Pencil-Like Sketch Rendering of 3D Scenes Using Trajectory Planning and Dynamic Tracking

Günay Orbay
Carnegie Mellon University

Levent Burak Kara
Carnegie Mellon University

***Abstract*—We present a new non-photorealistic rendering method to render 3D scenes in the form of pencil-like sketches. This work is based on the observation that the dynamic feedback mechanism involving the human visual system and the motor control of the hand collectively generates the visual characteristics unique to hand-drawn sketches. At the heart of our approach is a trajectory planning and tracking algorithm that generates the sketch using a dynamic pen model. A set of target strokes are generated from the silhouette lines, edges, and shaded regions which serve as the target trajectory for a closed-loop dynamic pen model. The pen model then produces the rendered sketch, whose characteristics can be adjusted with a set of trajectory and tracking parameters. We demonstrate our approach with examples.**

I. Introduction

We describe a pencil-like sketch rendering method of 3D scenes using a range of styles spanning novice drawers to trained artists. The proposed method produces the visual features unique to sketching through a dynamic modeling of the drawing process. The perceived style of the resulting sketches can be manipulated using the parameters of the system.

As suggested in [1], hand-drawn sketches exhibit a variety of other artifacts such as overtracing, hooks at the stroke ends, tonal variation in stroke intensities, lifting versus non-lifting strokes, and layered hatching and cross hatching. These features typically vary based on the skill level of the sketcher, the sketcher's particular precision during sketching, the level of detail to be included in the sketch, and on the dynamics of the pen, hand and the arm. Collectively, these phenomena give rise to a rich set of stylistic variations in sketches, which have been difficult to represent and reproduce algorithmically.

As one step toward addressing this challenge, we describe a method to incorporate and control such effects using a two step process. This involves a stroke field design and dynamic stroke tracking. At the heart of our approach is the design of a 3D pen trajectory from render buffers of a 3D scene through edge detection, region clustering and stroke generation algorithms. When combined with a dynamic pen model, this approach helps produce the unique characteristics observed in pencil sketches. The first step of stroke field design identifies the silhouette, edge and hatching regions, and establishes the stroking behaviors to be applied to them. This results in a 3D target pen trajectory that serves as a reference to be tracked. The second step involves the tracking of this reference by a dynamic pen model. This model enables a rich set of artifacts including a variation in tracking accuracy, stroke skipping, overshoots, pen pressure, non-lifting strokes, and muscle jitter through a set of dynamic parameters.

II. Related Work

A. Line Rendering of 3D Objects

1) Silhouette and Edge Rendering: We discuss the prior work on non-photorealistic rendering of silhouettes and edges in three subgroups. The first group of techniques aim to identify the most representative set of silhouettes and edges that best articulate the shape. DeCarlo *et al.* [2] described suggestive contours, Kalogerakis *et al.* [3] introduced a real-time rendering method that uses learned curvatures variations, Zhang *et al.* [4] utilized the Laplacian of the surface illumination for contour detection. Inspired by these works, we use discrete Laplacian kernels on the depth and normal buffers to identify the silhouettes and sharp edges from input images.

The second group of work focus on rendering identified silhouettes and edges in prescribed styles. Markosian *et al.* [5] described real-time rendering algorithms combined with stylized stroke rendering. Yeh and Ouhyoung [6] developed stroke rendering algorithms that helps mimic Chinese inking styles. Hertzmann *et al.* [7] describe an algorithm to enable curve analogies by learning a statistical model from an input pattern and replicate it on another curve. Barla *et al.* [8] developed statistical models that can learn and produce a wider range of patterns. Brunn *et al.* [9] developed a multi-resolution framework for encoding and generating curve patterns. While these studies cover a large variety of stylistic rendering, little or no emphasis is given to pencil-like renderings. This leaves the simulation of characteristic features contained in pencil sketches an open challenge. This work aims to enable the generation of such features within a wide variety of stylistic effects.

2) Hatching Rendering: Winkenbach and Salesin [10] developed stroke textures to render both textures and tone with line drawings. Salisbury *et al.* propsed a similar approach that produces scale-dependent renders on different scales of sketches. Winkenbach and Salesin [11] later extended their work to rendering parametric surfaces with lines following parametric derivatives on the surface. Salisbury *et al.* [12] proposed an interactive design system that allows the users to quickly design directional fields to support line renderings.

Inspired by the works of Winkenbach *et al.* , a multitude of methods were developed to render lines by tracing the directional fields computed from 3D geometries. Hertzmann and Zorin [13] presented a set of algorithms to calculate hatching lines that follow directional fields defined on surfaces. Praun *et al.* [14] proposed a method for real-time rendering using tonal maps. Palacios and Zhang [15] described an interactive method for field design on surfaces using rotational tensor symmetry. Paiva *et al.* [16] proposed a physically inspired directional field design method that calculates fluid-based hatching strokes. Jodoin *et al.* [17] also used sample drawing methods that enables the reproduction of recorded hatching patterns. Kalnins *et al.* [18] incorporated similar pattern learning and reusing techniques in an interactive non-photorealistic rendering system. Similarly, Kalogerakis*et al.* [19] target learning hatching preferences of an artist from a sample sketch that is drawn over a 3D render.

Most of the prior work in this area is based on the observation that hatching strokes are commonly drawn in directions that closely align with the underlying geometry. Although this observation is valid for in contexts where precise and accurate 3D renderings are desired, it may not apply in other contexts involving casually drawn pencil sketches. In this work, we base our hatching patterns on the shape of the shaded regions.

III. Rendering Approach

Our approach consists of two main steps: (1) Reference trajectory design (2) Trajectory tracking via a dynamic pen model. Within each step, a number of parameters are noted to facilitate the discussions in Section IV. These parameters are highlighted at their point of definition.

A. *Reference Trajectory Design*

This step takes as input the conventional render, normal and depth buffers of a 3D scene. From these buffers, it produces the curves and regions to be rendered with strokes. We use the depth and normal buffers to identify the silhouettes and edges, and the render buffer for shading region identification and tonal control.

From these buffers, our method generates a 2D stroke pattern that traces the identified curves, and shades the regions. For each stroke generated during this process, its position, shape, length, and intensity variation is computed using the render buffers. Spatial and angular perturbations are then applied to the strokes to simulate the inaccuracies with the human motor control system. Finally, this stroke pattern is transformed into a single, time-dependent 3D space curve. The x and y components of this resulting curve control the stroke's final shape in the image plane, while the z coordinate dictates whether the stroke leaves a trace on the paper, and if so, the stroke's thickness and intensity. The transformation from a 2D pattern to a 3D curve allows the subsequent pen model to follow a 3D path, thereby allowing a rich set of visual artifacts to be created on its projection to the image plane.

1) Silhouettes and Edges: To design the stroke trajectory for the silhouettes and edges of an object, we use conventional edge detection methods on the depth and normal buffers. However, other methods such as suggestive contours [2], [3], [4] could also be used without loss of generality. In our implementation, we use the Laplacian convolution kernel on

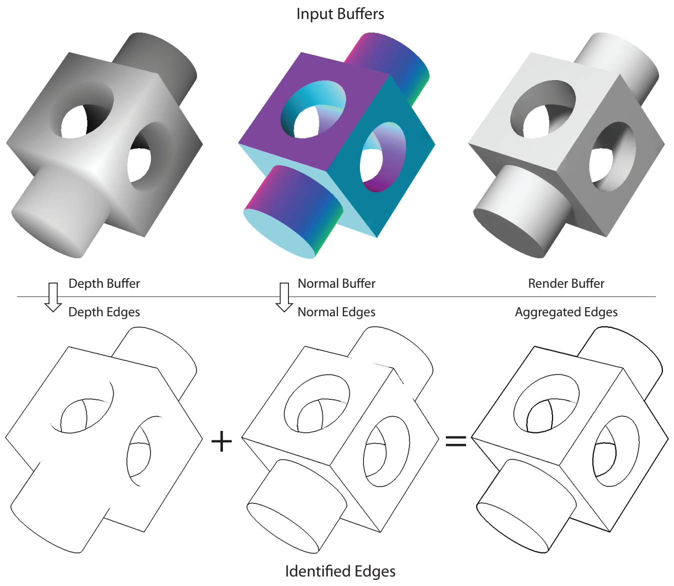

Fig. 1. The input buffers: Depth, normal, and render buffers. The silhouettes calculated from the depth buffer and the sharp edges identified from the normal buffer are combined to identify the silhouettes and edges. The render buffer is later used for hatching.

the depth and normal buffers to identify the curves at which the depth and normal vectors vary abruptly. (Fig. 1). When combined, this results in the extraction of silhouettes and sharp edges of the model as a set of polylines. From this point onward, we do not distinguish between a silhouette or an edge curve, and instead simply refer to them as ***siledge*** curves.

While this analysis may produce clean siledges , it may also produce disconnected islands of such curves. In either case, we assemble all siledge curves into a single chain using their spatial proximity. In addition to its spatial configuration, the siledge path also maintains information regarding the underlying image intensity from the render buffer, which is later used to generate the height (z)coordinates.

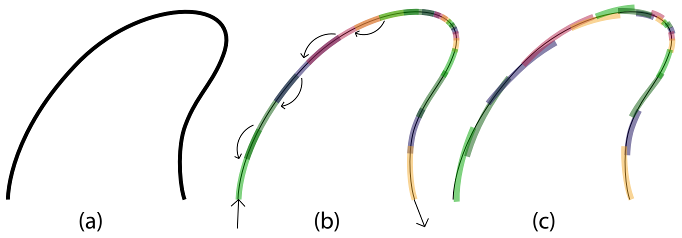

Fig. 2. Each siledge is converted into a series of strokes. The stroke lengths are determined adaptively according to the curvature along the siledge. The magnitude of overlap between consecutive strokes is determined probabilistically from a normal distribution. The strokes are altered by random perturbations in orientation, length, and scale. Strokes are rendered thicker than normal for demonstration.

The next step involves stroke design on this siledge path. The local curvature of the path has an influence on the length of the strokes to be generated. High curvature regions typically result in shorter strokes, while low curvature sections can be traced using longer strokes. Once stroke lengths are determined, we use a normal distribution to generate variations in the amount of overlap between consecutive strokes, as a function of the strokes' lengths. Figure 2b demonstrates this idea. We control the average length of the strokes and the overlap with two parameters, $\mathbf{P}_{\mathbf{L}_{\mathbf{mean}}}$ and $\mathbf{P}_{\mathbf{overlap}}$, respectively. Finally, the stroke orientations, lengths and positions are randomly perturbed. We do so using a single parameter $\mathbf{P}_{\mathbf{siledge}}$ that concurrently controls the standard deviations of these attributes, as shown in Figure 2c.

Once the size, shape and position of the strokes are determined, points are sampled along the stroke to mimic equal time intervals during traversal. Similar to stroke length determination, we simulate the effect of the pen slowing down around high curvature regions using curvature-adaptive sampling. High curvature regions are thus sampled more densely compared to low curvature regions. In the following sections, we discuss the effects of the pen speed on tracking, and thus the stroke renditions produced by our pen model.

2) Hatching: We use the render buffer to identify the regions to be hatched for shading. For this, we apply a low pass Gaussian filter to the render image and use an intensity cut-off value ($\mathbf{P}_{\mathbf{int-cutoff}}$) to isolate the shading regions. The first row of Figure 3 illustrates the idea. This process results in a set of bounded regions, each treated as a separate area to be shaded. Within each region, a hatching pattern is designed that maintains a flow congruent with that region's geometry.

If the previously computed siledge path crosses through an identified region, that region is further subdivided into smaller regions. Since siledges correspond to discontinuities in the depth and normal maps, this subdivision helps our system produce distinct hatching patterns within each subregion. This desirably helps the hatching to preserve such discontinuities.

As shown in the second row of Figure 3, the identified regions often exhibit complex boundaries. This prevents a straightforward adoption of a hatching direction, as the resulting hatching behavior would require frequent pen-up/pen-down motions. Artists typically prefer to shade such regions by dividing it into multiple smaller groups that can be continuously hatched with a series of strokes. Inspired by this observation, we compute the medial axis of the region, which reduces our analysis to a group of branches. To identify the salient regions to be distinguished, we seek to merge branches that are connected with smooth transitions. This is done by studying the curvature of a compound curve that is formed by joining the two branches. If the discrete curvature of the compound curve at the joint is less than a threshold, the branches are merged (Figure 3f). This process is repeated until no more mergers are available. Next, short branches are identified and removed, thereby leaving a small set of skeletal curves forming the axes of the subregions to be hatched.

We next generate a hatching pattern for each identified region using each region's medial axis branch as a reference. Figure 4 illustrates this process. For a given region, the hatching strokes are created from arc segments sharing a fixed radius. This mimics the arc-like strokes caused by the motion of the fingers and the wrist of the artist about a virtual pivot

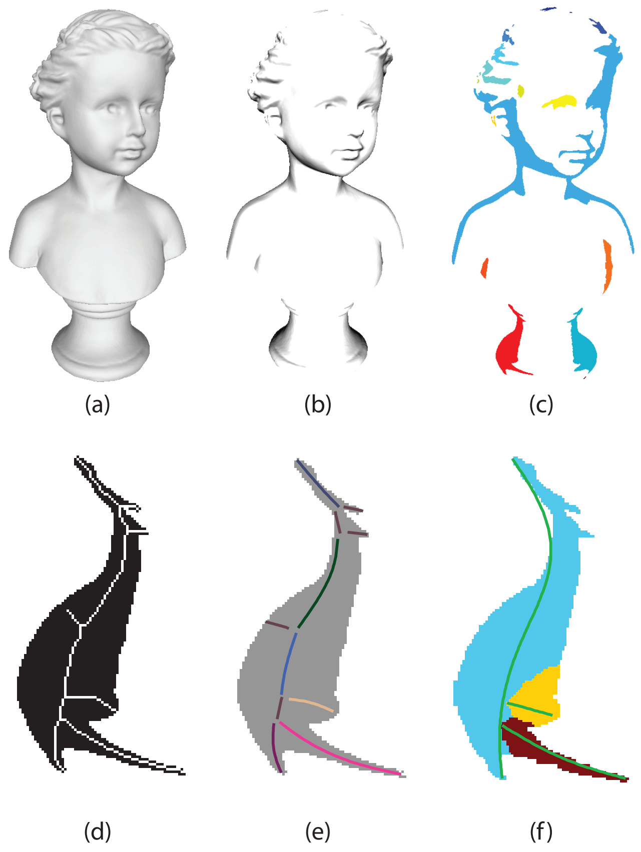

Fig. 3. Identification of hatching regions. (a) The render buffer is processed to determine (b) the hatching regions and shading tones (c) resulting in islands of regions. (d) For each region, the skeleton is calculated (e) and the branches are segmented. (f) Continuous branches are merged to produce compound branches, from which a reduced set of regions are determined.

point. The orientation of the hatching strokes are chosen such that the generated pattern is a function of the region's shape. For slender regions, the strokes are configured such that they are always offset by a prescribed angle relative to the medial axis. In such cases, the hatching pattern closely follows the shape of the medial axis. However, this requirement is relaxed for wider regions so that the long hatching strokes produced for wider regions do not form fan-like patterns in an effort to follow the tight turns of the medial axis. This effect is formulated by assigning an offset angle along the medial axis, and smoothing this angle vector in relation to the region's width along the medial axis[1]. After the hatching strokes are generated, parts of the strokes falling outside the region are masked out, leaving the hatching only in the region to be shaded.

Similar to siledges, perturbations are applied to the orientation, position and size of the hatching strokes in order to simulate the inaccuracies of the human arm. This is controlled with a single parameter $\mathbf{P}_{\mathbf{hatching}}$. Additionally two other

[1]Note that the medial axis encodes the distance to the closest boundary points, thereby allowing the thickness of the region to be trivially computed as the medial axis is traversed.

parameters in this process control the mean frequency of the hatching strokes $\mathbf{P_{htc-freq}}$, and the deviation of the hatching frequency within the region $\mathbf{P_{htc-freq-dev}}$. Similar to the siledge strokes, hatching strokes are lastly sampled to generate points along their traversal. Since hatches are formed by constant radius arcs, the curvature-sensitive sampling produces equally sampled points along these strokes.

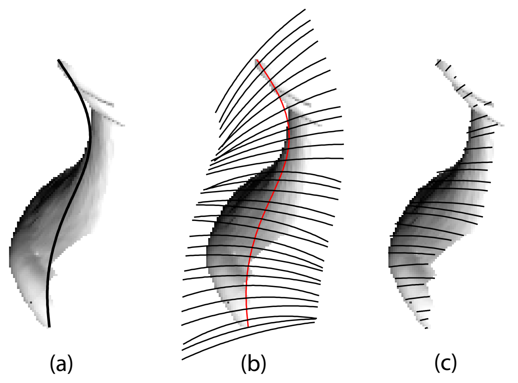

Fig. 4. Hatching stroke generation. A series of arcs are oriented along the medial axis. The arcs are trimmed with the hatching region, and perturbed in orientation and scale generating the final hatching strokes.

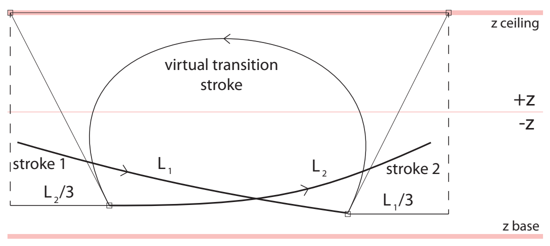

Fig. 5. The virtual transition strokes connect consecutive strokes. The strokes attain z coordinates based on the underlying intensities. The virtual transition strokes ascend into $+z$ space, though not exceeding $\mathbf{z_{ceiling}}$. The follow through of the transition strokes parallel to the virtual paper is determined by the underlying stroke's length.

3) Stroke Connections: At this point, the image intensity values along the siledge and hatching strokes (obtained from the render image) are used to create the height (z) coordinates for each stroke. This is facilitated by considering a virtual paper surface representing the $z = 0$ plane. All strokes generated thus far lie below this plane, as this indicates the pen being in contact with the paper. Darker intensities are transformed into deeper z coordinates whereas lighter ones are interpreted as shallow depths. Image intensities in the range $[0, 1]$ are linearly mapped to the z coordinates $[0, \mathbf{P_{z_{base}}}]$, where $\mathbf{P_{z_{base}}}$ is the parameter governing the maximum penetration depth as shown in Fig. 5.

After stroke heights are computed, successively generated strokes are attached to each other using virtual transition strokes. This simulates the pen lifting off the surface after one stroke ends, and reentering the paper for the next stroke. We model this effect using cubic Beziér curves as shown in Fig. 5. The design of these strokes is critical, as they govern segments of the trajectory that our dynamic pen model will be tracking. To this end, we introduce a control parameter $\mathbf{P_{z_{ceiling}}}$ that determines the maximum height the pen is allowed to ascend during transitions. Additionally, to account for longer strokes causing higher resistance for the pen to change direction at the liftoff point, we extend a vector proportional to the stroke's length. We use the right triangle formed by the ceiling, and the extension vector of the stroke to determine the tangent vectors of the Beziér curve. This allows a natural *follow through* for each of the transition strokes, based on the strokes they connect. Using this formulation, we generate a single trajectory that the dynamic pen model will next follow. Figure 6 shows the final 3D trajectory produced in this way.

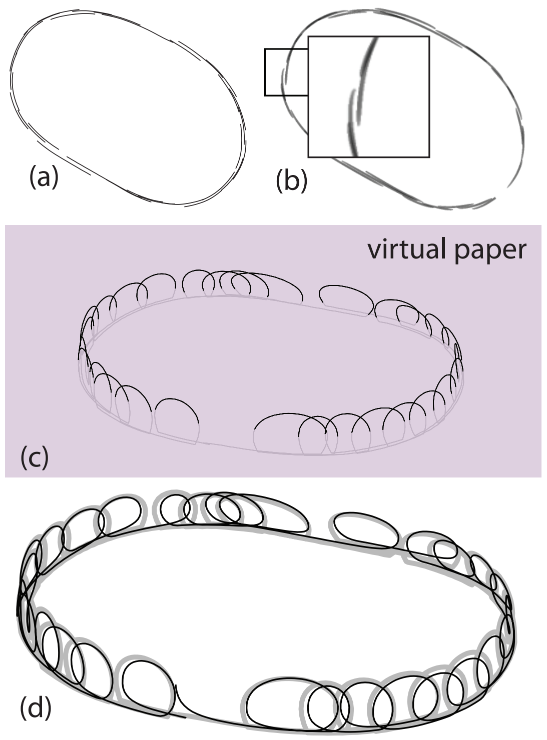

Fig. 6. (a) A series of strokes sampled in 2D (b) The resulting rendered strokes. (c) 2D strokes are transformed into 3D reference trajectory that intersects the virtual plane at multiple locations. (d) The reference trajectory (gray) is tracked using a dynamic pen (black), producing the render in (a).

B. Dynamic Pen Model

The calculated reference trajectory is next tracked by a dynamic pen model which produces the final rendered strokes (Fig. 7). The model consists of a closed loop control system representing the visual system and the motor control of the human. A free point mass represents the lumped pen, hand, and arm masses. The input to the system is the continuous 3D reference trajectories computed previously, and the output is the actual 3D trajectory traced by the point mass. A muscle model mathematically equivalent to a PID controller is used for tracking. This formulation is similar to that used in [20]. We enhance this model by adding random muscle jitter to the guidance force applied to the point mass. The jitter is modeled by a first order low pass filter that rejects frequencies above a cut-off frequency. The equivalent mass is controlled by the parameter $\mathbf{P_{mass}}$, while the amount of jitter and the filter cut-off frequency are linearly controlled by the parameter $\mathbf{P_{jitter}}$.

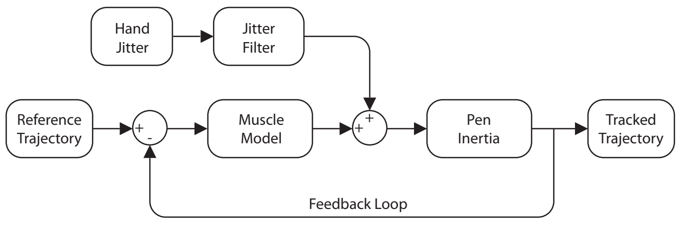

Fig. 7. Dynamic pen model is a closed loop control system that guides a pen/arm mass along a reference trajectory. The force on the pen (*i.e.* the force that the muscle model applies to the pen inertia) is perturbed to model the jitter in the motor control system.

We model the motion in three dimensions as three independent systems of equations and use a Runge Kutta solver to obtain the pen's motion in space as a function of time. The dynamic pen properties can be adjusted in the (x, y) and z directions independent from each other. Figure 6d shows the reference trajectory (gray) and the resulting pen trajectory (black) produced using this process. The final rendered strokes are computed from the intersections of the calculated pen trajectory with the virtual plane. The segments of the pen trajectory under the $z = 0$ plane form the strokes to be rendered. Furthermore, using the penetration depths of the trajectory, intensity values are assigned along the strokes similar to the way they were computed in Section III-A3.

IV. Results and Discussions

A. Performance

Our method is implemented in MATLAB. The hatching region extraction described in Section III-A2 is currently the most time consuming step. The computational cost of other calculations including stroke generation and perturbation, reference trajectory design, reference tracking using the dynamic pen model and image differencing are negligible compared to the region extraction step. The time required for the calculations depend on the resolution of the render buffers, the number of silhouette edges generated, and the total area of the regions to be hatched. The overall processing times varies between 30 seconds to 3 minutes of a Core 2 Dup computer.

B. Examples

Figure 8 presents example renderings produced by our system. The input renders are obtained from software that can export such buffers. The blow out windows show various phenomena generated from our dynamic tracking, including variations in siledge strokes, continuous versus distinct hatching strokes, cross hatching strokes, varying pen pressure and stroke intensities. In these examples, we have manually used two layers of hatching strokes where hatching directions make about 60 degrees between each other. Figure 9 shows the variations in sketch styles that can obtained by varying the described sketch parameters.

C. Limitations

Our region segmentation analysis is able to produce accurate partitions for smooth boundary regions. However, in cases where the boundaries exhibit significant waviness, the algorithm may produce fragmented regions. These regions, when hatched, can cause undesirable hatching patterns that are visually unappealing (in Fig. 8, the left leg of the second child from the right). A similar situation occurs when a short siledge is trapped within a large hatching region. This causes fragmented regions, where the hatching angles change abruptly within the region. One solution may involve a region analysis algorithm that also takes into account additional factors such as intensity variations within the region, together with the image skeleton approach we use.

V. Conclusion

We described a new non-photorealistic rendering method to render 3D objects in the form of pencil-like sketches. Our method is based on the observation that the dynamic feedback mechanism involving the human visual system and the motor control of the hand collectively generates the visual characteristics unique to hand-drawn sketches. To this end, we developed a stroke planning and dynamic tracking algorithm that produces a sketch render using silhouette, edge and hatching strokes. This approach allows visual artifacts unique to hand drawn sketches to be reproduced.

The mapping between our algorithmic parameters and the three sketch descriptors we present provides a convenient basis for studying the stylistic variations produced by our system. We believe this opens a new avenue for future work where the technical parameters can be learned and mapped to the descriptor parameters using data-driven approaches, thereby providing a link between artistic and algorithmic languages. We also believe our system may also serve as a training and visualization tool, as our approach helps produce a temporal progression of the sketch.

References

[1] K. Eissen and R. Steur, *Sketching: drawing techniques for product designers*. Bis, 2007.

[2] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella, "Suggestive contours for conveying shape," *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 22, no. 3, pp. 848–855, Jul. 2003.

[3] E. Kalogerakis, D. Nowrouzezahrai, P. Simari, J. McCrae, A. Hertzmann, and K. Singh, "Data-driven curvature for real-time line drawing of dynamic scenes," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 1, p. 11, 2009.

[4] L. Zhang, Y. He, X. Xie, and W. Chen, "Laplacian lines for real-time shape illustration," in *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. ACM, 2009, pp. 129–136.

[5] L. Markosian, M. Kowalski, D. Goldstein, S. Trychin, J. Hughes, and L. Bourdev, "Real-time nonphotorealistic rendering," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1997, pp. 415–420.

[6] J. Yeh and M. Ouhyoung, "Non-photorealistic rendering in chinese painting of animals," *Journal of System Simulation*, vol. 14, no. 6, pp. 1220–1224, 2002.

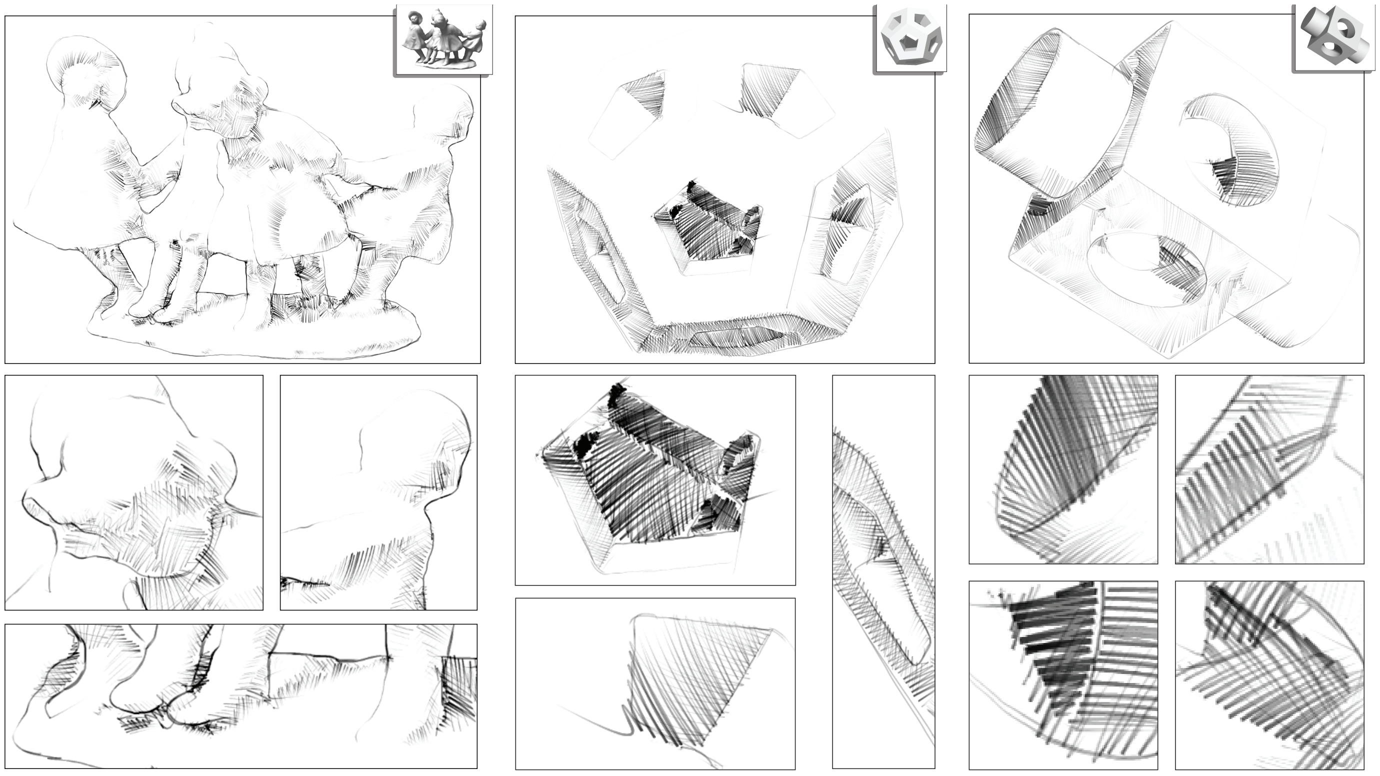

Fig. 8. Results on various models. Note the hatching behaviors and dynamic effects illustrated in the blow out views.

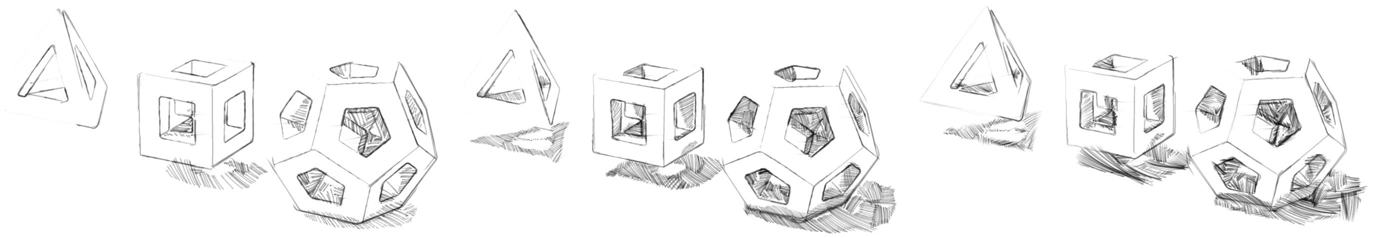

Fig. 9. The renders resulting from different values of sketch parameters. Please note the varying amounts of strokes and different hatching behaviors.

[7] A. Hertzmann, N. Oliver, B. Curless, and S. Seitz, "Curve analogies," in *Proceedings of the 13th Eurographics workshop on Rendering*. Eurographics Association, 2002, pp. 233–246.

[8] P. Barla, S. Breslav, J. Thollot, F. Sillion, and L. Markosian, "Stroke pattern analysis and synthesis," in *Computer Graphics Forum*, vol. 25, no. 3. Wiley Online Library, 2006, pp. 663–671.

[9] M. Brunn, M. Sousa, and F. Samavati, "Capturing and re-using artistic styles with reverse subdivision-based multiresolution methods," *International Journal of Image and Graphics*, vol. 7, no. 4, pp. 593–615, 2007.

[10] G. Winkenbach and D. Salesin, "Computer-generated pen-and-ink illustration," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM, 1994, pp. 91–100.

[11] ——, "Rendering parametric surfaces in pen and ink," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 469–476.

[12] M. Salisbury, M. Wong, J. Hughes, and D. Salesin, "Orientable textures for image-based pen-and-ink illustration," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1997, pp. 401–406.

[13] A. Hertzmann and D. Zorin, "Illustrating smooth surfaces," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 517–526.

[14] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein, "Real-time hatching," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, p. 581.

[15] J. Palacios and E. Zhang, "Rotational symmetry field design on surfaces," *ACM Trans. Graph.*, vol. 26, Jul. 2007.

[16] A. Paiva, E. Vital Brazil, F. Petronetto, and M. Sousa, "Fluid-based hatching for tone mapping in line illustrations," *The Visual Computer*, vol. 25, no. 5, pp. 519–527, 2009.

[17] P. Jodoin, E. Epstein, M. Granger-Piché, and V. Ostromoukhov, "Hatching by example: a statistical approach," in *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*. ACM, 2002, pp. 29–36.

[18] R. Kalnins, L. Markosian, B. Meier, M. Kowalski, J. Lee, P. Davidson, M. Webb, J. Hughes, and A. Finkelstein, "Wysiwyg npr: Drawing strokes directly on 3d models," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 755–762, 2002.

[19] E. Kalogerakis, D. Nowrouzezahrai, S. Breslav, and A. Hertzmann, "Learning Hatching for Pen-and-Ink Illustration of Surfaces," *ACM Transactions on Graphics*, vol. 31, no. 1, 2011.

[20] D. House and M. Singh, "Line drawing as a dynamic process," *Pacific GraphicsŠ 07*, pp. 351–360, 2007.