Journal of Visual Languages and Computing 25 (2014) 481–493

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

Pencil-like sketch rendering of 3D scenes using trajectory planning and dynamic tracking

Günay Orbay, Levent Burak Kara [*]

Mechanical Engineering Department, Carnegie Mellon University, Pittsburgh, PA 15213, United States

A R T I C L E   I N F O

Article history:
Received 31 October 2013
Received in revised form
8 February 2014
Accepted 20 February 2014
This paper has been recommended for acceptance by S.-K. Chang
Available online 28 February 2014

Keywords:
Non-photorealistic rendering
Dynamic pen model
Drawing style

A B S T R A C T

Objective: We present a new non-photorealistic rendering method to render 3D scenes in the form of pencil-like sketches.

Methods: 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 generate the sketch in multiple layers using a dynamic pen model. On each layer, 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. This process continues in several layers until the tonal difference between the sketch and the original 3D render is minimized.

Results: We demonstrate our approach with examples that are created by controlling the parameters of our sketch rendering algorithms.

Conclusion: The examples not only show typical sketching artifacts that are common to human-drawn sketches but also demonstrate that it is capable of producing multiple sketching styles.

1. 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 three global parameters that simulate (1) the skill level of the sketcher, (2) the neatness of the sketching behavior, and (3) the desired detail level in the final sketch.

Previous studies in stylized pencil/ink rendering have primarily focused on specific elements of interest. These include rendering silhouettes, edges and contours [27,7], hatching for shading [11,20], stippling [25], half-toning [21], and texture and pattern rendering [28,2]. As suggested in [8], however, 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 (Fig. 1). 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, the 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

* Corresponding author.
E-mail address: lkara@cmu.edu (L. Burak Kara).

http://dx.doi.org/10.1016/j.jvlc.2014.02.002
1045-926X © 2014 Elsevier Ltd. All rights reserved.

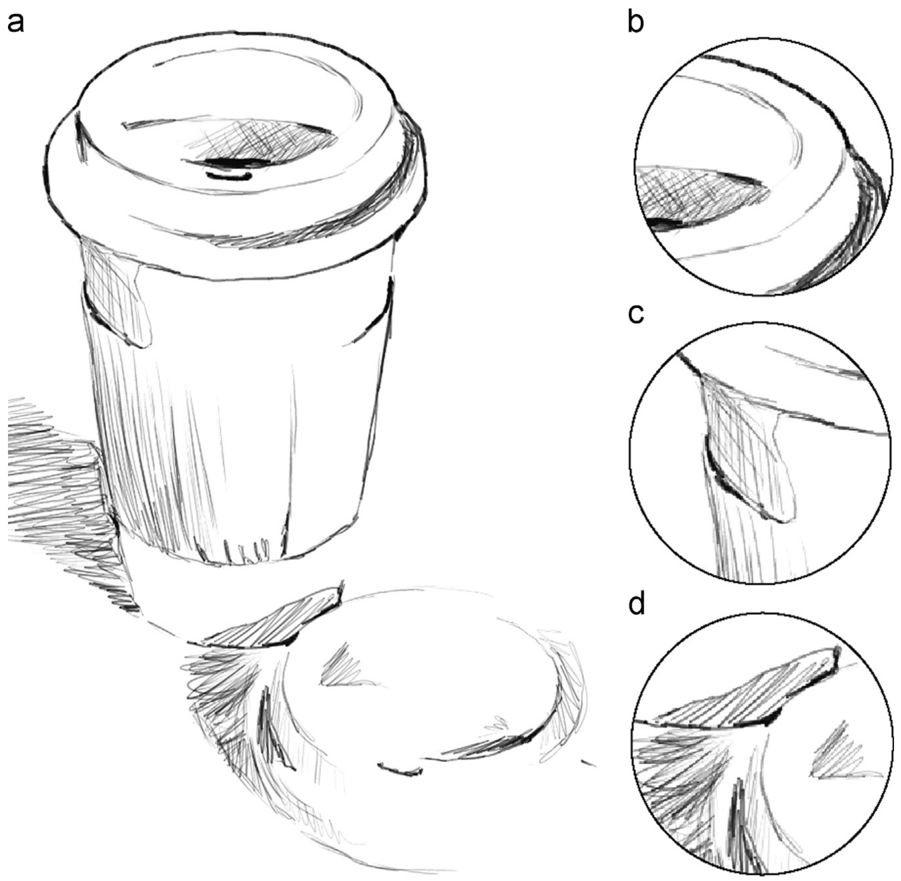

Fig. 1. Commonly observed features in hand-drawn pencil sketches: (a) overtracing and overlapping strokes; (b, c) hatching for shading in multiple layers; (d) a series of continuous (no pen lift) hatching strokes.

using a two step process. This involves a stroke field design and dynamic stroke tracking, all performed in multiple layers using a feedback mechanism. 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. Each application of these two steps produces a rendered sketch, which is then compared against the original 3D render within a feedback loop. This allows the final sketch to evolve in layers, until the tonal difference between the reference image and the resulting sketch is minimized.

Inspired by [23] who categorizes line drawings into a small set of qualitative descriptors, we develop three such descriptors using which the resulting sketch characteristics can be manipulated. These descriptors aim to mimic (1) the skill level of the sketcher, (2) the precision or neatness of the sketcher at the particular task, and (3) the amount of detail the sketcher is permitted to articulate in the sketch. These dimensions form a convenient basis to generate and study the range of stylistic variations enabled by our approach.

2. Related work

2.1. Line rendering of 3D objects

2.1.1. 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 aims to identify the most representative set of silhouettes and edges that best articulate the shape. DeCarlo et al. [7] described suggestive contours, Kalogerakis et al. [16] introduced a real-time rendering method that uses learned curvatures variations, Zhang et al. [32] 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. Without loss of generality, our techniques can work with the these other contour extraction methods. In our experiments, the choice of Laplacian kernels had a minimal effect on the end results in terms of the variety of sketch characteristics that can be produced.

The second group of work focuses on rendering identified silhouettes and edges in prescribed styles. Markosian et al. [18] described real-time rendering algorithms combined with stylized stroke rendering. Yeh and Ouhyoung [31] developed stroke rendering algorithms that helps mimic Chinese inking styles. Hertzmann et al. [10] 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. [2] developed statistical models that can learn and produce a wider range of patterns. Brunn et al. [4] developed a multi-resolution framework for encoding and generating curve patterns. Cole and Finkelstein [5] developed edge detection and stylized render algorithms. Kalnins et al. [14], Bernard et al. [3], and DeCarlo et al. [6] described coherent stylized rendering algorithms that are invariant to animations and view point changes. 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.

In the last group we find algorithms for modeling stylistic variations observed in pencil and ink drawings. Sousa and Buchanan [26] developed a rendering system that produces graphite pencil sketches from 3D polygonal models. Sousa and Prusinkiewicz [27] proposed calculating a chain of lines that are smoothed to replicate commonly observed dynamic effects. Goodwin et al. developed an inking algorithm based on the abrupt changes on the illumination intensities on the models to overlay multiple silhouette renders mimicking overlapping strokes. Although these methods individually produce one or more characteristics seen in pencil sketches, shading through hatching is not addressed. In this work, we develop a generative model that incorporates multiple unique characteristics pertinent to silhouettes, edges, and multi-layer shading.

2.1.2. Hatching rendering

Winkenbach and Salesin [29] developed stroke textures to render both textures and tone with line drawings. Salisbury et al. proposed a similar approach that produces scale-dependent renders on different scales of sketches. Winkenbach and Salesin [30] later extended their work to rendering parametric surfaces with lines following parametric derivatives on the surface. Salisbury et al. [24] 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 was developed to render lines by tracing the directional fields computed from 3D geometries. Hertzmann and Zorin [11] presented a set of algorithms to calculate hatching lines that follow directional fields defined on surfaces. Praun et al. [22] proposed a method for real-time rendering using tonal maps. Webb et al. [28] later used tonal maps to enable enhanced tonal control. Palacios and Zhang [20] described an interactive method for field design on surfaces using rotational tensor symmetry. Paiva et al. [19] proposed a physically inspired directional field design method that calculates fluid-based hatching strokes. Jodoin et al. [13] also used sample drawing methods that enable the reproduction of recorded hatching patterns. Kalnins et al. [15] incorporated similar pattern learning and reusing techniques in an interactive non-photorealistic rendering system. Similarly, Kalogerakiset al. [17] 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. For example, design sketches in [8] typically contain quickly drawn hatching patterns whose directions are primarily dictated by the shape of the shaded region and not the underlying geometry. This difference in hatching preferences plays an important role in distinguishing computed versus hand-made drawings. In this work, we base our hatching patterns on the shape of the shaded regions.

2.2. Simulation of pen dynamics

A group of studies used pen and arm models to mimic the dynamic effects seen in ink drawings. Fujioka and Kano [9] developed a two link arm model combined with a 3D brush to simulate inking dynamics. By controlling the parameters associated with the arm and the 3D brush model, they achieved different stroke transitions with varying stroke thicknesses. House and Singh [12] utilized a simpler dynamic pen model to render 3D object silhouettes in various styles achieved through variations in the dynamic properties. These works do not focus on the design and the rendition of hatching strokes, thus strictly producing ink renderings of silhouettes and edges. Almeraj et al. [1] also utilize an arm model, however to render silhouettes and edges together with equally spaced unidirectional hatching strokes. Unlike the results seen in other dynamic modeling works, the results contain little or no cues of smooth, continuous transitions between consecutive strokes. Our method attempts to produce characteristics more commonly in pencil-like sketches by combining the effects related with the design of the strokes and the effects related with the dynamics of the drawing process.

3. Rendering approach

Our approach consists of three main steps: (1) reference trajectory design; (2) trajectory tracking via a dynamic pen model; (3) multi-layer tonal matching via visual feedback. Fig. 2 illustrates these components, which are detailed in the following sections. Within each step, a number of parameters are noted to facilitate the discussions in Section 4. These parameters are highlighted at their point of definition.

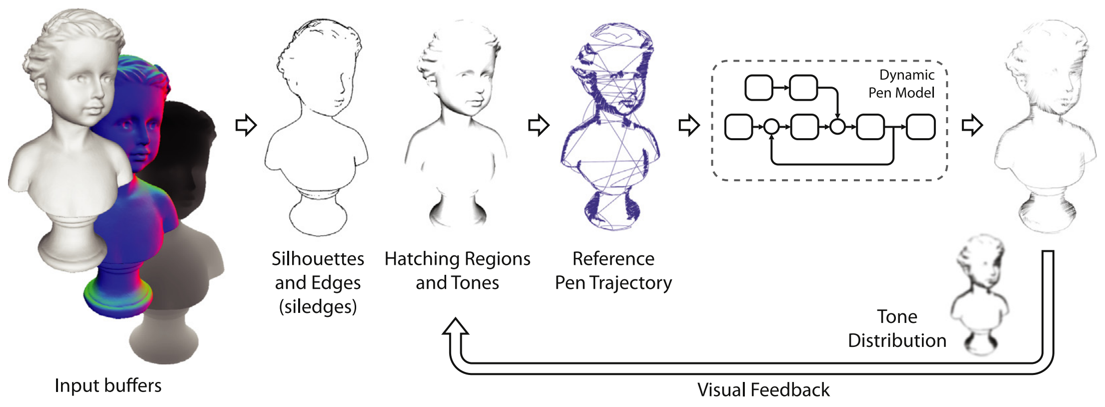

Fig. 2. The inputs to our system are the three render buffers. A series of silhouettes and edges are identified together with the regions to be shaded. A reference trajectory is designed for a dynamic pen model and later tracked using a closed control loop. In the outer feedback loop, the resulting render is compared to the reference render to minimize the tonal difference using hatching in multiple layers.

3.1. *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 are 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, it dictates the stroke's thickness and intensity as well. 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.

3.1.1. *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 [7,16,32] could also be used without loss of generality. In our implementation, we use the Laplacian convolution kernel on the depth and normal buffers to identify the curves at which the depth and normal vectors vary abruptly (Fig. 3). 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 such as those shown in Fig. 3, it may also produce disconnected islands of such curves as shown in Fig. 2. In either case, we assemble all siledge curves into a single chain using their spatial proximity. We do so using a formulation akin to the Traveling salesman problem (TSP) in which each siledge's start and end points form the nodes to be visited, and each siledge must lie on the resulting path. This assembly of the siledges effectively establishes their temporal order of rendition. 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.

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. As shown in Fig. 1, 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. Fig. 4b demonstrates this idea. We control the average length of the strokes and the overlap with two parameters, $\mathbf{P}_{\mathbf{L}_{\mathbf{mean}}}$ and $\mathbf{P}_{\mathbf{overlap}}$. 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 Fig. 4c.

Using the above parameters, strokes are sampled as follows: we start by instantiating a curve segment on one end of the siledge curve according to the stroke length distribution. If the turning angle of the sampled segment, which is the angular difference between end tangent and start tangent vectors, exceeds a prescribed turning angle (20° in our implementation), we truncate the segment back until the point where the maximum turning angle is reached. Subsequent curves are selected in a similar way. However, their starting points are determined probabilistically according to the stroke overlap distribution ($\mathbf{P}_{\mathbf{overlap}}$), which creates overlapping final sampled strokes.

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

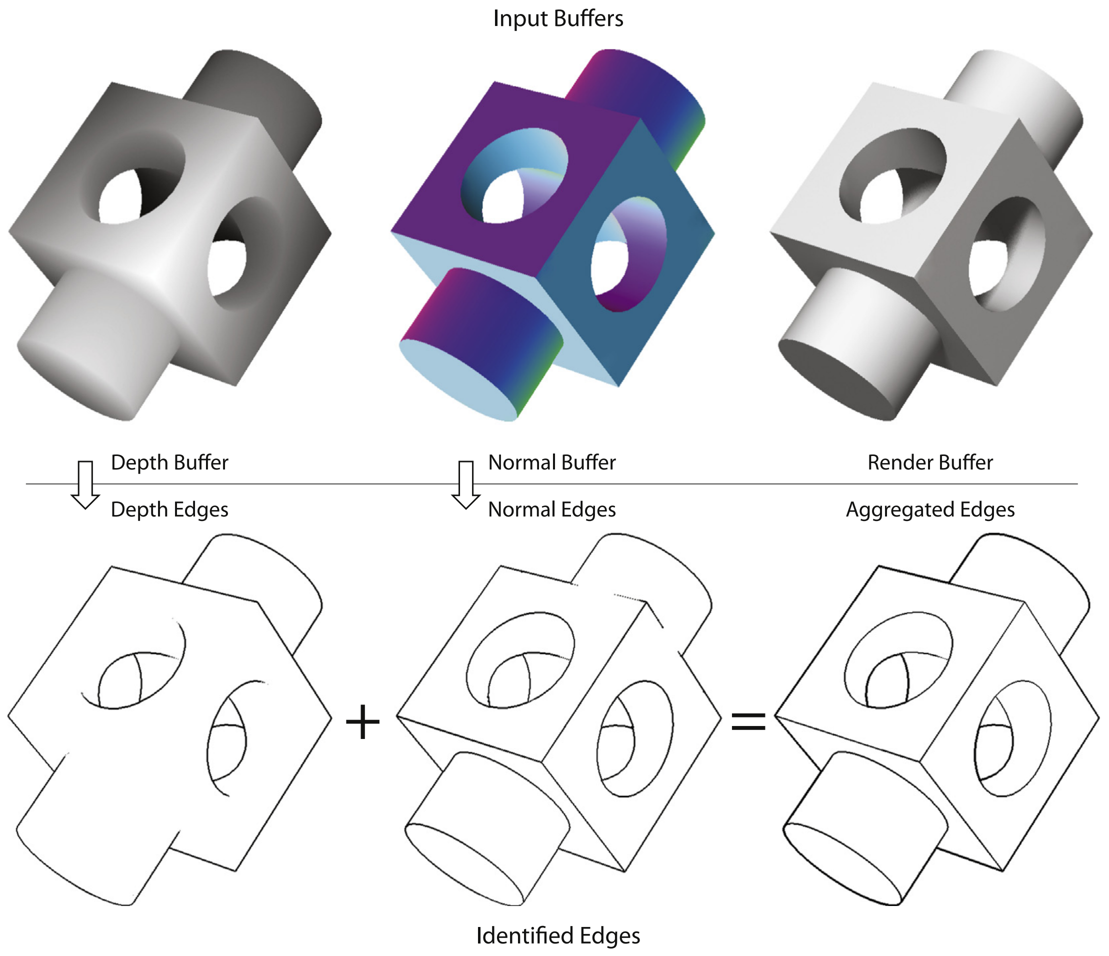

Fig. 3. The input buffers: depth, normal, and render buffers. The silhouettes calculated from the depth buffer and the sharp edges identified from the normal buffer are combined to identify the silhouettes and edges. The render buffer is later used for hatching.

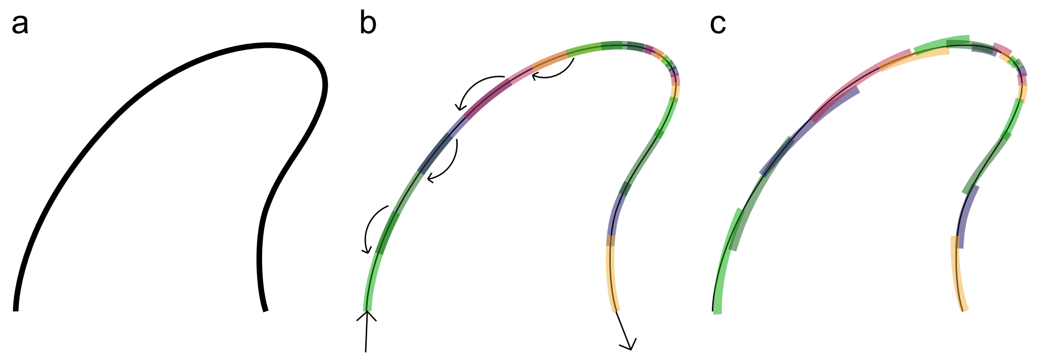

Fig. 4. Each siledge is converted into a series of strokes. The stroke lengths are determined adaptively according to the curvature along the siledge. The magnitude of overlap between consecutive strokes is determined probabilistically from a normal distribution. The strokes are altered by random perturbations in orientation, length, and scale. Strokes are rendered thicker than normal for demonstration.

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.

3.1.2. *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 ($\mathbf{P}_{\text{int-cutoff}}$) to isolate the shading regions. The first row of Fig. 5 illustrates the idea. This process results in a set of bounded regions, each treated as a separate area to be shaded. As described later, this region identification is performed several times in multiple layers, each time on progressively smaller regions arising from our feedback

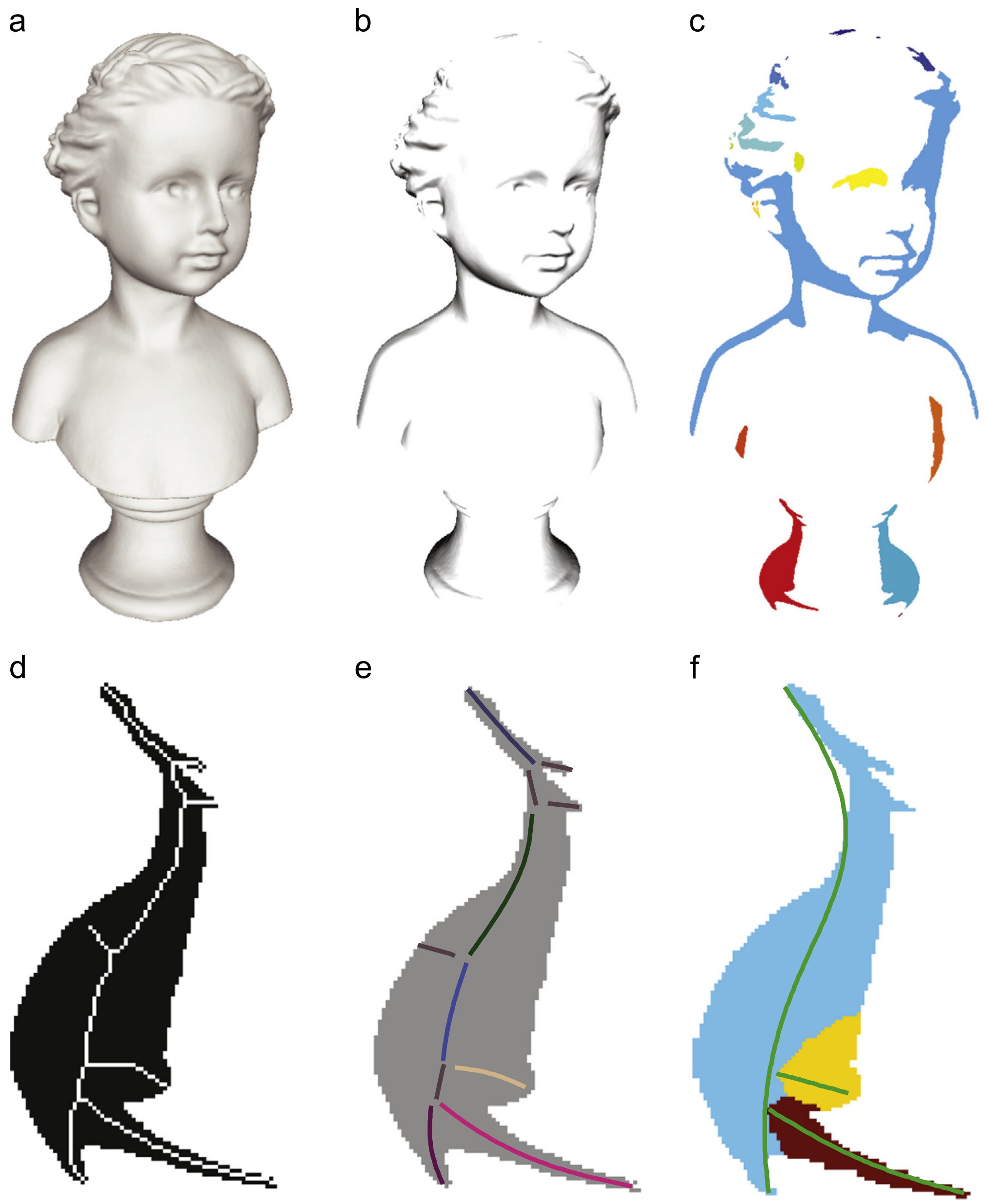

Fig. 5. Identification of hatching regions. (a) The render buffer is processed to determine (b) the hatching regions and shading tones (c) resulting in islands of regions. (d) For each region, the skeleton is calculated (e) and the branches are segmented. (f) Continuous branches are merged to produce compound branches from which a reduced set of regions are determined.

system. 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 to produce distinct hatching patterns within each subregion. This desirably helps the hatching to preserve such discontinuities.

As shown in the second row of Fig. 5, 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 (Fig. 5f). This process is repeated until no more mergers are available. Next, short branches are identified and removed, thereby leaving a small set of

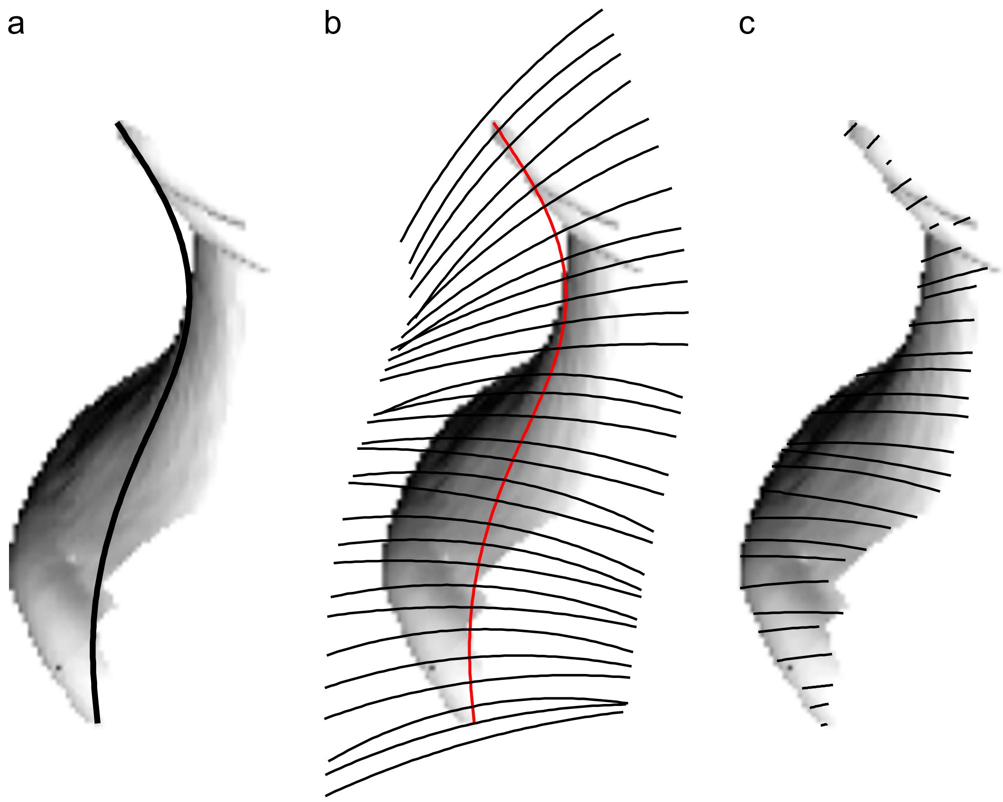

Fig. 6. Hatching stroke generation. A series of arcs are oriented along the medial axis. The arcs are trimmed with the hatching region, and perturbed in orientation and scale generating the final hatching strokes.

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. Fig. 6 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 point. The orientation of the hatching strokes is 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.

[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.

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 parameters in this process control the mean frequency of the hatching strokes $\mathbf{P}_{\mathbf{htc-freq}}$, and the deviation of the hatching frequency within the region $\mathbf{P}_{\mathbf{htc-freq-dev}}$. 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.

3.1.3. 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 $[0, \mathbf{P}_{\mathbf{z_{base}}}]$, where $\mathbf{P}_{\mathbf{z_{base}}}$ is the parameter governing the maximum penetration depth as shown in Fig. 7.

After stroke heights are computed, successively generated strokes are attached to each other using virtual transition strokes. This simulates the pen lifting off the

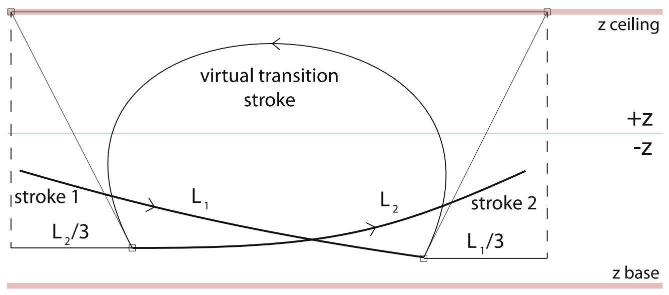

Fig. 7. The virtual transition strokes connect consecutive strokes. The strokes attain z coordinates based on the underlying intensities. The virtual transition strokes ascend into $+z$ space, though not exceeding $\mathbf{z_{ceiling}}$. The follow through of the transition strokes parallel to the virtual paper is determined by the underlying stroke's length.

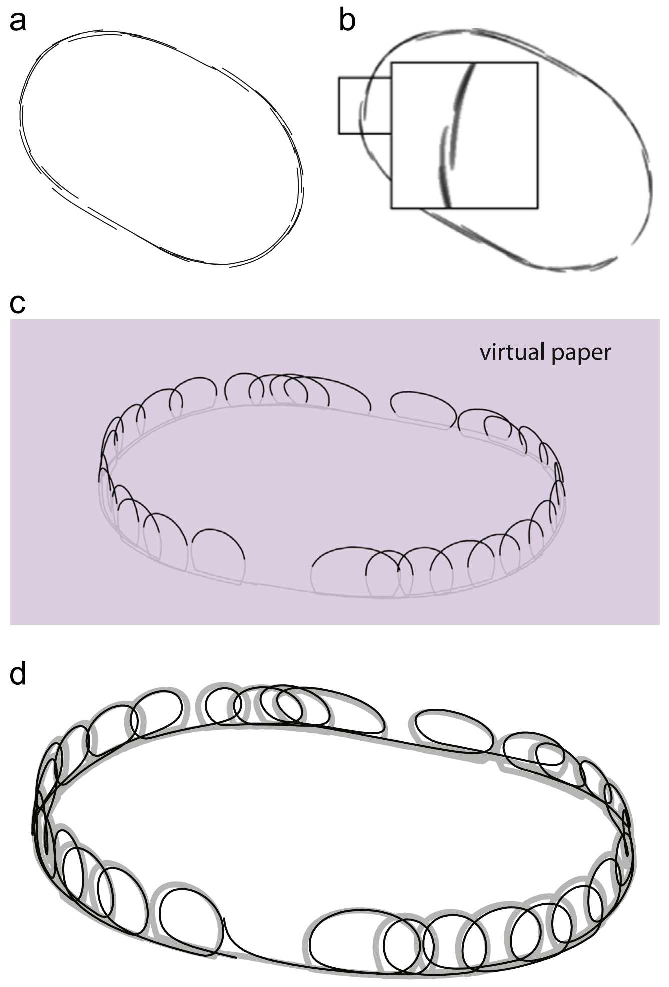

Fig. 8. (a) A series of strokes sampled in 2D. (b) The resulting rendered strokes. (c) 2D strokes are transformed into 3D reference trajectory that intersects the virtual plane at multiple locations. (d) The reference trajectory (gray) is tracked using a dynamic pen (black), producing the render in (a).

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. 7. 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 $\mathbf{P}_{\mathbf{z}_{\mathbf{ceiling}}}$ 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. Fig. 8 shows the final 3D trajectory produced in this way.

Using the above method, we generate a single reference trajectory for the collection of siledge strokes, and another one for the entire set of hatching strokes. We maintain the difference between the two trajectories to allow the dynamic pen model to utilize type-sensitive dynamic properties for tracking.

3.2. Dynamic pen model

The calculated reference trajectory is next tracked by a dynamic pen model which produces the final rendered strokes (Fig. 9). 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 [12]. 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 $\mathbf{P_{jitter}}$.

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. Fig. 8d 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 3.1.3.

3.3. Visual feedback and multi-layer rendition

While the previous step generates a rendered sketch image, the shading tones and the intensity of the sketch often do not match the tones of the original 3D render image. This is because, unlike the continuous tonal shades present in the render image, the hatching strokes form a discrete pattern. Additionally, the dynamics in the pen model causes the resulting strokes to deviate from the true tonal intensities in the render image.

To alleviate these differences, we adopt a feedback informed multi-layer hatching process. In essence, this mimics the human visual system which continuously compares a developing sketch with a real or mental reference image. In our algorithm, this comparison is calculated between the rendered strokes and the tones on the reference image. We convert our strokes into a tonal map image using a Gaussian kernel (Fig. 10b,c).

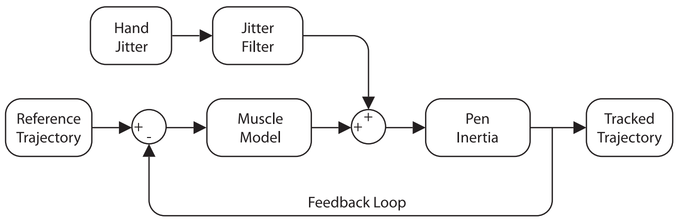

Fig. 9. Dynamic pen model is a closed loop control system that guides a pen/arm mass along a reference trajectory. The force on the pen (i.e. the force that the muscle model applies to the pen inertia) is perturbed to model the jitter in the motor control system.

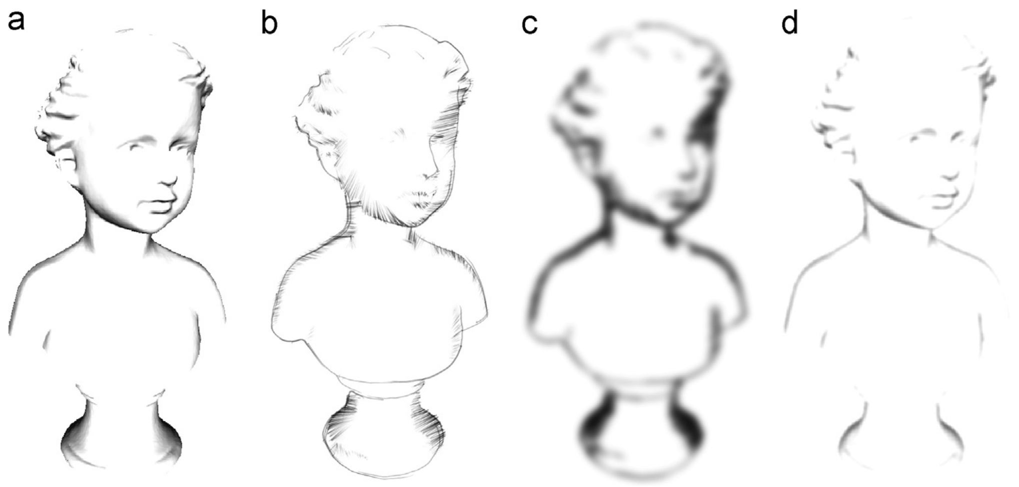

Fig. 10. (a) Original render buffer image. (b) First layer of rendered sketch. (c) A tonal map computed on the first layer using Gaussian kernels. (d) The difference image between (a) and (c) results in new regions to be sketched in the next layer of hatching.

This map is next compared with the reference image. The difference is fed back to the trajectory design step where the shading regions that require additional hatching are identified from which a new reference trajectory is calculated. This process continues until the tones on the rendered sketch match the tones of the reference image. Fig. 10d, e illustrates an example.

In each rendered layer, the general orientation of the hatching strokes relative to the medial axis is determined such that they cross the already rendered hatching strokes at maximally right angles. This arrangement is necessary to ensure that each layer of strokes causes an substantial increase in the tone intensities. As we do not limit the number of layers that can be rendered, our method can produce a variety of cross hatching patterns, created in a multitude of layers. In the following section, we demonstrate these aspects of our method with example sketch renderings.

4. Results and discussions

In this section, the rendered sketches may be best studied using the zoom functions of the digital viewers.

4.1. Performance

Our method is implemented in MATLAB. The hatching region extraction described in Section 3.1.2 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 is negligible compared to the region extraction step. The time required for the calculations depends on the resolution of the render buffers, the number of silhouette edges generated, the total area of the regions to be hatched, and the number of layers it takes to minimize the tonal difference between the sketch

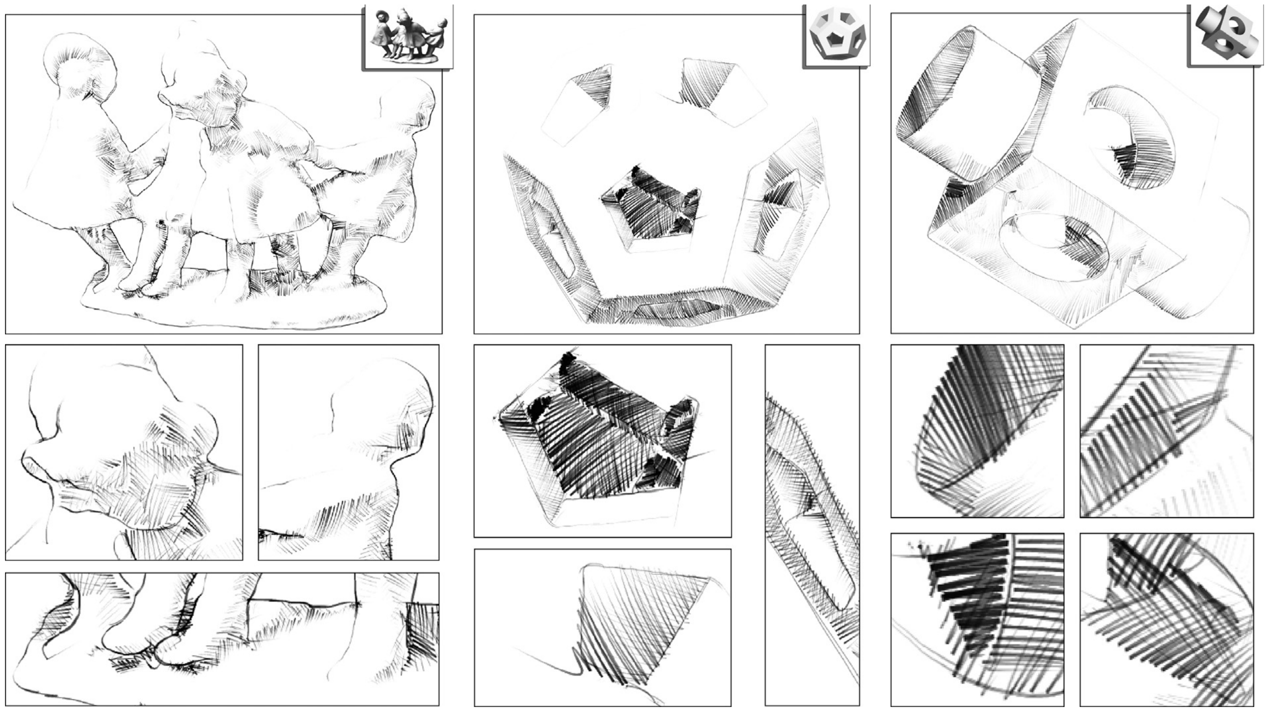

Fig. 11. Results on various models. Note the hatching behaviors and dynamic effects illustrated in the blow out views.

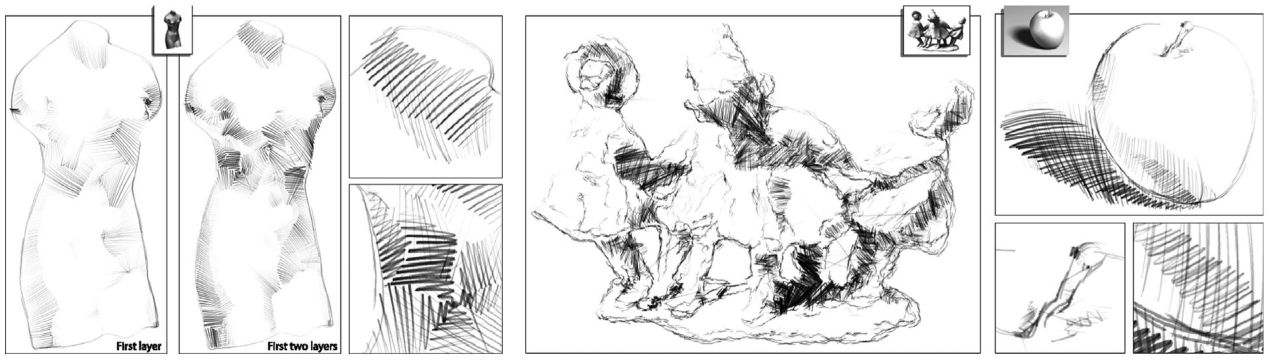

Fig. 12. Results on various models. Note the hatching behaviors and dynamic effects illustrated in the blow out views. In the dancing children sketch in particular, the short but hastily drawn strokes produce a unique impression.

and the reference render. For the sketches generated in this paper, the overall processing time varies between 30 s (apple) and 3 min (dancing children) on a Core 2 Duo computer with 3 GBs of RAM.

4.2. Examples

Figs. 11 and 12 present 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 and layers, including variations in siledge strokes, continuous versus distinct hatching strokes, cross hatching strokes, varying pen pressure and stroke intensities.

4.3. Sketch characterization and generation

Inspired by the categorization of line drawings presented in [23], we develop three qualitative descriptors that we use to systematically generate different visual effects. Each descriptor encodes a conjoint variation of a subset of the parameters that we have described in the previous sections.

Our qualitative descriptors aim to mimic (1) the skill level of the sketcher, (2) the precision or neatness of the sketcher at the particular task, and (3) the amount of detail the sketcher is permitted to articulate in the sketch. The skill level descriptor represents the amount of control and confidence a virtual artist has with the pencil. Such effects are manifested in the lengths and overlaps of the sampled strokes on siledges, the perturbations on siledge and hatching strokes, the variations on the hatch spacings, and the amount of jitter in the tracking model. By varying this parameter, our system can mimic short, unskilled strokes versus long, confident, and precise strokes.

The neatness descriptor encodes the time and care the virtual artist invests. Roughly speaking, it can be measured by the time the artists are permitted to spend on a given sketch, for a fixed number of strokes. A lack of neatness is characterized by stroke skipping, undesired extensions at stroke ends, and continuous, zigzag hatching patterns representing hastily drawn shadows.

The level of detail descriptor controls the amount of information presented in a sketch. This is measured by the size of covered regions, and the number of number sketched strokes. High level of detail corresponds to larger shaded regions hatched with denser strokes, and also with multiple layers of hatching. We can indirectly force more layers by assigning low intensities to drawn layers through the base z value. As a result, the system needs to produce many layers to achieve tonal matching.

Table 1 shows the qualitative relationships we establish between the parameters described in the previous sections and the three sketch descriptors. The sense of the correlation determines whether an increase in a parameter increases or decreases a sketch descriptor.

Table 1
Table of parameters and their variations with skill, neatness and detail level.

Parameters	Skill ↗	Neatness ↗	Detail ↗
$\mathbf{P}_{\mathbf{siledge}}$	−	−	
$\mathbf{P}_{\mathbf{hatching}}$	−	−	
$\mathbf{P}_{\mathbf{htc-freq}}$			+
$\mathbf{P}_{\mathbf{htc-freq-dev}}$	−	−	
$\mathbf{P}_{\mathbf{z_{base}}}$			+
$\mathbf{P}_{\mathbf{z_{ceiling}}}$		+	
$\mathbf{P}_{\mathbf{L_{mean}}}$	+		
$\mathbf{P}_{\mathbf{overlap}}$	−		
$\mathbf{P}_{\mathbf{mass}}$		+	
$\mathbf{P}_{\mathbf{jitter}}$	+		
$\mathbf{P}_{\mathbf{int-cutoff}}$			+

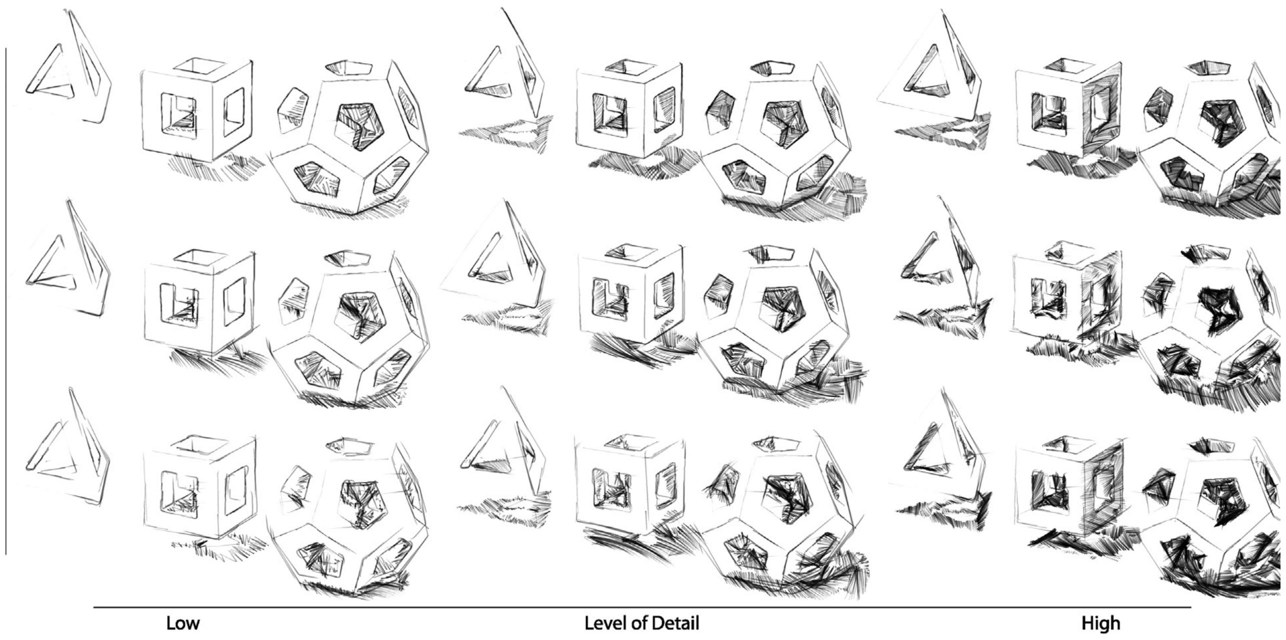

Fig. 13. Neatness versus level of detail (constant skill): the renders resulting from different values of sketch parameters according to Table 1. Increased level of detail results in more number of strokes and larger shaded regions. Increased neatness level results in higher precision in the rendered strokes.

Figs. 13 and 14 show the variation in sketch styles we obtain as a function of the three sketch descriptors. In each diagram, two descriptors are varied, while the third is kept constant.

4.4. 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. 11, 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. As shown in Fig. 15, 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.

5. 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

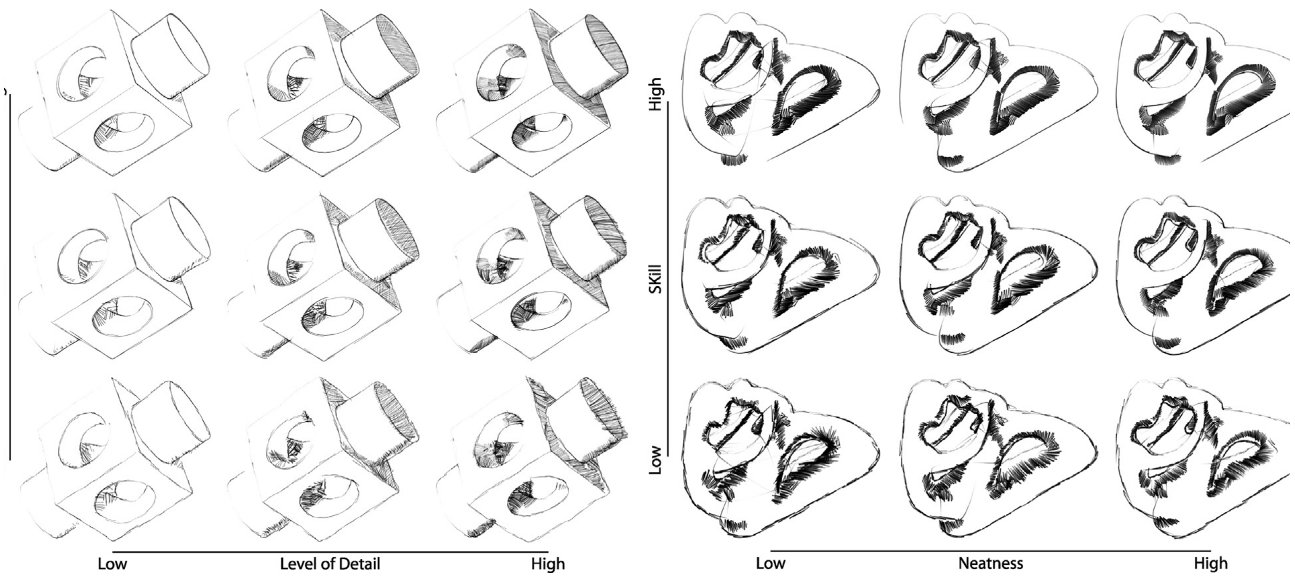

Fig. 14. Left: Skill versus level of detail (constant neatness). Right: Skill versus neatness (constant level of detail): the renders resulting from different values of sketch parameters. In both diagrams, increased skill level results in higher stroke qualities and longer, more confident strokes especially in the siledge strokes. Left: Higher levels of detail results in more strokes and larger shaded regions. Right: Higher neatness results in tidier sketches.

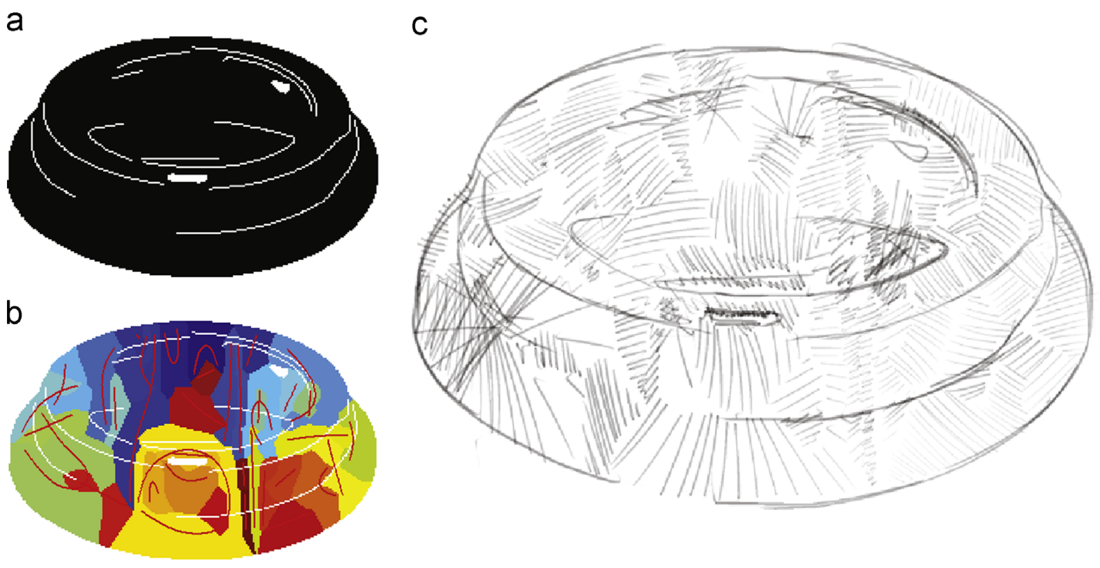

Fig. 15. Our region analysis may produce fragmented shading strips in case of intricate edges and boundaries, causing undesirable hatching patterns.

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 in multiple layers 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 that our system may also serve as a training and visualization tool, as our approach helps produce a progression of sketches in layers.

References

[1] Z. AlMeraj, B. Wyvill, T. Isenberg, A.A. Gooch, R. Guy, Automatically mimicking unique hand-drawn pencil lines, Comput. Graph. 33 (4) (2009) 496–508.

[2] P. Barla, S. Breslav, J. Thollot, F. Sillion, L. Markosian, Stroke pattern analysis and synthesis. in: Computer Graphics Forum, vol. 25, pp. 663–671, Wiley, Online Library, 2006.

[3] P. Bénard, F. Cole, A. Golovinskiy, A. Finkelstein, Self-similar texture for coherent line stylization. in: Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering, ACM, 2010, pp. 91–97.

[4] M. Brunn, M.C. Sousa, F.F. Samavati, Capturing and re-using artistic styles with reverse subdivision-based multiresolution methods, Int. J. Image Graph. 7 (4) (2007) 593–615.

[5] F. Cole, A. Finkelstein, Two fast methods for high-quality line visibility, IEEE Trans. Vis. Comput. Graph. 16 (5) (2010) 707–717.

[6] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, Interactive rendering of suggestive contours with temporal coherence. in: Proceedings of the 3rd International Symposium on Non-photorealistic Animation and Rendering, ACM, 2004, pp. 15–145.

[7] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, Anthony Santella, Suggestive contours for conveying shape, ACM Trans. Graph. (Proc. SIGGRAPH) 22 (3) (2003) 848–855.

[8] K. Eissen, R. Steur, Sketching: Drawing Techniques for Product Designers, Bis, 2007.

[9] H. Fujioka, H. Kano, Design of cursive characters using robotic arm dynamics as generation mechanism. in: Proceedings of 2006 IEEE International Conference on Robotics and Automation, 2006 (ICRA 2006), IEEE, 2006, pp. 3195–3200.

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

[11] A. Hertzmann, 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.

[12] D.H. House, M. Singh, Line drawing as a dynamic process, in: Pacific Graphics 07, 2007, pp. 351–360.

[13] P.M. Jodoin, E. Epstein, M. Granger-Piché, 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.

[14] R.D. Kalnins, P.L. Davidson, L. Markosian, A. Finkelstein, Coherent stylized silhouettes. in: ACM Transactions on Graphics (TOG), vol. 22, ACM, 2003, pp. 856–861.

[15] R.D. Kalnins, L. Markosian, B.J. Meier, M.A. Kowalski, J.C. Lee, P. L. Davidson, M. Webb, J.F. Hughes, A. Finkelstein, WYSIWYG NPR: drawing strokes directly on 3D models, ACM Trans. Graph. 21 (3) (2002) 755–762.

[16] E. Kalogerakis, D. Nowrouzezahrai, P. Simari, J. McCrae, A. Hertzmann, K. Singh, Data-driven curvature for real-time line drawing of dynamic scenes, ACM Trans. Graph. 28 (1) (2009) 11.

[17] Evangelos Kalogerakis, Derek Nowrouzezahrai, Simon Breslav, Aaron Hertzmann, Learning hatching for pen-and-ink illustration of surfaces, ACM Trans. Graph. 31 (1) (2011).

[18] L. Markosian, M.A. Kowalski, D. Goldstein, S.J. Trychin, J.F. Hughes, L. D. 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.

[19] A. Paiva, E. Vital Brazil, F. Petronetto, M.C. Sousa, Fluid-based hatching for tone mapping in line illustrations, Vis. Comput. 25 (5) (2009) 519–527.

[20] Jonathan Palacios, Eugene Zhang, Rotational symmetry field design on surfaces, ACM Trans. Graph. 26 (2007).

[21] W.M. Pang, Y. Qu, T.T. Wong, D. Cohen-Or, P.A. Heng, Structure-aware halftoning. in: ACM Transactions on Graphics (TOG), vol. 27, ACM, 2008, p. 89.

[22] E. Praun, H. Hoppe, M. Webb, A. Finkelstein, Real-time hatching. in: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, ACM, 2001, p. 581.

[23] Szymon Rusinkiewicz, Forrester Cole, Doug DeCarlo, Adam Finkelstein, Line drawings from 3D models. in: ACM SIGGRAPH 2008 Classes, SIGGRAPH '08, New York, NY, USA, 2008, ACM, pp. 39-1–39-356.

[24] M.P. Salisbury, M.T. Wong, J.F. Hughes, D.H. 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.

[25] A. Secord, Weighted Voronoi stippling. in: Proceedings of the 2nd International Symposium on Non-photorealistic Animation and Rendering, ACM, 2002, pp. 37–43.

[26] M.C. Sousa, J.W. Buchanan, Computer-generated graphite pencil rendering of 3D polygonal models. in: Computer Graphics Forum, vol. 18, 1999, pp. 195–208.

[27] M.C. Sousa, P. Prusinkiewicz, A few good lines: suggestive drawing of 3D models. in: Computer Graphics Forum, vol. 22, Wiley Online Library, 2003, pp. 381–390.

[28] M. Webb, E. Praun, A. Finkelstein, H. Hoppe, Fine tone control in hardware hatching. in: NPAR 2002: Symposium on Non Photorealistic Animation and Rendering, Citeseer, 2002.

[29] G. Winkenbach, D.H. 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.

[30] G. Winkenbach, D.H. Salesin, 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.

[31] J.W. Yeh, M. Ouhyoung, Non-photorealistic rendering in chinese painting of animals, J. Syst. Simul. 14 (6) (2002) 1220–1224.

[32] L. Zhang, Y. He, X. Xie, 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.