Real-Time Rendering of Stereo-Consistent Contours

Dejing He Rui Wang* Hujun Bao

State Key Lab of CAD&CG, Zhejiang University

ABSTRACT

Line drawing is an important and concise method to depict the shape of an object. Stereo line drawing, a combination of line drawing and stereo rendering, not only efficiently conveys shape but also provides users with a visual experience of a stereoscopic 3D world. Contours are the most important lines to draw. However, contours must be rendered consistently for two eyes because of their view-dependent nature; otherwise, they cause binocular rivalry and viewing discomfort. This paper proposes a novel solution to draw stereo-consistent contours in real time. First, we extend the concept of epipolar-slidability and derive a new criterion to check epipolar-slidability by the monotonicity of the trajectory of the viewpoints of contour points. Then, we design an algorithm to test the epipolar-slidability of contours by conducting an image space search rather than sampling multiple viewpoints. Results show that the proposed method has a much lower cost than that of previous works, therefore enables the real-time rendering and editing of stereo-consistent contours for users, such as changing camera viewpoints, editing object geometry, tweaking parameters to show contours with different details, etc.

Keywords: Stereo contour rendering, binocular rivalry, stereo consistency, epipolar-slidability, real-time rendering.

Index Terms: Computing methodologies—Computer graphics—Rendering;

1 INTRODUCTION

As the simplest form of shape, outline is one of the most striking features of an object. A number of algorithms have been developed to generate lines from 3D models automatically and render them interactively. This process is known as line drawing, a non-photorealistic rendering (NPR) technique. Two types of lines, namely, view-independent and view-dependent lines, are commonly used to convey the shapes of objects. View-independent lines, such as creases, are defined solely on the basis of the geometry of an object, i.e., they are defined statically on the object itself. By contrast, view-dependent lines, such as contours and suggestive contours, are dynamic features which are defined by an object's geometry and viewpoint.

Stereo rendering is a method used to stimulate the perception of depth from two eyes. It has been widely used in applications, such as stereoscopic 3D imaging and virtual reality. Stereo line drawing provides a unique visual experience for users to perceive outlines of objects in a 3D world by combining stereo rendering and line drawing. However, while stereo rendering produces a pair of images with different viewpoints for eyes, stereo line drawing may induce stereoscopic artifacts, such as binocular rivalry. This is mainly because view-dependent lines generated for each eye may be inconsistent in two different views. The stereo consistency of lines must be ensured to avoid binocular rivalry.

Kim et al. [11] addressed the problem of stereo line drawing by erasing contour segments that do not have a fusible counterpart in the other view. The key idea of their work is to examine the *epipolar-slidability* of one contour by rendering multiple images at viewpoints between two eyes and check the continuity of the contour among these images. However, it requires a large number of viewpoints to reduce the matching error for complex surfaces. As such, this method has a long rendering time that makes real-time stereo line drawing infeasible. While Kim et al. [11] solved the problem in image space, Bukenberger et al. [2] proposed another solution to draw stereo-consistent contours in object space. However, their approach relies on the simulation of contours observed from an arbitrary camera path in a precomputation process, which is not flexible for real-time applications.

In this paper, we present a new real-time solution for efficiently and flexibly drawing stereo-consistent contours. On the basis of Kim et al.'s work [11], we design an image space search algorithm to examine the *epipolar-slidabilities* of contour points. Technically, we observe that for each surface point along the epipolar curve, there exists one corresponding viewpoint lying at the baseline of the two eyes and seeing the surface point as a contour point. Then, we can convert the test of the *epipolar-slidabilities*, i.e., the stereo continuities of contour points, to checking the trajectory of these corresponding viewpoints of contour points. When the trajectory monotonously moves from the left eye to the right eye or vice versa, these contour points are *epipolar-slidable*. In this way, we only need to check some extreme points on the trajectory to guarantee the monotonicity. This can be performed in image space rather than sampling multiple viewpoints [11], thereby saving a lot of evaluation time. Additionally, to avoid missing matches caused by overlaps of contour points and extreme points, we use the per-pixel linked list on GPU [18] to keep multiple contour points and extreme points that may lie in one pixel while searching in image space. Results show that the proposed method has a much lower cost than that of previous works, therefore enables real-time rendering and editing of stereo-consistent contours for users, such as changing camera viewpoints, editing object geometry, tweaking parameters to show contours with different details, etc.

The main contributions of our work are summarized as follows:

- A mathematical derivation of epipolar-slidability that the stereo continuities of contour points along the epipolar curve can be evaluated by extreme points on the trajectory of the corresponding viewpoints of contour points.
- An image space search algorithm that can be utilized to check the epipolar-slidabilities of contours in the stereo line drawing.
- A real-time stereo line drawing method that enables stereo drawing of contours, suggestive contours, and stylized contours at a real-time rendering rate.

2 RELATED WORKS

Line drawing: To simulate traditional media, such as pen-and-ink or technical illustration, researchers developed various algorithms to draw lines from 3D models [4, 16]. In the field of line drawing, two types of lines are usually used: view-dependent lines and view-independent lines. Among view-dependent lines, contours [9], in which two neighboring triangles face in different directions from a viewpoint, are one of the most important lines to draw. Suggestive

*e-mail: rwang@cad.zju.edu.cn

2019 IEEE Conference on Virtual Reality and
3D User Interfaces
23-27 March, Osaka, Japan
978-1-7281-1377-7/19/$31.00 ©2019 IEEE

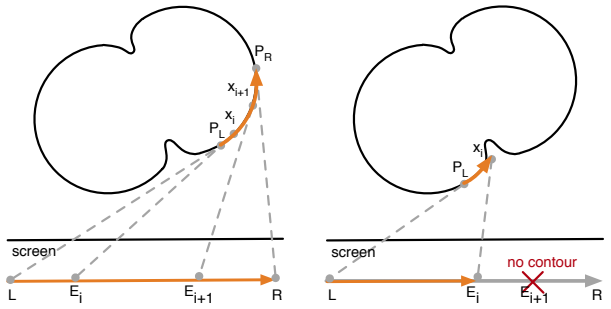


Figure 1: Epipolar-slidability. (a) P_L is epipolar-slidable. (b) P_L is not epipolar-slidable. (Figure is modeled after Figure 6 in Kim et al.'s work [11].)

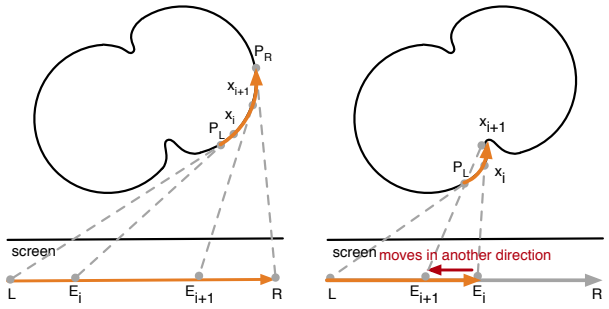


Figure 2: Corollary of epipolar-slidability. (a)P_L is epipolar-slidable. (b)P_L is not epipolar-slidable

contours [6], one type of contours that are not only for one viewpoint but also for nearby viewpoints, are another type of view-dependent lines that anticipate and extend contours. View-independent lines, such as creases [15], also reveal important features of a geometrical shape, but they are not of interest in our work because they are natively stereo-consistent.

Stereo-consistent NPR: Line drawing is one of non-photorealistic rendering (NPR) techniques. Studies on NPR have focused on combining stereo rendering to produce stereo-consistent NPR effects. To avoid artifacts in stereo rendering introduced by stroke-based rendering algorithms [8], Northam et al. [12, 13] presented algorithms that decompose the left and right views into discretized disparity layers and merge the corresponding layers into one layer, where the stylization of strokes takes place. In addition to image stylization techniques, stereo-consistent line drawing from 3D models has been investigated. Kim et al. [11] described the concept of stereo-consistent lines and proposed a method to establish the stereo coherency of lines by checking the fusible counterpart along the epipolar curve at multiple viewpoints. Bukenberger et al. [2] proposed a novel solution for stereo-consistent contours that interpolates contours in object space between different view positions. The stylization of lines [10, 14] is also of great concerns. Kim et al. [11] and Bukenberger et al. [2] both introduced how to stylize stereo-consistent contours in their works. Kim et al. [11] conducted stylization by propagating parameters between views via the correspondence between stereo-consistent contour pairs. In Bukenberger et al.'s work [2], stylization is based on the properties of an object's 3D shape for temporal coherency. In this paper, we initially discuss how to ensure the stereo-consistent rendering of contours and subsequently address the drawing of suggestive contours and the stylization of stereo-consistent contours.

Per-pixel linked list: Linked lists are a common data structure in computer science. They are used in various CPU algorithms, but non-trivial to implement on GPU. Yang et al. [18] introduced a fast method to construct linked lists on GPU. They used one buffer to store all linked list nodes and another buffer to store head pointers per pixel. Per-pixel linked list is the most general method to handle multiple fragments per pixel. To avoid problems that arise from overlaps, we use per-pixel linked lists to access data of multiple fragments at the same pixel. We only build linked lists on contour points or extreme points. As such, the memory consumption of per-pixel linked lists is considerably low.

3 Mathematical Formulation

In this section, we briefly describe the concept of epipolar-slidability introduced by Kim et al. [11] and present our derivation for the epipolar-slidability.

3.1 Epipolar-Slidability

Epipolar-slidability is defined as follows: let L and R denote left and right eyes, respectively, whereas P_L and P_R are the corresponding contour points (Fig. 1). While the viewpoint E moves from L to R, the corresponding contour point should move from P_L to P_R along a curve called *epipolar curve* [7]. If the contour points on the epipolar curve continues from P_L to P_R, P_L is *epipolar-slidable* (Fig. 1(a)). Otherwise, P_L is not epipolar-slidable and no stereo fusion is possible (Fig. 1(b)), indicating that P_L should not be drawn in stereo rendering.

Kim et al. [11] presented a method to examine the epipolar-slidability of P_L. Specifically, their method initially inserts multiple viewpoints between two eyes and subsequently renders the contours of the model for each viewpoint. When all adjacent contour points, for example x_i and x_{i+1} in Fig. 1(a), are within a certain vicinity, P_L is epipolar-slidable.

3.2 Corollary of Epipolar-Slidability

In our approach, we extend the idea of epipolar-slidability and provide a corollary by checking the trajectory of the corresponding viewpoints of contour points from P_L to P_R instead of evaluating at multiple viewpoints between L to R [11].

Our approach is based on the observation shown in Fig. 2. Suppose that a contour point x moves along the epipolar curve from P_L to P_R. If P_L is epipolar-slidable, the corresponding viewpoint for x moves monotonously from L to R (Fig. 2a). If P_L is not epipolar-slidable, the movement of the corresponding viewpoint E is not be always incremental before it reaches R (Fig. 2b).

Based on this observation, the attempt to examine the epipolar-slidability can be technically converted to examining the monotonicity of the trajectory of viewpoints. Theoretically, as long as the contour point x moves along the epipolar curve, the corresponding viewpoint E seeing it as a contour point can be computed as follows:

$$(E-P)\cdot N = 0 \tag{1}$$

where P and N denote the position and normal of the contour point x, respectively.

Considering that E is on the baseline between L and R, we can parameterize it by a variable t as follows:

$$E = (1-t)L + tR \tag{2}$$

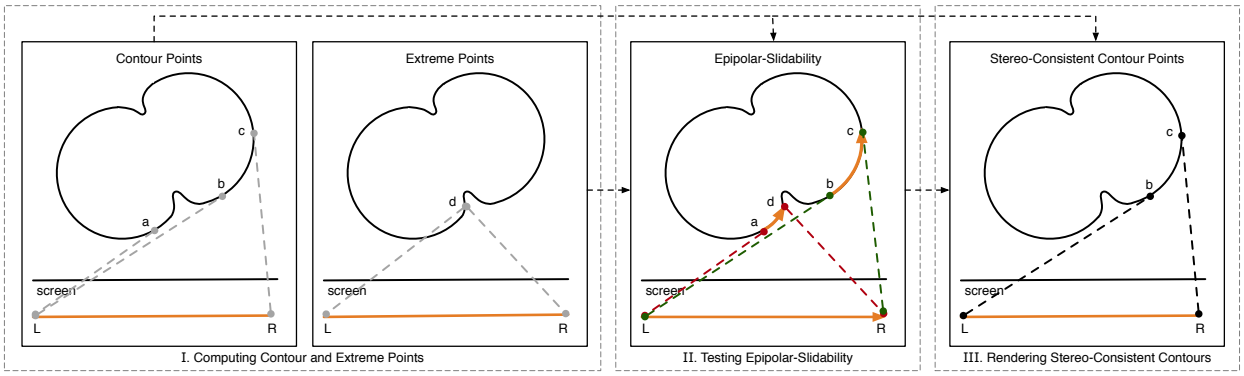


Figure 3: Illustration of our algorithm. For simplicity, we only show the process to compute view-dependent contours from L to R. First, contour points a, b, and c and an extreme point d are computed. Second, our algorithm tests the epipolar-slidability of a and b in image space, where a fails in the test by encountering the extreme point d, and b passes the test by finding the corresponding contour point c from the right eye. Finally, these stereo-consistent contour points b and c are rendered.

From Equation 1 and Equation 2 we have:

$$t = \frac{(P-L)\cdot N}{(R-L)\cdot N} \tag{3}$$

t is a function of surface point x. While x moves along the epipolar curve, $t(x)$ is the trajectory function of the corresponding viewpoint of the contour point x. To check the monotonicity of the trajectory function $t(x)$, we compute the derivative of $t(x)$ as follows:

$$t' = \frac{(P'\cdot N + (P-L)\cdot N')((R-L)\cdot N) - (R-L)\cdot N')((P-L)\cdot N)}{((R-L)\cdot N)^2} \tag{4}$$

where we omit x at both sides of the equation for simplicity. For each extreme point, when $t'(x) = 0$, the monotonicity of the trajectory may break. We then derive a simpler form of $t'(x)$ as

$$t'(x) = \frac{C(x)|P(x)-L||R-L|\sin\theta}{((R-L)\cdot N(x))^2} \tag{5}$$

where $C(x)$ is the curvature on the epipolar curve and θ is the angle between $P-L$ and $R-L$. A full derivation is provided in the Supplemental Material. Note that $t'(x) = 0$ may not be continuous because of the discrete representation of 3D models. Therefore, we compute extreme points where $t'(x-)t'(x+) < 0$ instead of computing $t'(x) = 0$.

We project each contour point on the epipolar curve onto the left and right image and search along the epipolar line instead of the epipolar curve. In this way, we can compute and store extreme points on images and realize the test of epipolar-slidability in image space.

4 Algorithm

Based on our derivation, the epipolar-slidability can be examined in image space by checking the extreme points of the trajectory function $t(x)$. Accordingly, we design an algorithm to enable real-time stereo line drawing with runtime epipolar-slidability tests. In this section, we summarize the overall pipeline of the algorithm, provide more details for each stage, and extend our solution to draw suggestive contours and stylized lines.

An illustration of our algorithm is shown in Fig. 3. It mainly involves three steps:

I. Computing Contour and Extreme Points: For each frame, we start with rendering basic contours and extreme points into per-pixel linked lists in one rendering pass.

II. Testing Epipolar-Slidability: We apply image space searches for contours computed in the previous step to test their epipolar-slidabilities.

III. Rendering Stereo-Consistent Contours: We render all epipolar-slidable contours onto stereo images.

On the basis of the algorithm, we design a real-time rendering system, which is shown in Fig. 4.

4.1 Computing Contour and Extreme Points

At the first stage, we rasterize the model and compute contours and extreme points. We largely follow the established rendering procedure outlined by Kalinins et al. [17] to find all visible contours for each eye. Generally, we compute all corresponding surface points of pixels where $N\cdot V = 0$ in a rendering pass. These pixels are contour pixels in image and surface points are contour points.

To compute extreme points, where $t'(x-)t'(x+) < 0$, we observe that all of the terms, except $C(x)$ in Equation 5, are positive (Supplementary Document). As such, we compute extreme points, where $C(x-)C(x+) < 0$, instead of computing on $t'(x)$. Note that $C(x)$ is the curvature at x, which can be computed at each triangle in a manner similar to determining contour points, where $N\cdot V = 0$. However, we design a more sophisticated algorithm to compute extreme points because of two following reasons. First, after 3D models are rasterized, normals of fragments are interpolated from vertex normals using barycentric coordinates. Accordingly, if an epipolar curve is projected to a triangle, the normal varies linearly along the projected line, indicating that $C(x)$ is a constant value inside a triangle. Therefore, extreme points only occur at edges of adjacent triangles. Second, all extreme points, including those occluded at one view, should be identified because they may be visible at the other view. Therefore, we are unable to directly use the shader derivative functions supported by the modern graphics APIs to compute $C(x-)$ and $C(x+)$ because these functions are operable for visible pixels only.

Our algorithm for computing extreme points is illustrated in Fig. 5, where $\triangle ABC$ and $\triangle ACD$ are two adjacent triangles. We use a geometry shader to emit the edge AC and pass the vertex positions and normals of two adjacent triangles to the next shader stage. Then, in the fragment shader, we shoot two rays from camera towards the points slightly jittered from the point x, i.e., to obtain two neighboring points $x+\Delta x$ and $x-\Delta x$ around x. Given the surface normals of x, $x+\Delta x$ and $x-\Delta x$, we can compute $C(x-)$ and $C(x+)$ and determine whether the extreme point exists. Extreme points are separately computed and stored for two eyes.

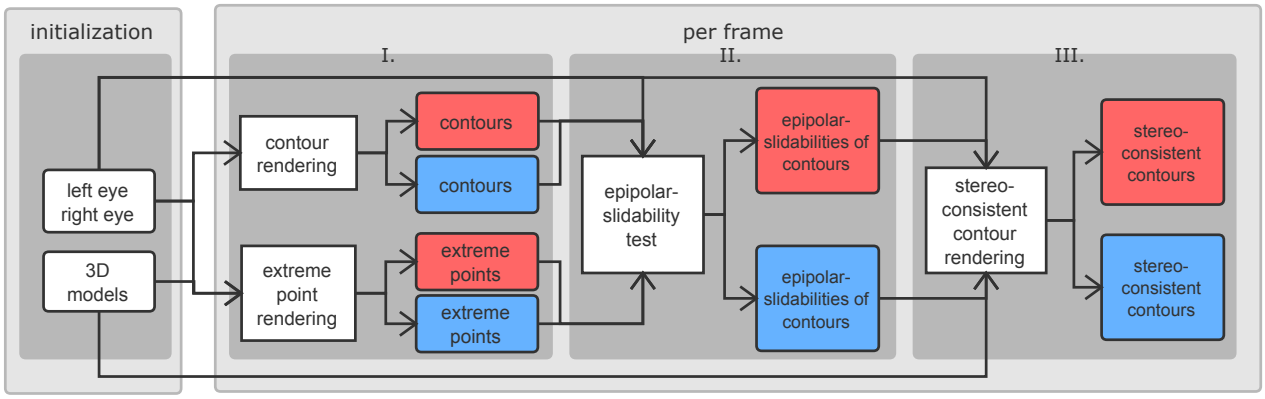
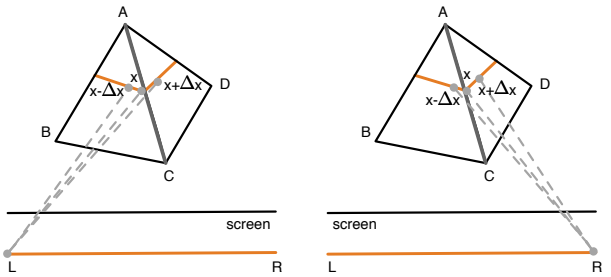


Figure 4: System overview. Boxes in blue/red stand for the instances of the left/right eye. 3D models are loaded, and cameras for two eyes are set up at initialization. For each frame, contour points and extreme points are initially rendered into per-pixel linked lists. Then, epipolar-slidability tests are performed on these linked lists. Finally, stereo-consistent contours are rendered based on epipolar-slidabilities obtained at the former stage.



(a) left eye (b) right eye

Figure 5: Computation of extreme points. $\triangle ABC$ and $\triangle ACD$ are two adjacent triangles. We perform extreme point test for each fragment of edge AC. (a) compute extreme points for left eye. (b) compute extreme points for right eye.

Some overlaps may exist among contour points and extreme points, i.e., multiple contour points and extreme points may lie in one pixel; as such, we use the per-pixel linked lists [18] to keep all of them. Given that contour points and extreme points occupy a small proportion of the image space, the memory consumption of per-pixel linked lists is considerably low.

4.2 Testing Epipolar-Slidability

We use two separated full screen post processes to examine the epipolar-slidabilities of contours at two views (left eye and right eye). Our search algorithm to test epipolar-slidability mainly involves three steps:

1. Reproject the contour point from the current view to the other view.

2. Determine the search direction (left or right) based on which side the contour point's surface normal points to.

3. Search in image space pixel by pixel. For each pixel, we iterate and check each point stored in the per-pixel linked list. The search process stops when we reach the pixel having the corresponding contour point of the other view or when we find that the monotonicity of the trajectory function $t(x)$ breaks at some extreme points.

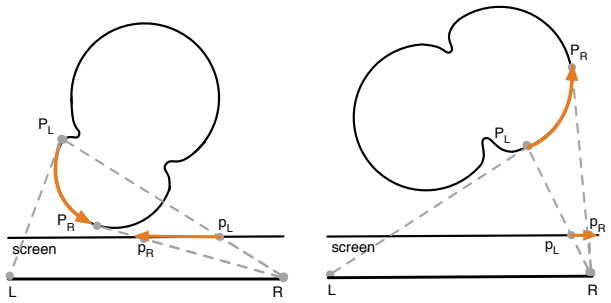
Fig. 6 illustrates two search processes that reach the corresponding contour point from right to left and vice versa. Fig. 7 shows two search processes that stop at an extreme point. We now take the search process from L to R as an example to explain the details of these steps. The process of searching from R to L is similar.

Search direction: The search direction is determined by which side the contour point's surface normal points to. Formally, if the cross product of the view ray and the surface normal, i.e., $cross(N,V)$, points to the upward direction of the epipolar plane, then the contour point's surface normal points to the left. In this way, we search from the reprojected point towards the left. Otherwise, we search towards the right.

False matches exclusion: False matches may occur when an epipolar curve is occluded by other objects. Fig. 8 shows such a case, where P^*_R is a contour point from another object, and p^*_R is its projection in image space. In this case, the search process stops at p^*_R rather than at p_R. As a result, contour points that should have been erased may be retained in the final result. Likewise, contour points that should have been kept may be falsely erased. To exclude these false matches, we store the mesh ID of each contour point and extreme point in per-pixel linked lists. With these IDs, we can ensure that the search processes can be stopped at contour points or extreme points from the same mesh.

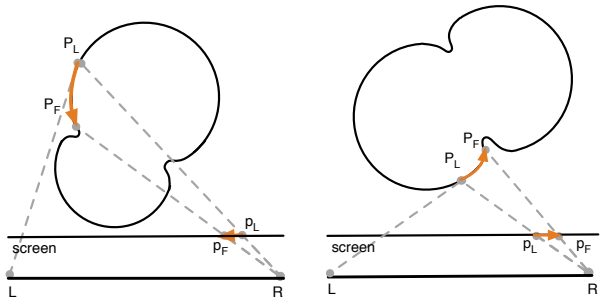
4.3 Stereo-Consistent Contour Rendering

After the epipolar-slidability test stage, we can obtain all epipolar-slidable contours. Then, we launch another rendering pass to draw all contours of the 3D models with the desired line width and remove non-epipolar-slidable contours. Additionally, on the basis of the spine test method from a previous work on line visibility [3], we apply a similar strategy that uses multiple epipolar-slidable probes along the tangent direction of the contour to reduce artifacts, such as aliasing and broken lines. The contour point with the closest depth is selected as a sample if multiple contour points are in the per-pixel linked list at one pixel.



(a) search to the left (b) search to the right

Figure 6: Search processes that reach corresponding contour point. p_L and p_R are projections of P_L and P_R, respectively.



(a) search to the left (b) search to the right

Figure 7: Search processes that stop at an extreme point. P_F is an extreme point. p_L and p_F are projections of P_L and P_F, respectively.

4.4 Suggestive Contours

Suggestive contours are points where the radial curvature is zero when viewed from the convex side [6], so they are considered as view-dependent lines. However, unlike contours, suggestive contours are partly view-independent because the radial curvatures of a point for two eyes can both be zero. Therefore, we erase suggestive contours that are not view-independent to ensure their stereo consistency. In this way, some suggestive contours that are stereo-consistent but not view-independent may be erased. In our experiments, we find that such a loss is insignificant, and view-independent portions are sufficient to convey the features of suggestive contours.

Specifically, suggestive contours are rendered into per-pixel linked lists, along with the contours at stage I. A tag is written in the linked list node to indicate which type of view-dependent line it stores. Then, at stage II, we reproject the suggestive contour points to another view and determine their consistency by checking the per-pixel linked lists without searching the entire epipolar curve like contours. Considering that suggestive contours visually anticipate and extend contours, we allow suggestive contours to be matched with contours, and vice versa.

4.5 Stylized Contours

Kim et al. [11] conducted stylization by propagating style parameters from one view to another, whereas Bukenberger et al. [2] extracted the stylization features in their object space solution. Our method also supports the stylized rendering of contours. Specifically, we apply a heuristically combined solution that propagates texture coordinates in image space and determines other parameters in object shape to draw stylized contours. Texture coordinates are propagated by contour point matching at stage II, and the properties of object shape are extracted and applied at stage III.

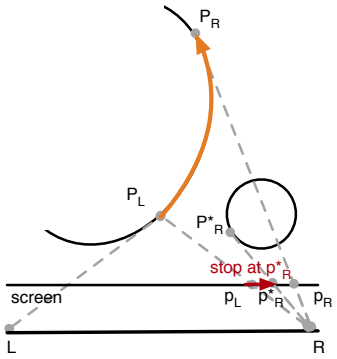


Figure 8: A false match caused by occlusion. We exclude such a case by mesh IDs.

5 Results

5.1 Quality

To evaluate our method, we compare stereo-consistent anaglyphs produced by our algorithm with those rendered individually for two eyes (Fig. 9). Three zoomed regions are highlighted for detailed comparisons. Our method effectively erases portions of contours and suggestive contours that are not stereo-consistent.

To further demonstrate the effectiveness of our algorithm, we show the comparisons of our results and those from previous works [2, 11] in Fig. 10. However, we do not know the exact parameters that they used in their methods. As such, we manually tweaked the parameters to approximate their results. Although results are different in some details, the qualitative comparisons reveal that our real-time stereo contour rendering algorithm achieves a similar stereo consistency.

Although our approach operates solely in image space, the hidden lines are matched correctly without any special treatment because the occluded contour points are stored in per-pixel linked lists. Fig. 11 shows a result that is similar to the one shown in a previous study [2] but is generated with our method.

Our method does not rely on any precomputation, so it supports dynamic scenes with changing viewpoints and real-time tweaked parameters. A supplemental video is provided to demonstrate these advantages of our method. Note that the parameter called “sc threshold” we tweaked in the video is a threshold to control the amount of suggestive contours.

5.2 Performance

We implement our stereo-consistent contour rendering system using OpenGL 4.6 on a PC with an Intel Xeon E3 CPU and an NVIDIA GeForce GTX 960 graphics card. We render the results at a resolution of 1024×768. Table 1 presents the rendering performance for Fig. 9 and Fig. 10. These timings are reported by disabling the stylization rendering. By contrast, the system implemented by Kim et al. [11] runs at 3 FPS with 30,000 vertices using GPU. Bukenberger et al. [2] implemented a more efficient system that reaches 24 FPS for meshes with 20,000 faces on CPU, but the GPU version is not mentioned in their paper. However, such a performance is achieved without correctly considering the view-dependent occlusions. The view graph algorithms of their approach take about 0.25 s for the contours of a small mesh, such as Utah Teapot (2464 faces), and about 5 s for a large mesh, such as Stanford Bunny, to establish correct view-dependent occlusions. In comparison with previous works, our method is faster and offers correct view-dependent contour rendering.

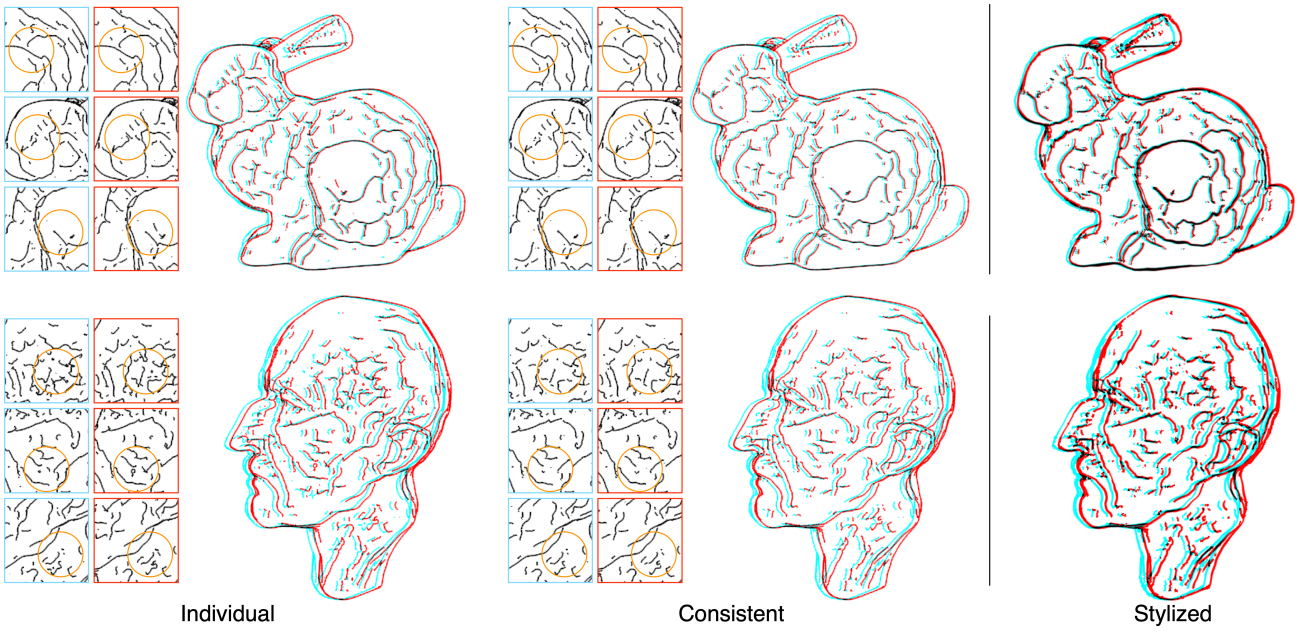


Figure 9: Comparisons of anaglyphs generated with a naive method (individual) and our method (consistent). 3D red (left)-cyan (right) glasses are recommended to view these anaglyphs correctly. Stylized anaglyphs based on consistent results are shown on the right. Models are from DeCarlo et al's gallery [5].

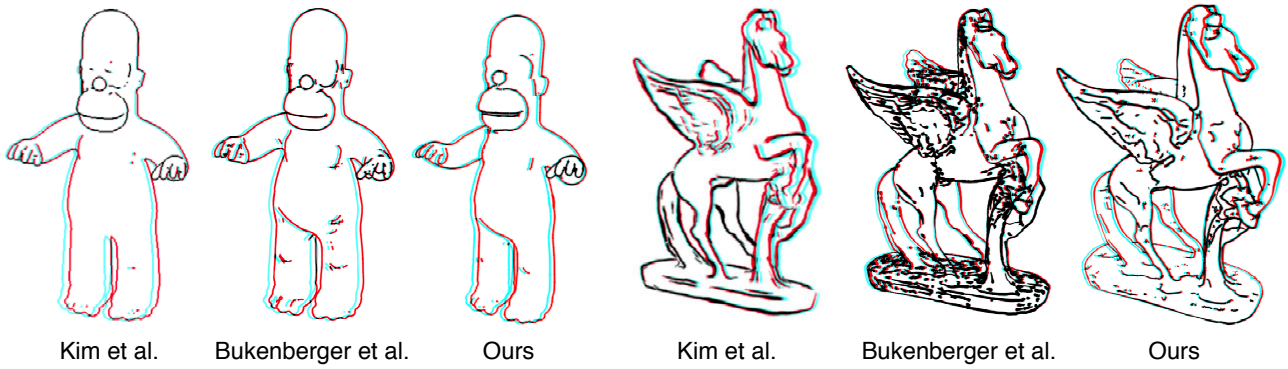


Figure 10: Comparisons of our results with previous results [2, 11]. Parameters were tweaked manually to approximate the results, but the results still differ in some details because of differences in the models and the implementation of contour rendering. The figure of Pegasus is stylized with stroke width scaled by z-depth. Models are provided courtesy of IMATI and CNR by the AIM@SHAPE-VISIONAIR Shape Repository [1].

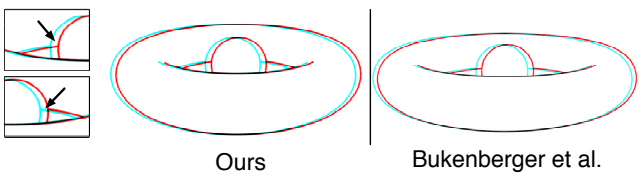


Figure 11: Matching with hidden lines. Boxes and arrows are used to point out the hidden lines. A similar result from another study [2] is provided on the right.

Table 1: Statistics of example scenes. From left to right: scene, number of vertices, number of faces, and the performance of stage I, II, III and all. Timings are recorded with the stylization rendering off.

Scene	Verts.	Faces	Performance (ms / FPS)			
			I	II	III	Total
Bunny (Fig. 9)	35947	69451	6.1	5.6	2.9	14.7/67.7
Max Planck (Fig. 9)	49132	98260	5.8	5.7	2.0	14.3/69.8
Homer (Fig. 10)	5103	10202	1.2	5.1	0.3	6.7/148.2
Pegasus (Fig. 10)	63544	127095	12.0	4.9	6.0	24.0/41.1

5.3 Limitations and Future Work

Although our method can render stereo-consistent contours in real time, it has some limitations. First, using mesh IDs to exclude false

matches may be unreliable when false matches are from the same mesh. A more reliable solution to exclude wrong matches should be developed in future works. Second, the temporal coherency of contours has not been considered yet. Therefore, contours may flicker in the video, especially when the stylization rendering is on. This is because that there may exist conflicts between stereo consistency and temporal coherency in terms of propagating style parameters or directly stylizing based on the properties of an object shape. An appropriate solution that considers the temporal coherency of contours worths exploring in the future.

6 Conclusion

We present a real-time rendering technique to draw stereo-consistent contours. Our basic idea is to examine contour continuity along an epipolar curve by conducting an image space search instead of sampling multiple viewpoints in a previous work. Specifically, we extend the concept of epipolar-slidability and derive a new criterion to check epipolar-slidability by the monotonicity of the trajectory of the viewpoints of contour points. On the basis of this derivation, we propose a multi-stage rendering algorithm that initially computes contours and extreme points of the trajectory function, and subsequently tests the epipolar-slidabilities of contours in image space. Our algorithm also supports for suggestive contours and stylized line drawing. Experiments demonstrate that our technique can erase portions of view-dependent lines that are not stereo-consistent while preserving high-quality stereo-consistent ones. The whole algorithm is GPU friendly and has been implemented using shaders. Since everything is computed from scratch in each frame, our approach is free of precomputation and allows users to manipulate objects interactively and tweak the parameters of contours as desired.

Acknowledgments

We would like to thank all reviewers for their insightful comments. This research was partially funded by National Key R&D Program of China (No. 2017YFB1002605), NSFC (No. 61872319) and Zhejiang Provincial NSFC (No. LR18F020002).

References

[1] Inria: Models from visionair shape repository. http://visionair.ge.imati.cnr.it/ontologies/shapes/, 2004. Accessed: 1-October-2018.

[2] D. R. Bukenberger, K. Schwarz, and H. P. Lensch. Stereo-consistent contours in object space. In *Computer Graphics Forum*, vol. 37, pp. 301–312. Wiley Online Library, 2018.

[3] F. Cole and A. Finkelstein. Two fast methods for high-quality line visibility. *IEEE transactions on visualization and computer graphics*, 16(5):707–717, 2010.

[4] F. Cole, A. Golovinskiy, A. Limpaecher, H. S. Barros, A. Finkelstein, T. Funkhouser, and S. Rusinkiewicz. Where do people draw lines? In *ACM Transactions on Graphics (TOG)*, vol. 27, p. 88. ACM, 2008.

[5] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella. Suggestive contour gallery. http://gfx.cs.princeton.edu/proj/sugcon/models/, 2003. Accessed: 1-October-2018.

[6] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella. Suggestive contours for conveying shape. *ACM Transactions on Graphics (TOG)*, 22(3):848–855, 2003.

[7] D. Geiger, B. Ladendorf, and A. Yuille. Occlusions and binocular stereo. *International Journal of Computer Vision*, 14(3):211–226, 1995.

[8] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 453–460. ACM, 1998.

[9] A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 517–526. ACM Press/Addison-Wesley Publishing Co., 2000.

[10] R. D. Kalnins, P. L. Davidson, L. Markosian, and A. Finkelstein. Coherent stylized silhouettes. In *ACM Transactions on Graphics (TOG)*, vol. 22, pp. 856–861. ACM, 2003.

[11] Y. Kim, Y. Lee, H. Kang, and S. Lee. Stereoscopic 3d line drawing. *ACM Transactions on Graphics (TOG)*, 32(4):57, 2013.

[12] L. Northam, P. Asente, and C. S. Kaplan. Consistent stylization and painterly rendering of stereoscopic 3d images. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, pp. 47–56. Eurographics Association, 2012.

[13] L. Northam, P. Asente, and C. S. Kaplan. Stereoscopic 3d image stylization. *Computers & Graphics*, 37(5):389–402, 2013.

[14] J. Northrup and L. Markosian. Artistic silhouettes: A hybrid approach. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pp. 31–37. ACM, 2000.

[15] R. Raskar. Hardware support for non-photorealistic rendering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pp. 41–47. ACM, 2001.

[16] S. Rusinkiewicz, F. Cole, D. DeCarlo, and A. Finkelstein. Line drawings from 3d models. In *ACM SIGGRAPH 2008 classes*, p. 39. ACM, 2008.

[17] N. WYSIWYG. Drawing strokes directly on 3d models. *ACM Trans. on Graphics*, 2002.

[18] J. C. Yang, J. Hensley, H. Grün, and N. Thibieroz. Real-time concurrent linked list construction on the gpu. In *Computer Graphics Forum*, vol. 29, pp. 1297–1304. Wiley Online Library, 2010.