Coherent 3D Line Drawings for Stylized Animation

Xu Xiang

School of Computer Engineering

A Thesis Submitted to the Nanyang Technological University
in Partial Fulfillment of the Requirement for the Degree of
Doctor of Philosophy

2016

Acknowledgments

I wish to render my heartfelt appreciation to my supervisor, Prof. Seah Hock Soon, who has offered me this precious opportunity to be a Ph.D. candidate in Nanyang Technological University. I thank his patient guidance and invaluable advice all the time during my candidature. Without his consistent support and help, the research would not have been so enriching and fulfilling.

I also wish to express my sincere thanks to my co-supervisor, Assoc. Prof Qian Kemao, for his precious advice and encouragement during my candidature. He has provided much invaluable suggestions for my work.

I also would like to thank Dr. Quah Chee Kwang, for being a great collaborating colleague. My appreciation also goes to all my fellow colleagues at Multi-plAtform Game Innovation Center and Game Lab for their support and generous sharing of ideas, and School of Computer Engineering for offering me this great research opportunity.

Last but not least, I would like to thank my lovely husband, Mr. Hao Jianan, for his continuous support and encouragement; my parents and friends for their care and love, from which I have gained confidence and support to finish this journey.

Xu Xiang

Abstract

Creating hand-drawn animation is traditionally a laborious task, in which artists are highly involved to draw key frames and, thereafter, create inbetween frames from each pair of key frames. Although the development of paperless animation systems and auto-inbetween techniques somehow relieves artists' burden of creating many inbetween frames by hand, artists still have to painstakingly do all key-frame drawings by hand or using a computer. An alternate approach to avoid the laborious manual drawing task is to incorporate automatic feature line extraction from animated 3D models to create 2D animations. Computer-generated lines could be used as initial drawings for artists, where they can further refine or exaggerate lines with personal styles for animation production.

This thesis proposes a solution consisting of several techniques and an integrated framework for automatic line extraction in 2D animation. The first technique is a 3D line extraction method to extract lines from 3D triangle mesh models to closely resemble what artists would perceive and draw. The second technique is a method to generate temporally coherent 3D lines from 3D models using innovative hysteresis thresholding. The third technique is a matching algorithm to create correspondence of 3D individual lines (i.e., paths) across a sequence of frames. The second and the third techniques are used for sequencing the extracted lines frame-by-frame into an animation for temporal coherence. Each of the three algorithms is a valued contribution to Non Photorealistic Rendering.

Finally, an integrated framework is proposed with a novel interactive tool set for

artists to create 2D stylized animations by incorporating automatic line extraction from animated 3D models. The framework is artist driven – it allows artists to edit or stylize line work in 2D, similar to what they do in a traditional animation pipeline but utilizing 3D information. The framework is validated with a 2D animation software. We demonstrate the effectiveness of the framework in creating an animation with the software.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Line drawings are commonly used as a simple and effective way to convey shapes in 2D animations. Generating such an animation in a typical traditional pipeline involves first drawing key frames and thereafter creating inbetween frames from each pair of key frames. The key frames capture the most critical animated motions, and thus require highly-skilled artistic interpretation for shapes. The inbetween frames, which fill in the intermediate motions between each pair of keys, rely on sophisticated technical skills to evolve the lines precisely. An 80-minute animation viewed at the rate of 24 frames per second typically requires over a million drawings, with an average of a quarter of these drawings being key frames [95]. Thus, creating line drawing animation is by its nature, a highly involved, labor-intensive and time-consuming operation for artists.

The development of paperless animation systems and auto-inbetween techniques relieves artists' burden of painstakingly creating many frames by hand. With the advances of computer graphics (CG) techniques, computer assistance in 2D animation production becomes quite common nowadays [25]. Commercial 2D animation systems have been used widespread to assist artists in the production [1, 3, 4]. In such systems, digital tablets partially replace sheets of paper for artists to directly draw frames in computer. Hand-made drawings can be also scanned and vectorized for further processing. With

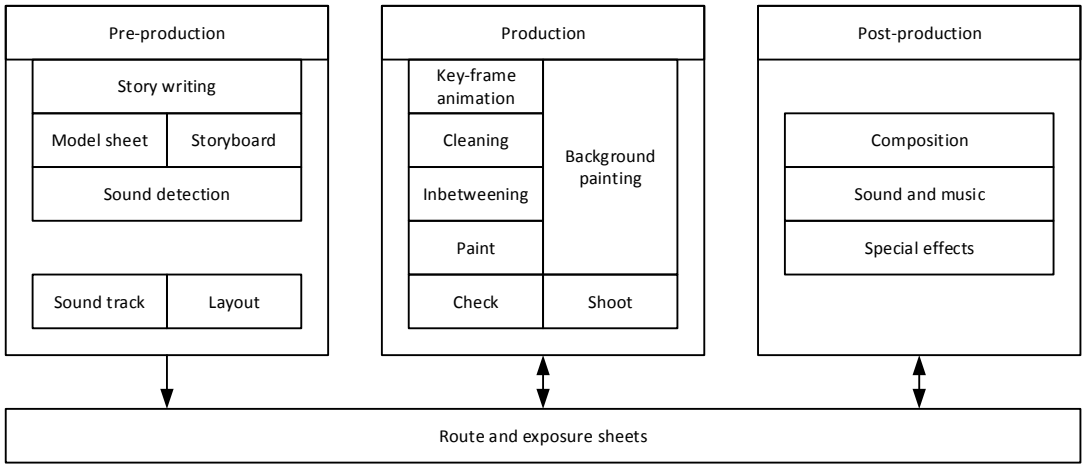


Figure 1.1: Traditional animation pipeline.

key frames digitalized, many algorithms for auto-inbetween are proposed [17, 31]. While the advent of these algorithms and systems have benefited some underlying animation mechanisms, the current 2D animation production follows much the same basic workflow as traditional animation [104] (also illustrated in Figure 1.1). For artists, they still have to painstakingly do all key-frame drawings by hand or by computer.

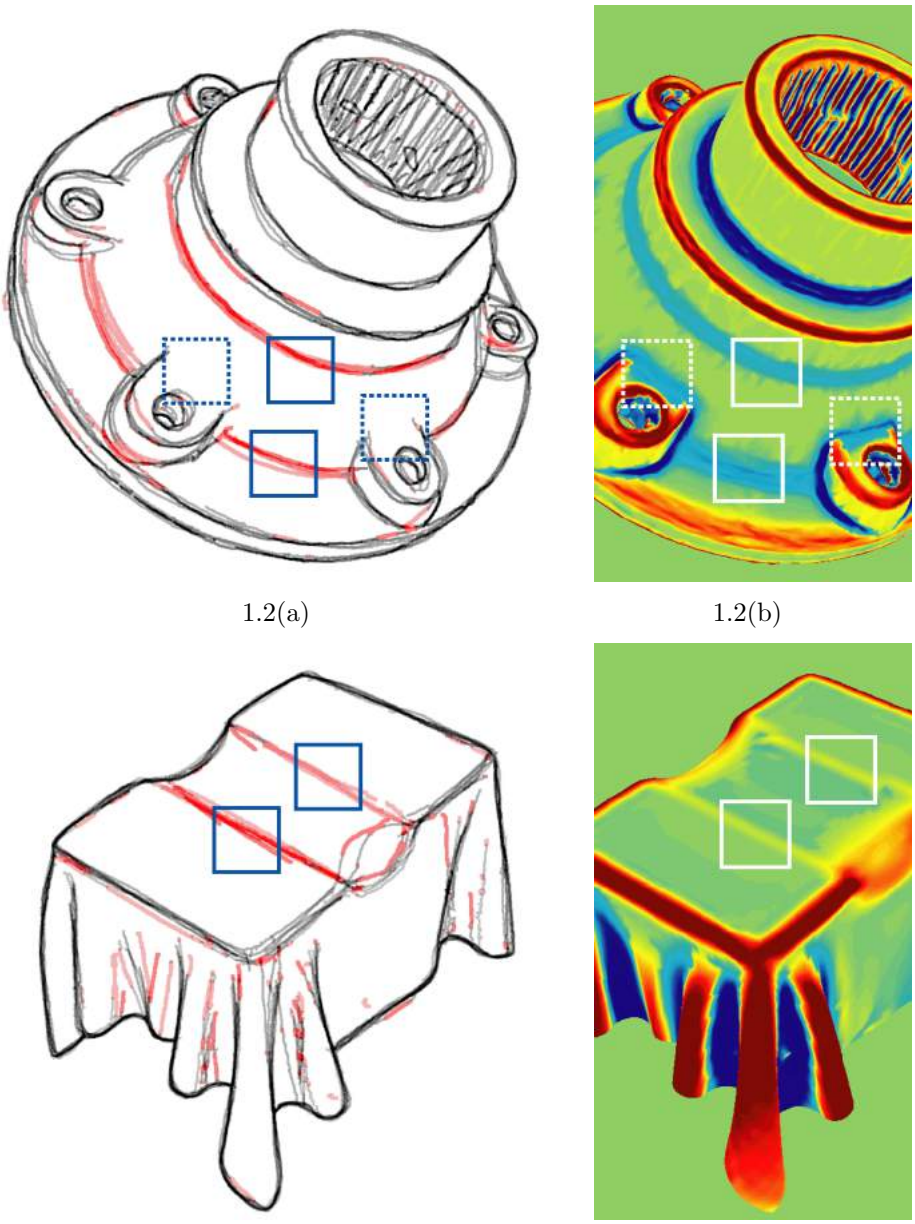
One possible alternative to the key-frame technique is to incorporate animated 3D models into the traditional pipeline, and derive line drawings automatically from the models [19, 32]. Each frame of line drawing can be extracted from the corresponding frame of the animated model, resulting in lines that artists would otherwise have to draw to convey the shapes. Repeating it frame-by-frame results in an animation sequence of line drawings. This technique avoids the process of drawing key and inbetween frames, and generates each frame of line drawings individually. Compared to the traditional methods, animated line drawings are expected to free artists from painstaking drawings. We are motivated by the hypothesis that computer-generated lines could be used as initial drawings for artists, where they can further refine or exaggerate lines with personal styles in animation production.

1.2 Objective

The objective of this thesis is to free artists from the laborious drawings of basic shapes and let them focus more on the part of creative work. To realize this target, we propose an integrated framework incorporating automatic line extraction from animated 3D models to create 2D animations. Our proposed framework is designed to be artist-driven, referring that artists are fully involved for controlling the entire production process for 2D animation.

A few critical problems exist when the technique of automatic line extraction is put into practical use in 2D animation. The first problem is where the extracted lines would be drawn on the 3D models, to closely resemble what artists would perceive and draw. The second is temporal coherence problems brought by sequencing the extracted lines frame-by-frame to form an animation. The third is how the computer-generated lines can be used in production process for 2D animation, where artists are involved to further edit or stylize the line work.

Many researchers have worked in the area of automatic feature line extraction, and a variety of line drawing algorithms have been reported, such as silhouettes [46], suggestive contours [27], ridges and valleys [76], and apparent ridges [51], etc. Despite advances in line drawing algorithms, computer-generated line drawings however could not achieve the same level of sophistication as hand-drawn illustrations. One reason is that most of the existing algorithms are based solely on local geometric properties, such as surface normals and curvatures. However, natural objects contain bumps, dips, and undulations of varying geometries, which are still challenging for algorithms to handle them nicely. Besides, it seems that artists' criteria for line selection are beyond the local features detected by algorithms. See Figure 1.2 for an example. In this thesis, we propose a new line drawing algorithm to extract lines from 3D triangle mesh models, aiming to convey the 3D shape of the model in a succinct way.



1.2(a) 1.2(b)

1.2(c) 1.2(d)

Figure 1.2: Artists' sophisticated line selection. The figures on the left column show two examples where artists choose to draw lines, while the figures on the right column show the color-coded features of the respective models. The red lines (highlighted by solid boxes) in this composite (a) are characterized as valley lines [76]. However, several artists have omitted locally stronger valley features (highlighted by dotted boxes), as shown by the maximum curvature of the model (b). The red lines in (c) are also ridge-like feature lines [76], but the curvature strength in the area is very low relative to the rest of the model (d). Images courtesy of Cole et al. [21].

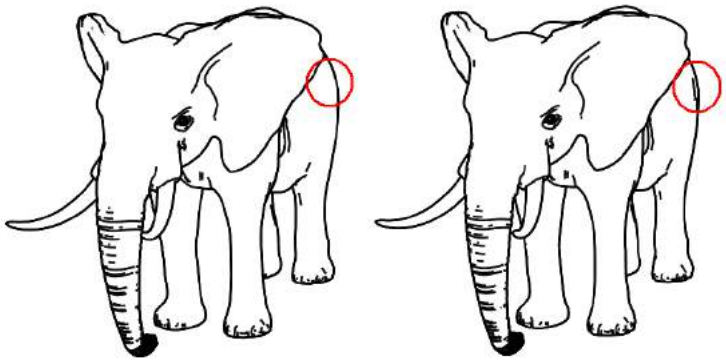


Figure 1.3: A small rotation of the model results in the line in the circle suddenly appearing from the image. One kind of line drawings is drawn here – apparent ridges [51].

Temporal coherence is a necessity for effective generation of line drawing animations. Temporal coherence can be loosely described as models or lines changing smoothly and continuously over time, with respect to small viewpoint and/or model movement [74]. Temporal incoherence can introduce severe visual artifacts into the animation, which are disturbing to audiences. An example is illustrated in Figure 1.3. Sequencing computer-generated line drawings frame-by-frame to form an animation would lead to temporal coherence problems, if it does not consider inter-frame correspondences of the respective lines [9, 10]. The lines usually do not stay in the original geometric positions or remain the same lengths in successive frames. They may randomly appear and disappear from time to time. These variation of the lines cause temporal artifacts like flickering and popping. The lines may also be arbitrarily broken into multiple pieces and several lines may merge into one piece. These variations of the lines are referred as topological discontinuities of lines, which cause temporal artifacts like splitting and sliding.

A number of methods have been proposed to achieve temporally coherent line drawing animations. However, most of the existing methods only tackle silhouettes, for which coherence has been achieved [12, 14, 52, 55]. A few methods tackle high-order lines, such as suggestive contours and apparent ridges. However, they are only suitable for specific temporal artifacts [10, 68] or line types [26]. As a result, coherence has not been

achieved satisfactorily, although these lines have been widely used for detailed depiction of the shapes of 3D models [76]. In this thesis, we propose a method to extract high-ordered lines from 3D models, which reduces temporal artifacts. We also propose method to establish 3D line correspondence across frames to minimize visual artifacts of the lines and provide a coherent shape and position of the lines.

Finally, a question exists on how the computer-generated lines (i.e., strokes) might be used in the process for 2D animation production. Many existing methods are available to create stylized animations in a variety of styles found in traditional animation [71, 74, 75, 82, 110]. However, the resulting lines are hard for artists to further edit, which limits their usage. In this thesis, we propose a framework of interactive tools that allow artists to manipulate, edit or stylize the line work. Correspondence could be incorporated in our framework to facilitate the control of the strokes across frames.

1.3 Contributions

The main contributions of this thesis are:

- **XPLs:** a novel line drawing algorithm called eXtended Phong Lines is proposed to extract lines automatically from a 3D triangle mesh model.
- **Coherent 3D line extraction:** an effective method is proposed to extract 3D lines from 3D models using hysteresis thresholding for temporal coherence.
- **Coherent 3D line evolution:** a fully automatic line matching algorithm is proposed to create correspondence of 3D lines across frames, and an effective approach is presented to improve temporal coherence based on the correspondence.
- **A fully artist-driven framework:** an integrated framework is presented to incorporate coherent 3D line extraction and correspondence information to create 2D stylized animations.

1.4 Organization

The rest of this thesis is organized as follows:

In Chapter 2, we review the relevant NPR algorithms towards line drawings. Particularly, we focus on how the lines are extracted automatically, how they are stylized, how their temporal coherence is maintained if animated, and how they are utilized in animation system. Firstly, the line extraction algorithms (namely line drawing algorithm in Chapter 2), can be classified into object-space, image-space and hybrid methods. We pay special attention to polygonal mesh where the lines are extracted from. A survey to evaluate line drawing algorithms are given as well. Secondly, the line stylization algorithms can be classified into vector-based and raster-based algorithms. A further investigation on vector-based stylization discusses the representations of vectorized line and texture mapping techniques. Thirdly, temporal coherence in line extraction and stylization steps are discussed respectively and algorithms are given accordingly. A survey to evaluate temporal coherence is also given. Lastly, an overview of animation systems are presented. The systems are classified by the dimension of the main working space.

In Chapter 3, we propose a novel line drawing algorithm called eXtended Photic Lines (XPLs), which extends previous work Photic Extremum Lines (PELs). PELs use Phong shading to illuminate 3D objects, and extract geometric feature lines from the illuminated model. We propose a refined lighting model called X-Phong which extends the basic Phong light model with a curvature light to enhance the shape depiction. XPLs are thus extracted with more details for perceived shape. We also provide parameters to control the level of details of X-Phong.

In Chapter 4, we propose an algorithm to extract coherent 3D lines using hysteresis thresholding. Our method focuses on object-space lines extracted from triangle mesh models with moderate or high geometric complexity. Our method is based the classic hysteresis model, and has two major benefits. Firstly, it achieves temporal coherence

on a per line basis but does not require predefined line correspondence. Secondly, it is generalized to handle all types of high-order lines, such as ridges/valleys, suggestive contours and our proposed XPLs. An evaluation is also given by comparing our method and a few representative line drawing algorithms. The methods are applied on a wide range of animated 3D models, including real-world and man-made models with both fine and coarse triangle tessellations.

In Chapter 5, we present an automatic approach to build line correspondence from the coherent lines generated in Chapter 4. The approach consists of three core algorithms. Firstly, a robust tracing algorithm is included to transfer collections of line segments on 3D models into individual 3D paths. Secondly, the 3D line matching algorithm is proposed to generate correspondent relationship among paths in successive frames of a line drawing animation. Thirdly, an automatic approach to evolve these correspondent 3D lines frame by frame is presented.

In Chapter 6, we introduce a fully artist-driven framework which integrates the proposed algorithms in Chapter 3 to 5 for stylized animations. The framework utilizes the materials from both 3D models and 2D images. It allows artists to edit or stylize line work in 2D, just like what they do in a traditional animation pipeline, while incorporating 3D information.

In Chapter 7, we conclude our work and give potential extensions for it. The possible improvements could be easily applied or integrated into our proposed animation framework.

Chapter 2

Related Work

In recent years, non-photorealistic rendering (NPR) has improved greatly on line drawings, especially in the fields of line extraction, stylization and dynamic interactive system. General introductions on line drawings can be found in [36, 46, 63, 92]. This chapter investigates the following specific research work: line drawing algorithms for automatic line extraction, line stylization algorithms and temporal coherence algorithms.

2.1 Line Drawing Algorithms

Line extraction from 3D models has received quite a lot of attention and a number of approaches have been proposed. These approaches can be classified into object-space, image-space and hybrid algorithms. Object-space algorithms generate lines that capture essential object properties based on surface geometry and viewing direction. In contrast, image-space algorithms extract lines by exploiting discontinuities in image buffer using image processing methods. Hybrid algorithms are slightly different from the image-space algorithms – they include an additional depth buffer for line extraction. In the following sections, we will introduce the three types of algorithms respectively.

2.1.1 Object-Space Algorithms

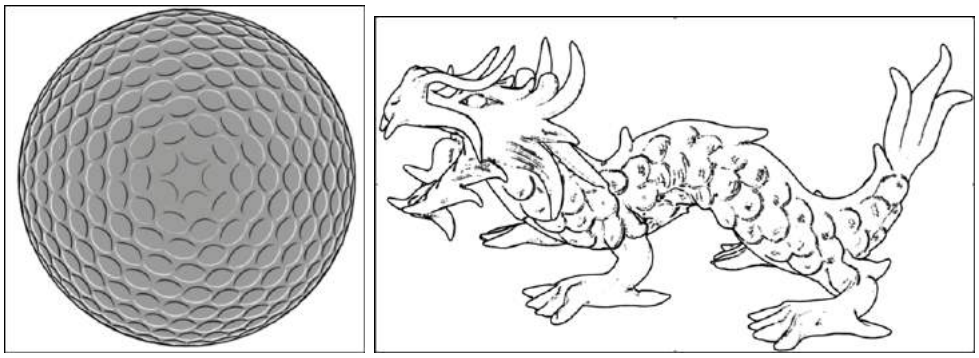
Polygonal meshes remain the most common and flexible form in line drawing history to approximate surfaces [41]. Early methods simply detect the folding angle or the segment chain for lines on simple polygonal mesh [7]. With the rise of the qualification of a single 3D model and the powerful processing ability on fine-tessellated mesh models, such methods are neither accurate or speed-optimized. The processing of finely-tessellated surface becomes common and more recent research shifts interest to algorithms that draw lines from such surface [42]. We focus on surfaces approximated by finely-tessellated triangle meshes in this thesis.

There are two fundamental types of lines that should be drawn: discontinuities in surface depths – *outlines* [35]; and discontinuities in surface orientations – *sharp creases* [38]. Both are classical elements in line drawings, and are frequently used in conveying 3D shapes in NPR. The discontinuities of surface geometry can be directly obtained as lines from the surface of a 3D model. However, many models used in animation are natural objects, which contain bumps, dips and undulations of varying geometry. They are generally too smooth to contain significant discontinuities in surface depth and orientation. Therefore, the two classical line types are generally insufficient to depict delicate surface shapes. In order to depict 3D shapes of such surfaces, a variety of object-space algorithms have been proposed (see Figure 2.1 for illustrations).

Surface differential geometry [18, 30, 58] is widely adopted among object-space algorithms to handle extraction of lines from triangle mesh surfaces. The lines are extracted as geometric local maxima of the surface, which are computed through solving derivative equations of differential geometry. Surface normal is often considered as a first-order quantity. Second-order structure refers to surface curvature computed from normal vectors. Third-order derivative is an additional notation that describes derivatives of curvature. A further variety of the algorithms uses position of the viewer. Lines,

2.1(a) 2.1(b) 2.1(c) 2.1(d)

2.1(e) 2.1(f) 2.1(g)



2.1(h) 2.1(i)

Figure 2.1: Computer-generated line drawings in object space: (a) contours [42], (b) suggestive contours [27], (c) PELs [109], (d) ridges (in black) and valleys (in white) [76], (e) demarcating curves [60], (f) apparent ridges [51], (g) principal highlights (in white) [28], (h) suggestive highlights [26] and (i) Laplacian Lines [119].

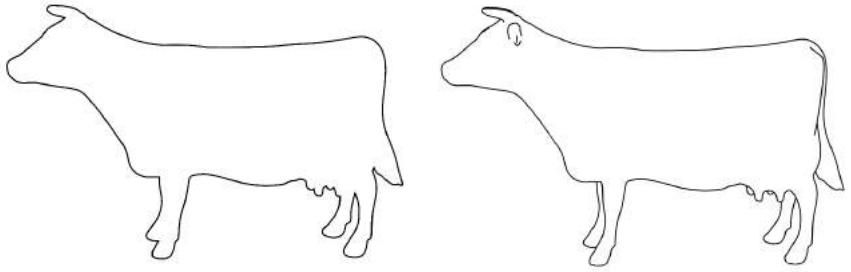


Figure 2.2: Comparison between silhouettes (left) and contours (right).

solely defined in terms of the geometry of the shape and position of the viewer, are *purely geometric lines*. Some recent research also considers lighting conditions of surface for line extraction, which is a simulation of 2D edge detection on 3D models. The name *edge-like lines* are used to describe such lines in the thesis. In the following sections, we introduce purely geometric lines and edge-like lines respectively. For the former, we review them by the order of the derivatives they use.

First-Order Lines. *Silhouettes* and *Occluding Contours* are first-order lines defined as the outline of a shape. Occluding contours are often short for *Contours*. Silhouettes and contours have been widely accepted in depicting 3D shapes and commonly used in NPR applications nowadays, and thus are paid most attention among all the object-space algorithms [38, 42, 71, 75]. Silhouettes are defined as lines that describe the outline of an object with a featureless interior. Contours describe a wider range of lines than silhouettes, marking any visual depth discontinuity where a surface normal would change its sign. Contours can be considered as interior and exterior silhouettes (see Figure 2.2 for the differences of these two lines). Both silhouettes and contours are view-dependent lines which depend not only on the differential geometric properties of the surface, but also on the viewing direction. They change whenever the camera changes its position or orientation.

A straightforward approach using hardware method to obtain silhouettes is to turn on depth buffering and render the model in two passes [38]. The first pass renders all

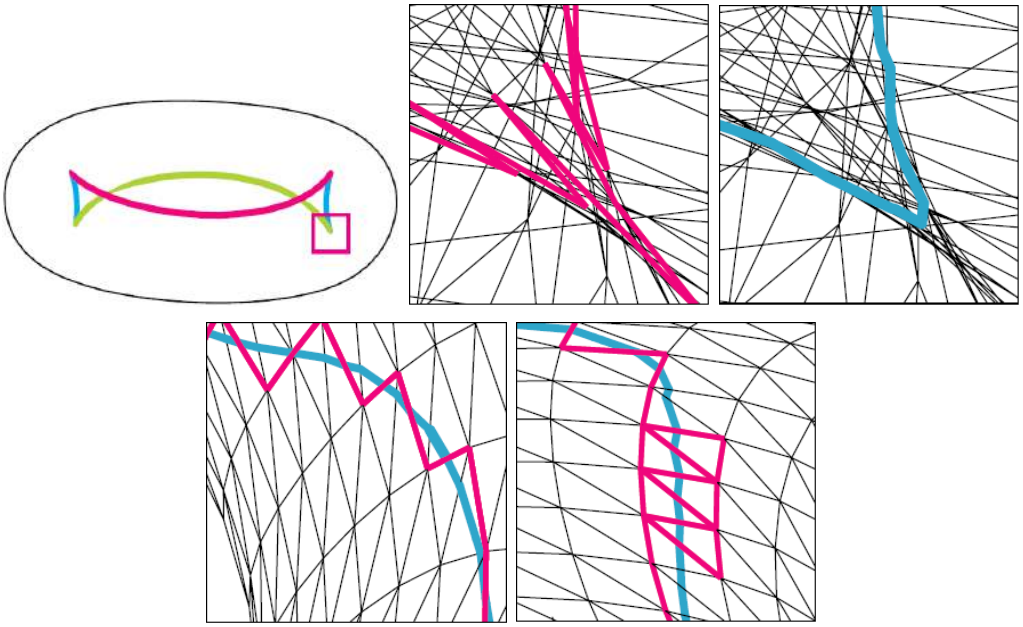


Figure 2.3: Illustrations for drawing smooth silhouettes (blue color) instead of polygonal approximation (magenta color) on mesh surface (images courtesy of Hertzmann et al. [42]).

the edges that the model contains as line strips (probably with a large line width). The second pass renders the mesh of the model in white color. This results in silhouette only and anything drawn inside the body of the mesh will be clipped. This method is simple and fast. Since no lines are truly extracted from the model, this method does not allow further stylization along the lines.

A software method to detect contours checks every triangle edge of the mesh to see if it connects a front-facing triangle and a back-facing triangle. Gooch et al. [38] propose a pre-computation of a spherical hierarchy data structure to speed up, but their algorithm is rather complicated and difficult to implement. Northrup and Markosian in two papers [71, 75] propose contour detection methods by checking a small fraction of triangle edges instead of all of them. When a new contour is found, its neighbors are checked for local continuation of the contour.

Hertzmann et al. [42] propose a deterministic algorithm for detecting contours defined

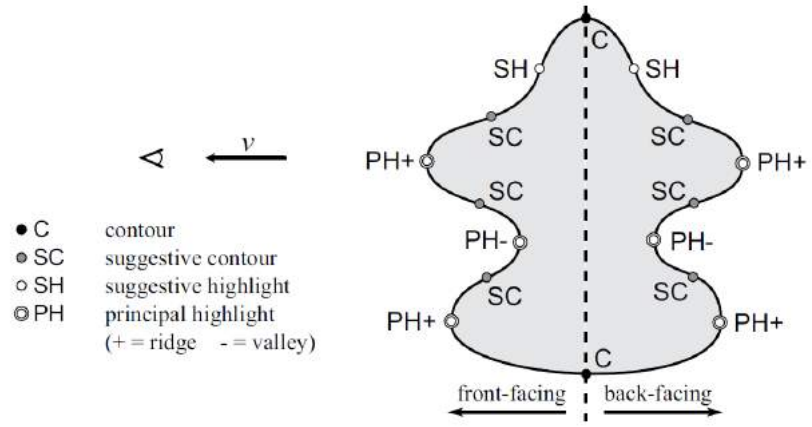
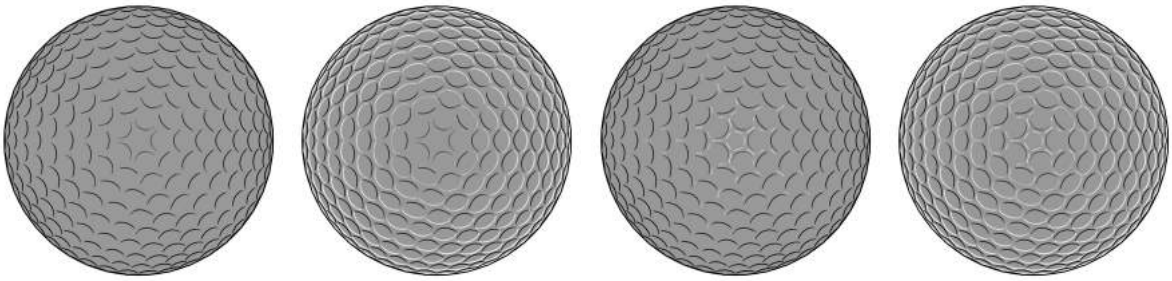


Figure 2.4: Different types of lines given the viewing direction **v** (for clarity this figure uses orthographic projection, hence **v** is constant, images courtesy of DeCarlo et al. [28]).



2.5(a) C+SC 2.5(b) C+SC+SH 2.5(c) C+SC+PH 2.5(d) C+SC+SH+PH

Figure 2.5: Different second-order lines to draw a golf ball in gray color (images courtesy of DeCarlo et al. [28]).

on a smooth surface. The contours are loci of points where their normals are perpendicular to the viewing position on the smooth surface. The lines rendered are illustrated in Figure 2.3. This way of rendering becomes popular among object-space line drawing algorithms developed recently.

Second-Order Lines. *Suggestive Contour*, *Suggestive Highlight* and *Principal Highlight* are second-order lines. They describe local maxima of surface normal variations while exploring a surface being viewed.

Suggestive contours proposed by DeCarlo et al. in [27] are found to be the lines where a surface bends sharply away from the viewer though it is still visible, which is helpful

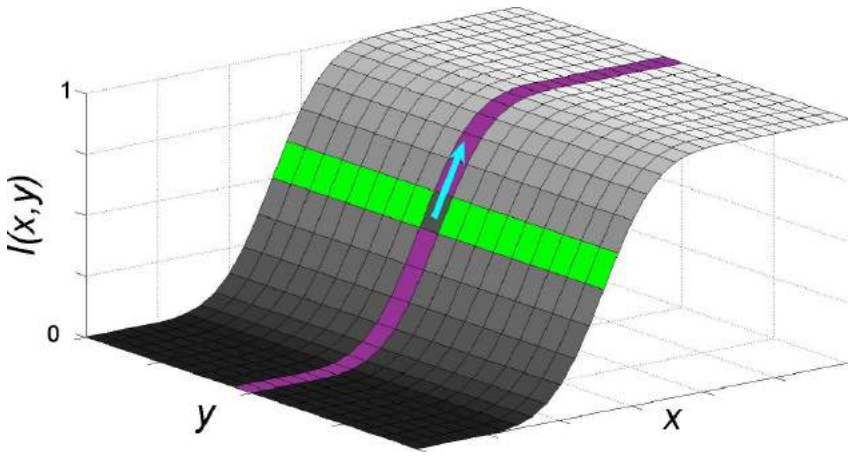


Figure 2.6: Local terrain (smoothed step edge); the demarcating curve in green; the cross section orthogonal to it in magenta; its local direction of curvature gradient in cyan (image courtesy of Kolomenkin et al. [60]).

in conveying shapes. It can be considered literally as extensions of contours from nearby viewpoints.

Suggestive Highlight and Principal Highlight proposed by DeCarlo et al. in [28] are lines meant to indicate or abstract shading highlights. Figure 2.4 conveys the geometric relationship of these two lines as well as suggestive contour and contour. Figure 2.5 illustrates the shape of a golf ball using lines selected from contour and all the second-order lines.

Third-Order Lines. *Ridges and Valleys*, *Apparent Ridges* and *Demarcating Lines* are third-order lines. They are the loci of points for which there is an extremum of the curvature in a certain gradient direction.

Ridges and valleys proposed by Ohtake et al. [76] are loci of points where the positive (negative) variation of the surface normal in the direction of its maximal changes attains a local maximum (minimum). They are view-independent lines, meaning that they do not change with respect to the viewing direction, and thus can be seen as markings sticking to surface.

Demarcating lines proposed by Kolomenkin et al. [60] are defined similarly to ridges

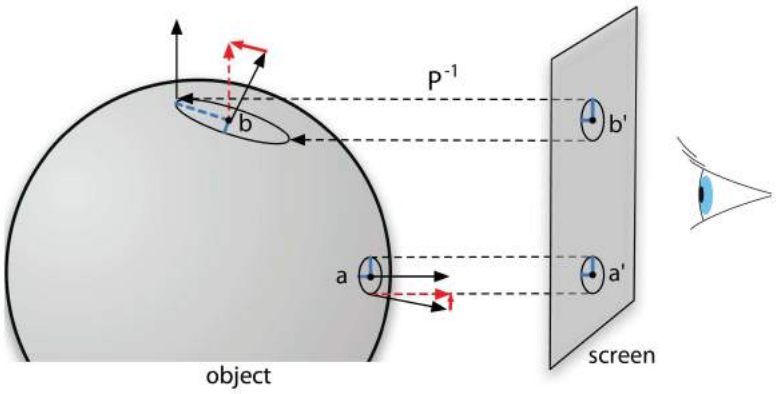


Figure 2.7: The maximum view-dependent curvature at b' is much larger than at a' uniquely because of projection (image courtesy of Judd et al. [51]).

and valleys, except that they lie at inflection points rather than extrema of principal curvatures. These inflection points often seem to correspond with an intuitive segmentation of a shape. Demarcating lines are used in particular for applications such as archeology, architecture and medicine. Figure 2.6 illustrates the position of demarcating lines on a surface.

Apparent ridges proposed by Judd et al. [51] are defined as ridges and valleys of a view-dependent curvature measurement that take foreshortening into account. When the effect of projection is small, at front facing parts of the model, ridges and apparent ridges are similar. On parts of the object that turn away from the viewer, apparent ridges adjust ridges to be more perceptually pleasing (Figure 2.7). The result of apparent ridges often extract more contour-like lines rather than just interior feature, because view-dependent curvature approaches a maximum of infinity at the contours and so contours are extracted as apparent ridges.

Edge-Like Lines. A few techniques, such as *Photoic Extremum Lines (PELs)* and *Laplacian Lines*, have been proposed to generate line drawings from surface illuminations. They use Phong lighting in surface illuminations. Figure 2.8 shows the comparisons of these two lines.

Figure 2.8: PELs (left) vs Laplacian Lines (middle). The right image shows the shaded image of the model from which the two types of lines are extracted (image courtesy of Zhang et al. [119]).

Photic Extremum Lines (PELs) proposed by Xie et al. [109] is a pioneer approach to characterize the high variations of illumination on 3D models. The approach mainly uses Phong shading for the illumination, which is a fairly rough approximation for rendering objects. To further extract detailed feature lines from the surface, it also allows users to interactively increase or remove detail through additional auxiliary spotlights. This costs extra computation time and the desirable result might be hard to achieve because a number of parameters have to be adjusted. Additional optimizations can help to adjust the parameters automatically, but it increases computation cost remarkably.

GPELs proposed by Zhang et al. [118] is an enhanced shading technique incorporating PELs, which replaces the complicated computation of diffuse shading in Phong light. The technique is also implementation-optimized in GPU to speed up.

Laplacian lines proposed by Zhang et al. [119] is defined as a view-dependent feature lines which generalize the Laplacian-of-Gaussion (LoG) edge detector to 3D surfaces. The algorithm is robust and insensitive to irregular tessellation for they choose a robust mesh Laplace operator.

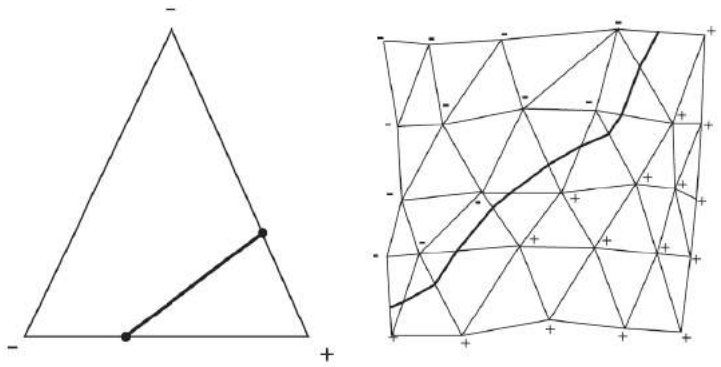


Figure 2.9: 3D ordered point list. The positive and negative signs are used as properties for extracting lines. The points are on the local maxima, which represent zero-crossings of the properties.

Tracing Lines. A commonly-used method to render lines is to trace the lines along a triangle mesh surface. This tracing method is suitable for both purely geometric lines and edge-like lines. Once differential derivatives are computed at each mesh vertex, we need to find the zero crossing and join them to get the lines (illustrated in Figure 2.9). Let $\mathbf{h}(\mathbf{v})$ equal to the differential value of vertex $\mathbf{v}$. For each edge $\mathbf{v_1v_2}$ of a triangle, if $\mathbf{h(v_1)h(v_2)} < 0$, a zero-crossing vertex $\mathbf{p}$ exists and can be estimated using linear interpolation to approximate it on the edge $\mathbf{v_1v_2}$:

$$\mathbf{p} = \frac{\mid \mathbf{h(v_2)} \mid \mathbf{v_1} + \mid \mathbf{h(v_1)} \mid \mathbf{v_2}}{\mid \mathbf{h(v_2)} \mid + \mid \mathbf{h(v_1)} \mid} \qquad \text{(Eq. 2.1)}$$

If two zero crossings are detected on the edges of a triangle, we connect them with a straight *line segment*. If all three edges of the triangle contain zero crossings, the centroid of the triangle and the three vertices are connected with three line segments.

Summary. Table 2.1 summaries the properties of the lines introduced in Section 2.1.1, including the core formula that the algorithms used and the dependencies of light or view.

Table 2.1: Comparison of object-space line drawing algorithms.

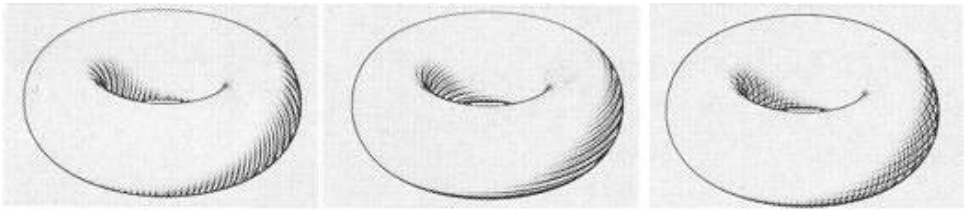
Algorithm	Equation	Dependency
Contours	$\mathbf{n} \cdot \mathbf{v} = 0$	view
Suggestive Contours	$\boldsymbol{\kappa}_\mathbf{r} = 0$ *and* $\begin{cases} \mathbf{D}_\mathbf{w}\boldsymbol{\kappa}_\mathbf{r} > 0 \text{ where } \mathbf{n} \cdot \mathbf{v} > 0 \\ \mathbf{D}_\mathbf{w}\boldsymbol{\kappa}_\mathbf{r} < 0 \text{ where } \mathbf{n} \cdot \mathbf{v} < 0 \end{cases}$	view
Suggestive Highlight	$\boldsymbol{\kappa}_\mathbf{r} = 0$ *and* $\begin{cases} \mathbf{D}_\mathbf{w}\boldsymbol{\kappa}_\mathbf{r} < 0 \text{ where } \mathbf{n} \cdot \mathbf{v} > 0 \\ \mathbf{D}_\mathbf{w}\boldsymbol{\kappa}_\mathbf{r} > 0 \text{ where } \mathbf{n} \cdot \mathbf{v} < 0 \end{cases}$	view
Principal Highlight	$\mathbf{w} \cdot \mathbf{e_1} = 0$ *and* $\begin{cases} \mathbf{D}_{\mathbf{w}_\perp}\boldsymbol{\tau}_\mathbf{r} < 0 \text{ where } \mathbf{n} \cdot \mathbf{v} > 0 \\ \mathbf{D}_{\mathbf{w}_\perp}\boldsymbol{\tau}_\mathbf{r} > 0 \text{ where } \mathbf{n} \cdot \mathbf{v} < 0 \end{cases}$	view
Ridges—Valleys	$\mathbf{D}_{\mathbf{e_{max}}}\mathbf{k_{max}} \mid\mid \mathbf{D}_{\mathbf{e_{min}}}\mathbf{k_{min}} = 0$	N/A
Demarcating Curves	$\mathbf{D}_\mathbf{g}\mathbf{D}_\mathbf{w}(\mathbf{n} \cdot \mathbf{v}) = 0$	N/A
Apparent Ridges	$\mathbf{D}_{\mathbf{e'_{max}}}\mathbf{k'_{max}} = 0$	view
Laplacian Lines	$\triangle \mathbf{I} = 0$	light, view
PELs	$\mathbf{D}_\mathbf{g}\|\nabla \mathbf{I}\| = 0$	light, view

2.1.2 Image-Space Algorithms

In image-space approaches, 3D objects are rendered as images or geometric buffers, and image processing methods yield output drawings. Image-space algorithms usually undergo two passes. In the first pass, they render 3D objects into an image describing how scene is illuminated, then blur it and save it to texture memory. In the second pass, they apply line detectors to the image and render the detected lines with desired styles. In the following sections, we will first introduce image processing methods to detect lines in the rendered image, followed by the algorithms for enhancing local feature, which can be applied in rendering for line detections.

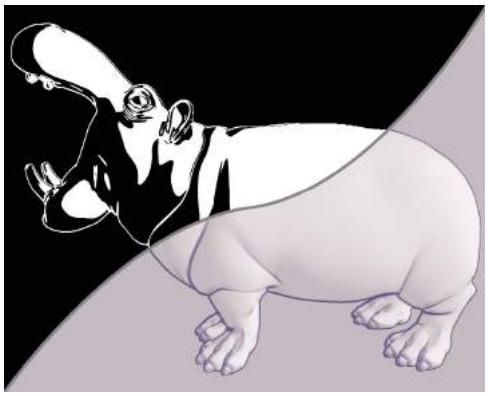
Line Detection. In image-space approaches, objects are rendered to form an image or geometric buffer, and image processing methods, such as edge detection [54, 66, 88], yield an output drawing. The result is visually pleasing, but pixel-based representation suffers from the loss of geometric information and is unsuitable for additional processing.

Saito and Takahashi [88] encode geometric information on image buffer and apply



2.10(a)

2.10(b)



2.10(c)

2.10(d)

2.10(e)

Figure 2.10: Computer-generated line drawings from image space: (a) illustration in lines and hatching effect [88], (b) abstract shading [66] in lines with tone shading effect (c) apparent relief [99] in two different shading effects, (d) flow-field based anisotropic Difference-of-Gaussian (DoG) filtering [54] and (e) bilateral and DoG filtering [62].

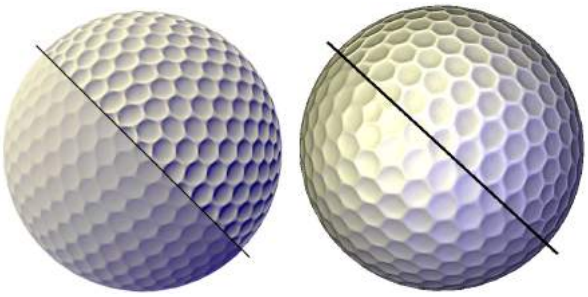
several edge detectors such as Sobel operator to extract discontinuities of an image, from which contours and other feature lines can be drawn for comprehensive shapes rendering (Figure 2.10(a)). It however fails to separate these two different types of lines.

Lee et al. [66] develop a GPU-based algorithm for ridge detection from a tone-shaded image (Figure 2.10(b)). The main job of the algorithm is to compute opacity to composite the line colors into the frame buffer. Besides, it also involves a filtering technique that allows line thickness to be controllable.

Vergne et al. [98] introduce a novel view-dependent shape descriptor called *Apparent Relief*, from which continuous shape cues are easily extracted (Figure 2.10(c)). The descriptor consists of a combination of object and image space attributes, which facilitates simple manipulation for users, as well as a vast range of styles and natural levels of details.

Kang et al.[54] propose a flow-field based anisotropic Difference-of-Gaussian (DoG) filtering framework. Their work results in long and smooth lines without breaking with the target of spatial coherence by creating clean lines (Figure 2.10(d)). Kyprianidis and Döllner [62] extend this approach and present separable implementations of the bilateral and difference of Gaussians filters that are aligned to the local image structure (Figure 2.10(e)).

Kim et al. [57] propose a new way to render line-art illustrations from dynamic 3D surfaces, while stylizing specular highlight (reflective and refractive) with hatching. Their method starts with a real-time principal direction estimation, followed by a novel stroke propagation algorithm using multi-perspective projections, which models the local reflective and refraction as General Linear Camera (GLC) [116]. Lastly, an image-space stroke mapping algorithm is applied to map the stroke hatching to the 3D surfaces using the directions of the computed or propagated strokes. One limitation to this method is, like object-space methods, the lines rendered using Kim et al.'s method still suffer from breaking into pieces under coarse input mesh. This is because the normal rays computed for lines have numerical errors.



2.11(a) 2.11(b)

Figure 2.11: (a) Exaggerated Shading. (b) Light Warping. Images courtesy of Rusinkiewicz et al. [87] and Vergne et al. [99].

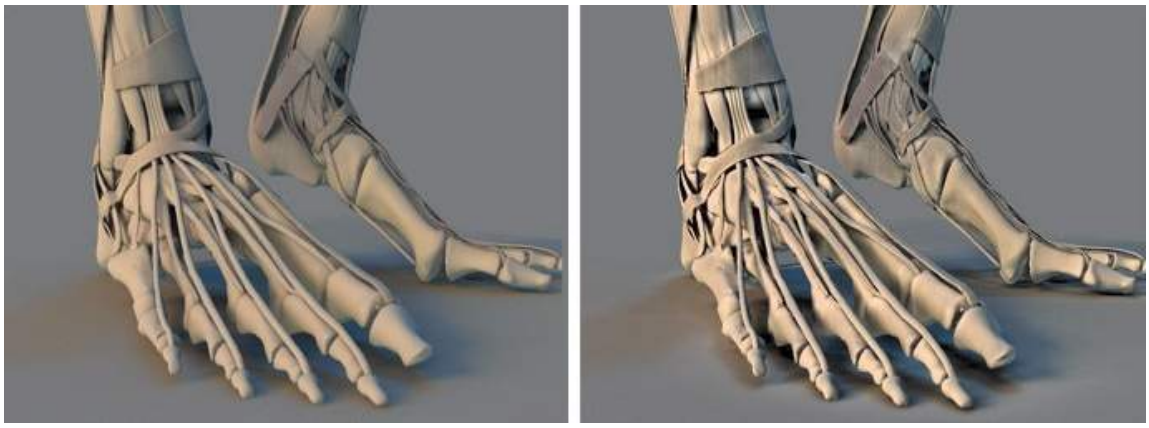


Figure 2.12: 3D unsharp masking (images courtesy of Ritschel et al. [84]).

Enhanced Feature Depiction. With standard shading models such as Phong, Toon and global illumination, it may be difficult to depict detail everywhere on a surface from a single view, since fine structure is typically revealed only with direct lighting. Amended lighting model can be used to increase the amount of information conveyed by a single view. The idea is applied in shape depiction to enhance surface features for depicting details, such as geometry-based lighting [64], 3D unsharp masking [84], exaggerated shading [87] and light warping [99].

Rusinkiewicz et al. [87] perform a multi-scale shading to convey the detail of surface in different levels, with each shading using successively smoother normals for lighting calculation (Figure 2.11(a)). Local light direction per vertex per scale could be adjusted

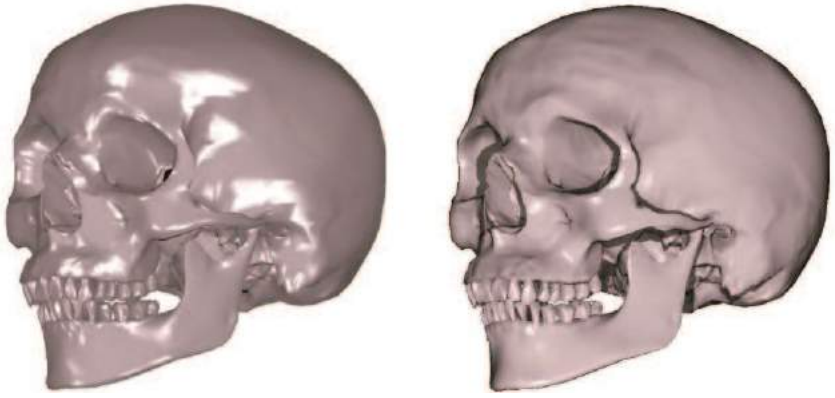


Figure 2.13: Geometry-based lighting (images courtesy of Lee et al [64]).

to arbitrary 3D objects for better descriptions of surface details. Therefore, it is simple to produce effects that vary the relative emphasis for user-specified region of interest.

Vergne et al. [99] propose a light warping function to locally compress or stretch the illumination directions and exaggerate the deformation of reflection patterns that naturally occur on curved objects (Figure 2.11(b)). One can observe that changing the direction of incoming light rays has the effect of contracting a wider region of the environment, as if the object were more curved.

Ritschel et al. [84] introduce 3D unsharp masking that enhances local scene contrast. The unsharp mask first smoothens the entire objects and then adds back the scaled difference between the original and the smooth surface (Figure 2.12). The difference is not only based on surface geometry, but also includes reflectance of the surface material and incident light.

Lee et al. [64] propose a geometry-based lighting framework that local regions can be illuminated by lights based on the local surface geometry (Figure 2.13). The framework allows regions to be lit with local consistency and global discrepancy to enhance the perception of the shape. Extra silhouette and proximity shadows are also applied for feature enhancement.

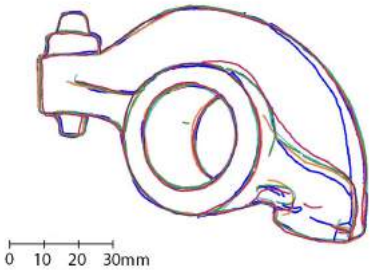
2.1.3 Hybrid Algorithms

Raskar and Cohen [83] propose a hybrid algorithm to extract silhouette using normal map and depth buffer. The algorithm overcomes the precision problem of the depth buffer and provides higher degree of control over the output of the silhouettes.

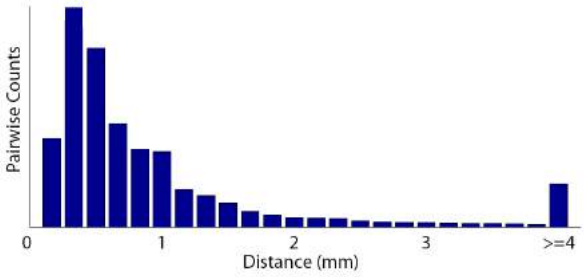
Zhang et al. [120] propose a hybrid algorithm called Splatting Line to draw lines from point clouds. The line is generated from depth image and two different images rendered by diffuse shading at different blur levels of illumination. The line is drawn by computing the difference between the two diffuse shading images. Besides point clouds, the extraction of splatting line can be extended to various types of 3D models, including volume meshes and polygonal meshes. It can also be applied to both static and dynamic models. However, this line is limited in two ways: firstly, it is hard to apply stylization to the line since the line is rendered in pixel level. Secondly, to render smooth lines, the point cloud is required to be fairly dense. Therefore, when the input is a polygonal mesh, the method requires point resampling to a high level of density, which makes it run slower than the existing object/image space methods.

2.1.4 Evaluation of Line Drawings

There are several strategies to assess the effectiveness of NPR algorithms. One approach is to quantify computer-generated images pixel-by-pixel with some defined ground truths. Cole et al. [21] study hand-made line drawings by artists and perform a formal quantitative evaluation on pixel-overlapping between computer-generated line drawings and hand-made drawings. The result is illustrated in Figure 2.14. Although their method is limited in many aspects such as potential bias, limited objects to draw, inaccurate per-pixel analysis and loss of view dependency, it provides an objective evaluation for NPR, which previously could only be target-driven or by perceptual evaluations that are criticized for their subjectiveness.



2.14(a)



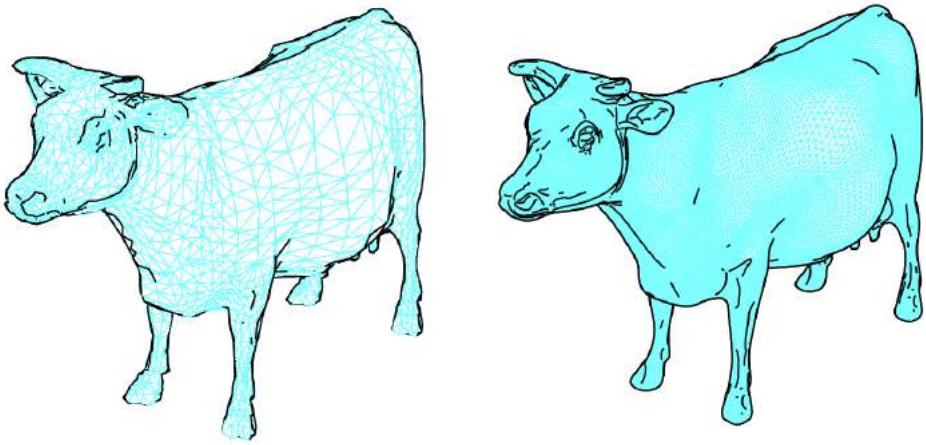
2.14(b)

Figure 2.14: Consistency of artists' lines. (a) Five superimposed drawings by different artists, each in a different color, showing that artists' lines tend to lie near each other. (b) A histogram of pairwise closest distances between pixels for all 48 prompts. Note that approximately 75% of the distances are less than 1mm (images courtesy of Cole et al. [21]).

A second approach is to survey users for their perceptual impressions. Isenberg et al. [48] do an observational study on qualitative evaluation of NPR images, with several groups of users as examiners to fulfill a study session: firstly, to perform unconstrained criteria on their own and sort the images according to the criteria, and then to answer several focused questions, like "which images do you like best and why?".

A third approach is to measure the performance in some tasks. Gooch et al. [37] evaluate the effectiveness of the resulting images through psychophysical studies to assess accuracy and efficiency in both recognition and learning tasks. Winnemöller et al. [106] compare the time that participants take to recognize shapes under several different perceptual shape cues such as shading, texture and contours, where target objects are moving rigid objects.

Finally, a fourth approach is to make use of techniques developed in perceptual psychology, such as gauge figure studies by Koenderink et al. [59]. Cole et al. [22] also follow this mechanism to evaluate feature lines.



2.15(a) Lines with low resolution 2.15(b) Lines with high resolution

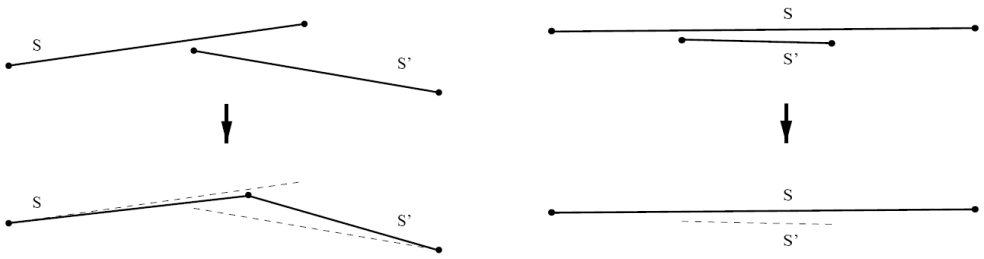
Figure 2.15: Two different triangle mesh resolutions of a cow model, with the same thresholds used in the algorithm.

2.2 Line Stylization Algorithms

The object-space and image-space methods both produce nice-looking results, even with stylized effects. However, they result in different line outputs. The lines obtained from object-space methods are extracted as ordered line segments, which can be easily traced into vector lines. The lines obtained from image-space methods result in raster images with edge pixels and non-edge pixels. It is difficult to subsequently obtain clean and smooth vector lines from these edge pixels. Comparing to raster-based stylization, vector-based stylization is more flexible as a vast range of styles can be applied to the vector lines. Therefore, it is more suitable for an application to use vector lines where further processing such as editing is required.

2.2.1 Vector-Based Stylization

We investigate vector-based stylization in two aspects: representation of the vectorized line and texture mapping methods that are applied on the lines.



2.16(a) 2.16(b)

Figure 2.16: Line corrections in two steps (images courtesy of Northrup and Markosian [75]).

Representation of Vectorized Line. The simplest way to obtain vectorized lines is connecting line segments extracted from a 3D model into line chains. Although this approach is straightforward and easily implemented, the quality of the result largely relies on mesh smoothness. If the mesh is not smooth enough, the lines may look unpleasant (see Figure 2.15 for example).

An alternative approach is to fit the line as a stroke, on which certain properties, such as thickness, can be applied. A few well-placed strokes often express the shape of a figure much more elegantly than a crowd of shorter marks. With this in mind, many contributions have been done to improve single line rendering. Northrup and Markosian [75] simplify silhouette edges extracted from models into paths that can be used to draw long, smooth strokes. Before they perform concatenation, two correction steps are carried out that promote longer and smoother paths: first, segments are corrected by redefining their overlapped endpoints to the midpoint between them. The angle between the segments is exaggerated for clarity (Figure 2.16(a)). Second, segments that overlap and are nearly parallel are merged; a segment is eliminated if it is adjacent and nearly parallel to another segment and shorter than that segment (Figure 2.16(b)).

Many line drawing systems improve stroke representations based on Northrup and Markosian's contribution. Winkenbach and Salesin [105] render strokes as paths of polygonal lines $[0, 1] \rightarrow \Re^2$, giving the overall 'sweep' of the stroke. Kalnins et al. [53] also

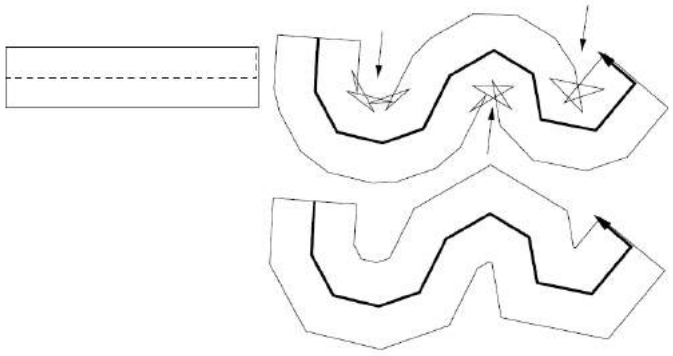


Figure 2.17: Left: stroke definition. Upper right: application of stroke by redrawing all the points along the normals. Wrinkled areas are highlighted by arrows. Lower right: the same stroke redrawn with mitre joints based on deformation model for mitre joints. Images courtesy of Hsu et al. [44].

adopt this method, except that the path is represented as a Catmull-Rom spline. Lee et al. [65] render strokes as Bézier curves, which are composed of sets of selective control points. Alternatively, Hsu et al. [44] propose a skeletal strokes which use arbitrary pictures as 'brush', and specifies how to transform the picture along the center path while maintaining the aspect ratio of selected features on the picture. See Figure 2.17 for an illustration.

Texture Mapping. There are several strategies for mapping textures onto lines. One strategy is to explore a style automatically from the scene using objects' geometry properties. Another is to use texture synthesis. If considering a line as a region in 2D space, traditional texture mapping can be used directly.

Grabli et al. [39] exploit strategies for automatic generation of line styles from 3D scenes. Style attributes are defined as global properties such as spatial distribution of strokes or the path each stroke follows in the drawing. One important attribute is stroke topology, which is the path followed by the artist's tool while drawing a single stroke without lifting the 'pen' (illustrated in Figure 2.18).

Figure 2.18: Stroke topology (images courtesy of Grabli et al. [39]). A planar graph (left). Middle and right show the same path in the graph drawn with the same low-level attributes but with three strokes and one stroke respectively.

Kalnins et al. [53] apply the policies of *Stretching* and *Tiling* along with 1D texture synthesis adopting Markov random field to reduce repetitions. Stretching policy stretches or compresses the texture so that it fits along the path a fixed number of times. The stretching policy produces high motion coherence under animation, but the stroke texture however will lose its character if the path is stretched or shrunk too much. Tiling policy establishes a fixed pixel length for the texture, and tiles the path with as many instances of the texture as can fit. During animation, however, the texture appears to slide on or off the ends of the path similarly to the shower door effect [74].

Praun et al. [82] propose an alternative way to the stretching and tiling policies for texture synthesis. The method defines a new texture structure called *artmap*, which is a set of N textures, where each texture has a particular target length in pixels. At each frame, the texture with the target length closest to the current path length is selected and drawn. This mechanism ensures that the brush texture never appears too exaggeratedly stretched.

2.2.2 Raster-Based Stylization

Lines from image-space methods result in edge pixels together with stylization effects. Raster-based stylization is simple to produce with natural coherence of *Level-of-Details*

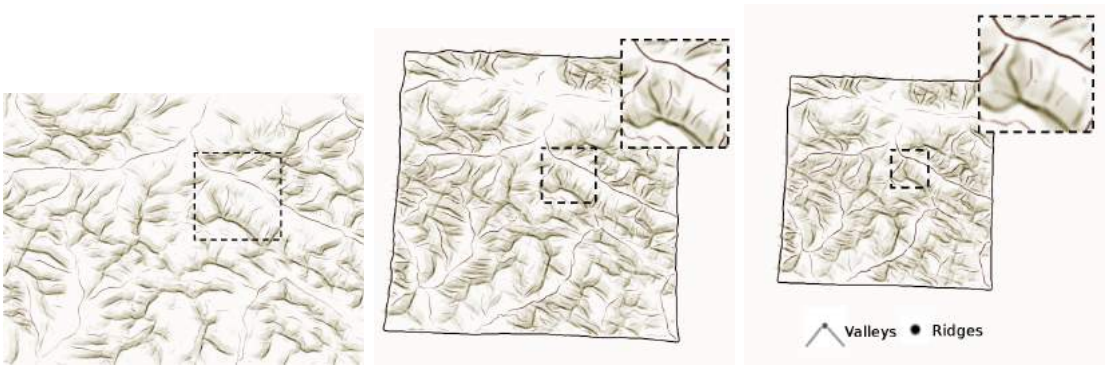


Figure 2.19: Illustrations of implicit brushes (images courtesy of Vergne et al. [100]).

(LOD) in 2D, but it supports limited style options due to the rasterized output. Thus, image-space method is suitable for direct output of drawings. Vergne et al. [100] propose 'implicit brush' defined at a given pixel by the convolution of a brush along a skeletal line (Figure 2.19). The skeleton is obtained from automatic line extraction from rendered animated 3D scenes along with featured parameters. These parameters are then mapped to brush attributes such as orientation, thickness and opacity.

2.3 Temporal Coherence Algorithms

This section investigates the algorithms which solve temporal coherence problem for stylized line drawing animation. Stylized lines have two properties that affect coherence: the position of the line and the parametrization that defines how a style is mapped onto that line. Various techniques have been developed to ensure coherence in both line extraction and stylizing phases. We provide an overview of these techniques respectively in Section 2.3.1 and Section 2.3.2.

2.3.1 Coherent Line Drawings

Lines may undergo topology changes when they are extracted from a moving model driven by camera motion and/or mesh deformation. They thus suffer from artifacts such

as merging, splitting and popping. Establishing correspondences of the lines over an animation to enforce their temporal continuity helps to cope with topology changes to prevent artifacts. It further allows the propagation of line attributes from one frame to another. Therefore, obtaining and incorporating correct line correspondence information is the most essential step in the solution for temporal coherence. It is also necessary, because it maintains a natural line movement instead of just a moving mark that sticks to an animated surface.

Line correspondence is established based on the correspondences of mesh vertices and faces which are implicitly ensured from the temporally constant topology of a surface model through time. When the viewing direction or orientation is changed, view-independent lines such as ridges and valleys remain unchanged. Thus, their correspondences have been ensured and they are already coherent. In contrast, view-dependent lines such as silhouettes, suggestive contours, and apparent ridges move over the surface and have different geometry for each viewpoint. Thus, their correspondences are unknown. When a model is deforming frame by frame, neither view-independent nor view-dependent lines could main the same locations on the surface or maintain the same geometry for each frame. Thus, both correspondences are unknown. In the following sections, we introduce existing coherence algorithms by categorizing them based on correspondence strategies.

Coherence Based on Triangle Correspondence. This is the simplest and the most straightforward correspondence which is directly obtained from the underlying constant topology of a mesh model. It is suitable if the stroke style does not require temporal coherence.

Bourdev [13] represents silhouette segments as a list of particles, which maintains local information such as location and style parameter, and they try to 'survive' and transfer this information across frames (segment definition c.f. Section 2.1.1, line tracing).

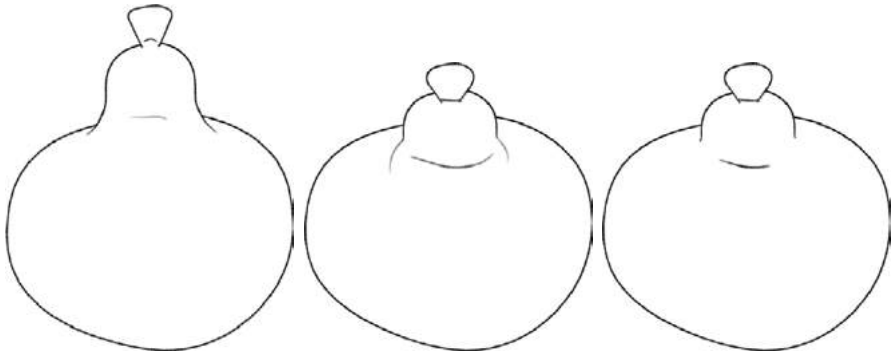


Figure 2.20: Effect of fading based on stroke speed (images courtesy of DeCarlo et al. [26]).

When the particle contains a segment that becomes non silhouette, it tries to find an appropriate substitute particle that is nearby on the screen plane. This correspondence is built through *ID reference image*, which is a frame buffer that encodes unique IDs of segments and faces of the mesh. Searching for the current particle starts from the middle pixel of the previous particle and goes along its perpendicular direction towards the current silhouette.

DeCarlo et al. [26] propose a method which is a direct extension of suggestive contours [27]. The method processes each segment of suggestive contours in two steps over an animation to achieve coherence. Firstly, the method searches for some abrupt temporal changes originated from instabilities in the line extraction with respect to small changes in the viewpoint for trimming. The speed of the motion of segments is analyzed over the surface and those moving too fast are discarded. Secondly, the method renders each segment with controlled light luminance, which works like 'fading' effect to achieve visually smooth effects. While binary black-and-white rendering might produce visual flickering, gray-colored rendering under this scheme looks visually smoother between consecutive frames, because the color transition is less conspicuous. Figure 2.20 shows an example of coherent suggestive contours using this method. The fading scheme is later applied to many other line drawing algorithms to improve their visual coherence. However, this

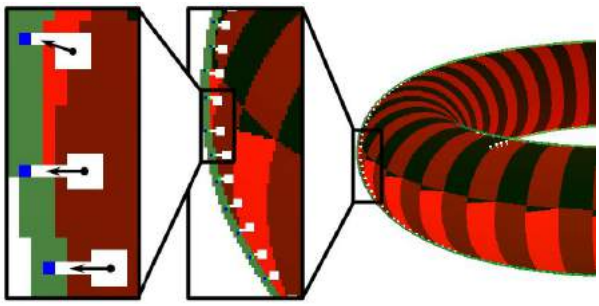


Figure 2.21: Sample propagation shown in successive enlargements of the ID image for a torus. Silhouette path IDs are green shades. Mesh faces are (striped) shades of orange. Search paths are marked in white. Blue pixels mark the successful termination of searches (images courtesy of Kalnins et al. [52]).

trimming scheme is hard to reuse on other kinds of lines. Also, the resulting lines from this method would be difficult for further stylization.

Coherence Based on Path Correspondence. In order to support stylization, most temporal coherence algorithms focus on using path correspondence. A *path* is an ordered list of edges linked through the connectivity information from triangle meshes [47, 75]. Simple heuristics on distances and angles are defined to solve ambiguities at path intersections. These linking algorithms may however be unstable when the viewpoint is changed or the mesh is deformed, i.e., the total number of paths and length of each path cannot guarantee to be the same for one-to-one correspondence. The lack of correspondence in paths from frame to frame leads to temporally incoherent animations.

The ID reference image mentioned above also appears commonly in some early work of NPR algorithms for path correspondence. Markosian [70] is the first to propose a path propagation uses ID reference image. Later the approach is adopted by many NPR systems such as WYSIWYG [53] and Gaze [20] for propagation.

Kalnins et al. [52] propose the first complete method to solve the problem of coherent stylized silhouettes. The correspondence of silhouette is built upon the ID image, assuming that current silhouette path should locate where nearby in screen space of previous silhouette path and nearby on the mesh, under arbitrary but moderate camera motion

Figure 2.22: Snaxels evolve on surface (images courtesy of Karsch and Hart [55]).

or mesh deformation. This is a safe assumption because silhouette is naturally coherent in screen space. The method thus tracks the silhouette pixel in nearby space for each line in each frame obtained from the ID image as illustrated in Figure 2.21. Besides, the method divides each silhouette path into several portions where each portion is called a brush path. Using brush paths ameliorates the swimming problem because stylization is more localized. This approach works well with silhouettes from simple models, where the number of the silhouettes is just a few, and the mapping between object-space and image-space silhouettes is almost one-to-one. However, ID image is not able to handle the matching of lines for models with moderate or high complexity such as the Stanford Bunny [97] because the models would produce too many line fragments that crisscross in the image space.

Recently some researchers apply *active contour* (a.k.a *snakes*) for the establishment of the correspondence of contours over an animation. Active contour is originally used for describing an object outline from a possibly noisy 2D image, through minimizing an energy associated to the current contour as a sum of an internal and external energy [56].

Karsch and Hart [55] propose *snaxels*, which are closed active contours that minimize an energy on a 3D meshed surface. They track snaxels on the 3D object to reproduce fragmental 3D contours rather than re-extracting them at each frame (see Figure 2.22). This method avoids propagation of parametrization and mapping is settled automatically from the snakes.

Bénard et al. [12] propose *active strokes*, which approximate lines extracted from a 3D model and iterate like snakes in screen space (see Figure 2.23). They also present several

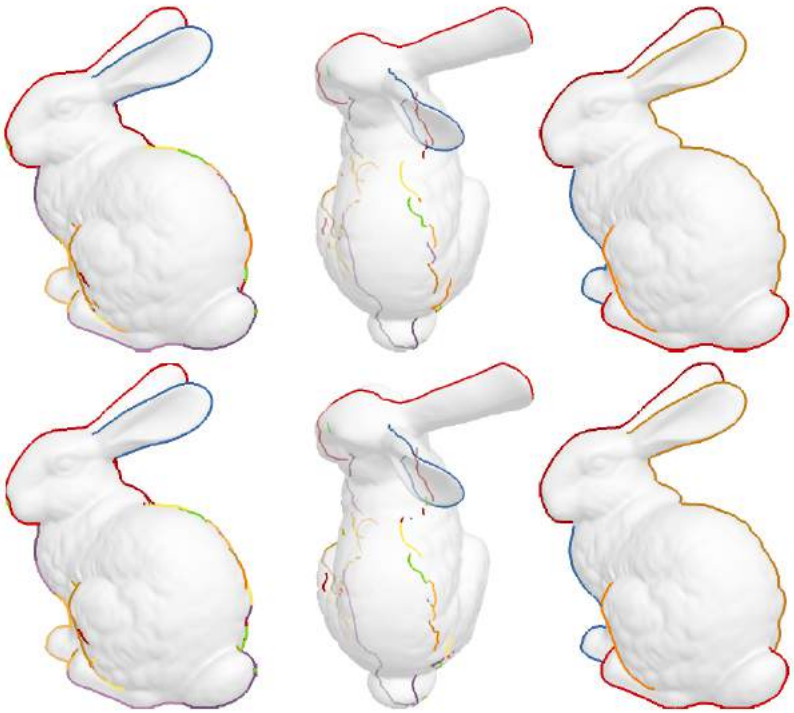
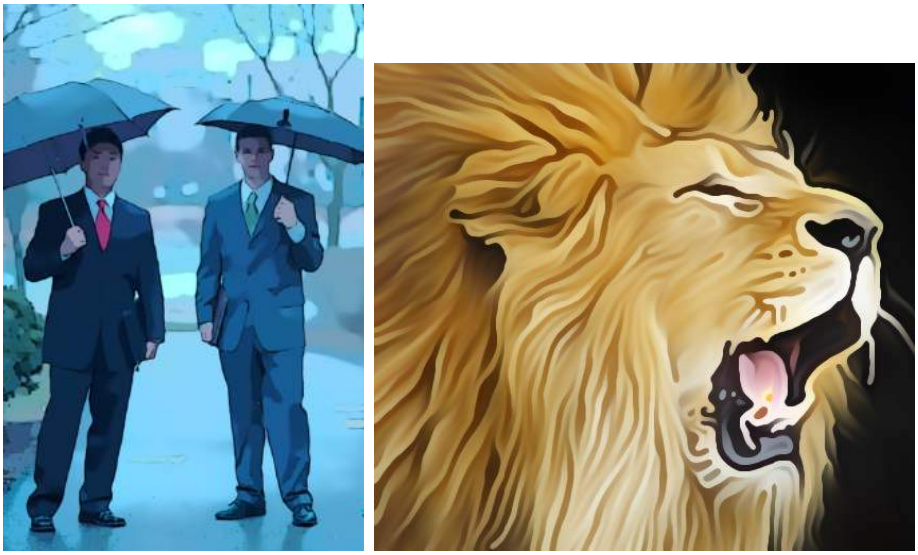


Figure 2.23: Active strokes throughout frames (images courtesy of Bénard et al. [12]). Left column: silhouette in 3D. Middle: side view of the 3D silhouettes. Right: snakes.



2.24(a) 2.24(b)

Figure 2.24: The result images from (a) bilateral filtering (image courtesy of Winnemoller et al. [107]) and (b) coherence-enhancing filter (image courtesy of Kyprianidis et al. [62]).

new functions to these strokes such as add, remove, trim, extend, split and merge for matching purpose. This method guarantees coherent stylization in screen space through active strokes along with the functions. Both methods are effective in processing models with moderate complexity that produce fragmental line segments, and are appropriate for both off-line animation as well as real-time application. However, it is hard to extend snakes to feature lines other than the contours, since they have no continuity as the contours and would accumulate error through each iteration of the snakes.

Image Processing. Image-space methods produce lines made of independent, unconnected pixels, offering neither correspondence nor parametrization across frames. To address this limitation, Agarwala et al. [5] propose a direct method to explicitly track lines from pixels in videos with Bézier curves. These curves can serve as a scaffold to animate brush strokes drawn by the artist. The system is however designed for off-line application and relies on heavy weight optimization and manual correction.

Various approaches have been proposed to produce cartoon animations from images

based on edge-preserving smoothing and enhancement filters. Winnemoller et al. [107] combine bilateral filtering with soft color quantization and difference of Gaussians edges to create cartoon-like effects with good temporal coherence (Figure 2.24(a)). Soft quantization produces results with higher temporal coherence than hard quantization. Kyprianidis et al. [62] propose a coherence-enhancing filter based on adaptive line integral convolution in combination with directional shock filtering (Figure 2.24(b)). The filter is guided by a flow field derived from the structure tensor to preserve the overall image structure and achieve a consistent level of abstraction.

Vergne et al. [100] propose a framework to combine image processing to extract feature skeletons from a height function (differential geometry on view-centred buffers such as surface gradient and curvature) and cubic polynomial fitting to obtain feature profiles. The profiles are rendered with stroke style implicitly defined as a spatially varying convolution to ensure flatness and produce temporally continuous stylized lines in real time.

2.3.2 Coherent Line Stylization

The purpose of coherent stylization is to translate the correspondence information from the line extraction step into coherent parametrization for each path, and apply texture onto the parameterized path to create stylized images. After screen projection and window clipping, paths need to be further parameterized (re-sampled) in each frame using screen arc-length. The path may however be unstable under change of viewpoint or deformation of mesh, i.e., the length of the paths cannot guarantee to be the same for one-to-one correspondence to allow a broad range of styles to be applied onto the paths.

Kalnins et al. [52] propose a coherent silhouette through its geometry, where the entire processing is illustrated in Figure 2.25. In the figure, one or more brush paths are generated along each silhouette. At each frame f_i, evenly spaced samples are taken for

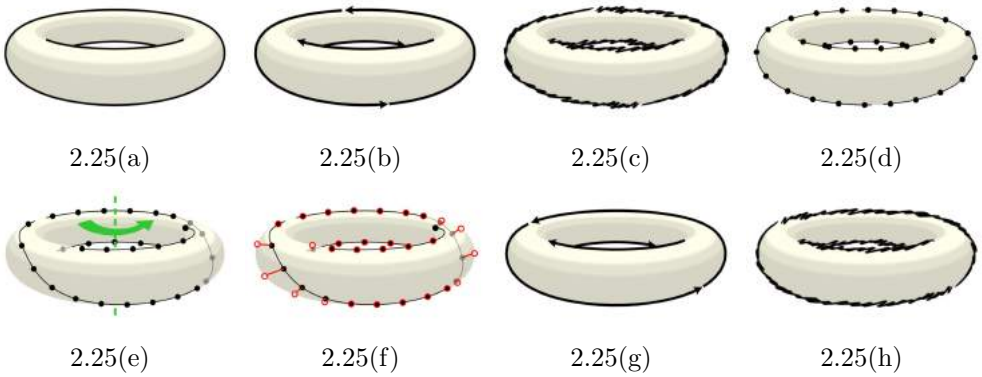


Figure 2.25: Overview of 'Coherent Stylized Silhouettes' (images courtesy of Kalnins et al. [52]). The input silhouettes (a) are split into continuous brush paths, parameterized and textured (b-c). This parameterization is sampled and propagated by reprojection in the new camera (d-e) and local search in 2D (f). New coherent brush paths are recovered from the votes (g); their parameterization is optimized to compromise between uniformity in 2D and coherence in 3D.

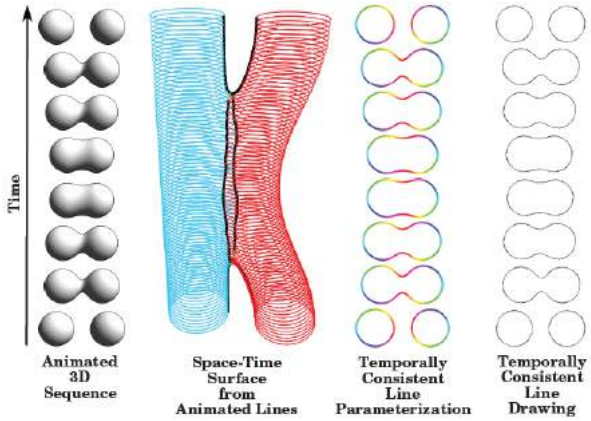
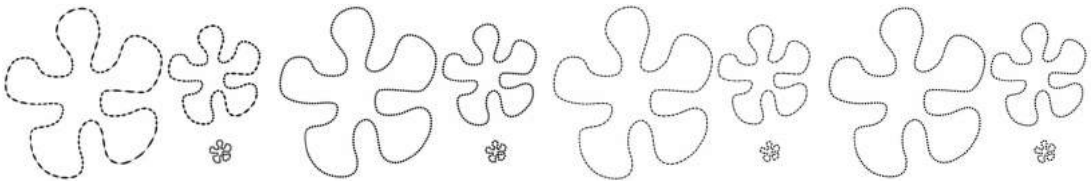


Figure 2.26: A temporally coherent parameterization for line drawings is computed by parameterizing the space-time surface swept by the line over time (images courtesy of Buchholz et al. [14]).



2.27(a) Stretching 2.27(b) Tiling 2.27(c) Artmap 2.27(d) SLAMs

Figure 2.27: Three strokes texture mapping policies. (a) Stretching ensures coherence during motion but deforms the texture. Conversely (b) tiling perfectly preserves the pattern, but produces sliding when the path is animated. (c) Artmap [82] avoids both problems at the price of fading artifacts because the hand-drawn texture pyramid is usually too sparse. (d) Self-Similar Line Artmap [10] solves this problem. Images courtesy of Bénard et al. [10].

each brush path. Each sample contains information including locations on the 3D mesh, brush path ID and parametrization for stylization. In subsequent frame f_{i+1}, sampled brush paths of silhouette are registered to propagate these information.

Buchholz et al. [14] propose a method to directly create path trajectories from extracted lines over an animation where temporal coherence is defined as a least-square optimization problem. The path trajectories form a space-time surface swept by a 2D line during the animation, from which a least-square optimized parameterization is built to generate stylized silhouettes (see Figure 2.26). However, the entire animation has to be known ahead of time. Thus, the method can only be used for off-line animation rather than real time application.

Bénard et al. [10] propose an example-based artmap synthesis approach called *Self-Similar Line Art Maps* (SLAMs) that generates an arbitrarily dense artmap based on a single exemplar. Unlike artmap [82], SLAMs allow for infinite zoom without visible interruption. It is especially useful for removing sliding (see Figure 2.27).

2.3.3 Evaluation of Coherence

Evaluation of temporal coherence is a longstanding question in NPR field, which until now most researchers only evaluate their methods through visual perception [9]. Various

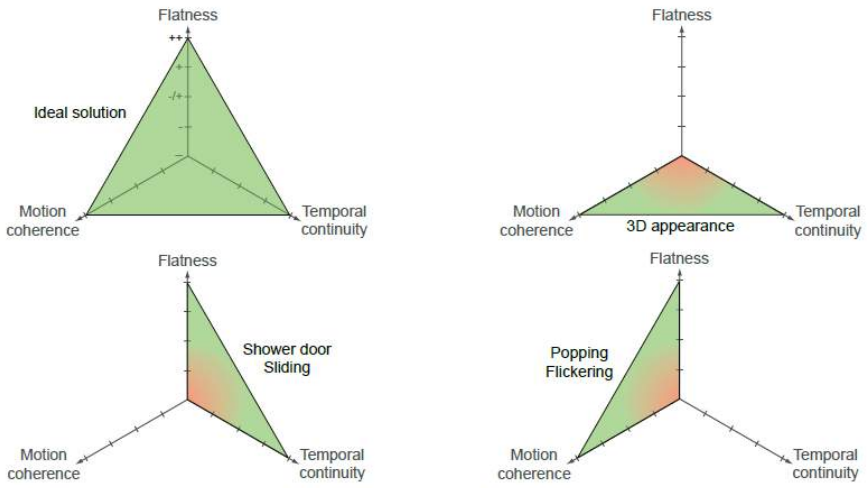


Figure 2.28: The temporal coherence problem involves three goals represented by the axes of these diagrams. Fully ignoring one of them produces the opposite artifacts (images courtesy of Bénard et al. [9]).

approaches have been proposed to characterize the perceptual impact and effectiveness of specific NPR techniques, such as human facial illustrations [37, 101], NPR abstractions [89], pen-and-ink line drawings [48], and line drawings from 3D models [21, 22]. However, no ground truth or quantitative evaluation has been proposed for temporal coherence of NPR animations.

In spite of lacking quantitative evaluation for coherence of NPR animations, temporal coherence has specific artifacts that a subject can observe. Perceptual measurements of these artifacts may give an indication of the performance of a specific solution. Starting from these perceptual measurements, objective metrics could be derived to assess qualitative evaluation of visual perception automatically.

Perceptual studies [90, 115] show that human observers are very sensitive to sudden temporal variations such as popping and flickering. The visibility and attributes of the marks should vary smoothly to ensure temporal continuity of the animations, which minimizes abrupt changes of the marks from frame to frame.

Benerd et al. [12] propose a formulation of temporal coherence problem of stylized

animation in terms of three goals: *motion coherence*, *temporal continuity* and *flatness*. These goals are represented in Figure 2.28 as three axes of a radar chart whose scales are purely qualitative. They argue that these goals are inherently contradictory. Thus, solutions are often compromised and neglect one or more criteria.

Meier [74] discusses the flatness by the existence of *shower door* effect when generating NPR drawings from 3D objects of a scene. 'Shower door' effect is when an animation appears as if it were being viewed through textured glass because brush strokes have no correlation with the motion of the scene. They look as if they stick to the view plane, not to the animating surfaces. Flatness gives the impression that the image is drawn on a flat canvas rather than painted over the 3D objects. Though the shower door effect ensures flatness, which is a key ingredient in generating computer animations that appear similar to traditional hand-drawn animations, it produces sliding artifacts and gives the impression that the scene is observed through a semi-transparent layer of strokes.

2.4 Animation System

Ed Catmull [17] is one of the first to discuss cel animation under computer assistance. The goal of computer assistance is mainly to relieve human labors from painstakingly drawing each animation frame. Inbetweening is one of major tasks in the animation that need assistance. However, lacking 3D information makes the computer hard to process such a task. Later, Curtis [25] in a well-known conference course brings up an idea that 3D computer animation is helpful in 2D animation, such as free inbetweening or perspective. But he at the same time argues that the drawback of the idea is obvious as well, such as the costly setup of the 3D model. Digital animation systems have been progressing constantly. These systems are classified as 3D, 2D, 2.5D and integrated 2D/3D. In 3D animation systems, the input model is 3D and the rendering is totally 3D. 2D animation refers to the traditional animation, where the artists drawings and computer generated

lines are either 2D vector based or pixel based. The 2.5D system is basically like 2D system, but the 3D information can be used in 2D drawings for assistance. In integrated 2D/3D systems, the initial input is 3D models, while the generated animation is a 2D animation. We will review some typical systems in the following sections.

2.4.1 3D Animation System

What-You-See-Is-What-You-Get (WYSIWYG) is proposed by Kalnins et al. [53] as a 3D stroke painting system. The system allows artists to draw NPR-style strokes directly on 3D models. Artists can simply draw strokes with 'decal' effect. The system can automatically extract and draw stylized strokes such as silhouette or creases (similar to ridge lines), or synthesize lines as the offsets based on examples provided by the artists. Several NPR styles are also supported in the system, such as toon shading and hatching. The authors who propose WYSIWYG have developed an integrated animation system prototype called JOT which is available online [50].

TextureBrush is proposed by Sun et al. [93] as an interactive 3D surface texturing system. The system allows users to draw free-form curves, and automatically generates local parameterizations in strips along the curves. The authors develop an extended exponential mapping algorithm to support the parameterization. The texture mapping is then created and mapped onto the 3D surface with anisotropy. A user study is provided to support the authors idea.

2.4.2 Paperless 2D Animation System

TicTacToon [31] is among the first paperless animation systems for professional 2D animation studios. The system provides vector based sketching and animating, allowing users to draw through a digital pen with pressure awareness. The drawings are translated into strokes of varying thicknesses in real time. Many other functionalities are integrated

into TicTacToon, such as resolution independence, dynamic management of perspective, complicated support for reuse of drawings and infinite number of layers, etc.

2.4.3 2.5D Animation System

Rivers et al. [85] present a novel structure called 2.5D cartoon model to simulate 3D rotations in cel animation. The 2.5D cartoon model is created from 2D vector drawings of a cartoon, which is easier than building a 3D model. The authors suggest three to four drawings are often enough to generate the 2.5D model. Their result shows the effectiveness of their method in producing new drawings from any viewpoint. However, their method is limited in several ways. Firstly, the method is not suitable for extreme sharp corners or highly concave/convex shapes which lead to severe distortion. Secondly, the method only supports silhouettes. For interior contours or feature lines such as ridge/valleys, it does not work. Thirdly, the method cannot solve more complicated inbetween generation, such as inbetween involving partial occlusion.

2.4.4 Integrated 2D/3D Animation System

Barbara Meier [74] proposes an effective approach to paint 3D surface as 2D brush strokes in screen space. The approach can eliminate 'shower door' effects while maintaining frame-to-frame coherence. The results in the paper are very impressive – they look like hand-made drawings. However, the technique is limited to rigid objects only, and is hard to be extended to deformable models.

Benard et al. [12] present a novel approach called active stroke to animate stylized silhouette with frame-to-frame coherence. The active strokes are vectorized 2D points sampled from 3D silhouettes. The vectorized lines are stylized with parameterizations evolved in screen space using active contour method to maintain coherence frame by frame. The resulting animations show the effectiveness of the method. However, this

method is hard to be extended for other feature lines such as ridges/valleys or suggestive contours.

2.5 Summary

In this section, we have presented algorithms related to NPR line drawing, including extraction from 2D/3D, the data structure and the texture mapping of stylizing lines, algorithms to keep lines coherent frame by frame, and current animation systems involving line drawings. The computer assistance in traditional animation is common. A number of animation systems have been developed. Integrated 2D/3D system has risen particular interest among researchers recently. However, since the time when Curtis proposes the idea of integrating 3D modeling into 2D animation, the research in this area is still focusing on how to improve the NPR result. Researchers seldom consider how the outputs rendering result could be used in real animation production but just treat the rendering result as the final result. The gap between the research and the real application in animation is huge. We will focus on this very task, trying to make the gap smaller.

Chapter 3

A Line Drawing Algorithm for Illustration

In this chapter, we propose a lighting model to enhance the shape depiction of 3D objects, and introduce an algorithm to generate line drawings from the lighting model as well as the 3D geometry of the object. The lighting model is called X-Phong lighting model, which extends the basic Phong lighting model with a curvature light to control the perceived shape and line detail. The generated line drawings are called X-Phong Lines (XPLs), which characterize more significant variations of illumination on 3D surfaces and are more flexible to achieve desirable visualization effects compared to the existing extracted lines. XPLs are applied to a wide range of real-world models to demonstrate its flexibility and effectiveness through qualitative and quantitative evaluations.

3.1 Overview

Although there are well-known photorealistic rendering (PR) techniques, such as global illumination and ray tracing, they may not be able to depict every detail on the surface in a single view. Fine structure is typically revealed only with grazing light, i.e., light incident to the surface at a very shallow angle [87]. Hence, a non-photorealistic (NPR) lighting can be used to enhance the amount of information conveyed in a single view (see

Figure 3.1: An example of shaded image from X-Phong lighting model (left) and generated line drawings (right).

Figure 3.1 for illustration). In Section 3.2, an extended Phong lighting model, X-Phong lighting model, is proposed to illuminate 3D objects. The lighting model keeps the light dependency property of the basic Phong lighting, which is highly desirable in the real-world applications, while depicting surface details through its NPR lighting component.

Many perceptual and cognitive observations show that the Human Vision System (HVS) is highly sensitive to edge-like features, i.e., points of high luminance variations [79], and perceives objects in different ways due to changes in illumination [15, 78]. Recent studies on computer-generated line drawings [21, 22] also demonstrate that edge lines show the strongest cue for conveying shapes. In Section 3.3, X-Phong Lines (XPLs) are proposed as a type of edge-like lines, generating from the surface illumination using X-Phong lighting model as well as the 3D geometry. XPLs are extracted via two stages. Firstly, the intensity of each vertex on a 3D model is calculated based on the X-Phong lighting model. Secondly, XPLs are extracted by detecting local extremum of intensity variation on the surface of the 3D model. We adopt the method used in Xie et al. [109] to detect such extremum from 3D objects.

3.2 X-Phong: Extended Phong Lighting

X-Phong lighting enhances depiction of surface geometry from the basic Phong model in salient features. It contains three main terms: ambient light $\mathbf{I_a}$, reflection light (a point source light) $\mathbf{I_r}$ and curvature light $\mathbf{I_c}$. X-Phong lighting model is defined as:

$$\mathbf{I} = \mathbf{w_a I_a} + \mathbf{w_r I_r} + \mathbf{w_c I_c} \tag{Eq. 3.1}$$

which is extended with surface geometry properties as:

$$\mathbf{I} = \mathbf{w_a k_a I_a} + \mathbf{w_r}(\mathbf{k_d I_d}(\mathbf{n} \cdot \mathbf{l}) + \mathbf{k_s I_s}(\mathbf{v} \cdot \mathbf{r})^{\mathbf{n_s}}) + \mathbf{w_c k_c I_c}(\mathbf{1} - \boldsymbol{\kappa}) \tag{Eq. 3.2}$$

where $\mathbf{n}$ and $\mathbf{l}$ denote surface normal and lighting vector respectively. $\mathbf{v}$ denotes viewing vector. $\mathbf{r}$ denotes reflection vector, which can be computed from the directions for $\mathbf{l}$ and $\mathbf{n}$. $\mathbf{n_s}$ represents the specular reflection exponent. $\boldsymbol{\kappa}$ denotes the mean curvature of every vertex. $\mathbf{k_a}$, $\mathbf{k_d}$, $\mathbf{k_s}$, $\mathbf{k_c}$ are the coefficients of the ambient light, diffuse light, specular light and curvature light and they all depend on surface materials. $\mathbf{I_a}$, $\mathbf{I_d}$, $\mathbf{I_s}$, $\mathbf{I_c}$ denote the respective intensities. $\mathbf{w_a}$, $\mathbf{w_r}$, $\mathbf{w_c}$ are respective weights of $\mathbf{I_a}$, $\mathbf{I_r}$ and $\mathbf{I_c}$. All the weights are normalized to sum up to 1.

Aiming at 3D shape illustration with a focus on human perceptual experiences, lighting models should emphasize illumination variations strongly related to shape variation. In X-Phong model, the three lights, ambient light, reflection light and curvature light, are suitable for extracting line drawings with reasonable detail but of different importance to the contributions of shape depiction. The ambient light is set to a uniform brightness. The reflection light (a point source light) is set to obtain the basic shape. The curvature light is set to describe the detail of surface geometry. In the following sections, we will explain these three lights accordingly.

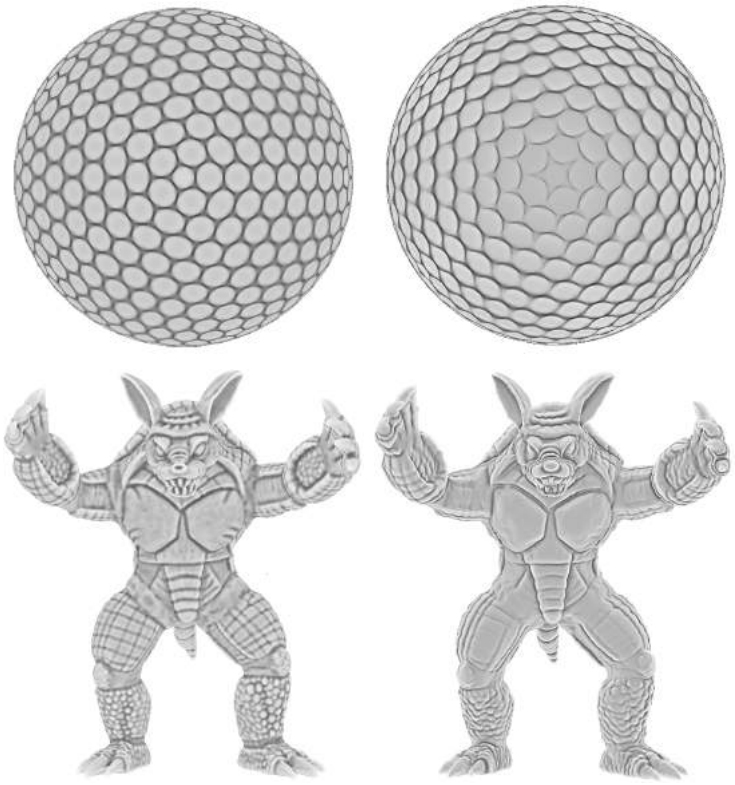


Figure 3.2: Comparisons between two curvatures for curvature light: left column uses mean curvature while right column uses radial curvature.

3.2.1 Ambient Light

Ambient light models light of uniform brightness of an entire object caused by multiple reflections of light from surface presents in the environment. Considering the ambient light alone, the illustration of the object appears as monochromatic silhouette since the intensity does not depend on surface geometry.

3.2.2 Curvature Light

Curvature light is a detail light which may enhance surface illustration. It is the main light among the X-Phong lights to depict surface features. Nevertheless, it is a perceptual light that does not exist in the real physical world. Since we want to shade a surface dark where the curvature of the surface is high, we use the inverse curvature $1 - \boldsymbol{\kappa}$ as

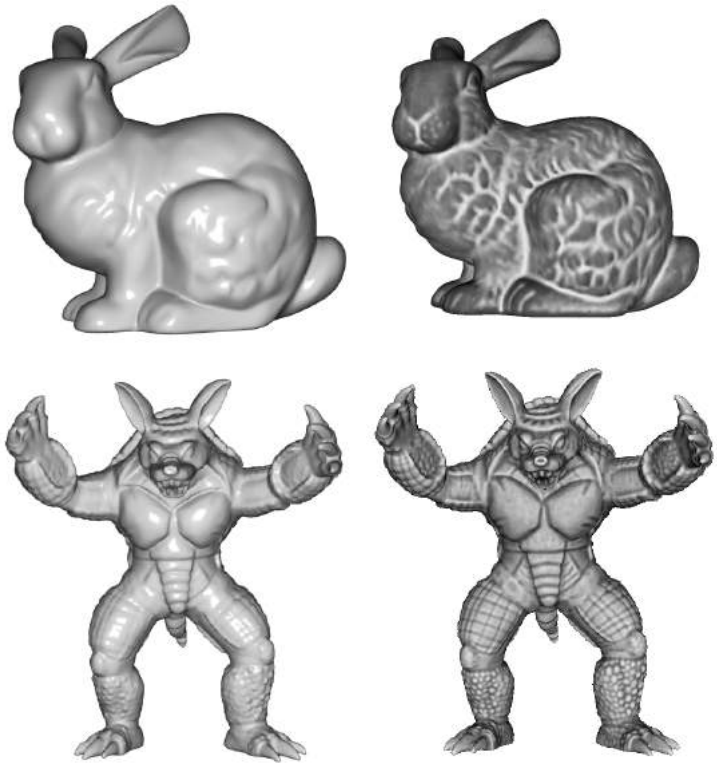


Figure 3.3: Comparisons between Phong light (left column) and X-Phong light (right column). X-Phong light could enhance surface feature compared to Phong light.

intensity value.

The curvature $\boldsymbol{\kappa}$ is the mean curvature of two principal curvatures, which are view-independent geometric properties. There are two reasons for using mean curvature. Firstly, the local surface geometry is argued to be emphasized only by curvatures of principal directions, i.e., the variation of the surface normal reaches maximal in absolute value [33]. Secondly, view-dependent curvature, such as radial curvature used by DeCarlo et al. in [27], creates many flickering lines during camera motions, which is undesirable for line drawings. Figure 3.2 demonstrates that curvature light using mean curvature could describe features of 3D shapes better than that using radial curvature.

3.2.3 Diffuse and Specular Light

An effective and simple way is to use lambertian diffuse reflections for X-Phong model to extract line drawings. This is also applied in PELs [109] and GPELs [118].

In X-Phong lighting model, the specular light works as an auxiliary light for shape depiction. Therefore, we wish to simplify the computation of the specular light. In our implementation, the specular light is defined as $\mathbf{k_s I_s}(\mathbf{n} \cdot \mathbf{h})^{\mathbf{n_s}}$, which is a simplified Phong specular model [40]. $\mathbf{h}$ is the halfway vector between the lighting vector $\mathbf{l}$ and viewing vector $\mathbf{v}$ to calculate the range of specular reflection, which is obtained as $\mathbf{h} = \frac{\mathbf{l}+\mathbf{v}}{|\mathbf{l}+\mathbf{v}|}$. The exponent $\mathbf{n_s}$ is fixed and set to a small value (≤ 10), so that the surface region covered by the specular light is lit up uniformly, which is better for shape depiction.

Figure 3.3 shows the comparison between two exemplar models lit by Phong and X-Phong lights, where X-Phong is a summation of lambertian Phong and curvature light.

3.3 X-Phong Lines (XPLs)

Based on X-Phong model, we are able to extract variations of illuminations and render them as line drawings. Section 3.3.1 briefly describes the mathematical definition of XPLs. Section 3.3.2 analyzes XPLs via their properties of light and view dependencies.

3.3.1 Mathematical Definition

Given a smooth surface, XPLs are a loci of points on the surface where variation of illumination in the direction of gradient reaches local extremum:

$$\mathbf{D_g} \parallel \boldsymbol{\nabla}\mathbf{f} \parallel = 0 \qquad \text{(Eq. 3.3)}$$

where the gradient of illumination $\mathbf{f}$ is denoted by $\boldsymbol{\nabla}\mathbf{f}$. $\mathbf{g}$ refers to the unit vector of $\boldsymbol{\nabla}\mathbf{f}$, i.e., $\mathbf{g} = \frac{\boldsymbol{\nabla}\mathbf{f}(\mathbf{p})}{\|\boldsymbol{\nabla}\mathbf{f}(\mathbf{p})\|}$.

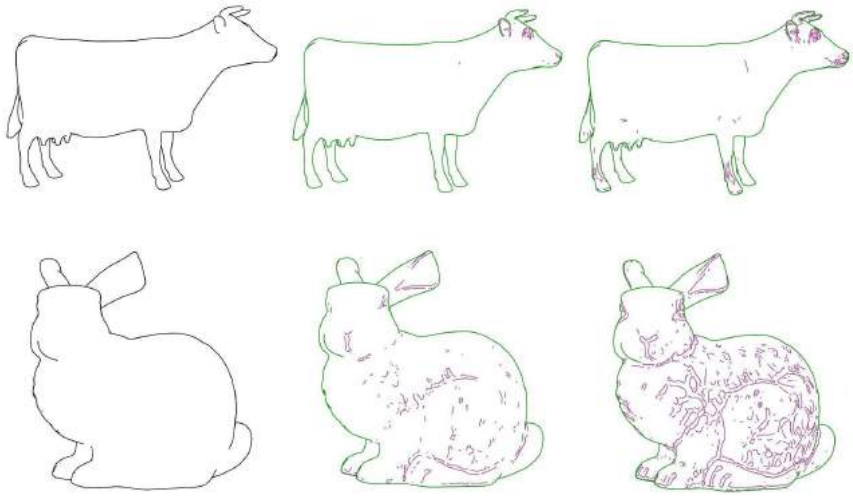


Figure 3.4: XPLs can be drawn with controllable details. (a) Contours only, the three light weights $\mathbf{w_a}$, $\mathbf{w_r}$, $\mathbf{w_c}$ are $1.0, 0.0, 0.0$. (b) Contours and a few details are added (drawn in different colors) when we increase $\mathbf{w_r}$, here the weights are $1.0, 0.7, 0.0$. (c) Contours and enhanced local features are added when $\mathbf{w_c}$ is increased, e.g., eyes, paws and back, here the weights are $1.0, 0.0, 0.7$.

Because of the three levels of information illumination given by the point light, ambient light and curvature light, XPLs are capable of illustrating controllable details. Figure 3.4 shows examples of XPLs describing the basic profile of 3D shapes as well as enhanced salient features. For the rest of our experiments, we set $(0.1, 0.2, 0.7)$ as the default values of the weights $\mathbf{w_a}$, $\mathbf{w_r}$, $\mathbf{w_c}$ respectively.

3.3.2 Light and View Dependency

Lighting affects the perception of shapes. When there is a fixed angle between the lighting and viewing directions, the best perception of illustration is achieved [92]. It is customary to locate the primary light at the upper left front at about 45° to the object [43]. Therefore, for the best illustration, we set the angle to be 45° between the initial position of the light rays and the viewing vector and assume that the light rays always change together with the viewing vector. This models a grazing light under changing viewpoints. As a result, the light moves when the viewpoint changes. Therefore, the

Table 3.1: Performance of XPLs. ♯△: number of triangles. The third and fourth columns show the rendering speed of Photic Extremum Lines (PELs) and X-Phong Lines (XPLs) in *Seconds/Frame*. PELs do not include auxiliary light and optimization. The fifth column is the overhead rendering time of XPLs to PELs, where $overhead = \frac{XPL-PEL}{PEL} \times 100\%$.

Model	♯△	PELs	XPLs	Overhead
Bunny	$144K$	0.297	0.299	0.67%
Golfball	$246K$	0.493	0.498	1.01%
Armadillo	$346K$	0.735	0.740	0.68%
Lion	$367K$	0.765	0.770	0.65%
Lucy	$526K$	1.125	1.135	0.89%
Brain	$588K$	1.200	1.220	1.67%

X-Phong lighting model is view/light dependent, which is highly desirable in the real 3D world application.

3.4 Evaluations and Discussions

3.4.1 Performance

Our algorithm runs on a 2.4GHz Intel Xeon CPU with 2GB RAM. 0.299 to 1.220 seconds per frame can be achieved with testing surfaces ranging from 144K to 588K triangles, as shown in Table 3.1. The intensity of each vertex is computed by CPU and standard OpenGL library is used for rendering. Users can change the threshold on the result through an interface similar to existing methods like PELs.

3.4.2 Qualitative Evaluation

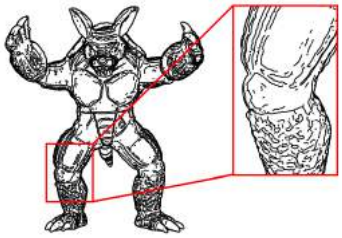
In this section we perform qualitative evaluation of our proposed line drawing algorithm by comparing it with the existing algorithms. Table 3.2 summarizes the properties of the surface geometry that the existing algorithms use.

Comparison to contours. Contours are fundamental types of lines: discontinuities in surface depths. They are loci of points which capture surface outline, but are quite

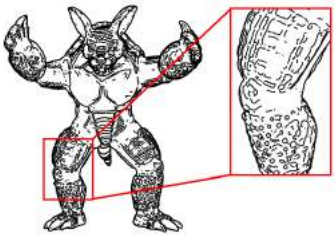
3.5(a) Phong shaded image

3.5(b) Enhanced Shaded image

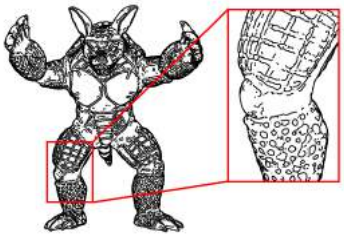
3.5(c) X-Phong Shaded image



3.5(d) PELs



3.5(e) GPELs

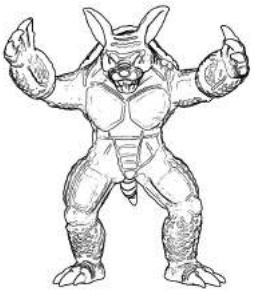


3.5(f) XPLs

3.5(g) Suggestive Contours

3.5(h) Ridge-Valleys

3.5(i) Apparent Ridges



3.5(j) Laplacian Lines

Figure 3.5: Top two rows display comparison between PELs, GPELs and XPLs and their respective shaded images. It shows that XPLs enhance local surface features thus giving a much clearer depiction. Bottom row displays suggestive contours, ridge-valley lines, apparent ridges and laplacian lines as comparisons to XPLs.

limited in describing interior details. As shown in Figure 3.4, contours (colored in green) fails to portray the whole object.

Comparison to suggestive contours, ridge-valley lines, apparent ridges and laplacian lines. The ability to convey shapes in detail is largely affected by the input geometry information. For suggestive contours which use second order derivatives, the information to draw lines comes from surface normals and curvatures, to locate the points which reach local minimum. For third order based algorithms, such as ridge-valley lines, apparent ridges, and laplacian lines, the information comes from surface normals, curvatures and laplacian operator of normals (its information entropy equals to curvatures) and their derivatives, which draws more details on the surface. However, all information comes from the surface geometry (laplacian operator for laplacian lines is computed geometrically as well). In contrast, much more information is involved when generating XPLs which enables us to draw more general lines.

Comparison to PELs and GPELs. For a smooth surface, PELs draw points from illumination model which is rendered by basic Phong $\mathbf{I} = \mathbf{k_a I_a} + \mathbf{k_d I_d}(\mathbf{n} \cdot \mathbf{v}) + \mathbf{k_s I_s}(\mathbf{v} \cdot \mathbf{r})^{\mathbf{n_s}}$. In most cases, PELs move ambient and specular light out of their lighting model and use a main light alone. Hence, the actual light that PELs use is $\mathbf{I} = \mathbf{k_d I_d}(\mathbf{n} \cdot \mathbf{v})$ (they also replace the lighting vector $\mathbf{l}$ with the viewing vector $\mathbf{v}$).

GPELs draw points from diffuse light $\mathbf{I}$, with a scaling factor related to the surface principal curvature $\mathbf{k_u}$ and $\mathbf{k_v}$. The enhanced illumination is computed as $\mathbf{I'} = \mathbf{I} \cdot (\boldsymbol{\lambda}_\mathbf{u} \boldsymbol{\lambda}_\mathbf{v})^\gamma$, where $\boldsymbol{\lambda}_{\mathbf{u|v}} = \mathbf{tan}(\mathbf{arctan}(\boldsymbol{\alpha} \mathbf{k_{u|v}})/6 + \boldsymbol{\pi}/4$ and $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ are set to 1 usually. Hence, GPELs use NPR lighting alone. However, as we discuss in Section 3.1, a lighting model is more powerful with both PR and NPR lightings for 3D illustrations.

XPLs are able to generate more desirable line drawings than PELs while they cost almost the same computation time (Table 3.1). This is because the curvature lighting is view-independent and can be pre-computed.

Table 3.2: Comparisons of differential geometry used by existing line drawing algorithms and XPLs.

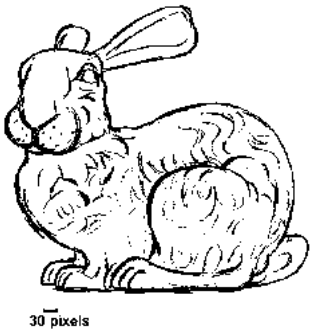
Algorithm	Equation	Deriv.	Dependency
Contours	$\mathbf{n} \cdot \mathbf{v} = 0$	1^{st}	view
Suggestive Contours	$\mathbf{D_w}(\mathbf{n} \cdot \mathbf{v}) = 0$	2^{nd}	view
Laplacian Lines	$\triangle \mathbf{I} = 0$	3^{rd}	light, view
Ridge-Valleys	$\mathbf{D_{e_{max}}}\mathbf{k_{max}} \parallel \mathbf{D_{e_{min}}}\mathbf{k_{min}} = 0$	3^{rd}	N/A
Apparent Ridges	$\mathbf{D_{e'_{max}}}\mathbf{k'_{max}} = 0$	3^{rd}	view
PELs	$\mathbf{D_g}\|\boldsymbol{\nabla}\mathbf{I}\| = 0$	3^{rd}	light, view
GPELs	$\mathbf{D_g}\|\boldsymbol{\nabla}\mathbf{I}'\| = 0$	3^{rd}	light, view
XPLs	$\mathbf{D_g}\|\boldsymbol{\nabla}\mathbf{I}''\| = 0$	3^{rd}	light, view

For illustration purposes, it is desirable to draw salient feature lines as much as possible. However, PELs are quite limited in describing such features in detail. For example, the details on the leg of the armadillo is hard for PELs to detect (Figure 3.5(d)) because the illumination variance is not as obvious as the variance of other region, e.g., face and paws. GPELs draw more details than PELs, nevertheless, it cannot draw lines as clear as XPLs and some details are lost (Figure 3.5(e)). In contrast, the shaded image from XPLs (Figure 3.5(c)) enhances features on leg regions and achieves a much clearer depiction (Figure 3.5(f)).

3.4.3 Quantitative Evaluation

We adopt a similar method proposed in Cole et al. [21] to quantitatively evaluate consistencies between artist's drawing and computer-generated line drawings. The consistencies are quantified by overlaps of pixels. The hand-made illustrations for tested models are drawn by artists in our project group, as well as the open data of drawings from Cole et al. [21].

To calculate the overlapping pixels, computer-generated drawings need to be registered to the artist's drawings. This is done in two steps. Firstly, all the feature lines are

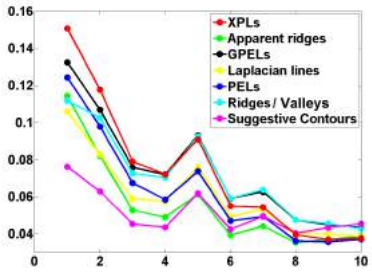


3.6(a)

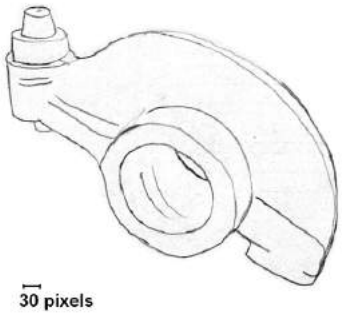
3.6(b)

3.6(c)

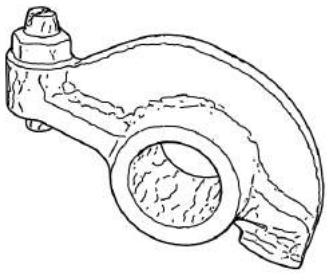
3.6(d)



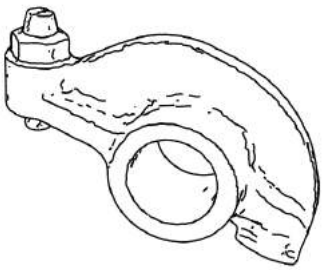
3.6(e)



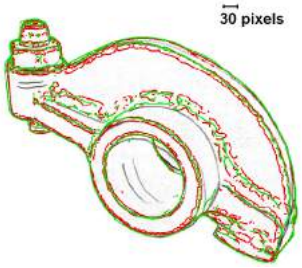
3.6(f)



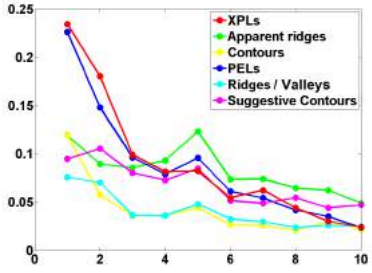
3.6(g)



3.6(h)



3.6(i)



3.6(j)

Figure 3.6: Similarity evaluation of hand-drawn illustrations. (a) and (c), (f) to (h) are artist drawings, XPLs and apparent ridges for bunny and rocker arm models respectively. (d) and (i) contain the superimposed drawings of the two models in different colors. (e) and (j) show pairwise closest distances between line pixels. The XPLs and the two hand-drawn illustrations have 51% and 67.8% line pixels overlapping within 5 pixels, and 76% and 91% line pixels overlapping within 10 pixels.

obtained from the same rendering environment under the same view point. Secondly, they are manually registered to artist's drawings.

Figure 3.6 shows our evaluation results. The horizontal dimensions of Figure 3.6(e)) and Figure 3.6(j) are pixels, while the vertical dimensions are the percentage in the form of number holding two digits after decimal point. As can be seen in Figure 3.6, for both models from different data sources, all the feature lines have high percentage of overlapping with the hand-drawn illustration within small pixel distances. Among these feature lines, XPLs show the best similarity to the hand-drawn illustration.

3.5 Summary

We have presented a novel and effective line drawing algorithm to convey the shapes of 3D model. Based on X-Phong lighting model which is flexible to describe profile and details, we propose a new type of lines – X-Phong Lines (XPLs) with reasonable time cost. This new type of lines can be used together with other types of lines extracted from the existing algorithms for enhanced depiction of the shapes of 3D models.

Chapter 4

Coherent Line Extractions Using 3D Hysteresis Thresholding

Line drawing animations can be produced by sequencing feature lines extracted from animated 3D models frame-by-frame. Temporal coherence problems such as flickering and popping are common for such animations. The problems are especially serious for object-space lines extracted from triangle mesh models with moderate or high geometric complexity. These lines are thus quite unsuitable for producing stylized animations as they require a lot of manual cleaning up. The key to solve such problems is to build inter-frame correspondence. A number of methods have been proposed to achieve this. Most of them only tackle silhouette correspondences, for which coherence has been achieved. A few methods tackle the correspondence of high-order lines, such as ridges/valleys and apparent ridges. However, building such correspondence is a non-trivial task and coherence has not been achieved satisfactorily. In this chapter, we propose a novel method to solve temporal coherence problems of high-order lines based on the classic hysteresis model. The method works as a generalized hysteresis thresholding in the temporal domain and on 3D models. Comparing to the existing methods, our method has two major benefits. Firstly, it requires no line correspondences but achieves temporal coherence on a per line basis. Secondly, it is generalized to handle all types of high-order lines. The resulting lines from our method are vector lines and a variety of styles can be applied to

produce 2D stylized animation. To evaluate our method, a few representative line drawing algorithms are selected and applied on a wide range of animated 3D models including real-world and man-made models with both fine and coarse triangle tessellations. The results demonstrate the effectiveness of our method in producing temporally coherent high-order lines for stylized animations.

4.1 Overview

Line drawings are traditionally used as a simple and effective method in animations. Computer generation of line drawings from animated 3D models is an alternative to the traditional key-frame technique for creating line drawing animations, which can be further stylized to produce 2D stylized animation [9]. A key challenge in this alternative approach is to maintain temporal coherence of the generated animation. Temporal coherence can be loosely described as lines changing smoothly and continuously over time, with respect to small viewpoint movement and/or model deformation [23, 74]. Temporal incoherence introduces severe visual artifacts into the animation, such as flickering and popping of lines, which are disturbing to the audience.

Simply sequencing computer-generated line drawings frame-by-frame to form an animation gives rise to temporal coherence problems. The problems are especially serious for existing object-space lines (e.g., silhouettes [42] and ridges/valleys [76] etc.) extracted from 3D triangle mesh models with moderate or high geometric complexity such as the Stanford bunny [97]. There are several reasons for this. Firstly, the number of lines that viewers can observe (i.e., visible curves) is very high. In one frame, this number can be a few hundreds or thousands. Secondly, the curves do not stay in the original geometric positions or remain the same lengths in successive frames, especially when the model is animated. The curves may randomly appear and disappear from time to time. They may also be arbitrarily broken into multiple pieces, or several curves merged into one

piece. These variations of the curves are referred as topological discontinuities, which cause temporal artifacts like popping, splitting and merging, etc.

A key to deal with such temporal coherence problems is to build correspondences for line drawings in successive frames. A number of approaches have been proposed in this research area. Among these approaches, silhouettes get the most attention. Many methods are proposed to build line correspondence for silhouettes, for which coherence has been achieved. However, these methods are suitable for silhouettes only [11, 12, 14, 52, 55]. In contrast, the coherence problems of high-order lines, such as ridges/valleys or apparent ridges, are not fully solved, although these lines have been widely used for detailed depiction of the shapes of 3D models. Existing methods for high-order lines either work on building line correspondence explicitly [10], or avoid line correspondence by processing on a per triangle basis [26]. The former method requires complicated computation and optimization. The latter method does not tackle topological discontinuity of lines. Neither of the methods is practically used for stylized animations.

In this chapter, we aim to deal with temporal coherence problems of high-order lines in object space. Inspired by the fundamental conception of *temporal dependency* of the hysteresis system [72, 73], we propose a novel method to generate high-order lines based on hysteresis model for temporal coherence. The model we adopt is a fundamental hysteresis operator called *non-ideal relays* [61, 81], which is a two-valued operator with two parameters working as thresholds. We find that the relay operator is highly suitable for achieving temporal coherence of lines due to its two properties: 1) a local memory, referring to the operator having memory of history which will be erased once its input reaches an extremum; 2) a double-threshold strategy, which prevents rapid changing of output as the input drifts around a set threshold.

Our proposed method uses the relay operator to perform some kind of hysteresis thresholding on line drawings, similar to the thresholding applied in Canny edge detection [16] but in the temporal domain. Our method includes the topological information

of lines from 3D models in the thresholding as well. Thus, the method works as a generalized hysteresis thresholding. It has two major benefits: firstly, it does not compute line correspondences directly, while tackling topological discontinuity and creates temporal coherence on a per line basis. Secondly, it is generalized to handle all types of lines. As a result, a variety of styles can be applied to produce 2D stylized animation.

4.2 Hysteresis Model

Non-ideal relay, as the simplest hysteresis operator, has been widely applied in different applications. A complicated operator can be represented as a superposition of the simplest operators. A new qualitative property emerges as a collective property of the complicated operator having large number of simple and qualitatively similar components. In this section, we first introduce the basic concept of the relay operator, followed by the complicated operator. Different applications may need to adjust the parameters/functions used in both operators accordingly.

4.2.1 Relay Operator

Non-ideal relay is a two-valued operator with two parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, denoted by $\boldsymbol{\gamma_{\alpha\beta}}$. $\boldsymbol{\gamma_{\alpha\beta}}$ can be represented by a rectangular loop on the input-output diagram (see Figure 4.1). Output $\mathbf{y(t)}$ of $\boldsymbol{\gamma_{\alpha\beta}}$ takes one of two values 0 or 1 to represent whether $\boldsymbol{\gamma_{\alpha\beta}}$ is 'switched-off' or 'switched-on'. At a time point, $\mathbf{y(t)}$ is expressed as:

$$\mathbf{y(t)} = \boldsymbol{\gamma_{\alpha\beta}}\mathbf{x(t)} = \begin{cases} 1, & \text{if } \mathbf{x(t)} > \boldsymbol{\beta} \\ 0, & \text{if } \mathbf{x(t)} < \boldsymbol{\alpha} \\ \boldsymbol{\eta}, & \text{if } \boldsymbol{\alpha} \leq \mathbf{x(t)} \leq \boldsymbol{\beta} \end{cases} \qquad \text{(Eq. 4.1)}$$

If $\mathbf{x(t)}$ is in the region of $\boldsymbol{\alpha} \leq \mathbf{x(t)} \leq \boldsymbol{\beta}$, $\boldsymbol{\eta}$ is either 1 or 0, depending on which region $\mathbf{x(t)}$ was in last time. When the input $\mathbf{x(t)}$ is monotonically increasing, the ascending branch *abcde* is followed. When $\mathbf{x(t)}$ is monotonically decreasing, the descending branch

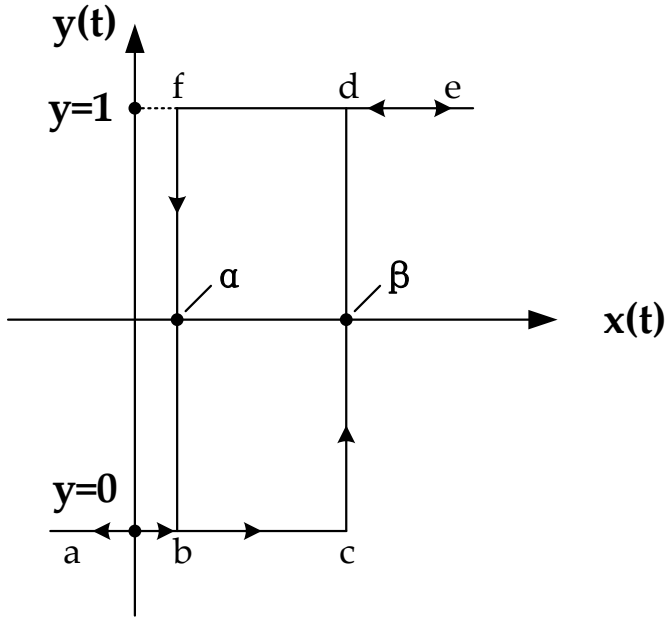


Figure 4.1: Non-ideal relay with loops.

edfba is traced. $\mathbf{y(t)}$ therefore behaves rather 'lazily': it prefers to remain unchanged, as long as $\mathbf{x(t)}$ varies within the range of $[\boldsymbol{\alpha}, \boldsymbol{\beta}]$.

4.2.2 Complicated Operator

If containing a finite number of relay operators, the complicated operator is shown as a parallel connection of relay operators. Consider each relay operator $\boldsymbol{\gamma_{\alpha\beta}}$ from the complicated operator as being endowed with an arbitrary weight function $\boldsymbol{\mu(\alpha, \beta)}$. The complicated operator can be written as follows, with each relay indexed by $\mathbf{j}$ $(1 \leq \mathbf{j} \leq \mathbf{N})$ and weighted by $\boldsymbol{\mu(\alpha_j, \beta_j)}$:

$$\mathbf{y(t)} = \mathbf{\Gamma x(t)} = \sum_{\mathbf{j=1}}^{\mathbf{N}} \boldsymbol{\mu(\alpha_j, \beta_j)} \boldsymbol{\gamma_{\alpha_j \beta_j}} \mathbf{x(t)} \qquad \text{(Eq. 4.2)}$$

$\mathbf{\Gamma}$ is used as a concise notation of the complicated operator. Eq. 4.2 is illustrated by the block diagram shown in Figure 4.2. From the figure we see that the same input $\mathbf{x(t)}$ is applied to each relay operator. Their individual outputs are multiplied by $\boldsymbol{\mu(\alpha, \beta)}$,

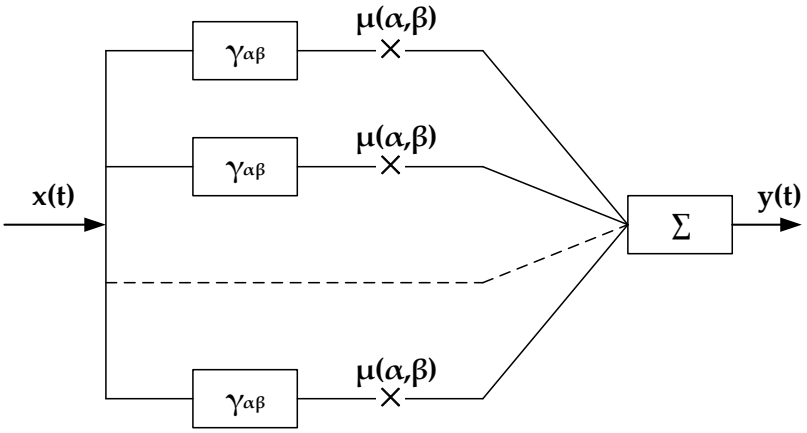


Figure 4.2: Block diagram of complicated model of hysteresis.

and then summed to obtain the output $\mathbf{y}(\mathbf{t})$. In the following sections we will use the name 'hysteresis model' to represent $\mathbf{\Gamma}$.

4.3 Hysteresis Application to Animation

The hysteresis model $\mathbf{\Gamma}$ has been widely adopted. The physical meaning of the input $\mathbf{x}(\mathbf{t})$ and the output $\mathbf{y}(\mathbf{t})$ of $\mathbf{\Gamma}$ is unspecified unless in particular applications. In this section, we will firstly overview the use of $\mathbf{\Gamma}$ in our particular application of generating line drawing animations, including specifying the physical meanings of $\mathbf{x}(\mathbf{t})$ and $\mathbf{y}(\mathbf{t})$. Later, we will explain how to compute each part of $\mathbf{\Gamma}$ (including the weight function $\boldsymbol{\mu}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ and the parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$).

4.3.1 Overview

In our line drawing generation system, the hysteresis model works as a generalized thresholding method. Its elementary relay operator $\boldsymbol{\gamma}_{\boldsymbol{\alpha\beta}}$ alone performs some kind of hysteresis thresholding, similar to the thresholding used in Canny edge detection [16] but in the temporal domain and on 3D models. $\boldsymbol{\gamma}_{\boldsymbol{\alpha\beta}}$ adopts a double-threshold strategy with $\boldsymbol{\alpha}$ and

Table 4.1: The input $\mathbf{x}(\mathbf{t})$ for thresholding used in existing line drawing algorithms.

Algorithm	Mesh Property
Suggestive contours	radial distance of viewpoint
Ridges/valleys	curvature
Apparent ridges	view-dependent curvature
PELs /XPLs	gradient of intensity of illumination

$\boldsymbol{\beta}$ working as lower and upper thresholds. The hysteresis model generalizes the hysteresis thresholding as a parallel connection of independent relay operators.

Definition 1 *The input* $\mathbf{x}(\mathbf{t})$ *is a value quantifying some kind of mesh property of a triangular mesh model. The output* $\mathbf{y}(\mathbf{t})$ *is a state value describing the drawing condition of a line.*

The I/O (Input/Output) of the hysteresis model $\boldsymbol{\Gamma}$ follows the same definition of the I/O of the usual single thresholding applied in existing line drawing generation systems. Both thresholding methods share the same $\mathbf{x}(\mathbf{t})$. The value of $\mathbf{x}(\mathbf{t})$ depends on the specific algorithms. For example, for ridges/valleys, $\mathbf{x}(\mathbf{t})$ is the curvature of the mesh on a per vertex/face basis [76]. Table 4.1 lists the mesh properties that some existing line drawing algorithms may use for $\mathbf{x}(\mathbf{t})$. However, the values of $\mathbf{y}(\mathbf{t})$ are different in the two thresholding methods. Single thresholding considers only the simplest condition, i.e., a line is either drawn or not. Thus, its $\mathbf{y}(\mathbf{t})$ is either 1 or 0. But, $\mathbf{y}(\mathbf{t})$ of $\boldsymbol{\Gamma}$ is an normalized value from the sum of the binary outputs of relay components. It is in a scalar value range of $[0, 1]$. Furthermore, the variation of $\mathbf{y}(\mathbf{t})$ in the temporal domain relies on the number of relay operators included in $\boldsymbol{\Gamma}$. If the number increases, the variation becomes smoother.

Definition 2 The $\mathbf{x}(\mathbf{t})$ *and* $\mathbf{y}(\mathbf{t})$ *of a hysteresis model as well as those of a relay operator are defined on a per triangle basis.*

We believe it is reasonable to define this in our system, due to the reason that the most of the existing line drawing algorithms extract their line drawings on a per triangle basis too.

Definition 3 *The hysteresis model is defined as a superposition of the relay operators corresponding to all triangles on the surface model.*

It is intuitive to imply Definition 3 based on Definition 2, because the components of the hysteresis model are required to be unchanged along the temporal domain. Definition 3 is always satisfied in our system, since the triangles remain the same as the mesh topology is unchanged during animation. However, practically it is unnecessary to consider all the triangles in the surface mesh for a triangle that is being processed. Usually a small neighborhood of triangles is enough. Therefore, we rely on the weight function to control the size of the neighborhood. Only those weights corresponding to the triangles within the neighborhood are larger than zero. The weights are zero corresponding to those triangles outside of the neighborhood.

Assertion 1 *The local information in each triangle is left unchanged. This information will be used with the thresholding results in the line rendering stage.*

Line drawing algorithms produce line drawings on a per triangle basis, which is a local information. The line drawings are represented as chains of feature points that make up the line segments on the triangles of a surface mesh to depict the appearance of the 3D objects. The precise positions of the feature points are located from the line drawing algorithms by analyzing the local geometry of the 3D objects and/or from the position of the view point. However, compared to the local information of line drawings, it is how the segments cluster into lines that matters more to the final look of the line drawings. By using the hysteresis model for thresholding, the geometric topology of triangles is taken into consideration, which produces more convincing thresholding result and leads to better clustering of the lines. Coherency of line drawings will improve from this.

Our line drawing generation system follows a similar workflow as the current line drawing generation systems to produce line drawing animations. It starts with the usual line extraction from 3D models in object space. For each frame, mesh data are imported

into hysteresis models for thresholding, which is an adaptive procedure with respect to the history of the previous as well as the current model geometry. State values are thus generated. The values work as a function of 'adaptation' to the extracted lines, from which new lines are produced. Sequencing the new lines frame-by-frame will form a new line drawing animation with improved temporal coherence.

4.3.2 Weight Function

A key problem in the use of a hysteresis model $\mathbf{\Gamma}$ for a triangle $\mathbf{T}$ is to build a weight function $\boldsymbol{\mu}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ for every relay operator $\boldsymbol{\gamma}_{\boldsymbol{\alpha\beta}}$. $\boldsymbol{\mu}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ has a finite support for $\mathbf{\Gamma}$ (i.e., $\boldsymbol{\mu}(\boldsymbol{\alpha}, \boldsymbol{\beta}) > 0$) within a region of interest (ROI), and outside of the ROI $\boldsymbol{\mu}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ is equal to zero. In the following, we define the ROI with respect to $\mathbf{T}$, and compute $\boldsymbol{\mu}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ based on the ROI.

A simplest and most straightforward method of creating ROI for each $\mathbf{T}$ is to use ROI= $\mathbf{T}$. It is equivalent to the use of the usual thresholding with two thresholds. But this is extremely limited to certain cases, where lines hardly move on the surface, such as ridges/valleys in rigid body undergoing a camera motion. It is much more common that lines are moving when the geometry of the model is deformed, and the triangles which contain the lines may be changed respectively. ROI is responsible for capturing the drawing information of the lines for $\mathbf{T}$, and thus it should cover a larger area around $\mathbf{T}$. We consider two types of triangles in the ROI: triangles that are topologically and temporally connected to $\mathbf{T}$.

Definition 4 *The ROI of a triangle* $\mathbf{T}$ *covers 1) triangles in which chains of line segments are contained that make up a line that draws on* $\mathbf{T}$*, and 2) triangles which may cover the potential movement of the line in successive frame.*

The following two-step approach is used for the automatic generation of ROI for each $\mathbf{T}$.

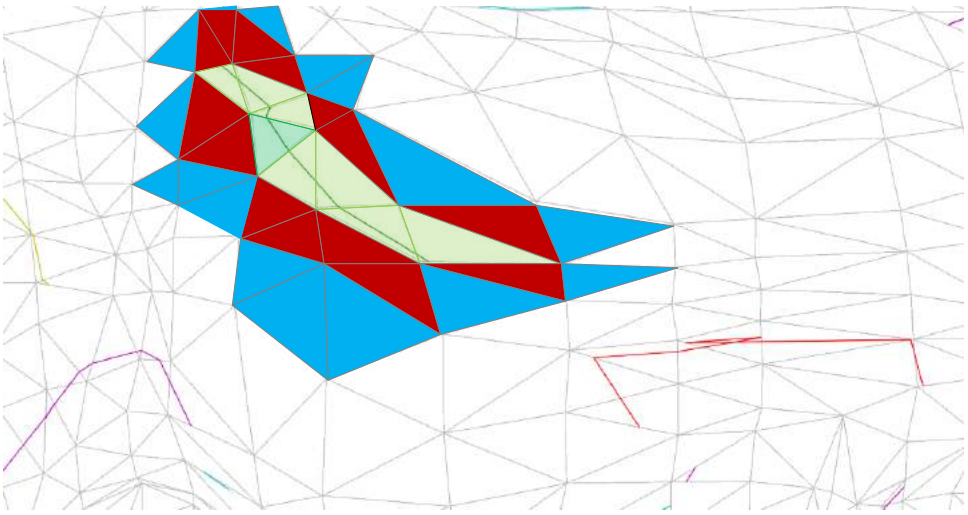


Figure 4.3: The illustration of ROI of triangle **T**. The semi-transparent region in green is **T**, and in light green is $\mathbf{R_0}$ with line drawn on **T**. 1-ring neighborhood is rendered in dark red and 2-ring neighborhood is rendered in blue.

Step 1. We apply a 3D tracing algorithm to trace the line that draws on **T**. Line tracing starts from **T**, and is achieved by traversing neighbor triangles using depth-first-search algorithm. If no line is drawn on **T**, the tracing stops and ROI= **T**. If a line is successfully traced, as a result, the triangles on which the line is drawn are linked into a triangle strip, and are added to the ROI.

Step 2. The triangles which may cover the potential movement of the line in a successive frame are obtained in the ring neighborhood of **T**'s ROI. Assuming 3D motions are coherent and each motion between two frames is very small, a line may move on the surface in the successive frame, but its movement would be within a neighboring area near its initial place in previous frame. Therefore, we can infer that, it is more accurate to obtain the drawing information of the line on **T** in the successive frame from the area rather than from the single correspondent triangle in the previous frame. However, we cannot predict the direction of the movement, since the line can either move toward an arbitrary direction on the surface, or just stay in the original place. Thus, such group of triangles would be added to current ROI to cover the possible line movements: ROI itself,

and all the neighborhood triangles that are adjacent to the ROI, i.e., 'ring neighborhood'. Figure 4.3 illustrates the extended ROI and the ring neighborhood that are being used. In this paper, for the experiments, we use a 2-ring neighborhood, which is reasonably robust with respect to mesh noises and tessellation.

The weight function $\boldsymbol{\mu}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ is used to weight the drawing probabilities of each triangle within the ROI. ROI can be divided by **i**-Ring neighborhood, and denoted by $\mathbf{R_i}$ respectively. Hence, the triangle strip found in step 1 is $\mathbf{R_0}$. The extended region found in step 2 is $\mathbf{R_i}$ ($\mathbf{i} \geq 1$). Weight is set to be to be the same for each triangle $\mathbf{T_i}$ in one $\mathbf{R_i}$, meaning that no matter how far $\mathbf{T_i}$ is from $\mathbf{T}$ on the surface, they are both of the same importance to the line which draws on them. The weights are different though, between each $\mathbf{R_i}$. The weight for $\mathbf{R_i}$, denoted by $\boldsymbol{\omega_i}$, is computed using the Gaussian function:

$$\boldsymbol{\omega_i} = \frac{\mathbf{1}}{\mathbf{n}} * \frac{\mathbf{1}}{\sqrt{\mathbf{2\pi\sigma^2}}} \mathbf{e}^{-\frac{(\mathbf{i}-\boldsymbol{\mu})^2}{\mathbf{2\sigma^2}}} \qquad \text{(Eq. 4.3)}$$

where $\mathbf{n}$ refers to the number of triangles in $\mathbf{R_0}$. We use $\boldsymbol{\mu} = 0, \boldsymbol{\sigma} = 1$ in Gaussian function for the experiments in this chapter. The value of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ in Eq. 4.3 are set due to the consideration of the 2-ring neighborhood used in ROI.

4.3.3 Parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$

Another key problem in the use of a hysteresis model $\boldsymbol{\Gamma}$ for triangle $\mathbf{T}$ is to compute each pair of $\boldsymbol{\alpha_i}$ and $\boldsymbol{\beta_i}$ of the relay operator components $\boldsymbol{\gamma}_{\boldsymbol{\alpha_i\beta_i}}$ corresponding to the **i**th triangle $\mathbf{T_i}$ in ROI. This has to be done because the input of each $\boldsymbol{\gamma}_{\boldsymbol{\alpha_i\beta_i}}$ is $\mathbf{x(t)}$, the same as that of $\boldsymbol{\Gamma}$, while each $\boldsymbol{\gamma}_{\boldsymbol{\alpha_i\beta_i}}$ is supposed to provide the thresholding result of $\mathbf{T_i}$, using the input $\mathbf{x_i(t)}$ from $\mathbf{T_i}$. Thus, we adjust $\boldsymbol{\alpha_i}$ and $\boldsymbol{\beta_i}$ instead of the input of $\boldsymbol{\gamma}_{\boldsymbol{\alpha_i\beta_i}}$ to have the equivalent result. Originally, for each $\boldsymbol{\gamma}_{\boldsymbol{\alpha_i\beta_i}}$, $\boldsymbol{\alpha_i} = \boldsymbol{\alpha}$, $\boldsymbol{\beta_i} = \boldsymbol{\beta}$. $\boldsymbol{\alpha_i}$ and $\boldsymbol{\beta_i}$ can be computed automatically through adding a displacement to $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. Assume $\boldsymbol{\Delta x} = \mathbf{x(t)} - \mathbf{x_i(t)}$, $\boldsymbol{\alpha_i}$ and $\boldsymbol{\beta_i}$ can be computed as:

$$\boldsymbol{\alpha_i} = \boldsymbol{\alpha} + \boldsymbol{\Delta x}, \boldsymbol{\beta_i} = \boldsymbol{\beta} + \boldsymbol{\Delta x} \qquad \text{(Eq. 4.4)}$$

This is equivalent to that input is $\mathbf{x_i(t)}$ for $\boldsymbol{\gamma}_{\boldsymbol{\alpha}_i\boldsymbol{\beta}_i}$.

4.3.4 Output

From the definition of ROI we can infer that all triangles that share the same 3D path will have the same ROI. Since the weight function $\boldsymbol{\mu}(\boldsymbol{\alpha}, \boldsymbol{\beta})$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are all computed based on ROI, triangles sharing the same ROI will have the same outputs. In other words, a consistent value is obtained per ROI. The value can be further used per line to modulate the final line rendering.

$\mathbf{y(t)}$ has a scalar value range in $[0, 1]$. The scalar value may be used as a *confidence value*, indicating how likely the line would be drawn. It could also be used to indicate stability of the line. To visualize the output, we may adopt a similar scheme as proposed by DeCarlo et al. [26], and use $\mathbf{y(t)}$ to control the luminance of the line, which works like 'fading' effect. However, different from the original scheme that renders such effect on a per line segment basis, we render on a per line basis. While binary black-and-white line drawings might produce visual flickering, gray-colored line drawings rendered under our scheme looks visually smoother between consecutive frames, because the color transition is less conspicuous.

4.4 Results and Discussions

Our line drawing generation system has been tested on a wide-range of animated 3D models, including real-world models and man-made models. The testing models contain both finely and coarsely tessellated models. Representative line drawing algorithms are applied in our system: second-ordered algorithms such as suggestive contours, third-ordered algorithms such as ridges/valleys (view independent), apparent ridge (view dependent), PELs (light dependent) and our proposed XPLs (light dependent). We simply require users to input two parameters as thresholds $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. Hysteresis models are built with the same

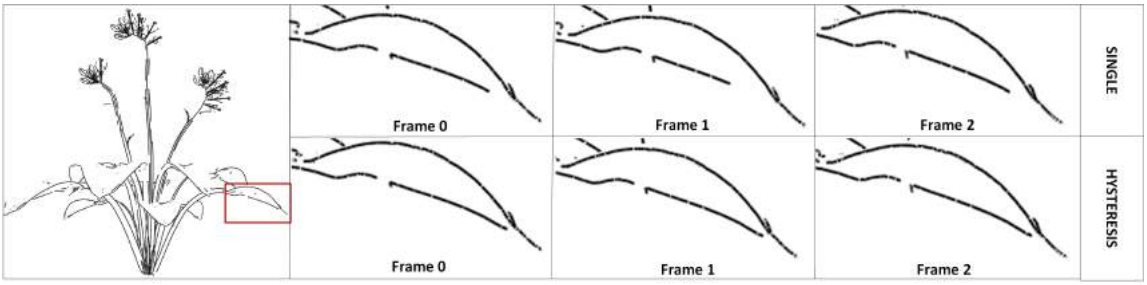


Figure 4.4: Model: Plant (fine-tessellated model, courtesy of Xfrog Inc. [45]). Line type: Ridges/valleys.

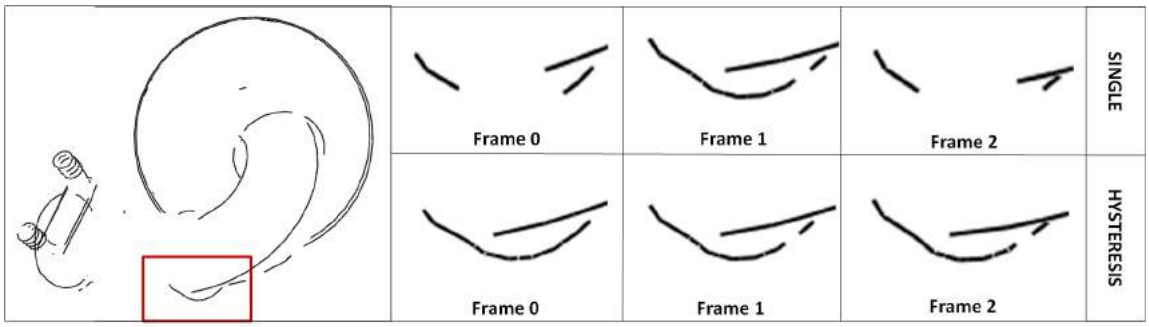


Figure 4.5: Model: Tube (man-made model, courtesy of TF3DM [2]). Line type: Ridges/valleys.

$\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ for each triangle on the surface. The models perform hysteresis thresholding on each-frame line drawings to obtain coherent result.

4.4.1 Results

We show some comparison experiments by using two types of thresholding strategies: single thresholding and hysteresis thresholding. The results are shown in Figure 4.4 to Figure 4.10. Each thresholding result is displayed in consecutive 3 frames. Enlarged figures show incoherent lines from single thresholding are improved by using the hysteresis thresholding, with both flickering and popping suppressed.

Selected sequential frames of line drawing animations generated by our system are shown in Figure 4.11 to Figure 4.16. Silhouettes may be illustrated in the figures as well but not applied with hysteresis thresholding. The results of line drawing animations

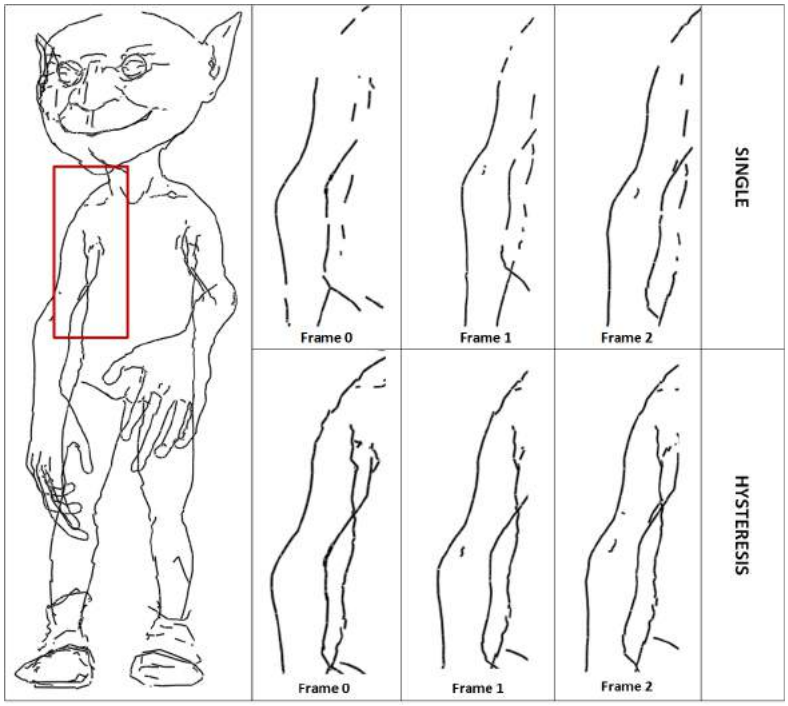


Figure 4.6: Model: Gremlin (animated FBX model, courtesy of Autodesk©software). Line type: Apparent Ridges.

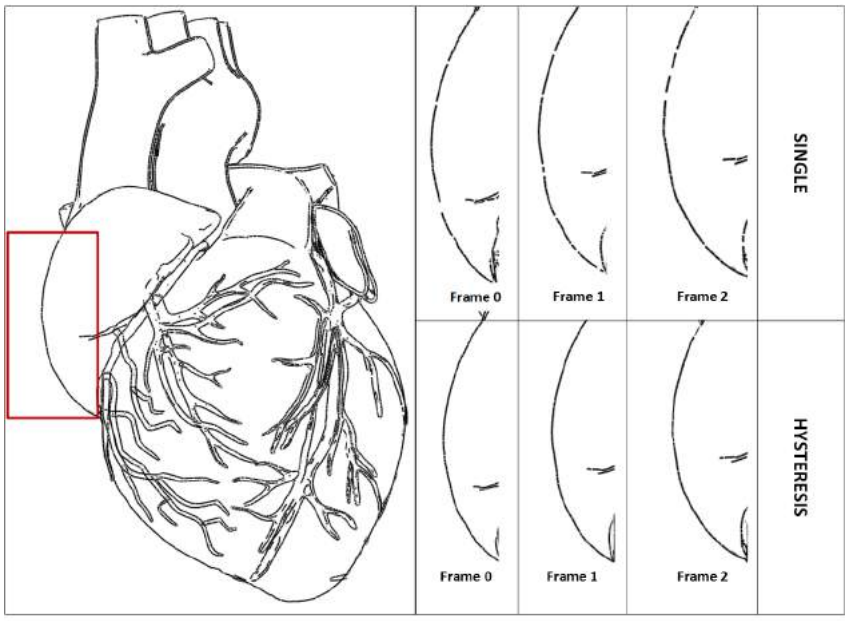


Figure 4.7: Model: Heart (fine-tessellated medical model, courtesy of Archive3D [103].) Line type: Apparent Ridges.

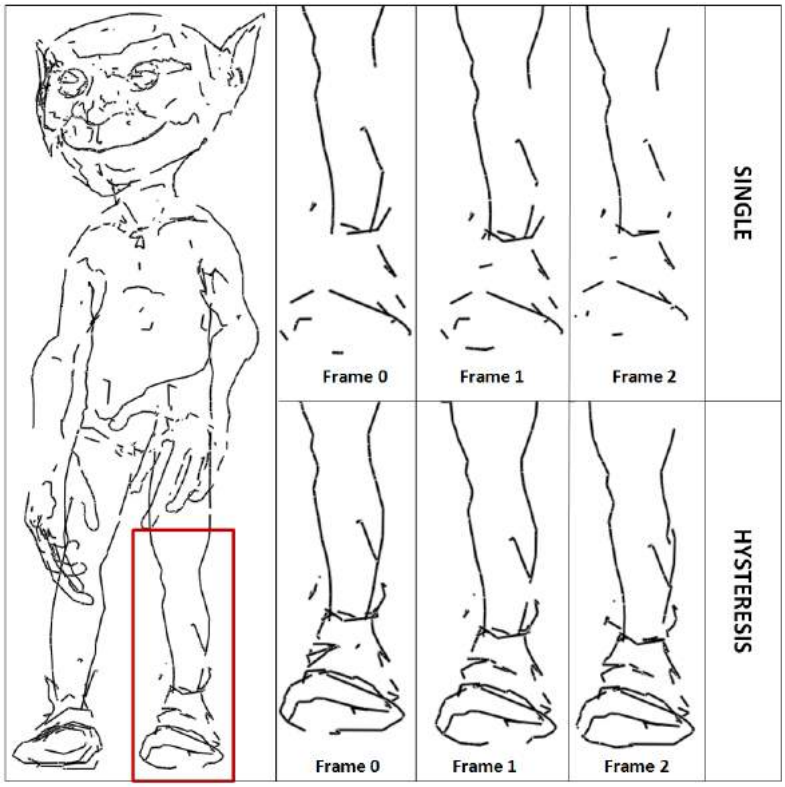


Figure 4.8: Model: Gremlin. Line type: PELs.

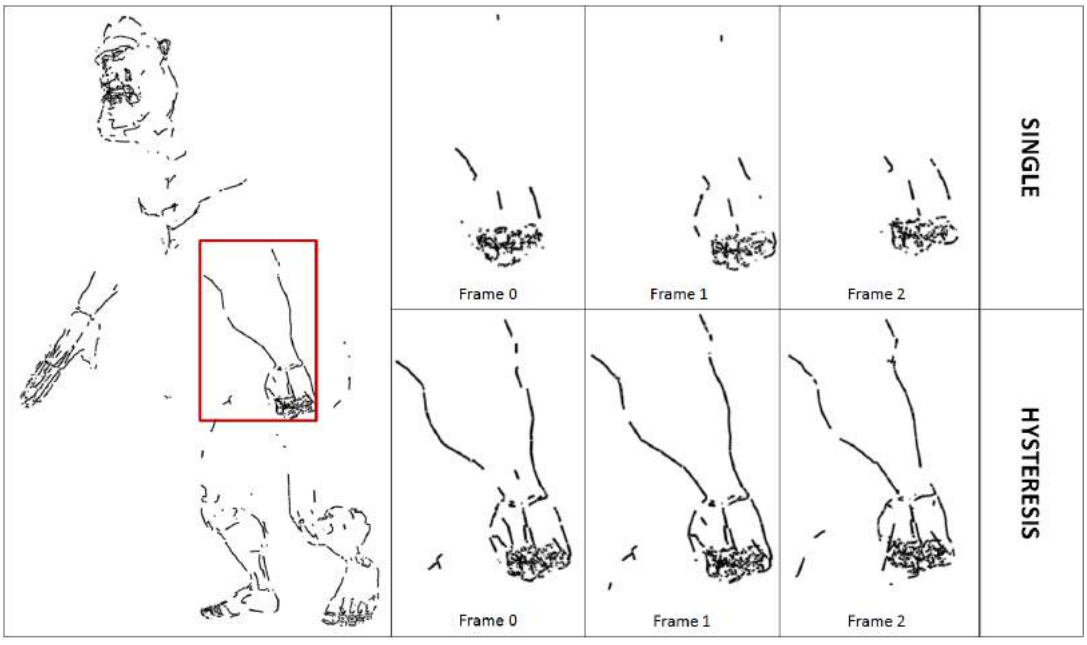


Figure 4.9: Model: Giant (animated FBX model, courtesy of Autodesk©software). Line type: XPLs.

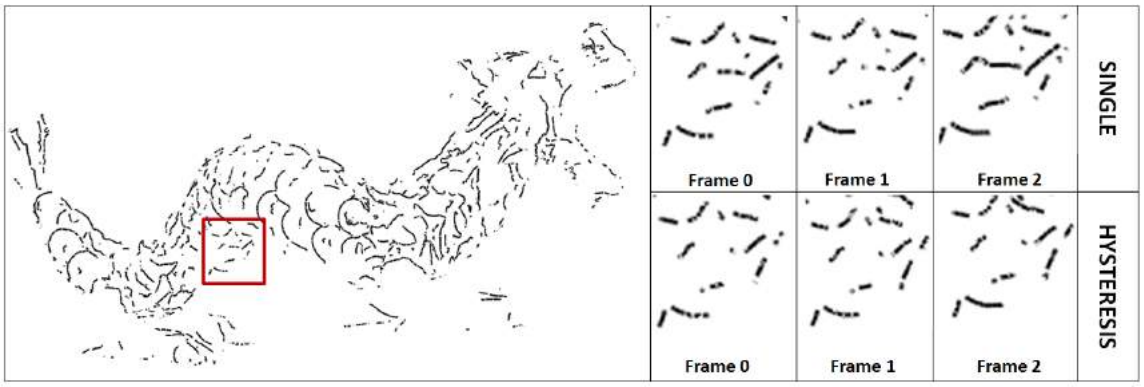


Figure 4.10: Model: Asian Dragon (3D scanning model, courtesy of Stanford Computer Graphics Laboratory [97]). Line type: Suggestive Contours.

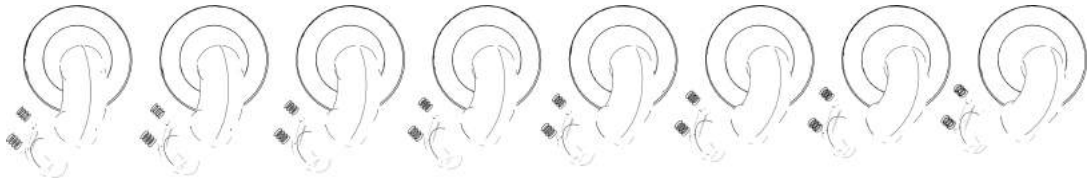


Figure 4.11: Ridges/valleys from hysteresis thresholding. Model: Tube.

demonstrate the effectiveness of our method in producing coherent high-order lines, on which a variety of styles can be applied for 2D stylized animation. More results are shown in supplementary video online [111].

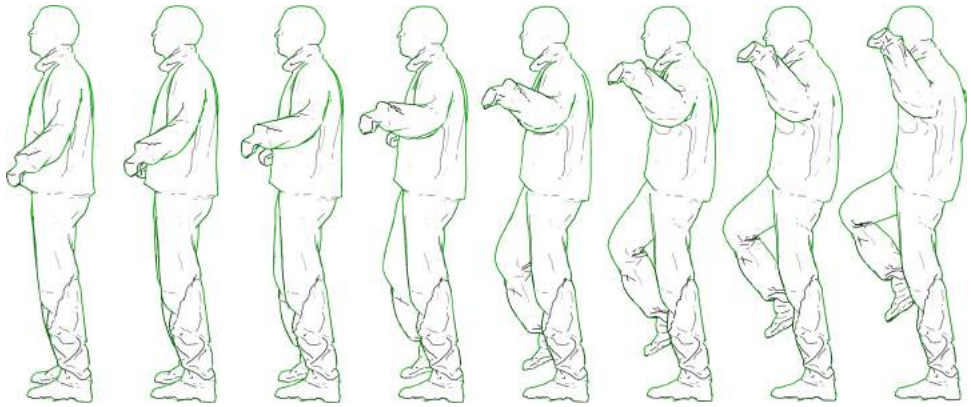


Figure 4.12: Ridges/valleys from hysteresis thresholding. Silhouettes are drawn in green lines for illustration purpose only. Model: Crane.

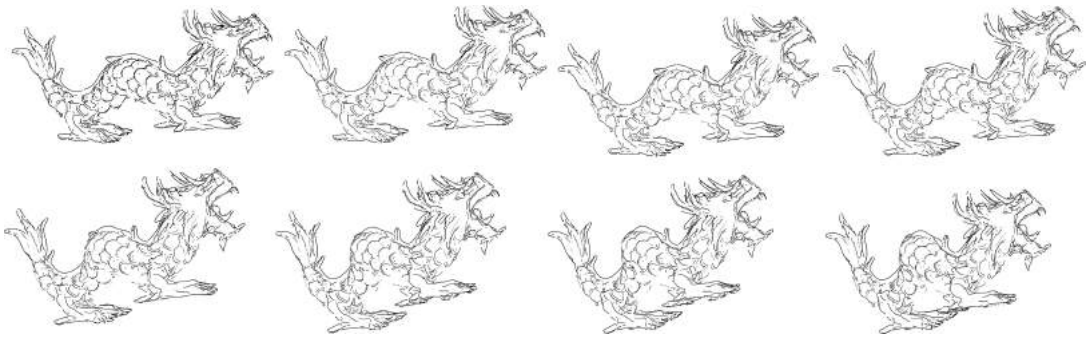


Figure 4.13: Suggestive contours from hysteresis thresholding. Model: Asian Dragon.

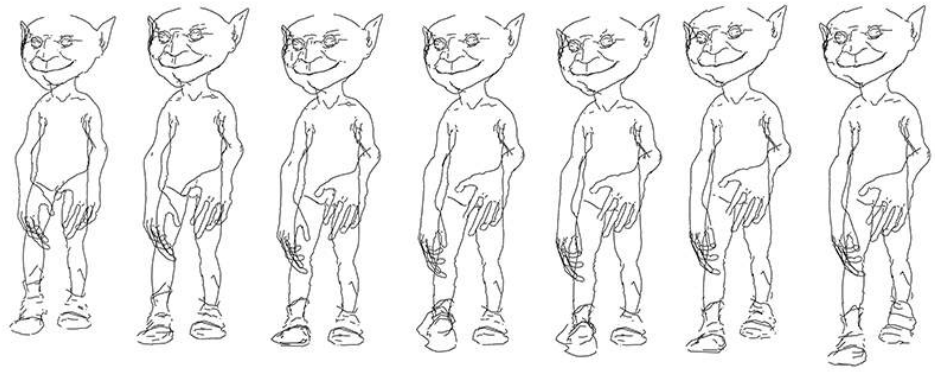


Figure 4.14: Apparent ridges from hysteresis thresholding. Model: Gremlin.

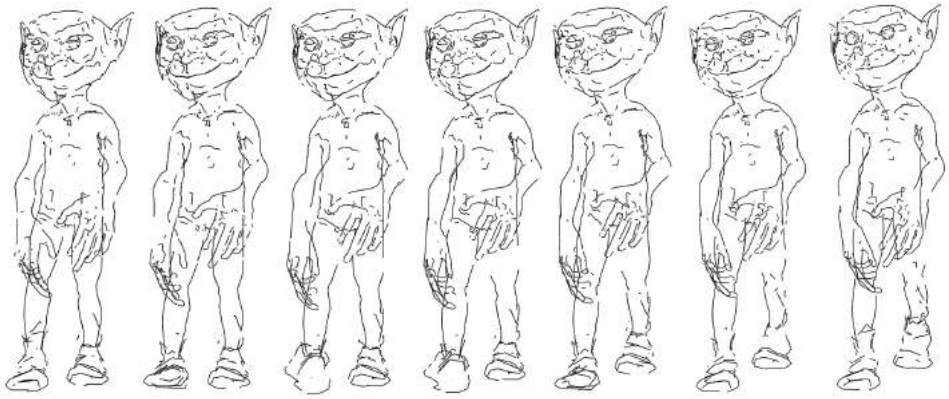


Figure 4.15: PELs from hysteresis thresholding. Model: Gremlin.

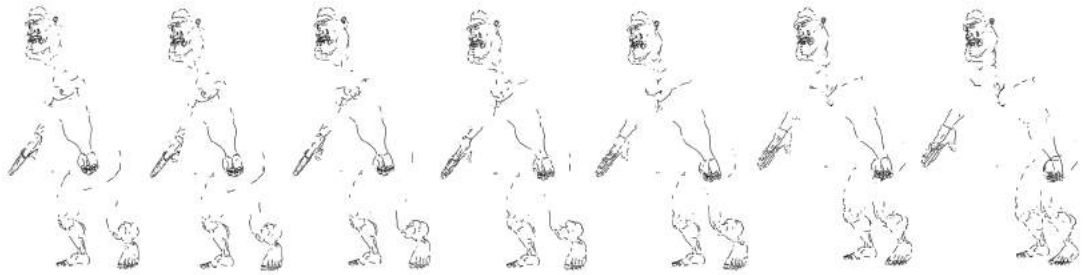


Figure 4.16: XPLs from hysteresis thresholding. Model: Giant.

4.4.2 Performance

The entire processing was done on a 3.3GHz Intel Xeon CPU with 8GB RAM. The system performs at interactive rates for generating each animation frame from 3D models with moderate and high geometric complexity (the number of faces are ranging from 8K to 180K for the models used in this paper, the number of lines are ranging from 200 to 2500).

4.4.3 Limitations

Our method works at a triangle level to adapt object-space lines. It is mesh resolution dependent, which leads to two limitations: firstly, the fineness of the input mesh is controlled by the users. If the mesh is not fine enough, the output lines may look unpleasant, for example, having zigzag patterns. Secondly, we have not considered the situation that a triangle is partially occluded, which causes the flickering artifacts on the *Tube* model.

4.5 Summary

We have proposed a novel method to solve temporal coherence problems of object-space line drawings using the classic hysteresis model. This method is especially suitable for

high-order lines. The resulting lines are vector lines and a variety of styles can be applied to produce 2D stylized animation.

Our proposed method is a triangle-basis approach, which does not deal with problems that happen on sub-triangle level. A more sophisticated occlusion test will be investigated to address this issue in the future.

Chapter 5

Coherent Path Evolution Using 3D Path Correspondence

This chapter presents an automatic approach to construct individual 3D paths from the coherent lines extracted in the previous chapter. 3D path correspondence is established and incorporated across frames to minimize visual artifacts of the lines. This method is especially suitable for high-order lines in object space. It consists of three core algorithms: (1) a robust tracing algorithm, to transfer collections of line segments extracted from 3D models into individual 3D paths; (2) a novel matching algorithm, to create correspondent trajectory of each individual path across frames, taking in account both the variations of numbers and lengths of the individual paths in each frame; and (3) an effective evolving algorithm, to apply global optimization on each trajectory. We evaluate our method on a wide range of animated 3D models to demonstrate its effectiveness in producing animations with temporally coherent 3D lines.

5.1 Overview

Line drawings animation can be produced by concatenating a sequence of images which contain line drawings. To generate such animation in a typical traditional pipeline, key frames are first drawn by artists and thereafter inbetween frames are created. Computers could be used for digital 2D animation production, assisting artists to draw key

frames [31] and automatically generating inbetween frames [104]. The creation of inbetween frames from each pair of key frames is one of the major tasks in producing an entire animation. However, without 3D information, particularly information of occluded part, the inbetween frames may be generated incorrectly for lines with extreme features such as a sharp bend. These lines are thus quite unsuitable for producing stylized animations as they require a lot of manual cleaning up. An alternative to the key-frame technique is to directly derive line drawings from animated 3D models, and create a set of individual 3D paths, by tracking unorganized line drawings for coherence in the consecutive frames.

However, it is a non-trivial problem to generate coherent 3D paths and produce nice and clean animated line drawings. A key to the problem is to build correspondences of the paths across the frames. This is especially challenging for high-order lines in object space. These lines are highly vulnerable to mesh tessellation and noises, which often results in large numbers of lines that viewers can observe (i.e., visible paths) in each frame, and varied numbers and lengths of paths in successive frames. Therefore, high-order lines are hard to be animated since their coherence in animation is subtle.

In this chapter, we present a fully automatic approach that construct paths from animated 3D triangle mesh models; builds temporal relationships of the paths; and creates animations with visual coherence. Figure 5.1 illustrates the workflow of our method, starting from inputting an animated 3D model for 3D line extraction and hysteresis thresholding, followed by our three-step strategy consisting of path tracing, matching and evolving to produce an image sequence of animated lines.

Input. Our input is an animated sequence of 3D triangle mesh models. We can use existing animated models generated from motion capture or manually create animations from tools like Maya, 3Ds Max, AutoCAD, SolidWorks, Motion Builder, etc. We support file formats such as FBX and OBJ. All animated meshes have vertex correspondence and share the same mesh topology as prior knowledge. The motions of the animation can be due to either camera motions or mesh deformations or both.

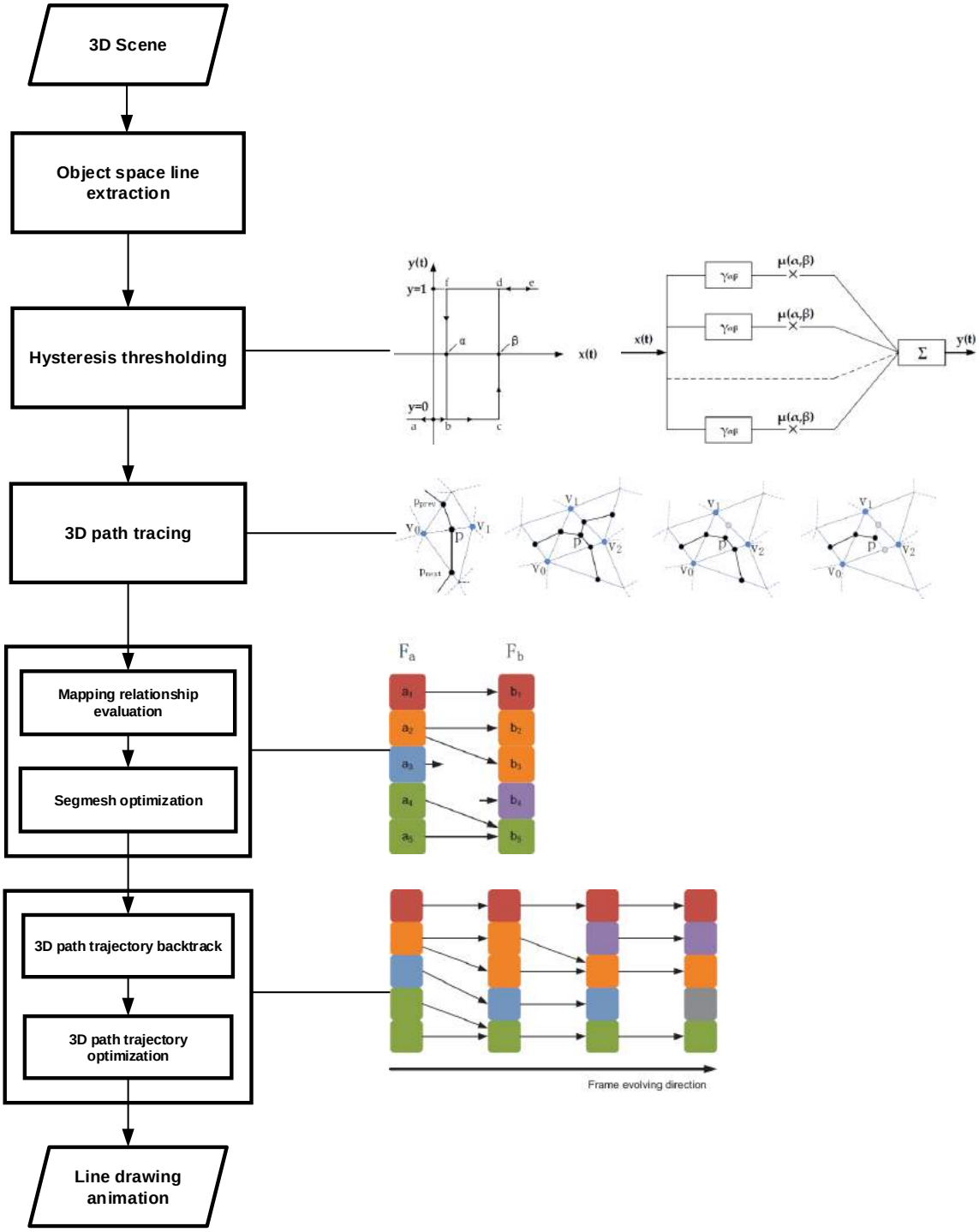


Figure 5.1: The workflow of our method.

Strategy. Object-space line drawing algorithms are applied automatically on the models of each frame to extract 3D line drawings, followed by:

1) Tracing 3D path: a collection of 3D paths is constructed by tracing the extracted 3D line drawings for each frame (Section 5.2).

2) Matching 3D path: two path collections among inter frames are matched to obtain their correspondence (Section 5.3). This step can be further developed into two sub-steps:

2.1) Inter-frame correspondence of path collections is described as a mapping relationship in Section 5.3.2, and the evaluation of the mapping relationships for matching is introduced in Section 5.3.3.

2.2) A regional segmentation method called 'Segmesh' is introduced in Section 5.3.4 for optimizing matching to reduce computation cost.

3) Evolving 3D path: global optimizations are applied to each path trajectory to establish coherent path evolution (Section 5.4). This step consists of two sub-steps:

3.1) Path trajectories are created across the animation sequence using back tracking in Section 5.4.1.

3.2) Each path trajectory is optimized according to the mapping information to generate coherent path evolution, which is described in Section 5.4.2.

A *path* denotes an individual 3D line on the surface. The line drawing algorithms tested in the workflow come from existing high-ordered algorithms, such as ridges and valleys [76], suggestive contours [27], apparent ridges [51], Photoic Extremum Lines (PELs) [109], as well as our proposed XPLs. Note that silhouettes [42] are used in our work for illustration only without any processing since their coherence is not an issue.

Output. Our output line drawing animation is a frame-by-frame animation rendered from the evolved 3D lines. Each line on the rendered frame can be re-projected to its 3D counterpart and can be tracked throughout the entire animation.

Figure 5.2: An example of Crane model rendered with 3D paths traced from ridges and valleys extracted from current view (left) and another view (middle). The overlapping 3D paths show their evolution during the animation (right).

5.2 Path Tracing

Given the extracted 3D line drawings from all the frames, we apply a tracing algorithm to construct their respective collections of 3D paths. The extracted line drawings are represented as chains of feature points that make up of line segments on the triangles of the surface mesh to depict the appearance of the 3D objects. The precise positions of the feature points are located from the line drawing algorithms by analyzing the local geometry of the 3D objects and/or the position of the view point. The collections of 3D paths are traced on the surface mesh from the collections of line segments and feature points before they are further processed.

Figure 5.2 illustrates an example of the Crane model rendered with 3D paths traced from ridges and valleys extracted from the current view (left) and another view (middle). The occluded 3D paths in the first view are also traced and visualized in the second view. We apply 3D path tracing instead of screen-space line tracing (actually it is a re-sampling procedure) because only in this way the entire 3D feature lines can be traced into paths

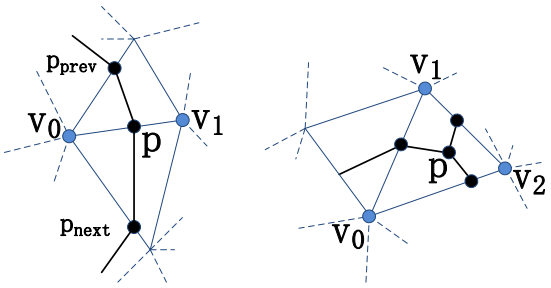


Figure 5.3: Triangles contain a non-umbilical point (left) and an umbilical point (right).

including both non-occluded and occluded parts. This occlusion information is needed for the subsequent matching and evolving processing.

5.2.1 3D Path Definition

A 3D *path* is defined as an ordered list containing *feature points* extracted from the line drawing algorithms. The feature points for ridges and valleys are classified into three categories: (1) a point that lies on a mesh edge; (2) a point that resides inside a mesh triangle; (3) a vertex of a triangle, which is considered as a special case of (1). Figure 5.3 illustrates the first two categories: (1) is named as *non-umbilical point*; (2) is related to local geometry features called *umbilical point* [6].

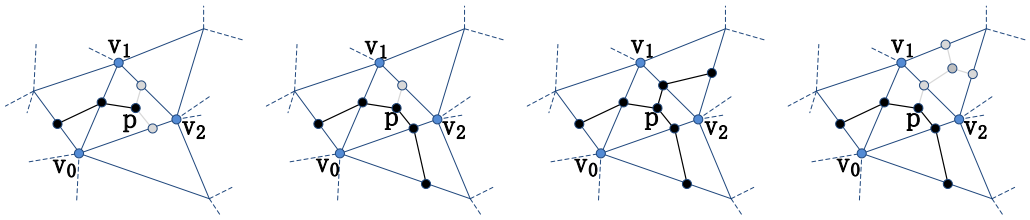
A non-umbilical point is defined as:

$$\mathbf{p_{nu}} = (\mathbf{1} - \mathbf{r})\mathbf{v_0} + \mathbf{r}\mathbf{v_1} \qquad \text{(Eq. 5.1)}$$

where $\mathbf{r}$ denotes an edge-normalized distance from a start vertex $\mathbf{v_0}$ to the end vertex $\mathbf{v_1}$. $\mathbf{p}_{prev}$ and $\mathbf{p}_{next}$ are two non-umbilical points on the surface mesh represented as triangle strips before and after $\mathbf{p_{nu}}$ respectively.

Umbilical point represents locally flat region on a surface. They are located at the triangle of the surface mesh where the principal curvatures of all the vertices are equal. Their precise locations are defined in a triangle as:

$$\mathbf{p_u} = (\mathbf{v_0} + \mathbf{v_1} + \mathbf{v_2})/3 \qquad \text{(Eq. 5.2)}$$



5.4(a) 5.4(b) 5.4(c) 5.4(d)

Figure 5.4: The current umbilical point is traced as (a) a single, (b) a wedge, or (c) a trisector. If two umbilical points meet, they will be collapsed into a single point (d).

where $\mathbf{v_i}, \mathbf{i} = 0, 1, 2$ denotes the three vertices of the triangle [76].

The shape of lines containing umbilical points often has more than one branch on the trunk, and possible shapes like V shapes, Y shapes and closed shapes may exist. In our work, we represent each branch of the line as a path and implement it as a polyline via a doubly-linked list, to efficiently support the many insertions and deletions of points as the path evolves over the surface mesh.

5.2.2 Tracing Algorithm

Tracing is to obtain collections of individual 3D paths from respective line drawings in each frame. Before tracing starts, all triangles on the surface mesh are traversed and those with drawn lines are flagged. We apply two tracing algorithms for non-umbilical and umbilical points on the lines respectively.

Tracing non-umbilical points. This starts with a randomly flagged triangle as a new path. A successfully processed triangle is marked. The current path stops when one of these cases happens: (1) the path ends without any adjacent triangle with drawn lines; (2) the path comes across a triangle without connected feature points; (3) the path comes back close to its starting point: in this case, a loop is created. Tracing ends when all triangles are traversed and processed.

Tracing umbilical points. This starts new paths from an initial triangle list, which covers all umbilical points on the surface mesh. Two paths get started if the umbilical

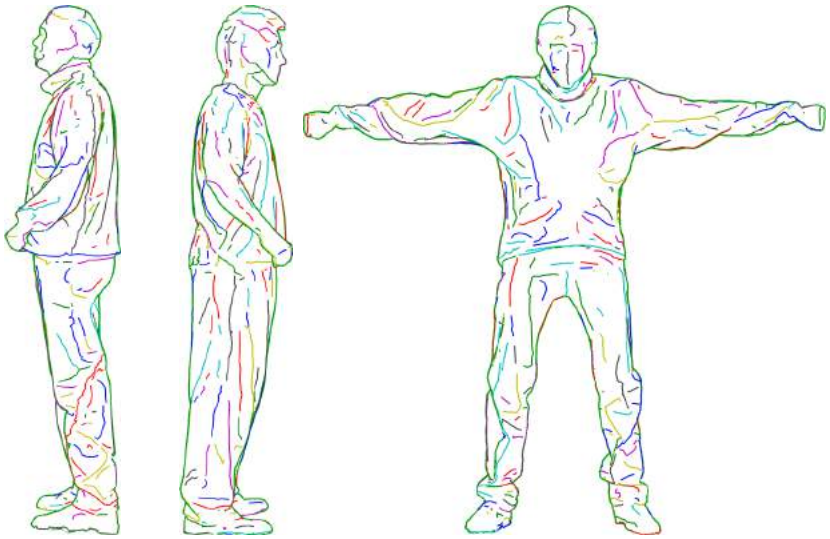


Figure 5.5: Tracing 3D paths of ridges and valleys for various models. From left to right, the models are Crane, Bouncing and Squat. Each path is given a particular color.

point is a wedge; while three paths get started if it is a trisector, to account for the local topology of the surface mesh (see Figs. 5.4(a) to 5.4(c)) [6]. Tracing stops when adjacent triangles do not have drawn lines any more or a triangle is without connected points, or a loop is detected. It also stops when another umbilical point is reached (see Figure 5.4(d)).

5.2.3 Tracing Result

The tracing algorithm has been tested through different line drawing algorithms on a wide range of real-world models. Some results are shown in Figure 5.5, where each individual path is rendered with a unique color. The number of paths traced from using different lines and models are shown in Table 5.1. We annotate each new path that is being traced by $\mathbf{p}$. Tracing for individual frames of line drawing animation (nth frame is marked as $\mathbf{F_n}$) results in collections of $\mathbf{p}$ (nth collection is marked as $\mathbf{P_n}$ respectively).

Table 5.1: Numbers of 3D paths that are being traced (with one model in a frame)

Model	Triangle	Vertex	Ridge	Valley	Suggestive Contour	Apparent Ridges	PELs
Bouncing	20000	10000	178	41	156	706	385
Tube	8492	4246	48	46	54	125	104
Asian Dragon	200000	100000	627	450	702	2015	1204

5.3 Path Matching

Given the collections of traced paths, we apply a matching algorithm to match the paths among the inter frames to obtain path correspondence information. Inspired by the work of Yu et al. [117] that analyzes stroke mapping to generate 2D stroke inbetweens, our algorithm establishes the correct mapping relationships (sets of mappings describing the respective path correspondences) for each pair of path collections between every two consecutive frames, by minimizing the energy functions.

The traced 3D paths in consecutive frames are not guaranteed to have the same numbers, lengths and positions. As a result, the paths in the two frames would not only have one-to-one mapping, but there are also many-to-many and part-to-whole mappings. In other words, each path can possibly correspond to a path, partially to a path, to a group of paths, or to nothing. Given a pair of frames containing $\mathbf{n}$ and $\mathbf{m}$ paths respectively to be corresponded, the search space of all mapping possibilities from $\mathbf{n}$ paths to $\mathbf{m}$ paths is much larger than $\mathbf{n^m}$. Therefore, the total computation cost for each evaluation function used for estimating the goodness of the mapping would be tremendously high. In this section, a regional segmentation method called 'Segmesh' is introduced for optimizing the matching to reduce computation cost.

Since the final output is a 2D animation, we also take line occlusion, which is caused by 2D projection from 3D models, into consideration in our matching algorithm. We separate all the paths in the path collection into two layers of visible and invisible paths respectively. The subsequent computations to evaluate the corresponding paths are restricted to one layer only, unless the paths cover both layers.

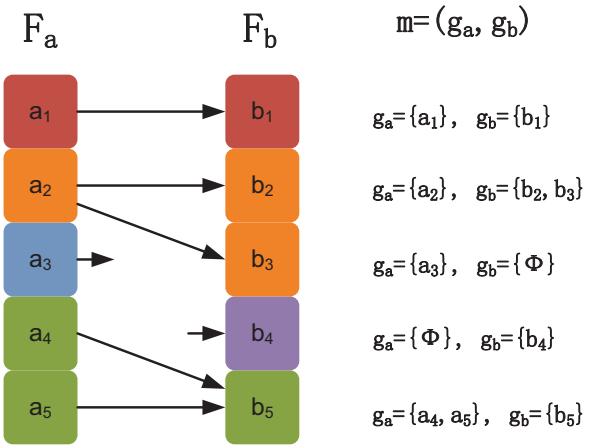


Figure 5.6: Mapping scenarios.

5.3.1 Representation of a Mapping

Given two frames $\mathbf{F_a}$ and $\mathbf{F_b}$, a *mapping* $\mathbf{m}$ is defined as a pair $(\mathbf{g_a}, \mathbf{g_b})$ where $\mathbf{g_a}$ and $\mathbf{g_b}$ represent two path *groups* in frames $\mathbf{F_a}$ and $\mathbf{F_b}$ respectively. The paths $\mathbf{p_a}$ and $\mathbf{p_b}$ inside the respective path groups $\mathbf{g_a}$ and $\mathbf{g_b}$ are correspondent.

In general, $\mathbf{m}$ defines inter frame path mapping and it is neither injective nor surjective. It describes one-to-one, many-to-many, zero-to-one and one-to-zero (the variable $\mathbf{m}$ for such scenarios is defined in the form of $(\emptyset, \mathbf{g})$ and $(\mathbf{g}, \emptyset)$ respectively) scenarios depending on the cardinalities of $\mathbf{g_a}$ and $\mathbf{g_b}$. This is illustrated in Figure 5.6.

5.3.2 Representation of a Mapping Relationship

A *mapping relationship* describes how two collections of paths $\mathbf{P_a}$ and $\mathbf{P_b}$ from frames $\mathbf{F_a}$ and $\mathbf{F_b}$ are matched to each other. It consists of a set of mappings $\mathbf{m_i} = (\mathbf{g_{a_i}}, \mathbf{g_{b_i}})$ and is denoted by $\mathbf{M}$.

We define a correspondence matrix $\mathbf{C}$ to explicitly denote a mapping relationship between one collection of paths and another [34]. The matrix is a 0-1 matrix, with 1 denoting that a path in one collection is corresponded to a path in the other collection.

The entry of the matrix is a set of correspondence variables $\mathbf{c_{ij}}$ that

$$\mathbf{c_{ij}} = \begin{cases} 1 & \text{if path } \mathbf{p_i} \text{ corresponds to path } \mathbf{p_j} \\ 0 & \text{otherwise.} \end{cases} \quad \text{(Eq. 5.3)}$$

The variables $\mathbf{c_{ij}}$ must satisfy assignment matrix constraints, i.e., either the rows or the columns must add up to 1, which allows many-to-one or one-to-many mappings but forbids many-to-many mappings. These constraints limit a mapping relationship to the following conditions:

• Every path in the frame must be included in a mapping, either to a corresponding path or to $\emptyset$.

• Any path in two different mappings are non-overlapped, i.e., many-to-many scenario strictly refers to one-to-many and many-to-one scenarios only.

• Each path group contains only one path for a mapping in the scenarios of zero-to-one and one-to-zero.

Note that the correspondence matrix of $\mathbf{P_a}$ to $\mathbf{P_b}$ in general is not equal to the matrix of $\mathbf{P_b}$ to $\mathbf{P_a}$.

Generally, Depth-First-Search (DFS) algorithm is applied to explore all possible mapping relationships from all mapping possibilities.

5.3.3 Evaluation of Mapping Relationships

As matching of two frames could result in a number of candidates of different mapping relationships, an evaluation is required to compare goodness among them to obtain the optimal one. The evaluation of a mapping relationship (denoted by $\mathbf{E(M)}$) is calculated by summing up the evaluation of each mapping $\mathbf{m_i}$ (denoted by $\mathbf{E(m_i)}$) in the relationship:

$$\mathbf{E(M)} = \sum_{\mathbf{m_i} \in \mathbf{M}} \mathbf{E(m_i)} \quad \text{(Eq. 5.4)}$$

In the matching algorithm, the evaluations of mapping relationships can be determined by different evaluation functions, such as taking into consideration of the cases of zero-to-one, one-to-zero or not considering them at all. The contribution of each mapping is equally the same for the goodness of a relationship. A lower value of the evaluation indicates a better result.

Evaluation of a Mapping Relationship Excluding $\emptyset$. First we consider the mapping relationship that contains all the mapping scenarios except the cases that either $\mathbf{g_a} \neq \emptyset$ or $\mathbf{g_b} \neq \emptyset$. According to the characteristics of the line drawing animation, since a mesh deforms continuously, a path could evolve to its neighboring vertices. Therefore, two paths in consecutive frames are likely to be matched (or partially matched) if they are very near to each other. The length variation of the two paths also indicates the possibility of matching. Consequently, we measure their distance and length variation by the functions $\mathbf{E_d}(\mathbf{m})$ and $\mathbf{E_l}(\mathbf{m})$ respectively, and use their product to assess the goodness of mapping (Eq. 5.5). Because all paths are traced on the surface mesh, all distances are computed in the form of geodesic distance [94].

$$\mathbf{E}(\mathbf{m}) = \mathbf{E_d}(\mathbf{m}) * \mathbf{E_l}(\mathbf{m}) \qquad \text{(Eq. 5.5)}$$

The distance of the mapping $\mathbf{m}$ (Eq. 5.6) is calculated as an average distance of two path groups $\mathbf{g_a}$ and $\mathbf{g_b}$ normalized by the sum of lengths of the paths in the two groups.

$$\mathbf{E_d}(\mathbf{m}) = \frac{\sum\limits_{\mathbf{t_i} \in \mathbf{T_{g_b}}} \mathbf{E_{d_{gt}}}(\mathbf{g_a}, \mathbf{t_i}) + \sum\limits_{\mathbf{t_j} \in \mathbf{T_{g_a}}} \mathbf{E_{d_{gt}}}(\mathbf{g_b}, \mathbf{t_j})}{\sum_{\mathbf{p_\alpha} \in \mathbf{g_a}} |\mathbf{p_\alpha}| + \sum_{\mathbf{p_\beta} \in \mathbf{g_b}} |\mathbf{p_\beta}|} \qquad \text{(Eq. 5.6)}$$

The distance from $\mathbf{g_a}$ to $\mathbf{g_b}$ is computed from the distances between $\mathbf{g_a}$ and all triangles $\mathbf{t_i}$ in $\mathbf{g_b}$, and vice versa. The distance from $\mathbf{g}$ to $\mathbf{t}$, denoted by $\mathbf{E_{d_{gt}}}(\mathbf{g}, \mathbf{t})$, is then calculated as the minimum of the distances between the triangle $\mathbf{t_i}$ from $\mathbf{g}$ and $\mathbf{t}$ on the surface by $\mathbf{E_{d_{tt}}}(\mathbf{t_i}, \mathbf{t})$ (Eq. 5.7).

$$\mathbf{E_{d_{gt}}}(\mathbf{g}, \mathbf{t}) = \min_{\mathbf{t_i} \in \mathbf{T_g}} \mathbf{E_{d_{tt}}}(\mathbf{t_i}, \mathbf{t}) \qquad \text{(Eq. 5.7)}$$

We define $\mathbf{T_g}$ for the triangle set that consists of all triangles $\mathbf{t}$ of paths $\mathbf{p}$ in $\mathbf{g}$ (Eq. 5.8).

$$\mathbf{T_g} = \bigcup_{\mathbf{p_i} \in \mathbf{g}, \mathbf{t_j} \in \mathbf{p_i}} \mathbf{t_j} \tag{Eq. 5.8}$$

The length of triangle set $\mathbf{T_g}$ is taken as the total number of triangles contained in the path group $\mathbf{g}$. The length variation of the triangle sets of two groups $\mathbf{T_{g_a}}$ and $\mathbf{T_{g_b}}$ is measured by $\mathbf{E_l(m)}$ (Eq. 5.9). The length variation is defined as the ratio of the larger length to the smaller length between the two groups.

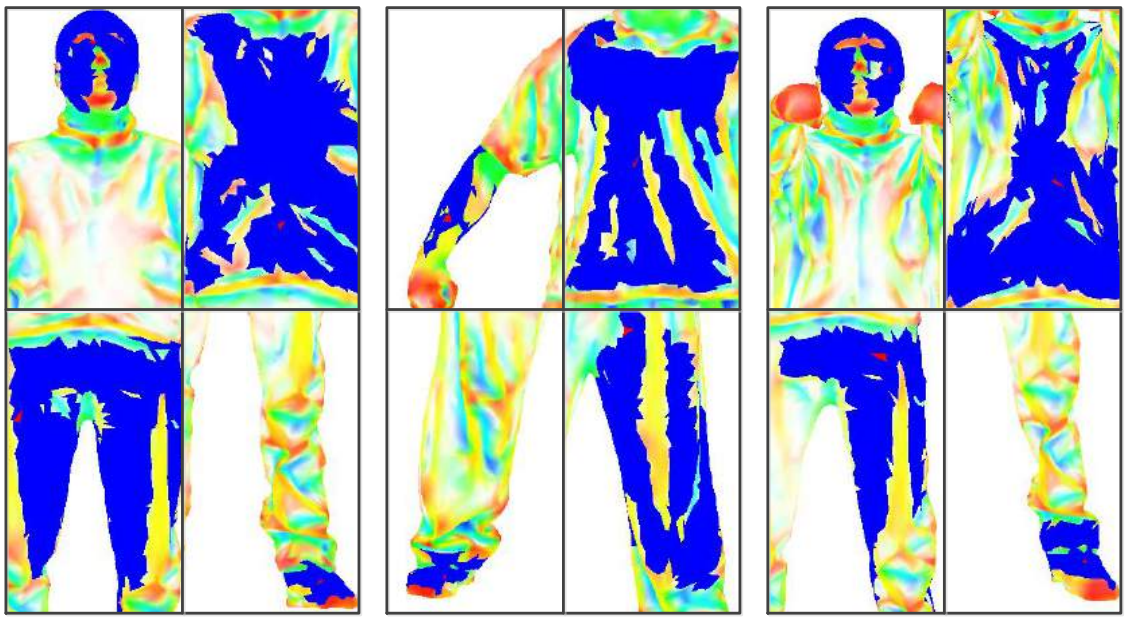
$$\mathbf{E_l(m)} = \mathbf{max}(\frac{|\mathbf{T_{g_a}}|}{|\mathbf{T_{g_b}}|}, \frac{|\mathbf{T_{g_b}}|}{|\mathbf{T_{g_a}}|}) \tag{Eq. 5.9}$$

Evaluation of a Mapping Relationship Including $\emptyset$. We consider the mapping relationship that contains all the mapping scenarios, including the cases of either $\mathbf{g_a} \neq \emptyset$ or $\mathbf{g_b} \neq \emptyset$. For mapping $\mathbf{m}$ that either $\mathbf{g_a} \neq \emptyset$ or $\mathbf{g_b} \neq \emptyset$, its goodness is defined as a weighted length of $\mathbf{T_g}$ (Eq. 5.10). Since $\mathbf{g}$ contains only one path for single $\mathbf{m}$ in the matching relationship $\mathbf{M}$, i.e., $\mathbf{g} = \{\mathbf{p}\}$, $|\mathbf{T_g}|$ is equal to the length of path $\mathbf{p}$. The weight $\boldsymbol{\omega}$ represents the tradeoff between general cases and special cases, where a larger $\boldsymbol{\omega}$ implies more tolerance to the special cases. In our work the goodness of the evaluation of the mapping relationship that excludes $\emptyset$ is the same as that includes $\emptyset$.

$$\mathbf{E(m)} = \boldsymbol{\omega} * |\mathbf{T_g}| = \boldsymbol{\omega} * |p| \tag{Eq. 5.10}$$

5.3.4 Optimization of Evaluating Mapping Relationships

Since the paths in two regions with different semantic meanings (e.g., the leg and the arm of a human body) should not be matched together, we can optimize the time cost of the evaluation of the mapping relationships. By replacing the entire mesh with a particular region of the mesh where the path lies on, the search space of possible mappings for each candidate of mapping relationship is decreased and thus the time cost is reduced.



5.7(a) 5.7(b) 5.7(c)

Figure 5.7: Examples of Segmeshes generated for models (a) Crane, (b) Bouncing, and (c) Squat. The initial triangle $\mathbf{t}$ is rendered in red on the mesh. Courtesy of Rusinkiewicz et al. [86] for their color-coded visualizations of computed curvature.

Segmesh Definition. We use principal curvature variations of triangle vertices of the surface mesh to indicate region boundaries. Thus, the surface is segmented into a number of regions. *Segmesh* is defined as such a region which covers relatively similar curvatures.

For each triangle $\mathbf{t}$, Segmesh is defined as a region of interest (ROI), i.e., a collection of triangles, where the curvatures of triangles are similar to the curvature of $\mathbf{t}$. Segmesh 'floods' those mesh areas with similar curvatures from $\mathbf{t}$, and adds all the qualified triangles into the ROI, until the border triangles reach a curvature variation larger than $\mathbf{c_{max}} - \mathbf{c_{min}}$, which denote the maximum and minimum curvatures of current ROI, or reach a triangle with curvature larger than the threshold $\boldsymbol{\delta}$ used in [76]. Figure 5.7 illustrates the result of creating Segmesh from different triangles for the models that are used in this section. The triangles within the generated Segmesh are shown in blue. The visualization of the principal curvatures of the mesh triangles is also given.

The flooding algorithm is easily trapped in local maximum and minimum. The watershed segmentation [69] solves this problem by applying a post-process merging to the segmented region, but in Segmesh, there is no way to do the merging due to overlapping regions. Instead, we apply a 'look ahead' strategy incorporated with the segmentation. Before each flooding, we check whether the principal curvatures of vertices of the current triangle are local extremum. The look ahead parameter $\boldsymbol{\xi}$ is defined to adjust the range of look ahead that will be checked for extremum. By default $\boldsymbol{\xi}$ is defined to 3, which means 3 rings of neighbor triangles directly outside the current triangle. Larger values of $\boldsymbol{\xi}$ help to eliminate more tiny local fluctuations for smoother result. Therefore, we would not be restricted to the local extremum and our ROI result would be a global extremum.

The Segmesh of a path $\mathbf{p}$, denoted by $\mathbf{S}$, is defined as a union of Segmeshes that start from each triangle $\mathbf{t_i}$ of $\mathbf{T_g}$ of the path (c.f. Eq. 5.8).

Optimizing with Segmesh. Given two path collections $\mathbf{P_a}$ and $\mathbf{P_b}$ from frames $\mathbf{F_a}$ and $\mathbf{F_b}$, to find all the possible mappings of each path in $\mathbf{P_a}$ (denoted by $\mathbf{p_{a_i}}$), the Segmesh of $\mathbf{p_{a_i}}$ is first created (denoted by $\mathbf{S_{a_i}}$). According to the mesh and vertex correspondence, $\mathbf{S_{a_i}}$ from the mesh that corresponds to $\mathbf{F_a}$ can be easily mapped to the mesh that corresponds to $\mathbf{F_b}$ to form a correspondent Segmesh (denoted by $\mathbf{S_{b_i}}$). Therefore, among all the paths in $\mathbf{P_b}$, only those paths found to be within or partially within the Segmesh $\mathbf{S_{b_i}}$ can be mapped to $\mathbf{p_{a_i}}$ (these mapped paths are denoted as $\mathbf{p_{b_j}}$). The optimization thus reduces the search space of $\mathbf{p_{a_i}}$ from the entire mesh to $\mathbf{S_{a_i}}$.

To obtain the goodness of a mapping with Segmesh optimization, the evaluation of $\mathbf{E}(\mathbf{m})$ (i.e., Eq. 5.5) is modified to be weighted by the proportion of triangle numbers of mapped paths $\mathbf{p_{b_j}}$ located inside $\mathbf{S_{b_i}}$.

5.3.5 Matching Result

Our matching algorithm is tested on different types of lines from a wide range of 3D models including real-world models and man-made models. Figure 5.8 displays the matching

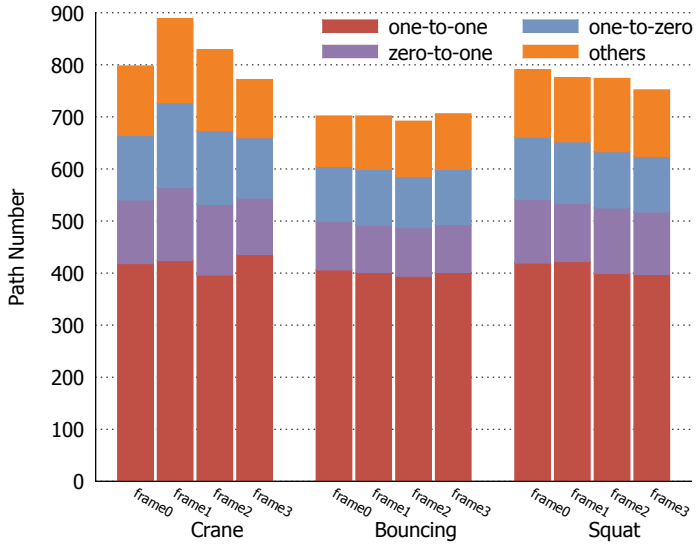


Figure 5.8: Mapping information of ridges and valleys for Crane, Bouncing and Squat models in consecutive frames.

information of the selected consecutive four frames of the Crane, Bouncing and Squat models, by counting the total numbers of mapping scenarios and showing them as colored blocks with various sizes. The size of the block illustrates the number of the mapping scenario and its percentage to the number of total mappings.

Selected matching results are displayed in Figure 5.9 to Figure 5.16. Correspondent paths are indicated by the same colors. The evaluation of matching quantitative lines is quite subjective since there is no benchmark. From the perceptional view, the matching result is promising.

5.4 Path Evolution

Given the matching information between each pair of consecutive frames, we can obtain the knowledge of path trajectory (i.e., path's life cycle) in the animation. With this knowledge global optimization can be applied to each path trajectory to generate coherent path evolution.

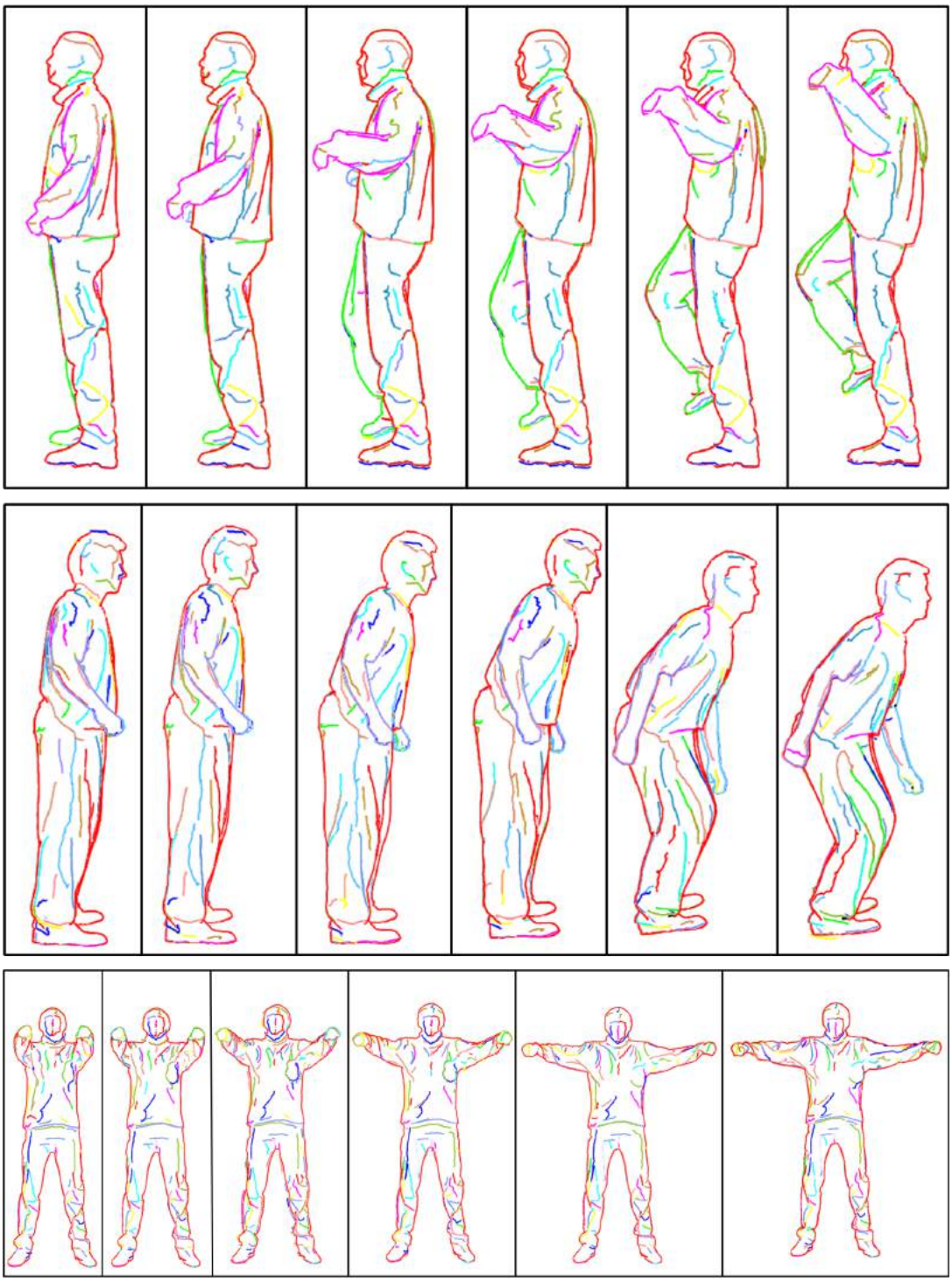


Figure 5.9: Matching results of the inter-frame sequences of models from the first to the third rows: Crane, Bouncing and Squat models. Line type: Ridges/Valleys, Contours.

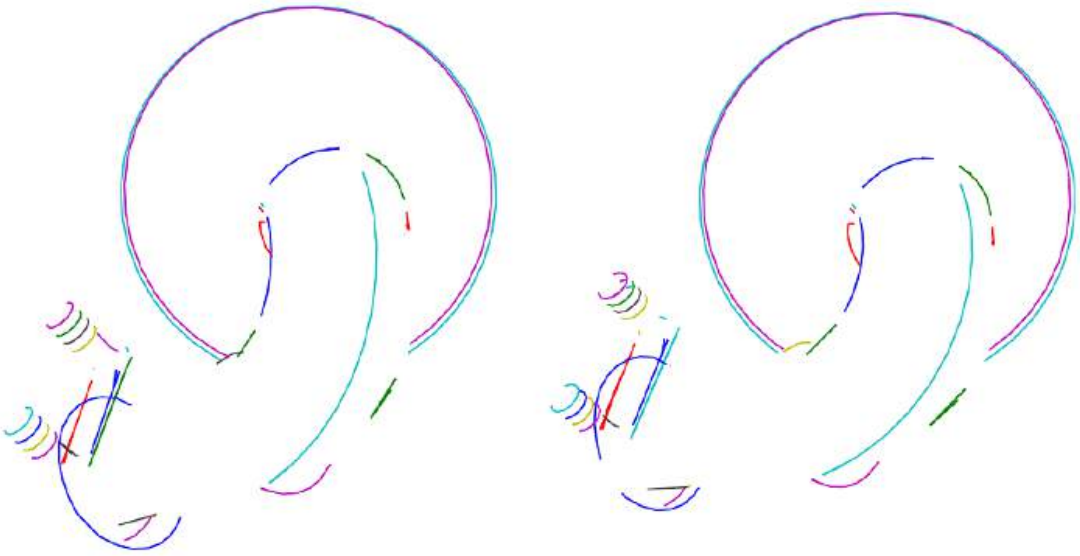


Figure 5.10: Matching results of two successive frames of Tube models. Line type: Ridges.

Figure 5.11: Matching results of two successive frames of Plant models. Line type: Ridges.

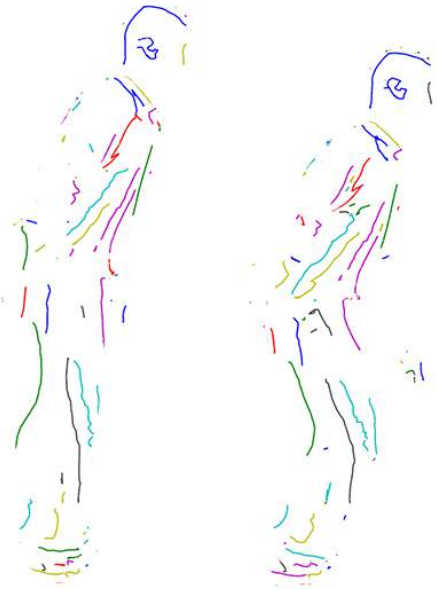


Figure 5.12: Matching results of two successive frames of Bouncing models. Line type: Suggestive Contours.

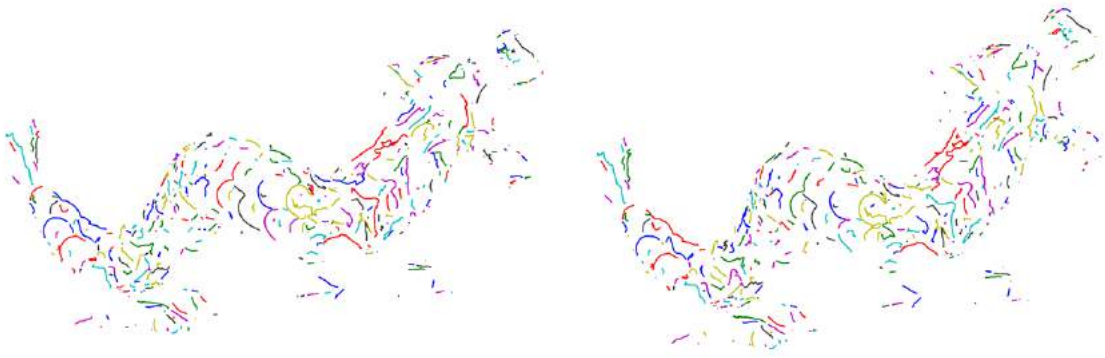


Figure 5.13: Matching results of two successive frames of Asian Dragon models. Line type: Suggestive Contours.

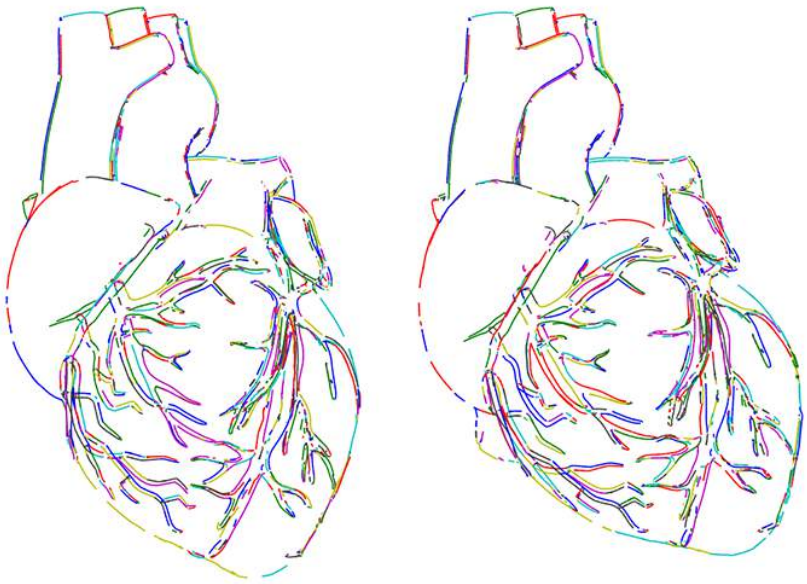


Figure 5.14: Matching results of two successive frames of Heart models. Line type: Apparent Ridges.

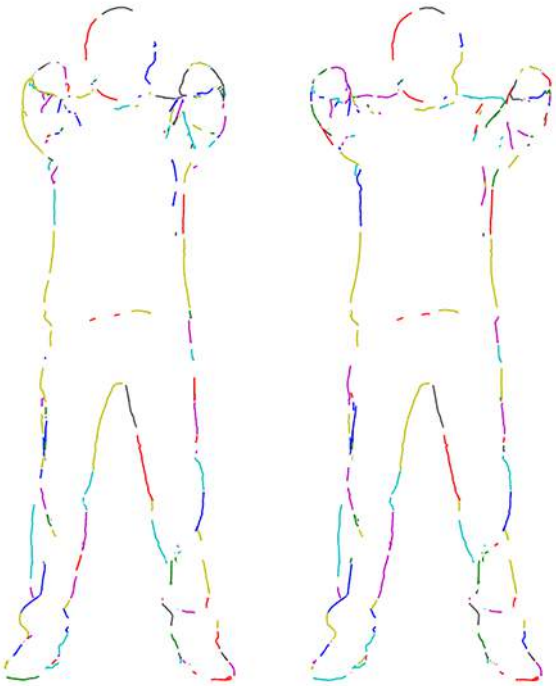


Figure 5.15: Matching results of two successive frames of Squat models. Line type: PELs.

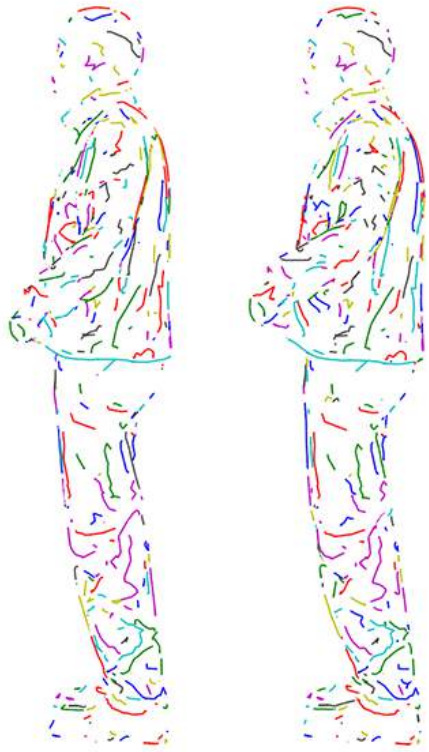


Figure 5.16: Matching results of two successive frames of Crane models. Line type: XPLs.

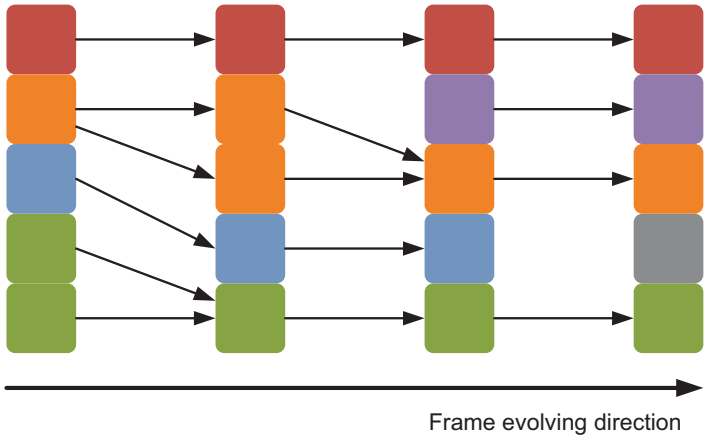
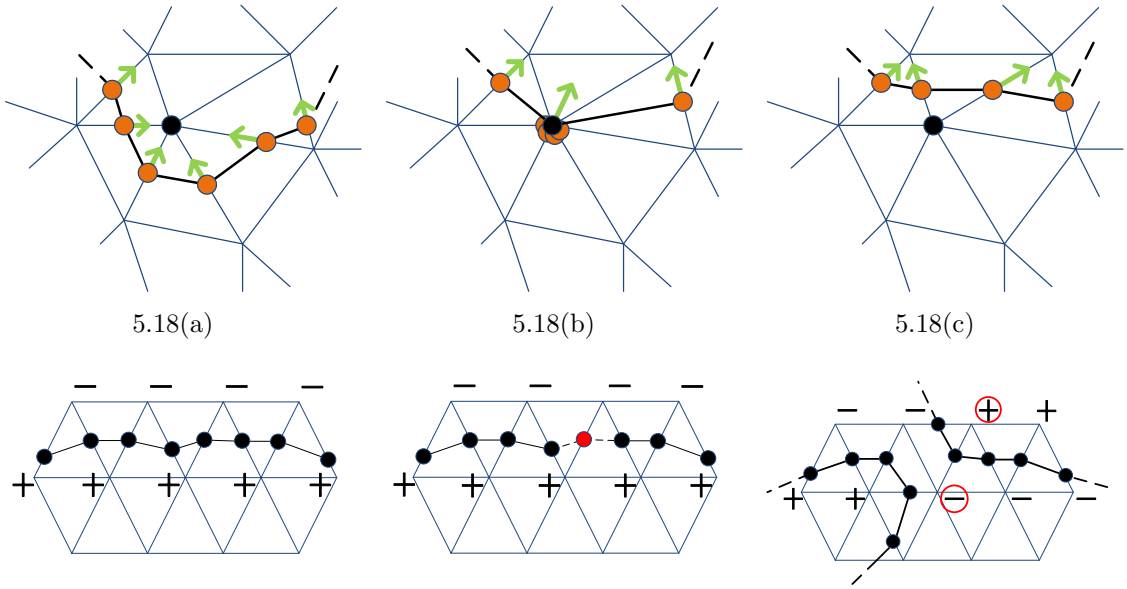


Figure 5.17: Path trajectories evolving through time.



5.18(a) 5.18(b) 5.18(c)

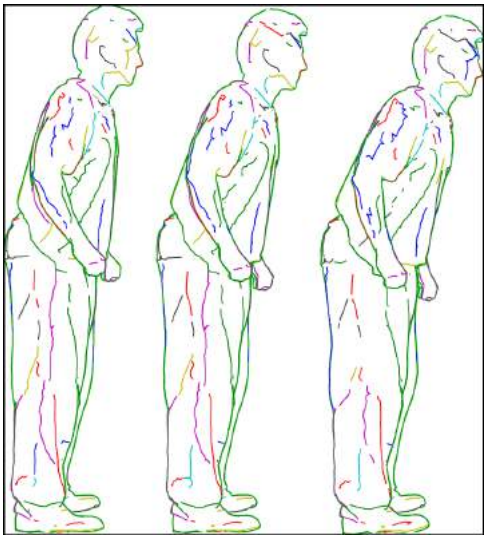
5.18(d) 5.18(e) 5.18(f)

Figure 5.18: Path moving on a surface mesh (a) before coming across a vertex; (b) on the vertex; (c) after coming across the vertex. An example of path splitting is shown in (d) to (f).

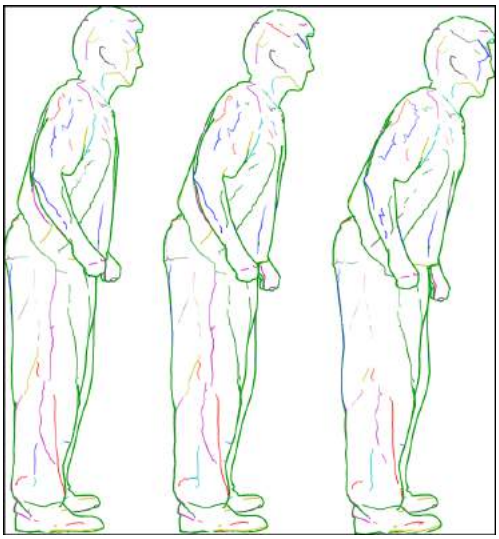
5.4.1 Path Trajectory

A path trajectory is a set of correspondent paths evolving through time (see Figure 5.17). Path topology is the relationship among all the paths in a path collection, which is similar to the definition in [55]. The topology remains unchanged unless the paths collide on the surface mesh and their number is changed. Figs. 5.18(a) to 5.18(c) illustrate an example of a path evolving through triangles and vertices on a surface mesh with its topology unchanged.

It can be seen when a path becomes shorter, the points on the path shrink to the vertices. On the other hand, when it gets extended, the points are appended after crossing over the vertex. The example of changing topology is shown in Figs. 5.18(d) to 5.18(f).



5.19(a) Constant-weight paths.



5.19(b) Faded paths.

Figure 5.19: Different drawing styles produced by our system.

5.4.2 Path Trajectory Optimization

Global optimizations can be applied to each path trajectory to produce coherent path evolution. We apply three optimization strategies for the one-to-one mappings, many/one mappings (including many-to-one and one-to-many mappings) and zero/one mappings (including one-to-zero and zero-to-one mappings) respectively.

One/One Mappings. When a path moves on the surface mesh without meeting any other path (which means that there is no path collision), it evolves in a one-to-one mapping for consecutive frames and its topology remains unchanged. This progressive evolution is considered as topologically coherent. However, the use of a single threshold might cause a large variation of lengths or even shapes between one correspondent path and another, leading to flickering.

Given the paths that have flickering, the most straightforward method to reduce the flickering is to apply fitting between the correspondent paths. However, it is quite challenging to fit 3D lines and keep the fitted lines on the surface. Considering the difficulty to adjust the locations of 3D paths, we focus on improving their appearance in

a coherent way in the image space. Our improved process derives geometric properties of the 3D paths from the surface mesh of the model and proceeds to obtain coherent rendering result.

We adopt a path fading scheme similar to the work of DeCarlo et al. [26] to improve coherent rendering of paths. It computes a positive weight $\boldsymbol{\omega}$ for every point on the path, which controls the transparency of a point. The weight $\boldsymbol{\omega}$ is zero when the point should be fully transparent (i.e., invisible), otherwise it is partially transparent. For each point of a path, partial transparency is simulated by blending point colors. The weight of the point is initially computed to correspond to the local curvatures of the point where it lies on the surface mesh of the model. Then an extra cut-off is applied and those weights below the threshold are set to zero. We compute the adjusted point color as a blending between the background color (white) [1,1,1] and the path color [r,g,b] where the point belongs to using the weight ω:

$$\frac{[1,1,1] + \boldsymbol{\omega} \cdot [r,g,b]}{1 + \boldsymbol{\omega}} \qquad \text{(Eq. 5.11)}$$

Figure 5.19 illustrates unweighted/weighted rendering paths.

Many/One Mappings. If a path collides with some other path(s) and all the paths join together to become one path in the subsequent frames, it evolves in a many-to-one mapping which means a path *merging* happens. The colliding paths in the previous frame are called *component paths*, while the merged path is called *compound path*. A compound path is formed by two or more component paths. Accordingly, one compound path splitting into two or more component paths forms a one-to-many mapping and is called path *splitting*. Path merging or splitting would change the path topology, causing locally abrupt change of paths.

In our work, the optimization on merging or splitting is to simplify the path topology. The lengths of compound path and component paths play a vital role in the optimization.

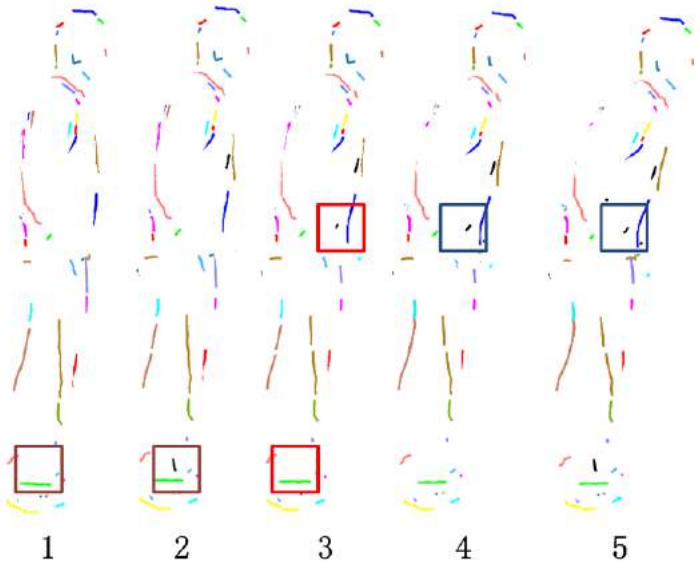


Figure 5.20: An example of processing popping and disappearing paths with valley lines.

In the consecutive frames, if the length of one of component paths covers a small ratio of whole length of the compound path, this component path could be ignored. The correction of paths is done by replacing all component paths by the compound path.

Zero/One Mappings. A path stopping from evolving and disappearing from the surface mesh is referred as *disappearing*, whereas a new path appearing and starts evolving is referred as *popping*. The key point of path optimization for disappearing or popping is to bring about stable path evolutions and maintain their life cycles as long as possible on the surface mesh, while discarding paths that have short lives.

Reference frames are used to decide whether a popping or disappearing path should be discarded or kept. For a popping path, the reference frames are defined to be those subsequent frames next to the frame containing the popping path. For a disappearing path, the reference frames are those frames preceding the frame containing the disappearing path. The path would be kept if their correspondent paths exist in reference frames and their mapping is one-to-one, indicating that the path evolves stably. Otherwise, the path would be discarded accordingly. By default the number of reference frames is set

Table 5.2: Average processing time (in seconds) for 3 models of the 3 steps.

Model	Trace	Match	Evolve
Crane	0.047	6.351	0.011
Bouncing	0.043	4.902	0.015
Squat	0.042	6.643	0.017
asian Dragon	0.284	143.9	0.047
tube	0.049	7.751	0.021

to 2.

Figure 5.20 shows an example of optimizing two path trajectories. Frame 3 in the figure contains popping and disappearing paths highlighted in the red rectangles respectively. Our optimization chooses two sets of reference frames for the popping path (Frame 4 and 5) and the disappearing path (Frame 1 and 2). The result would keep the path in the blue rectangles and discard the path in the brown rectangles.

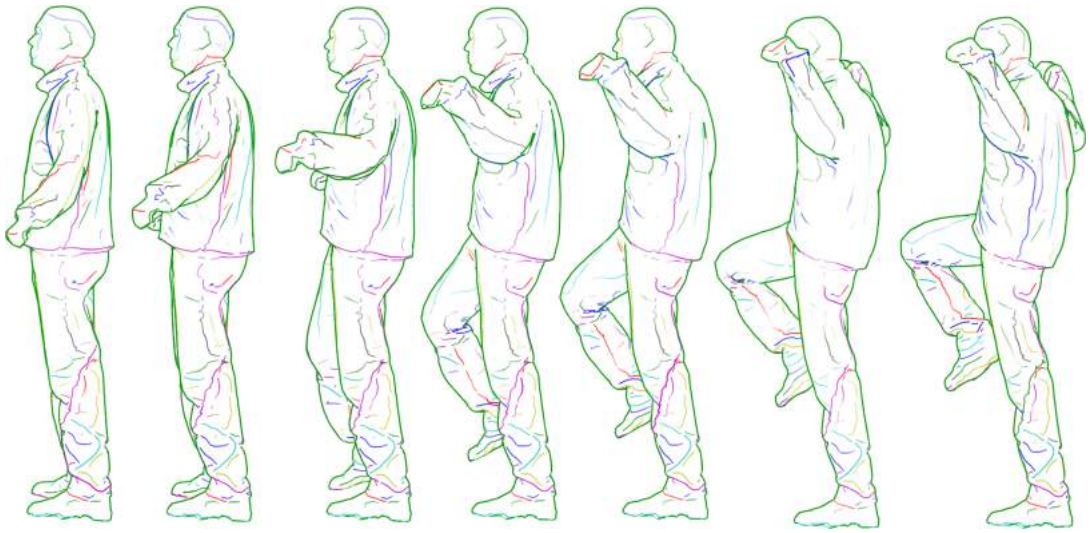
5.5 Results and Discussions

5.5.1 Results

Selected frame sequences of a line drawing animation generated from our method are shown in Figure 5.21 to Figure 5.29. These results of frame sequences demonstrate the paths evolve with temporal coherence. More results are shown in supplementary videos online [112, 113].

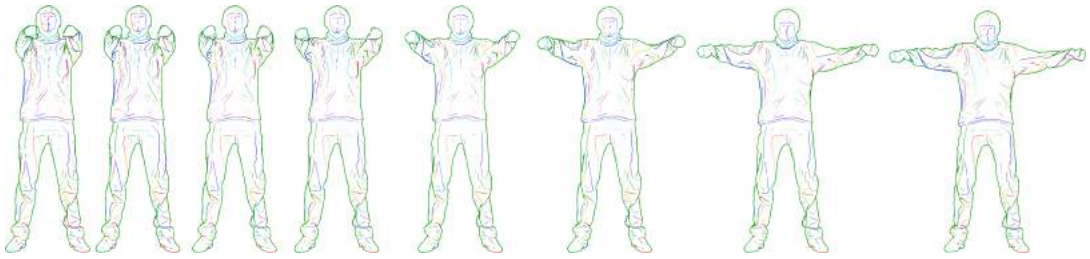
5.5.2 Performance

The entire processing was done on a 3.3GHz Intel Xeon CPU with 8GB RAM. The system performs at interactive rates for models with moderate complexity (up to 20k faces used in this work) excluding matching step (c.f. Table 5.2 for detail). Our implementation is primarily CPU-based but most part of the processing can be easily transferred to GPU, i.e., tracing, matching (including the creation of Segmesh) and path trajectory optimization can be directly parallelized.



5.21(a) Crane

5.21(b) Bouncing



5.21(c) Squat

Figure 5.21: Partial frame sequence of line drawing animation generated from our method. Line type: Ridges/Valleys. Contours are included in green for illustration only.

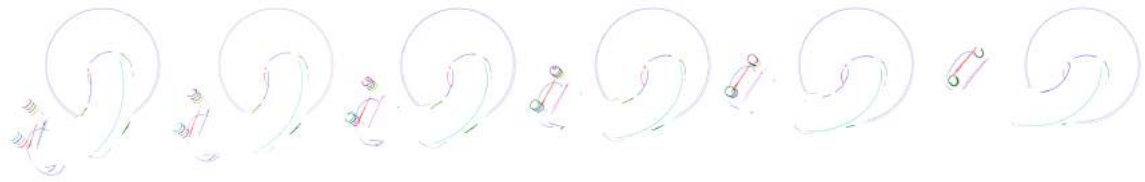


Figure 5.22: Partial frame sequence of line drawing animation from Tube Model. Line type: Ridges.

Figure 5.23: Partial frame sequence of line drawing animation from Plant Model. Line type: Ridges.

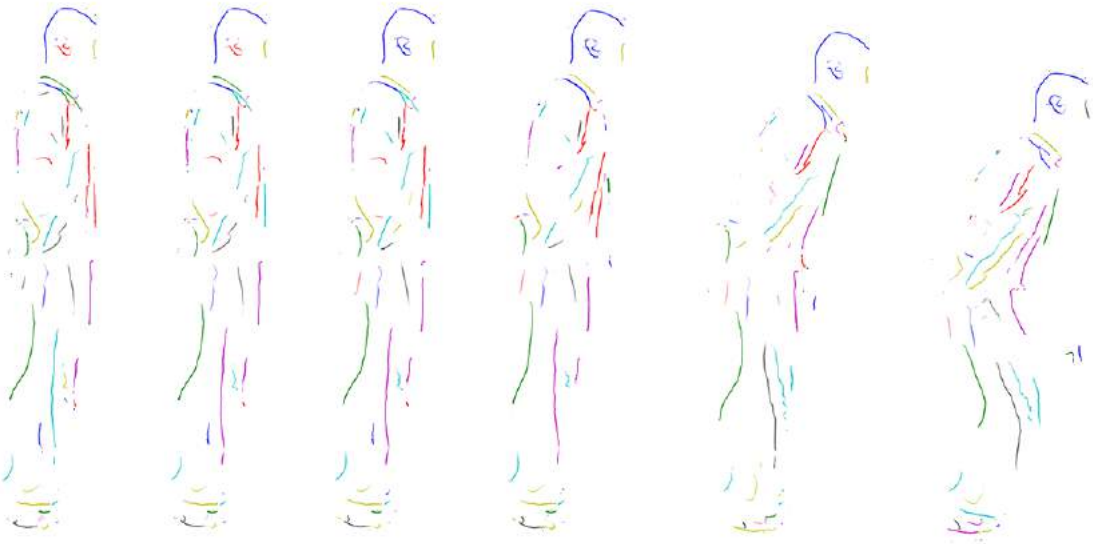


Figure 5.24: Partial frame sequence of line drawing animation from Bouncing Model. Line type: Suggestive Contours.

Figure 5.25: Partial frame sequence of line drawing animation from Asian Dragon Model. Line type: Suggestive Contours.

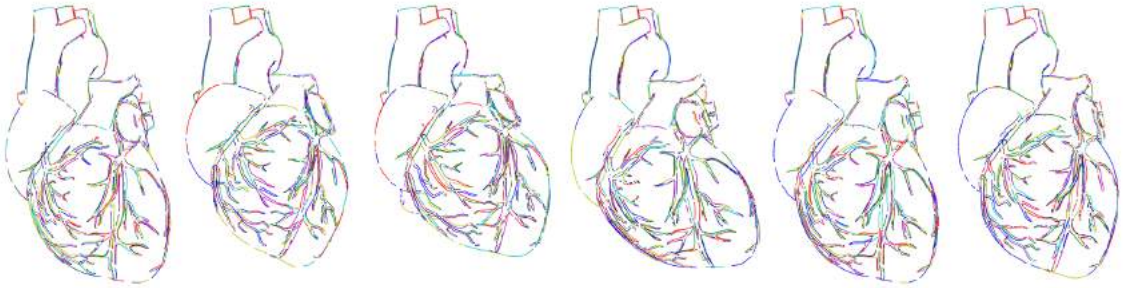


Figure 5.26: Partial frame sequence of line drawing animation from Heart Model. Line type: Apparent Ridges.

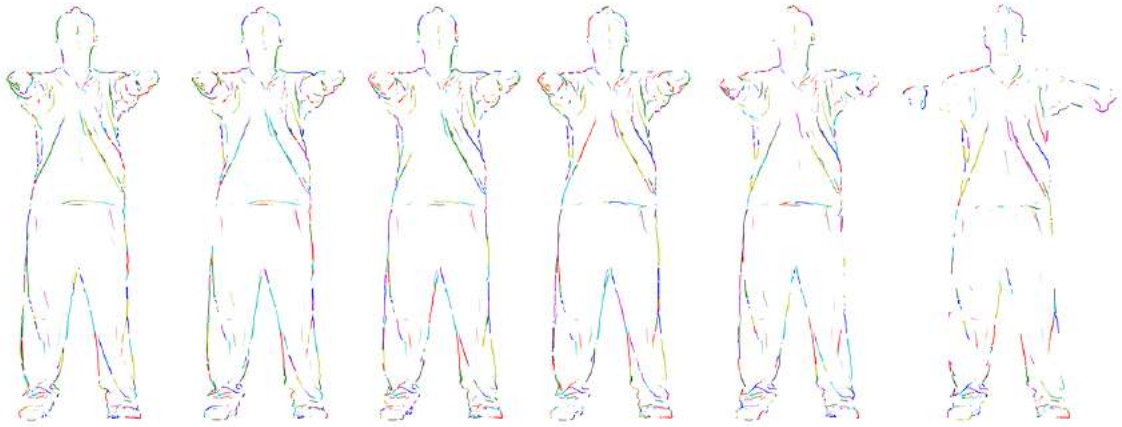


Figure 5.27: Partial frame sequence of line drawing animation from Squat Model. Line type: Apparent Ridges.

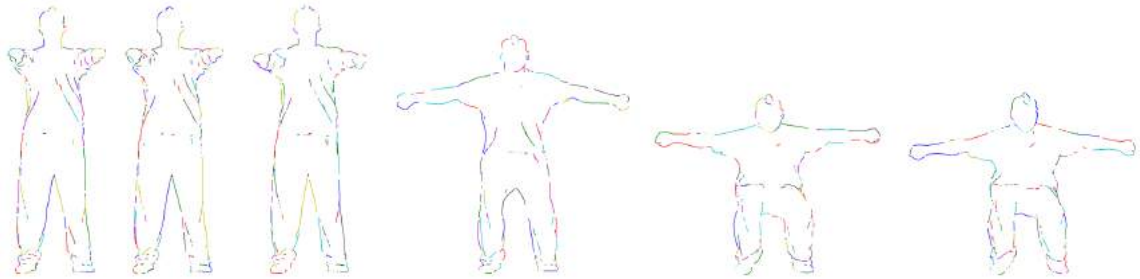


Figure 5.28: Partial frame sequence of line drawing animation from Squat Model. Line type: PELs.

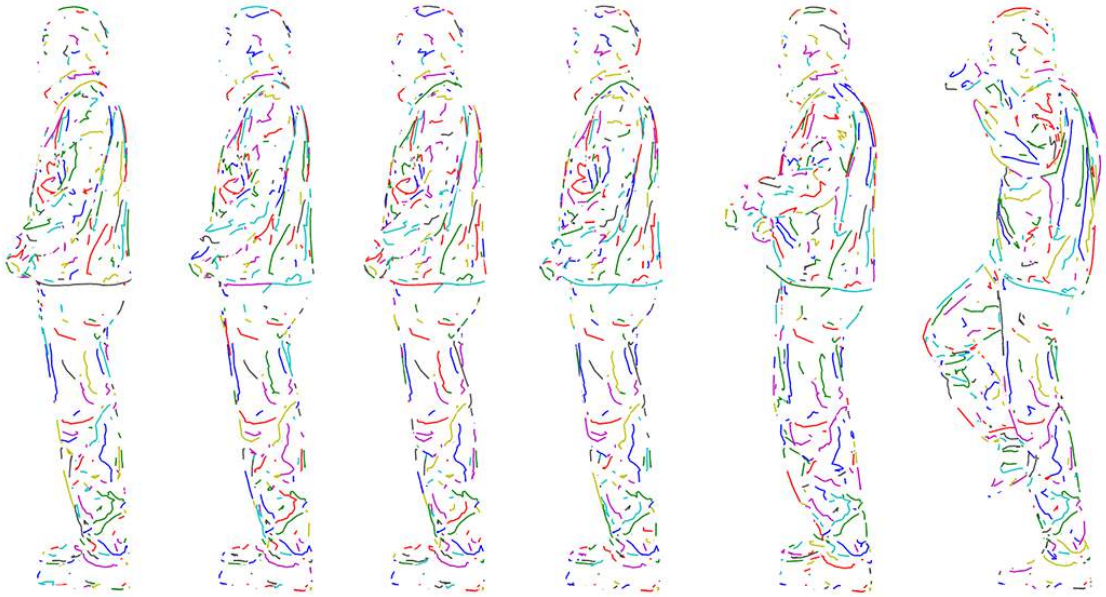


Figure 5.29: Partial frame sequence of line drawing animation from Crane Model. Line type: XPLs.

Table 5.2 shows the average tracing, matching and evolving time for processing ridges and valleys of each pair of models. Note that matching time includes Segmesh optimization but excludes the precomputation time of geodesic distance.

5.5.3 Limitation

The limitation of our approach is that the path trajectory optimization for merging and splitting is made locally (i.e., solely based on the lengths of the compound and component paths). There is no explicit mid- or high- level optimization of such scenarios. As a result, the frames, consisting of paths after merging or splitting, are labeled as frames similar to 'key frames'. These frames should be additionally processed for coherence but it is beyond the scope of this chapter.

5.6 Summary

In this chapter we have introduced an automatic approach consisting of 3D path tracing, matching and evolving. We proposed a novel matching algorithm for 3D path matching. We also proposed a path optimization method to reduce artifacts considerably. The results of our work demonstrate the effectiveness of the approach to generate line drawing animation with temporal coherence.

Currently, the line drawing animation generated from our approach is an intermediate result which can be further refined with line stylization. The lines in the animation, although rendered in screen space, still contain 3D information (e.g., 3D point position, vertex curvature, etc.). The stylization of the lines can be applied in two ways. Firstly, we can directly extend the lines with the techniques proposed by previous work [12, 52] for computer-assisted stylization. Secondly, we can combine 1) the traditional NPR look which can be easily accessed by 2D artists with 2) various 3D effects rendered from 3D information to produce even more fascinating NPR effects.

Chapter 6

A Framework for Stylized Animation from Coherent 3D Paths

In this chapter, we propose an integrated framework for artists to create stylized animations using materials from both 3D models and 2D images. The framework integrates the 3D techniques developed in the previous chapters, such as single-frame line extractions, coherent line thresholding, path tracing and matching, within the production process of 2D animation. The framework is artist driven – it allows artists to edit or stylize line work in 2D, similar to what they do in a traditional animation pipeline, while incorporating 3D information. A 2D stylized animation is created to illustrate the effectiveness of the framework.

6.1 Overview

The framework consists of three core modules, starting with 3D Line Extraction (3DLE), followed by Curve Render Engine (CurRE) and Interactive Tool Set (InterTS). The detailed workflow of the framework is illustrated in Figure 6.1.

Input. The input of the workflow is an animated sequence of 3D triangle mesh models. Currently we support file formats such as OBJ and FBX, which can be obtained from motion capture data or generated by animation software such as Maya or 3Ds Max.

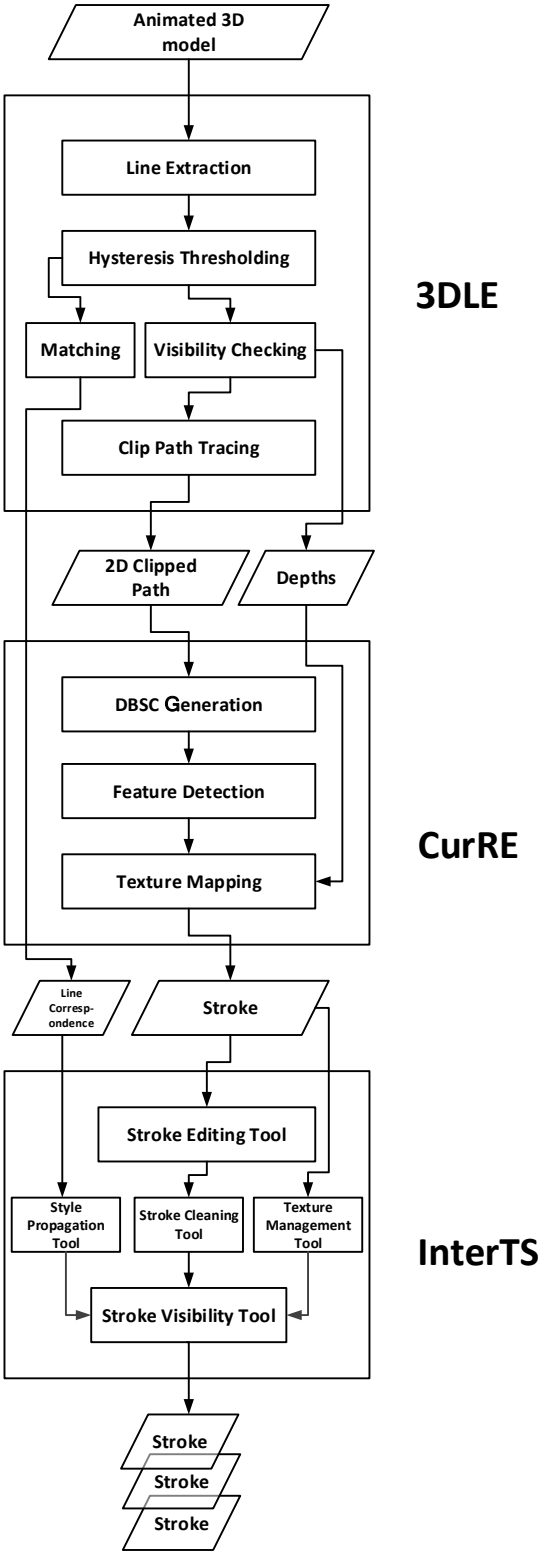


Figure 6.1: Workflow for the core modules in our framework.

We assume all animated meshes have vertex correspondence and share the same mesh topology. The motions of the animation can be due to either camera motions or mesh deformations or both.

3D Line Extraction (3DLE). The main purpose of this module is to generate initial line drawings (i.e., polylines) from which artists can use to create a stylized animation. The module takes animated 3D models as input, and:

1) Extracts 3D lines.

2) Checks 3D line visibility.

3) Traces and projects the 3D lines into vectorized 2D clipped paths (clipped to CurRE canvas) for output.

The line drawing algorithms for 3D line extraction can be one or more algorithms from existing algorithms [27, 42, 51, 76, 109], as well as the algorithm proposed in Chapter 3. Coherent line thresholding algorithm will be the hysteresis thresholding from Chapter 4. Besides the main purpose, this module also outputs two pieces of information for later use. One is the correspondence of 3D paths, obtained from the matching algorithm proposed in Chapter 5. Correspondence will be passed into InterTS module to facilitate the control of the paths. The other is depth information attached to each path which can be passed to CurRE module to create special visual effects. A brief explanation of this module is presented in Section 6.2.

Curve Render Engine (CurRE). The main purpose of this module is to convert rough line drawings generated in 3DLE module into *artistic strokes*, which support a variety of stylization options, and render them in a 2D canvas effectively and efficiently. The CurRE module takes 2D clipped paths as input, and:

1) Generates Disk B-spline Curve (DBSC) from the clipped paths.

2) Detects features of DBSC.

3) Maps textures onto DBSC for stylization.

DBSC is applied in CurRE to represent the artistic strokes. Detailed explanations will be introduced in Section 6.3.

Interactive Tool Set (InterTS). The main purpose of this module is to provide a variety of interactive tools for artists to further edit on the artistic strokes. The most commonly-used tools that we have explored for our framework are listed as follows:

• Stroke editing tool: edit strokes, such as modify strokes (by position, properties, etc), delete strokes or add new strokes.

• Texture management tool: manage texture resources, such as loading and grouping.

• Style propagation tool: propagate style properties of strokes (such as textures and stroke thickness) to corresponding strokes automatically.

• Stroke cleaning tool: help close small gaps or remove intersections between strokes.

Details of each interactive tool will be introduced in Section 6.4.

Output. The output of the workflow is an animation sequence of 2D vectorized strokes, as well as extra information such as texture resources and color information. The animation sequence can be further rendered into images and exported as a 2D stylized animation.

6.2 3D Line Extraction

6.2.1 Line Visibility

A substantial challenge for making 2D stylized line drawing from 3D models is visibility computation. Conventionally, it is unnecessary to consider the visibility of 3D lines in a 3D rendering environment, because the lines are partially or fully occluded by the model itself during rendering [20]. However, the CurRE rendering is purely 2D, where the occlusion information are not readily available. This enforces explicit visibility computation of lines in 3DLE.

Our computation for the visibility of 3D lines is based on the visibility of mesh triangles of the 3D models. We have yet to consider the situation when a triangle is partially occluded in 3DLE. A more sophisticated occlusion test will be investigated to address this issue in the future.

6.2.2 Line Vectorization

As a result of the visibility computation of the 3D lines, visible and invisible lines are obtained separately as two sets of lines. The following procedures are applied respectively to each set to obtain vectorized 2D lines:

1) Path tracing (c.f. Chapter 5, Section 5.2) for vectorized 3D paths.

2) 2D projection w.r.t the clipping window of the canvas of CurRE for 2D clipped paths.

A point on the 2D clipped path is defined as a 3D point. The x and y coordinates store the projected screen coordinate, while z coordinate stores the depth information.

6.2.3 Correspondence

The correspondence used in our framework is for the strokes defined in CurRE module. This correspondence is derived from path correspondence directly, and inherits the complicated mapping relationship of the path correspondence, i.e., one-to-one, many-to-one, one-to-many, zero-to-one and one-to-zero mappings (c.f. Chapter 5, Section 5.3). In our framework the stroke correspondence is stored by indices. Each frame maintains an index map. The mapping relationship is handled through logical index mapping rather than physical duplication of strokes. Strokes in different frames sharing the same index are correspondent.

Strokes and indices in index map are not bijective. Each stroke in one frame is indexed by a unique number. However, not every index refers to a real stroke. Index duplications and *place holders* are the exceptional cases. Index duplication is applied

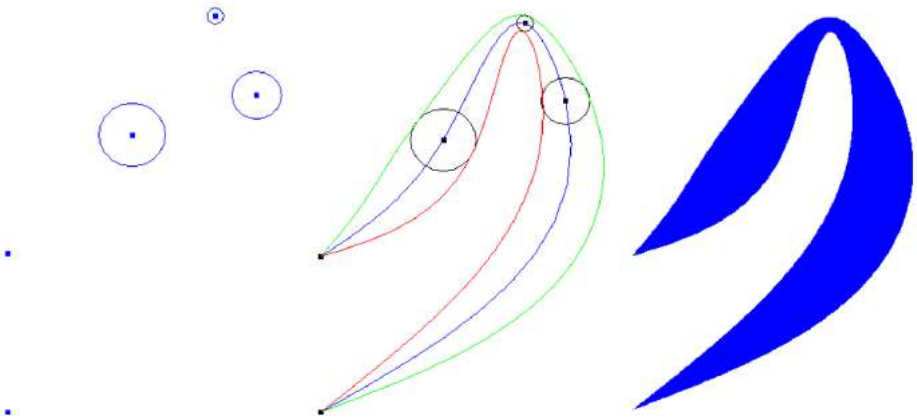


Figure 6.2: An open DBSC (rendered in blue region) interpolated from a set of control points and radii. Images courtesy of Seah et al. [91].

in the map for many-to-one/one-to-many mappings. Place holder is a special symbol indicating non-existing stroke such as zero-to-one/one-to-zero mappings.

6.3 Artistic Stroke

The representation of an artistic stroke is fundamental for computer-generated paintings and animations. Vectorization is commonly required for such strokes for further editing. Efficient storage of stroke data is also a key problem for vector-based production.

6.3.1 Disk B-Spline Curve

We introduce the representation of freeform 2D shapes called Disk B-spline curve [91, 108], short for DBSC. It is a generalization of Disk Bezier curve [67] which defines a **region** as well as center curve. Some advantages of using DBSC to represent strokes are:

- Solid mathematical fundamentals
- Flexible manipulations like deformation
- Small dataset

Definition. Assuming $\mathbf{N_{i,d}(u)}$ is the $\mathbf{i}^{th}$ B-spline basis [80] of degree $\mathbf{d}$ with knot vector $\mathbf{U}$ containing $\mathbf{m+1}$ knots, where the first $\mathbf{d+1}$ and the last $\mathbf{d+1}$ knots are identical:

$$\mathbf{U} = [\mathbf{u_0}, ...\mathbf{u_m}] = \{\underbrace{a, ...a}_{\mathbf{d+1}}, \mathbf{u_{d+1}}, ..., \mathbf{u_{m-d-1}}, \underbrace{b..., b}_{\mathbf{d+1}}\} \quad \text{(Eq. 6.1)}$$

and a disk $\mathbf{D}$ in the plane $\mathbb{R}^2$ is defined to be a set:

$$\langle \mathbf{D} \rangle := \{\mathbf{x} \in \mathbb{R}^2 \mid\mid \mathbf{x} - \mathbf{c} \mid \leq \mathbf{r}, \mathbf{c} \in \mathbb{R}^2, \mathbf{r} \in \mathbb{R}^+\} \quad \text{(Eq. 6.2)}$$

which can also be written as

$$\langle \mathbf{D} \rangle = \langle \mathbf{c};\ \mathbf{r} \rangle \quad \text{(Eq. 6.3)}$$

where $\mathbf{c}$ is the center of the disk, $\mathbf{r}$ is the radius of the disk, the set of real numbers is denoted by $\mathbb{R}$ and the set of nonnegative real numbers by $\mathbb{R}^+$.

A Disk B-spline Curve $\mathbf{\Gamma}$ is defined as:

$$\mathbf{\Gamma} = \langle \mathbf{D} \rangle(\mathbf{u}) = \sum_{\mathbf{i=0}}^{\mathbf{n}} \mathbf{N_{i,d}}(\mathbf{u}) \langle \mathbf{p_{c_i}};\ \mathbf{r_{c_i}} \rangle \quad \text{(Eq. 6.4)}$$

where $\mathbf{p_c}$ is the control point, $\mathbf{r_c}$ is the control radius.

A DBSC can be viewed as two parts: a center curve (a B-spline curve) and a radius function (a B-spline scalar function). Therefore, most of properties and algorithms of the DBSC can be obtained by applying B-spline curve and function to the two parts of DBSC respectively.

Visualization. Given a group of points and a group of values setting as control points and radii respectively, a DBSC can be obtained by interpolation using B-spline curve interpolation and scalar function. Figure 6.2 illustrates an example of an open DBSC from the given points and radii.

6.3.2 Texture Mapping

DBSC forms a region which we can triangulate for texture mapping. The result of the mapping is visualized as stylization, which is resolution-independent. See Figure 6.3 for illustration.

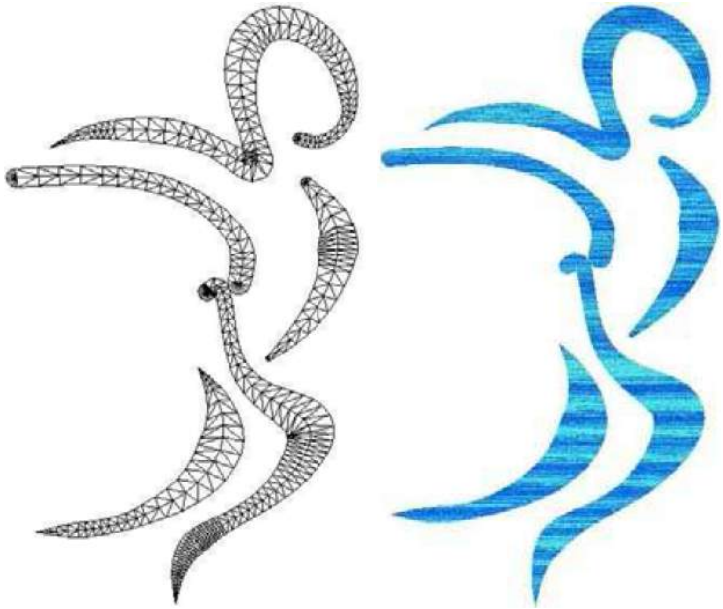


Figure 6.3: Triangulating and texture mapping with DBSCs. Images courtesy of Seah et al. [91].

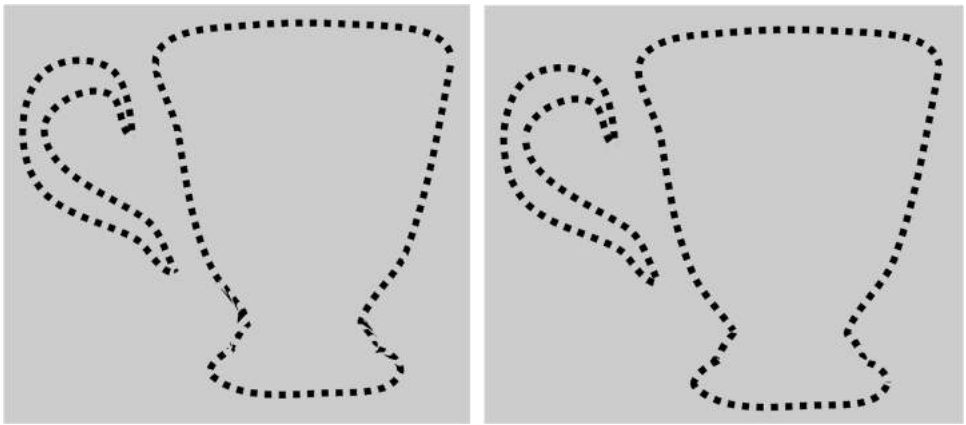


Figure 6.4: Results for mapping a checker texture without (left) or with (right) feature detection for DBSCs.

6.3.3 Feature Detection

If a DBSC has sharp turns, a texture mapped onto the triangulated region may be distorted seriously. The distortion problem is especially obvious for those textures with regular patterns. The resulting DBSC stylization therefore contains visual artifacts. Thus, we apply automatic feature detection on DBSC before they are sent for triangulation. A feature of a DBSC resides at where the curvature of the curve reaches its local extremum, i.e., a sharp corner. In practice, this location is fitted to a nearby knot of the curve and saved as a feature knot.

Given the feature knots, we apply segmentations to the curve which results in several sub-curves. We apply triangulation and texture mapping to each sub-curve, by considering their relative positions in the original curve. The comparison results of a checker texture mapping with and without feature detection for a curve are shown in Figure 6.4. As we can see from the figure, the rendered checkerboard stylization looks more distorted in the sharp corners of the cup in the left subfigure than that in the right subfigure, especially those areas locating at the cup base.

6.4 Interactive Tool Set

In this section we introduce an interactive tool set (InterTS) that artists can use to create stylized line work.

6.4.1 Stroke Editing Tool

Artists are allowed to apply editing functions such as add, modify or delete to strokes through mouse interactions. For each function the stroke index will be handled by the tool respectively and automatically.

Add. The tool contains different types of strokes to be added. The following types are supported currently: free-form drawing (defined as DBSC), Bezier curve, polyline, line

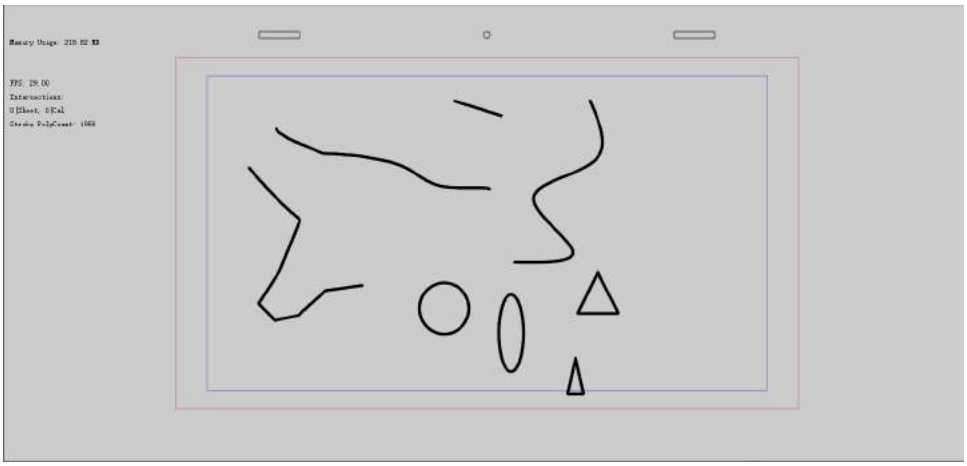


Figure 6.5: Editing tool: all supported types of strokes.

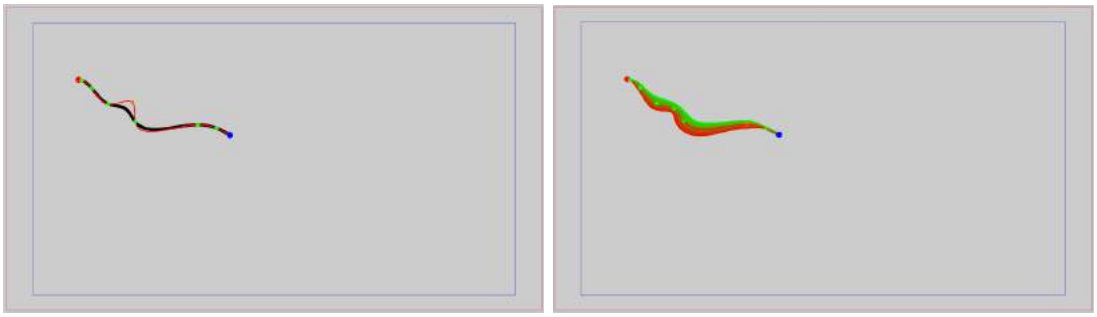


Figure 6.6: Editing tool: modify stroke position (left) and thickness (right).

segment, eclipse and triangle. Figure 6.5 illustrates all the supported types of strokes. The newly added strokes will be given indices in order, starting with the current largest index plus 1.

Modify. The tool provides the following modifications to a stroke: the position, the thickness and the color. The first two are accessed through the control points of the stroke and manipulated by mouse dragging. The modification of the color of the stroke is accessed through a pre-defined color panel. See Figure 6.6 for illustration of the first two modifications.

Delete. In order not to destroy stroke correspondence across frames, deleting stroke does not impact the index map, but place-holder label will be attached to the index

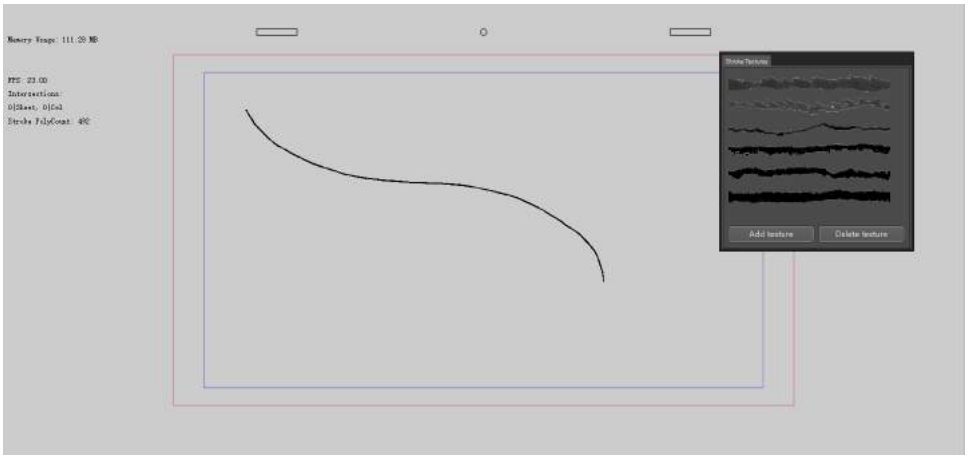


Figure 6.7: Texture management tool.

corresponding to the deleted stroke.

6.4.2 Texture Management Tool

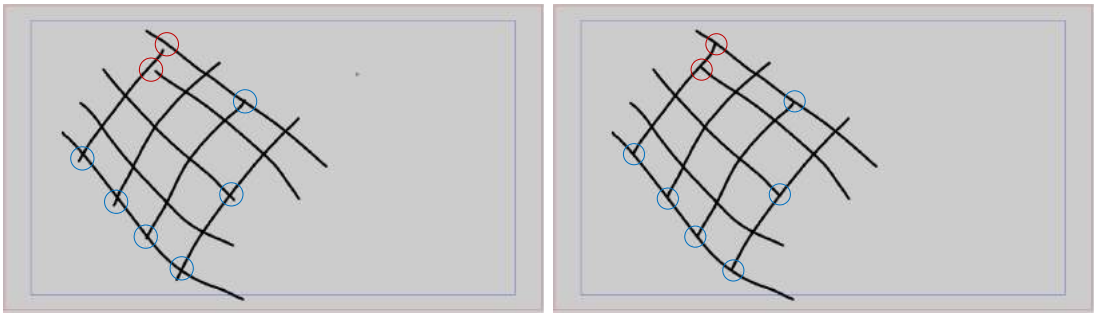
Our framework supports multiple textures input. Artists are allowed to manage texture resources, such as loading and grouping, through texture management tool. A texture, or a set of textures selected from the tool panel can be applied to strokes. See Figure 6.7 for illustration. Re-application of textures can be done as many times as needed.

6.4.3 Style Propagation Tool

This tool is developed to automatically propagate the style of strokes in one frame to corresponding strokes in the rest of the frames of an animation sequence. This avoids manual modifications frame-by-frame. Currently the styles for propagation are: stroke color, thickness and textures.

6.4.4 Stroke Cleaning Tool

This tool is developed to close small gaps and intersections between existing strokes in one frame. Firstly, artists need to select a target region containing all the strokes for clean-up. The tool will automatically detect stroke gaps and intersections within the



6.8(a) Before 6.8(b) After

Figure 6.8: Cleaning tool: cleaning gaps (highlighted in red circles) and intersections (highlighted in blue circles) between strokes.

region. As a result, all the gaps which are smaller than a preset minimum distance will be connected. See Figure 6.8 for illustrations. The minimum distance can be changed through a tool panel.

6.5 Results and Discussions

6.5.1 Style Examples

With the information (mainly depths) from 3D models and its own characteristics, a DBSC is capable of supporting various styles in 2D, as stated in the following.

Varied stroke width is used to simulate the effect of perspective projection for strokes, by setting different control radii to each control point of the DBSC strokes. The value of the radius of each control point comes from the depth of its 3D point counterpart. A large width for stroke indicates a nearer position to the camera, while a small width indicates a farther position to the camera. This style is automatically generated in CurRE. Figure 6.9 illustrates an example of a scene with varied stroke widths.

Various styles in one frame are achieved because texture information is attached to each stroke rather than the entire frame. Figure 6.10 demonstrates this by applying the textures shown at the bottom to the strokes at the top.

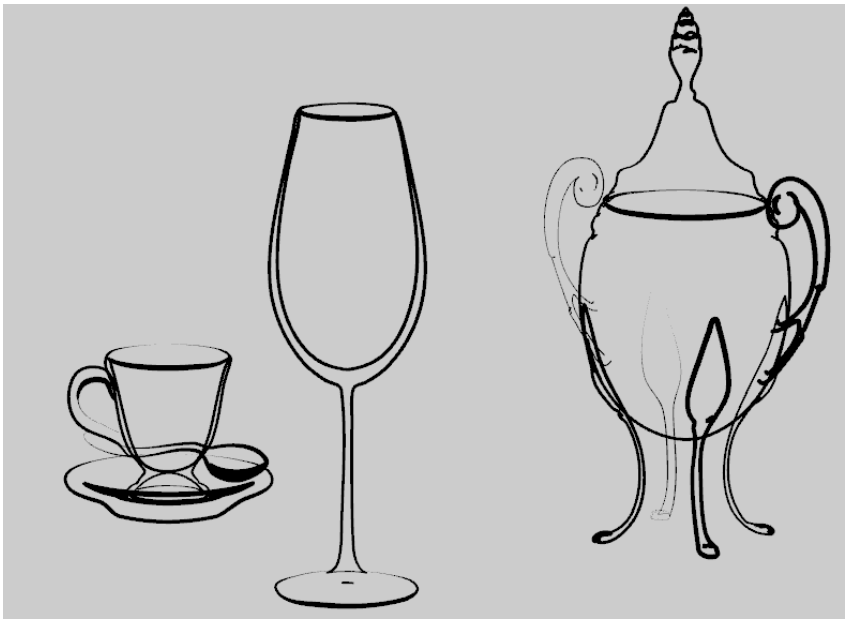


Figure 6.9: Strokes with varied widths. Model: Tableware. Note that visibility checking is not applied in this image.

Figure 6.10: Various styles in one frame.

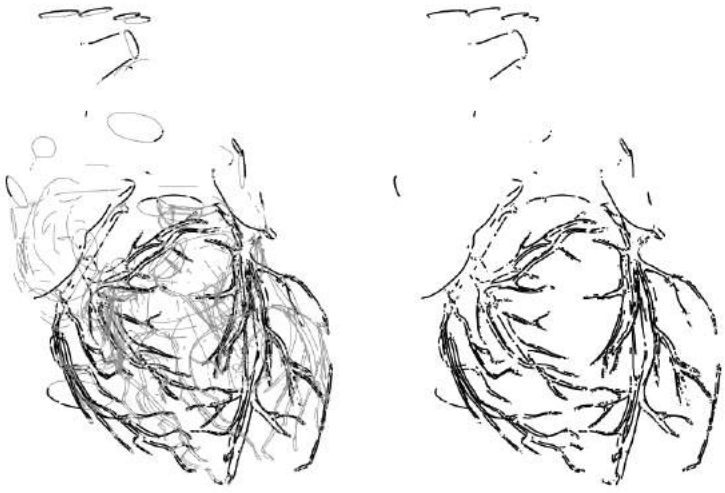


Figure 6.11: Heart model. Left: visible lines (thick black lines) and invisible lines (thin gray lines). Right: visible lines only.

Hidden-line effect is achieved by applying different styles on visible and invisible strokes. The visibility information is derived from 3DLE using line occlusion test, and exported together with lines into CurRE for stylization. Figure 6.11 illustrates this effect on a heart model.

6.5.2 2D Stylized Animation

Figure 6.12 shows a 2D animation that our team has worked with, which is made with a 2D animation software, as well as its 3D Line Extraction plugin developed from the 3D techniques proposed in this thesis. This animation is a hybrid animation. The Dracula model (i.e., the human character) is hand-drawn and scanned into the computer and vectorised. The cup, glass and sugar pot are all 3D FBX models which were animated to match with their original sketches. The mosquito model is manually drawn using the software. We present a basic workflow of our hybrid animation. In the software, we open the plugin from the menu. We load the FBX model within the plugin, and select one or more types of lines to draw. Lines are then automatically generated and imported into

the software. Once in the software, we can modify, delete and add new lines, as well as organizing them into layers for easier editing. Additionally, we can choose to stylize these lines and modify their thicknesses, either propagating a source style within a layer or adjusting each line individually. The entire animation is available online [114].

6.5.3 Performance

The entire processing was done on a 3.3GHz Intel Xeon CPU with 8GB RAM. The system performs at interactive rates for CurRE and 3DLE with models of moderate complexity (time is ranging from 20 to 50 fps for CurRE, 10 to 20 fps for 3DLE).

6.6 Summary

We have proposed an integrated framework incorporating automatic 3D line extraction to create 2D stylized animation. The framework provides a variety of interactive tools for artists to further manipulate, edit or deform strokes. A 2D stylized animation is created using the framework to demonstrate its effectiveness.

Figure 6.12: 2D stylized animation produced using our framework.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we have proposed an integrated framework towards fulfilling the task of releasing artists from the laborious drawings and allowing them to focus more on the part of creative work. The framework incorporates automatic 3D line extraction from which artists can begin with to create 2D animation. It provides as well a variety of interactive tools for artists to edit or stylize line work in 2D, similar to what they do in a traditional animation pipeline. The framework is artist-driven, where artists fully control the entire production process for 2D animation.

We have also proposed a few 3D algorithms which are integrated into our framework: 1) a novel line drawing algorithm called eXtended Phong Lines to extract lines automatically from 3D models; 2) an effective method to extract 3D lines from 3D models using hysteresis thresholding for temporal coherence; and 3) a fully automatic path matching algorithm to create correspondence of 3D paths across frames. Each of the three algorithms is an important contribution to NPR.

To demonstrate the effectiveness of our framework and respective NPR algorithms, we have performed a wide range of experiments on animated 3D models as well as a real animation production. We collaborated with 2D and 3D artists to create a 2D stylized animation.

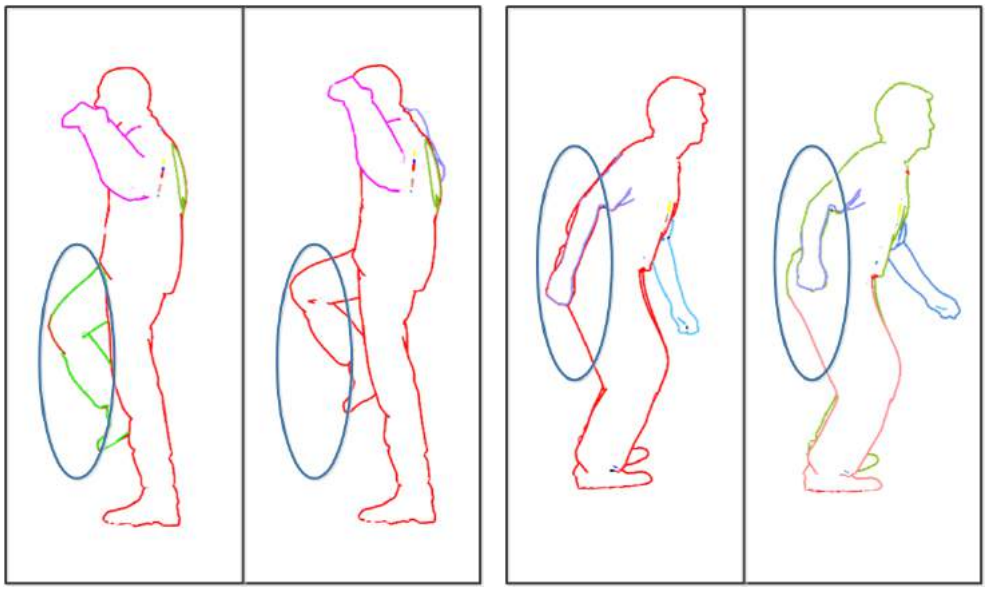
7.2 Future Work

There are several research areas that we can explore from our proposed framework. The possible improvements could be easily applied or integrated into our existing framework.

3D Line Drawings. Up to date, a great number of line drawings have been developed by artists and illustrators to convey shapes effectively and impressively. However, only a very small set of styles are nicely emulated using computer at the current stage. Computer-generated line drawings still cannot be of the same quality as hand-made drawings, which are usually the only choice for high quality. Even though we have proposed X-Phong lines to exploit feature lines, line extraction is still far away from comprehending artists' idea. Therefore, our future research includes this part.

We are currently at the initial stage of understanding where and how many lines to draw on the surface, i.e., calculating the feature lines based merely on viewpoint or geometry information. In the future, cognitive information (e.g., human perception [29, 115]) may be incorporated for automatic or semi-automatic line extraction. Effective and expressive illustrations for conveying shapes may be an integration of both computation capability and artificial intelligence.

Line Topology. Vectorizing and matching lines in 2D and 3D animations are important research areas in computer vision and graphics. The space distribution of the lines in a 2D image, or on a 3D surface is treated as line topology [9, 11, 12, 52]. The topology often changes in animations, because lines in two animation frames are not guaranteed to have the same numbers, lengths and positions. A few methods have been proposed to handle line topology problem. However, they either lack accuracy for complex scenarios or only suitable for a certain type of lines such as silhouettes. Compared to previous research [8, 11, 68], our proposed method is more general to handle line topology. But topology built from our method is based on a purely geometric metric, which is not applicable to some extreme scenarios where geometric metric alone cannot handle the



7.1(a) 7.1(b)

Figure 7.1: Unhandled cases of (a) merging and (b) splitting scenarios.

topology. For example, Figure 7.1 illustrates two extreme cases which rely on human interpretation. The paths highlighted by the eclipse merge into one path in Figure 7.1(a), while one path splits into two in Figure 7.1(b). In the future, our framework could involve machine learning algorithms, and develop a fully- or semi-automatic method, to solve line topology problem.

Model/Stroke Manipulation. Sketching is a natural and easy way to communicate ideas and create scenes quickly. The 2D lines and strokes are the core elements to the formation of sketches. Sketch-based modeling has been substantially explored by many researchers (c.f. survey papers [24, 77]). However, existing modeling and sketching methods pose many difficulties for creating complex 3D animation, namely, they are not intuitive and require skillful sculpting and ad-hoc tweaking by artists [49, 96, 102]. Our proposed framework provides a ready-to-use sketching interface, which may be used for the manipulation of lines for animated models.

In the future, our current framework could be extended to a general 3D/2D model-

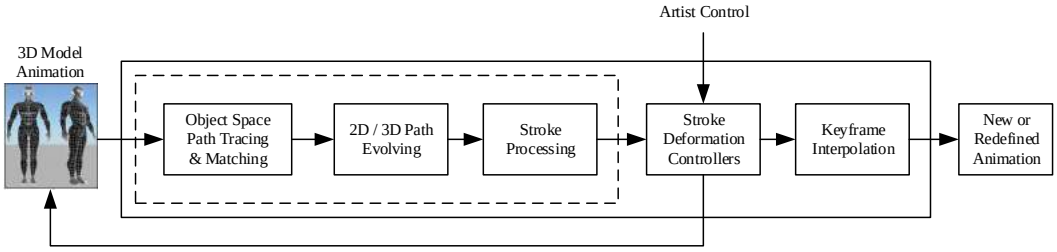


Figure 7.2: Framework overview for model/stroke manipulation.

based framework for producing 3D animation via 2D lines and vice-versa. Benefiting from available 3D models and corresponding lines, the framework would create similar 3D animation via minor tweaking on current animated 3D model. The links established between 2D and 3D through line extraction from 3D animation provide us with the insight and possibilities to mix 2D and 3D components seamlessly and easily. Figure 7.2 illustrates a blueprint of future framework on model/stroke manipulation for 3D animation.

References

[1] Retas!Pro. http://www.celsys.co.jp/en/index.html.

[2] Tf3dm. http://tf3dm.com.

[3] Toon boom animation. https://www.toonboom.com.

[4] Toonz. http://www.toonz.com.

[5] Aseem Agarwala, Aaron Hertzmann, David H. Salesin, and Steven M. Seitz. Keyframe-based tracking for rotoscoping and animation. *ACM Transaction on Graphics*, 23(3):584–591, August 2004.

[6] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Lévy, and Mathieu Desbrun. Anisotropic polygonal remeshing. In *SIGGRAPH '03: Proceedings of the 30th annual conference on Computer graphics and interactive techniques*, pages 485–493, New York, NY, USA, 2003. ACM.

[7] Arthur Appel. The notion of quantitative invisibility and the machine rendering of solids. In *Proceedings of the 1967 22nd national conference*, pages 387–393, New York, NY, USA, 1967. ACM.

[8] Pascal Barla, Joëlle Thollot, and François X. Sillion. Geometric clustering for line drawing simplification. In *SIGGRAPH '05: Proceedings of the 32nd annual conference on Computer graphics and interactive techniques*, page 96, New York, NY, USA, 2005. ACM.

[9] Pierre Bénard, Adrien Bousseau, and Joëlle Thollot. State-of-the-art report on temporal coherence for stylized animations. *Computer Graphics Forum*, 30(8):367–386, 2011.

[10] Pierre Bénard, Forrester Cole, Aleksey Golovinskiy, and Adam Finkelstein. Self-similar texture for coherent line stylization. In *NPAR '10: Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, pages 91–97, New York, NY, USA, 2010. ACM.

[11] Pierre Bénard, Aaron Hertzmann, and Michael Kass. Computing smooth surface contours with accurate topology. *ACM Transaction on Graphics*, 33(2):19:1–19:21, April 2014.

[12] Pierre Bénard, Jingwan Lu, Forrester Cole, Adam Finkelstein, and Joëlle Thollot. Active strokes: coherent line stylization for animated 3D models. In *NPAR '12: Proceedings of the 10th International Symposium on Non-Photorealistic Animation and Rendering*, pages 37–46, Aire-la-Ville, Switzerland, Switzerland, 2012. Eurographics Association.

[13] Lubomir Bourdev. Rendering nonphotorealistic strokes with temporal and arc-length coherence. Technical report, 1998.

[14] Bert Buchholz, Noura Faraj, Sylvain Paris, Elmar Eisemann, and Tamy Boubekeur. Spatio-temporal analysis for parameterizing animated lines. In *NPAR '11: Proceedings of the 9th International Symposium on Non-Photorealistic animation and rendering*, pages 85–92, New York, NY, USA, 2011. ACM.

[15] Franck Caniard and Roland W. Fleming. Distortion in 3d shape estimation with changes in illumination. In *APGV '07: Proceedings of the 4th symposium on Applied*

perception in graphics and visualization, pages 99–105, New York, NY, USA, 2007. ACM.

[16] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[17] Edwin Catmull. The problems of computer-assisted animation. *ACM Transaction on Graphics*, 12(3):348–353, August 1978.

[18] Roberto Cipolla and Peter Giblin. *Visual Motion of Curves and Surfaces*. Cambridge University Press, 2000.

[19] Forrester Cole, Doug DeCarlo, and Adam Finkelstain. In *SIGGRAPH '08: ACM SIGGRAPH 2008 Courses*, New York, NY, USA, 2008. ACM.

[20] Forrester Cole, Doug DeCarlo, Adam Finkelstein, Kenrick Kin, Keith Morley, and Anthony Santella. Directing gaze in 3D models with stylized focus. *Eurographics Symposium on Rendering*, pages 377–387, June 2006.

[21] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. Where do people draw lines? In *SIGGRAPH '08: Proceedings of the 35th annual conference on Computer graphics and interactive techniques*, pages 1–11, New York, NY, USA, 2008. ACM.

[22] Forrester Cole, Kevin Sanik, Doug DeCarlo, Adam Finkelstein, Thomas Funkhouser, Szymon Rusinkiewicz, and Manish Singh. How well do line drawings depict shape? In *SIGGRAPH '09: Proceedings of the 36th annual conference on Computer graphics and interactive techniques*, pages 1–9, New York, NY, USA, 2009. ACM.

[23] John P. Collomosse, David Rowntree, and Peter M. Hall. Stroke surfaces: Temporally coherent artistic animations from video. *IEEE Transactions on Visualization and Computer Graphics*, 11:540–549, September 2005.

[24] Matthew T. Cook and Arvin Agah. A survey of sketch-based 3-D Modeling Techniques. *Interacting with Computers*, 21(3):201–211, July 2009.

[25] Cassidy Curtis. Non-photorealistic animation. In *SIGGRAPH '99: ACM SIGGRAPH 1999 Courses*, 1999.

[26] Doug DeCarlo, Adam Finkelstein, and Szymon Rusinkiewicz. Interactive rendering of suggestive contours with temporal coherence. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic Animation and Rendering*, pages 15–145, New York, NY, USA, 2004. ACM.

[27] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. In *SIGGRAPH '03: Proceedings of the 30th annual conference on Computer graphics and interactive techniques*, pages 848–855, New York, NY, USA, 2003. ACM.

[28] Doug DeCarlo and Szymon Rusinkiewicz. Highlight lines for conveying shape. In *NPAR '07: Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, pages 63–70, New York, NY, USA, 2007. ACM.

[29] Doug DeCarlo and Matthew Stone. Visual explanations. In *NPAR '10: Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, pages 173–178, New York, NY, USA, 2010. ACM.

[30] Manfredo P. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, February 1976.

[31] Jean-Daniel Fekete, Érick Bizouarn, Éric Cournarie, Thierry Galas, and Frédéric Taillefer. Tictactoon: a paperless system for professional 2d animation. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 79–90, New York, NY, USA, 1995. ACM.

[32] John Fujii. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, New York, NY, USA, 2005. ACM.

[33] Ahna Girshick, Victoria Interrante, Steven Haker, and Todd Lemoine. Line direction matters: an argument for the use of principal directions in 3d line drawings. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 43–52, New York, NY, USA, 2000. ACM.

[34] Steven Gold, Anand Rangarajan, Chien-ping Lu, and Eric Mjolsness. New algorithms for 2D and 3D point matching: pose estimation and correspondence. *Pattern Recognition*, 31:957–964, 1997.

[35] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 447–452, New York, NY, USA, 1998. ACM.

[36] Bruce Gooch and Amy Gooch. *Non-Photorealistic Rendering*. A. K. Peters, Ltd., Natick, MA, USA, 2001.

[37] Bruce Gooch, Erik Reinhard, and Amy Gooch. Human facial illustrations: Creation and psychophysical evaluation. *ACM Transaction on Graphics*, 23(1):27–44, 2004.

[38] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld. Interactive technical illustration. In *I3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 31–38, 1999.

[39] Stéphane Grabli, Emmanuel Turquin, Frédo Durand, and François X. Sillion. Programmable rendering of line drawing from 3d scenes. *ACM Transaction on Graphics*, 29(2):1–20, 2010.

[40] Donald D. Hearn and M. Pauline Baker. *Computer Graphics with OpenGL*. Prentice Hall Professional Technical Reference, 2003.

[41] Aaron Hertzmann. Introduction to 3D non-photorealistic rendering: silhouettes and outlines. In *SIGGRAPH '99: ACM SIGGRAPH 1999 Courses*, 1999.

[42] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 517–526, New York, NY, USA, 2000. ACM.

[43] Elaine R.S. Hodges. *The Guide Handbook of Scientific Illustration*. Wiley; Second Edition, 2003.

[44] Siu Chi Hsu and Irene H. H. Lee. Drawing and animation using skeletal strokes. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 109–118, New York, NY, USA, 1994. ACM.

[45] Xfrog Inc. http://xfrog.com/category/xfrogplants.html.

[46] T. Isenberg, B. Freudenberg, N. Halper, S. Schlechtweg, and T. Strothotte. A developer's guide to silhouette algorithms for polygonal models. *IEEE Computer Graphics and Applications*, 23(4):28–37, July 2003.

[47] Tobias Isenberg, Nick Halper, and Thomas Strothotte. Stylizing silhouettes at interactive rates: From silhouette edges to silhouette strokes. *Computer Graphics Forum*, 21(3):249–258, 2002.

[48] Tobias Isenberg, Petra Neumann, Sheelagh Carpendale, Mario Costa Sousa, and Joaquim A. Jorge. Non-photorealistic rendering in context: an observational study. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, pages 115–126, New York, NY, USA, 2006. ACM.

[49] Eakta Jain, Yaser Sheikh, Moshe Mahler, and Jessica Hodgins. Three-dimensional proxies for hand-drawn characters. *ACM Transaction on Graphics*, 31(1):8:1–8:16, February 2012.

[50] Jot. http://jot.cs.princeton.edu/.

[51] Tilke Judd, Frédo Durand, and Edward H. Adelson. Apparent ridges for line drawing. *ACM Transaction on Graphics*, 26(3):19, 2007.

[52] Robert D. Kalnins, Philip L. Davidson, Lee Markosian, and Adam Finkelstein. Coherent stylized silhouettes. *ACM Transaction on Graphics*, 22(3):856–861, 2003.

[53] Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes, and Adam Finkelstein. WYSIWYG NPR: drawing strokes directly on 3D models. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 755–762, New York, NY, USA, 2002. ACM.

[54] Henry Kang, Seungyong Lee, and Charles K. Chui. Coherent line drawing. In *NPAR '07: Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering*, pages 43–50, New York, NY, USA, 2007. ACM.

[55] Kevin Karsch and John C. Hart. Snaxels on a plane. In *NPAR '11: Proceedings of the 9th International Symposium on Non-Photorealistic Animation and Rendering*, pages 35–42, New York, NY, USA, 2011. ACM.

[56] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[57] Yongjin Kim, Jingyi Yu, Xuan Yu, and Seungyong Lee. Line-art illustration of dynamic and specular surfaces. *ACM Transactions on Graphics (SIGGRAPH ASIA 2008)*, 27(5), December 2008.

[58] Jan J. Koenderink. *Solid Shape*. Artificial Intelligence Series. MIT Press, Cambridge, MA, USA, 1990.

[59] Jan J. Koenderink, Andrea J. Van Doorn, and Astrid M.L. Kappers. Surface perception in picutres. *Perception & Psychophysics*, 52:487–496, 1992.

[60] Michael Kolomenkin, Ilan Shimshoni, and Ayellet Tal. Demarcating curves for shape illustration. *ACM Transaction on Graphics*, 27(5):1–9, 2008.

[61] A.M Krasnosel'skii, M.A Krasnosel'skii, and A.V Pokrovskii. Encyclopedia of mathematics. http://www.encyclopediaofmath.org/index.php?title=Non-ideal_relay&oldid=13966.

[62] Jan Eric Kyprianidis and Henry Kang. Image and video abstraction by coherence-enhancing filtering. *Computer Graphics Forum*, 30(2):593–602, 2011.

[63] Kai Lawonn, Alexandra Baer, Patrick Saalfeld, and Bernhard Preim. Comparative Evaluation of Feature Line Techniques for Shape Depiction. In Jan Bender, Arjan Kuijper, Tatiana von Landesberger, Holger Theisel, and Philipp Urban, editors, *Vision, Modeling & Visualization*. The Eurographics Association, 2014.

[64] Chang Ha Lee, Xuejun Hao, and Amitabh Varshney. Geometry-dependent lighting. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):197–207, March 2006.

[65] Hyo Keun Lee, Young Sup Park, and Kyung Hyun Yoon. A study on the dynamic painterly stroke generation for 3D animation. In *ICCSA'03: Proceedings of the 2003 international conference on Computational Science and its Applications*, pages 317–325, Berlin, Heidelberg, 2003. Springer-Verlag.

[66] Yunjin Lee, Lee Markosian, Seungyong Lee, and John F. Hughes. Line drawings via abstracted shading. In *SIGGRAPH '07: Proceedings of the 34th annual conference on Computer graphics and interactive techniques*, page 18, New York, NY, USA, 2007. ACM.

[67] Qun Lin and J.G. Rokne. Disk bézier curves. *Computer Aided Geometric Design*, 15(7):721 – 737, 1998.

[68] Dongquan Liu, Quan Chen, Jun Yu, Huiqin Gu, Dacheng Tao, and Hock Soon Seah. Stroke correspondence construction using manifold learning. *Computer Graphics Forum*, 30(8):2194–2207, 2011.

[69] Alan P. Mangan and Ross T. Whitaker. Partitioning 3d surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 5:308–321, October 1999.

[70] Lee Markosian. *Art-based modeling and rendering for computer graphics.* PhD thesis, Providence, RI, USA, 2000. Adviser-Hughes, John F.

[71] Lee Markosian, Michael A. Kowalski, Daniel Goldstein, Samuel J. Trychin, John F. Hughes, and Lubomir D. Bourdev. Real-time nonphotorealistic rendering. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 415–420, New York, NY, USA, 1997. ACM.

[72] I.D. Mayergoyz. *Mathematical Models of Hysteresis and Their Applications.* Electromagnetism. Elsevier Science, New York, 2003.

[73] ID Mayergoyz and G Friedman. Generalized preisach model of hysteresis. *IEEE Transactions on Magnetics*, 24(1):212–217, 1988.

[74] Barbara J. Meier. Painterly rendering for animation. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 477–484, New York, NY, USA, 1996. ACM.

[75] J. D. Northrup and Lee Markosian. Artistic silhouettes: a hybrid approach. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 31–37, New York, NY, USA, 2000. ACM.

[76] Yutaka Ohtake, Alexander Belyaev, and HansPeter Seidel. Ridge-valley lines on meshes via implicit surface fitting. In *SIGGRAPH '04: Proceedings of the 31st annual conference on Computer graphics and interactive techniques*, pages 609–612, New York, NY, USA, 2004. ACM.

[77] Luke Olsen, Faramarz F. Samavati, Mario Costa Sousa, and Joaquim A. Jorge. Technical section: Sketch-based modeling: A survey. *Computer Graphics*, 33(1):85–103, February 2009.

[78] James P. O'Shea, Martin S. Banks, and Maneesh Agrawala. The assumed light direction for perceiving shape from shading. In *APGV '08: Proceedings of the 5th symposium on Applied perception in graphics and visualization*, pages 135–142, New York, NY, USA, 2008. ACM.

[79] Stephen Palmer. *Vision Science: Photons to Phenomenology.* MIT Press, Cambridges, MA, 2000.

[80] Les Piegl and Wayne Tiller. *The NURBS Book (2nd Ed.).* Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[81] Alexei Pokrovskii. http://euclid.ucc.ie/hysteresis.

[82] Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, page 581, New York, NY, USA, 2001. ACM.

[83] Ramesh Raskar and Michael Cohen. Image precision silhouette edges. In *I3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 135–140, New York, NY, USA, 1999. ACM.

[84] Tobias Ritschel, Kaleigh Smith, Matthias Ihrke, Thorsten Grosch, Karol Myszkowski, and Hans-Peter Seidel. 3d unsharp masking for scene coherent enhancement. In *SIGGRAPH '08: Proceedings of the 35th annual conference on Computer graphics and interactive techniques*, pages 1–8, New York, NY, USA, 2008. ACM.

[85] Alec Rivers, Takeo Igarashi, and Frédo Durand. 2.5D cartoon models. *ACM Transactions on Graphics*, 29(4):59:1–59:7, July 2010.

[86] Szymon Rusinkiewicz. Estimating curvatures and their derivatives on triangle meshes. In *3DPVT'04: Symposium on 3D Data processing, visualization, and transmission*, September 2004.

[87] Szymon Rusinkiewicz, Michael Burns, and Doug DeCarlo. Exaggerated shading for depicting shape and detail. *ACM Transaction on Graphics*, 25(3), July 2006.

[88] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-D shapes. *ACM Transaction on Graphics*, 24(4):197–206, 1990.

[89] Anthony Santella and Doug DeCarlo. Visual interest and npr: An evaluation and manifesto. In *NPAR '04: Proceedings of the 3rd International Symposium on Non-photorealistic animation and rendering*, pages 71–150, New York, NY, USA, 2004. ACM.

[90] Michael Schwarz and Marc Stamminger. On predicting visual popping in dynamic scenes. In *APGV '09: Proceedings of the 6th symposium on Applied perception in graphics and visualization*, pages 93–100, 2009.

[91] Hock Soon Seah, Zhongke Wu, Feng Tian, Xian Xiao, and Boya Xie. Artistic brushstroke representation and animation with disk b-spline curve. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 88–93, New York, NY, USA, 2005. ACM.

[92] Thomas Strothotte and Stefan Schlechtweg. *Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation*. Morgan Kaufmann, San Francisco, 2002.

[93] Qian Sun, Long Zhang, Minqi Zhang, Xiang Ying, Shi-Qing Xin, Jiazhi Xia, and Ying He. Texture brush: An interactive surface texturing interface. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '13, pages 153–160, New York, NY, USA, 2013.

[94] Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J. Gortler, and Hugues Hoppe. Fast exact and approximate geodesics on meshes. *ACM Transaction on Graphics*, 24(3):553–560, July 2005.

[95] Frank Thomas and Ollie Johnston. *The illusion of life : Disney animation*. Disney Editions, New York, 1981.

[96] Matthew Thorne, David Burke, and Michiel van de Panne. Motion doodles: An interface for sketching character motion. *ACM Transaction on Graphics*, 23(3):424–431, August 2004.

[97] Greg Turk and Marc Levoy. http://graphics.stanford.edu/data/3Dscanrep, 1994.

[98] Romain Vergne, Pascal Barla, Xavier Granier, and Christophe Schlick. Apparent relief: A shape descriptor for stylized shading. In *NPAR '04: Proceedings of the 6th International Symposium on Non-photorealistic animation and rendering*, pages 23–29, New York, NY, USA, 2008. ACM.

[99] Romain Vergne, Romain Pacanowski, Pascal Barla, Xavier Granier, and Christophe Schlick. Light warping for enhanced surface depiction. *ACM Transaction on Graphics*, 28(3):1–8, 2009.

[100] Romain Vergne, David Vanderhaeghe, Jiazhou Chen, Pascal Barla, Xavier Granier, and Christophe Schlick. Implicit brushes for stylized line-based rendering. *Computer Graphics Forum*, 30(2):513–522, 2011.

[101] Christian Wallraven, Heinrich H. Bülthoff, Douglas W. Cunningham, Jan Fischer, and Dirk Bartz. Evaluation of real-world and computer-generated stylized facial expressions. *ACM Transaction on Applied Perception*, 4(3), November 2007.

[102] Xiaolin Wei and Jinxiang Chai. Videomocap: Modeling physically realistic human motion from monocular video sequences. *ACM Transaction on Graphics*, 29(4):42:1–42:10, July 2010.

[103] Simon West. http://archive3d.net.

[104] B. Whited, G. Noris, M. Simmons, R. Sumner, M. Gross, and J. Rossignac. Betweenit: An interactive tool for tight inbetweening. *Computer Graphics Forum*, 29(2):605–614, 2010.

[105] Georges Winkenbach and David H. Salesin. Computer-generated pen-and-ink illustration. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 91–100, New York, NY, USA, 1994. ACM.

[106] Holger Winnemöller, David Feng, Bruce Gooch, and Satoru Suzuki. Using NPR to evaluate perceptual shape cues in dynamic environments. In *NPAR '07: Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, pages 85–92, New York, NY, USA, 2007. ACM.

[107] Holger Winnemöller, Sven C. Olsen, and Bruce Gooch. Real-time video abstraction. *ACM Transaction on Graphics*, 25(3):1221–1226, July 2006.

[108] Zhongke Wu, Hock Soon Seah, Feng Tian, Xian Xiao, and Xuexiang Xie. Simulating artistic brushstrokes using disk b-spline curves. In *MAAP'04: Proceedings of the Conference of Multimedia Arts Asia Pacific*, 2004.

[109] Xuexiang Xie, Ying He, Feng Tian, Hock Soon Seah, Xianfeng Gu, and Hong Qin. An Effective Illustrative Visualization Framework Based on Photic Extremum Lines (PELs). *IEEE Transactions on Visualization and Computer Graphics*, 13:1328–1335, 2007.

[110] Songhua Xu, Yingqing Xu, Sing Bing Kang, David H. Salesin, Yunhe Pan, and Heung-Yeung Shum. Animating chinese paintings through stroke-based decomposition. *ACM Transaction on Graphics*, 25(2):239–267, 2006.

[111] Xiang Xu. https://youtu.be/A8PBdTjXQHs.

[112] Xiang Xu. https://youtu.be/KWhxoXC-SiA.

[113] Xiang Xu. https://youtu.be/OevOA4Vff3U.

[114] Xiang Xu. https://youtu.be/iFhKQ77OYJc.

[115] S. Yantis and J. Jonides. Abrupt visual onsets and selective attention: Evidence from visual search. *Journal of Experimental Psychology: Human Perception and Performance*, 10:601–621, 1984.

[116] Jingyi Yu and L. McMillan. Multiperspective projection and collineation. In *Proceedings of the 10th IEEE International Conference on Computer Vision*, volume 1 of *ICCV '05*, pages 580–587, Oct 2005.

[117] Jun Yu, Dongquan Liu, Dacheng Tao, and Hock-Soon Seah. Complex object correspondence construction in two-dimensional animation. *IEEE Transction on Image Processing*, 20(11):3257–3269, 2011.

[118] Long Zhang, Ying He, and Hock Soon Seah. Real-time computation of photic extremum lines (PELs). *The Visual Computer*, 26:399–407, 2010.

[119] Long Zhang, Ying He, Jiazhi Xia, Xuexiang Xie, and Wei Chen. Real-time shape illustration using laplacian lines. *IEEE Transactions on Visualization and Computer Graphics*, 17(7):993–1006, July 2011.

[120] Long Zhang, Qian Sun, and Ying He. Splatting lines: An efficient method for illustrating 3D surfaces and volumes. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '14, pages 135–142, New York, NY, USA, 2014.