Noname manuscript No.
(will be inserted by the editor)

Chinese Painting Strokes via Shape-Cues Shading

the date of receipt and acceptance should be inserted later

Abstract In this paper, we present a NPR technique for simulating the Chinese ink strokes which is considered as a special type of line-drawings. The method automatically transforms 3D model to ink-style image via continuous Shape-cues shading. Geometrically we define our shape-cues as the composition of view-dependent and view-independent shape descriptors to convey ink-style strokes. In addition, depth information is used to compensate the final expression of strokes. View-dependent lines give the skeletal-lines, and view-independent (principle curvatures) as tex-coordinates extend lines to strokes by picking intensity in stylization texture. We show that the method is easy-to-implement and more appropriate to produce realistic ink strokes in real-time.

Keywords non-photorealistic rendering · artistic media simulation · line drawings · shape depiction

1 Introduction

Non-photorealistic Rendering is an important field of modern rendering techniques, the goal of NPR is not only rendering objects more comprehensible but also creating artistic works. Recently, simulating specific art media becomes a dramatic role in the appearance of NPR. Our method is to generate Chinese painting style images like hand-drawings directly from 3D models (e.g Fig. 2). The most important difference between Western painting and Eastern painting is that Eastern paintings are composed primarily of lines. Chinese ink painting has acquired different characteristics after many generations of development. In general, two methods of strokes(lines) exist: Gong Bi(Realistic strokes), that is, drawing details with fine strokes and rich color, and Yi Bi (Leisurely strokes), drawing with rough strokes and light color. Because of water-absorbing cotton paper and various brush lines, the rough and simple strokes are preferable than fine and complicated ones. According to such characteristics, we define Chinese painting strokes as a kind of line-drawing composed of more rough and thick lines with gradient intensity extension.

The line-illustration is proved to be the effective method in shape representation because of the straightforward understanding of images [11]. Suppose you have a 3D model of some object and you want to illustrate its shape into 2D images. Regardless of the media style, we should define a shape descriptor in order to draw shape-represented lines. *The key question is: where do we locate the lines?* Various rules have been proposed. Some previous approaches focus on a single body of silhouette extraction algorithm which describes the shape according to the discontinuities of shape features (for instance, depth, curvatures, orientations, object IDs, etc [13][17]). However, another alternative is to utilize continuous shape features and convey the shape through shading. We try to render such continuous features into each pixel of an image plane as shape-cues. Therefore, the shape of a 3D object is implied in the 2D image from which will produce the final picture with stylized processes. In other words: our strategy is to utilize the implicit cues shading to decide where to draw lines when you want to convey shape effectively only by providing some line-drawing.

The remaining question is: what kind of continuous shape features do we use to depict shape through shading? The preferable way is to make the line-drawing independent of the rendering parameters like BRDF and illumination and only reserve surface normal as the original shape feature so that the final locations are only view dependent but stable across other rendering conditions. Many previous works describe shapes by utilizing normal and its high-order derivative (see Sect. 2). Particularly, Chinese painting strokes as we mentioned is more rough and thick, and previous meth-

Address(es) of author(s) should be given

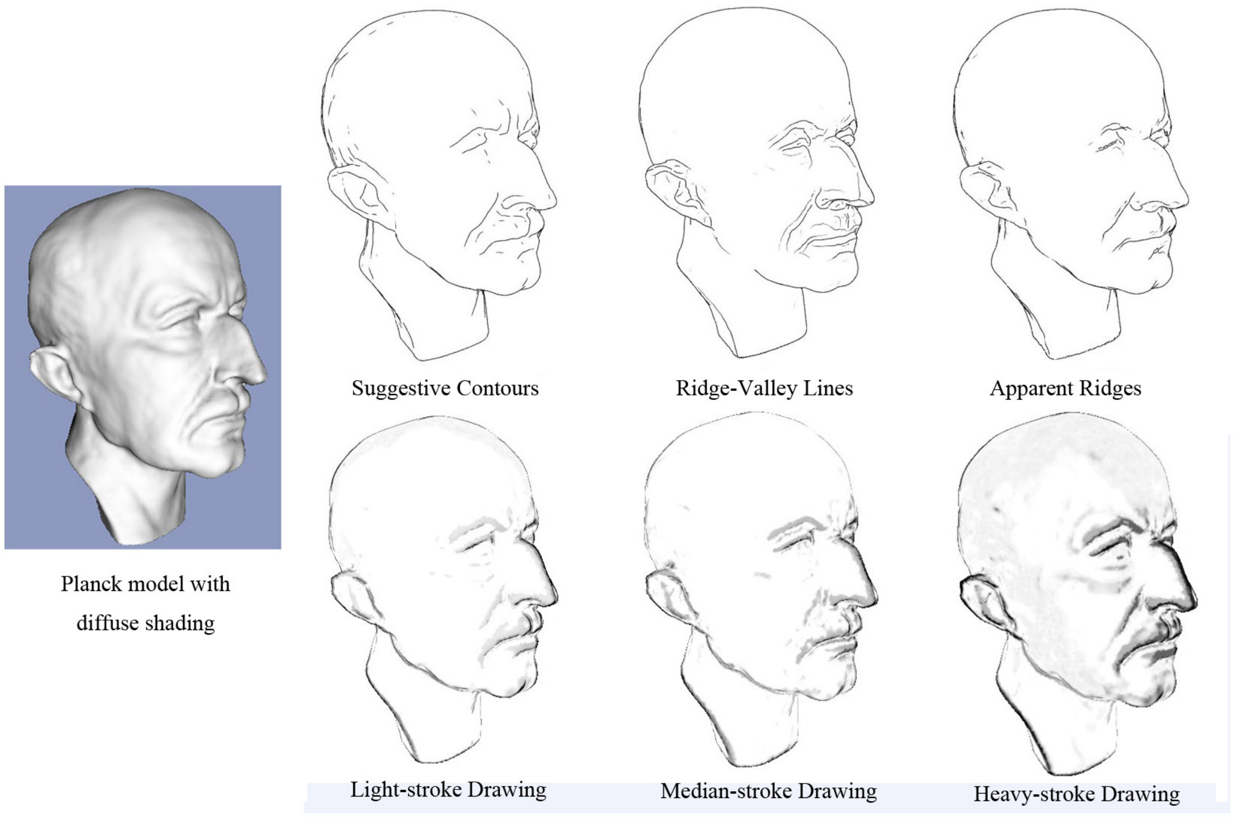

Fig. 1 Planck model with different line-drawing strategies. The left image is the normal display in diffuse shading. The top figures are line-drawings respectively using *Suggestive Contours*, *Ridge-Valley*, and *Apparent Ridges*. The bottom figures are the results with different intensity and width of the ink stroke which are controlled by our *Shape-cues*.

ods are not fulfilled to such goal. As showed in Fig. 1, there is presently no single rule which can generate the desirable lines, so we prefer to multiple combination of certain shape cues. We design our method as a multi-pass rendering process using both view-dependent and view-independent shape cues including view-curvature, depth and principle curvatures. The shape-cues are obtained in the first pass and then we do necessary calculations in next passes, and finally render ink strokes according to them.

The main contribution of this paper is to introduce a Chinese painting simulation system which can work in real-time based on GPU accelerating. The shape cues are acquired through shading, from which we give a stylized expression of the objects' shape. Moreover, in order to extend lines to ink strokes, we combine view-dependent lines with view-independent strokes. By investigating the relation between the shape and view-independent shape cues we get the similar simulation with a well-designed stylized-texture.

The remainder of this paper is organized as following: Sect. 2 reviews related works in line-drawing and Chinese painting simulation, Sect. 3 introduces the definition and computation of our shape-cues, Sect. 4 describes how to extend lines to strokes, Sect. 5 present the whole algorithm in detail, Sect. 6 is about discussion of the method and future works.

2 Previous Work

Silhouette/Contour Algorithm. The early classic silhouette extraction method is a simple *image-space method* which works as frame-buffer process, by tracking discontinuities of shape features (depth [26], object IDs [15], direction of normals). And another classic one is *object-space method* which detects the feature lines in object space and renders these lines as strokes with various stylized types. Many such approaches have been proposed where the *silhouette abstract* algorithm is regarded as the key process for its strong ability to differentiate object and background [14][16]. Markosian first introduced a real-time silhouette extraction using interframe coherence and fast visibility determination [20]; Olson attempted to extracted silhouette in another special space [24]. Northrup first use a hybrid method which combined the advantages of image- and object-space algorithms but is complex [22].

Besides silhouettes, other feature lines are also considered, including suggestive contours which locates sugges-

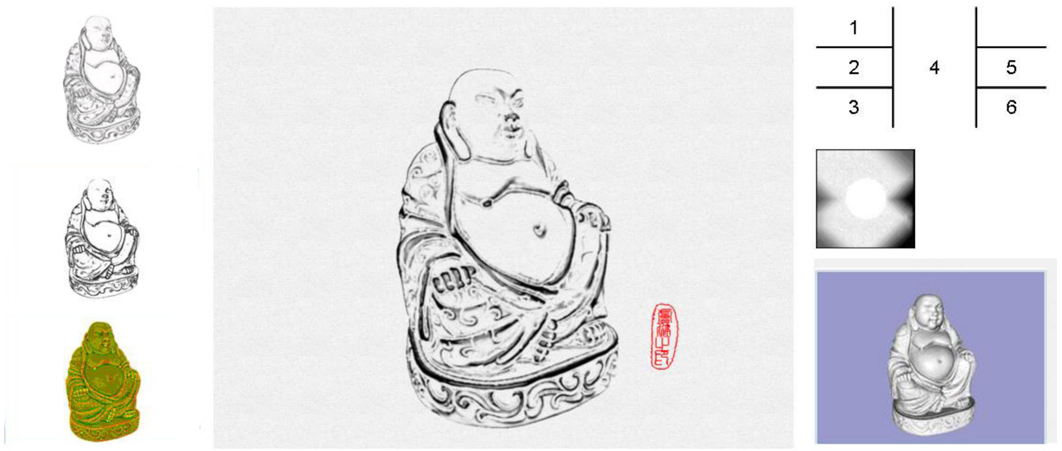

Fig. 2 An example of Eastern ink Painting Simulation. (1) is LoG of C_{view} and (2) is the gradient of C_{view}, both (1) and (2) provide a view-dependent sketch of the silhouette; (3) is the *principal curvatures* cue which is used for view-independently picking up intensity in a designed stylization texture in order to extend the silhouette sketch to strokes; (5) is a certain type of stylization texture we used in practice; (4) and (6) are auto-generated result and the common diffuse vision of the 3D mesh polygon.

tive contours alongside contours to convey a more complete shape impression [8][9], geometric Ridges and Valleys [23], and Apparent Ridges [18] which is a new object-space definition of lines that extracts ridges using a view-dependent transfer to curvatures. Ni et al. [21] extend suggestive contours algorithm to a multi-scale which make it suit LODs automatically.

Using Continuous shape shading. There appear some novel methods using continuous shape cues through shading, we call them *continuous shape shading* method. Highlight lines [10] is an extension of suggestive contours which use diffuse shading information. And Lee et al. [19] rendered a tone image to describe the scene and then render the lines according to the tone image. Apparent relief [28] is another type of shape descriptor where the continuous shape cues are principal curvatures. Zhang et al. [32] presented a real-time Laplacian Lines drawing algorithm which inspired by the Laplacian-of-Gaussian (LOG). Most of these methods are two-pass process which first renders the shape cues into the image plane and then detect the features lines in the second pass to produce more detailed crease lines.

Chinese painting simulation. Most methods of Chinese painting simulation are WACOM-based which are more likely CAD tools for artists. For example, Chu and Tai simulated ink dispersion in absorbent paper [2][3], and Yeh et al. [29] used force feedback device to construct a 3D brush and simulated the ink-water transfer system . There are also some other simulation methods for special objects: Zhang et al. [33] focused on water animation, Yu et al. [31] synthesized Chinese landscape painting using an image-based approach. Yeh and Ouhyoung [30] automatically generated Chinese painting from simple 3D animal models.

3 Shape Cues

3.1 Background and Definition

Feature lines. In a view of a smooth and closed surface $\mathscr{S}$, the *silhouette* (named *contour* in some reference) is defined as the loci of points that lie on this surface and satisfy: $\mathbf{n}(\mathbf{p}) \cdot \mathbf{v}(\mathbf{p}) = 0$, where $\mathbf{p} \in \mathscr{S}$ is a point on the surface, $\mathbf{n}(\mathbf{p})$ is the unit surface normal at $\mathbf{p}$, $\mathbf{c}$ is the camera position, and $\mathbf{v}$ is the view vector: $\mathbf{v}(\mathbf{p}) = \mathbf{c} - \mathbf{p}$. When working in a polygon mesh, the *silhouette edge* is the edge shares one front- and back-facing polygon. Besides, *creases* are edges should always be drawn, typically identified by comparing the dihedral angle with a threshold. Most present line-drawing methods add crease lines to silhouette lines in order to enrich the description of shapes.

Curvature domain. For a plane curve, the curvature $\kappa(\mathbf{p})$ at a point $\mathbf{p}$ is the reciprocal of the radius of the circle that best approximate the curve at $\mathbf{p}$. Curvatures make this sense: discontinuities in curvature values represent silhouettes or creases, concavities correspond to positive curvature and convexities correspond to negative curvature, the inflection points correspond to zero curvature. When a one-dimensional curve lies on a two-dimensional surface embedded in three dimensions $\mathscr{R}^3$, the measure of curvature should take the surface unit-normal $\mathbf{n}$ into account. Any non-singular curve on a smooth surface will have its tangent vector $\mathbf{T}$ lying in the tangent plane of the surface orthogonal to $\mathbf{n}$. The normal curvature κ_n is the curvature of the curve projected onto the plane containing the curves tangent $\mathbf{T}$ and the surface normal $\mathbf{n}$. All curves with the same tangent vector will have the same normal curvature, taking all possible tangent vectors then the maximum and minimum values of the normal curvature at a point are called the *principal curvatures* (κ_1 and

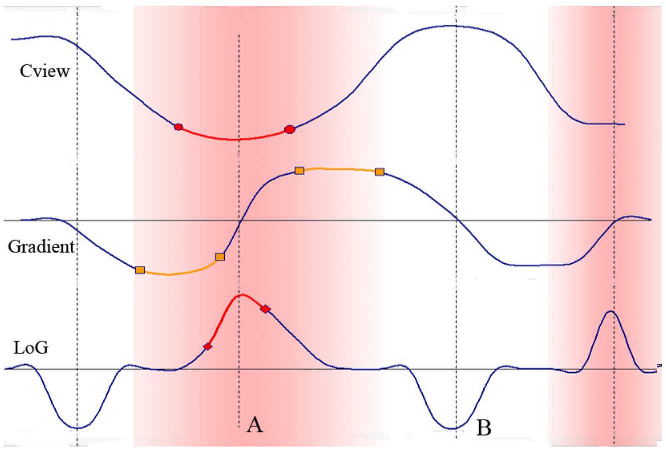

Fig. 3 A 1D example of view-dependent cues: the first row is values of C_{view} and the first order and second order derivatives which are computed as the gradient and LoG (Laplacian of a Gaussian) of the C_{view}. In the 1D filed, the neighborhood (shaded part) of maximal LoG value and extreme gradient should be considered as the sketch strokes.

κ_2), and the directions of the corresponding tangent vectors are called *principal directions*.

View dependence. Nearly all the successful line-drawing algorithms depict shape with view dependent feature lines, [8][9][18][28][32] because it is nature whether artists draw lines along one edge of objects is determined by where he is standing. In hand-drawing paintings, lines appear differently from the different viewpoints. Therefore, View-independent lines alone always do provide a natural shape looking over the surface. However, Chinese painting is a plane painting and strokes are needed not only on the border but also in the center of the view area. For this reason, our approach combines the view dependent and independent information, both are largely adapted from or inspired by methods already described in literatures for efficient line-drawing rendering.

3.2 View Dependent Cues

We define *View Curvature* to approximate curvatures in the view direction. Inspired from the image-space solution to suggestive contour generator[8], we compute view dependent cues in the image space: $C_{view} = |dot(\mathbf{V},\mathbf{N})|$. DeCarlo's image-based approximation is to detect the suggestive contour generator where is defined as the minima of $\mathbf{n}\cdot\mathbf{v}$. However, due to the rough character of ink strokes, our approach is distinct from suggestive contours: we compute first and second order derivatives of C_{view} alternatively and expand the minimal area, such alteration makes strokes rough and ink diffused. In discrete computation we use Sobel gradient and Laplacian of a Gaussian (LoG) to estimate first- and second-derivatives. Fig. 4 illustrates the difference between classic suggestive contours and our sketch strokes.

To explain our method more clearly, we make a 1D illustration (see Fig. 3): first of all, the minimum of C_{view} should locate where the second derivative is the positive extremum. Therefore, the point with maximal LoG is on the feature line which we called sketch-point. In the figure, the segment around A is feature loci but B is not. However, in order to extend lines to strokes we should also expand the minimal part around the sketch-point. Intuitively, points with extreme gradient beside the minimal part are able to supplement to strokes.

Referring to Fig 3, extreme-gradient points closed to A should be more considered as the strokes part than those far away A. For this purpose, we apply a modified-Gaussian kernel on gradients. The process is: we first apply Sobel operator and LoG operator [12] on C_{view} to detect gradients and LoGs. Viewing charts of the 1st- and 2nd-derivatives, between the two extreme gradient part around the sketch-point (minimal C_{view} and maximal LoG), the LoG value decreases and $|gradient|$ increases away from A. Inspired by this, we modified the weights of a simple 3×3 Gaussian kernel as showed in Fig. 5. We alter the value of w1-w2 and w3-w4 responding to relation g1-g2 and g3-g4 (LoG values in the corresponding lattice). For example, when g1 $<$ g2 we increase the w1 and decrease w2 accordingly because we consider that g2 is closer to sketch-point than g1 and the value of $|gradient|$ corresponding to g1 is larger. The kernel will increases the $|gradient|$ near the sketch-point and make the strokes more smooth.

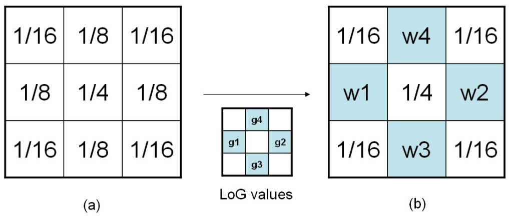

Fig. 5 The modified Gaussian kernel. We alter the value of w1-w2 and w3-w4 responding to relation g1-g2 and g3-g4.

View-dependent lines will be weak or disappeared when the object is too close to the viewpoint, one method to overcome this weakness is to revise the direction of view-vector $\mathbf{v}$, but we prefer to apply a depth-based exponential enhancement on the view-dependent cues at each pixel:

$$g = \mathscr{G}(\text{Sobel}(C_{view})), l = \text{LoG}(C_{view})$$
$$e = a \times [b - smoothstep(0.0, D_0, D)] - 1.0;$$
$$G = g^e, L = l^e$$

where $\mathscr{G}$, Sobel, LoG are respectively modified Gaussian, Sobel and LoG operator, D and D_0 are current and moderate depth value in scenes, G and L are final gradient and LoG values. The function smoothstep(min,max,x) returns 0 if x<min and 1 if x>max, returns a smooth Hermite

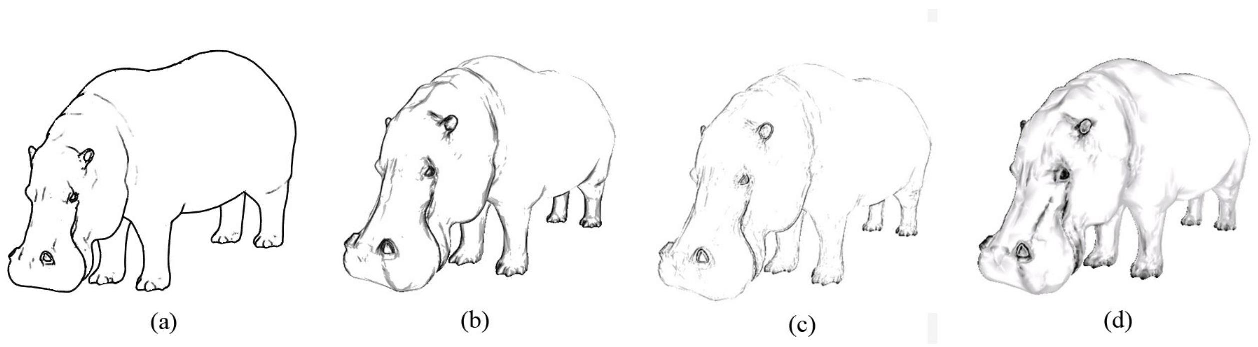

Fig. 4 Hippo model rendered with suggestive contours and our feature liens. (a) is the result of suggestive lines; (b)(c) are the visualizations of the gradient and LoG of C_{view}; (d) is the final result of our methods which is combined with the view-independent information together.

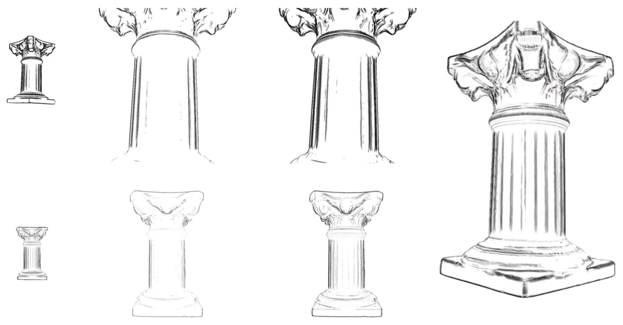

Fig. 6 Depth-based enhancement. We apply a depth-enhancement on gradient (top row) and LoG (bottom row). The left column is in moderate view distance; the middle is in the close view distance without depth-enhancement; the right column is the result with view-independent cues. The final result with view-independent rendering is displayed also.

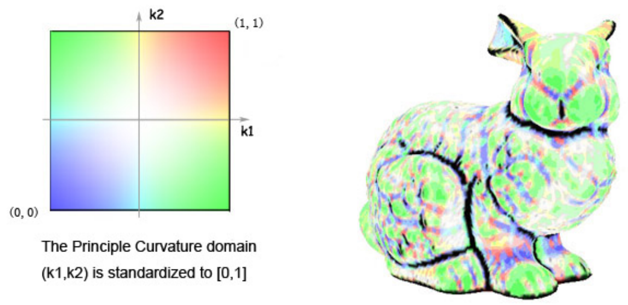
k2
(1, 1)
k1
(0, 0)
The Principle Curvature domain
(k1,k2) is standardized to [0,1]

Fig. 7 Principal Curvature Space(PCS). The left image is a color-coded illustration of PCS where (κ_1, κ_2) is standard-normalized to $[0,1]$; on the right there is a "bunny" using the left image as the stylizing-texture with $\alpha = 10.0$ (see Eq. 1).

interpolation between 0 and 1 if x is in the range [min, max]. We use $a = b = 2.0$ and $D_0 = 5.0$ in practice. As showed in Fig. 6, with all the parameters same, the method with depth-enhancement lose less lines in a very close view distance. Moreover, after combining with view-independent cues rendering, this artifact will be ameliorated further.

3.3 Principal Curvature Space

Inspired by Apparent relief [28], we utilize the shape cues space where the axes correspond to the principal curvatures κ_1 and κ_2. For each pixel we extract a vector: $\mathbf{C}_\kappa = (\kappa_1, \kappa_2)$ in the Principal Curvature Space (see Fig. 7). The orientation of $\mathbf{C}_\kappa$ provides the information about the surface convexity and the magnitude measures the surface curvedness. Apparent relief divides the computation into direction and magnitude parts for the view dependency purpose and its view dependence comes through the image-space curvedness computation according to the variation across normals in pixel neighborhood. However, we consider $\mathbf{C}_\kappa$ totally as view-independent cues and render them into pixel plane directly because we already have C_{view} as view-dependent cues.

To estimate principle curvatures, considering a surface $\mathscr{S}$ and its Gauss map from point $\mathbf{p}$ of $\mathscr{S}$ to oriented normal vector $\mathbf{n}(\mathbf{p})$, the derivative or Jacobian $\nabla\mathbf{n}(\mathbf{p})$ of Gauss map measures the variation of the normal in the neighborhood of $\mathbf{p}$. And the eigenvalues and eigenvectors of $\nabla\mathbf{n}(\mathbf{p})$ are the principal curvatures and principal directions at $\mathbf{p}$. Based on this, we compute curvatures of each vertex in the object space as the pre-process computation. Then we render curvatures into image plane participating the following rendering pipeline. Moreover, there must be some sharp transitions in the curvature image and we should avoid such discontinuities for the aim of the smooth varying curvature shading among the regions with different convexities, so there will also be a blur-process for smooth purpose.

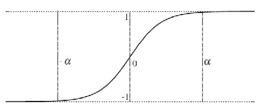

Fig. 8 Scaling function.

Before the smooth step we firstly remap the range of the curvatures because PCS values in $[-\infty, +\infty]$ should be transfer to $[-1, -1]$, which will be regarded as texture coordinates in the next stages. We apply the following simple scaling function which remaps $[-\infty, +\infty]$ to $[-1, -1]$, and for each $|x| > \alpha$ the value will be limited to 1 or -1.

$$\varphi_\alpha(x) = \tanh(\frac{x}{\alpha}) \tag{1}$$

4 Where to draw strokes?

Cole [4][5] had presented the study in analyzing the artists' hand-drawing with the computer generated line-drawing and investigate the inherence of how people locate lines. However, Chinese painting is more ruleless than common line-drawings and it seems impossible to make a reliable comparison to the artists' work. What is more, since our method is almost image-based rendering, it is not easy to control the stylized extension from lines to strokes as we mentioned above. Yet we can give an approximated illustration about where to draw strokes according to view independent cues we focus on. By using a set of color-coded textures we attempt to explore the relationship between such cues and the location of strokes.

Rule1: the boundary of PCS should be reserved as the strokes locating where the shape changes tempestuously. The color-coded texture showed in the left image of Fig. 7 is a radial gradient map with different color except its boundary with a 2-pixel width dark color. According to the "bunny" result on the right, we can realize that the black colored areas just present the basic silhouette strokes of models. Using a more obvious boundary texture we obtain basic silhouette strokes as a result. (see the first row in Fig. 9)

Rule2: the diagonal corner of PCS should be reserved as the watery strokes presenting the hunch-up or indent surface. According to the second row in Fig. 9, we can realize that the top-right corner of PCS (positive great κ_1 and κ_2) represents the hunch-up shape of the object. Moreover, the bottom-left corner of PCS (negative great κ_1 and κ_2) represents the indent shape of the object. Both of the two parts should be rendered with watery intensity.

Rule3: the bottom-right corner of PCS should be reserved as the dark strokes presenting the peaks or end-points. The top-left corner should be neglected because theoretically there is no such pair of curvatures with negative κ_1 and positive κ_2 and it clearly showed in the 3rd row of Fig. 9 that there is no need to concern the bottom-left corner. Moreover, the bottom-right corner part of PCS should be covered with dark intensity notablely, since this part often appears on the shape peaks or the end-points of the object. (see the horn of the cow)

Rule4: The center part of PCS should be left with a blank space to present the smooth surface. As showed in the fourth row of Fig. 9, the center part of PCS (small κ_1 and κ_2) always represents the smooth surface of the object.

Rule5: the vertical middle part of PCS should be neglected. As showed in the 5th row of Fig. 9 by using vertical-colored texture, most of objects we tested make no response to such part except some extraordinarily complex with tiny reactions. In order to make clear, we show two complex models with only few reactive pixels.

Rule6: the right part of horizontal middle part of PCS should be reserved with the watery strokes and the left with weaker ones. We find that the left part (negative k1 and k2=0) represents the concave and makes less contribution to the final strokes. Moreover the right part (positive k1 and k2=0) represents the convex and we find that in most of our tested model it represents the well-shaped convex, especially tube shape. (see the last row of Fig. 9, fingers, leg of horse, and tail of cow.)

5 Algorithm

5.1 Algorithm pipeline

Pre-process We first load the 3D model and compute the principal curvatures for each vertex on the polygon in object space. In practice, we apply the algorithm in [25] which gives a general and robust algorithm for estimating smooth normals and its high-order derivatives on surfaces approximated in triangle meshes.

1st Rendering Pass In the first rendering pass, we produce both view dependent and independent maps with the help of Multiple Render Target(MRT) technique. [6][7] We render the values of $C_{view}, (\kappa_1, \kappa_2)$ and z-buffer into textures in a single pass.

2nd Rendering Pass As mentioned in Sect. 3, we blur principal curvatures and compute view-dependent cues in the fragment shader. We render these cues into two distinct textures named as BlurMap with 4 tunnels $[\kappa_1, \kappa_2, z, C_{view}]$ and CuesMap with 4 tunnels [Gx, Gy, Gradient, LoG]. In fact, only (κ_1, κ_2), Gradient, depth and LoG would be used in the next pass, we fulfill textures only for the possible future extension.

During the blur process, we simply integrate principal curvatures in small neighborhoods of the image plane by using a smooth integration kernel. In practice, we apply a 2D Gaussian blur to (κ_1, κ_2) tunnels:

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Fig. 11 5x5 LoG kernel.

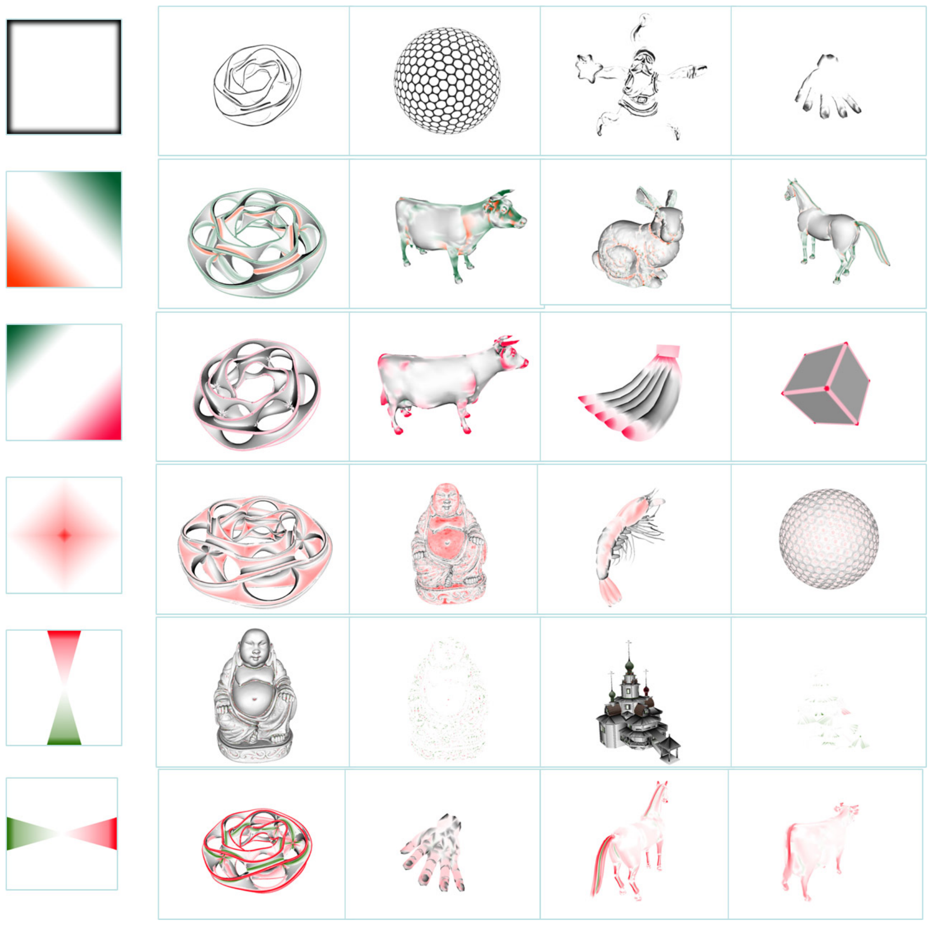

Fig. 9 Test results with different colored-coded textures. In order to observe easily, we render some results with diffuse mode first and then add the color.

$\kappa_1^{blur}(x,y) = \kappa_1(x,y) * g(x,y,\sigma)$
$\kappa_2^{blur}(x,y) = \kappa_2(x,y) * g(x,y,\sigma)$
where σ is a scale parameter of the blur kernel. We handle the blur process with $\sigma = 3$. Meanwhile, we apply Sobel and LoG operator[12] on C_{view} map to detect Gradient and second derivative. In our practice Sobel is computed in a 3×3 kernel:
$G_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$
$G_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$
$Gradient = |G_x| + |G_y|$
and the expression of LoG is:

$$\nabla^2 h(r) = -[\frac{r^2 - \sigma^2}{\sigma^4}]e^{-\frac{r^2}{2\sigma^2}}$$

which is a 5×5 kernel for discrete computing. (see Fig. 11)

3rd Rendering Pass In the final pass, we obtain the texture coordinates from vector(κ_1, κ_2) and pick up intensity from the stylized-texture (see Fig. 10) which is designed carefully by the rules we mentioned in Sect. 4. Then we composite gradient and LoG of C_{view} with some adjustable parameters to decide the final intensity at each pixel.

5.2 Parameters

We set several parameters to control the effect of the final image. The first parameter is α which controls the remapping range of the principal curvatures (see Eq. 1) together with $C_{threshold}$ which limits the blank space by setting pixels white:

$$I_0 = \begin{cases} float4(1,1,1,1) & : \sqrt{u^2+v^2} < C_{threshold} \\ tex2D(texSampler,(u,v)) & : otherwise \end{cases}$$

α is very important to range the blank area and control the interior ink-diffused characteristics of Chinese painting which is determined by the proportion of blank area and the intensity of diffused area. (see Fig. 12) Moreover, the moderate value of α will vary largely across polygon meshes with different complexity.

The second parameter f_g is the factor of C_{view} influence by which we combine sketch lines with strokes. We use linear function in this part:

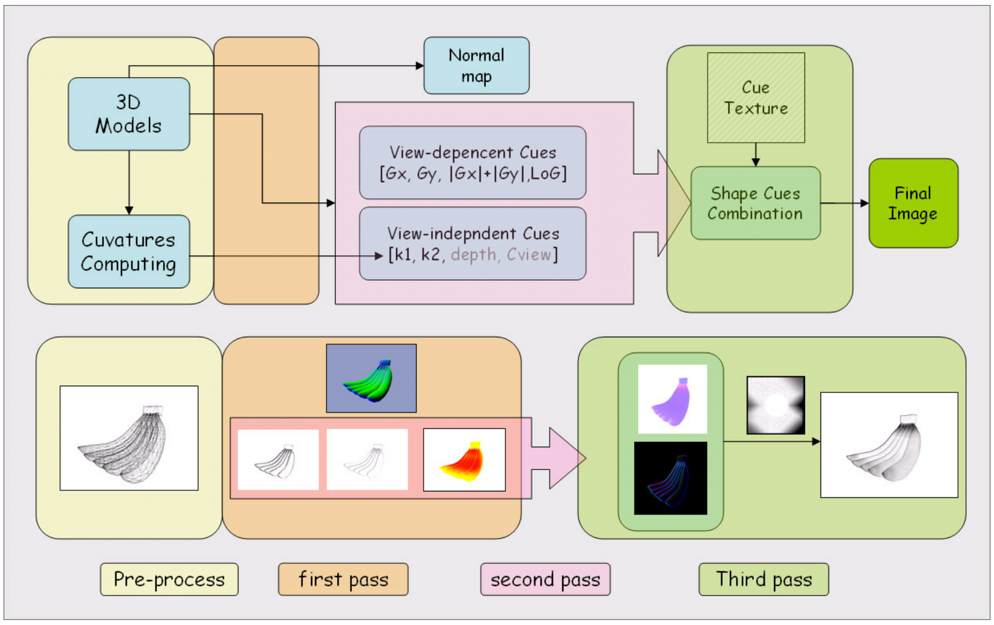

Fig. 10 The pipeline of our algorithm. Our method combines with a pre-process and a multi-pass rendering process. The top is the flow chart of the whole algorithm, and the bottom is the illustration of rendering a "banana" model.

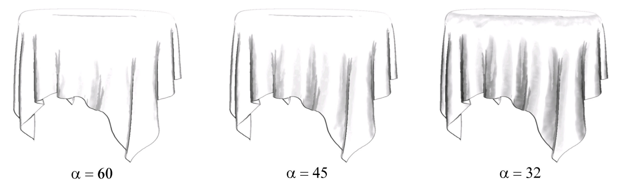

Fig. 12 Interior space's diffused intensity controlled by alpha parameter. With same $C_{threshold}$, the diffused intensity increases if α decreases.

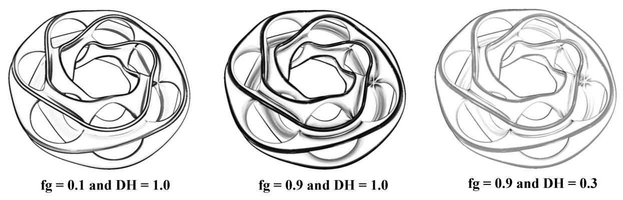

Fig. 13 The richness styles of the stroke. With same value of α, different f_g give different stroke richness corresponding to realistic or leisurely styles. Additional, with a low DH the leisurely stroke is more similar.

$I_{gradient} = 1.0 - gradient, \quad I_{LoG} = 1.0 - LoG;$
$if(I_{gradient} > G_{threshold})I_{gradient} = 1.0;$
$if(I_{LoG} > L_{threshold})I_{LoG} = 1.0$
$I = min(I_0 \times f_g + gradient \times (1.0 - f_g), I_{LoG})$

From the intuitive visual sense, the role of f_g is to control the richness between leisurely and realistic strokes with the parameter DH together. (see Fig. 13) DH is the threshold of Dot(V,N) which controls the overall density:

$I = I * (1.0f + saturate(dot(\mathbf{V}, \mathbf{N}) - Dot_{threshold}));$

where saturate(x) clamps x within the range of [0,1].

As we enhance view-dependent cues according to z-values, we also provide a depth-based control on the "line width" according to the distance from the viewpoint. We apply the third parameter f_d ranged in [0,1] to amend $C_{threshold}$:

$$a(D) = 1.0 + smoothstep(-\beta, \beta, \frac{D - \frac{D_{min}+Dmax}{2}}{D_{max} - D_{min}}) \times f_d$$

where D is the current depth and D_{min} and D_{max} are nearest and farthest depth of the scenes, β is a positive value. (we use $\beta = 0.5$ in practice) We divide $C_{threshold}$ by α in order to change the stokes' width, it will reduce the width if $\alpha < 1.0$ and increase if $\alpha > 1.0$. Therefore, both f_g and f_d provide a control on the stroke width. However, in a sense, f_g provides interior and detailed changes and f_d provides outside and rough changes. (see Fig. 14) In fact, the width control of final strokes is always an integrated result. (see the bottom of Fig. 1)

6 Discussion and future work

6.1 Image-based Rendering

Except the pre-computation of principle curvatures in object space, our method is almost image-based. Therefore, the results are automatic LODs obtained. (see (a) of Fig. 15) Moreover, computation in image-space is appropriate to parallel computing on GPU graphics card and so our method is totally real-time with compared with suggestive contours or Apparent lines in interactive framerates.

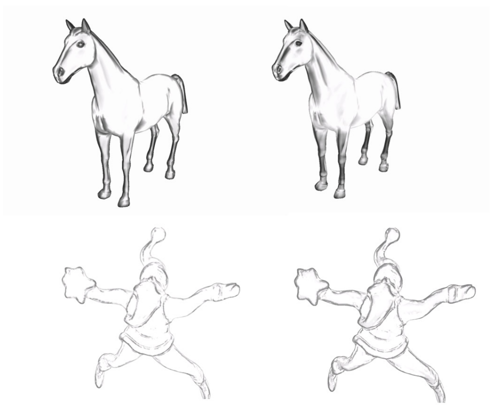

Fig. 14 Varying the width of strokes. Top row is width change due to f_g: $f_g = 0.2$ and $f_g = 0.7$ with same $f_d = 0$; Bottom row is width change due to f_d: $f_d = 0$ and $f_d = 0.30$ with same $f_g = 0.5$..

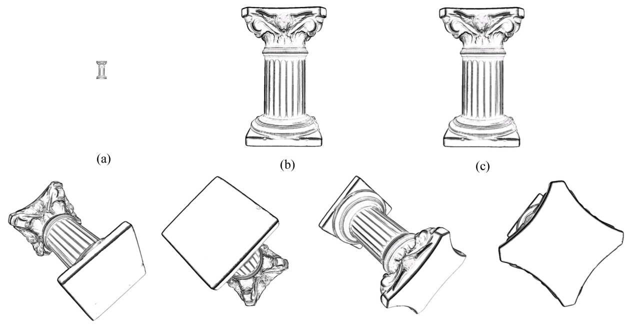

Fig. 15 Image-based Rendering with a blur post-process. (a) is a distant view, (b) and (c) are moderate views respectively with and without blur post-process. The bottom row is screenshots through the viewpoints around the column.

Because our method totally works in the image-space, the width of lines cannot be uniform and maybe some tiny zigzag appears on the edge because of limited image resolution. For improvement purpose, we add a Gaussian blur post-process to the final image. As showed in Fig. 15, such post-process will improve the performance but not influence the coherence across different viewpoints. On the top of the column (c) is more smooth than (b) and frames change smoothly when we rotate the viewpoint.

Another weakness of image-based methods for NPR purpose is difficult stylization. It is more hard to stylize lines in the image space than in the object space since little information of lines is reserved in the rasterized image plane. Our strategy is to utilize a specific stylization texture which appears below again in Sect. 6.3

6.2 Parameters discussion

Frankly speaking, one limitation of our method is that there are too many parameters to adjust the final result, but we provide a simple and intuitive user interface to choose the values of the parameters. In addition, the value selections of α and f_g is critical to the final effect, and for different complexities of the polygon meshes, their values vary dramatically. One of our future work is to decrease the number of parameters and try to seek a uniform pattern among differently complex models.

6.3 Stylization texture

The stylization texture in our system is designed by ourselves by following rules described in Sect. 4. However, the stylization texture itself also has the potential for its alterable stylization picking in the 2D space. By applying different stylization textures we can obtain different shading style results. For example, by selecting fantastic textures and considering texture as color instead of intensity, we can obtain some other interesting NPR results. (See Fig. 16)

6.4 Ink Dispersion

During simulating Chinese paiting, there are three things that we should notice in terms of composing methods: brush, ink, paper. To simulate more similarly, ink dispersion in absorbent paper must be considered. However, ink dispersion in paper itself if a individual complex problem. One of practical methods is to consider ink and water dispersion as fluid simulation and provide a physically-based solution. Our approach presently is geometrically-based method and our future work is to extend our work with ink dispersion simulation. For example, we will utilize discrete Lattice Boltzmann Method (LBM) [1][27] which is a class of computational fluid dynamics methods for fluid simulation.

Acknowledgements

References

1. Begum, R., Basit, M. A., Lattice Boltzmann Method and its Applications to Fluid Flow Problems, European Journal of Scientific Research, 22, 2, 216-231. (2008)
2. Chu, Nelson S. -H., and Tai Chiew-Lan, Real-Time Painting with an Expressive Virtual Chinese Brush, *IEEE Computer Graphics and Applications*, 24, 5, 76-85. (2004)
3. Chu, Nelson S. -H., and Tai Chiew-Lan, MoXi: real-time ink dispersion in absorbent paper.*ACM Transactions on Graphics (TOG)*, 24(3), 504 - 511. (2005)
4. Cole, F., Golovinskiy, A., Limpaecher, A., Barros, H.S., Finkelstein, A., Funkhouser, T., and Rusinkiewicz, S., Where do people draw lines? *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 27(3), 1 - 11 (2008)
5. Cole, F., Sanik, K., DeCarlo, D., Finkelstein, A., Funkhouser, T., Rusinkiewicz, S., and Singh, M. , How well do line drawings depict shape? *ACM Transactions on Graphics*, 28(3), 1 - 9 (2009)

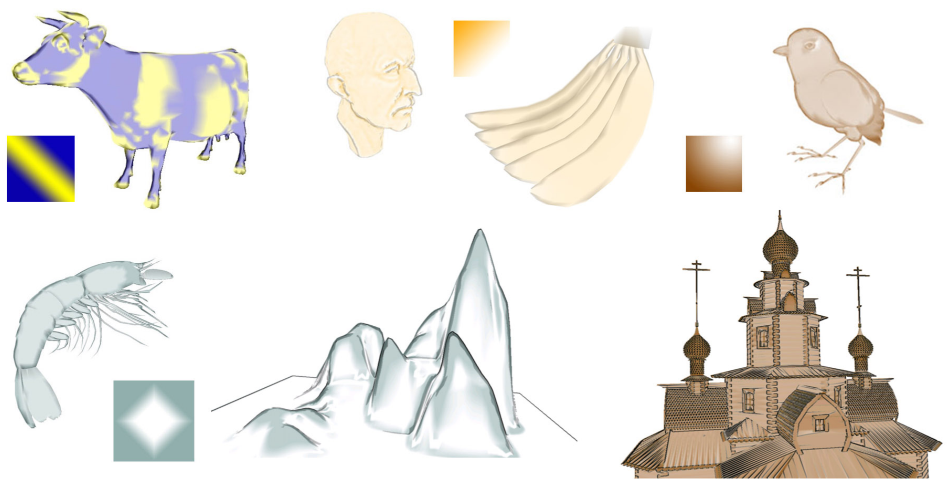

Fig. 16 Results with various stylization textures. With different simple stylization textures, one can obtain various kinds of NPR results.

6. Dachsbacher, C., and Stamminger, M., Reflective Shadow Maps. *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, 203-213. (2005)
7. Dachsbacher, C., and Stamminger, M., Splatting Indirect Illumination. *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, 93-100. (2006)
8. DeCarlo, D., Finkelstein, A., Rusinkiewicz, S., and Santella, A., Suggestive contours for conveying shape. *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, 848 - 855. (2003)
9. DeCarlo, D., Finkelstein, A., and Rusinkiewicz, S, Interactive rendering of suggestive contours with temporal coherence. In *NPAR 2004: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering,* 15-24. (2004)
10. DeCarlo, D. and Rusinkiewicz, S., Highlight lines for conveying shape. *Proceeding of NPAR 2007: the 5th international symposium on Non-photorealistic animation and rendering*, 63-70. (2007)
11. Elder, J.H., Are Edges Incomplete? *International Journal of Computer Vision*, 34(2), 97-122 (1999)
12. Gonzalez, R.C., and Woods, R.E., Digital Image Processing, Second Edition. Prentice Hall. (2002)
13. Gooch, B., Hartner, M., and Beddes, N., Silhouette Algorithms. Lecture about silhouette algorithms, University of Utah.
14. Gooch, B., Sloan, P.J., Gooch, A., Shirley, P., Riesenfeld, R.F., Interactive technical illustration. *Proceedings of the 1999 symposium on Interactive 3D graphics*, 31-38. (1999)
15. Hertmann, A., Introduction to 3D Non-Photorealistic Rendering: Silhouettes and Outlines. *ACM SIGGRAPH 99 Course Notes.* Course on Non-Photorealistic Rendering, S.Green, Ed.ch.7. (1999)
16. Hertzmann, A., and Zorin, D., Illustrating smooth surfaces. *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 517-526. (2000)
17. Isenberg, T., Freudenberg, B., Halper, N., Schlechtweg, S., and Strothotte, T., A Developer's Guide to Silhouette Algorithms for Polygonal Models. *IEEE Computer Graphics and Applications*, Volume 23(4): 28 - 37. (2003)
18. Judd, T., Durand, F., and Adelson, E.H., Apparent ridges for line drawing. *ACM Transactions on Graphics*, 26(3). (2007)
19. Lee, Y., Markosian, L., Lee, S., and Hughes J.F., Line drawings via abstracted shading. *ACM Transactions on Graphics*, Volume 26, Issue 3, July 2007. (2007)
20. Markosian, L., Kowalski, M.A., Trychin, S.J., Bourdev, L.D., Goldstein, D., Hughes, J.F., Real-Time Nonphotorealistic Renedering. *Proceeding of SIGGRAPH 97*, 415-420. (1997)
21. Ni, A., Jeong, K., Lee, S., and Markosian, L., Multi-scale line drawings from 3D meshes. *Proceeding of the 2006 Symposium on Interactive 3D Graphics and Games*, 133-137. (2006)
22. Northrup, J.D. and Markosian, L., Artistic silhouettes: a hybrid approach. *Proceedings of NPAR 2000: the 1st international symposium on Non-photorealistic animation and rendering*, 31-37. (2000)
23. Ohtake, Y., Belyaev, A., and Seidal, HP., Ridge-valley lines on meshes via implicit surface fitting. *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, 609 - 612. (2004)
24. Olson, M. and Zhang, H., Silhouette Extraction in Hough Space. *Computer Graphics Fortum*, Volume 25,Issue 3, 273-282. (2006)
25. Rusinkiewicz, S., Estimating Curvatures and Their Derivatives on Triangle Meshes. *Proceedings of 3DPVT 2004: the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium*, 486-493. (2004)
26. Saito, T. and Takahashi, T., Comprehensible Rendering of 3D Shapes. *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, 24(4): 197-206. (1990)
27. Succi, S., *The lattice Boltzmann equation for fluid dynamics and beyond.* Oxford University Press. (2001)
28. Vergne, R., Barls, P., Granier, X., and Schlick, C., Apparent relief: a shape descriptor for stylized shading. *NPAR '08: Proc. International symposium on Non-photorealistic animation and rendering, ACM*, 23-29. (2008)
29. Yeh Jeng-sheng, Lien Ting-yu, and Ouhyoung Ming. On the Effects of Haptic Display in Brush and Ink Simulation for Chinese Painting and Calligraphy. *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, Page: 439. (2002)
30. Yeh Jun-Wei and Ouhyoung Ming. Non-Photorealistic Rendering in Chinese Painting of Animals. *Journal of System Simulation*, Vol. 14, No. 6, pp. 1220–1224 and pp.1262, 2002.
31. Yu JinHui, Luo GuoMing, and Peng QunSheng. Image-based synthesis of Chinese landscape painting. *Journal of Computer Science and Technology*, Volume 18, Issue 1, Pages: 22 - 28. (2003)
32. Zhang, L., He, Y., Xie, X.X., and Chen, W., Laplacian lines for real-time shape illustration. *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, 129-136. (2009)
33. Zhang Song-Hai, Chen Tao, Zhang Yi-Fei, Hu Shi-Min, and Ralph R. Martin. Video-based running water animation in Chinese painting style. *Science in China Series F: Information Sciences*, Vol. 52, Nr. 2, 162-171. (2009)