2011 12th International Conference on Computer-Aided Design and Computer Graphics

IdiotPencil: An Interactive System for Generating Pencil Drawings From 3D Polygonal Models

Ning WANG, Bao-Gang HU
LIAMA/NLPR, Institute of Automation, Chinese Academy of Sciences, 100190 Beijing, China
ningwang711@gmail.com, hubg@nlpr.ia.ac.cn

Abstract

This paper describes an interactive system, called "IdiotPencil", that enables users to design pencil drawings directly from 3D polygonal models. The potential users of the system are professional designers or non-professional users. The system will automatically generates feature strokes, and provides fast and easy-to-use tools for hatching stroke design. Users only need to specify hatching carriers according to their experiences or intentions. The hatching tools can perceive hatching regions and arrange hatching strokes automatically. In this way, the system allocates most of manual, yet tedious works to the computer while retaining aesthetic control for users. Compared with the existing manual pencil drawing method, the system can speed up the user's drawing and reduce their drawing skill requirements. Several examples of drawings are given by the IdiotPencil system. The compelling features are obtained from viewpoints of both handcrafted effect and high efficiency when using the computer-aided drawing system.

1. Introduction

Although photorealistic rendering can give us a realistic representation of message, sometimes it lacks the ability of omitting extraneous details and emphasizing on the most important information to us. Non-photorealistic rendering (NPR) is an abstract representation of objects. It reflects the understanding of information by human brain and sometimes can convey object information, personal cognition or feelings more efficiently. Until now there are many techniques to simulate physical materials used for non-photorealistic rendering, for example pen-and-ink [1] [2] [3], graphite pencil [4]–[7], wax crayons [8], charcoal [9] [10] and watercolor [11].

This paper describes a pencil drawing system which allows non-professional users generating pencil strokes from 3D polygonal models. Figure 1 is the overall architecture of IdiotPencil drawing system. All the pencil strokes in drawing picture can be roughly divided into two categories: feature strokes and hatching strokes. IdiotPencil will generate paths of these two kinds of strokes separately, and then convert paths to strokes for rendering. Although IdiotPencil can also automatically generate all the feature strokes and hatching strokes, in this paper we mainly focus on the interactive way rather than the automatic way. Since we followed the philosophy of Kalnins [12] and Durand [13] that we do not intend to develop a system to replace the work of artist, but to provide fast and easy-to-use tools for them to create vivid pencil drawings. In fact, our ultimate goal is to produce a system that provide an automated way to place strokes but that also allows users to interactively make changes afterwards and add new strokes. Such a system is often favorite by users, as mentioned by Isenberg [14]. In this paper, we discuss tools that allow users add strokes by themselves, the automatic generation and edit method will be described in another paper. In Figure 1, we divide IdiotPencil system into three modules: feature path module, hatching path module and stroke generation & rendering module, and we will discuss these modules in Section 3.1, Section 3.2 and Section 3.3-3.5 respectively.

2. Related Works

Since there are too many non-photorealistic rendering related works, here we only involve those related to pencil modeling and rendering. Early pencil drawing works begin with the observation of the microstructure of drawing paper [4] [5] [6]. They are mainly focused on simulation of pencil drawing materials. For example, Takagi et al [4] developed a volumetric modeling method to produce images of colored pencil drawing. Their model consists of three sub-models: the microstructure of paper, pigment distribution on paper and pigment redistribution. Sousa and Buchanan's observational model [5] [6] is an accurate quantitative model. They modeled various drawing tools (pencils of various hardness, blenders and erasers) interacting with paper.

Another pencil rendering technique developed by Lee et al [15] is based on texture mapping. They mapped pencil stroke textures according to principal curvature directions on geometric models. Since texture mapping is a very mature technology on current graphics pipeline, their method can get a real-time rendering result. The only shortcoming of their generated result is that the pictures look uniform (or even) and lack of hand-drawn effect.

978-0-7695-4497-7/11 $26.00 © 2011 IEEE
DOI 10.1109/CAD/Graphics.2011.50

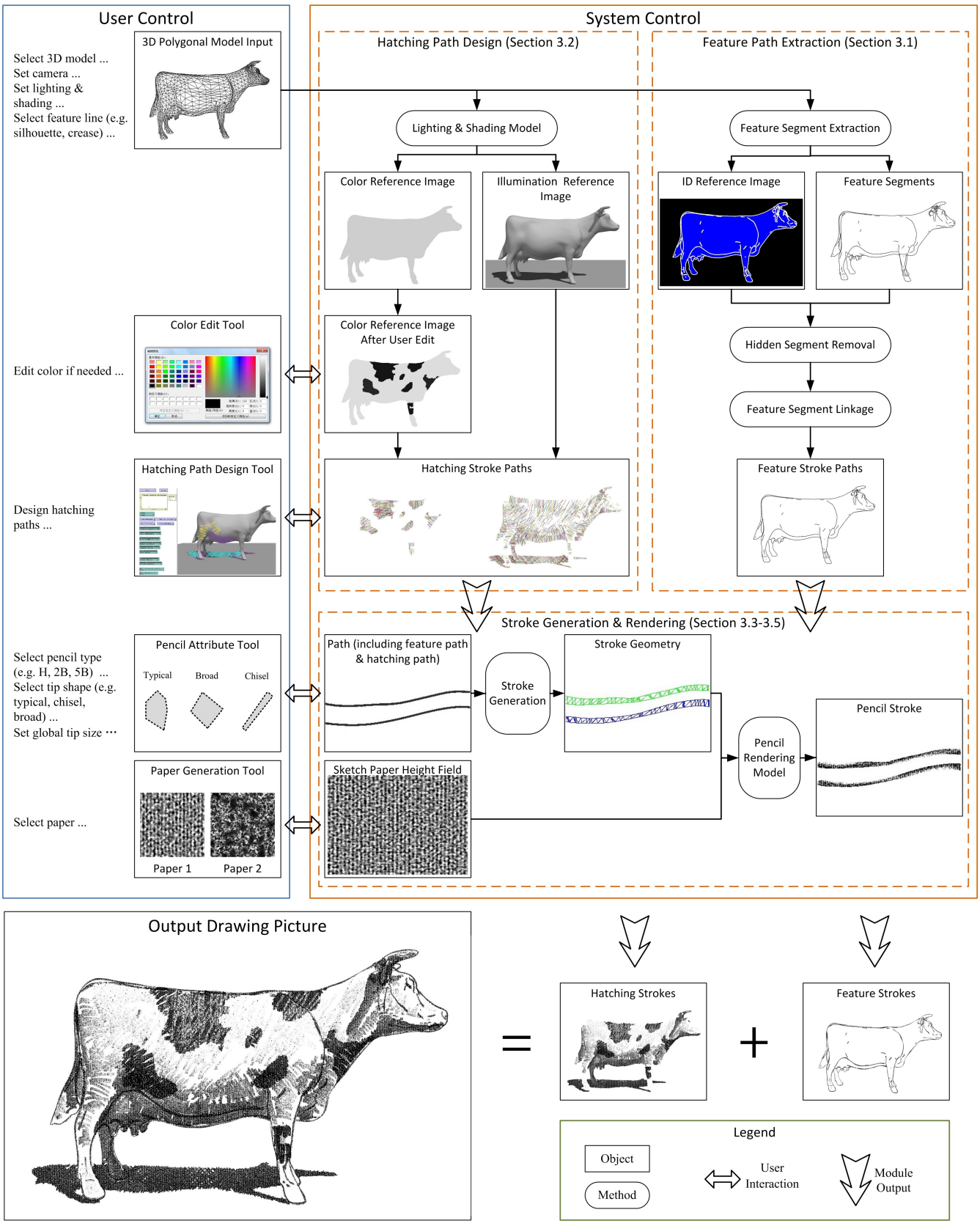


Figure 1. Schematic illustration of IdiotPencil system

Almeraj et al [7] developed a system that can synthesize pencil lines drawn by humans. Their technique is based on the observations of human-drawing pencil lines obtained through user studies. They combine texture synthesis with a physically based model of human arm movement to generate pencil paths and textures.

The last kind of pencil rendering works use 2D image as input [16]–[19]. They are all based on a texture vector field visualization technique called line integral convolution [20].

IdiotPencil is a stroke based pencil rendering system, similar with [5] [6] [7]. Compared with texture mapping based method [15] and image based method [16]–[19]. The stroke based method is faithful to real pencil drawing process, and is easy to edit and modify, thus has higher practical value.

3. The Pencil Drawing System

Currently IdiotPencil system use 3D polygonal mesh as input, but most methods can also be used for image. For image, we can use image based edge detection and edge tracing techniques to replace feature path extraction from 3D models. The input image itself is the combination of color reference image and illumination reference image.

3.1. Feature Path Extraction

The first step in the system is extracting relevant feature paths from the mesh. We use silhouette [21], suggestive contour [22], [23], crease and boundary edge here, same as method of Grabli [24]. Although there are many other feature lines such as ridges, valleys [25] and apparent ridges [26], we find that silhouette, suggestive contour, crease and boundary edge are enough for our application. Excessive feature lines will not lead to better visual effects. On the contrary, in some cases it will make the result even worse. However, as mentioned before, IdiotPencil is a stroke based system, and stroke is easy for edit and modification. In the future works we can let the system generate more feature paths and let users delete or modify unsatisfied part of generated paths.

Since feature paths are computed on triangles of input mesh, until now they are set of segments. During rendering the z-buffer loses depth information of original triangle mesh, so we need do hidden segment removal manually. We use *ID reference image* method [27] [28] since it takes full advantage of the hardware performance of z-buffer. Moreover, by utilizing the feature of geometry shader, feature segments generation and hidden segments removal can be totally implemented in graphics processing unit. After hidden segment removal, we link feature segments to form feature paths.

3.2. Designing Hatching Path

Drawing pictures with hatching strokes can simultaneously convey illumination, color, material and reveal shape. In the system, hatching paths can be designed from *color reference image* and *illumination reference image*. They correspond to the *intrinsic* and *extrinsic* properties of objects respectively [29]. Discriminating intrinsic property and extrinsic property in the system will facilitate user editing color reference image or changing illumination model (e.g. [30] [31]) independently. This discrimination strategy has also been well represented in Dodson's book [32]. There are some key elements for hatching design. They are hatching type, hatching region and shape, hatching direction and hatching tone. We will describe these key elements in the following.

3.2.1. Shape Consciousness. The artists seem to prefer to understand drawing object by shape. They split the scene into regions of various sizes, according to lighting, color, material and object. As depicted by Edouard Manet: "there are no lines in nature, only areas of color, one against another [32]." Some works relying image as input have already followed that idea [17] [18]. They use image segmentation techniques to divide the scene into various regions and apply pencil strokes accordingly. In this paper, we also treat the scene as regions, and design hatching paths according region shape formed by illumination reference image and color reference image, as can be seen in Figure 1. Perceiving shape will be one of the characteristics of our hatching tool. We will depict this in Section 3.2.3.

3.2.2. Discussion on Hatching Direction Preference. Many works relying 3D models as input usually use principal direction as the mark direction [33] [21] [15]. These literatures have already illustrated that lines in the principal curvature directions may communicate surface shape better than lines in other directions. Moreover, principal directions have the quality of geometric invariance, which is a good characteristic for real-time animation. However, hatching in principal curvature direction is not general in pencil drawings, as can be seen in [34] [32]. According to our observation, the artist's most favorite hatching direction is in the direction of top-right and bottom-left. One of reasons is possibly that human anatomical system has the maximum range of movements in this direction [35]. Meulenbroek and Thomassen [36] studied stroke direction preferences from the viewpoint of Ergonomics. They find that people avoid using orthogonal movement directions when they are unable to monitor their movements visually. This viewpoint is particularly important for drawing since artists usually use a skill called

blind drawing [32], which means that during drawing the artist's eyes remain on the subject, meanwhile their hand do not stop. Meulenbroek and Thomassen also noted that people prefer to produce pen jumps in oblique directions, while pen jumps are common in hatching design. Durand [29] pointed that picture space can provide more flexibility and fit better the mental process of picture production. Considering the complexity of hatching direction, here we use simple interaction to let user choose their favorite hatching direction. IdiotPencil system will handle other tedious works such as hatching path arrangement.

3.2.3. Interactive Hatching Tool. Durand [29] pointed that producing a picture is essentially an *optimization process*, which is defined by an objective function related to the user's given purpose. Durand also recognize that this optimization problem should most of the time be solved by the user, and the user needs specific tools for efficient interaction. IdiotPencil system provides several hatching tools for massive designing pencil strokes on reference images. They are called *parallel curve hatching*, *wave hatching* and *cross hatching*, as shown in Figure 2. With parameter control, parallel curve hatching can be degenerated to *parallel line hatching*, and wave hatching can be degenerated to *zigzag hatching*. We also provide *free hatching* [12] for user to draw arbitrary strokes.

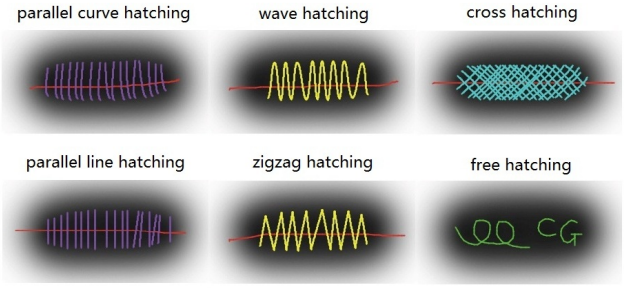


Figure 2. Types of hatching lines

We use *hatching carriers* (similar to stroke carriers [13]) as the supporter of hatching paths, which is marked as red lines in Figure 2. During drawing, the user only needs to select a hatching tool, and drawing hatching carriers on a certain region with specified direction. The hatching tool will automatically get tonal value from reference image and arrange hatching path for user. The algorithm implementation is as follows: first the hatching carriers will record the pixel color it passed and compute the median color $c_m \in [0,1]$. Then we use some particles $p_1, p_2, \ldots, p_i, p_j, \ldots, p_n$ as control points and initial them on hatching carriers. These control particles have several properties such as velocity (v_k^x, v_k^y) and initial color $c_k^0 \in [0,1]$, where $k = 1, 2, \ldots, n$. The distance between p_i and p_j is $\delta_d^{(c_i^0+c_j^0)/2}$, where δ_d is a density parameter in pixels. The control particles can walk on the reference image with rules according to their hatching type (see white arrow and path in Figure 3). We let control particles can only live in the region where $|c_k^0 - c_m| < \delta_c$, where δ_c is the color difference in this region and can be adjusted by the user. In such way we keep the hatching only be processed in the main color region that stroke carrier covered. δ_c also relates to strength of soft edge and hard edge (see interpretation in [32]). Compared with previous NPR method [37] [17] [18], our hatching tools do not need edge detection or image segmentation in advance, but can also express the region shape and preserve feature lines. Finally, we generate hatching path from control particle using Bézier curve interpolation or Catmull-Rom spline interpolation.

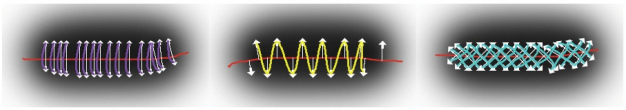


Figure 3. Walk path of control particle (white line) and corresponding hatching style

3.3. Generating Strokes From Paths

Stroke has its own geometry and properties, therefore it is suitable for being the basic elements of drawing pictures. Pictures generated by strokes can be output at any quality. Many drawing materials can also be simulated on strokes.

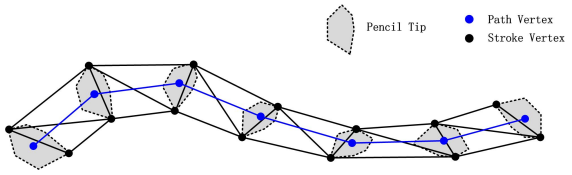


Figure 4. Pencil stroke generation

In the system, the stroke quadrilateral is generated according to stroke path and pencil tip shape, as shown in Figure 4. To avoid long strokes, stroke path could be split at the maximum curvature vertex or the maximum length vertex. Pencil tip shapes are defined by a series of standard shapes, such as typical, chisel and broad, as used by Sousa [6]. User choice pencil tip shape and size according to their wishes. The tip size can also be affected by the geometric details of input mesh, since we know that people usually use thinner pencil to draw object details. We also add a factor f_p to represent the pressure on tip. Thus the tip size S can be

formulated as:

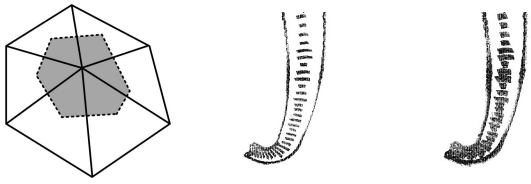
$$S = s_t \times f_g \times f_p \tag{1}$$

where s_t is the user selected global tip size. f_g is the geometric detail factor, which is defined as:

$$f_g = \max(1 - \varphi^{\sqrt{A_v}}, 0.5) \tag{2}$$

where A_v is the one-ring Voronoi area of input mesh vertex, as shown in Figure 5(a). $0 < \varphi < 1$ is a control factor.

The effect of f_g can be seen in Figure 5(b) and Figure 5(c). In these two figures, all the parameters are the same with Figure 6, except that in Figure 5(c) we let f_g always be 1. We can see that without geometric detail control, pencil strokes in details region will be blurred.



(a) (b) (c)

Figure 5. Pencil thickness is adjusted by geometry details

Some special techniques are also used. For feature strokes, we rotate the pencil tip along the stroke path and change the pressure distribution on it to simulate the variety of pencil strokes. For parallel hatching strokes, we attenuate the pressure gradually from start point to end point, thus the parallel strokes will have a start dot and taper at end point. These will make strokes more vivid to achieve handcrafted look.

3.4. Sketch Paper Generation

Drawing on different types of paper will cause different drawing effects. Here we take the snow mountain laid paper (www.xueshanpaper.com) as an example. One feature of this brand of sketch paper is having vertical linear stripe after drawing, as can be seen in the result section. We use Perlin noise [38] to simulate the height field of sketch paper, similar to Curtis's method [11].

3.5. Rendering

Our pencil rendering model is based on Sousa's graphite pencil model [6]. In their model, the smallest element of the paper's roughness is called *grain*, which is defined by four adjacent paper heights. The grain based structure is suitable for implementing the model in the pixel shader of graphic processing unit. We pack the paper height, amount of pencil ingredients (consist of graphite, clay and wax) into texture's R, G, B, A component respectively. During drawing, we iteratively update packed texture according to Sousa's physical pencil models. The shading of pencil drawing can be computed from the graphite content in packed texture.

4. Results

Several pencil drawing pictures generated by IdiotPencil can be seen from Figure 6 to Figure 11. All the pictures are drawn by unprofessional users, and all render targets are 2048×2048 pixels. We list some statistical data in Table 1. All the feature strokes are generated automatically in a few seconds, hence almost all of the time is cost by hatching path design. The drawing speed with our hatching tool is about 5-11 strokes per second, which is much faster than artificial hatching speed.

A simple pencil sketch drawing is shown in Figure 6, with the input 3D model on the left. We can clearly see feature strokes and several types of hatching strokes on it. More results are shown in Figure 7-8. We can see that the hatching tool can handle both soft edge and hard edge nicely. In Figure 9, we designed zebra stripes from the color reference image, which is projected from texture color of input model. If the input model has no texture with it, we can also edit the color reference image to add color information manually. Editing color in color reference image is much easier than in 3D geometry space. We have given an example of color edit in the cow example in Figure 1. Finally, two colored pencil drawings are given in Figure 10 and 11. From these pictures we can see that IdiotPencil system allows ordinary people to produce vivid pencil drawings.

Table 1. Statistical data of result drawings.

Figure	N_m[1]	N_f[2]	N_h[3]	t_f[4]	t_h[5]
Figure 6	39290	291	777	9.4	1-2
Figure 7	6320	38	2555	5.6	4
Figure 8	3673	1605	4232	8.3	12
Figure 9	8741	150	3280	7.1	12
Figure 10	27884	266	7231	8.1	14
Figure 11	103680	766	2631	23.6	8

[1] Number of triangles in input mesh.
[2] Number of feature strokes.
[3] Number of hatching strokes.
[4] Time cost by automatic feature stroke generation (sec).
[5] Time spent on interactive hatching stroke design (min).

5. Conclusion and Future Work

In this paper we present a pencil drawing system that especially suitable for users without any artistic skills. We

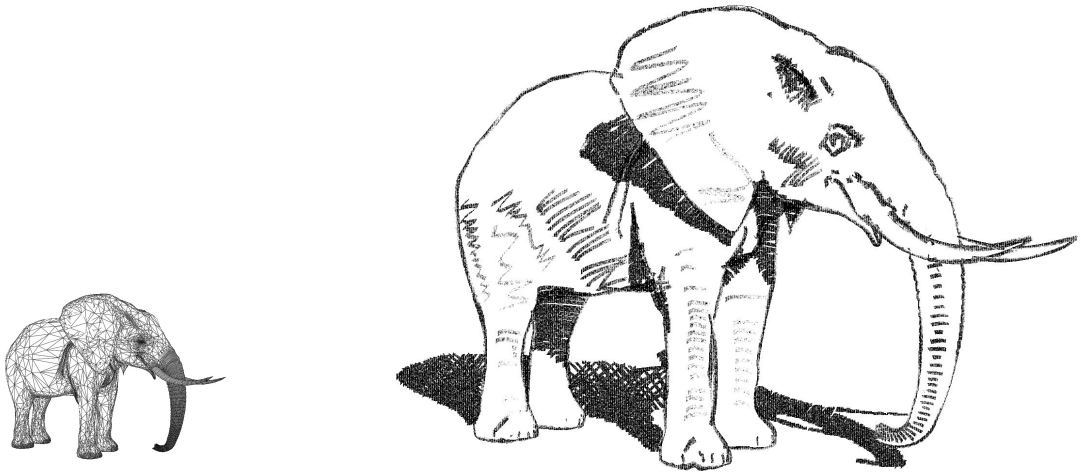


Figure 6. Example 1: simple pencil sketch of an elephant

Figure 7. Example 2: pencil drawing of teapot

let computer take care of most physically-based and tedious works, while try to provide minimal interactions for users to give them aesthetic control thus keep the drawing looks handcrafted. The current system uses mouse as input equipment, but the system is also suitable for input equipment like pressure-sensitive graphics tablet, as used in [9].

As future work, we can add more feature lines into current system under the conditions that an interactive tool is provided to allow user selecting and editing feature lines. Non-photorealistic lighting and shading model can also be considered. For example Gooch model [31] uses both luminance and hue to indicated surface orientation, which is suitable for traditional art style. The current system aims to generate a single pencil drawing. Adding frame-to-frame coherence to current system can be one of future works.

References

[1] G. Winkenbach and D. H. Salesin, "Computer-generated pen-and-ink illustration," in *Proceedings of SIGGRAPH '94*, 1994, pp. 91–100.

[2] G. Winkenbach and D. H. Salesin., "Rendering parametric surfaces in pen and ink," in *Proceedings of SIGGRAPH '96*, 1996, pp. 469–476.

[3] O. Deussen and T. Strothotte, "Computer-generated pen-and-ink illustration of trees," in *Proceedings of SIGGRAPH '00*, 2000, pp. 13–18.

[4] S. Takagi, M. Nakajima, and I. Fujishiro, "Volumetric modeling of colored pencil drawing," in *Proceedings of PG '99*, 1999, pp. 250–254.

[5] M. C. Sousa and J. W. Buchanan, "Computer generated graphite pencil rendering of 3D polygonal models," in *Proceedings of Eurographics' 99*, 1999, pp. 195–208.

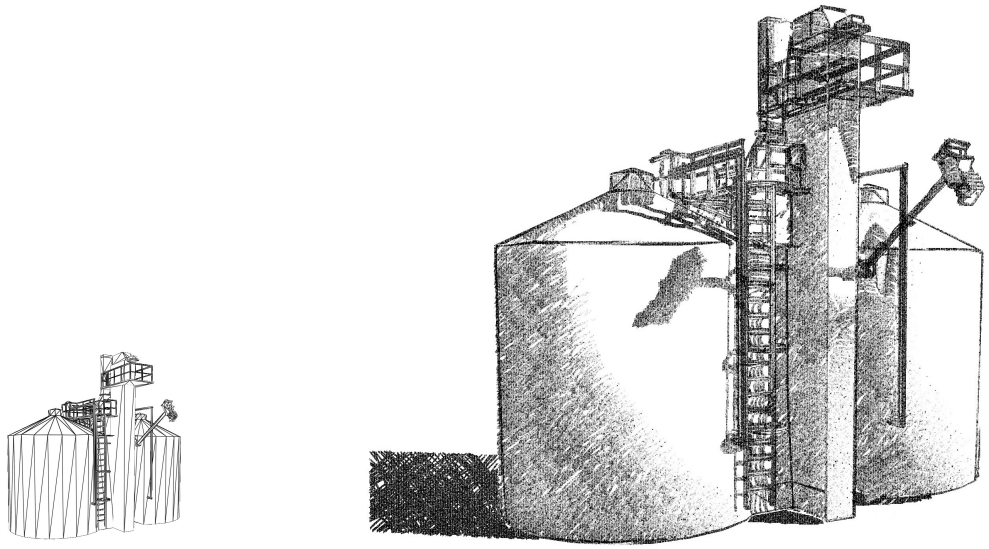


Figure 8. Example 3: pencil drawing of an industrial building

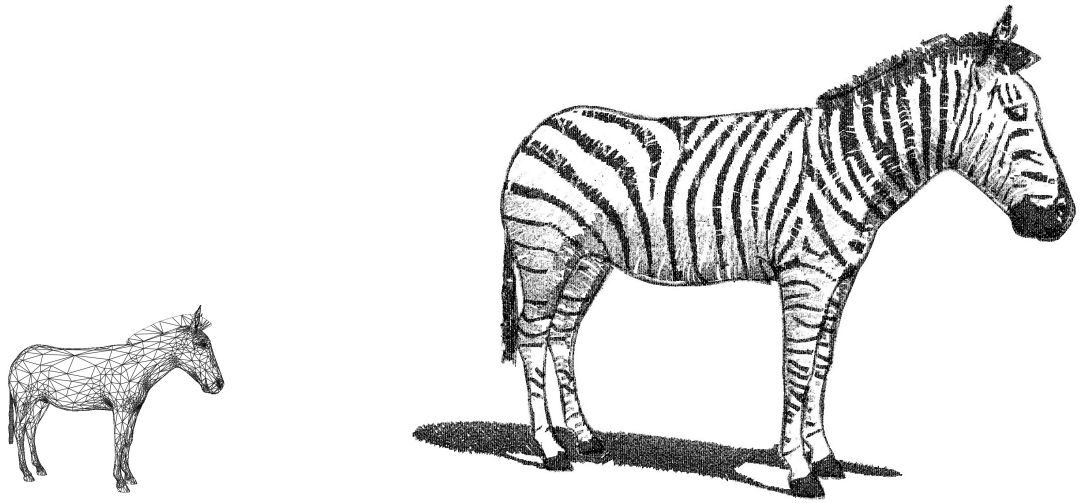


Figure 9. Example 4: pencil drawing of a zebra.

[6] ——, "Observational models of graphite pencil materials," in *Computer Graphics Forum*, 2000, pp. 27–49.

[7] Z. Almeraj, B. Wyvill, T. Isenberg, A. A. Gooch, and R. Guy, "Automatically mimicking unique hand-drawn pencil lines," *Computers & Graphics*, vol. 33, pp. 496–508, 2009.

[8] D. Rudolf, D. Mould, and E. Neufeld, "Simulating wax crayons," in *Proceedings of PG '03*, 2003, pp. 163–172.

[9] T. W. Bleser, J. L. Sibert, and J. P. McGee, "Charcoal sketching: returning control to the artist," *ACM Transactions on Graphics*, vol. 7, no. 1, pp. 76–81, 1988.

[10] A. Majumder and M. Gopi, "Hardware accelerated real time charcoal rendering," in *Proceedings of NPAR '02*, 2002, pp. 59–66.

[11] C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischer, and D. H. Salesin, "Computer-generated watercolor," in *Proceedings of SIGGRAPH '97*, 1997, pp. 421–430.

[12] R. D. Kalnins, L. Markosian, B. J. Meier, M. A. Kowalski, and J. C. Lee, "WYSIWYG NPR: Drawing strokes directly on 3D models," in *Proceedings of SIGGRAPH' 02*, 2002, pp. 755–762.

[13] F. Durand, V. Ostromoukhov, M. Miller, F. Duranleau, and J. Dorsey, "Decoupling strokes and high-level attributes for interactive traditional drawing," in *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, 2001, pp. 71–82.

[14] T. Isenberg, P. Neumann, S. Carpendale, M. C. Sousa, and

Figure 10. Example 5: colored pencil drawing of pot plant in the window

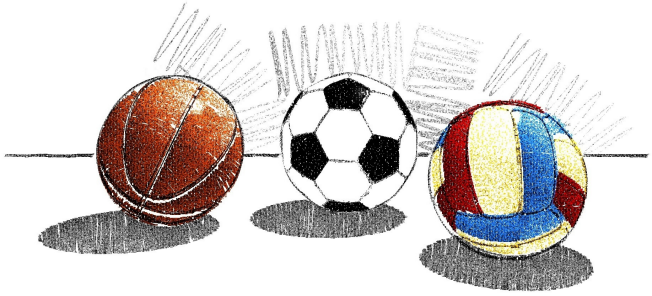


Figure 11. Example 6: colored pencil drawing of basketball, football and volleyball.

J. A. Jorge, "Non-photorealistic rendering in context: an observational study," in *Proceedings of NPAR '06*, 2006, pp. 115–126.

[15] H. Lee, S. Kwon, and S. Lee, "Real-time pencil rendering," in *Proceedings of NPAR '06*, 2006, pp. 37–45.

[16] X. Mao, Y. Nagasaka, and A. Imamiya, "Automatic generation of pencil drawing from 2D images using line integral convolution," in *proceedings of the 7th International Conference on Computer Aided Design and Computer Graphics*, 2001, pp. 240–248.

[17] N. Li and Z. Huang, "A feature-based pencil drawing method," in *Proceedings of GRAPHITE '03*, 2003, pp. 135–142.

[18] S. Sun and D. Huang, "Efficient region-based pencil drawing," in *Proceedings of WSCG '07*, 2007, pp. 279–286.

[19] D.-E. Xie, Y. Zhao, D. Xu, and X. Yang, "Convolution filter based pencil drawing and its implementation on gpu," in *Proceedings of the 7th international conference on Advanced parallel processing technologies*, 2007, pp. 723–732.

[20] B. Cabral and L. C. Leedom, "Imaging vector fields using line integral convolution," in *Proceedings of SIGGRAPH '93*, 1993, pp. 263–270.

[21] A. Hertzmann and D. Zorin, "Illustrating smooth surfaces," in *Proceedings of SIGGRAPH '00*, 2000, pp. 517–526.

[22] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella, "Suggestive contours for conveying shape," in *Proceedings of SIGGRAPH '03*, 2003, pp. 848–855.

[23] D. DeCarlo, A. Finkelstein, and S. Rusinkiewicz, "Interactive rendering of suggestive contours with temporal coherence," in *Proceedings of NPAR '04*, 2004, pp. 15–145.

[24] S. Grabli, E. Turquin, F. Durand, and F. Sillion, "Programmable style for NPR line drawing," in *Proceedings of Eurographics Symposium on Rendering '04*, 2004.

[25] Y. Ohtake, A. Belyaev, and H.-P. Seidel, "Ridge-valley lines on meshes via implicit surface fitting," in *Proceedings of SIGGRAPH '04*, 2004, pp. 609–612.

[26] T. Judd, F. Durand, and E. Adelson, "Apparent ridges for line drawing," in *Proceedings of SIGGRAPH '07*, 2007, pp. 19:1–19:7.

[27] M. A. Kowalski, L. Markosian, J. D. Northrup, L. Bourdev, R. Barzel, L. S. Holden, and J. F. Hughes, "Art-based rendering of fur, grass, and trees," in *Proceedings of SIGGRAPH '99*, 1999, pp. 433–438.

[28] J. D. Northrup and L. Markosian, "Artistic silhouettes: a hybrid approach," in *Proceedings of NPAR '00*, 2000, pp. 31–37.

[29] F. Durand, "An invitation to discuss computer depiction," in *Proceedings of NPAR '02*, 2002, pp. 111–124.

[30] R. Barzel, "Lighting controls for computer cinematography," *J. Graph. Tools*, vol. 2, pp. 1–20, 1997.

[31] A. Gooch, B. Gooch, P. Shirley, and E. Cohen, "A non-photorealistic lighting model for automatic technical illustration," in *Proceedings of SIGGRAPH '98*, 1998, pp. 447–452.

[32] B. Dodson, *Keys to drawing*. North Light Books, 1990.

[33] A. Girshick, V. Interrante, S. Haker, and T. Lemoine, "Line direction matters: an argument for the use of principal directions in 3D line drawings," in *Proceedings of NPAR '00*, 2000, pp. 43–52.

[34] G. Franks, *The art of pencil drawing*. Walter Foster Publishing, Inc, 2004.

[35] V. Sommers, *Drawing and cognition*. Cambridge university press, 1984.

[36] R. G. Meulenbroke and A. J. Thomassen, "Stroke-direction preferences in drawing and handwriting," *Human Movement Science*, vol. 10, pp. 247–270, 1991.

[37] P. Litwinowicz, "Processing images and video for an impressionist effect," in *Proceedings of SIGGRAPH '97*, 1997, pp. 407–414.

[38] K. Perlin, "An image synthesizer," in *Proceedings of SIGGRAPH '85*, 1985, pp. 287–296.