Expressive line drawings of human faces from range images

HUANG YueZhu[1,2†], MARTIN Ralph R.[2], ROSIN Paul L.[2], MENG XiangXu[1] & YANG ChengLei[1]

[1] Department of Computer Science and Technology, Shandong University, Jinan 250101, China;

[2] School of Computer Science, Cardiff University, Cardiff, United Kingdom

We propose a novel technique to extract features from a range image and use them to produce a 3D pen-and-ink style portrait similar to a traditional artistic drawing. Unlike most previous template-based, component-based or example-based face sketching methods, which work from a frontal photograph as input, our system uses a range image as input. Our method runs in real-time for models of moderate complexity, allowing the pose and drawing style to be modified interactively. Portrait drawing in our system makes use of occluding contours and suggestive contours as the most important shape cues. However, current 3D feature line detection methods require a smooth mesh and cannot be reliably applied directly to noisy range images. We thus present an improved silhouette line detection algorithm. Feature edges related to the significant parts of a face are extracted from the range image, connected, and smoothed, allowing us to construct chains of line paths which can then be rendered as desired. We also incorporate various portrait-drawing principles to provide several simple yet effective non-photorealistic portrait renderers such as a pen-and-ink shader, a hatch shader and a sketch shader. These are able to generate various life-like impressions in different styles from a user-chosen viewpoint. To obtain satisfactory results, we refine rendered output by smoothing changes in line thickness and opacity. We are careful to provide appropriate visual cues to enhance the viewer's comprehension of the human face. Our experimental results demonstrate the robustness and effectiveness of our approach, and further suggest that our approach can be extended to other 3D geometric objects.

portrait drawing, non-photorealistic rendering, line drawing, suggestive contour, occluding contour, feature line, stylization

1 Introduction

Human faces are fascinating, and very diverse. They are so expressive of the most delicate emotional nuances that many artists have devoted their lives to portraiture. Line drawing is both concise and expressive, and is important in both art and computer graphics. Although considerable research has been done on non-photorealistic rendering (NPR) of line drawings[1−8], computer generated portraiture is still a challenging task. This is not only because of the complexity of facial appearance, but also due to the fact that human observers
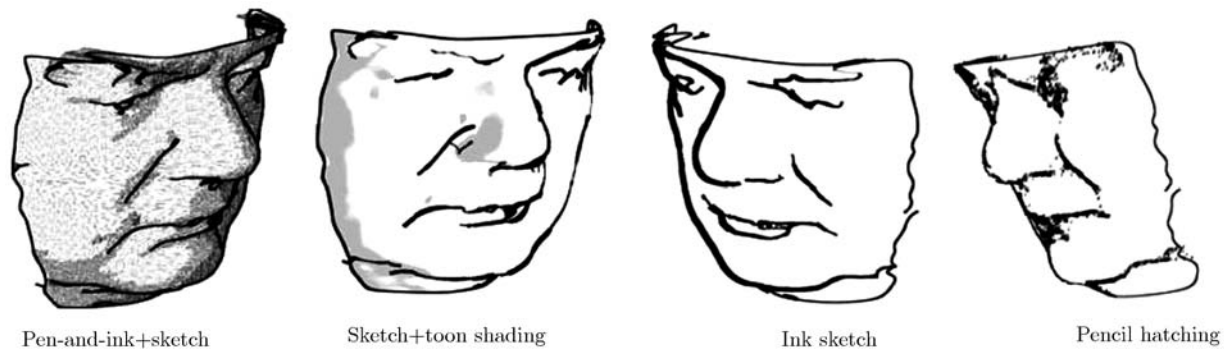
Figure 1 Portraits produced by our system in different styles and from different viewpoints.

are extremely sensitive to minor details. Making 'real' looking drawings is hard.

With the development of digital entertainment, expressive facial rendering is in increasing demand. Meanwhile, recent advances in three-dimensional sensor technologies have made possible the rapid acquisition of high quality 3D data from human faces in the form of range images or 3D meshes.

Unlike many current frontal photographic image based caricature systems[9−11] which we discuss later, we propose a novel technique to extract features from a range image and use them to produce a pen-and-ink style portrait rendering similar to a traditional artistic portrait drawing. By using a 3D model, we can alter the viewpoint to one chosen by the user and update the rendering in real time as shown in Figure 1.

A huge range of guidance is available to artists concerning portrait drawings[12−14]. Here, we summarize some suggested approaches to portrait drawing:

1) Multiple lines may be used to emphasize important facial characters. For example, in Figure 2, multiple lines are used for the sides of the nose.

2) Hatching is often used to suggest tone and shape.

3) Lines should vary in thickness, density, and opacity according to the shape of the underlying surface, and the shapes of the lines themselves. Curvature variation is most important.

4) Lips are usually drawn by hatching accompanied by an outline.

5) Eyes are usually carefully and precisely drawn in portraits.

Clearly, many other rules are also useful when drawing portraits. In this paper, we attempt to take into account some of these rules in our rendering system, in particular to ensure that we clearly capture characteristic features and the personality of the face. We render these in a way which mimics the pen-and-ink style of portrait drawing. In real range data, the face area is often incompletely acquired, particularly around the eyes (for example, due to self-occlusion and related problems during scanning), and can contain high levels of noise. Our method is designed to cope with such defects in scans, although clearly the result will be lacking

Figure 2 Portrait drawing from an art student's work. Note the highlight on the lips and multiple lines on the sides of the nose.

in detail in such areas. We also note that range scanners cannot capture details of hair, and we omit sketching of hair from consideration. The main contributions of our research are as follows:

1) We extend previous feature line detection methods in a way which performs much better with typical noisy range images (suggestive contours[15,16] are an example of the kind of feature lines intended).

2) We present a strategy for generating long smooth feature line paths which can be flexibly used as a basis for various stylization approaches.

3) We apply expressive yet flexible portrait rendering tools which are based on suggested artistic rules for portrait drawing.

In the remainder of the paper, section 2 gives an overview of the related work, while section 3 describes our extended feature line detection algorithm, and presents our chaining strategy for generating long feature line paths. Section 4 focuses on rendering. Section 5 demonstrates our results, while section 6 draws conclusions.

2 Related work

2.1 Related work

Computer graphics has typically pursued the goal of realism. However, in many cases, photorealism is not the best way to efficiently express ideas, especially if we want to concentrate on the most important aspects of a scene or object. Instead, we may prefer artistic depictions which attract the viewers' attention to significant aspects and which omit superfluous details. NPR can provide the means for visual abstraction and effective communication. A major area within NPR concerns stylistic computer-generated line drawing of 3D models. Here, we briefly review NPR work with particular relevance to NPR rendering of human faces.

Humans interpret line drawings remarkably well, being able to perceive and understand 3D structure from very sparse collections of lines. A portrait is a concise yet expressive representation of a given person. However, there have been relatively few attempts to interactively or automatically generate a stylistic sketch of a face similar to those drawn by artists. Akleman[17] described a template-based fa-

cial caricature system which simply linked face feature points using image processing methods; it produces rather lifeless sketches. Gooch[1] presented a method for creating black-and-white illustrations from photographs of human faces; furthermore, he evaluated the effectiveness of the resulting images using psychophysical studies which assessed accuracy and speed in both recognition and learning tasks. Liang et al.[9−11] reported caricaturing systems which preserve individually recognizable facial features of the person whose portrait is being produced. Chen[10] developed an example-based character drawing system which allows exaggeration. This work provided an alternative way to create caricatures from the typical approach of exaggerating the difference from the mean. Such methods do not strictly represent the original face characters, and the results often tend to beautify or exaggerate features of the human face. Luo[18] presented an image processing method for generating cartoon facial expressions from a frontal photograph. Xu[19] proposed a hierarchical composition model for facial representation and sketching allowing the production of portrait sketches at different resolutions given an input face image and a dictionary of sketches.

Our approach differs from these methods in that we start from 3D range data, allowing portraits to be drawn from different views, unlike photograph-based methods which usually draw a frontal caricature. Previous approaches also often need many tedious manual operations such as marking feature points and so on.

Other work has also considered creating caricatures from 3D data. Fujiwara[20] used a mesh model to cover a head and then generated a face representation by finding the differences between the input face and a mean face. Boyer[21] produced a 3D face mesh by comparing a face to a canonical model, but simply rendered feature lines without any stylization. Often, such research focuses on how to exaggerate the face shape, and then renders it in simple caricature style, taking little account of the methods of traditional portrait drawing.

2.2 Feature lines

As Cole[22] points out, feature lines, including sil-

houettes, ridges, valleys, suggestive contours, apparent ridges, and others, play an important role in shape perception. He presents the results of a study in which artists made line drawings intended to convey specific 3D shapes. Lines drawn by artists in this study often overlapped one another, particularly along the occluding contours of the object. A natural question to ask is: "How well can current line drawing algorithms describe human artists' lines?" This study revealed that feature line types which can be automatically extracted, such as occluding contours, suggestive contours, and apparent ridges, account for about 80% of artistic lines. Inspired by the effectiveness and aesthetic appeal of artistic line drawings, many image-based and object-based algorithms have been proposed for automatically generating such line drawings. Decarlo[23] provides a useful bibliography concerning the production of line drawings from 3D data, and the perception of line drawings; here we simply review the most recent developments in line drawing.

Based on the insight that a line drawing can be understood as an abstraction of a shaded image, Decarlo[24] suggests use of two kinds of highlight lines where diffuse shading would yield thin bright areas using a single point light located at the camera. Lee[25] extends Decarlo's work to cases involving diffuse and specular highlights under arbitrary illumination, and gives a' GPU-based algorithm for rendering a 3D model as a line drawing. However, drawing lighting-dependent lines can result in certain important features not being depicted under particular lighting setups.

Decarlo[15,16] describes suggestive contours as a new type of feature line that can be combined with silhouettes to produce effective line drawings of smooth shapes. Suggestive contours are curves along which the radial curvature is zero: the view vector v is projected onto the local surface tangent plane at the point p to obtain w; the radial plane spans the surface normal n and w, slicing the surface along the radial curve, whose curvature gives the radial curvature (see Figure 3). Judd[26] defines apparent ridges as the loci of points that maximize view-dependent curvature, and defines

view-dependent curvature as the variation of the surface normal with respect to a projection plane. Unfortunately, suggestive contours and apparent ridges can only be reliably computed on smooth meshes due to their sensitivity to noise: suggestive contours are based on second derivatives of surface information while apparent ridges are based on third derivatives. Directly computing suggestive contours or apparent ridges from limited-resolution range data meshes typically produces results which are too noisy. We instead propose the use of a novel feature-line detection algorithm which is efficient yet robust. It is based on reliable suggestive contours and follows the object space algorithm described by Decarlo[15,16]. It takes a set of visible silhouette edges produced from the range image, and efficiently chains them to form long smooth line paths, to which stylization algorithms may be effectively applied. Our algorithm gains efficiency and robustness over existing methods by directly exploiting the analytic connectivity information provided by the range image and minimizing the impact of noise during rendering. In addition, we filter artifacts in connected edges during the process to improve the visual quality of strokes after stylization.

3 Computing extended feature lines

Our goal is to turn a range image of a face into a 3D portrait with a similar appearance to an artist's work, allowing real-time interaction to choose the point of view, and rendering style. We have broken down the problem into four separate parts. Firstly, since we only focus on the face, we trim away irrelevant parts of the range image. Secondly, we extract feature lines including occluding contours and extended suggestive contours. Thirdly, we chain all feature lines into long curves and smooth them. Finally, we render these curves with a chosen stylistic shading method. This section describes our extended suggestive contour algorithm in detail.

3.1 Extended suggestive contours algorithm

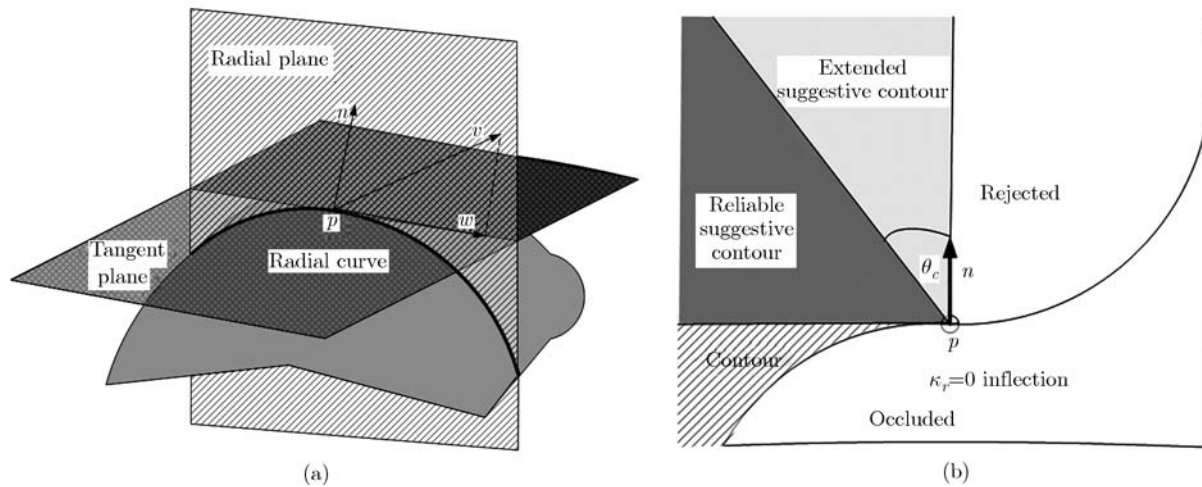We regard occluding contours and suggestive con-

Figure 3 (a) Radial curvature; (b) a radial plane section of the surface around point p. A reliable suggestive contour appears at the inflection point p on the radial curve. If the camera is located in the dark grey region, the surface will have a reliable suggestive contour at p and will extend to the light grey region.

tours as the major shape cues to be used during portrait drawing. Decarlo[15,16] describe an object space algorithm for finding suggestive contours. If we directly apply it to range data, it typically produces too many noisy lines. Although thresholding can be used to eliminate some of the noisy or too-short lines, the results still look quite unpleasant. If we choose a larger threshold, too few lines are extracted; if we choose a smaller threshold, too many broken, unordered, noisy lines are extracted. One might consider the use of mesh smoothing[27] to eliminate the noise, but if sufficient mesh smoothing is used to eliminate the noise, it typically also eliminates certain geometric features of the face which are essential for portrait drawing. Instead, we have designed a direct feature line extraction method which is robust in the presence of noise, inspired by the idea of hysteresis thresholding[28]. The assumption is that if a suggestive contour passes through a certain triangle, it is very likely to continue to pass through adjacent triangles. Therefore, if strong evidence for a suggestive contour exists in one place, it may be extended to neighbouring triangles, even if evidence for the contour there is weaker. Our extended suggestive contour algorithm first finds an initial reliable suggestive contour segment, and then extends it into selected surrounding areas.

We first briefly recall the definition of sugges-

tive contour[15]. Suggestive contours are the set of points on a surface at which its radial curvature k_r is 0, and the directional derivative of k_r in the direction of w is positive (w is the projection of the view vector on the local surface tangent plane):

$$k_r(p) = 0, \tag{1}$$
$$D_w k_r(p) > 0. \tag{2}$$

Here, the directional derivative $D_w k_r$ is defined as the derivative of $k_r(p)$ with respect to distance, measured along w. These curvatures and their directional derivatives may be computed using a curvature tensor fitting scheme as described in Decarlo[15,16]. In the magenta region (see Figure 3(b)), the suggestive contour is unstable because it changes drastically with small changes in the viewpoint (because w changes direction quickly). Further instability arises due to errors in curvature estimation. To find reliable suggestive contours, we use the method following Decarlo[15], which further requires

$$0 < \theta < \cos^{-1}\left(\frac{\boldsymbol{n}(p) \cdot \boldsymbol{v}(p)}{\| \boldsymbol{v}(p) \|}\right), \tag{3}$$

$$t_d < D_{\boldsymbol{w}} k_r / \| \boldsymbol{w} \|, \tag{4}$$

where θ_c, t_d are thresholds. This gives suggestive contour points which are stable in the presence of noise. We choose these thresholds so that they work consistently for range image meshes showing

a range of objects and viewpoints. They should be set large enough to ensure the stability of the contours found, and to minimize the impact of errors and noise. As can be seen from Figure 3(b) and eqs. (3) and (4), the larger θ_c and t_d, the fewer reliable suggestive contours will be extracted. In all the examples produced by our system, we set θ_c and t_d as suggested by Decarlo.

Having found initial reliable suggestive contour segments, we extend them to a k-ring neighborhood on the mesh by detecting suggestive contours without using the thresholds θ_c and t_d: we extend the initial reliable suggestive contours one ring at a time to the 3-ring neighborhood area in Figure 4. The size k of the neighborhood can be specified by the user. Note that we only extend suggestive contours where they would be visible. This provides the following advantages:

1) All feature lines found are visible, avoiding time consuming visibility tests in subsequent chaining and stylization processing.

2) Lines tend to be connected to form long paths, which is important for coherent rendering.

Our algorithm extends suggestive contours across the mesh until they become occluded as shown in Figure 3(b). But note that the algorithm

is designed to extend any initial reliable suggestive contours. Usually, various unwanted noisy lines exist together with these initial suggestive contours, and our algorithm will extend these too. To reduce the visible effects of such cases, we make the algorithm terminate in any case after reaching the specified final ring of the neighborhood. When extending the suggestive contours, we ignore the thresholds θ_c and t_d, and simply add suggestive contours which satisfy eqs. (1) and (2). In fact, we could require these suggestive contours to in addition satisfy thresholds which are less stringent than those used to find the initial reliable suggestive contour segments, but it is not easy to tune such thresholds. Thus, for simplicity we omit their use: we find that we can still obtain good results without using relaxed thresholds. The process is demonstrated in Figure 4, where we assume we find an initial reliable suggestive contour segment represented by AB, which then results in an extended suggestive contour $JIHGABCDEF$ by considering the 3-ring neighborhood of triangle $P_0P_1P_2$.

3.2 Line chaining

The classical path-and-style metaphor for non-photorealistic rendering, which draws lines of vary-
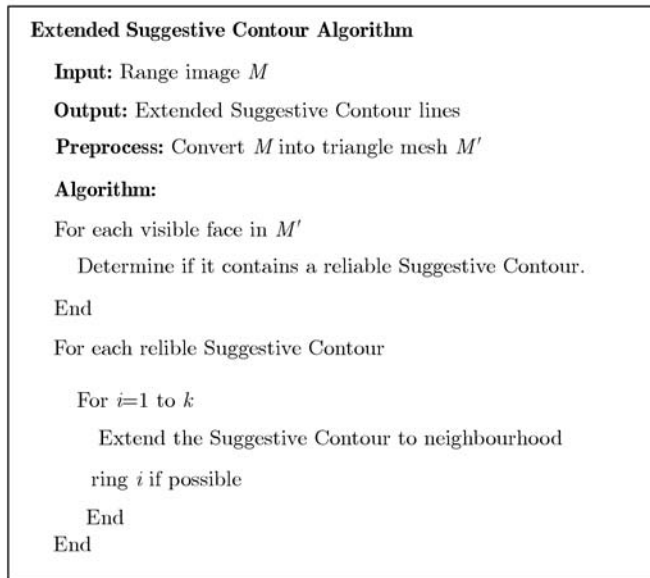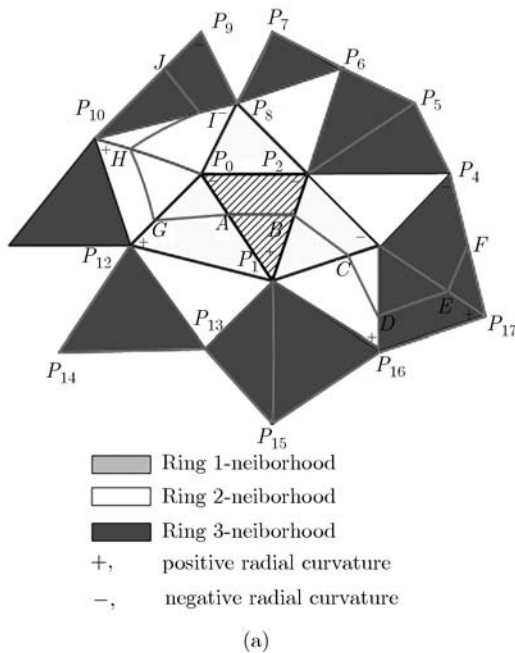

(a)

(b)

Figure 4 (a) An extended suggestive contour starts from AB and is extended as $JINGABCDEF$. (b) Algorithm for extended suggestive contours.

ing styles based on underlying paths, was first used by Finkelstein[29]. It can be applied to domains ranging from vector graphics to pixel graphics. Our approach for finding suggestive contours generates small discrete pieces of feature line, and these must be chained to give longer smooth connected paths as a basis for drawing strokes using this metaphor. Although much literature discusses stylistic rendering[3−8], few papers focus on the chaining strategy.

Sousa[30] mentions the silhouette edge chaining problem, but does not explicitly state any chaining strategy. Image-based contour generation methods like active contours[31], and intelligent scissors[32], focus on generating object boundaries to segment an image represented as a pixel array, while we wish to join line segments on a mesh. To address this problem, our approach is based on the concept that a segment should only be added to the end of an existing feature line if it agrees in direction, and it is near enough. We construct a directed graph comprising all feature line segments, and chain those line paths which satisfy eqs. (5) and (6):

$$D(\boldsymbol{e}_i, \boldsymbol{e}_j) \leqslant \xi, \tag{5}$$

$$\max\{w_{\mathrm{dis}}(\boldsymbol{e}_i, \boldsymbol{e}_j) + w_{\mathrm{dir}}(\overrightarrow{\boldsymbol{e}_i}, \overrightarrow{\boldsymbol{e}_j})\} \geqslant c, \quad i \neq j. \tag{6}$$

Here, $\boldsymbol{e}$ denotes a feature line path composed of line segment $\boldsymbol{s}_i$, and $\overrightarrow{\boldsymbol{e}}$ denotes its weighted average direction taking into account all segment directions contained in this line path using

$$\overrightarrow{\boldsymbol{e}} = \left(1 - \sum_{\boldsymbol{s}_i \in e}^{n} w_i\right) \overrightarrow{\boldsymbol{s}_0} + \sum_{\boldsymbol{s}_i \in e}^{n} w_i \overrightarrow{\boldsymbol{s}_i}, \tag{7}$$

where $w_i = 2^{-(i+1)}, i = 1, \ldots, n$. $D(\boldsymbol{e}_i, \boldsymbol{e}_j)$ represents the Euclidian distance between the nearest end points of $\boldsymbol{e}_i, \boldsymbol{e}_j$. $w_{\mathrm{dis}}(\boldsymbol{e}_i, \boldsymbol{e}_j)$ is a distance weight function defined by

$$w_{\mathrm{dis}}(\boldsymbol{e}_i, \boldsymbol{e}_j) = \begin{cases} a & \text{if } D(\boldsymbol{e}_i, \boldsymbol{e}_j) < \xi_1, \\ \exp(1 - D(\boldsymbol{e}_i, \boldsymbol{e}_j)/\xi_1) & \text{otherwise,} \end{cases} \tag{8}$$

and $w_{\mathrm{dir}}(\overrightarrow{\boldsymbol{e}_i}, \overrightarrow{\boldsymbol{e}_j})$ gives directional weighting defined

by

$$w_{\mathrm{dir}}(\overrightarrow{\boldsymbol{e}_i}, \overrightarrow{\boldsymbol{e}_j}) =$$
$$\begin{cases} b & \text{if } \overrightarrow{\boldsymbol{e}_i} \cdot \overrightarrow{\boldsymbol{e}_j} > \theta, \\ \exp((1 + \overrightarrow{\boldsymbol{e}_i} \cdot \overrightarrow{\boldsymbol{e}_j})/2) & \text{otherwise.} \end{cases} \tag{9}$$

Note that $\xi_1, \xi, \theta, a, b, c$ are constants which are specified by the user (typically by regarding the model size as 1, default values are $\xi = 0.08, \xi_1 = 0.25\xi, \theta = 0.9, a = 5, b = 4, c = 4.5$). $w_{\mathrm{dir}}(\overrightarrow{\boldsymbol{e}_i}, \overrightarrow{\boldsymbol{e}_j})$ and $w_{\mathrm{dis}}(\boldsymbol{e}_i, \boldsymbol{e}_j)$ are given additional importance as shown if $\overrightarrow{\boldsymbol{e}_i} \cdot \overrightarrow{\boldsymbol{e}_j}$ or distance weight $D(\boldsymbol{e}_i, \boldsymbol{e}_j)$ lies within the specified thresholds ξ_1 and θ respectively. From eq. (5), we first limit the chaining of line paths using the threshold distance ξ; then we further constrain the chaining using a tradeoff between direction and distance, as expressed by eq. (6).

Figure 5(a) explains the overall strategy of our line chaining algorithm. The input comprises the line segments S resulting from the extending suggestive contour algorithm. These line segments are chained only if the sum of their distance weight evaluated by eq. (8) and direction weight evaluated by eq. (9) is greater than a user-specified value c. Find_And_Add_Predecessor(P_i, G) returns the first found predecessor P_j of P_i for which the distance between P_i and P_j is 0, and adds the edge $P_j P_i$ to the graph. Find_And_Add_Successor (P_i, G) similarly returns the first successor P_j of P_i for which the distance between P_i and P_j is 0, and adds the edge $P_i P_j$ to the graph. An example is shown in Figure 5. The initi-al direction of input line segments is assigned at random, e.g. $P_0 P_1$ or $P_1 P_0$ (see Figure 5(b)). After Build_All_Line_Paths (G, L) the line paths are $P_9 P_8 P_0 P_1 P_{17} P_{16} P_{12} P_{13} P_4 P_5$ $P_6 P_7 P_2 P_3$ and $P_{14} P_{15} P_{11} P_{10} P_4 P_5 P_6 P_7 P_2 P_3$ (see Figure 5(c)). Since they have common points like P_4, P_5, they are then merged (see Figure 5(d)). The algorithm ensures that we chain all compatible line segments together. In the worst case the algorithm takes time $O(|E|^2)$ where $|E|$ is the number of edges in G.

3.3 Refinement

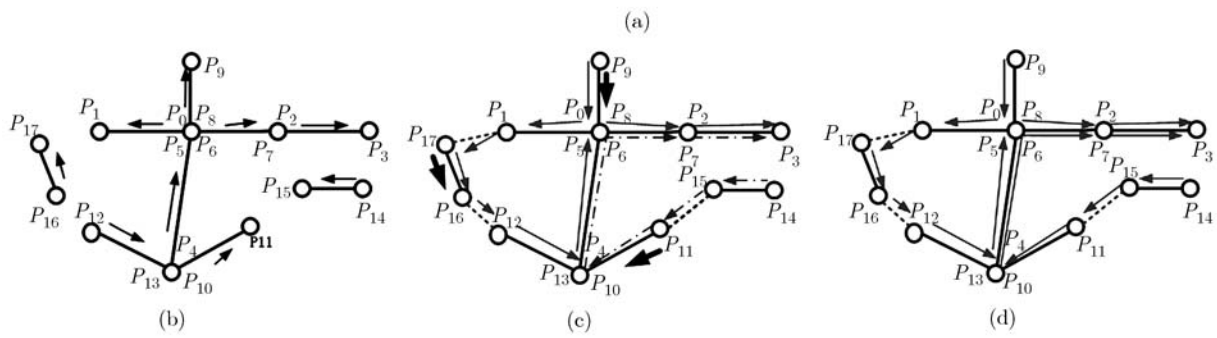If we directly render using the feature line paths

Figure 5 (a) Line chaining algorithm; (b) the initial G; (c) G after Build_All_line_paths (G, L); (d) G after merge. Arrows denotes edge directions; thick arrow denotes edge direction changes.

generated above, the resulting drawings still look unpleasant because of remaining noise and artifacts. We therefore further refine the generated paths.

3.3.1　Point clustering. When constructing the graph, several nodes may actually represent a single point. See Figure 5(b), where P_0, P_5, P_6, P_8 are all the same points. We need to cluster them as one point. At the same time, and more importantly, if the distances between any two points on feature lines are too small (typically 0.5% of the model size), we cluster them as a single point, which improves the connectivity and shape of feature lines.

3.3.2　Line path smoothing. Walking along each line path in turn, we use eq. (10) to iteratively smooth the path by averaging adjacent points. We first smooth each line path in a forward direction,

and then in the reverse direction.

$$P_i = \frac{P_{i-1} + P_i + P_{i+1}}{3},$$

$$P_i \text{ on linepath}, \ i = 1, \ldots, n-1. \quad (10)$$

3.3.3　Line length filter. Often, it is hard to decide whether a short piece of line is due to noise or represents a real feature. We simply regard line paths which are too short as noise, and discard them, as they are not important for rendering the main features of a face, and more likely to be a distraction. We typically set the length filter to be 5% of the model size.

4　Stylistic rendering

Portraiture books[12−14] show that even when only using pencil, artists are able to create great visual

impact. The most frequently used line styles are hatching lines and over-sketching. We attempt to simulate such effects using a pen-and-ink shader, a hatching shader and an over-sketching shader. Inspired by previous research on stylistic stroke rendering[3−8,30], we apply stroke shaders to the line paths we have already computed.

4.1 Portrait rendering

4.1.1 Pen-and-ink shader. Pen-and-ink style is commonly used for portrait drawing. Winkenbach[8] discusses use of pen-and-ink illustration for architecture. In portrait rendering, we find that it is almost impossible to give the tone of each facial part precisely; thus, instead, we control the overall tone by using a series of textures. We follow the method describe in Lake[33] to achieve a real-time pencil portrait sketching effect. We use $v \cdot n$ (v denotes the light direction, n is the normal of the triangle face) to determine the texturing: regions receiving less light are given a higher density of pencil strokes. This coincides with the rules used by an artist when applying pen-and-ink strokes. Typical results are shown in Figures 1, 7(k), and 8.

4.1.2 Hatching shader. Hatching is a widespread NPR style that uses patterns and arrangements of line strokes to represent tones and shapes; usually simple line strokes are used in groups of approximately parallel strokes. The basic principle for hatching lines is that they should be placed over feature lines to convey geometric shape and lighting. The key challenge in incorporating these ideas into our portrait hatching shader is how to arrange hatching strokes. Artists tend to place the hatching strokes around feature lines. Our hatching shader accordingly places hatching strokes along line paths to enhance shape and tone perception. An example can be seen in Figure 8, where we render line strokes following the lip shape, and use hatch lines to enhance shape cues while leaving a certain are white to provide the highlight. This effect is similar to line drawings executed by hand: see the lip in Figure 2.

4.1.3 Over-sketching shader. Sketching involves being free and bold while not worrying about making mistakes. Over-sketching is widely used in por-

trait drawing, and we achieve this effect by drawing multiple strokes with random small offsets in the normal direction. This effect is shown in Figure 7(h). The offset stroke points are given by

$$p_i = p_i + \text{rand}()n_i, \quad p_i \text{ in stroke}, \quad i = 0, \ldots, n, \tag{11}$$

where rand() returns a random value between 0 and 1, and n_i is the normal to point p_i.

4.1.4 Combination of shaders. There is no single way to generate all desirable portrait styles, and the usual practice is to combine multiple shaders. Our system flexibly allows combination of shaders as chosen by the user. We believe the more flexible the combination can be, the more potentially creative the final results.

4.2 Stylistic control

Within each shader, we still can create different visual effects by varying the stroke type, thickness, and opacity.

4.2.1 Thickness control. Many NPR systems employ strokes to convey shape. Often more attention is paid to where to draw lines, rather than how to determine stroke thickness. However, it is generally believed that curvature driven thickness and viewing distance driven thickness give good results. Goodwin[34] gives an approach for determining stroke thickness in computer-generated illustrations, but the method is intended for smooth surfaces without creases or boundaries, and is not suitable for portrait drawing. We extend their work for portrait drawing, modifying it to provide smooth variations in thickness by considering the context of the current stroke. We determine the stroke thickness by not only considering its own properties but those of the preceding stroke. This can be simply achieved by using a weighted average (see eq. (8)) of the thickness of the current stroke and previous strokes:

$$t = \left(1 - \sum_{i=1}^{n} w_i\right) t + \sum_{i=1}^{n} w_i t_i, \tag{12}$$

where t denotes the stroke thickness, and t_i denotes the ith previous stroke thickness. A weight $w_i = 2^{-(i+1)}, i = 1, \ldots, n$ is used to progressively decrease the effect of previous strokes. Results can be seen in Figure 7(a) and (b).
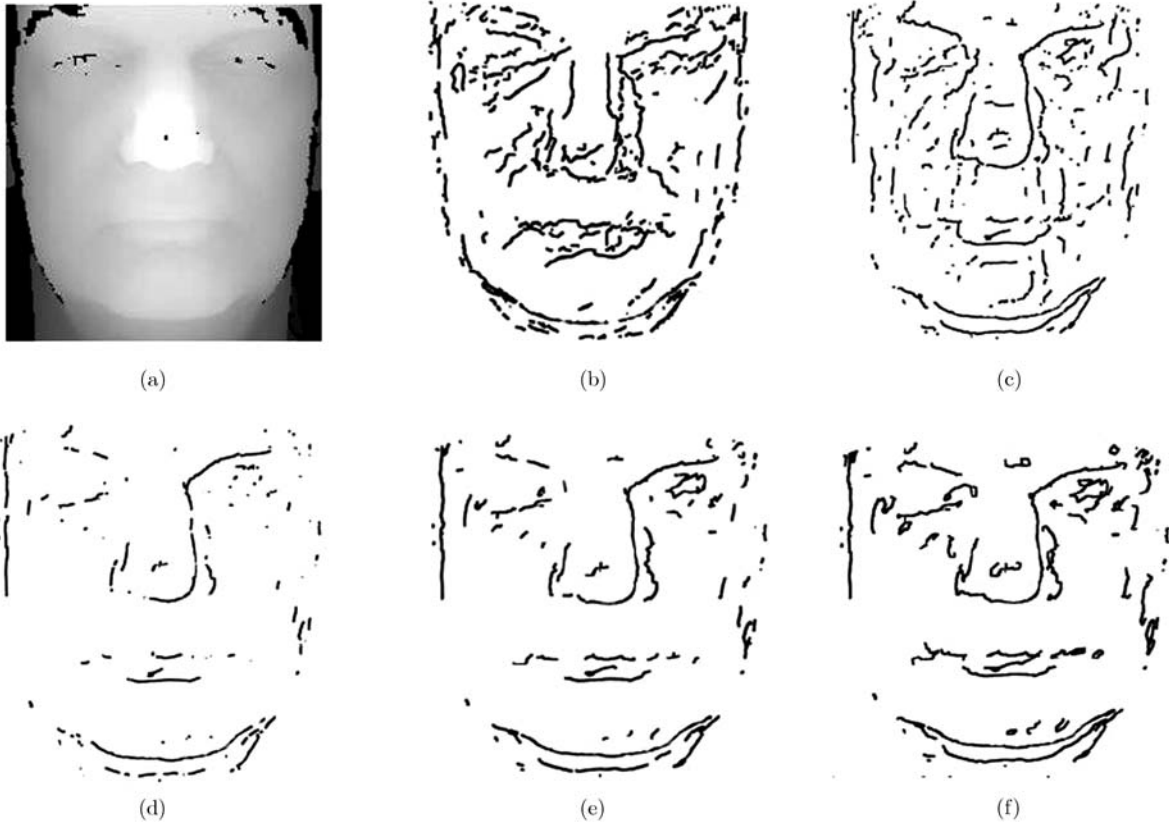
Figure 6 Feature lines produced by our algorithm and other methods. At each point $\theta_c = n(p) \cdot v(p)$. (a) Original range image; (b) apparent ridges; (c) suggestive contours t_d=0.05; (d) suggestive contours t_d=0.1; (e) extended suggestive contours t_d=0.1, 1-ring; (f) extended suggestive contours t_d=0.1, 3-ring.

4.2.2 Stroke type variation. Artists use pencils with different shapes of tip (sharp, or with an edge rubbed flat, etc.) to produce different line types. In our system, we provide the user with different stroke types to flexibly produce different visual effects. We can see different stroke types applied in Figure 7(e) and (f).

4.2.3 Opacity control. To produce smoother lines and prevent them from appearing abruptly as viewpoint changes, we add opacity control to each stroke, again taking into consideration opacity of the preceding stroke. We use $D_w k_r / \parallel w \parallel$ as the stroke's initial opacity, and then average the stroke opacity using

$$ o = \left(1 - \sum_{i=1}^{n} w_i\right) o + \sum_{i=1}^{n} w_i o_i, \qquad (13) $$

where o denotes the stroke thickness, o_i denotes the ith previous stroke thickness, and $w_i = 2^{-(i+1)}, i = 1, \ldots, n$ as before.

5 Results and discussion

All experiments in this paper were performed on a laptop with 1.66 GHz Intel processor and 1 GB RAM, without code optimization. The range images we used come from the Biometrics Database of the University of Notre Dame[35]. As shown in Table 1, the feature line extraction process is quite fast, and the time for line chaining is approximately quadratic in the number of feature lines, as expected; the bottleneck of our system lies in stylistic rendering stage. The number of feature lines increases with the model size. When the number of feature lines reaches around 2k, the total time becomes more than 1.5 s. However, modern hardware with a newer graphics card, and optimised code should provide much faster rendering. Our system can currently generate images for moderate resolution models (under 10k triangles) in less than a second and can be thus used in real time applications.
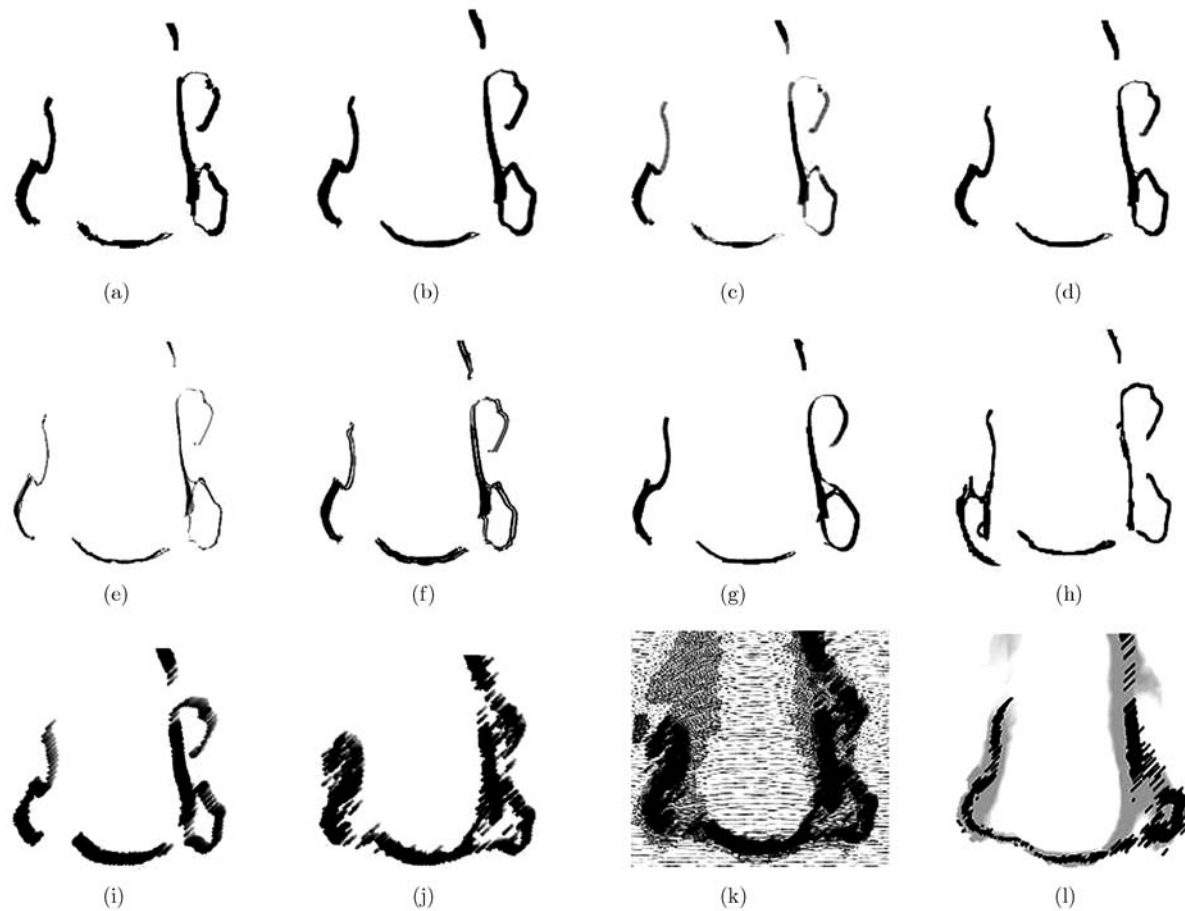
Figure 7 Noses drawn with different stylistic control. (a) Thickness smoothing off; (b) thickness smoothing on; (c) opacity variation off; (d) opacity variation on; (e) sketch with light strokes; (f) sketch with double lines; (g) smoother sketch; (h) over-sketching; (i) hatching strokes; (j) diffuse hatching strokes; (k) diffuse hatching+hatching texture; (l) hatch-shaded.
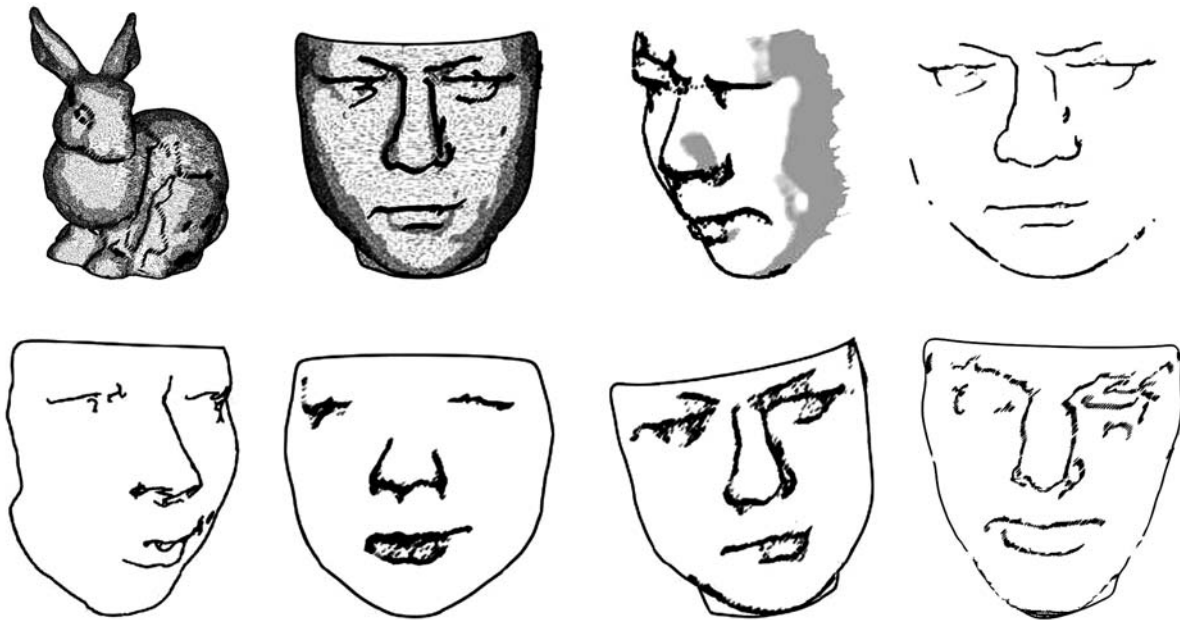

Figure 8 The Stanford bunny, and further portraits drawn by our system.

Table 1 Time taken to render various face models[a)]

Model	FN	LN	ET	CT	TT
04267d141	7158	1050	0.012 s	0.136 s	0.367 s
04311d182	9170	1381	0.015 s	0.258 s	0.681 s
04366d74	7957	1037	0.012 s	0.148 s	0.459 s
02463d452-1	9028	1339	0.014 s	0.240 s	0.686 s
02463d452-2	13325	1973	0.023 s	0.516 s	1.521 s
02463d452-3	18513	2508	0.028 s	0.404 s	4.295 s

a) FN: Number of triangles; FL: number of feature lines; ET: time for extracting feature lines; CT: time for line chaining and smoothing; TT: total time including ET, CT and rendering.

Here we compare the output resulting when finding apparent ridges, suggestive contours and our extended suggestive contours. As mentioned before, if we directly apply Decarlo's method[15,16] to range data, it typically produces too many noisy lines (see for example Figure 6(c)). Although thresholding can be used to eliminate some of the noisy or too-short lines, the result still looks quite unpleasant (see Figure 6(d)). Our extended suggestive contour algorithm is simple, but works quite well for noisy range images. Figure 6 shows that our method produces better feature lines although some noise still exists. However, we note that while producing better feature lines, it also enhances some unwanted lines caused by noise, as can be seen for example on the nose tip in Figure 6(d). Fortunately, many such unwanted lines are discarded by the subsequent chaining and smoothing processes, as shown in Figures 7 and 8.

We note that our approach for extending suggestive contours can in principle be applied to other kinds of feature lines such as apparent ridges[26]. However, on attempting to extend apparent ridges on medium-resolution range images of faces, we encounter the problem that locating initial apparent ridges cannot be done reliably, as their computa-

tion is too sensitive to noise. This leads to rendering results which are worse than those provided by extended suggestive contours (see Figure 6).

We further note that our approach of extending suggestive contours is useful not only for facial range images, but also works well for more general meshes, as can be seen in the top left picture in Figure 8.

In our system, the user can control the stroke type, thickness, opacity and smoothness using real time interaction. Figure 7 demonstrates drawing a nose with different stylistic control, giving different visual effects. We demonstrate more expressive portrait drawings in Figures 1 and 8.

6 Conclusions

We have described an effective yet robust method for drawing portraits from noisy range images in a way which suggests traditional artistic portrait drawing. Our approach first finds pieces of suggestive contours, then chains them by carefully considering whether they are compatible. We allow flexible control of style by varying stroke type, thickness, smoothness, and opacity, producing real-time portraits which imitate artistic style.

1 Gooch B, Reinhard E, Gooch A. Human facial illustrations: Creation and psychophysical evaluation. ACM Trans Graph, 2004, 23(1): 27–44

2 Hertzmann A, Zorin D. Illustrating smooth surfaces. In: Proc of SIGGRAPH, 2000, 517–526

3 Isenberg T, Halper N, Strothotte T. Stylizing silhouettes at interactive rates: From silhouette edges to silhouette strokes.

In: Proc of EuroGraphics, 2002, 249–258

4 Kalnins R, Markosian L, Meier B, et al. WYSIWYG NPR: Drawing strokes directly on 3d models. In: Proc of SIGGRAPH, 2002. 755–762

5 Kalnins R, Davidson P, Markosian L, et al. Coherent stylized silhouettes. ACM Trans Graph, 2003, 22(3): 856–861

6 Northrup J, Markosian L. Artistic silhouettes: a hybrid ap-

proach. In: Proc of NPAR, 2000. 31–38

7 Sousa C M, Buchanan W J. Computer-generated graphite pencil rendering of 3d polygonal models. In: Proc of EuroGraphics, 1999. 195–208

8 Winkenbach G, Salesin H D. Computer-generated pen-and-ink illustration. In: Proc of SIGGRAPH, 1994. 91–100

9 Liang L, Chen H, Xu Y Q, et al. Example-based caricature generation with exaggeration. In: Proc of 10th Pacific Conf on Computer Graphic and Application, 2002. 386–393

10 Chen H, Liu Z Q, Rose C, et al. Example-based composite sketching of human portraits. In: Proc of NPAR, 2004. 95–153

11 Tresset P, Leymarie F F. Generative Portrait Sketching. In: 11th Inter Conf on Virtual Systems and Multi-Media, 2005. 739–748

12 Douglas G R. Drawing Portraits. New York: Watson-Guptill Publications, 1983

13 Freeman J. Portrait Drawing. Marlborough: Crowood Press, 2006

14 Blake W, Lawn J. Portrait Drawing: A Step-by-Step Art Instruction Book. New York: Watson-Guptill Publications, 2006

15 Decarlo D, Finkelstein A, Rusinkiewicz S, et al. Suggestive contours for conveying shape. ACM Trans Graph, 2003, 22(3): 848–855

16 Decarlo D, Finkelstein A, Rusinkiewicz S. Interactive rendering of suggestive contours with temporal coherence. In: Proc of NPAR, 2004. 15–24

17 Akleman E. Making caricature with morphing. In: Proc of SIGGRAPH, 1997. 145

18 Luo Y, Gavrilova M L, Sousa M C. NPAR by example: line drawing facial animation from photographs. In: IEEE Inter Conf on Computer Graphics, Imaging and Visualization, 2006. 514–521

19 Xu Z J, Chen H, Zhu S C, et al. A hierarchical compositional model for face representation and sketching. IEEE Trans Patt Anal Mach Intell, 2008, 30(6): 955–969

20 Fujiwara T, Koshimizu H, Fujimura K, et al. 3D modeling system of human face and full 3D facial caricaturing. In: Proc of 3D Digital Imaging and Modeling, 2001. 39–51

21 Boyer V. An artistic portrait caricature model. In: Proc of Inter Symp of Visual Computing, 2005. 595–600

22 Cole F, Golovinskiy A, Limpaecher A, et al. Where do people draw lines? ACM Trans Graph, 2008, 22(3): 44–55

23 Decarlo D, Finkelstein A. Line drawings from 3D models. In: Proc of SIGGRAPH, Course Note, 2005

24 Decarlo D, Rusinkiewicz S. Highlight lines for conveying shape. In: Proc of NPAR, 2007. 63–70

25 Lee Y J, Markosian L, Lee S, et al. Line drawings via abstracted shading. ACM Trans Graph, 2007, 26(3): 18

26 Judd T, Durand F, Adelson E. Apparent ridges for line drawing. ACM Trans Graph, 2007, 26(3): 19

27 Sun X F, Rosin P L, Martin R R, et al. Fast and effective feature-preserving mesh denoising. IEEE Trans Visual Comput Graph, 2007, 13(5): 925–938

28 Canny J. A computational approach to edge detection. IEEE Trans Patt Anal Mach Intell, 1986, 8(6): 679–698

29 Finkelstein A, Salesin D H. Multiresolution curves. In: Proc of SIGGRAPH, 1994, 261–268

30 Sousa M C, Prusinkiewicz P. A few good lines: suggestive drawing of 3D models. Comput Graph Forum, 2003, 22(3): 381–390

31 Kass M, Witkin A, Terzopoulos D. Snakes: active contour models. Inter J Comput Vision, 1988, 1(4): 321–331

32 Mortensen E, Barrett W. Intelligent scissors for image composition. In: Proc of SIGGRAPH, 1995. 191–198

33 Lake A, Marshall C, Harris M, et al. Stylized rendering techniques for scalable real-time 3d animation. In: Proc of NPAR, 2000. 13–20

34 Goodwin T, Vollick I, Hertzmann A. Isophote distance: A shading approach to artistic stroke thickness. In: Proc of NPAR, 2007. 53–62

35 Flynn J P, Bowyer W K, Phillips J P. Assessment of time dependency in face recognition: An initial study. In: Audio and Video-Based Biometric Person Authentication, 2003. 44–51