of offsetting curves per each side, and propose an extended trimming routine that removes overlaps between multiple filled strokes.

Finally, we point out a few research efforts that have also investigated stylized shadows. For instance, DeCoro et al. [43] describe controls for the inflation, brightness, softness and abstraction of shadows, while Todo et al. [44] integrate localized shading into conventional lighting techniques.

3 METHOD

In this section, we detail our process of computing visible surface contours with temporally consistent normals. Equipped with oriented curves, we then generate ribbons by offsetting curve points and trimming those geometry strips to prevent overlapping areas. An outline of our method is given in Algorithm 1. A summary of all the parameters used can be found in Table 1 at the end of the paper.

3.1 Computing Visible Curves

Our method starts with curves extracted from the surface mesh (one could use a combination of techniques summarized by Bénard et al. [12]). In our examples, we include silhouette curves generated by tracing the zero levelset of the dot product between normals at mesh vertices and the camera direction [14], as well as surface intersections computed via boolean operations [15] (Figure 3a). All these curves are represented by 3D polylines.

From the surface contours, we need to determine which points are visible. Our approach shares commonalities with the method proposed by Grabli et al. [27]. We calculate ray tests for multiple samples scattered around each curve point (Figure 3c). Then, we define the likelihood that a curve point is visible as the ratio of visible samples versus the total samples (Figure 3d). A curve point should be considered visible if approximately half of the samples are visible. Hence, to map the likelihoods to the *visibility probability*, we remap all values greater than zero by adding 0.5 and clipping to one. This remapping ensures that completely occluded points still have zero probability while visible points have values close to one.

We generate samples ($n = 20$) inside two cylinders placed relative to the camera frustum space, each running halfway along the curve edges adjacent to each curve point (Figure 3c). Using cylinders in the camera frustum, instead of the 2D image or 3D world space, provides adaptive sampling that accounts for the curves' depth. The cylinder radius is a user adjustable parameter measured in pixels, which we default to 0.3 in our experiments. If the radius is too small, points which are slightly occluded receive small probabilities, creating gaps in the transitions between silhouette and intersection curves. On the other hand, if the radius is too large, then points which are occluded at intersections receive inflated probabilities, resulting in curves extending too far into occluded regions.

Our next step is to consolidate the visibility probability of individual points into a probability per curve indicating if the curve is either fully or partly visible. To this end, we use a 2D chopping scheme (informed by Hertzman and Zorin [14]) that splits curves into sections at sharp angles (we used a threshold of 100 degrees) and at 2D intersections computed after projecting the curves to the camera plane (Figure 3b). We then extract the visible contours as a sequence of points with probability greater than 0.8, or entire curve sections with an average visibility probability greater than 0.8. The remaining points are removed (Figure 3e).

We also apply a *clean up* pass that simplifies visible curves in order to avoid visual clutter (Figure 4a). Numerous other works had addressed a similar problem when simplifying sketch drawings [45], [46], [47]. First, we consolidate visible curves that are close together in the image space. If these curves are left as distinct parts, clipping artifacts can occur when they are extruded. Informed by Northrup and Markosian [48], we search for curve points whose screen-space distance to any edge segment is within a threshold (0.15 pixels), and whose difference in camera depth is smaller than another threshold (0.15 world space camera depth) to preserve depth discontinuities. We then merge these points to the midpoint of their corresponding closest edges (Figure 4b). We also combine collinear edges that often appear due to sharp switchbacks along a curve. In this case, we look for any angle between two segments which is less than 30 degrees. If one of these edges is the end of a curve, we remove it, otherwise the

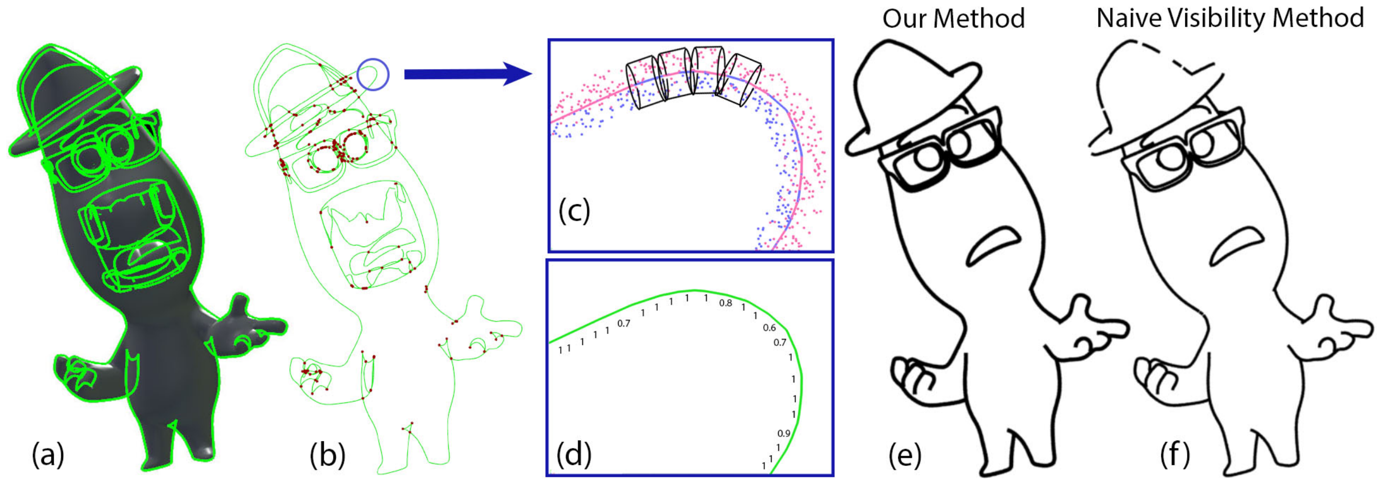

Fig. 3. **Visible Contours:** Starting with silhouette and intersection contours detected on a 3D model (a), we split curves into sections at intersections and sharp angles shown in red (b). We scatter samples in cylinders shown in black around the 2D curves and mark the visible ones in pink and those occluded in blue (c). We then use these labeled samples to calculate visibility probabilities for every curve point (d). Our method produces smooth visible curves (e), while the traditional single ray test creates gaps in the lines (f). ©Pixar

shorter edge is deleted and a new edge is created between the other endpoints (Figure 4c).

Drawing inspiration from vectorization methods [49], we also remove curve loops which have an area smaller than 0.4 percent of the image resolution. We delete every loop edge except for edges which form nearly straight lines with their non-loop neighboring edges (Figure 4d). Finally, we discard short edges of length less than two pixels and reconnect the remaining edges into new curves breaking at intersections (Figure 4e). The steps are run through one time in the order presented due to each step never introducing any edge connections that a previous step would remove. While small loops and edges are not noticeable when directly rendered, these artifacts can create sharp peaks and temporal inconsistencies when offsetting the curves.

While these methods were used as input to the next steps for creating consist normals and ribbons, any previous methods [15], [27] that generate stable clean 3D curves could also be used for pleasing results.

3.2 Normal Orientation Enforcement

In order to create effects inside and outside visible contours, we now describe how to generate 2D normals which are consistent along the curves and across frames. We begin by setting the (unoriented) normal direction of each curve point to the 2D angle bisector between adjacent curve edges. At curve endpoints, we use the direction perpendicular to the line segment.

The next step is to determine on which side of the curve the normal should be pointing while accounting for temporal consistency. A simplistic option is to ray trace 2D curve points back to the 3D surface along the camera direction, and align the angle bisectors to the normals copied from the first ray hit on the mesh projected to the camera plane. While copying mesh normals works well in some cases, the correspondence from camera to surface can be ill-defined, especially near intersection curves (Figure 5b), and even introduce skewed projected normals (Figure 5a). Instead, we propose to quantify the uncertainty of the normal estimation by assigning a *normal probability* to each curve point. Similar to §3.1, we sample n points in two cylinders around each half of the edges incident per curve point (our examples used $n = 20$ and a cylinder radius of 0.25 pixels). Then 2D normals are calculated for every sample point by tracing the closest mesh normal projected to the camera canvas. We orient the angle bisector at a curve point to the direction of the majority of the sample normals. We set the normal probability to 1 if the percentage of the predominant normals is greater than 90%, otherwise we set the probability to 0.5 indicating that the normal is between conflicting areas.

For silhouette curves on the edge of a character, this probability will always result in a value of 1. For intersection curves or silhouette curves close to opposing pieces of the mesh (eg. a closed mouth), the resulting probability depends on the orientation of the intersecting geometry relative to the camera. For example, if two intersecting patches are oblique to the camera, our frustum-based sampling will bias towards the region that is closer to the camera. Only if the intersections are facing the camera, the normal probability will be closer to 0.5, but this is a special case.

At the first frame of an animation sequence, we use these normal estimates and probabilities to initialize an optimization that infers spatially smooth orientations along the curves. We use a Markov Random Field [50] and define the potential function (a.k.a., correlation function) as the dot product of the initial normals between neighboring curve points. Hence, neighboring

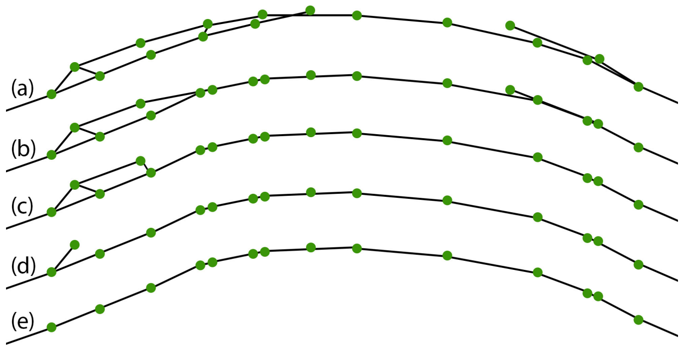

Fig. 4. **Curve Clean Up:** Starting from curves projected to the camera canvas (a), our method merges curves close together (b), removes colinear edges (c), and deletes small loops (d) and short edges (e).

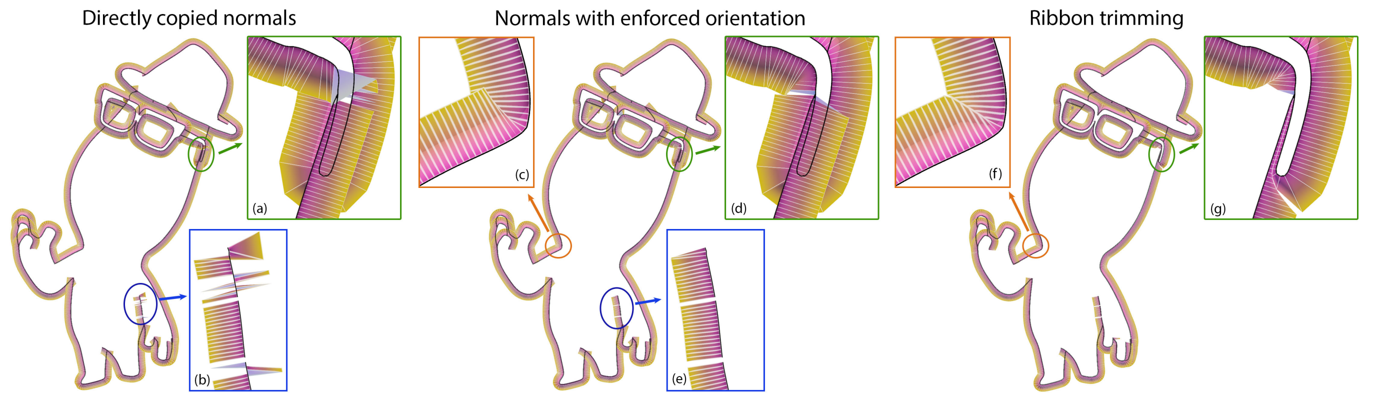

Fig. 5. **Orienting Normals and Trimming Ribbons:** We render the original curve in black and ribbons with colors varying from pink to yellow indicating the distance to the curve. If we extrude curves using the normals directly copied from the surface mesh, the resulting ribbons have artifacts due to skewed (a) and inconsistent (b) normals. Our method resolves these artifacts by producing smooth oriented normals both along curves and across frames (d,e). By trimming ribbons, we also prevent intersections (c,f) and overlaps (g). To see video comparisons between the steps in the normal orientation enforcement method, please see the supplemental materials. ©Pixar

normals which face opposite directions are penalized. With this setup, we perform the optimization using a linear programing relaxation [51], resulting in consistent 2D orientations for sections of the curves in the first animation frame (Figure 5d,e).

For subsequent frames, we adapt the Markov Random Field to also generate temporally consistent normals. First, we use the method described above to calculate the normal probability for each point in the current frame. For every point with a probability of 0.5, we try to improve the normal estimate by searching for a matching location in the previous frame. To determine correspondences between curves from different frames, we advect the curves from the previous frame based on the mesh movement between frames [28]. When the corresponding curves are close together, we copy the previous normal and set the normal probability to one, otherwise we keep the estimated normal and probability of 0.5. Using these new probabilities and the same potential function as before, we optimize the Markov Random Field once again for every animation frame sequentially.

3.3 Ribbon Trimming

We define a *ribbon* as a geometry strip created by extruding curve points along either side of their respective 2D normals based on a user specified width. We generate an outside ribbon when points are offset along the curve normal, and an inside ribbon using the offset in the opposite direction. However, these thickened curves may exhibit areas where the strip faces overlap (Figure 5c) or cross each other (Figure 5d). In order to produce clean ribbons for stylization, we present a routine for trimming ribbons.

Our approach operates by trimming width edges, which are 2D line segments connecting curve points to their corresponding offset points (Figure 7a). To remove local overlaps, we calculate the medial axis for each group of connected curves and trim the width edges at the points intersecting their respective medial axes (Figure 5f). Since curves from our clean up pass were split at intersections, a connected group of curves are those curves which share end points. In our implementation, we sample every visible contour densely in 2D (0.1 pixels between points) and approximate the medial axis using the method of Amenta et al. [52], which constructs a Voronoi diagram and filters out Voronoi edges intersecting the curve segments (Figure 7b). However, medial axes are sensitive to small perturbations of the curves. Hence, we add a filtering step to prune unwanted branches based on a scale axis representation [53] by removing those edges that lie within a Voronoi cell of the original curve points (1 pixel between points) (Figure 7b). In addition, we disregard intersections if the width edge and medial axis edge are approximately parallel (Figure 7c,d). While we chose this particular implementation of medial axis creation and filtering for these experiments, other techniques could be used instead.

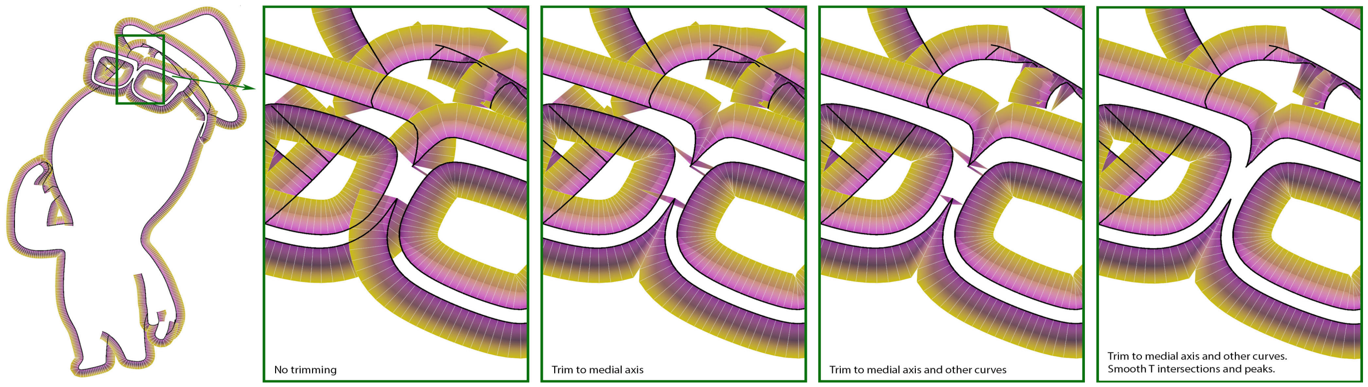
No trimming
Trim to medial axis
Trim to medial axis and other curves
Trim to medial axis and other curves.
Smooth T intersections and peaks.

Fig. 6. **Ribbon Trimming Steps:** The ribbons are trimmed to the medial axis and other curves. Then, offsets at any T intersections or peaks are smoothed. ©Pixar

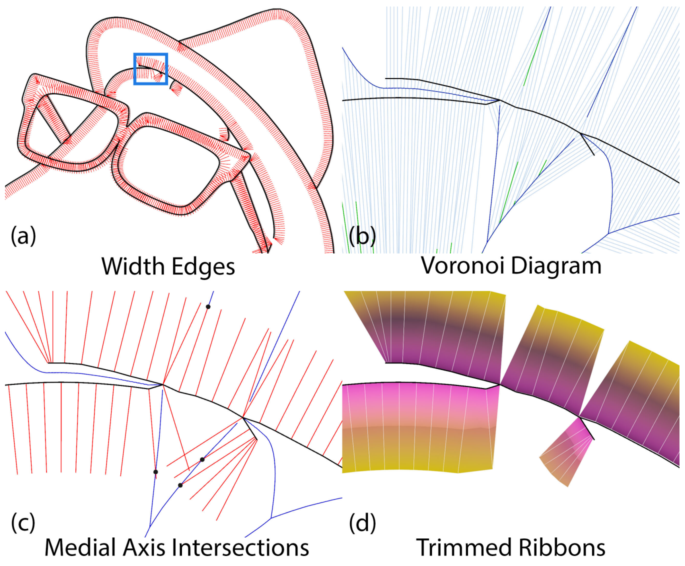

Fig. 7. **Trimming to Medial Axis:** In (a), the original curves are displayed in black with width edges in red. The following images (b,c,d) are from the close up highlighted by the blue box in (a). Image (b) visualizes the Voronoi diagram (in light blue) generated from the densely sampled curves. Removing the Voronoi edges intersecting the curve segments still results in some edges (in green) that need to be filtered to achieve the medial axis (in dark blue). In (c), we indicate the intersection trim points (black dots) calculated from the width edges and filtered medial axis. The final trimmed ribbons are shown in (d).

To remove overlaps between different groups of connected curves, we copy the camera depth from the original curve points to their respective width edges. We then locate intersections between width edges and all other original curves, and set the intersection depth to the original curve's camera depth at that point. We trim a width edge when its camera depth is within a small threshold (0.2) or greater than its corresponding intersection point camera depth. Thus, we prevent a ribbon behind a piece of geometry from overlapping that area when projected to the camera canvas (Figure 2c and 5g).

Lastly, we address the special case when a curve slightly extends past a T-junction intersection without trimming. While on a single frame, this artifact may not be noticeable, switching from a clean to an extended curve over time causes flickering. To resolve this problem, we detect T-junction intersections and, if the last few points of the curve (we used 3 points) are untrimmed, we trim them to the same amount as the closest curve point (Figure 6).

For more examples of each step in the ribbon trimming process, please see the supplemental materials.

4 Results

We now showcase various stylization effects produced from our oriented ribbons. We point the reader to the accompanying video and supplemental materials for the complete animation sequences. Note that these results are test sequences, this technique has not yet been used in any productions. Our method was implemented in Python and C++ as a digital asset in Houdini [55], with the Markov Random Field solver available in Müller and Behnke [51]. Each part of our process has a few parameters, as described above, whose same settings were used in all of the shots. For profiling purposes, we computed the average timing per frame spent by

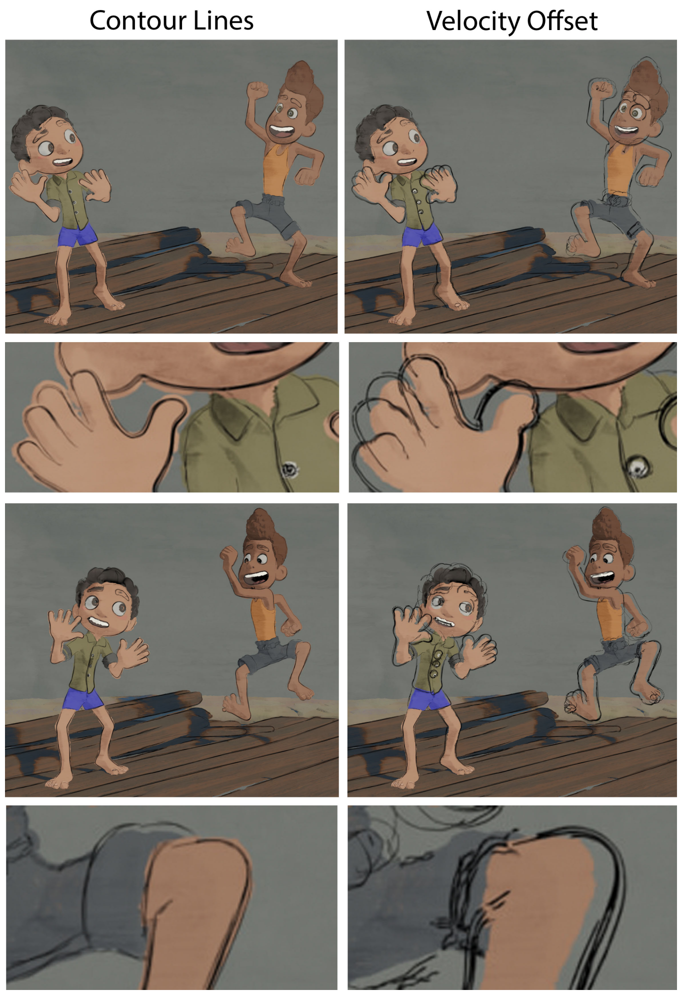

Fig. 8. **Velocity-based Offsets:** We apply two edge stylizations to a watercolor-shaded animation. In the left, smooth visible contours generated by our method are rendered using an ink texture with varying thickness based on the distance to the curve endpoints. In the right, we use our temporally consistent normals to offset curves by a width proportional to the velocity magnitude. ©Pixar

each step of our algorithm for the example in Figure 1 (334 frames, 488,712 polygons, clocked on a 2.3 GHz Intel XeonE5-2699 with 12 cores). In the curve creation, the visibility culling took 70 seconds and the clean up pass lasted 2 seconds. The curve registration between frames lasted 9 seconds, while normal enforcement finished in 12 seconds and the ribbon trimming in 63 seconds.

In Figure 1, we present a breakdown of our method applied to a single animation frame. First, we show our smooth contour curves rendered with strokes of constant thickness. While curves are automatically generated over the whole body, artists prefer to have manual control over the acting around the eyes and mouth. Therefore, our system allows for a mask to be painted on the characters to disable curves in such regions - which we used in this example. From the visible contours, we add outside ribbons rendered with a red sponge texture of constant thickness, followed by inside ribbons with a blue sponge texture at half of that thickness. At last, we combine the inside and outside ribbons together and trim them to resolve overlaps and occlusions. Additionally, we include several frames from this animation in

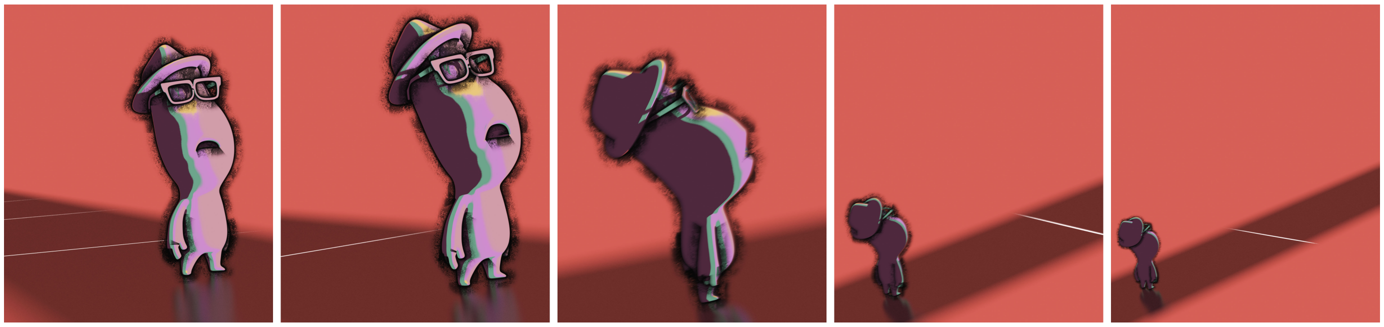

Fig. 9. **Ribbons with Camera-based Widths:** For this style, the character is rendered with a toon shader and an outside ribbon is applied with the width varying based on the distance to the camera. As the character moves farther away from the camera, the ribbon width decreases. ©Pixar

Figure 16 to illustrate that our results are spatially and temporally consistent.

We also used this same animation sequence to compute the stylization shown in Figure 10. In this case, we applied a projection of a stylized caustic texture inside the character and attached ribbons inside the surface contours with varying width in order to produce a wave-like look. This wave effect was controlled by a sine function parametrized based on locations along the curves tracked across the frames. We also animated the wave magnitudes by scaling the ribbon width based on the distance to the curve endpoints.

Figure 8 (57 frames) shows an animation of two boys dancing with a watercolor look as the base. In this example, we first highlight the contours of the characters by rendering our smooth visible curves with ink textures tapered close to the ends. To emphasize the characters' movement, we also used our temporally coherent 2D normals to offset the contour curves by a distance proportional to the magnitude of the animation velocity at the curves. Moreover, the thickness of the strokes were adjusted so that faster moving curves are thicker.

In Figure 9 (257 frames), we applied a stylization effect that emulates a glow over a character. To this end, we rendered only the outside ribbons with width growing thicker as the character moves closer to the camera. We also modulated the density of the rendered textures within the ribbons using the distance to the curves, with denser textures close to the curve and less dense father away.

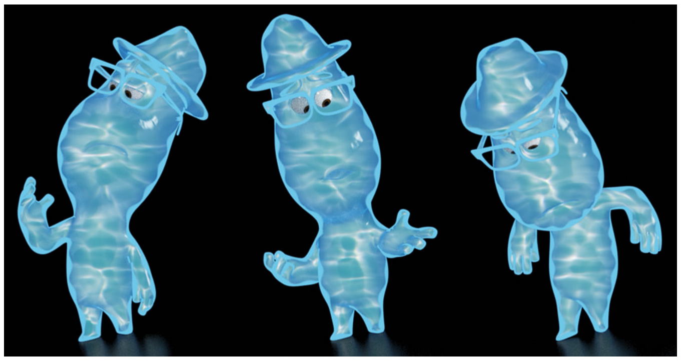

Fig. 10. **Ribbons with Animated Width:** In this example, the character is rendered with a caustic effect and has an inside ribbon with the width controlled by a sine function that varies based on the frame and on the location along the curves. ©Pixar

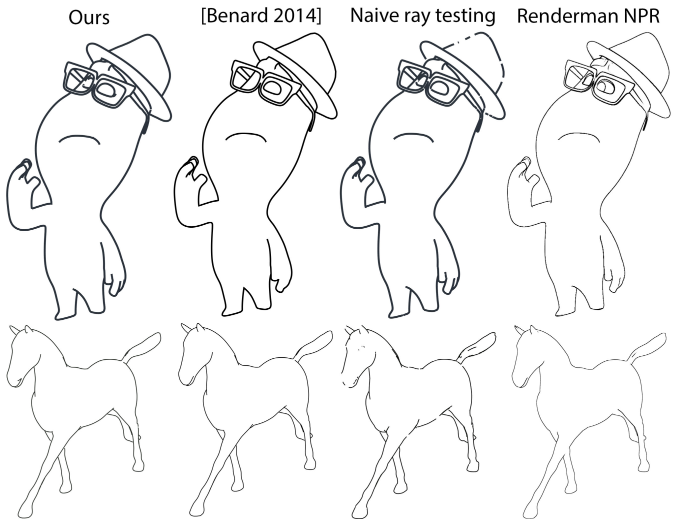

Fig. 11. **Comparison:** Comparing our method to previous work on the example from Figure 1 and an animated horse model from [54]. ©Pixar

Additionally, we considered the stylization of shadow edges in Figure 14 (30 frames). To generate the silhouettes along shadows, we replace the camera direction by the light source and then compute visibility relative to the light. We also included curves inside the shadow towards and away from the light and applied different styles for the inside and outside ribbons. In contrast to previous shading approaches [43], our method provides shadow effects that maintain constant width along the contours.

In Figure 12, the shadow is projected down the steps in front of the character. With a simple projection, the ribbon would stretch in a diagonal down each step. To wrap the ribbon around the step, stretched edges, where the projected edge is 1.5 times the original edge length, are subsampled, with the new points moved to the closest position on the ground. This process is repeated until all stretched edges are less than the minimum original edge length. For the color effect, the variation is determined by the height from the ground of the pre-projected curves.

Figure 15 showcases both shadow ribbons and character ribbons. The inside shadow ribbons have their width controlled by tapering along the curves and the pre-projected distance to the ground. The character ribbons are tapered as well with color bands ranging from red, yellow, and blue for skinnier to thicker areas.

To determine the reliability of our method, we compare our generated silhouette curves with a naive visibility ray test, the method from Bénard et al. [15], and RenderMan's NPR lines

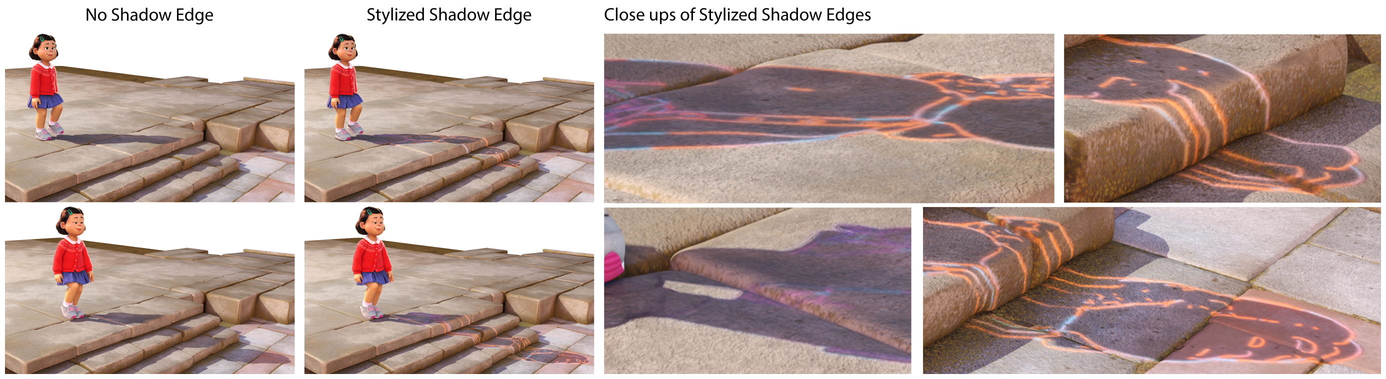

Fig. 12. **Deformed Shadow Edges:** Stylized outside ribbons away from the light are projected down the steps with their color varying due to the distance from the ground. ©Pixar

[56] (Figure 11). The comparison videos of multiple shots can be found in the supplemental materials. The results generated from our method, the ray test, and Bénard et al. [15] all generate 3D curves on the geometry while RenderMan's curves are generated in raster space. Our curves and Bénard et al.'s [15] are comparable in quality with slight differences depending on which frames are chosen. Generally, Bénard et al. tend to produce silhouettes with less flickering than ours. However, our method is significantly faster averaging 33 seconds per frame vs 854 seconds per frame for Figure 1.

5 Limitations

During the curve generation and cleanup process, stability in the lines can be affected by some parameters. For instance, slight variations in the scene's camera position or geometry could result in some flickering in lines of areas where the viewer perceives no motion. This flickering is a result of the visibility voting scheme. While stability is not guaranteed, due the scene changing slightly (numerically but not visibly), stability is very likely as seen with the lines on the floorboards in Figure 8. One way to guarantee the curve visibility, which we leave to future implementation, is to track the visibility probabilities over time. Another possible limitation is the number of parameters used to determine the curve visibility and filtering. In our various results, we have only used the one set of parameters as mentioned in the method section. While these parameters are available for users to tweak to achieve different results, we have found that it is rarely needed.

Although our method favors consistency across frames, flickering may still be found in some cases. For example, when computing visible contours, small gaps in lines, which the viewer assumes are continuous, can occur where different curves switch visibility. These split 2D curves can cause issues during normal enforcement in areas such as the mouth, shown in Figure 13. For some frames, the mouth is a continuous line. However for other frames, it presents small gaps that prevent the upward pointing normals from the body's silhouette from propagating to the intersection curve, which faces down (Figure 13b,c). As the character continues to turn, these curves merge and the tracked normals align with the silhouette (Figure 13d). In our results, we solve this problem by overwriting the normals of the bottom lip to always point downwards (Figure 13e,f,g,h). While this option of overwriting normals, gives artists more control of the ribbon direction in separate parts of the character, it is rarely necessary in

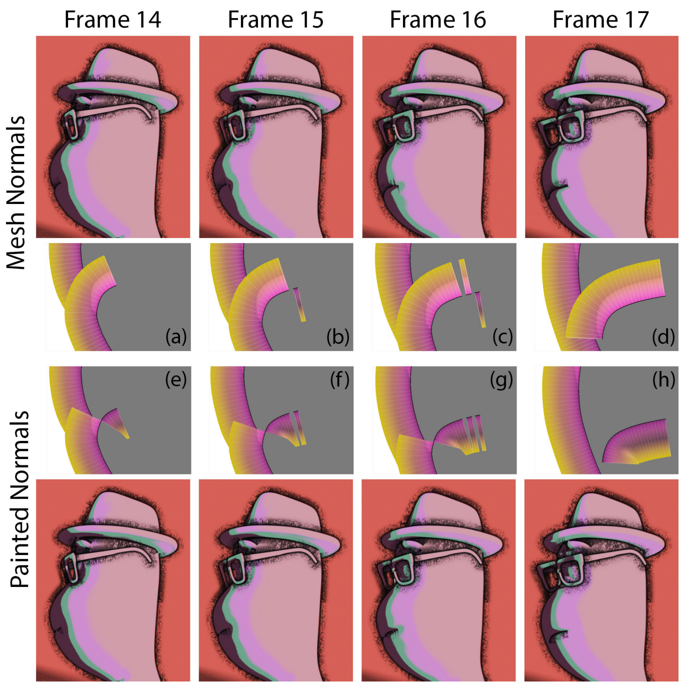

Fig. 13. **Limitation:** The mouth corner has splits in visible curves which cause our method to produce unaligned normals (top-row). One possible solution is to paint fallback values for the normals in some parts of the model, thus biasing our normal calculation (bottom-row). ©Pixar

practice. This issue also motivates the need for continuous curves before applying ribbons. In another case, when trimming ribbons, flickering can be introduced if the depth distance between two curves oscillates around the trimming threshold.

6 Conclusion

We have presented a three part method to allow the stylization of either the inside or outside of an object or shadow contour. Our algorithm determines visible contours, enforces temporally consistent normal orientation along curves, and generates trimmed ribbons. With our ribbon geometry, artists are able to achieve a variety of effects and styles that have previously not been easily accessible.

As future work, an area to explore is to integrate our stylized ribbons with other stylization methods for interior regions [7], [8] in order to give artists even more flexibility. In addition, better

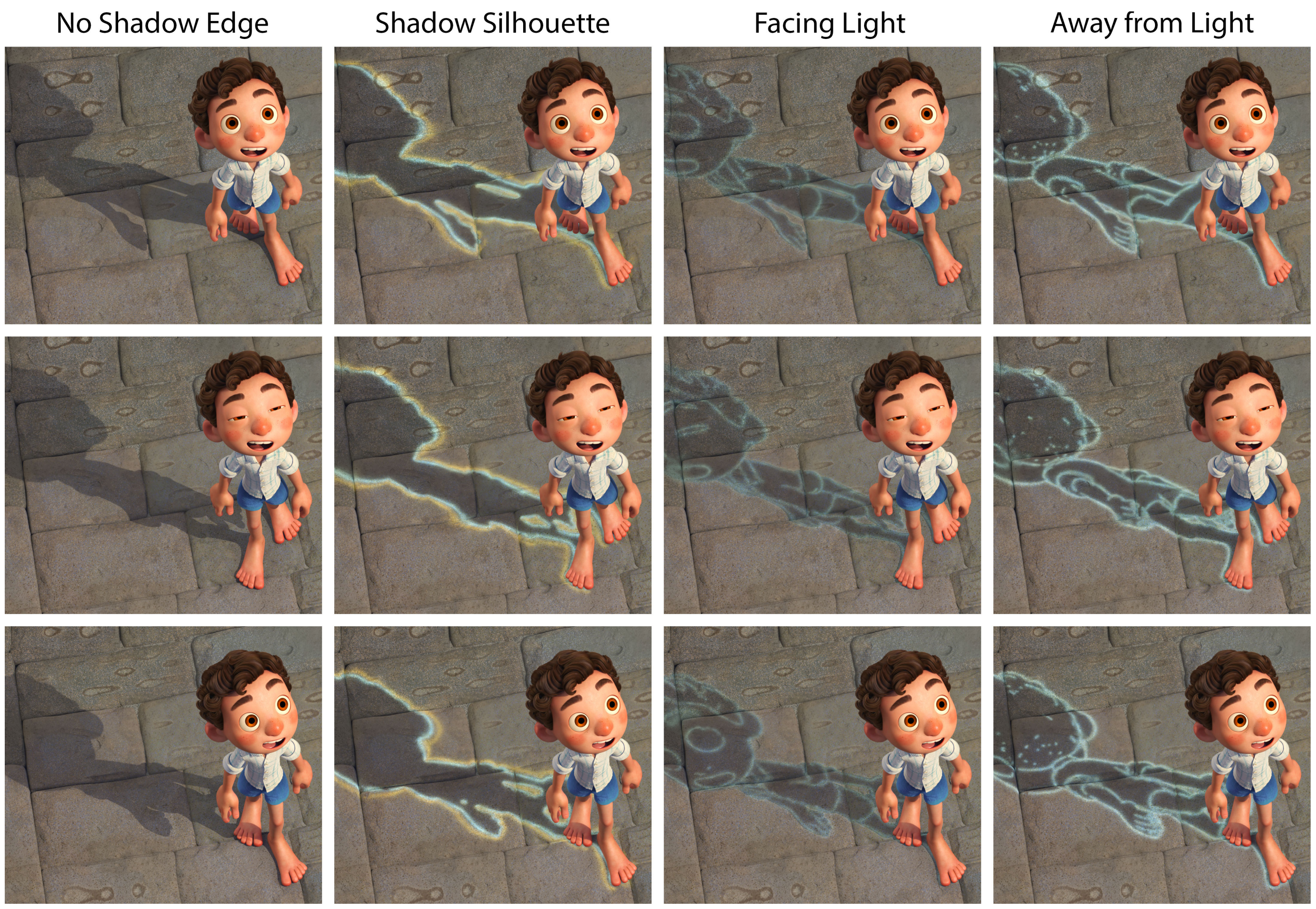

Fig. 14. **Stylized Shadow Edges:** We use our oriented ribbons to apply a variety of edge stylizations to shadows. For reference, we first show the original render without stylization (left). Along the shadow silhouette, a blue thin ribbon is applied to the inside of the curves with a thick yellow ribbon on the outside (left-center). For silhouette and intersection curves facing towards the light, a thin blue ribbon is applied only to the inside (center-right). We also stylize curves away from the light using ribbons on the outside (right). ©Pixar

controls over the placement of contour lines, especially in the facial area, is challenging but a common desire by artists, as previously demonstrated by Whited et al. [13]. Therefore, we are also interested in investigating tools that combine automatic and manual creation and editing of curve strokes. We hope this work inspires more exploration of different styles and workflows for animated productions.

7 Acknowledgment

The authors would like to thank Stacy Truman for her help creating results.

References

[1] S. Pablos and C. M. López, “Klaus,” 2019, the SPA Studios.

[2] B. Persichetti, P. Ramsey, and R. Rothman, “Spider-Man: Into the Spider-Verse,” 2018, Sony Pictures Entertainment.

[3] P. Docter and K. Powers, “Soul,” 2020, Pixar Animation Studios.

[4] D. Dimian, “Spider-Man: Into the Spider-Verse,” in *SIGGRAPH Computer Animation Festival*. ACM, 2019.

[5] P. Coleman, L. Murphy, M. Kranzler, and M. Gilbert, “Making souls: Methods and a pipeline for volumetric characters,” in *ACM SIGGRAPH Talks*, 2020.

[6] M. Zheng, A. Milliez, M. Gross, and R. W. Sumner, “Example-based brushes for coherent stylized renderings,” in *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, 2017, pp. 1–10.

[7] P. Bénard, F. Cole, M. Kass, I. Mordatch, J. Hegarty, M. S. Senn, K. Fleischer, D. Pesare, and K. Breeden, “Stylizing animation by example,” *ACM Transactions on Graphics*, vol. 32, no. 4, pp. 1–12, 2013.

[8] O. Jamriška, Šárka Sochorová, O. Texler, M. Lukáč, J. Fišer, J. Lu, E. Shechtman, and D. Sýkora, “Stylizing video by example,” *ACM Transactions on Graphics*, vol. 38, no. 4, 2019.

[9] F. Cole, K. Sanik, D. DeCarlo, A. Finkelstein, T. Funkhouser, S. Rusinkiewicz, and M. Singh, “How well do line drawings depict shape?” in *ACM SIGGRAPH*, 2009.

[10] A. Hertzmann, “Why do line drawings work? a realism hypothesis,” *Perception*, vol. 49, no. 4, pp. 439–451, 2020.

[11] J. Kahrs, “Paperman,” 2012, Walt Disney Animation Studios.

[12] P. Bénard and A. Hertzmann, “Line drawings from 3d models: A tutorial,” *Foundations and Trends® in Computer Graphics and Vision*, vol. 11, no. 1-2, pp. 1–159, 2019.

[13] B. Whited, E. Daniels, M. Kaschalk, P. Osborne, and K. Odermatt, “Computer-assisted animation of line and paint in Disney’s Paperman,” in *ACM SIGGRAPH Talks*, 2012.

[14] A. Hertzmann and D. Zorin, “Illustrating smooth surfaces,” in *ACM SIGGRAPH*, 2000, pp. 517–526.

[15] P. Bénard, A. Hertzmann, and M. Kass, “Computing smooth surface contours with accurate topology,” *ACM Transactions on Graphics*, vol. 33, no. 2, pp. 1–21, 2014.

[16] D. DeCarlo, A. Finkelstein, and S. Rusinkiewicz, “Interactive rendering of suggestive contours with temporal coherence,” in *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, 2004, pp. 15–145.

[17] D. DeCarlo and S. Rusinkiewicz, “Highlight lines for conveying shape,” in *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, 2007, p. 63–70.

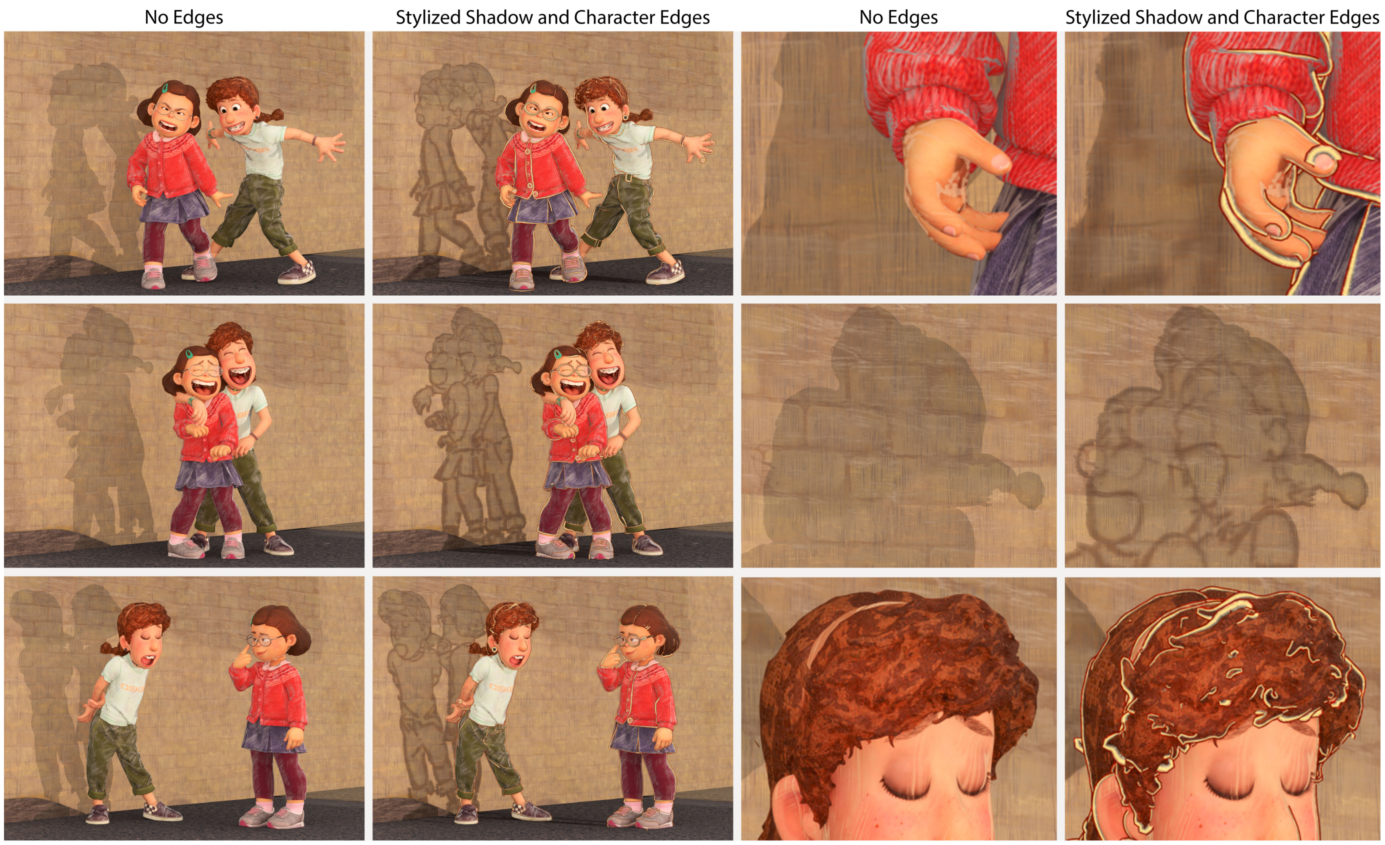
No Edges
Stylized Shadow and Character Edges
No Edges
Stylized Shadow and Character Edges

Fig. 15. **Stylized Shadow and Character Edges:** For silhouette and intersection curves facing towards the light, an inside ribbon which varies in thickness due to the distance from the ground is projected up the wall. The character ribbons use the distance from the curve to control the color banding. ©Pixar

[18] T. Judd, F. Durand, and E. H. Adelson, "Apparent ridges for line drawing," *ACM Transactions on Graphics*, vol. 26, no. 3, p. 19, 2007.

[19] W. Jiang, G. Li, Y. Nie, and C. Xian, "GPU-Driven Real-Time Mesh Contour Vectorization," in *Eurographics Symposium on Rendering*, A. Ghosh and L.-Y. Wei, Eds. The Eurographics Association, 2022.

[20] N. Ben-Zvi, J. Bento, M. Mahler, J. Hodgins, and A. Shamir, "Line-drawing video stylization," in *Computer Graphics Forum*, vol. 35, no. 6. Wiley Online Library, 2016, pp. 18–32.

[21] R. D. Kalnins, P. L. Davidson, L. Markosian, and A. Finkelstein, "Coherent stylized silhouettes," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 856–861, 2003.

[22] D. Liu, M. Nabail, A. Hertzmann, and E. Kalogerakis, "Neural contours: Learning to draw lines from 3d shapes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 5428–5436.

[23] D. Liu, M. Fisher, A. Hertzmann, and E. Kalogerakis, "Neural strokes: Stylized line drawing of 3d shapes," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 14 204–14 213.

[24] C. Liu, P. Bénard, A. Hertzmann, and S. Hoshyari, "ConTesse: Accurate occluding contours for subdivision surfaces," *ACM Trans. Graph.*, vol. 42, no. 1, Feb. 2023. [Online]. Available: https://doi.org/10.1145/3544778

[25] E. Eisemann, H. Winnemöller, J. C. Hart, and D. Salesin, "Stylized vector art from 3d models with region support," in *Proceedings of the Nineteenth Eurographics Conference on Rendering (EGRS)*, 2008, p. 1199–1207.

[26] K. Karsch and J. C. Hart, "Snaxels on a plane," in *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, 2011, pp. 35–42.

[27] S. Grabli, E. Turquin, F. Durand, and F. X. Sillion, "Programmable rendering of line drawing from 3d scenes," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 2, pp. 1–20, 2010.

[28] P. Bénard, J. Lu, F. Cole, A. Finkelstein, and J. Thollot, "Active strokes: coherent line stylization for animated 3d models," in *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, 2012, p. 37–46.

[29] B. Buchholz, N. Faraj, S. Paris, E. Eisemann, and T. Boubekeur, "Spatio-temporal analysis for parameterizing animated lines," in *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, 2011, pp. 85–92.

[30] P. Bénard, F. Cole, A. Golovinskiy, and A. Finkelstein, "Self-similar texture for coherent line stylization," in *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, 2010, pp. 91–97.

[31] J. Lengyel, E. Praun, A. Finkelstein, and H. Hoppe, "Real-time fur over arbitrary surfaces," in *Proceedings of the Symposium on Interactive 3D Graphics*, 2001, pp. 227–232.

[32] S. Hunter, "Out," 2020, Pixar Animation Studios.

[33] H. Huang, D. Li, H. Zhang, U. Ascher, and D. Cohen-Or, "Consolidation of unorganized point clouds for surface reconstruction," *ACM Transactions on Graphics*, vol. 28, no. 5, p. 1–7, 2009.

[34] G. Metzer, R. Hanocka, D. Zorin, R. Giryes, D. Panozzo, and D. Cohen-Or, "Orienting point clouds with dipole propagation," *arXiv preprint arXiv:2105.01604*, 2021.

[35] A. Hornung and L. Kobbelt, "Robust reconstruction of watertight 3d models from non-uniformly sampled point clouds without normal information," in *Proceedings of the Symposium on Geometry Processing*. Eurographics Association, 2006, p. 41–50.

[36] P. Mullen, F. de Goes, M. Desbrun, D. Cohen-Steiner, and P. Alliez, "Signing the unsigned: Robust surface reconstruction from raw pointsets," *Computer Graphics Forum*, vol. 29, no. 5, pp. 1733–1741, 2010.

[37] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[38] J. E. Lenssen, C. Osendorfer, and J. Masci, "Deep iterative surface normal estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[39] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," in *ACM SIGGRAPH*, 1992, pp. 71–78.

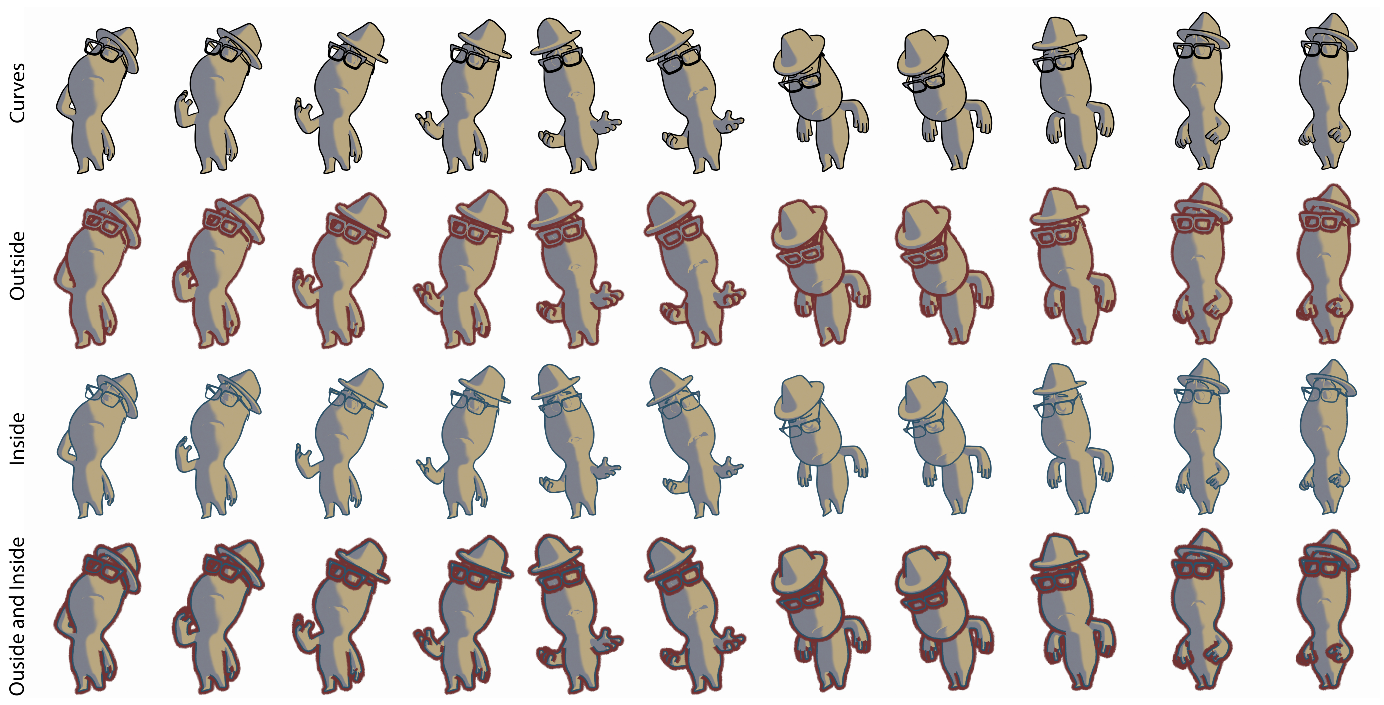

Fig. 16. **Stylized Ribbons:** Several frames of an animation stylized with curve strokes of constant width, a thick outside effect, a thinner inside effect, and a combination of outside and inside ribbons. ©Pixar

[40] P. J. Asente, "Folding avoidance in skeletal strokes," in *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, 2010, pp. 33–40.

[41] D. Nehab, "Converting stroked primitives to filled primitives," *ACM Transactions on Graphics*, vol. 39, no. 4, pp. 137–1, 2020.

[42] M. J. Kilgard, "Polar stroking: new theory and methods for stroking paths," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 145–1, 2020.

[43] C. DeCoro, F. Cole, A. Finkelstein, and S. Rusinkiewicz, "Stylized shadows," in *Proceedings of the Symposium on Non-photorealistic Animation and Rendering*, 2007, pp. 77–83.

[44] H. Todo, K.-I. Anjyo, W. Baxter, and T. Igarashi, "Locally controllable stylized shading," *ACM Transactions on Graphics*, vol. 26, no. 3, 2007.

[45] X. Liu, T.-T. Wong, and P.-A. Heng, "Closure-aware sketch simplification," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, pp. 1–10, 2015.

[46] C. Liu, E. Rosales, and A. Sheffer, "Strokeaggregator: Consolidating raw sketches into artist-intended curve drawings," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–15, 2018.

[47] E. Simo-Serra, S. Iizuka, K. Sasaki, and H. Ishikawa, "Learning to simplify: fully convolutional networks for rough sketch cleanup," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–11, 2016.

[48] J. Northrup and L. Markosian, "Artistic silhouettes: A hybrid approach," in *Proceedings of the Symposium on Non-photorealistic Animation and Rendering*, 2000, pp. 31–37.

[49] M. Bessmeltsev and J. Solomon, "Vectorization of line drawings via polyvector fields," *ACM Transactions on Graphics*, vol. 38, no. 1, pp. 1–12, 2019.

[50] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 23, no. 11, pp. 1222–1239, 2001.

[51] A. C. Müller and S. Behnke, "pystruct - learning structured prediction in Python," *Journal of Machine Learning Research*, vol. 15, no. 59, pp. 2055–2060, 2014.

[52] N. Amenta, M. Bern, and D. Eppstein, "The crust and the B–Skeleton: Combinatorial curve reconstruction," *Graph. Models Image Process.*, vol. 60, no. 2, p. 125–135, 1998.

[53] M. Postolski, M. Couprie, and M. Janaszewski, "Scale filtered Euclidean medial axis and its hierarchy," *Computer Vision and Image Understanding*, vol. 129, pp. 89–102, 2014.

[54] R. W. Sumner and J. Popovic, "Mesh data from deformation transfer for triangle meshes," 2013.

[55] Side Effects, "Houdini Engine," 2020, http://www.sidefx.com.

[56] Lollipop Shaders, "RenderMan Lollipop Shaders." [Online]. Available: http://www.lollipopshaders.com/

Nora S Willett Nora is a Research Scientist at Pixar. Her research explores creating systems and algorithms to improve artists' workflows. She interned Adobe and Autodesk research labs. Her PhD is from Princeton in 2019 and her undergraduate degree is from Stanford.

Fernando de Goes Fernando is a Principal Research Scientist at Pixar. His research centers on numerical methods for geometry processing and computational physics. He received a PhD from Caltech in 2014 supported by a Google PhD Fellowship. At Unicamp (Brazil) he earned an engineering degree and a masters.

Kurt Fleischer Kurt is a Senior Research Scientist at Pixar. He contributed significantly to Pixar's lighting and animation tools, and worked on inverse kinematics, non-photorealistic rendering, and character rigging. Kurt attended Brown (BS), Stanford (MS) and Caltech (PhD).

Mark Meyer Mark is the Director of Research at Pixar. His research interests include geometry processing, physical simulation, rendering and machine learning. He received his PhD in Computer Science from Caltech in 2004 and undergraduate degree from Northwestern.

Chris Burrows Chris is a Shading Artist at Pixar. He has been involved in developing pattern and illumination techniques for films including "Luca", "Finding Dory" and "Wall•E". He graduated with a BS in Fine Art from Skidmore College.

		How value change effects the result	
Name	Value	Decreasing	Increasing
Computing Visible Contours			
Cylindrical samples	20	Not enough accuracy	Increased time
Cylindrical radius	0.3 pixels	Higher chance of gaps in long silhouette edges	Samples in areas of mesh that are not topologically adjacent
Angle splitting	100 degrees	Not splitting often enough if a turn in the curve is not very clean due to polyline sampling	Splitting too much on polylines of sharp curvature
Visible contour probability cutoff	0.8	Too many points visible	Some segments missing or holes in the silhouette
Clean up: screen space merging distance	0.15 pixels	Too many lines close together causing clutter	Can merge important features like a hat and brow line
Clean up: depth merging distance	0.15 world space camera depth	Too many lines close together	Can merge important features like a hat and brow line
Clean up: colinear merging	30 degrees	Misses merging some switch-backs along curves	Cuts off ends as a curve turns around
Clean up: loop deletion area	0.4 percent of image resolution	Leaves messing lines in complicated areas such as eye corners that cause flickering when expanded to ribbons	Can remove important features like fingernails and buttons
Clean up: short edge deletion	2 pixels	Jagged edges on the ends of lines	Since points are already sampled 1 pixel apart, increasing this value will not change the result
Normal Orientation Enforcement			
Cylindrical samples	20	Not enough accuracy	Increased time
Cylindrical radius	0.25 pixels	Not enough variety in normal direction to determine if inside a crevice	Samples from areas of mesh that are not topologically adjacent
Threshold for normal probability	0.9 percent	Push orientation too much in one direction instead of prioritizing temporal coherence	Mislabel some silhouette points as having unstable normals
Ribbon Trimming			
Curve sampling resolution	1 pixel	Increases time complexity	Not enough resolution along curves
Voronoi curve sampling resolution	0.1 pixel	Increases time complexity without improving medial axis	Too many medial axis lines causing unnecessary trimming
Width edge trimming depth	0.2 world space	Not trim ribbon enough at sharp turns in the curve causing ribbon faces to overlap	Trim too much at T intersections creating a V in the ribbon when depth ordering would have fixed the overlap
T junction trimming range	3 points	Sharp peaks at T intersections can appear causing noticeable flickering	Trims unnecessary points along the curve

TABLE 1
Parameters for each step of our method