Image-based Real-time Hatching of Scene Traveling

Jiajun Bu Wen Yan Chun Chen Mingli Song
College of Computer Science
Zhejiang University
P.R.China 310027, Hangzhou
bjj@zju.edu.cn yanwen@cad.zju.edu.cn {chenc,brooksong}@zju.edu.cn

ABSTRACT

Real-time rendering of a complete 3D scene with hatching strokes is an important direction in NPR field. In this paper, a comprehensive solution is presented to render complicate scenes with pen-and-ink style in real time. With the help of powerful programmable graphics hardware, our real-time system includes many features as hatching, continuous tones, silhouettes, and shadows. We build our approach in image-space, while allowing for the stroke directions and frame coherence.

In our method, various features of pen-and-ink drawings are derived from 3-D information through multi-pass rendering. After synthesis, the desired shading tone is achieved by mapping preprocessed stroke textures to the screen. As a tip for saving texture memory, we prepare a few stroke textures with only certain tones and directions, and compose requisite stroke types in real time. Furthermore, we develop some forms of "indication" to convey the impression of a texture without drawing every single stroke, which makes the result look more natural and art-stylized.

Keywords

Non-photo realistic rendering, real-time hatching, texture indication, G-buffer

1. INTRODUCTION

Pen-and-ink drawing is an important form of pictorial representation [Art77] and an effective way to convey lighting, directions and texture properties. It turns to be a key research branch in non-photorealistic rendering which brings lots of art-like styles. Several features of current GPU-the programmable graphics hardware, can be used to facilitate the art-like real-time rendering, such as large texture capacity, screen texture, multipass rendering, per-pixel process, etc. Currently, many pen-and-ink systems aim to add more features to their renderings while maintaining high frame rate. Most of these systems prefer stroke textures that keep series styles of stroke hatching rather than generating every single stroke in real-time because it is too time-consuming.

In order to generate the pen-and-ink drawing automatically, two types of approaches have been evolved: pen-and-ink drawing in object space and that in image space. The former, called object-based method, attaches stroke textures to polygonal surfaces and render scenes in the traditional rendering pipeline. The latter, called image-based method, parses the scene to get information like tone, silhouette and texture properties and project stroke texture to screen space to achieve different features.

Both these two approaches have merits and demerits. In object-based methods, frame-to-frame coherence can be achieved easily by attaching the stroke texture to scene objects. In this way, strokes can always move with their corresponding objects, and therefore temporal coherence is preserved. However, such methods cannot simulate certain expressive drawing styles conveniently, such as allowing strokes to cross boundaries. Besides, object-based methods require 3D geometry with proper texture coordinates and consequently cannot be applied to images or videos. Moreover, strokes in the screen space are preferred to be independent of the object scale and orientation, namely, uniformly distributed, which is hard to achieve in object-based approaches.

On the other hand, image-based methods overcome many restrictions of object-based methods while introducing some new challenges. They are never fettered by the boundary problem. With

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

image-based technique, strokes can cross polygon edges naturally and be uniformly distributed. However, image-based methods suffer from the undesirable "shower door" effect, which seems that strokes "stick" in screen space independent of object movement. Another problem is that the image-based methods are difficult to control stroke directions to represent scene objects' shapes as ideally as the object-based methods do.

In this paper, we present a novel image-based system that not only preserves the image-based advantages but also adds control to stroke directions as well as reducing "shower door" effect. Our system applies multipass rendering and makes plenty use of G-buffer to achieve desired drawing features. The problem of "stroke direction" mentioned above is tackled by introducing UV map, which stores the pairs of principal directions of each pixel. UV map, produced in rendering pass, is conveyed as a kind of G-buffer for following step to adjust the stroke direction. The direction angle in a certain range is resolved into a same degree in order to avoid too dense variation which usually causes strokes to be severed. Also, by adoption of directions, the "shower door" problem can be weakened because strokes always changes with object's shape and orientation according to view transformation. Moreover, we bring in the idea of indication to enhance the artistic effect of hatching which can also fortify the tone variation and outlines.

The remainder of this paper is organized as follows. In Section 2, we review previous work. The novel rendering algorithm is described in Section 3. A discussion of the implementation and results follows in Sections 4 and 5. The paper ends with a short conclusion and an outlook on future work.

2. Previous Work

Our work aims to render 3D scenes in real time with stroke textures to represent tone, silhouettes, shadows and etc, while avoiding the monotony stroke direction and spatial discontinuity. Related work includes automatic generation of pen-and-ink illustration and silhouettes from 3D scenes, real-time hatching.

Off-line rendering:
Lots of previous work focused on generating high-quality pen-and-ink drawing in an off-line process. Some aimed to create still images of 3D scenes [Deu00, Sai90, Win94, Win96, Sal97, Elb99, Sou99a, Sou99b, Her00]. Saito and Takahashi [Sai90] post-process the rendered framebuffer and overlay image-space strokes. Winkenbach and Salesin [Win94], and Salisbury et al. [Sal97] introduce prioritized stroke textures, which can indicate both texture and tone. Despite the great advance in processing performance, the sheer stroke-drawing hinder these methods from running in real-time.

Real-time Outlines:
Outlines can be generalized as silhouettes, boundaries and creases. They are widely used in various kinds of NPR drawings. Several previous approaches tried to generate outlines in interactive rate [Elb99, Goo99, Her99, Mar97, Nor00, Ras99, Her00, Ras01]. These works can also be divided into two ways: image-based and object-based. The image-based methods often use rendered images as input and are limited by images' resolution. But they are more efficient, easy to implement and sufficient for most projects. The work of Gooch et al. [Goo99] and Raskar [Ras99] can be a typically representation of image-based approach. Object-based approaches aim to produce more precise outlines which are more suitable for additional processing. Hertzmann and Zorin [Her00] create high-quality silhouettes based on geometric duality. Raskar [Ras01] has developed a hardware-support approach to generate outlines by introducing new polygons with only polygonal vertices as input.

Real-time hatching:
Durand et al. [Dur01] create hatched images from photographs in real-time using hardware acceleration to perform anti-aliased threshold. Markosian et al. [Mar97] introduced a simple hatching style indicative of a light source near the camera, by scattering a few strokes on the surface near (or parallel to) silhouettes. Elber [Elb99] shows how to render line art for parametric surfaces in real time; he renders objects by choosing a fixed density of strokes on the surface. Lake et al. [Lak00] describe an interactive hatching system with stroke coherence in image space.

Freudenberg [Fre01a] builds mipmap texture with uniform lines while maintaining the frame-coherence in blending. Praun et al. [Pra01] suggested tonal art map (TAM) for real-time hatching. This algorithm maintains frame-to-frame coherence using a "stroke nesting property", where strokes on a TAM image appear in all the darker images of the same resolution and in the higher resolution images of the same tone. Praun et al. [Pra02] improved their work by providing control of tone to avoid blending and aliasing artifacts in original system. Fung [Fun03] extends the TAM approach and enable generating TAM images of arbitrary textures. Freudenberg's approach [Fre01b, Fre04] express different halftone patterns with stroke textures and implement fast halftone rendering with pixel shading hardware.

3. OUR NOVEL APPROACH

In our pen-and-ink drawing system, the basic features are achieved, like desired tone, outlines and shadows. In the meanwhile, we try to describe the shape of scene objects by fine control of the stroke direction

and give the viewer some indications to avoid monotony. The tone aims at simulating the light shading and can be achieved by composition of TAM (Tonal Art Map) [Pra01] which are generally texture groups of continuous tone and resolution. Because our process is image-based, we need only the finest resolution of the TAM textures. With the aid of G-buffer, 3-D scenes are rendered through several passes to acquire different features. And the indication is generated based on the tip that areas around edges should be paid more ink than the broad unanimous areas which are usually the center of a large surface.

Tone Expression

The terms "value" and "tone" have the same meaning when referred to the amount of visible light reflected from a point to the viewer's eyes. We can use arbitrary lighting model to evaluate the tone at each pixel. The desired tone can be achieved by controlling the ratio of black ink and white paper and we need construct a sequence of hatching images with discrete tones. Abiding by the nesting principle of TAM, our stroke texture series consists of 6 levels of tone in single resolution, as illustrated in Figure 1. This ensures that strokes in the image of lighter tone will also appear in the image of darker tone and therefore temporal coherence (continuity in tone) can be maintained. Actually we only prepare three textures of lighter tone and compose the remainder hatching textures with preceding ones. This strategy can lend economy to texture memory and flexibility to direction control. Since automatic generation of TAM textures is not the goal of this paper, we just use the finest level of TAM textures from Praun's work [Pra01]. In final rendering process, each pixel will be sampled with blend between two textures of consecutive tones.

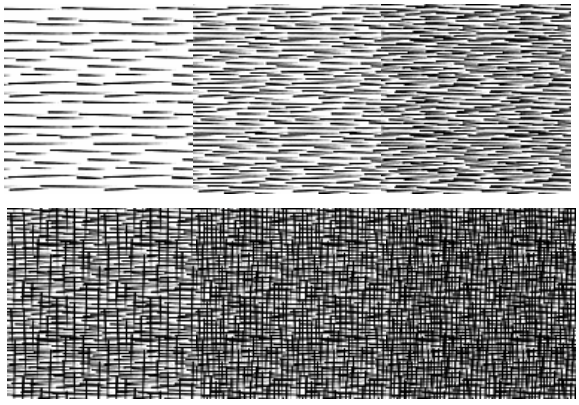


Figure 1. Tonal Map Series. The nether 3 are composed from upper 3 with different rotation angle. Here are only the upright directions.

G-buffer Utility

G-buffer was put forward by Saito and Takahashi [Sai90]. It is a conception that the 2D data structure can be coded to store 3D information. In the rendering process, certain 3D information can be stored into a texture image, each pixel of which should record relevant 3D surface's certain kind of value. The powerful pixel shader can do us a favor to store desired 3D information in RGBA color channels. G-buffer can include many kinds of 3D properties, such as Z buffer, normal buffer, material buffer, shadow buffer, and etc. The following buffers are employed in our system:

- Z buffer: each pixel's corresponding vertex's depth information. Need one float storage.
- Normal buffer: each pixel's corresponding vertex's normal value. Need three-float storage.
- Shadow buffer: shadow map in the view from the light point for each light. The depth value from light point to nearest pixel. Need one float storage.
- UV buffer: principal directions of each pixel recorded as two rotation angle from x-axis or their sine/cosine value. Need at least two-float storage.

To produce these buffers, the rendering process go through several passes and in some cases, two kinds of buffers can be acquired in single pass. Accordingly, two buffers can share one texture image as storage in separate color components. The detail of the rendering process will be mentioned in the Section 4.

Using the G-buffer mentioned above, we can easily generate various features for pen-and-ink drawings. All these features are well developed for many different methods both in image space and object space. For the sake of real-time rendering, we adopt the most effective approaches that can be implemented easily. The following will briefly review these classical methods.

Outline: For polygonal meshes, outlines at silhouettes, ridges, valleys are the key edges for their shape. The silhouette edges consist of edges that are shared by both back-facing polygons and front-facing polygons. A ridge is a crease edge whose dihedral angle between adjacent polygons is less than a threshold while a valley's dihedral the angle is greater than a threshold.

To generate the outline, normal and depth buffers are transferred to a pixel shader which is a filter for postprocessing. For this filter, all we need is to sample pixels around our current pixel and make a comparison between the calculated value and threshold. When detecting silhouettes, it just needs to

check whether the z value of each sampler's normal is of the same sign. When detecting creases, we apply Sobel filter as our convolution matrix to the samplers. Let $I(x, y)$ be a grayscale image. Edge images of I are computed by discrete 2D convolution:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$I_x(x,y) = I(x,y) \otimes S_x, \quad I_y(x,y) = I(x,y) \otimes S_y$$

The optimistic estimation of gradient can be measured as:

$$I_{mag}(x,y) = \sqrt{I_x^2(x,y) + I_y^2(x,y)}$$

Actually, when implemented, even I_x+I_y will be acceptable. And the edge can be detected by the equation below:

$$Edge(x,y) = \begin{cases} 1 & I_{mag}(x,y) \geq T \\ 0 & I_{mag}(x,y) < T \end{cases}$$

Shadow: The approach of shadow map [Seg92] is very suitable for our G-buffer framework. It is a two-pass algorithm. In the first pass, depth map or shadow map can be rendered as mask in view of light's point and naturally stored in G-buffer. Each pixel of the map records the depth of closest pixel to the light. In the second pass, the scene will be rendered from the eye's point while each rasterized fragment's XYZ position is determined relative to the light in order to match the frustum used to create the shadow map. If fragment's light position Z is greater than the depth value at light position XY in the depth map, this fragment is shadowed. Shadow map is stored as a kind of G-buffer.

Indication

The idea of adding some indication is most inspired by the work of Winkenbach who listed indication as one of pen-and-ink drawing's principles. In merits, "Indication" lends economy to an illustration, and also makes an illustration more impressive by engaging the imagination of the viewer rather than revealing everything. Winkenbach aimed to produce static drawings, so he can employ lots of user interaction and make excellent effect. But in real-time implementation we can only introduce very little indication by automatically composing indication map. Our implementation involves two measures.

The indication fields should be depicted with fewer details, and therefore the basic principle is to generate indication in the area far from outlines. Our first measure borrows the idea "field" [Bei92] to evaluate indication. In Winkenbach's work, user distributes some "detail segments" along silhouettes on prepared image and calculates a field $w_1(x, y)$ for each pixel according to these segments. In this calculation, l indicates the nearest "detail segment" labeled by user and a_1, b_1, c_1 are used to adjust the weight.

$$w_1(x,y) = (a_1 + b_1 \times dis((x,y),l))^{-c_1}$$

Without interactively placing "detail segments", we have to generate a rough field map for every frame. An easy approach to achieve the fast distributing requirement is to produce a "General Field Map (GFM)", and attach this GFM to each plane as texture. GFM should be preprocessed according to $w_1(x, y)$, simply with its edges indicating l. The field maps will transform with the scene objects and the indication map of each frame would be composed by them. Figure 3 can illustrate the main idea.

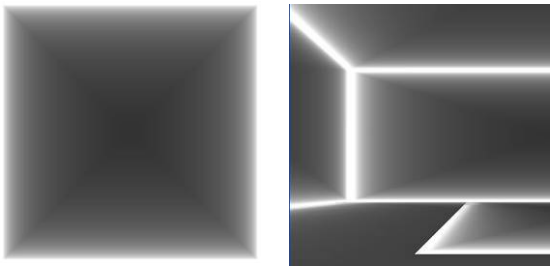


Figure 3: (a) General Field Map: produced according to $w_1(x, y)$, and l indicates the edge of the square map. (b) Composed Field map: Indication appears in dark areas while the lighter the more details to appear.

Without loss of generality, the square GFM can apply to various polygons by adjusting their texture coordinates. This method can work pretty well when the scene is built up by planes connected edge by edge. However, when it comes to curving surface and the intersection of two surfaces, this method seems a little helpless. The area of intersection can not be totally weighted with detail by indication of GFM. In implementation, we can avoid intersection of planes by incising the big plane into small pieces at the place of intersection. But it is not a generic approach and lead to a heavy workload on modeling.

Our second measure deals with the indication by silhouette. In normal depth map, at each pixel, we uniformly sample nearby pixels around it with a certain distance R. Then the difference between central and nearby sampling pixels of normal and depth map can be calculated and compared to a threshold to detect whether the two pixels are separated by silhouettes. And the field of the central pixel can be weighted by:

$$w_2(x,y) = (a_2 + b_2 \times field(N(x,y,R)))^{-c_2}$$

$$field(n) = \alpha / (n^{\beta} + \lambda)$$

$N(x, y, R)$ equals the number of the nearby pixels apart from central pixel by silhouette, where R is the radius of sampled pixel from center. And distance of the current central pixel from silhouette can be weighted by a *field(n)*. This approach can approximate field weight limited to the area of distance R from silhouettes. The precision of field can be enhanced by increasing the sampling pixel number whereas could be very time consuming. Then we have to weigh a tradeoff between precision and running efficiency.

By integrating the above two measures, we can cover most of the high-weight fields (detailed) and approximate the indication area successfully. The overall field weight can be evaluated by:

$$w(x,y) = a \times w_1(x,y) + b \times w_2(x,y)$$

Direction Control

In the ideal state, strokes are supposed to follow the principal direction in each pixel. However, when scaled or rotated, a polygonal plane will be distorted and UV direction within different areas of the plane can never be kept in parallel. This is not a great problem in object-space because the stroke textures could be attached to facets along UV direction and these textures are being transformed with objects. Nevertheless, strokes can be seriously compressed and aliased when facets are nearly perpendicular to the screen. Even TAM textures in object-based methods can not perfectly settle this problem.

In image-space, we only need to concentrate on the UV direction problem as the strokes can be uniformly distributed through the whole screen space and they will not be distorted in size. In our algorithm, UV direction has been designated in the object space. When rasterized, in each pixel, UV direction will be transformed to the angle deviated from the X axis and be stored in G-buffer. Then in texture mapping, each pixel's coordinates should be applied to rotation matrix according to the UV map. In the UV coordinates' transformation, serious aliasing occurs as the stroke direction changes continuously. To get local coherence, we set a threshold T so that the direction can only change in leap. This can be simply implemented as:

$$angle = sign(angle) \times \lfloor abs(angle) / T \rfloor \times T$$

The process is demonstrated in the Figure 4. To show the direction transformation clearly, only one tone has been used in the following illustration. And the result image with direction and tone control is shown in Figure 5

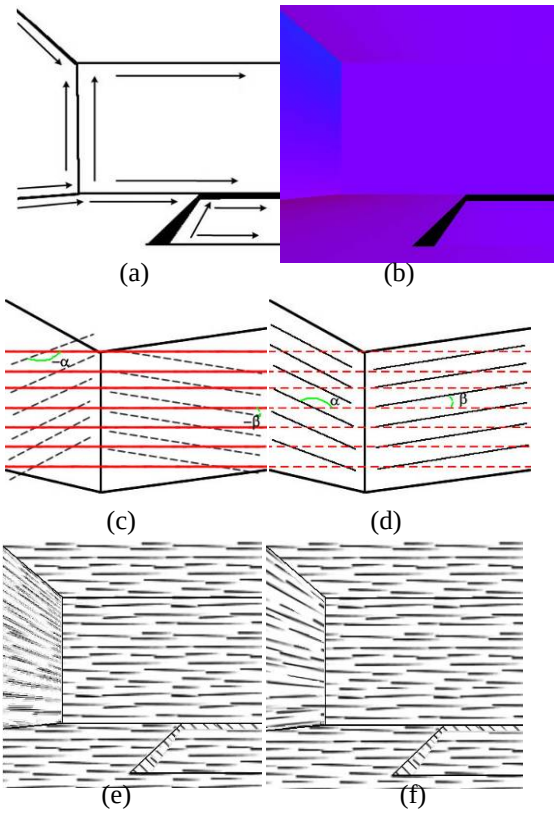


Figure 4: Stroke direction control. (a) UV direction vectors attached to each plane transform with coordinates transformation. (b) Record UV rotation angle to G-buffer. (c)&(d) adjust the texture coordinates to sample stroke texture. Red lines indicate stroke textures direction. Black lines indicate UV directions. To be noticed, texture coordinates should be rotated an inverse angle of UV direction. (e) Stroke directions without the threshold subsection. (f) Stroke directions with threshold subsection.

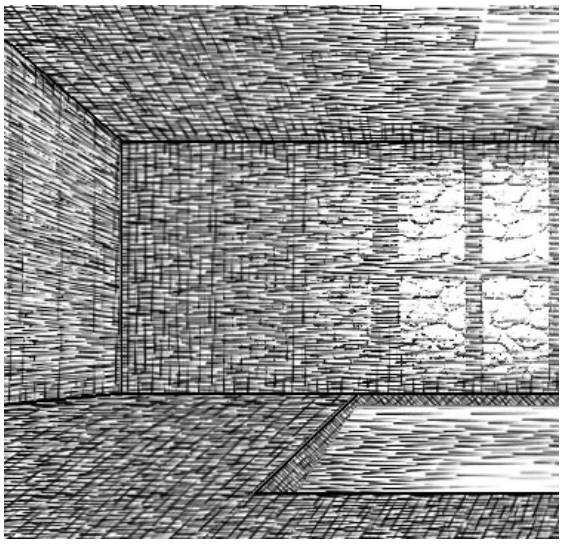


Figure 5: Hatching with stroke direction: This figure also includes tones and outlines.

4. IMPLEMENTATION

Our image-based hatching system takes advantage of GPU hardware ability, and is able to render scenes in a few passes to achieve multiple features in real-time. Optionally, we could skip some of features to pursuit high frame rate or to make one of them more distinct. The choice lies on the user. Figure 6 illustrates the relation between G-buffer and features. The composed images are illustrated in Figure 8.

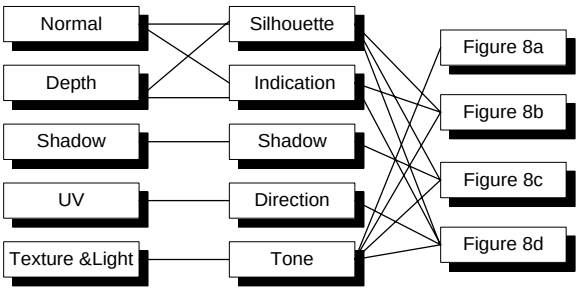


Figure 6: Relation between G-buffer and features

The implementation is based on C++ and OpenGL. The framework runs with a 1.2 GHz CPU and Radeon 9550 graphics card which can provide us Pixel Shader2.0 functionality. Many tools can be used to record G-buffer attributes as texture images, like "RenderTexture", "PBuffer" and etc. We choose hardware supported "Frame Buffer Object" which is proved to be the fastest. To add all the mentioned features, we need our scenes go through 6 passes. In each pass, special shader is designed for desired rendering purpose. The following framework pipeline (in Figure 7) can illustrate the whole process.

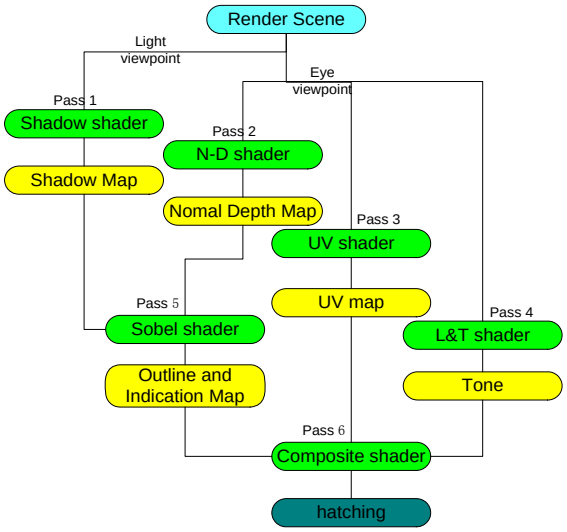


Figure 7: Multipass Rendering Pipeline. Blue component indicates input data. Green ones indicate rendering shaders and yellow ones indicate G-buffer after each pass.

With the six passes, a comprehensive balance strategy is considered in our approach. We try to compress two or more shaders into one pass while avoiding the overload in single shader or possible bottle neck. The shadow buffer can be integrated into the Outline&Indication buffer, each of which occupies one color channel originally, so as to reduce sampling computation in Pass 6. In Pass 2, RGBA channels can not be shared with other buffer because they are fully occupied by Normal&Depth buffer. In Pass 3, direction buffer is stored as (cosU, sinU, cosV, sinV) in RGBA. As for Pass 4, the edge-detection on textures is performed since the textures themselves also have complicate paintings and we would like their edges to be accentuated. And tone value in Pass 4 needs only one channel if the hatching image is in grey level without color texture.

5. CONCLUSION and FUTURE WORK

In this paper, we present a system for image-based hatching that can efficiently render 3-D scenes in real-time by GPU. The power of GPU brings facility to the implementation of our rendering algorithm and makes it possible that lots of features can be integrated into rendering in real-time. The main contribution of this paper is that of simulating the idea of indication in pen-and-ink paintings and stroke direction controlling in image space. Both of them can make the painting more natural, reveal objects' shape and weaken the "shower door" effect to some extent,.

However, our approach of stroke direction control may cause the stroke to be ruptured and lose spatial coherence when directions vary sharply in small area. Therefore, our future work aims to maintain the full coherence while revealing the stroke direction. In addition, our approach to generate indication is not so effective in complicated scenes. The implementation of the "field map" could be improved by using the ability to write texture in pixel shader in up-to-date GPU. Then many of samplings are not necessary when approximating the indication field, which costs lots of computation. Furthermore although the "shower door" effect is reduced in our approach, further effort is needed to eliminate it thoroughly.

6. REFERENCES

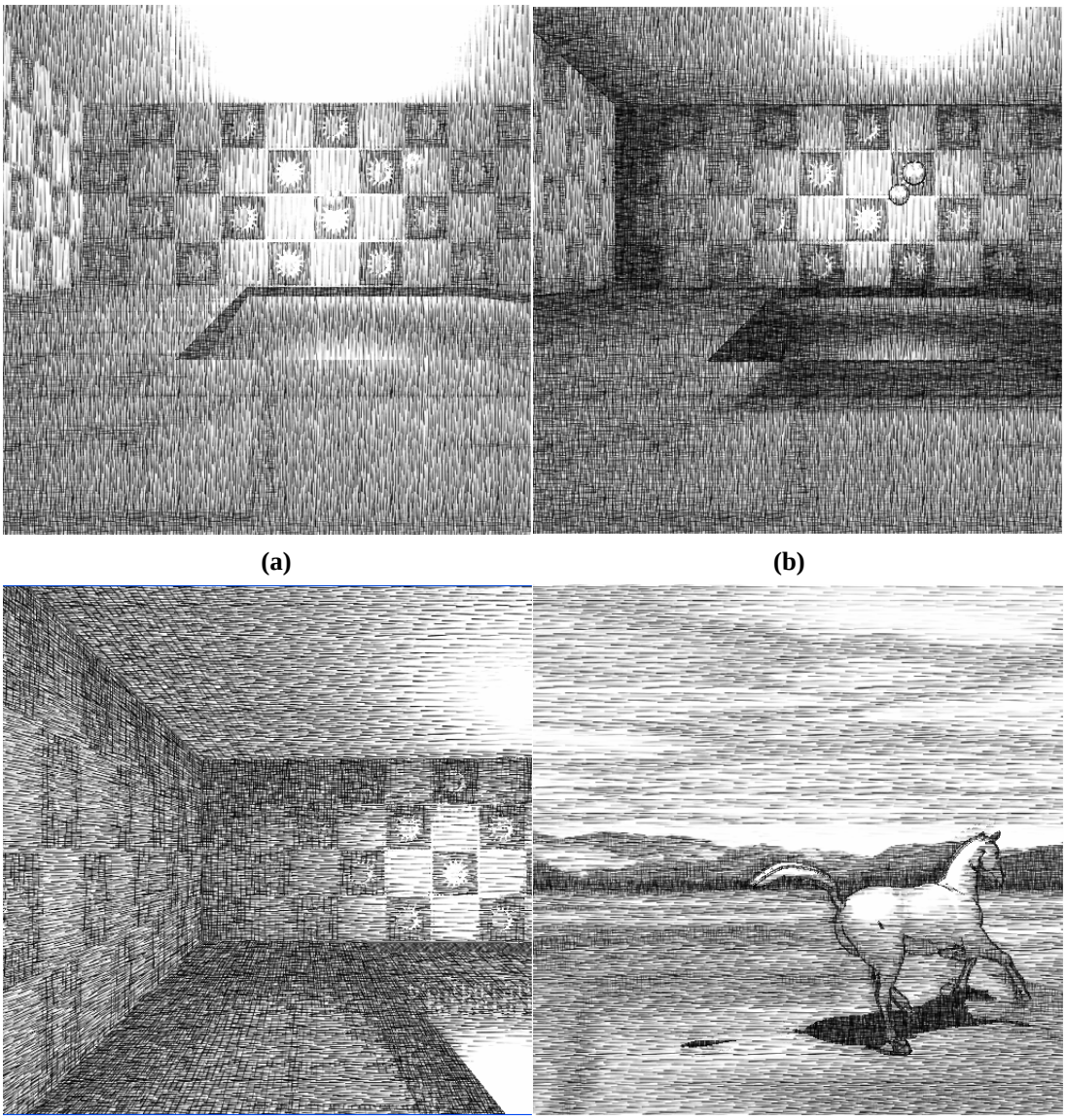
[Art77] Arthur, L. Guptill. Rendering in Pen and Ink. Watson-Guptill Publications, New York, 1977.

[Bei92] Beier, T. and Neely, S. Feature-based image metamorphosis. Proceedings of SIGGRAPH '92 In Computer Graphics 26, 2 (July 1992), 35–42.

[Sai90] Saito, T. and Takahashi, T. Comprehensible Rendering of 3D Shapes. Proceedings of SIGGRAPH 90, pp. 197–206.

[Win94] Winkenbach, G. and Salesin, D.H.

Computer-Generated Pen-and-Ink Illustration. Proceedings of SIGGRAPH 94, Computer Graphics, Annual Conference Series, pp. 91–100.
[Sal97] Salisbury, M.P., Wong, M.T., Hughes, J.F., and Salesin, D.H. Orientable Textures for Image-Based Pen-and-Ink Illustration. Proceedings of SIGGRAPH 97, pp. 401–406.
[Deu00] Deussen, O., and Strothotte, T. Computer-Generated Pen-and-Ink Illustration of Trees. Proceedings of SIGGRAPH 2000, 13–18.
[Elb99] Elber, G. Interactive Line Art Rendering of Freeform Surfaces. Computer Graphics Forum 18, 3 (September 1999), pp. 1–12.
[Sou99a] Sousa, M.C., and Buchanan, J.W. Observational Model of Blenders and Erasers in Computer-Generated Pencil Rendering. Proceedings of Graphics Interface'99, 157 – 166.
[Sou99b] Sousa, M.C., and Buchanan, J. W. Computer-Generated Graphite Pencil Rendering of 3D Polygonal Models. Computer Graphics Forum 18, 3 (September1999), pp. 195–208.
[Win96] Winkenbach, G., and Salesin, D.H. Rendering Parametric Surfaces in Pen and Ink. Proceedings of SIGGRAPH 96, pp. 469–476.
[Her00] Hertzmann, A., and Zorin, D. Illustrating Smooth Surfaces. Proceedings of SIGGRAPH 2000, Computer Graphics, Annual Conference Series, pp. 517–526.
[Goo99] Gooch, B., Sloan, P.-P. J., Gooch, A., Shirley, P., and Riesenfeld, R. Interactive Technical Illustration. 1999 ACM Symposium on Interactive 3D Graphics, pp. 31–38.
[Mar97] Markosian, L., Kowalski, M.A., Trychin, S.J., Bourdev, L.D., Goldstein, D., and Hughes, J.F. Real-Time Nonphotorealistic Rendering. Proceedings of SIGGRAPH 97, pp. 415–420.
[Nor00] Northrup, J.D., and Markosian, L. Artistic Silhouettes: A Hybrid Approach. Proceedings of NPAR 2000, pp. 31–38.
[Ras99] Raskar, R., and Cohen, M. Image Precision Silhouette Edges. 1999 ACM Symposium on Interactive 3D Graphics, pp. 135–140.
[Ras01] Raskar.R Hardware Support for Non-photorealistic Rendering. 2001 Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, pp. 41-47
[Her99] Hertzmann, A. Introduction to 3D Non-Photorealistic Rendering: Silhouettes and Outlines. SIGGRAPH Course Notes. 1999.
[Lak00] Lake, A., Marshall, C., Harris, M., and Blackstein, M. Stylized Rendering Techniques for Scalable Real-Time 3d Animation. Proceedings of NPAR2000, pp.13–20.
[Pra01] Praun, E., Hoppe, H., Webb, M., and Finkelstein, A. Real-Time Hatching. Proceedings of SIGGRAPH 2001, Computer Graphics, Annual Conference Series, pp. 579-584.
[Pra02] Webb, M., Praun, E., Finkelstein. A., and Hoppe, H. Fine Tone Control in Hardware Hatching. Proceedings of NPAR 2002. pp. 53-58
[Fre01a] Freudenberg, B., Masuch, M. and Strothotte, T. Walk-Through Illustrations: Frame-Coherent Pen-and-Ink Style in a Game Engine. Proceedings of Eurographics 2001, pp. 184-191.
[Fun03] Fung, J., Veryovka, O. Pen-and-ink textures for real-time rendering. Proceedings of Graphics Interface 2003. pp. 131-138
[Fre04] Freudenberg, B., Masuch, M. and Strothotte, T. Real-Time Halftoning: Fast and Simple Stylized Shading. Game Programming Gems 4, Charles River Media, 2004.
[Fre01b] Freudenberg, B. Real-Time Stroke Textures. SIGGRAPH 2001 Conference Abstracts and Applications, pp. 252.
[Seg92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, Paul Haeberli: Fast shadows and lighting effects using texture mapping. SIGGRAPH 1992, pp.249-252.
[Dur01] Durand, F., Ostromoukhov, V., Miller, M., Duranleau, F. and Dorsey, J. Decoupling Strokes and High Level Attributes for Interactive Traditional Drawing. Proceedings of Eurographics Rendering Workshop 2001, pp. 71-82.



(a) **(b)**

(c) **(d)**

Figure 8: Different combination of features