Feature-constrained Texturing System for 3D Models

Tong-Yee Lee and Shaur-Uei Yan

Department of Computer Science and Information Engineering,
National Cheng-Kung University, Tainan, Taiwan, Republic of China
email:tonylee@mail.ncku.edu.tw

Abstract. Significant number of parameterization methods has been proposed to perform good quality of texturing 3D models. However, most methods are hard to be extended for handling the texture mapping with constraints. In this paper, we develop a new algorithm to achieve the matching of the features between the model and texture image. To minimize the distortion artifacts from the matching algorithm, a L2 stretch metric is also applied to optimize the u,v map defined in parameterization domain.

1 Introduction

In computer graphics, texture mapping is a very common technique to enhance the visualization of 3D meshes. Texture mapping adds the detail of pictures to the 3D meshes and let the 3D meshes looks more vividly. Surface parameterization is a common solution to the texture mapping problem. Parameterization commonly maps 3D meshes to a 2D domain that defines the (u, v) texture coordinates in parameterization domain. There is no isometric parameterization to map a general surface patch to a plane [1] and many previous methods [2–10, 16, 21] have been proposed to minimize various kind of distortion for achieving an acceptable visual effect. Most of them do not take feature matching into account. They only provide general solutions to texture mapping without constraints. However, without the correspondence of features, a 3D model with texture coordinates would look strange as shown in Figure 1. For example, the eyes on both texture and model do not match. Therefore, we need to find a method to deal with this constraint issue.

2 Related Work

Many papers have addressed issues in surface parameterization. Surface parameterization can be used for texture mapping. Tutte [15] introduces the barycentric map which guarantees the existence of a one-to-one mapping, i.e., foldover free, for parameterizing a model into the u,v domain. Eck et al. [4] minimize the harmonic energy to approximate the harmonic map and this method can be solved efficiently by a linear sparse matrix. Floater [16] develops the mean value

Fig. 1. Texture mapping using a naive non-constrained surface parameterization

coordinates to mimic the discrete harmonic map and this mapping is bijective. Hormann and Greiner [6] derive a formula that measures the distortion of the parameterization and is also invariant for affine transformation. This method does not require the boundary vertices to be fixed on the convex 2D polygon. However, the optimization of solution is slow due to the non-linear property. Sander et al. [3] define a geometric stretch metric for surface parameterization. This approach uses a relaxation approach similar to MIPS [6] to iteratively flatten 3D surfaces. Some papers address the feature correspondence problem between model and texture coordinates. Such topic is called constrained texture mapping. Levy [17] minimizes the distortion and matches features in a least square sense. This approach doesn't exactly match the constrained positions of features and may produce folded triangles in texture domain. Eckstein et al. [18] uses the graph theorem to meet the position constraints, but the method seems too complicated to handle any general case. Kraevoy et al. [19] describes a matching algorithm to align the position of features. This method uses a brute-force approach to satisfy the constraints.

3 Parameterization with feature points

In this paper, we develop a novel approach to texture mapping with constraints. Initially, a user only needs to specify several feature correspondences between the model and the texture and then the algorithm would automatically do anything without user-interaction.

3.1 Initial parameterization

We concentrate the feature-correspondence problem in this paper. The topology of the input 3D mesh is homeomorphic to a disk. If objects are not belonging to this type, it is easy to use extra cutting to form an open disk-like surface, i.e., surface with a boundary [10]. Then, we can use any parameterization method mentioned in the related work. At the current implementation, we adopt the mean value coordinates [16] with a convex boundary in uv domain (see Figure 2 and Figure 3 (c)). Figure 2 shows the weights of the mean value coordinates for each vertex of the model. If the feature points are specified on the border of models, a virtual boundary (Figure 3(d)) can be added to the outside of the parameterized surface boundary to keep all features in the interior of the u,v map. This arrangement can facilitate our method described in Section 3.2.

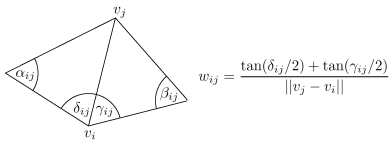

Fig. 2. v_j is the one-ring neighbor of v_i for triangulation mesh

3.2 Feature matching using a partitioning approach

This section is the core of this paper. The matching idea is mainly based on the procedure of the recursive partitioning in uv domain. In our setup, all computation of the matching stage is performed between the uv map and the texture. After initial parameterization, each vertex of the 3D model has an initial (u, v) coordinate. But the feature vertices don't be matched with corresponding positions on the texture. We wrap the feature vertices to their desired positions by the partitioning approach.

Partitioning. For each partitioning step, we can group the feature vertices into two disjoint sets as follows. First, feature vertices are sorted by the u (or v) coordinate on the texture domain and then the median u_m (or v_m) is computed. Let the line segment L_t: $x = u_m$ (or $y = v_m$) be the partitioning line on the texture domain to separate the feature vertices into two groups G1 and G2 (see Figure 4 (a)). Then we perform the path finding algorithm in the next section to find a path P to isolate the counterparts of G1 and G2 in the uv domain (Figure 4 (b)). A path P has the same starting and ending positions as those of the line L_t. Then, after partitioning by a path P, we can have two disjoint sub-patches SP1 and SP2 in the uv domain. In the next step, we align the path P with the line L_t, i.e., straightening the path P, and then re-parameterize the sub-patches SP1 and SP2 (see Figure 4 (c)). A partitioning step ends. We will iteratively above partitioning tasks on the uv map until each patch contains only

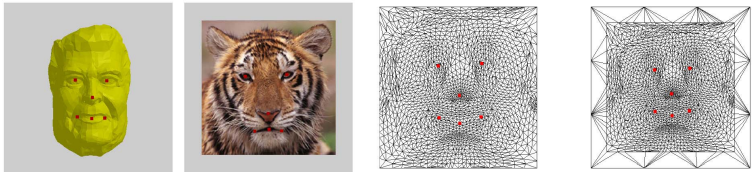

Fig. 3. (a) and (b): feature correspondence between a 3D model and a texture image; (c) and (d): the uv map of the mean value coordinates with and without virtual boundary

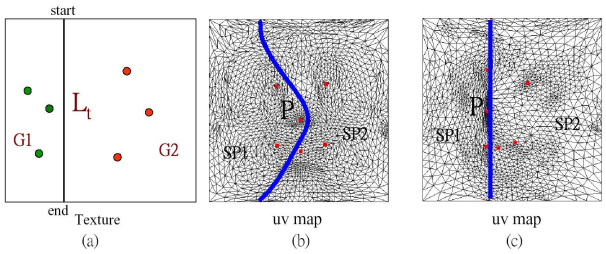

Fig. 4. Each partitioning step determines two separate groups of features in texture and uv domains.

one feature point, say f. Finally, we will align each feature f in uv domain with its counterpart f' in the texture domain by a quad-partitioning of each patch as shown in Figure 5. For this purpose, we move a vertex f to a new position f', i.e., correct corresponding position in texture domain. Then, we straighten all paths connecting f and four corners and re-parameterize these four sub-patches.

Finding a partitioning path in uv domain. First, we apply a constrained Delaunay triangulation to the u,v map considering the feature vertices, S, and E (Figure 6 (b)). Both S and E are the starting and ending points for a partitioning path. Second, we compute the minimal spanning trees (MST) for the group G1 and G2 (Figure 6 (c)), respectively. If a MST does not exist, we will apply 1-to-4 subdivision on this triangulation map to have additional paths for finding MSTs. After finding MSTs, we find all edges, i.e., marked by X, in which one of the end points belongs to G1 and the other belongs to G2. When connecting the middle points of these edges, we can form a connected path P like Figure 6 (d). Finally, the starting and ending points of P connect to S and E by the shortest paths L1 and L2 to determine the final partitioning path. This partitioning path divides features into two disjoint groups in uv domain.

3.3 UV optimization

After matching features, the uv map is distorted to some extent. The uv optimization is a process to improve the uv map as the smoothing stage in [19].

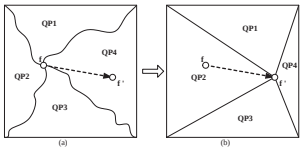

Fig. 5.

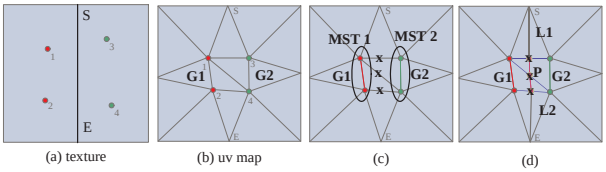

Fig. 6. Finding a partitioning path in u,v domain

It is an iterative version of parameterization. It moves each vertex except for features within the range of one ring at each iteration to gradually minimize the distortion of uv map. We use the L2 stretch metric [3] instead of the harmonic map in [19]. In our experiment, the L2 stretch metric generally produces better results than the harmonic map (Figure 7).

4 Preliminary results

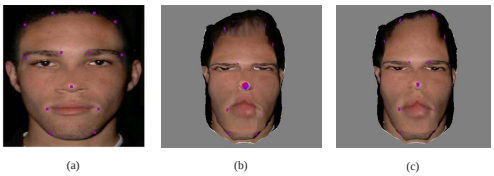

Fig. 7. Smoothing u,v map. (a):original texture, (b):smoothed with a harmonic map and (c):smoothed with L2 stretch metric.

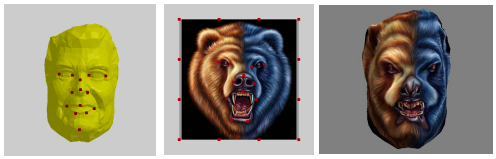

Fig. 8. Texturing an old man with a bear image.

We demonstrate some preliminary results using the proposed method. Figure 7 gives the comparison between harmonic map and L2 stretch metric for the smoothing the u,v map. In Figure 7, (c) yields better visual effect than (b).

In Figure 8, the positions of corresponding features for an old man and a bear image are very different. The proposed method produces a not bad result. Note that in this example, we specify the features on the border of texture image, therefore we need to add virtual boundary for the u,v map. Finally, we show another interesting example in Figure 9. In this case, we do not require virtual boundary. We perform the experiments on a PC with Pen-tium IV 2.4 GHz and 512 MB RAM. On the average, it takes a minute to finish a texture mapping for these examples.

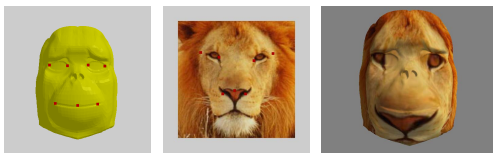

Fig. 9. Texturing a monkey face with a lion image.

5 Conclusion

In this paper, we have presented a new algorithm for the constrained texture mapping. Preliminary results show that this new method is very promising. We can successfully handle texture mapping with constraints well. In future, there are many works to be done. For example, there is also a need for matching correspondence in 3D metamorphosis application and consistent surface parameterization. We will plan to explore the possibility of our approach to these important applications. Furthermore, in the current method, we need many re-parameterizations of the patches and therefore the computation cost can be expensive as the features are increased drastically. We would like to find a better approach to reduce the number of re-parameterization in near future. Another interesting and our ongoing research is to compute progressive texture transfer between two models in metamorphosis applications [22, 23].

Acknowledgement. This paper is supported by the National Science Council, Taiwan, Republic of China, under contract No. NSC-93-2213-E-006-026 and NSC-93-2213-E-006-060.

References

1. L. V. Ahlfors and L. Sario ; *Riemann Surfaces* ; Princeton University Press, Princeton, New Jersey, 1960
2. J. Maillot, H. Yahia, and A. Verroust ; *Interactive texture mapping* ; Proceedings of SIGGRAPH, 1993, pp. 27-34

3. P. Sander, J. Snyder, S. Gortler and H. Hoppe ; *Texture mapping progressive meshes* ; Proceedings of SIGGRAPH, 2001, pp. 409-416
4. M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle ; *Multiresolution analysis of arbitrary meshes* ; Proceedings of SIGGRAPH, 1995, pp. 173-182
5. M. S. Floater ; *Parametrization and smooth approximation of surface triangulations* ; Computer Aided Geometric Design, 14(3):231-250, 1997
6. K. Hormann and G. Greiner ; *Mips: an efficient global parameterization method* ; Curve and Surface Design: St. Malo 1999, pages 153-162, Vanderbilt University Press, 2000
7. Sorkine, D. Cohen-Or, R. Goldenthal, and D. Lischinski ; *Bounded-distortion Piecewise Mesh Parameterization* ; IEEE Visualization, 2002, pp. 355-362
8. G. Zigelman, R. Kimmel, and N. Kiryati ; *Texture mapping using surface flattening via multidimensional scaling* ; IEEE Transactions on Visualization and Computer Graphics, Vol. 8, No. 2, pp. 198-207, 2002
9. G. Piponi and D. Borshukov ; *Seamless Texture Mapping of Subdivision Surfaces by Model Pelting and Texture Blending* ; Proceedings of SIGGRAPH, 2000, pp. 471-478
10. X. Gu, S. J. Gortler, and H. Hoppe ; *Geometry Images* ; Proceedings of SIGGRAPH, 2002, pp. 355-361
11. C. Gotsman, X. Gu, and A. Sheffer ; *Fundamentals of Spherical Parameterization for 3D Meshes* ; Proceedings of SIGGRAPH, 2003, pp. 358-363
12. T. Y. Lee and P. H. Huang ; *Fast and Institutive Polyhedra Morphing Using SMCC Mesh Merging Scheme* ; IEEE Transactions on Visualization and Computer Graphics, Vol. 9, No. 1, pp. 85-98, 2003
13. Sheffer and E. D. Sturler ; *Smoothing an Overlay Grid to Minimize Linear Distortion in Texture Mapping* ; ACM Transactions on Graphics, Vol. 21, Issue 4, pp. 874-890, 2002
14. U. Pinkall and K. Polthier ; *Computing discrete minimal surfaces and their conjugates* ; Experimental Mathematics, 2(1):15-36, 1993
15. W. Tutte ; *Convex representation of graphs* ; In Proc. London Math. Soc., volume 10, 1960
16. M. S. Floater ; *Mean value coordinates* ; Computer Aided Geometric Design, 20(1):19-27, 2003
17. B. Levy ; *Constrained Texture Mapping for Polygon Meshes* ; ACM SIGGRAPG 2001, 417-424
18. I. Eckstein, V. Surazhsky, and C. Gotsman ; *Texture Mapping with Hard Constraints* ; Computer Graphics Forum 20, 3, 95-104
19. V. Kraevoy, A. Sheffer, and C. Gotsman ; *Matchmaker: constructing constrained texture maps* ; ACM SIGGRAPH 2003, 326-333
20. J. Pach and R. Wenger ; *Embedding planar graphs with fixed vertex locations* ; Proceedings of Graph drawing '98. Lecture Notes in Computer Science 1547, Springer-Verlag, 1998, 263-274
21. Tong-Yee Lee and Shaur-Uei Yan ; *Texture Mapping on Arbitrary 3D Surfaces* ; Lecture Notes on Computer Science 3024, Springer-Verlag, pp. 721-730, 2004
22. Tong-Yee Lee and P.H Huang ; *Fast and Institutive Polyhedra Morphing Using SMCC Mesh Merging Scheme* ; IEEE Transactions on Visualization and Computer Graphics, Vol. 9, No. 1, pp. 85-98, 2003
23. Chao-Hung Lin and Tong-Yee Lee ; *Metamorphosis of 3D Polyhedral Models Using Progressive Connectivity Transformations* ; IEEE Transactions on Visualization and Computer Graphics, Jan./Feb. Issue, Vol. 11, No.1, pp. 2-12, 2005

Real-time 3D Artistic Rendering System

Tong-Yee Lee, Shaur-Uei Yan, Yong-Nien Chen, Ming-Te Chi

Department of Computer Science and Information Engineering,
National Cheng-Kung University, Tainan, Taiwan, Republic of China
email:tonylee@mail.ncku.edu.tw

Abstract. This paper presents an artistic rendering system for generating 3D images of Chinese paintings using graphics hardware. The user can adjust suitable parameters flexibly to generate different brush styles as his/her hobby, and see rendering results in real time. In this system, we propose a hardware-accelerated method to draw Chinese painting strokes efficiently along visible silhouettes. Three-dimensional texture and multi-texture from normal graphics hardware is used to speed up generating various brushes with Chinese painting stylized strokes. The features of the traditional Chinese painting such as ink diffusion and moisture effects are simulated. Several examples of aesthetically pleasing Chinese-paintings rendered from 3D models are demonstrated using the proposed method.

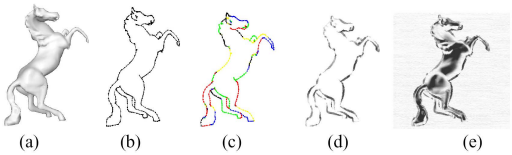

Fig. 1. System overview. (a) Input model, (b) visibility testing, (c) visible segment linking, (d) stroke placement and (e) interior shading. In (c), we color each linked segment.

1 Introduction

In the past, most non-photo-realistic rendering (NPR) researches focus on the western painting styles such as pen-and-ink, watercolor, hatching and so on. However, few works in NPR are about Chinese paintings and most of them focus on simulating delicate effects of brush, black ink and papers. Furthermore, most of them are 2D Chinese drawing works and computationally expensive for real-time applications. These researches are interested in its simulated quality rather than in its processing time. However, when generating the scene of Chinese

painting style in games or virtual environment, the real-time performance is required and we cannot use these previous works directly. In this paper, we present a real time NPR system for generating 3D Chinese paintings. The system pipeline consists of four stages and it is illustrated in Fig. 1.

2 Related Work

Strassmann models hairy brushes in his 2D oriental black-ink painting system [1]. This work represents each stroke with a cubic spline and renders the stroke using polygons with texture. Lee [2] designs a 3D brush model with elastic bristles that respond elastically to the force exerted by an artist against the paper. To simulate realistic diffuse effects of blank-ink paintings, Guo et al. consider the sorbency of paper, the liquid density and flow [3]. Zhang et al. [4] propose to use cellular automation-based simulation of ink behavior to render 3D trees in sumie style. Way et al [5] propose a method of synthesizing rock texture in Chinese landscape painting. Later, they further develop methods to simulate 3D Chinese painting trees using silhouettes with texture strokes [6]. Chan et al. [7] exploit existing software packages such as Maya and RenderMan to create 3D Chinese painting animation. Chu et al. [8] develop a system utilizing Haptic input device to track 3D brush movement to give more accurate brush simulation. Yeh et al. [9] propose a set of algorithms to render 3D animal models in Chinese painting style. To shade the interiors of animals, several basic image processing techniques such as color quantization, ink diffusion and box filtering are used.

3 Stylizing Silhouettes with Brush Strokes

3.1 Stroke Paths and Widths Generation

The idea for drawing view-dependent silhouettes of 3D model with stylized strokes is popular in NPR. We adopt Isenberg et al.' approach [10] to find visible silhouettes and concatenate silhouette segments into long stroke paths. Before applying stylizations onto stroke paths, control points along paths need to be interpolated using cubic spline to smooth the curvature of stroke paths. To make a stylized stroke path, we need to grow various widths at control points. In traditional Chinese painting, the brush width starts with thin stroke and gradually grows to thick stroke, and turns back to thin stroke as the brush stroke progresses. Yeh et al. [9] assign stroke width based on the order of control points only and potentially generate less smooth transition between different brush stroke widths when a long brush path contains fewer control points. To solve this problem, we consider distance between control points as another parameter to control the width of brush stroke. We use Eq. (1) and (2) to compute the width. In Eq. (1), ncp represents the total number of control points on a given stroke path. Eq. (2) represents the width of the stroke at a given control point i, which is 0 at both starting and ending points of the brush path. Eq. (1) returns the width to add or to subtract given the condition at the ith control point,

where Vlength[i] is distance between the ith and the (i-1)th control points and width_step is a predetermined width step value. We demonstrate an example to compare [9] and our approach in Fig. 2. The proposed approach yields better visual stroke appearance than [9].

$$add[i] = \begin{cases} +VLength[i] * width\_step, & if \quad i < \frac{1}{2} \times ncp \quad and \quad width[i] < MAX\_WIDTH \\ -VLength[i] * width\_step, & if \quad i \geq \frac{1}{2} \times ncp \quad and \quad width[i] \geq 0 \\ 0, & else \end{cases} \tag{1}$$

$$width[i] = \begin{cases} 0 & ,if \quad i = 0 \quad or \quad i = ncp \\ width[i-1] + add[i] & ,else \end{cases} \tag{2}$$

3.2 Brush-Strokes in Chinese Painting Style

A brush consists of many bristles. In a microscopic view, when a single bristle draws on the paper, the effects it can produce different ink shades such are dark, light, dry and wet when ink diluted with water; with different pressure, direction in brush stroke, different ink tones can produce millions of variations of touches on the paper. Various brush-strokes are usually used to represent different texture of the subjects in Chinese painting. To simulate different shades of ink, we define the term "pattern" to refer to the ink traces on the paper left by a bristle. The pattern setup is illustrated in Fig. 3 by combining an intensity map and an opacity (i.e., alpha) map. We can use 2D texture to store each pattern. When painting an absorbent paper, an artist can control the water content in the brush to make ink look sear, soggy or wet. The opacity map is used to control water content and therefore it is called a moisture map, too. The intensity map is used to control ink shades such as dark and light. With different combinations of bristle patterns, different style of brush strokes can be produced; Fig. 4 shows example of our simulated brush strokes in the style of flying-white (fei-bei) technique and slanted brush technique in Chinese painting. To efficiently generate the brush stroke patterns in real time, the hardware-accelerated 3D volume texture and multi-texture techniques are used in our system. Intensity changes are mapped to a 3D texture level to encode intensity change into the

Fig. 2. Different stroke widths generated by different methods. Left: stroke generated by [9] Right: stroke generated by our system.

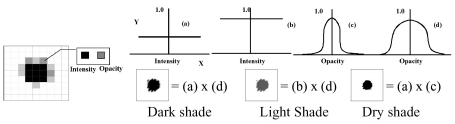

Fig. 3. Brush stroke setup on paper

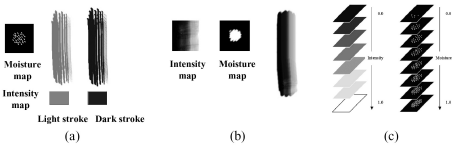

Fig. 4. (a) flying-white (fei-bei) brush path (Left). (b) slanted brush stroke path. (c) 3D texture representing intensity and moisture maps for brush strokes.

third dimension of 3D texture. Moisture changes are also mapped to another 3D texture level to account the moisture change in the third dimension as well. See figure 4(c) for a visual representation of this idea. The user just needs to prepare two sets of predetermined intensity and moisture maps. Then, we load these two sets into 3D volume texture. At running time, when an arbitrary intensity value is specified, corresponding 2D texture pattern can be interpolated efficiently from two neighboring intensity value automatically by the graphics hardware, thus a intensity map of the brush at the given intensity value is obtained. Similarly, the moisture map of a given moisture value can be computed in this hardware accelerated manner. With OpenGL extension [11], it allows us to enable multi-texture technique to combine the two maps into a new brush pattern. In this way, the system can deliver brush pattern with desired intensity and moisture value very fast using hardware accelerated 3D volume texture and multi-texture techniques. Next, we will give details about how to provide both intensity and moisture value at running time.

In Section 3.1, control points along every stroke path contain its original 3D coordinates and normal vector. Given a light source, we can compute a light vector from a control point to a light source. The intensity value of each control point is computed by the dot product of these two vectors and this dot product is normalized to the range of (0,1), so that all paths have different intensity values and are influenced by the lighting condition; distribution of bright and dark stroke can be gathered by giving different configuration to lighting as illustrated in Fig. 5. When an artist actually draws on paper, the moisture of the brush changes from wet to dry from the beginning of a stroke to the end of it. In order to simulate this effect, the control points in the beginning of each path are assigned a predetermined amount of moisture, and the moisture level in consequent control points drops as the distance from the first control point increases. See Fig. 5(c) for an example of the moisture effect. After the intensity and moisture value are

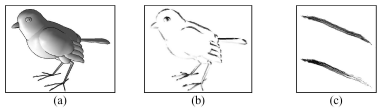

Fig. 5. By placing light, stroke brightness distribution can be controlled(Middle). Brush moisture effect(Left).

found at each point along a brush path, we can use them as the third dimension of 3D volume texture to fast compute corresponding intensity and moisture stroke "pattern".

4 Interior Shading

In this section, we present a method to draw colors in the interior area of models. The goal of this method is to fast simulate ink color change from dark to light like ink diffusion in Chinese painting. We use Eq. (3) to simulate this change in the interior of models.

$$Opacity = A * \cos^{n}(\frac{\theta}{W}) \tag{3}$$

Where θ is the angle between a vertex normal and a light vector from a vertex to light, A, n and W are constants to control shape of this function. To implement this simulation, the fragment shader provided by Nvidia's CG language [12] is used to do per pixel opacity value calculation using Eq. (3). By using the calculated opacity values and user-defined ink intensity for shading the object, the change of brightness can be seen after we enable blending. The brightness change can be seen as ink diffusion in Chinese ink painting. To avoid the dull appearance of uniform color distribution, noise functions such as Perlin noise can be used to add some randomness. In the next Section, several interesting results will be demonstrated to verify the proposed method for interior shading. In contrast to other ink diffusion approach [3, 4], the proposed method computes very fast but it yields not bad results.

5 Experimental Results

The experimental setup in this paper is a program written in C++ and OpenGL using Microsoft Visual C++ 6.0 compiler running on an Intel Pentium 4 R 2.2Ghz machine with Microsoft Windows 2000. Graphics card is Nvidia GeForce FX5900 with 256MB frame buffer. In Table 1, three models are used to test rendering speed measured in frame per second (fps) at 800x 600 screen resolutions. Because our system is implemented in four different stages, we list the frame rate (i.e., fps) at different stages. We can see the current performance bottleneck is limited by the stroke placement stage. The rendering performance we achieve is fast enough for user interaction in real time. Fig. 6 shows rendered results for Table 1. More results are demonstrated in Fig. 7.

Models	Teapot	Sparrow	Horse
Num of vertices	530	4502	6918
Num of faces	992	9184	13832
Visibility test	71.8 fps	54.9 fps	48.6 fps
Path linking	70.5 fps	51.4 fps	43.9 fps
Stroke placement	58.1 fps	20.9 fps	20.4 fps
Interior shading	54.0 fps	20.2 fps	20.2 fps

Table 1. Performance breakdown for Fig. 6.

Fig. 6. We show three rendered results used in Table 1.

6 Conclusion and Future Work

This paper presents a real-time NPR rendering system to generate traditional Chinese ink paintings for 3D models. The proposed method is accelerated by the normal graphics hardware. This method consists of four main steps: 1) visibility testing, 2) path linking, 3) stroke placement and 4) interior shading. For the stroke placement, the 3D volume texture and multi-texture techniques are used to fast compute ink and moisture information. We also attempt to simulate ink diffusion effect to paint the interiors of the models. As a result, many aesthetically pleasing Chinese-paintings rendered in real-time from 3D models are demonstrated using the proposed method. There are many possible future work can be further explored based on our current work. For example, we plan to consider motion issue in NPR such as sparrow jumping or waving its wings created by using bone and skin deformation techniques. In this situation, we need to consider the stroke coherence problem. Without treating well this issue, it is very easy to create popping effect during animation or deformation. An-

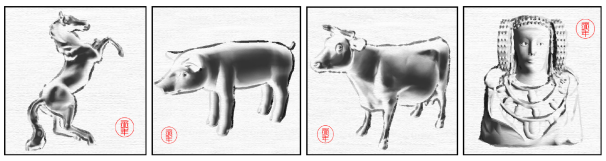

Fig. 7. More rendered results.

other our research direction is to how to express model metamorphosis [13, 14] or human face [15] in Chinese painting style or in western painting style [16].

Acknowledgement. This paper is supported by the National Science Council, Taiwan, Republic of China, under contract No. NSC-93-2213-E-006-026 and NSC-93-2213-E-006-060.

References

1. Strassmann, S., "Hairy brushes," Proc. SIGGRAPH 86, 20(4): 225-232, August 1986.
2. J. Lee, "Simulating oriental black-ink painting," Computer Graphics and Applications, IEEE, vol. 19(3), pp. 74-81, May-June 1999.
3. Q. Guo and T. Kunii, "Modeling the Diffuse Painting of Sumie'," Modeling in Computer Graphics (Proc. IFIP WG5.10), T. Kunii, ed., Springer-Verlag, Tokyo, 1991,pp. 329-338.
4. Q. Zhang, Y. Sato, J. Takahashi, K. Muraoka and N. Chiba, "Simple cellular automation-based simulation of ink behavior and its application to Suibokuga-like 3D rendering of trees," Journal of Visualization and Computer Animation, 1999.
5. Way, D. L., and Shih, Z. C. "The Synthesis of Rock Texture in Chinese Landscape Painting," Computer Graphics Forum, Vol. 20, No. 3, pp. C123-C131, 2001.
6. Way, D. L., Lin, Y. R. and Shih, Z. C. "The Synthesis of Trees Chinese Landscape Painting Using Silhouette and Texture Strokes." Journal of WSCG, Vol. 10, No. 2, pp.499-506, 2002.
7. C. Chan, E. Akleman, and J. Chen, "Two methods for creating Chinese painting," Proceedings.10th Pacific Conference on, October 2002, pp. 403 - 412.
8. N. S.-H. Chu and C.-L. Tai, "An efficient brush model for physically based 3d painting," in Proceedings of 10th Pacific Conference on Computer Graphics and Applications, 2002, 2002, pp. 413-421.
9. Jun-Wei Yeh, Ming Ouhyoung, "Non-Photorealistic Rendering in Chinese Painting of Animals," Journal of System Simulation, Vol. 14, No. 6, 2002, pp. 1220-1224.
10. T. Isenberg, N. Halper, and T. Strothotte, "Stylizing silhouettes at interactive rates: From silhouette edges to silhouette strokes," in Computer Graphics Forum, Proceedings of Eurographics 2002, vol. 21, September 2002, pp. 249-258.
11. http://www.opengl.org
12. http://developer.nvidia.com/page/cg_main.html
13. Tong-Yee Lee, P.H Huang, "Fast and Institutive Polyhedra Morphing Using SMCC Mesh Merging Scheme," IEEE Transactions on Visualization and Computer Graphics, Vol. 9, No. 1, pp. 85-98, 2003.
14. Chao-Hung Lin, Tong-Yee Lee, "Metamorphosis of 3D Polyhedral Models Using Progressive Connectivity Transformations," IEEE Transactions on Visualization and Computer Graphics, Jan./Feb. Issue, Vol. 11, No.1, pp. 2-12, 2005
15. Tong-Yee Lee, Ping-Hsien Lin, Tz-Hsien Yang, "Photo-realistic 3D Head Modeling Using Multi-view Images," in Lecture Notes on Computer Science (LNCS) 3044, Springer-Verlag, pp. 713-720, May 2004.
16. Ming-Te Chi, Tong-Yee Lee, "Stylized and Abstract Painterly Rendering System Using a Multi-Scale Segmented Sphere Hierarchy," to appear in IEEE Transactions on Visualization and Computer Graphics.